MiniSail

Mark P. Wassell

March 17, 2025

Abstract

MiniSail is a kernel language for Sail [1], an instruction set architecture (ISA) specification language. Sail is an imperative language with a light-weight dependent type system similar to refinement type systems such as [2]. From an ISA specification, the Sail compiler can generate theorem prover code and C (or OCaml) to give an executable emulator for an architecture. The idea behind MiniSail is to capture the key and novel features of Sail in terms of their syntax, typing rules and operational semantics, and to confirm that they work together by proving progress and preservation lemmas. We use the Nominal2 library to handle binding.

Contents

Chapter 1

Prelude

Some useful Nominal lemmas. Many of these are from Launchbury.Nominal-Utils.

1.1 Lemmas helping with equivariance proofs

lemma *perm-rel-lemma*:
  assumes $\bigwedge \pi\ x\ y.\ r\ (\pi \cdot x)\ (\pi \cdot y) \Longrightarrow r\ x\ y$
  shows $r\ (\pi \cdot x)\ (\pi \cdot y) \longleftrightarrow r\ x\ y$ (**is** $?l \longleftrightarrow ?r$)
by (*metis* (*full-types*) *assms permute-minus-cancel*(*2*))

lemma *perm-rel-lemma2*:
  assumes $\bigwedge \pi\ x\ y.\ r\ x\ y \Longrightarrow r\ (\pi \cdot x)\ (\pi \cdot y)$
  shows $r\ x\ y \longleftrightarrow r\ (\pi \cdot x)\ (\pi \cdot y)$ (**is** $?l \longleftrightarrow ?r$)
by (*metis* (*full-types*) *assms permute-minus-cancel*(*2*))

lemma *fun-eqvtI*:
  assumes *f-eqvt*[*eqvt*]: $(\bigwedge p\ x.\ p \cdot (f\ x) = f\ (p \cdot x))$
  shows $p \cdot f = f$ **by** *perm-simp rule*

lemma *eqvt-at-apply*:
  assumes *eqvt-at f x*
  shows $(p \cdot f)\ x = f\ x$
by (*metis* (*opaque-lifting*, *no-types*) *assms eqvt-at-def permute-fun-def permute-minus-cancel*(*1*))

lemma *eqvt-at-apply'*:
  assumes *eqvt-at f x*
  shows $p \cdot f\ x = f\ (p \cdot x)$
by (*metis* (*opaque-lifting*, *no-types*) *assms eqvt-at-def*)

lemma *eqvt-at-apply''*:
  assumes *eqvt-at f x*
  shows $(p \cdot f)\ (p \cdot x) = f\ (p \cdot x)$
by (*metis* (*opaque-lifting*, *no-types*) *assms eqvt-at-def permute-fun-def permute-minus-cancel*(*1*))

lemma *size-list-eqvt*[*eqvt*]: $p \cdot$ *size-list* $f\ x =$ *size-list* $(p \cdot f)\ (p \cdot x)$
proof (*induction x*)
  case (*Cons x xs*)

```
  have f x = p · (f x) by (simp add: permute-pure)
  also have ... = (p · f) (p · x) by simp
  with Cons
  show ?case by (auto simp add: permute-pure)
qed simp
```

1.2 Freshness via equivariance

```
lemma eqvt-fresh-cong1: (⋀p x. p · (f x) = f (p · x)) ⟹ a ♯ x ⟹ a ♯ f x
  apply (rule fresh-fun-eqvt-app[of f])
  apply (rule eqvtI)
  apply (rule eq-reflection)
  apply (rule ext)
  apply (metis permute-fun-def permute-minus-cancel(1))
  apply assumption
  done

lemma eqvt-fresh-cong2:
  assumes eqvt: (⋀p x y. p · (f x y) = f (p · x) (p · y))
  and fresh1: a ♯ x and fresh2: a ♯ y
  shows a ♯ f x y
proof−
  have eqvt (λ (x,y). f x y)
    using eqvt
    apply (− , auto simp add: eqvt-def)
    by (rule ext, auto, metis permute-minus-cancel(1))
  moreover
  have a ♯ (x, y) using fresh1 fresh2 by auto
  ultimately
  have a ♯ (λ (x,y). f x y) (x, y) by (rule fresh-fun-eqvt-app)
  thus ?thesis by simp
qed

lemma eqvt-fresh-star-cong1:
  assumes eqvt: (⋀p x. p · (f x) = f (p · x))
  and fresh1: a ♯* x
  shows a ♯* f x
  by (metis fresh-star-def eqvt-fresh-cong1 assms)

lemma eqvt-fresh-star-cong2:
  assumes eqvt: (⋀p x y. p · (f x y) = f (p · x) (p · y))
  and fresh1: a ♯* x and fresh2: a ♯* y
  shows a ♯* f x y
  by (metis fresh-star-def eqvt-fresh-cong2 assms)

lemma eqvt-fresh-cong3:
  assumes eqvt: (⋀p x y z. p · (f x y z) = f (p · x) (p · y) (p · z))
  and fresh1: a ♯ x and fresh2: a ♯ y and fresh3: a ♯ z
  shows a ♯ f x y z
proof−
  have eqvt (λ (x,y,z). f x y z)
    using eqvt
```

apply (− , *auto simp add*: *eqvt-def*)
by(*rule ext*, *auto*, *metis permute-minus-cancel*(*1*))
moreover
have *a* ♯ (*x*, *y*, *z*) **using** *fresh1 fresh2 fresh3* **by** *auto*
ultimately
have *a* ♯ (λ (*x*,*y*,*z*). *f x y z*) (*x*, *y*, *z*) **by** (*rule fresh-fun-eqvt-app*)
thus *?thesis* **by** *simp*
qed

lemma *eqvt-fresh-star-cong3*:
assumes *eqvt*: ($\bigwedge$*p x y z*. *p* · (*f x y z*) = *f* (*p* · *x*) (*p* · *y*) (*p* · *z*))
and *fresh1*: *a* ♯* *x* **and** *fresh2*: *a* ♯* *y* **and** *fresh3*: *a* ♯* *z*
shows *a* ♯* *f x y z*
by (*metis fresh-star-def eqvt-fresh-cong3 assms*)

1.3 Additional simplification rules

lemma *not-self-fresh*[*simp*]: *atom x* ♯ *x* ⟷ *False*
by (*metis fresh-at-base*(*2*))

lemma *fresh-star-singleton*: { *x* } ♯* *e* ⟷ *x* ♯ *e*
by (*simp add*: *fresh-star-def*)

1.4 Additional equivariance lemmas

lemma *eqvt-cases*:
fixes *f x π*
assumes *eqvt*: $\bigwedge$*x*. *π* · *f x* = *f* (*π* · *x*)
obtains *f x f* (*π* · *x*) | ¬ *f x* ¬ *f* (*π* · *x*)
using *assms*[*symmetric*]
by (*cases f x*) *auto*

lemma *range-eqvt*: *π* · *range Y* = *range* (*π* · *Y*)
unfolding *image-eqvt UNIV-eqvt* ..

lemma *case-option-eqvt*[*eqvt*]:
π · *case-option d f x* = *case-option* (*π* · *d*) (*π* · *f*) (*π* · *x*)
by(*cases x*)(*simp-all*)

lemma *supp-option-eqvt*:
supp (*case-option d f x*) ⊆ *supp d* ∪ *supp f* ∪ *supp x*
apply (*cases x*)
apply (*auto simp add*: *supp-Some*)
apply (*metis* (*mono-tags*) *Un-iff subsetCE supp-fun-app*)
done

lemma *funpow-eqvt*[*simp*,*eqvt*]:
π · ((*f* :: ′*a* ⇒ ′*a*::*pt*) ^^ *n*) = (*π* · *f*) ^^ (*π* · *n*)
by (*induct n*,*simp*, *rule ext*, *simp*, *perm-simp*,*simp*)

lemma *delete-eqvt*[*eqvt*]:

$\pi \cdot$ *AList.delete* $x\ \Gamma$ = *AList.delete* $(\pi \cdot x)$ $(\pi \cdot \Gamma)$
by (*induct* Γ, *auto*)

lemma *restrict-eqvt*[*eqvt*]:
$\pi \cdot$ *AList.restrict* $S\ \Gamma$ = *AList.restrict* $(\pi \cdot S)$ $(\pi \cdot \Gamma)$
unfolding *AList.restrict-eq* **by** *perm-simp rule*

lemma *supp-restrict*:
supp (*AList.restrict* $S\ \Gamma$) ⊆ *supp* Γ
by (*induction* Γ) (*auto simp add*: *supp-Pair supp-Cons*)

lemma *clearjunk-eqvt*[*eqvt*]:
$\pi \cdot$ *AList.clearjunk* Γ = *AList.clearjunk* $(\pi \cdot \Gamma)$
by (*induction* Γ *rule*: *clearjunk.induct*) *auto*

lemma *map-ran-eqvt*[*eqvt*]:
$\pi \cdot$ *map-ran* $f\ \Gamma$ = *map-ran* $(\pi \cdot f)$ $(\pi \cdot \Gamma)$
by (*induct* Γ, *auto*)

lemma *dom-perm*:
dom $(\pi \cdot f) = \pi \cdot$ (*dom* f)
unfolding *dom-def* **by** (*perm-simp*) (*simp*)

lemmas *dom-perm-rev*[*simp*,*eqvt*] = *dom-perm*[*symmetric*]

lemma *ran-perm*[*simp*]:
$\pi \cdot$ (*ran* f) = *ran* $(\pi \cdot f)$
unfolding *ran-def* **by** (*perm-simp*) (*simp*)

lemma *map-add-eqvt*[*eqvt*]:
$\pi \cdot (m1 ++ m2) = (\pi \cdot m1) ++ (\pi \cdot m2)$
unfolding *map-add-def*
by (*perm-simp*, *rule*)

lemma *map-of-eqvt*[*eqvt*]:
$\pi \cdot$ *map-of* l = *map-of* $(\pi \cdot l)$
by (*induct* l, *simp add*: *permute-fun-def*,*simp*,*perm-simp*,*auto*)

lemma *concat-eqvt*[*eqvt*]: $\pi \cdot$ *concat* l = *concat* $(\pi \cdot l)$
by (*induction* l)(*auto simp add*: *append-eqvt*)

lemma *tranclp-eqvt*[*eqvt*]: $\pi \cdot$ *tranclp* $P\ v_1\ v_2$ = *tranclp* $(\pi \cdot P)$ $(\pi \cdot v_1)$ $(\pi \cdot v_2)$
unfolding *tranclp-def* **by** *perm-simp rule*

lemma *rtranclp-eqvt*[*eqvt*]: $\pi \cdot$ *rtranclp* $P\ v_1\ v_2$ = *rtranclp* $(\pi \cdot P)$ $(\pi \cdot v_1)$ $(\pi \cdot v_2)$
unfolding *rtranclp-def* **by** *perm-simp rule*

lemma *Set-filter-eqvt*[*eqvt*]: $\pi \cdot$ *Set.filter* $P\ S$ = *Set.filter* $(\pi \cdot P)$ $(\pi \cdot S)$
unfolding *Set.filter-def*
by *perm-simp rule*

lemma *Sigma-eqvt′*[*eqvt*]: $\pi \cdot$ *Sigma* = *Sigma*

apply (*rule ext*)
apply (*rule ext*)
apply (*subst permute-fun-def*)
apply (*subst permute-fun-def*)
unfolding *Sigma-def*
apply *perm-simp*
apply (*simp add*: *permute-self*)
done

lemma *override-on-eqvt*[*eqvt*]:
$\pi \cdot$ (*override-on m1 m2 S*) = *override-on* ($\pi \cdot$ *m1*) ($\pi \cdot$ *m2*) ($\pi \cdot S$)
by (*auto simp add*: *override-on-def*)

lemma *card-eqvt*[*eqvt*]:
$\pi \cdot$ (*card S*) = *card* ($\pi \cdot S$)
by (*cases finite S*, *induct rule*: *finite-induct*) (*auto simp add*: *card-insert-if mem-permute-iff permute-pure*)

lemma *Projl-permute*:
assumes *a*: $\exists\, y.\ f = Inl\ y$
shows ($p \cdot$ (*Sum-Type.projl f*)) = *Sum-Type.projl* ($p \cdot f$)
using *a* **by** *auto*

lemma *Projr-permute*:
assumes *a*: $\exists\, y.\ f = Inr\ y$
shows ($p \cdot$ (*Sum-Type.projr f*)) = *Sum-Type.projr* ($p \cdot f$)
using *a* **by** *auto*

1.5 Freshness lemmas

lemma *fresh-list-elem*:
assumes $a \sharp \Gamma$
and $e \in set\ \Gamma$
shows $a \sharp e$
using *assms*
by(*induct* Γ)(*auto simp add*: *fresh-Cons*)

lemma *set-not-fresh*:
$x \in set\ L \Longrightarrow \neg(atom\ x \sharp L)$
by (*metis fresh-list-elem not-self-fresh*)

lemma *pure-fresh-star*[*simp*]: $a \sharp* (x :: 'a :: pure)$
by (*simp add*: *fresh-star-def pure-fresh*)

lemma *supp-set-mem*: $x \in set\ L \Longrightarrow supp\ x \subseteq supp\ L$
by (*induct L*) (*auto simp add*: *supp-Cons*)

lemma *set-supp-mono*: $set\ L \subseteq set\ L2 \Longrightarrow supp\ L \subseteq supp\ L2$
by (*induct L*)(*auto simp add*: *supp-Cons supp-Nil dest*:*supp-set-mem*)

lemma *fresh-star-at-base*:

fixes x :: $'a$ :: *at-base*
shows $S \sharp{*}\ x \longleftrightarrow$ *atom* $x \notin S$
by (*metis fresh-at-base*(*2*) *fresh-star-def*)

1.6 Freshness and support for subsets of variables

lemma *supp-mono*: *finite* (B::$'a$::*fs set*) $\Longrightarrow A \subseteq B \Longrightarrow$ *supp* $A \subseteq$ *supp* B
by (*metis infinite-super subset-Un-eq supp-of-finite-union*)

lemma *fresh-subset*:
finite $B \Longrightarrow x \sharp$ (B :: $'a$::*at-base set*) $\Longrightarrow A \subseteq B \Longrightarrow x \sharp A$
by (*auto dest*:*supp-mono simp add*: *fresh-def*)

lemma *fresh-star-subset*:
finite $B \Longrightarrow x \sharp{*}$ (B :: $'a$::*at-base set*) $\Longrightarrow A \subseteq B \Longrightarrow x \sharp{*}\ A$
by (*metis fresh-star-def fresh-subset*)

lemma *fresh-star-set-subset*:
$x \sharp{*}$ (B :: $'a$::*at-base list*) $\Longrightarrow$ *set* $A \subseteq$ *set* $B \Longrightarrow x \sharp{*}\ A$
by (*metis fresh-star-set fresh-star-subset*[*OF finite-set*])

1.7 The set of free variables of an expression

definition *fv* :: $'a$::*pt* $\Rightarrow$ $'b$::*at-base set*
where *fv* $e = \{v.$ *atom* $v \in$ *supp* $e\}$

lemma *fv-eqvt*[*simp*,*eqvt*]: $\pi \cdot$ (*fv* e) = *fv* ($\pi \cdot e$)
unfolding *fv-def* **by** *simp*

lemma *fv-Nil*[*simp*]: *fv* [] = {}
by (*auto simp add*: *fv-def supp-Nil*)
lemma *fv-Cons*[*simp*]: *fv* (x # xs) = *fv* x ∪ *fv* xs
by (*auto simp add*: *fv-def supp-Cons*)
lemma *fv-Pair*[*simp*]: *fv* (x, y) = *fv* x ∪ *fv* y
by (*auto simp add*: *fv-def supp-Pair*)
lemma *fv-append*[*simp*]: *fv* (x @ y) = *fv* x ∪ *fv* y
by (*auto simp add*: *fv-def supp-append*)
lemma *fv-at-base*[*simp*]: *fv* a = {a::$'a$::*at-base*}
by (*auto simp add*: *fv-def supp-at-base*)
lemma *fv-pure*[*simp*]: *fv* (a::$'a$::*pure*) = {}
by (*auto simp add*: *fv-def pure-supp*)

lemma *fv-set-at-base*[*simp*]: *fv* (l :: ($'a$ :: *at-base*) *list*) = *set* l
by (*induction* l) *auto*

lemma *flip-not-fv*: $a \notin$ *fv* $x \Longrightarrow b \notin$ *fv* $x \Longrightarrow (a \leftrightarrow b) \cdot x = x$
by (*metis flip-def fresh-def fv-def mem-Collect-eq swap-fresh-fresh*)

lemma *fv-not-fresh*: *atom* $x \sharp e \longleftrightarrow x \notin$ *fv* e
unfolding *fv-def fresh-def* **by** *blast*

lemma *fresh-fv*: *finite* (*fv e* :: *'a set*) $\Longrightarrow$ *atom* (*x* :: (*'a*::*at-base*)) ♯ (*fv e* :: *'a set*) $\longleftrightarrow$ *atom x* ♯ *e*
unfolding *fv-def fresh-def*
by (*auto simp add*: *supp-finite-set-at-base*)

lemma *finite-fv*[*simp*]: *finite* (*fv* (*e*::*'a*::*fs*) :: (*'b*::*at-base*) *set*)
proof−
have *finite* (*supp e*) **by** (*metis finite-supp*)
hence *finite* (*atom* −‘ *supp e* :: *'b set*)
apply (*rule finite-vimageI*)
apply (*rule inj-onI*)
apply (*simp*)
done
moreover
have (*atom* −‘ *supp e* :: *'b set*) = *fv e* **unfolding** *fv-def* **by** *auto*
ultimately
show *?thesis* **by** *simp*
qed

definition *fv-list* :: *'a*::*fs* ⇒ *'b*::*at-base list*
where *fv-list e* = (*SOME l*. *set l* = *fv e*)

lemma *set-fv-list*[*simp*]: *set* (*fv-list e*) = (*fv e* :: (*'b*::*at-base*) *set*)
proof−
have *finite* (*fv e* :: *'b set*) **by** (*rule finite-fv*)
from *finite-list*[*OF finite-fv*]
obtain *l* **where** *set l* = (*fv e* :: *'b set*)**..**
thus *?thesis*
unfolding *fv-list-def* **by** (*rule someI*)
qed

lemma *fresh-fv-list*[*simp*]:
a ♯ (*fv-list e* :: *'b*::*at-base list*) $\longleftrightarrow$ *a* ♯ (*fv e* :: *'b*::*at-base set*)
proof−
have *a* ♯ (*fv-list e* :: *'b*::*at-base list*) $\longleftrightarrow$ *a* ♯ *set* (*fv-list e* :: *'b*::*at-base list*)
by (*rule fresh-set*[*symmetric*])
also have ... $\longleftrightarrow$ *a* ♯ (*fv e* :: *'b*::*at-base set*) **by** *simp*
finally show *?thesis***.**
qed

1.8 Other useful lemmas

lemma *pure-permute-id*: *permute p* = (λ *x*. (*x*::*'a*::*pure*))
by *rule* (*simp add*: *permute-pure*)

lemma *supp-set-elem-finite*:
assumes *finite S*
and (*m*::*'a*::*fs*) ∈ *S*
and *y* ∈ *supp m*
shows *y* ∈ *supp S*
using *assms supp-of-finite-sets*
by *auto*

lemmas *fresh-star-Cons* = *fresh-star-list*(*2*)

lemma *mem-permute-set*:
  shows $x \in p \cdot S \longleftrightarrow (- p \cdot x) \in S$
  by (*metis mem-permute-iff permute-minus-cancel*(*2*))

lemma *flip-set-both-not-in*:
  assumes $x \notin S$ **and** $x' \notin S$
  shows $((x' \leftrightarrow x) \cdot S) = S$
  unfolding *permute-set-def*
  by (*auto*) (*metis assms flip-at-base-simps*(*3*))+

lemma *inj-atom*: *inj atom* **by** (*metis atom-eq-iff injI*)

lemmas *image-Int*[*OF inj-atom*, *simp*]

lemma *eqvt-uncurry*: *eqvt f* $\Longrightarrow$ *eqvt* (*case-prod f*)
  unfolding *eqvt-def*
  by *perm-simp simp*

lemma *supp-fun-app-eqvt2*:
  assumes *a*: *eqvt f*
  shows *supp* (*f x y*) $\subseteq$ *supp x* $\cup$ *supp y*
proof−
  from *supp-fun-app-eqvt*[*OF eqvt-uncurry* [*OF a*]]
  have *supp* (*case-prod f* (*x*,*y*)) $\subseteq$ *supp* (*x*,*y*).
  thus *?thesis* **by** (*simp add*: *supp-Pair*)
qed

lemma *supp-fun-app-eqvt3*:
  assumes *a*: *eqvt f*
  shows *supp* (*f x y z*) $\subseteq$ *supp x* $\cup$ *supp y* $\cup$ *supp z*
proof−
  from *supp-fun-app-eqvt2*[*OF eqvt-uncurry* [*OF a*]]
  have *supp* (*case-prod f* (*x*,*y*) *z*) $\subseteq$ *supp* (*x*,*y*) $\cup$ *supp z*.
  thus *?thesis* **by** (*simp add*: *supp-Pair*)
qed

lemma *permute-0*[*simp*]: *permute 0* = (λ *x*. *x*)
  by *auto*
lemma *permute-comp*[*simp*]: *permute x* $\circ$ *permute y* = *permute* (*x* + *y*) **by** *auto*

lemma *map-permute*: *map* (*permute p*) = *permute p*
  apply *rule*
  apply (*induct-tac x*)
  apply *auto*
  done

lemma *fresh-star-restrictA*[*intro*]: *a* ♯∗ Γ $\Longrightarrow$ *a* ♯∗ *AList.restrict V* Γ
  by (*induction* Γ) (*auto simp add*: *fresh-star-Cons*)

lemma *Abs-lst-Nil-eq*[*simp*]: [[]]*lst*. (*x*::′*a*::*fs*) = [*xs*]*lst*. *x*′ ⟷ (([],*x*) = (*xs*, *x*′))
apply *rule*
apply (*frule Abs-lst-fcb2*[**where** *f* = λ *x y* - . (*x*,*y*) **and** *as* = [] **and** *bs* = *xs* **and** *c* = ()])
apply (*auto simp add*: *fresh-star-def*)
done

lemma *Abs-lst-Nil-eq2*[*simp*]: [*xs*]*lst*. (*x*::′*a*::*fs*) = [[]]*lst*. *x*′ ⟷ ((*xs*,*x*) = ([], *x*′))
by (*subst eq-commute*) *auto*

lemma *prod-cases8* [*cases type*]:
obtains (*fields*) *a b c d e f g h* **where** *y* = (*a*, *b*, *c*, *d*, *e*, *f*, *g*,*h*)
by (*cases y*, *case-tac g*) *blast*

lemma *prod-induct8* [*case-names fields*, *induct type*]:
(⋀*a b c d e f g h*. *P* (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*)) ⟹ *P x*
by (*cases x*) *blast*

lemma *prod-cases9* [*cases type*]:
obtains (*fields*) *a b c d e f g h i* **where** *y* = (*a*, *b*, *c*, *d*, *e*, *f*, *g*,*h*,*i*)
by (*cases y*, *case-tac h*) *blast*

lemma *prod-induct9* [*case-names fields*, *induct type*]:
(⋀*a b c d e f g h i*. *P* (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*)) ⟹ *P x*
by (*cases x*) *blast*

named-theorems *nominal-prod-simps*

named-theorems *ms-fresh Facts for helping with freshness proofs*

lemma *fresh-prod2*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*) = (*x* ♯ *a* ∧ *x* ♯ *b*)
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod3*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c*)
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod4*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*,*d*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c* ∧ *x* ♯ *d*)
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod5*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*,*d*,*e*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c* ∧ *x* ♯ *d* ∧ *x* ♯ *e*)
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod6*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*,*d*,*e*,*f*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c* ∧ *x* ♯ *d* ∧ *x* ♯ *e* ∧ *x* ♯ *f*)
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod7*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*,*d*,*e*,*f*,*g*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c* ∧ *x* ♯ *d* ∧ *x* ♯ *e* ∧ *x* ♯ *f* ∧ *x* ♯ *g*)
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod8*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*,*d*,*e*,*f*,*g*,*h*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c* ∧ *x* ♯ *d* ∧ *x* ♯ *e* ∧ *x* ♯ *f* ∧ *x* ♯ *g* ∧ *x* ♯ *h*)

using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod9*[*nominal-prod-simps,ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod10*[*nominal-prod-simps,ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i,j) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i \wedge x \sharp j)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod12*[*nominal-prod-simps,ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i,j,k,l) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i \wedge x \sharp j \wedge x \sharp k \wedge x \sharp l)$
using *fresh-def supp-Pair* **by** *fastforce*

lemmas *fresh-prodN* = *fresh-Pair fresh-prod3 fresh-prod4 fresh-prod5 fresh-prod6 fresh-prod7 fresh-prod8 fresh-prod9 fresh-prod10 fresh-prod12*

lemma *fresh-prod2I*:
fixes *x* **and** *x1* **and** *x2*
assumes $x \sharp x1$ **and** $x \sharp x2$
shows $x \sharp (x1,x2)$ **using** *fresh-prod2 assms* **by** *auto*

lemma *fresh-prod3I*:
fixes *x* **and** *x1* **and** *x2* **and** *x3*
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$
shows $x \sharp (x1,x2,x3)$ **using** *fresh-prod3 assms* **by** *auto*

lemma *fresh-prod4I*:
fixes *x* **and** *x1* **and** *x2* **and** *x3* **and** *x4*
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$ **and** $x \sharp x4$
shows $x \sharp (x1,x2,x3,x4)$ **using** *fresh-prod4 assms* **by** *auto*

lemma *fresh-prod5I*:
fixes *x* **and** *x1* **and** *x2* **and** *x3* **and** *x4* **and** *x5*
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$ **and** $x \sharp x4$ **and** $x \sharp x5$
shows $x \sharp (x1,x2,x3,x4,x5)$ **using** *fresh-prod5 assms* **by** *auto*

lemma *flip-collapse*[*simp*]:
fixes $b1::'a::pt$ **and** $bv1::'b::at$ **and** $bv2::'b::at$
assumes *atom bv2* $\sharp$ *b1* **and** *atom c* $\sharp$ *(bv1,bv2,b1)* **and** $bv1 \neq bv2$
shows $(bv2 \leftrightarrow c) \cdot (bv1 \leftrightarrow bv2) \cdot b1 = (bv1 \leftrightarrow c) \cdot b1$
proof −
have $c \neq bv1$ **and** $bv2 \neq bv1$ **using** *assms* **by** *auto+*
hence $(bv2 \leftrightarrow c) + (bv1 \leftrightarrow bv2) + (bv2 \leftrightarrow c) = (bv1 \leftrightarrow c)$ **using** *flip-triple*[*of c bv1 bv2*] *flip-commute* **by** *metis*
hence $(bv2 \leftrightarrow c) \cdot (bv1 \leftrightarrow bv2) \cdot (bv2 \leftrightarrow c) \cdot b1 = (bv1 \leftrightarrow c) \cdot b1$ **using** *permute-plus* **by** *metis*
thus *?thesis* **using** *assms flip-fresh-fresh* **by** *force*
qed

lemma *triple-eqvt*[*simp*]:
$p \cdot (x, b,c) = (p \cdot x, p \cdot b , p \cdot c)$
proof −

```
  have (x,b,c) = (x,(b,c)) by simp
  thus ?thesis using Pair-eqvt by simp
qed

lemma lst-fst:
  fixes x::'a::at and t1::'b::fs and  x'::'a::at and t2::'c::fs
  assumes  ([[atom x]]lst. (t1,t2) = [[atom x']]lst. (t1',t2'))
  shows  ([[atom x]]lst. t1 = [[atom x']]lst. t1')
proof −
  have (∀ c. atom c ♯ (t2,t2') ⟶ atom c ♯ (x, x', t1, t1') ⟶ (x ↔ c) • t1 = (x' ↔ c) • t1')
  proof(rule,rule,rule)
    fix c::'a
    assume atom c ♯  (t2,t2') and atom c ♯ (x, x', t1, t1')
    hence atom c ♯ (x, x', (t1,t2), (t1',t2'))  using fresh-prod2 by simp
    thus   (x ↔ c) • t1 = (x' ↔ c) • t1' using assms Abs1-eq-iff-all(3) Pair-eqvt by simp
  qed
  thus ?thesis using Abs1-eq-iff-all(3)[of x t1 x' t1' (t2,t2')] by simp
qed

lemma lst-snd:
  fixes x::'a::at and t1::'b::fs and  x'::'a::at and t2::'c::fs
  assumes  ([[atom x]]lst. (t1,t2) = [[atom x']]lst. (t1',t2'))
  shows  ([[atom x]]lst. t2 = [[atom x']]lst. t2')
proof −
  have (∀ c. atom c ♯ (t1,t1') ⟶ atom c ♯ (x, x', t2, t2') ⟶ (x ↔ c) • t2 = (x' ↔ c) • t2')
  proof(rule,rule,rule)
    fix c::'a
    assume atom c ♯  (t1,t1') and atom c ♯ (x, x', t2, t2')
    hence atom c ♯ (x, x', (t1,t2), (t1',t2'))  using fresh-prod2 by simp
    thus   (x ↔ c) • t2 = (x' ↔ c) • t2' using assms Abs1-eq-iff-all(3) Pair-eqvt by simp
  qed
  thus ?thesis using Abs1-eq-iff-all(3)[of x t2 x' t2' (t1,t1')] by simp
qed

lemma lst-head-cons-pair:
  fixes y1::'a ::at and y2::'a::at and x1::'b::fs and x2::'b::fs and xs1::('b::fs) list and xs2::('b::fs) list
  assumes [[atom y1]]lst. (x1 # xs1) = [[atom y2]]lst. (x2 # xs2)
  shows [[atom y1]]lst. (x1,xs1) = [[atom y2]]lst. (x2,xs2)
proof(subst  Abs1-eq-iff-all(3)[of y1  (x1,xs1)  y2  (x2,xs2)],rule,rule,rule)
  fix c::'a
  assume atom c ♯ (x1#xs1,x2#xs2) and atom c ♯ (y1, y2, (x1, xs1), x2, xs2)
  thus  (y1 ↔ c) • (x1, xs1) = (y2 ↔ c) • (x2, xs2)  using assms Abs1-eq-iff-all(3) by auto
qed

lemma lst-head-cons-neq-nil:
  fixes y1::'a ::at and y2::'a::at and x1::'b::fs and x2::'b::fs and xs1::('b::fs) list and xs2::('b::fs) list
  assumes [[atom y1]]lst. (x1 # xs1) = [[atom y2]]lst. (xs2)
  shows xs2 ≠ []
proof
  assume as:xs2 = []
  thus  False using Abs1-eq-iff(3)[of y1 x1#xs1 y2 Nil]  assms as by auto
qed
```

lemma *lst-head-cons*:
fixes *y1*::′*a* ::*at* **and** *y2*::′*a*::*at* **and** *x1*::′*b*::*fs* **and** *x2*::′*b*::*fs* **and** *xs1*::(′*b*::*fs*) *list* **and** *xs2*::(′*b*::*fs*) *list*
assumes [[*atom y1*]]*lst*. (*x1* # *xs1*) = [[*atom y2*]]*lst*. (*x2* # *xs2*)
shows [[*atom y1*]]*lst*. *x1* = [[*atom y2*]]*lst*. *x2* **and** [[*atom y1*]]*lst*. *xs1* = [[*atom y2*]]*lst*. *xs2*
using *lst-head-cons-pair lst-fst lst-snd assms* **by** *metis*+

lemma *lst-pure*:
fixes *x1*::′*a* ::*at* **and** *t1*::′*b*::*pure* **and** *x2*::′*a* ::*at* **and** *t2*::′*b*::*pure*
assumes [[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2*
shows *t1*=*t2*
using *assms Abs1-eq-iff-all*(*3*) *pure-fresh flip-fresh-fresh*
by (*metis Abs1-eq*(*3*) *permute-pure*)

lemma *lst-supp*:
assumes [[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2*
shows *supp t1* − {*atom x1*} = *supp t2* − {*atom x2*}
proof −
have *supp* ([[*atom x1*]]*lst*.*t1*) = *supp* ([[*atom x2*]]*lst*.*t2*) **using** *assms* **by** *auto*
thus *?thesis* **using** *Abs-finite-supp*
by (*metis assms empty-set list.simps*(*15*) *supp-lst.simps*)
qed

lemma *lst-supp-subset*:
assumes [[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2* **and** *supp t1* ⊆ {*atom x1*} ∪ *B*
shows *supp t2* ⊆ {*atom x2*} ∪ *B*
using *assms lst-supp* **by** *fast*

lemma *projl-inl-eqvt*:
fixes π :: *perm*
shows π · (*projl* (*Inl x*)) = *projl* (*Inl* (π · *x*))
unfolding *projl-def Inl-eqvt* **by** *simp*

end

Chapter 2

Syntax

Syntax of MiniSail programs and the contexts we use in judgements.

2.1 Program Syntax

2.1.1 AST Datatypes

type-synonym *num-nat = nat*

atom-decl *x*
atom-decl *u*
atom-decl *bv*

type-synonym *f = string*
type-synonym *dc = string*
type-synonym *tyid = string*

Basic types. Types without refinement constraints

nominal-datatype *b =*
  B-int | *B-bool* | *B-id tyid*
| *B-pair b b* (‹$[- , -]^b$›)
| *B-unit* | *B-bitvec* | *B-var bv*
| *B-app tyid b*

nominal-datatype *bit = BitOne* | *BitZero*

Literals

nominal-datatype *l =*
  L-num int | *L-true* | *L-false* | *L-unit* | *L-bitvec bit list*

Values. We include a type identifier, tyid, in the literal for constructors to make typing and well-formedness checking easier

nominal-datatype *v =*
  V-lit l (‹$[-]^v$›)
  | *V-var x* (‹$[-]^v$›)
  | *V-pair v v* (‹$[- , -]^v$›)
  | *V-cons tyid dc v*

| *V-consp tyid dc b v*

Binary Operations

nominal-datatype *opp* = *Plus* (‹*plus*›) | *LEq* (‹*leq*›) | *Eq* (‹*eq*›)

Expressions

nominal-datatype *e* =
AE-val v (‹[-]^e›)
| *AE-app f v* (‹[- (-)]^e›)
| *AE-appP f b v* (‹[- [-] (-)]^e›)
| *AE-op opp v v* (‹[- - -]^e›)
| *AE-concat v v* (‹[- @@ -]^e›)
| *AE-fst v* (‹[#*1*-]^e›)
| *AE-snd v* (‹[#*2*-]^e›)
| *AE-mvar u* (‹[-]^e›)
| *AE-len v* (‹[| - |]^e›)
| *AE-split v v* (‹[- / -]^e›)

Expressions for constraints

nominal-datatype *ce* =
CE-val v (‹[-]^{ce}›)
| *CE-op opp ce ce* (‹[- - -]^{ce}›)
| *CE-concat ce ce* (‹[- @@ -]^{ce}›)
| *CE-fst ce* (‹[#*1*-]^{ce}›)
| *CE-snd ce* (‹[#*2*-]^{ce}›)
| *CE-len ce* (‹[| - |]^{ce}›)

Constraints

nominal-datatype *c* =
C-true (‹*TRUE*› [] *50*)
| *C-false* (‹*FALSE*› [] *50*)
| *C-conj c c* (‹- *AND* - › [*50*, *50*] *50*)
| *C-disj c c* (‹- *OR* - › [*50*,*50*] *50*)
| *C-not c* (‹¬ - › [] *50*)
| *C-imp c c* (‹- *IMP* - › [*50*, *50*] *50*)
| *C-eq ce ce* (‹- == - › [*50*, *50*] *50*)

Refined types

nominal-datatype τ =
T-refined-type x::*x b c*::*c* **binds** *x* **in** *c* (‹⦃ - : - | - ⦄› [*50*, *50*] *1000*)

Statements

nominal-datatype
s =
AS-val v (‹[-]^s›)
| *AS-let x*::*x e s*::*s* **binds** *x* **in** *s* (‹(*LET* - = - *IN* -)›)
| *AS-let2 x*::*x* τ *s s*::*s* **binds** *x* **in** *s* (‹(*LET* - : - = - *IN* -)›)
| *AS-if v s s* (‹(*IF* - *THEN* - *ELSE* -)› [*0*, *61*, *0*] *61*)
| *AS-var u*::*u* τ *v s*::*s* **binds** *u* **in** *s* (‹(*VAR* - : - = - *IN* -)›)
| *AS-assign u v* (‹(- ::= -)›)
| *AS-match v branch-list* (‹(*MATCH* - *WITH* { - })›)

| *AS-while s s* (‹(*WHILE* - *DO* { - })› [*0*, *0*] *61*)
| *AS-seq s s* (‹(- ;; -)› [*1000*, *61*] *61*)
| *AS-assert c s* (‹(*ASSERT* - *IN* -)›)
and *branch-s* =
AS-branch dc x::*x s*::*s* **binds** *x* **in** *s* (‹(- - ⇒ -)›)
and *branch-list* =
AS-final branch-s (‹{ - }›)
| *AS-cons branch-s branch-list* (‹(- | -)›)

Function and union type definitions

nominal-datatype *fun-typ* =
AF-fun-typ x::*x b c*::*c* τ::τ *s*::*s* **binds** *x* **in** *c* τ *s*

nominal-datatype *fun-typ-q* =
AF-fun-typ-some bv::*bv ft*::*fun-typ* **binds** *bv* **in** *ft*
| *AF-fun-typ-none fun-typ*

nominal-datatype *fun-def* = *AF-fundef f fun-typ-q*

nominal-datatype *type-def* =
AF-typedef string (*string* ∗ τ) *list*
| *AF-typedef-poly string bv*::*bv dclist*::(*string* ∗ τ) *list* **binds** *bv* **in** *dclist*

lemma *check-typedef-poly*:
AF-typedef-poly ''option'' bv [(*''None''*, ⦃ *zz* : *B-unit* | *TRUE* ⦄), (*''Some''*, ⦃ *zz* : *B-var bv* | *TRUE* ⦄)] =
AF-typedef-poly ''option'' bv2 [(*''None''*, ⦃ *zz* : *B-unit* | *TRUE* ⦄), (*''Some''*, ⦃ *zz* : *B-var bv2* | *TRUE* ⦄)]
by *auto*

nominal-datatype *var-def* = *AV-def u* τ *v*

Programs

nominal-datatype *p* =
AP-prog type-def list fun-def list var-def list s (‹*PROG* - - - -›)

declare *l.supp* [*simp*] *v.supp* [*simp*] *e.supp* [*simp*] *s-branch-s-branch-list.supp* [*simp*] τ*.supp* [*simp*] *c.supp* [*simp*] *b.supp*[*simp*]

2.1.2 Lemmas

These lemmas deal primarily with freshness and alpha-equivalence

Atoms

lemma *x-not-in-u-atoms*[*simp*]:
fixes *u*::*u* **and** *x*::*x* **and** *us*::*u set*
shows *atom x* ∉ *atom‘us*
by (*simp add*: *image-iff*)

lemma *x-fresh-u*[*simp*]:
fixes *u*::*u* **and** *x*::*x*

shows *atom x* ♯ *u*
by *auto*

lemma *x-not-in-b-set*[*simp*]:
fixes *x*::*x* **and** *bs*::*bv fset*
shows *atom x* ∉ *supp bs*
by(*induct bs*,*auto*, *simp add*: *supp-finsert supp-at-base*)

lemma *x-fresh-b*[*simp*]:
fixes *x*::*x* **and** *b*::*b*
shows *atom x* ♯ *b*
apply (*induct b rule*: *b.induct*, *auto simp*: *pure-supp*)
using *pure-supp fresh-def* **by** *blast*+

lemma *x-fresh-bv*[*simp*]:
fixes *x*::*x* **and** *bv*::*bv*
shows *atom x* ♯ *bv*
using *fresh-def supp-at-base* **by** *auto*

lemma *u-not-in-x-atoms*[*simp*]:
fixes *u*::*u* **and** *x*::*x* **and** *xs*::*x set*
shows *atom u* ∉ *atom‘xs*
by (*simp add*: *image-iff*)

lemma *bv-not-in-x-atoms*[*simp*]:
fixes *bv*::*bv* **and** *x*::*x* **and** *xs*::*x set*
shows *atom bv* ∉ *atom‘xs*
by (*simp add*: *image-iff*)

lemma *u-not-in-b-atoms*[*simp*]:
fixes *b* :: *b* **and** *u*::*u*
shows *atom u* ∉ *supp b*
by (*induct b rule*: *b.induct*,*auto simp*: *pure-supp supp-at-base*)

lemma *u-not-in-b-set*[*simp*]:
fixes *u*::*u* **and** *bs*::*bv fset*
shows *atom u* ∉ *supp bs*
by(*induct bs*, *auto simp add*: *supp-at-base supp-finsert*)

lemma *u-fresh-b*[*simp*]:
fixes *x*::*u* **and** *b*::*b*
shows *atom x* ♯ *b*
by(*induct b rule*: *b.induct*, *auto simp*: *pure-fresh*)

lemma *supp-b-v-disjoint*:
fixes *x*::*x* **and** *bv*::*bv*
shows *supp* (*V-var x*) ∩ *supp* (*B-var bv*) = {}
by (*simp add*: *supp-at-base*)

lemma *supp-b-u-disjoint*[*simp*]:
fixes *b*::*b* **and** *u*::*u*
shows *supp u* ∩ *supp b* = {}

by(*nominal-induct b rule*:*b.strong-induct*,(*auto simp add*: *pure-supp b.supp supp-at-base*)+)

lemma *u-fresh-bv*[*simp*]:
fixes *u*::*u* **and** *b*::*bv*
shows *atom u* ♯ *b*
using *fresh-at-base* **by** *simp*

Basic Types

nominal-function *b-of* :: $\tau \Rightarrow b$ **where**
b-of ⦃ *z* : *b* | *c* ⦄ = *b*
apply(*auto*,*simp add*: *eqvt-def b-of-graph-aux-def*)
by (*meson* τ*.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *supp-b-empty*[*simp*]:
fixes *b* :: *b* **and** *x*::*x*
shows *atom x* ∉ *supp b*
by (*induct b rule*: *b.induct*, *auto simp*: *pure-supp supp-at-base x-not-in-b-set*)

lemma *flip-b-id*[*simp*]:
fixes *x*::*x* **and** *b*::*b*
shows $(x \leftrightarrow x') \cdot b = b$
by(*rule flip-fresh-fresh*, *auto simp add*: *fresh-def*)

lemma *flip-x-b-cancel*[*simp*]:
fixes *x*::*x* **and** *y*::*x* **and** *b*::*b* **and** *bv*::*bv*
shows $(x \leftrightarrow y) \cdot b = b$ **and** $(x \leftrightarrow y) \cdot bv = bv$
using *flip-b-id* **apply** *simp*
by (*metis b.eq-iff*(*7*) *b.perm-simps*(*7*) *flip-b-id*)

lemma *flip-bv-x-cancel*[*simp*]:
fixes *bv*::*bv* **and** *z*::*bv* **and** *x*::*x*
shows $(bv \leftrightarrow z) \cdot x = x$ **using** *flip-fresh-fresh*[*of bv x z*] *fresh-at-base* **by** *auto*

lemma *flip-bv-u-cancel*[*simp*]:
fixes *bv*::*bv* **and** *z*::*bv* **and** *x*::*u*
shows $(bv \leftrightarrow z) \cdot x = x$ **using** *flip-fresh-fresh*[*of bv x z*] *fresh-at-base* **by** *auto*

Literals

lemma *supp-bitvec-empty*:
fixes *bv*::*bit list*
shows *supp bv* = {}
proof(*induct bv*)
case *Nil*
then show *?case* **using** *supp-Nil* **by** *auto*
next
case (*Cons a bv*)
then show *?case* **using** *supp-Cons bit.supp*
by (*metis* (*mono-tags*, *opaque-lifting*) *bit.strong-exhaust l.supp*(*5*) *sup-bot.right-neutral*)
qed

lemma *bitvec-pure*[*simp*]:
  fixes *bv*::*bit list* **and** *x*::*x*
  shows *atom x* ♯ *bv* **using** *fresh-def supp-bitvec-empty* **by** *auto*

lemma *supp-l-empty*[*simp*]:
  fixes *l*:: *l*
  shows *supp* (*V-lit l*) = {}
  by(*nominal-induct l rule*: *l.strong-induct*,
    auto simp add: *l.strong-exhaust pure-supp v.fv-defs supp-bitvec-empty*)

lemma *type-l-nosupp*[*simp*]:
  fixes *x*::*x* **and** *l*::*l*
  shows *atom x* ∉ *supp* (⦃ *z* : *b* | $[[z]^v]^{ce}$ == $[[l]^v]^{ce}$ ⦄)
  using *supp-at-base supp-l-empty ce.supp*(*1*) *c.supp τ.supp* **by** *force*

lemma *flip-bitvec0*:
  fixes *x*::*bit list*
  assumes *atom c* ♯ (*z*, *x*, *z'*)
  shows (*z* ↔ *c*) · *x* = (*z'* ↔ *c*) · *x*
proof −
  have *atom z* ♯ *x* **and** *atom z'* ♯ *x*
    using *flip-fresh-fresh assms supp-bitvec-empty fresh-def* **by** *blast*+
  moreover have *atom c* ♯ *x* **using** *supp-bitvec-empty fresh-def* **by** *auto*
  ultimately show *?thesis* **using** *assms flip-fresh-fresh* **by** *metis*
qed

lemma *flip-bitvec*:
  assumes *atom c* ♯ (*z*, *L-bitvec x*, *z'*)
  shows (*z* ↔ *c*) · *x* = (*z'* ↔ *c*) · *x*
proof −
  have *atom z* ♯ *x* **and** *atom z'* ♯ *x*
    using *flip-fresh-fresh assms supp-bitvec-empty fresh-def* **by** *blast*+
  moreover have *atom c* ♯ *x* **using** *supp-bitvec-empty fresh-def* **by** *auto*
  ultimately show *?thesis* **using** *assms flip-fresh-fresh* **by** *metis*
qed

lemma *type-l-eq*:
  shows ⦃ *z* : *b* | $[[z]^v]^{ce}$ == $[V\text{-}lit\ l]^{ce}$ ⦄ = (⦃ *z'* : *b* | $[[z']^v]^{ce}$ == $[V\text{-}lit\ l]^{ce}$ ⦄)
  by(*auto*,*nominal-induct l rule*: *l.strong-induct*,*auto*, *metis permute-pure*, *auto simp add*: *flip-bitvec*)

lemma *flip-l-eq*:
  fixes *x*::*l*
  shows (*z* ↔ *c*) · *x* = (*z'* ↔ *c*) · *x*
proof −
  have *atom z* ♯ *x* **and** *atom c* ♯ *x* **and** *atom z'* ♯ *x*
    using *flip-fresh-fresh fresh-def supp-l-empty* **by** *fastforce*+
  thus *?thesis* **using** *flip-fresh-fresh* **by** *metis*
qed

lemma *flip-l-eq1*:
  fixes *x*::*l*
  assumes (*z* ↔ *c*) · *x* = (*z'* ↔ *c*) · *x'*

shows $x' = x$
proof –
have *atom z* ♯ *x* **and** *atom c* ♯ *x'* **and** *atom c* ♯ *x* **and** *atom z'* ♯ *x'*
using *flip-fresh-fresh fresh-def supp-l-empty* **by** *fastforce+*
thus *?thesis* **using** *flip-fresh-fresh assms* **by** *metis*
qed

Types

lemma *flip-base-eq*:
fixes *b::b* **and** *x::x* **and** *y::x*
shows $(x \leftrightarrow y) \cdot b = b$
using *b.fresh* **by** (*simp add*: *flip-fresh-fresh fresh-def*)

Obtain an alpha-equivalent type where the bound variable is fresh in some term t

lemma *has-fresh-z0*:
fixes *t::'b::fs*
shows $\exists z.$ *atom z* ♯ $(c',t) \wedge (\{\!|z' : b \mid c' |\!\}) = (\{\!| z : b \mid (z \leftrightarrow z') \cdot c' |\!\})$
proof –
obtain *z::x* **where** *fr*: *atom z* ♯ (c',t) **using** *obtain-fresh* **by** *blast*
moreover hence $(\{\!| z' : b \mid c' |\!\}) = (\{\!| z : b \mid (z \leftrightarrow z') \cdot c' |\!\})$
using *τ.eq-iff Abs1-eq-iff*
by (*metis flip-commute flip-fresh-fresh fresh-PairD(1)*)
ultimately show *?thesis* **by** *fastforce*
qed

lemma *has-fresh-z*:
fixes *t::'b::fs*
shows $\exists z\ b\ c.$ *atom z* ♯ $t \wedge \tau = \{\!| z : b \mid c |\!\}$
proof –
obtain *z'* **and** *b* **and** *c'* **where** *teq*: $\tau = (\{\!| z' : b \mid c' |\!\})$ **using** *τ.exhaust* **by** *blast*
obtain *z::x* **where** *fr*: *atom z* ♯ (t,c') **using** *obtain-fresh* **by** *blast*
hence $(\{\!| z' : b \mid c' |\!\}) = (\{\!| z : b \mid (z \leftrightarrow z') \cdot c' |\!\})$ **using** *τ.eq-iff Abs1-eq-iff*
flip-commute flip-fresh-fresh fresh-PairD(1) **by** (*metis fresh-PairD(2)*)
hence *atom z* ♯ $t \wedge \tau = (\{\!| z : b \mid (z \leftrightarrow z') \cdot c' |\!\})$ **using** *fr teq* **by** *force*
thus *?thesis* **using** *teq fr* **by** *fast*
qed

lemma *obtain-fresh-z*:
fixes *t::'b::fs*
obtains *z* **and** *b* **and** *c* **where** *atom z* ♯ $t \wedge \tau = \{\!| z : b \mid c |\!\}$
using *has-fresh-z* **by** *blast*

lemma *has-fresh-z2*:
fixes *t::'b::fs*
shows $\exists z\ c.$ *atom z* ♯ $t \wedge \tau = \{\!| z : b\text{-}of\ \tau \mid c |\!\}$
proof –
obtain *z* **and** *b* **and** *c* **where** *atom z* ♯ $t \wedge \tau = \{\!| z : b \mid c |\!\}$ **using** *obtain-fresh-z* **by** *metis*
moreover then have *b-of* $\tau = b$ **using** *τ.eq-iff* **by** *simp*
ultimately show *?thesis* **using** *obtain-fresh-z τ.eq-iff* **by** *auto*
qed

lemma *obtain-fresh-z2*:

fixes *t*::*'b*::*fs*
obtains *z* **and** *c* **where** *atom z* ♯ *t* ∧ *τ* = ⦃ *z* : *b-of τ* | *c* ⦄
using *has-fresh-z2* **by** *blast*

Values

lemma *u-notin-supp-v*[*simp*]:
fixes *u*::*u* **and** *v*::*v*
shows *atom u* ∉ *supp v*
proof(*nominal-induct v rule*: *v.strong-induct*)
case (*V-lit l*)
then show *?case* **using** *supp-l-empty* **by** *auto*
next
case (*V-var x*)
then show *?case*
by (*simp add*: *supp-at-base*)
next
case (*V-pair v1 v2*)
then show *?case* **by** *auto*
next
case (*V-cons tyid list v*)
then show *?case* **using** *pure-supp* **by** *auto*
next
case (*V-consp tyid list b v*)
then show *?case* **using** *pure-supp* **by** *auto*
qed

lemma *u-fresh-xv*[*simp*]:
fixes *u*::*u* **and** *x*::*x* **and** *v*::*v*
shows *atom u* ♯ (*x*,*v*)
proof −
have *atom u* ♯ *x* **using** *fresh-def* **by** *fastforce*
moreover have *atom u* ♯ *v* **using** *fresh-def u-notin-supp-v* **by** *metis*
ultimately show *?thesis* **using** *fresh-prod2* **by** *auto*
qed

Part of an effort to make the proofs across inductive cases more uniform by distilling the non-uniform parts into lemmas like this

lemma *v-flip-eq*:
fixes *v*::*v* **and** *va*::*v* **and** *x*::*x* **and** *c*::*x*
assumes *atom c* ♯ (*v*, *va*) **and** *atom c* ♯ (*x*, *xa*, *v*, *va*) **and** (*x* ↔ *c*) · *v* = (*xa* ↔ *c*) · *va*
shows ((*v* = *V-lit l* ⟶ (∃ *l'*. *va* = *V-lit l'* ∧ (*x* ↔ *c*) · *l* = (*xa* ↔ *c*) · *l'*))) ∧
((*v* = *V-var y* ⟶ (∃ *y'*. *va* = *V-var y'* ∧ (*x* ↔ *c*) · *y* = (*xa* ↔ *c*) · *y'*))) ∧
((*v* = *V-pair vone vtwo* ⟶ (∃ *v1' v2'*. *va* = *V-pair v1' v2'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'* ∧ (*x* ↔ *c*) · *vtwo* = (*xa* ↔ *c*) · *v2'*))) ∧
((*v* = *V-cons tyid dc vone* ⟶ (∃ *v1'*. *va* = *V-cons tyid dc v1'*∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*))) ∧
((*v* = *V-consp tyid dc b vone* ⟶ (∃ *v1'*. *va* = *V-consp tyid dc b v1'*∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)))
using *assms* **proof**(*nominal-induct v rule*:*v.strong-induct*)
case (*V-lit l*)
then show *?case* **using** *assms v.perm-simps*

empty-iff flip-def fresh-def fresh-permute-iff supp-l-empty swap-fresh-fresh v.fresh
by (*metis permute-swap-cancel2 v.distinct*)
next
case (*V-var x*)
then show *?case* **using** *assms v.perm-simps*
empty-iff flip-def fresh-def fresh-permute-iff supp-l-empty swap-fresh-fresh v.fresh
by (*metis permute-swap-cancel2 v.distinct*)
next
case (*V-pair v1 v2*)
have (*V-pair v1 v2* = *V-pair vone vtwo* ⟶ (∃ *v1' v2'*. *va* = *V-pair v1' v2'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'* ∧ (*x* ↔ *c*) · *vtwo* = (*xa* ↔ *c*) · *v2'*)) **proof**
assume *V-pair v1 v2*= *V-pair vone vtwo*
thus (∃ *v1' v2'*. *va* = *V-pair v1' v2'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'* ∧ (*x* ↔ *c*) · *vtwo* = (*xa* ↔ *c*) · *v2'*)
using *V-pair assms*
by (*metis* (*no-types*, *opaque-lifting*) *flip-def permute-swap-cancel v.perm-simps*(*3*))
qed
thus *?case* **using** *V-pair* **by** *auto*
next
case (*V-cons tyid dc v1*)
have (*V-cons tyid dc v1* = *V-cons tyid dc vone* ⟶ (∃ *v1'*. *va* = *V-cons tyid dc v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)) **proof**
assume *as*: *V-cons tyid dc v1* = *V-cons tyid dc vone*
hence (*x* ↔ *c*) · (*V-cons tyid dc vone*) = *V-cons tyid dc* ((*x* ↔ *c*) · *vone*) **proof** −
have (*x* ↔ *c*) · *dc* = *dc* **using** *pure-permute-id* **by** *metis*
moreover have (*x* ↔ *c*) · *tyid* = *tyid* **using** *pure-permute-id* **by** *metis*
ultimately show *?thesis* **using** *v.perm-simps*(*4*) **by** *simp*
qed
then obtain *v1'* **where** (*xa* ↔ *c*) · *va* = *V-cons tyid dc v1'* ∧ (*x* ↔ *c*) · *vone* = *v1'* **using** *assms V-cons*
using *as* **by** *fastforce*
hence *va* = *V-cons tyid dc* ((*xa* ↔ *c*) · *v1'*) ∧ (*x* ↔ *c*) · *vone* = *v1'* **using** *permute-flip-cancel empty-iff flip-def fresh-def supp-b-empty swap-fresh-fresh*
by (*metis pure-fresh v.perm-simps*(*4*))

thus (∃ *v1'*. *va* = *V-cons tyid dc v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)
using *V-cons assms* **by** *simp*
qed
thus *?case* **using** *V-cons* **by** *auto*
next
case (*V-consp tyid dc b v1*)
have (*V-consp tyid dc b v1* = *V-consp tyid dc b vone* ⟶ (∃ *v1'*. *va* = *V-consp tyid dc b v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)) **proof**
assume *as*: *V-consp tyid dc b v1* = *V-consp tyid dc b vone*
hence (*x* ↔ *c*) · (*V-consp tyid dc b vone*) = *V-consp tyid dc b* ((*x* ↔ *c*) · *vone*) **proof** −
have (*x* ↔ *c*) · *dc* = *dc* **using** *pure-permute-id* **by** *metis*
moreover have (*x* ↔ *c*) · *tyid* = *tyid* **using** *pure-permute-id* **by** *metis*
ultimately show *?thesis* **using** *v.perm-simps*(*4*) **by** *simp*
qed
then obtain *v1'* **where** (*xa* ↔ *c*) · *va* = *V-consp tyid dc b v1'* ∧ (*x* ↔ *c*) · *vone* = *v1'* **using** *assms V-consp*
using *as* **by** *fastforce*

hence *va* = *V-consp tyid dc b* ((*xa* ↔ *c*) · *v1'*) ∧ (*x* ↔ *c*) · *vone* = *v1'* **using** *permute-flip-cancel empty-iff flip-def fresh-def supp-b-empty swap-fresh-fresh*
pure-fresh v.perm-simps
by (*metis* (*mono-tags*, *opaque-lifting*))
thus (∃ *v1'*. *va* = *V-consp tyid dc b v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)
using *V-consp assms* **by** *simp*
qed
thus *?case* **using** *V-consp* **by** *auto*
qed

lemma *flip-eq*:
fixes *x*::*x* **and** *xa*::*x* **and** *s*::*'a*::*fs* **and** *sa*::*'a*::*fs*
assumes (∀ *c*. *atom c* ♯ (*s*, *sa*) ⟶ *atom c* ♯ (*x*, *xa*, *s*, *sa*) ⟶ (*x* ↔ *c*) · *s* = (*xa* ↔ *c*) · *sa*) **and** *x* ≠ *xa*
shows (*x* ↔ *xa*) · *s* = *sa*
proof −
have ([[*atom x*]]*lst*. *s* = [[*atom xa*]]*lst*. *sa*) **using** *assms Abs1-eq-iff-all* **by** *simp*
hence (*xa* = *x* ∧ *sa* = *s* ∨ *xa* ≠ *x* ∧ *sa* = (*xa* ↔ *x*) · *s* ∧ *atom xa* ♯ *s*) **using** *assms Abs1-eq-iff*[*of xa sa x s*] **by** *simp*
thus *?thesis* **using** *assms*
by (*metis flip-commute*)
qed

lemma *swap-v-supp*:
fixes *v*::*v* **and** *d*::*x* **and** *z*::*x*
assumes *atom d* ♯ *v*
shows *supp* ((*z* ↔ *d*) · *v*) ⊆ *supp v* − { *atom z* } ∪ { *atom d*}
using *assms*
proof(*nominal-induct v rule*:*v.strong-induct*)
case (*V-lit l*)
then show *?case* **using** *l.supp* **by** (*metis supp-l-empty empty-subsetI l.strong-exhaust pure-supp supp-eqvt v.supp*)
next
case (*V-var x*)
hence *d* ≠ *x* **using** *fresh-def* **by** *fastforce*
thus *?case* **apply**(*cases z* = *x*) **using** *supp-at-base V-var* ‹*d*≠*x*› **by** *fastforce+*
next
case (*V-cons tyid dc v*)
show *?case* **using** *v.supp*(*4*) *pure-supp*
using *V-cons.hyps V-cons.prems fresh-def* **by** *auto*
next
case (*V-consp tyid dc b v*)
show *?case* **using** *v.supp*(*4*) *pure-supp*
using *V-consp.hyps V-consp.prems fresh-def* **by** *auto*
qed(*force+*)

Expressions

lemma *swap-e-supp*:
fixes *e*::*e* **and** *d*::*x* **and** *z*::*x*
assumes *atom d* ♯ *e*
shows *supp* ((*z* ↔ *d*) · *e*) ⊆ *supp e* − { *atom z* } ∪ { *atom d*}
using *assms*

proof(*nominal-induct e rule*:*e.strong-induct*)
  case (*AE-val v*)
  then show *?case* **using** *swap-v-supp* **by** *simp*
next
  case (*AE-app f v*)
  then show *?case* **using** *swap-v-supp* **by** (*simp add*: *pure-supp*)
next
  case (*AE-appP b f v*)
  hence *df*: *atom d* ♯ *v* **using** *fresh-def e.supp* **by** *force*
  have *supp* ((*z* ↔ *d*) · (*AE-appP b f v*)) = *supp* (*AE-appP b f* ((*z* ↔ *d*) · *v*)) **using** *e.supp*
    by (*metis b.eq-iff*(*3*) *b.perm-simps*(*3*) *e.perm-simps*(*3*) *flip-b-id*)
  also have ... = *supp b* ∪ *supp f* ∪ *supp* ((*z* ↔ *d*) · *v*) **using** *e.supp* **by** *auto*
  also have ... ⊆ *supp b* ∪ *supp f* ∪ *supp v* − { *atom z* } ∪ { *atom d*} **using** *swap-v-supp*[*OF df*]
pure-supp **by** *auto*
  finally show *?case* **using** *e.supp* **by** *auto*
next
  case (*AE-op opp v1 v2*)
  hence *df*: *atom d* ♯ *v1* ∧ *atom d* ♯ *v2* **using** *fresh-def e.supp* **by** *force*
  have ((*z* ↔ *d*) · (*AE-op opp v1 v2*)) = *AE-op opp* ((*z* ↔ *d*) · *v1*) ((*z* ↔ *d*) · *v2*) **using**
      e.perm-simps flip-commute opp.perm-simps AE-op opp.strong-exhaust pure-supp
    by (*metis* (*full-types*))

  hence *supp* ((*z* ↔ *d*) · *AE-op opp v1 v2*) = *supp* (*AE-op opp* ((*z* ↔ *d*) ·*v1*) ((*z* ↔ *d*) ·*v2*)) **by** *simp*
  also have ... = *supp* ((*z* ↔ *d*) ·*v1*) ∪ *supp* ((*z* ↔ *d*) ·*v2*) **using** *e.supp*
    by (*metis* (*mono-tags*, *opaque-lifting*) *opp.strong-exhaust opp.supp sup-bot.left-neutral*)
  also have ... ⊆ (*supp v1* − { *atom z* } ∪ { *atom d*}) ∪ (*supp v2* − { *atom z* } ∪ { *atom d*}) **using**
swap-v-supp AE-op df **by** *blast*
  finally show *?case* **using** *e.supp opp.supp* **by** *blast*
next
  case (*AE-fst v*)
  then show *?case* **using** *swap-v-supp* **by** *auto*
next
  case (*AE-snd v*)
  then show *?case* **using** *swap-v-supp* **by** *auto*
next
  case (*AE-mvar u*)
  then show *?case* **using**
    Diff-empty Diff-insert0 Un-upper1 atom-x-sort flip-def flip-fresh-fresh fresh-def set-eq-subset supp-eqvt
swap-set-in-eq
    by (*metis sort-of-atom-eq*)
next
  case (*AE-len v*)
  then show *?case* **using** *swap-v-supp* **by** *auto*
next
  case (*AE-concat v1 v2*)
  then show *?case* **using** *swap-v-supp* **by** *auto*
next
  case (*AE-split v1 v2*)
  then show *?case* **using** *swap-v-supp* **by** *auto*
qed

lemma *swap-ce-supp*:

```
  fixes e::ce and d::x and z::x
  assumes atom d ♯ e
  shows supp ((z ↔ d ) • e) ⊆ supp e − { atom z } ∪ { atom d}
  using assms
proof(nominal-induct e rule:ce.strong-induct)
  case (CE-val v)
  then show ?case using swap-v-supp ce.fresh ce.supp by simp
next
  case (CE-op opp v1 v2)
  hence df: atom d ♯ v1 ∧ atom d ♯ v2 using fresh-def e.supp by force
  have ((z ↔ d ) • (CE-op opp v1 v2)) = CE-op opp ((z ↔ d ) • v1) ((z ↔ d ) • v2) using
      ce.perm-simps flip-commute opp.perm-simps CE-op opp.strong-exhaust x-fresh-b pure-supp
    by (metis (full-types))

  hence supp ((z ↔ d) • CE-op opp v1 v2) = supp (CE-op opp ((z ↔ d) •v1) ((z ↔ d) •v2)) by simp
  also have ... = supp ((z ↔ d) •v1) ∪ supp ((z ↔ d) •v2) using ce.supp
    by (metis (mono-tags, opaque-lifting) opp.strong-exhaust opp.supp sup-bot.left-neutral)
  also have ... ⊆ (supp v1 − { atom z } ∪ { atom d}) ∪ (supp v2 − { atom z } ∪ { atom d}) using
swap-v-supp CE-op df by blast
  finally show ?case using ce.supp opp.supp by blast
next
  case (CE-fst v)
  then show ?case  using ce.supp ce.fresh swap-v-supp by auto
next
  case (CE-snd v)
  then show ?case  using ce.supp ce.fresh swap-v-supp by auto
next
  case (CE-len v)
  then show ?case  using ce.supp ce.fresh swap-v-supp by auto
next
  case (CE-concat v1 v2)
  then show ?case using ce.supp ce.fresh swap-v-supp ce.perm-simps
  proof −
    have ∀ x v xa. ¬ atom (x::x) ♯ (v::v) ∨ supp ((xa ↔ x) • v) ⊆ supp v − {atom xa} ∪ {atom x}
      by (meson swap-v-supp)
    then show ?thesis
      using CE-concat ce.supp by auto
  qed
qed

lemma swap-c-supp:
  fixes c::c and d::x and z::x
  assumes atom d ♯ c
  shows supp ((z ↔ d ) • c) ⊆ supp c − { atom z } ∪ { atom d}
  using assms
proof(nominal-induct c rule:c.strong-induct)
  case (C-eq e1 e2)
  then show ?case using swap-ce-supp by auto
qed(auto+)

lemma type-e-eq:
  assumes atom z ♯ e and atom z′ ♯ e
```

shows $\{\!\!\{\ z : b \ \mid\ [[z]^v]^{ce} == e\ \}\!\!\} = (\{\!\!\{\ z' : b \ \mid\ [[z']^v]^{ce} ==\ \ e\ \}\!\!\})$
by (*auto,metis* (*full-types*) *assms*(*1*) *assms*(*2*) *flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*))

lemma *type-e-eq2*:
assumes *atom* $z \sharp e$ **and** *atom* $z' \sharp e$ **and** $b{=}b'$
shows $\{\!\!\{\ z : b \ \mid\ [[z]^v]^{ce} == e\ \}\!\!\} = (\{\!\!\{\ z' : b' \ \mid\ [[z']^v]^{ce} == e\ \}\!\!\})$
using *assms type-e-eq* **by** *fast*

lemma *e-flip-eq*:
fixes *e*::*e* **and** *ea*::*e*
assumes *atom* $c \sharp (e, ea)$ **and** *atom* $c \sharp (x, xa, e, ea)$ **and** $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
shows $(e = AE\text{-}val\ w \longrightarrow (\exists w'.\ ea = AE\text{-}val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$
$(e = AE\text{-}op\ opp\ v1\ v2 \longrightarrow (\exists v1'\ v2'.\ ea = AS\text{-}op\ opp\ v1'\ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1')$
$\wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2') \vee$
$(e = AE\text{-}fst\ v \longrightarrow (\exists v'.\ ea = AE\text{-}fst\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
$(e = AE\text{-}snd\ v \longrightarrow (\exists v'.\ ea = AE\text{-}snd\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
$(e = AE\text{-}len\ v \longrightarrow (\exists v'.\ ea = AE\text{-}len\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
$(e = AE\text{-}concat\ v1\ v2 \longrightarrow (\exists v1'\ v2'.\ ea = AS\text{-}concat\ v1'\ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1')$
$\wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2') \vee$
$(e = AE\text{-}app\ f\ v \longrightarrow (\exists v'.\ ea = AE\text{-}app\ f\ \ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v'))$
by (*metis assms e.perm-simps permute-flip-cancel2*)

lemma *fresh-opp-all*:
fixes *opp*::*opp*
shows $z \sharp opp$
using *e.fresh opp.exhaust opp.fresh* **by** *metis*

lemma *fresh-e-opp-all*:
shows $(z \sharp v1 \wedge z \sharp v2) = z \sharp AE\text{-}op\ opp\ v1\ v2$
using *e.fresh opp.exhaust opp.fresh fresh-opp-all* **by** *simp*

lemma *fresh-e-opp*:
fixes *z*::*x*
assumes *atom* $z \sharp v1 \wedge$ *atom* $z \sharp v2$
shows *atom* $z \sharp AE\text{-}op\ opp\ v1\ v2$
using *e.fresh opp.exhaust opp.fresh opp.supp* **by** (*metis assms*)

Statements

lemma *branch-s-flip-eq*:
fixes *v*::*v* **and** *va*::*v*
assumes *atom* $c \sharp (v, va)$ **and** *atom* $c \sharp (x, xa, v, va)$ **and** $(x \leftrightarrow c) \cdot s = (xa \leftrightarrow c) \cdot sa$
shows $(s = AS\text{-}val\ w \longrightarrow (\exists w'.\ sa = AS\text{-}val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$
$(s = AS\text{-}seq\ s1\ s2 \longrightarrow (\exists s1'\ s2'.\ sa = AS\text{-}seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x$
$\leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2') \vee$
$(s = AS\text{-}if\ v\ s1\ s2 \longrightarrow (\exists v'\ s1'\ s2'.\ sa = AS\text{-}if\ seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge$
$(x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2' \wedge (x \leftrightarrow c) \cdot c = (xa \leftrightarrow c) \cdot v')$
by (*metis assms s-branch-s-branch-list.perm-simps permute-flip-cancel2*)

2.2 Context Syntax

2.2.1 Datatypes

Type and function/type definition contexts

type-synonym Φ = *fun-def list*
type-synonym Θ = *type-def list*
type-synonym $\mathcal{B}$ = *bv fset*

datatype Γ =
  GNil
  | *GCons* $x{*}b{*}c$ Γ (**infixr** ‹$\#_\Gamma$› *65*)

datatype Δ =
  DNil (‹$[]_\Delta$›)
  | *DCons* $u{*}\tau$ Δ (**infixr** ‹$\#_\Delta$› *65*)

2.2.2 Functions and Lemmas

lemma Γ-*induct* [*case-names GNil GCons*] : P *GNil* $\Longrightarrow$ ($\bigwedge x\ b\ c\ \Gamma'.\ P\ \Gamma' \Longrightarrow P\ ((x,b,c)\ \#_\Gamma\ \Gamma')) \Longrightarrow P\ \Gamma$
proof(*induct* Γ *rule*:Γ.*induct*)
  case *GNil*
  then show *?case* **by** *auto*
next
  case (*GCons x1 x2*)
  then obtain x **and** b **and** c **where** $x1=(x,b,c)$ **using** *prod-cases3* **by** *blast*
  then show *?case* **using** *GCons* **by** *presburger*
qed

instantiation Δ :: *pt*
begin

primrec *permute-*Δ
  where
    DNil-eqvt: $p \cdot DNil = DNil$
  | *DCons-eqvt*: $p \cdot (x\ \#_\Delta\ xs) = p \cdot x\ \#_\Delta\ p \cdot (xs{::}\Delta)$

instance **by** *standard* (*induct-tac* [!] x, *simp-all*)
end

lemmas [*eqvt*] = *permute-*Δ*.simps*

lemma Δ-*induct* [*case-names DNil DCons*] : P *DNil* $\Longrightarrow$ ($\bigwedge u\ t\ \Delta'.\ P\ \Delta' \Longrightarrow P\ ((u,t)\ \#_\Delta\ \Delta')) \Longrightarrow P\ \Delta$
proof(*induct* Δ *rule*: Δ.*induct*)
  case *DNil*
  then show *?case* **by** *auto*
next
  case (*DCons x1 x2*)
  then obtain u **and** t **where** $x1=(u,t)$ **by** *fastforce*
  then show *?case* **using** *DCons* **by** *presburger*

qed

lemma Φ-*induct* [*case-names PNil PConsNone PConsSome*] : P [] ⟹ (⋀*f x b c* τ *s*′ Φ′. P Φ′ ⟹ P ((*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*′))) # Φ′)) ⟹
(⋀*f bv x b c* τ *s*′ Φ′. P Φ′ ⟹ P ((*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*′))) # Φ′)) ⟹ P Φ
proof(*induct* Φ *rule*: *list.induct*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons x1 x2*)
  then obtain *f* **and** *t* **where** *ft*: *x1* = (*AF-fundef f t*)
    by (*meson fun-def.exhaust*)
  then show *?case* **proof**(*nominal-induct t rule*:*fun-typ-q.strong-induct*)
    case (*AF-fun-typ-some bv ft*)
    then show *?case* **using** *Cons ft*
      by (*metis fun-typ.exhaust*)
  next
    case (*AF-fun-typ-none ft*)
    then show *?case* **using** *Cons ft*
      by (*metis fun-typ.exhaust*)
  qed
qed

lemma Θ-*induct* [*case-names TNil AF-typedef AF-typedef-poly*] : P [] ⟹ (⋀*tid dclist* Θ′. P Θ′ ⟹ P ((*AF-typedef tid dclist*) # Θ′)) ⟹
(⋀*tid bv dclist* Θ′. P Θ′ ⟹ P ((*AF-typedef-poly tid bv dclist*) # Θ′)) ⟹ P Θ
proof(*induct* Θ *rule*: *list.induct*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons td T*)
  show *?case* **by**(*cases td rule*: *type-def.exhaust*, (*simp add*: *Cons*)+)
qed

instantiation Γ :: *pt*
begin

primrec *permute*-Γ
  where
    GNil-eqvt: *p* · *GNil* = *GNil*
  | *GCons-eqvt*: *p* · (*x* $\#_\Gamma$ *xs*) = *p* · *x* $\#_\Gamma$ *p* · (*xs*::Γ)

instance by *standard* (*induct-tac* [!] *x*, *simp-all*)
end

lemmas [*eqvt*] = *permute*-Γ.*simps*

lemma *G-cons-eqvt*[*simp*]:
  fixes Γ::Γ
  shows *p* · ((*x*,*b*,*c*) $\#_\Gamma$ Γ) = ((*p* · *x*, *p* · *b* , *p* · *c*) $\#_\Gamma$ (*p* · Γ)) (**is** *?A* = *?B*)

using *Cons-eqvt triple-eqvt supp-b-empty* **by** *simp*

lemma *G-cons-flip*[*simp*]:
fixes x::x **and** Γ::Γ
shows $(x \leftrightarrow x') \cdot ((x'',b,c) \#_\Gamma \Gamma) = (((x \leftrightarrow x') \cdot x'', b , (x \leftrightarrow x') \cdot c) \#_\Gamma ((x \leftrightarrow x') \cdot \Gamma))$
using *Cons-eqvt triple-eqvt supp-b-empty* **by** *auto*

lemma *G-cons-flip-fresh*[*simp*]:
fixes x::x **and** Γ::Γ
assumes *atom* $x \sharp (c,\Gamma)$ **and** *atom* $x' \sharp (c,\Gamma)$
shows $(x \leftrightarrow x') \cdot ((x',b,c) \#_\Gamma \Gamma) = ((x, b , c) \#_\Gamma \Gamma)$
using *G-cons-flip flip-fresh-fresh assms* **by** *force*

lemma *G-cons-flip-fresh2*[*simp*]:
fixes x::x **and** Γ::Γ
assumes *atom* $x \sharp (c,\Gamma)$ **and** *atom* $x' \sharp (c,\Gamma)$
shows $(x \leftrightarrow x') \cdot ((x,b,c) \#_\Gamma \Gamma) = ((x', b , c) \#_\Gamma \Gamma)$
using *G-cons-flip flip-fresh-fresh assms* **by** *force*

lemma *G-cons-flip-fresh3*[*simp*]:
fixes x::x **and** Γ::Γ
assumes *atom* $x \sharp \Gamma$ **and** *atom* $x' \sharp \Gamma$
shows $(x \leftrightarrow x') \cdot ((x',b,c) \#_\Gamma \Gamma) = ((x, b , (x \leftrightarrow x') \cdot c) \#_\Gamma \Gamma)$
using *G-cons-flip flip-fresh-fresh assms* **by** *force*

lemma *neq-GNil-conv*: $(xs \neq GNil) = (\exists\, y\ ys.\ xs = y \#_\Gamma ys)$
by (*induct xs*) *auto*

nominal-function *toList* :: $\Gamma \Rightarrow (x*b*c)$ *list* **where**
toList GNil = []
| *toList* (*GCons xbc G*) = *xbc*#(*toList G*)
apply (*auto*, *simp add*: *eqvt-def toList-graph-aux-def*)
using *neq-GNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*)
by *lexicographic-order*

nominal-function *toSet* :: $\Gamma \Rightarrow (x*b*c)$ *set* **where**
toSet GNil = {}
| *toSet* (*GCons xbc G*) = {*xbc*} ∪ (*toSet G*)
apply (*auto*,*simp add*: *eqvt-def toSet-graph-aux-def*)
using *neq-GNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*)
by *lexicographic-order*

nominal-function *append-g* :: $\Gamma \Rightarrow \Gamma \Rightarrow \Gamma$ (**infixr** ‹@› *65*) **where**
append-g GNil g = *g*
| *append-g* ($xbc \#_\Gamma g1$) *g2* = ($xbc \#_\Gamma$ (*g1*@*g2*))
apply (*auto*,*simp add*: *eqvt-def append-g-graph-aux-def*)
using *neq-GNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *dom* :: $\Gamma \Rightarrow x$ *set* **where**

dom Γ = (*fst*‘ (*toSet* Γ))
  apply *auto*
unfolding *eqvt-def dom-graph-aux-def lfp-eqvt toSet.eqvt* **by** *simp*
nominal-termination (*eqvt*) **by** *lexicographic-order*

Use of this is sometimes mixed in with use of freshness and support for the context however it makes it clear that for immutable variables, the context is 'self-supporting'

nominal-function *atom-dom* :: Γ ⇒ *atom set* **where**
atom-dom Γ = *atom*‘(*dom* Γ)
  apply *auto*
unfolding *eqvt-def atom-dom-graph-aux-def lfp-eqvt toSet.eqvt* **by** *simp*
nominal-termination (*eqvt*) **by** *lexicographic-order*

2.2.3 Immutable Variable Context Lemmas

lemma *append-GNil*[*simp*]:
GNil @ *G* = *G*
by *simp*

lemma *append-g-toSetU* [*simp*]: *toSet* (*G1*@*G2*) = *toSet G1* ∪ *toSet G2*
by(*induct G1*, *auto*+)

lemma *supp-GNil*:
shows *supp GNil* = {}
by (*simp add*: *supp-def*)

lemma *supp-GCons*:
fixes *xs*::Γ
shows *supp* (*x* $\#_\Gamma$ *xs*) = *supp x* ∪ *supp xs*
by (*simp add*: *supp-def Collect-imp-eq Collect-neg-eq*)

lemma *atom-dom-eq*[*simp*]:
fixes *G*::Γ
shows *atom-dom* ((*x*, *b*, *c*) $\#_\Gamma$ *G*) = *atom-dom* ((*x*, *b*, *c'*) $\#_\Gamma$ *G*)
using *atom-dom.simps toSet.simps* **by** *simp*

lemma *dom-append*[*simp*]:
atom-dom (Γ@Γ') = *atom-dom* Γ ∪ *atom-dom* Γ'
using *image-Un append-g-toSetU atom-dom.simps dom.simps* **by** *metis*

lemma *dom-cons*[*simp*]:
atom-dom ((*x*,*b*,*c*) $\#_\Gamma$ *G*) = { *atom x* } ∪ *atom-dom G*
using *image-Un append-g-toSetU atom-dom.simps* **by** *auto*

lemma *fresh-GNil*[*ms-fresh*]:
shows *a* ♯ *GNil*
by (*simp add*: *fresh-def supp-GNil*)

lemma *fresh-GCons*[*ms-fresh*]:
fixes *xs*::Γ
shows *a* ♯ (*x* $\#_\Gamma$ *xs*) ⟷ *a* ♯ *x* ∧ *a* ♯ *xs*
by (*simp add*: *fresh-def supp-GCons*)

lemma *dom-supp-g*[*simp*]:
  atom-dom G $\subseteq$ *supp G*
  apply(*induct G rule*: Γ-*induct*,*simp*)
  using *supp-at-base supp-Pair atom-dom.simps supp-GCons* **by** *fastforce*

lemma *fresh-append-g*[*ms-fresh*]:
  fixes *xs*::Γ
  shows $a \sharp (xs\ @\ ys) \longleftrightarrow a \sharp xs \wedge a \sharp ys$
  by (*induct xs*) (*simp-all add*: *fresh-GNil fresh-GCons*)

lemma *append-g-assoc*:
  fixes *xs*::Γ
  shows $(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
  by (*induct xs*) *simp-all*

lemma *append-g-inside*:
  fixes *xs*::Γ
  shows $xs\ @\ (x\ \#_\Gamma\ ys) = (xs\ @\ (x\ \#_\Gamma\ GNil))\ @\ ys$
  by(*induct xs*,*auto*+)

lemma *finite*-Γ:
  finite (*toSet* Γ)
  by(*induct* Γ *rule*: Γ-*induct*,*auto*)

lemma *supp*-Γ:
  supp Γ = *supp* (*toSet* Γ)
proof(*induct* Γ *rule*: Γ-*induct*)
  case *GNil*
  then show *?case* **using** *supp-GNil toSet.simps*
    by (*simp add*: *supp-set-empty*)
next
  case (*GCons x b c* Γ')
  then show *?case* **using** *supp-GCons toSet.simps finite*-Γ *supp-of-finite-union*
    using *supp-of-finite-insert* **by** *fastforce*
qed

lemma *supp-of-subset*:
  fixes *G*::(*'a*::*fs set*)
  assumes *finite G* **and** *finite G'* **and** $G \subseteq G'$
  shows *supp G* $\subseteq$ *supp G'*
  using *supp-of-finite-sets assms* **by** (*metis subset-Un-eq supp-of-finite-union*)

lemma *supp-weakening*:
  assumes *toSet G* $\subseteq$ *toSet G'*
  shows *supp G* $\subseteq$ *supp G'*
  using *supp*-Γ *finite*-Γ **by** (*simp add*: *supp-of-subset assms*)

lemma *fresh-weakening*[*ms-fresh*]:
  assumes *toSet G* $\subseteq$ *toSet G'* **and** $x \sharp G'$
  shows $x \sharp G$
proof(*rule ccontr*)

assume ¬ *x* ♯ *G*
hence *x* ∈ *supp G* **using** *fresh-def* **by** *auto*
hence *x* ∈ *supp G′* **using** *supp-weakening assms* **by** *auto*
thus *False* **using** *fresh-def assms* **by** *auto*
qed

instance Γ :: *fs*
by (*standard*, *induct-tac x*, *simp-all add*: *supp-GNil supp-GCons finite-supp*)

lemma *fresh-gamma-elem*:
fixes Γ::Γ
assumes *a* ♯ Γ
and *e* ∈ *toSet* Γ
shows *a* ♯ *e*
using *assms* **by**(*induct* Γ,*auto simp add*: *fresh-GCons*)

lemma *fresh-gamma-append*:
fixes *xs*::Γ
shows *a* ♯ (*xs* @ *ys*) ⟷ *a* ♯ *xs* ∧ *a* ♯ *ys*
by (*induct xs*, *simp-all add*: *fresh-GNil fresh-GCons*)

lemma *supp-triple*[*simp*]:
shows *supp* (*x*, *y*, *z*) = *supp x* ∪ *supp y* ∪ *supp z*
proof −
have *supp* (*x*,*y*,*z*) = *supp* (*x*,(*y*,*z*)) **by** *auto*
hence *supp* (*x*,*y*,*z*) = *supp x* ∪ (*supp y* ∪ *supp z*) **using** *supp-Pair* **by** *metis*
thus *?thesis* **by** *auto*
qed

lemma *supp-append-g*:
fixes *xs*::Γ
shows *supp* (*xs* @ *ys*) = *supp xs* ∪ *supp ys*
by(*induct xs*,*auto simp add*: *supp-GNil supp-GCons*)

lemma *fresh-in-g*[*simp*]:
fixes Γ::Γ **and** *x′*::*x*
shows *atom x′* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ = (*atom x′* ∉ *supp* Γ′ ∪ *supp x* ∪ *supp b0* ∪ *supp c0* ∪ *supp* Γ)
proof −
have *atom x′* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ = (*atom x′* ∉ *supp* (Γ′ @((*x*,*b0*,*c0*) $\#_\Gamma$ Γ)))
using *fresh-def* **by** *auto*
also have ... = (*atom x′* ∉ *supp* Γ′ ∪ *supp* ((*x*,*b0*,*c0*) $\#_\Gamma$ Γ)) **using** *supp-append-g* **by** *fast*
also have ... = (*atom x′* ∉ *supp* Γ′ ∪ *supp x* ∪ *supp b0* ∪ *supp c0* ∪ *supp* Γ) **using** *supp-GCons supp-append-g supp-triple* **by** *auto*
finally show *?thesis* **by** *fast*
qed

lemma *fresh-suffix*[*ms-fresh*]:
fixes Γ::Γ
assumes *atom x* ♯ Γ′@Γ
shows *atom x* ♯ Γ
using *assms* **by**(*induct* Γ′ *rule*: Γ*-induct*, *auto simp add*: *append-g.simps fresh-GCons*)

lemma *not-GCons-self* [*simp*]:
fixes *xs*::Γ
shows $xs \neq x \#_{\Gamma} xs$
by (*induct xs*) *auto*

lemma *not-GCons-self2* [*simp*]:
fixes *xs*::Γ
shows $x \#_{\Gamma} xs \neq xs$
by (*rule not-GCons-self* [*symmetric*])

lemma *fresh-restrict*:
fixes *y*::*x* **and** Γ::Γ
assumes *atom y* ♯ (Γ′ @ (*x*, *b*, *c*) $\#_{\Gamma}$ Γ)
shows *atom y* ♯ (Γ′@Γ)
using *assms* **by**(*induct* Γ′ *rule*: Γ*-induct*, *auto simp add*:*fresh-GCons fresh-GNil*)

lemma *fresh-dom-free*:
assumes *atom x* ♯ Γ
shows (*x*,*b*,*c*) ∉ *toSet* Γ
using *assms* **proof**(*induct* Γ *rule*: Γ*-induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x′ b′ c′* Γ′)
hence *x*≠*x′* **using** *fresh-def fresh-GCons fresh-Pair supp-at-base* **by** *blast*
moreover have *atom x* ♯ Γ′ **using** *fresh-GCons GCons* **by** *auto*
ultimately show *?case* **using** *toSet.simps GCons* **by** *auto*
qed

lemma Γ*-set-intros*: *x* ∈ *toSet* ($x \#_{\Gamma} xs$) **and** *y* ∈ *toSet xs* ⟹ *y* ∈ *toSet* ($x \#_{\Gamma} xs$)
by *simp*+

lemma *fresh-dom-free2*:
assumes *atom x* ∉ *atom-dom* Γ
shows (*x*,*b*,*c*) ∉ *toSet* Γ
using *assms* **proof**(*induct* Γ *rule*: Γ*-induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x′ b′ c′* Γ′)
hence *x*≠*x′* **using** *fresh-def fresh-GCons fresh-Pair supp-at-base* **by** *auto*
moreover have *atom x* ∉ *atom-dom* Γ′ **using** *fresh-GCons GCons* **by** *auto*
ultimately show *?case* **using** *toSet.simps GCons* **by** *auto*
qed

2.2.4 Mutable Variable Context Lemmas

lemma *supp-DNil*:
shows *supp DNil* = {}
by (*simp add*: *supp-def*)

lemma *supp-DCons*:

fixes *xs*::Δ
shows *supp* (*x* $\#_\Delta$ *xs*) = *supp x* ∪ *supp xs*
by (*simp add*: *supp-def Collect-imp-eq Collect-neg-eq*)

lemma *fresh-DNil*[*ms-fresh*]:
shows *a* ♯ *DNil*
by (*simp add*: *fresh-def supp-DNil*)

lemma *fresh-DCons*[*ms-fresh*]:
fixes *xs*::Δ
shows *a* ♯ (*x* $\#_\Delta$ *xs*) ⟷ *a* ♯ *x* ∧ *a* ♯ *xs*
by (*simp add*: *fresh-def supp-DCons*)

instance Δ :: *fs*
by (*standard*, *induct-tac x*, *simp-all add*: *supp-DNil supp-DCons finite-supp*)

2.2.5 Lookup Functions

nominal-function *lookup* :: Γ ⇒ *x* ⇒ (*b*∗*c*) *option* **where**
lookup GNil x = *None*
| *lookup* ((*x*,*b*,*c*)$\#_\Gamma$*G*) *y* = (*if x*=*y then Some* (*b*,*c*) *else lookup G y*)
by (*auto*,*simp add*: *eqvt-def lookup-graph-aux-def*, *metis neq-GNil-conv surj-pair*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *replace-in-g* :: Γ ⇒ *x* ⇒ *c* ⇒ Γ (‹-[-⟼-]› [*1000*,*0*,*0*] *200*) **where**
replace-in-g GNil - - = *GNil*
| *replace-in-g* ((*x*,*b*,*c*)$\#_\Gamma$*G*) *x′ c′* = (*if x*=*x′ then* ((*x*,*b*,*c′*)$\#_\Gamma$*G*) *else* (*x*,*b*,*c*)$\#_\Gamma$(*replace-in-g G x′ c′*))
apply(*auto*,*simp add*: *eqvt-def replace-in-g-graph-aux-def*)
using *surj-pair* Γ.*exhaust* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

Functions for looking up data-constructors in the Pi context

nominal-function *lookup-fun* :: Φ ⇒ *f* ⇒ *fun-def option* **where**
lookup-fun [] *g* = *None*
| *lookup-fun* ((*AF-fundef f ft*)#Π) *g* = (*if* (*f*=*g*) *then Some* (*AF-fundef f ft*) *else lookup-fun* Π *g*)
apply(*auto*,*simp add*: *eqvt-def lookup-fun-graph-aux-def*)
by (*metis fun-def*.*exhaust neq-Nil-conv*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *lookup-td* :: Θ ⇒ *string* ⇒ *type-def option* **where**
lookup-td [] *g* = *None*
| *lookup-td* ((*AF-typedef s lst*) # (Θ::Θ)) *g* = (*if* (*s* = *g*) *then Some* (*AF-typedef s lst*) *else lookup-td* Θ *g*)
| *lookup-td* ((*AF-typedef-poly s bv lst*) # (Θ::Θ)) *g* = (*if* (*s* = *g*) *then Some* (*AF-typedef-poly s bv lst*) *else lookup-td* Θ *g*)
apply(*auto*,*simp add*: *eqvt-def lookup-td-graph-aux-def*)
by (*metis type-def*.*exhaust neq-Nil-conv*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *name-of-type* ::*type-def* ⇒ *f* **where**
name-of-type (*AF-typedef f -*) = *f*
| *name-of-type* (*AF-typedef-poly f - -*) = *f*
apply(*auto*,*simp add*: *eqvt-def name-of-type-graph-aux-def*)

using *type-def.exhaust* **by** *blast*
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *name-of-fun* ::*fun-def* ⇒ *f* **where**
name-of-fun (*AF-fundef f ft*) = *f*
apply(*auto*,*simp add*: *eqvt-def name-of-fun-graph-aux-def*)
using *fun-def.exhaust* **by** *blast*
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *remove2* :: *'a*::*pt* ⇒ *'a list* ⇒ *'a list* **where**
remove2 x [] = [] |
remove2 x (*y* # *xs*) = (*if x* = *y then xs else y* # *remove2 x xs*)
by (*simp add*: *eqvt-def remove2-graph-aux-def*,*auto*+,*meson list.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *base-for-lit* :: *l* ⇒ *b* **where**
base-for-lit (*L-true*) = *B-bool*
| *base-for-lit* (*L-false*) = *B-bool*
| *base-for-lit* (*L-num n*) = *B-int*
| *base-for-lit* (*L-unit*) = *B-unit*
| *base-for-lit* (*L-bitvec v*) = *B-bitvec*
apply (*auto simp*: *eqvt-def base-for-lit-graph-aux-def*)
using *l.strong-exhaust* **by** *blast*
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *neq-DNil-conv*: (*xs* ≠ *DNil*) = (∃ *y ys*. *xs* = *y* $\#_\Delta$ *ys*)
by (*induct xs*) *auto*

nominal-function *setD* :: Δ ⇒ (*u*∗*τ*) *set* **where**
setD DNil = {}
| *setD* (*DCons xbc G*) = {*xbc*} ∪ (*setD G*)
apply (*auto*,*simp add*: *eqvt-def setD-graph-aux-def*)
using *neq-DNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *eqvt-triple*:
fixes *y*::*'a*::*at* **and** *ya*::*'a*::*at* **and** *xa*::*'c*::*at* **and** *va*::*'d*::*fs* **and** *s*::*s* **and** *sa*::*s* **and** *f*::*s*∗*'c*∗*'d* ⇒ *s*
assumes *atom y* ♯ (*xa*, *va*) **and** *atom ya* ♯ (*xa*, *va*) **and**
∀ *c*. *atom c* ♯ (*s*, *sa*) ⟶ *atom c* ♯ (*y*, *ya*, *s*, *sa*) ⟶ (*y* ↔ *c*) • *s* = (*ya* ↔ *c*) • *sa*
and *eqvt-at f* (*s*,*xa*,*va*) **and** *eqvt-at f* (*sa*,*xa*,*va*) **and**
atom c ♯ (*s*, *va*, *xa*, *sa*) **and** *atom c* ♯ (*y*, *ya*, *f* (*s*, *xa*, *va*), *f* (*sa*, *xa*, *va*))
shows (*y* ↔ *c*) • *f* (*s*, *xa*, *va*) = (*ya* ↔ *c*) • *f* (*sa*, *xa*, *va*)
proof −
have (*y* ↔ *c*) • *f* (*s*, *xa*, *va*) = *f* ((*y* ↔ *c*) • (*s*,*xa*,*va*)) **using** *assms eqvt-at-def* **by** *metis*
also have ... = *f* ((*y* ↔ *c*) • *s*, (*y* ↔ *c*) • *xa* ,(*y* ↔ *c*) • *va*) **by** *auto*
also have ... = *f* ((*ya* ↔ *c*) • *sa*, (*ya* ↔ *c*) • *xa* ,(*ya* ↔ *c*) • *va*) **proof** −
have (*y* ↔ *c*) • *s* = (*ya* ↔ *c*) • *sa* **using** *assms Abs1-eq-iff-all* **by** *auto*
moreover have ((*y* ↔ *c*) • *xa*) = ((*ya* ↔ *c*) • *xa*) **using** *assms flip-fresh-fresh fresh-prodN* **by** *metis*
moreover have ((*y* ↔ *c*) • *va*) = ((*ya* ↔ *c*) • *va*) **using** *assms flip-fresh-fresh fresh-prodN* **by** *metis*
ultimately show *?thesis* **by** *auto*

qed
also have ... = *f* ((*ya* ↔ *c*) · (*sa*,*xa*,*va*)) **by** *auto*
finally show *?thesis* **using** *assms eqvt-at-def* **by** *metis*
qed

2.3 Functions for bit list/vectors

inductive *split* :: *int* ⇒ *bit list* ⇒ *bit list* ∗ *bit list* ⇒ *bool* **where**
split 0 xs ([], *xs*)
| *split m xs* (*ys*,*zs*) ⟹ *split* (*m*+*1*) (*x*#*xs*) ((*x* # *ys*), *zs*)
equivariance *split*
nominal-inductive *split* **.**

lemma *split-concat*:
assumes *split n v* (*v1*,*v2*)
shows *v* = *append v1 v2*
using *assms* **proof**(*induct* (*v1*,*v2*) *arbitrary*: *v1 v2 rule*: *split.inducts*)
case *1*
then show *?case* **by** *auto*
next
case (*2 m xs ys zs x*)
then show *?case* **by** *auto*
qed

lemma *split-n*:
assumes *split n v* (*v1*,*v2*)
shows *0* ≤ *n* ∧ *n* ≤ *int* (*length v*)
using *assms* **proof**(*induct rule*: *split.inducts*)
case (*1 xs*)
then show *?case* **by** *auto*
next
case (*2 m xs ys zs x*)
then show *?case* **by** *auto*
qed

lemma *split-length*:
assumes *split n v* (*v1*,*v2*)
shows *n* = *int* (*length v1*)
using *assms* **proof**(*induct* (*v1*,*v2*) *arbitrary*: *v1 v2 rule*: *split.inducts*)
case (*1 xs*)
then show *?case* **by** *auto*
next
case (*2 m xs ys zs x*)
then show *?case* **by** *auto*
qed

lemma *obtain-split*:
assumes *0* ≤ *n* **and** *n* ≤ *int* (*length bv*)
shows ∃ *bv1 bv2*. *split n bv* (*bv1* , *bv2*)
using *assms* **proof**(*induct bv arbitrary*: *n*)
case *Nil*
then show *?case* **using** *split.intros* **by** *auto*

```
next
  case (Cons b bv)
  show ?case proof(cases n = 0)
    case True
    then show ?thesis using split.intros by auto
  next
    case False
    then obtain m where m:n=m+1 using Cons
      by (metis add.commute add-minus-cancel)
    moreover have 0 ≤ m using False m Cons by linarith
    then obtain bv1 and bv2 where split m bv (bv1 , bv2) using Cons m by force
    hence split n (b # bv) ((b#bv1), bv2) using m split.intros by auto
    then show ?thesis by auto
  qed
qed

end
```

Chapter 3

Immutable Variable Substitution

Substitution involving immutable variables. We define a class and instances for all of the term forms

3.1 Class

class *has-subst-v* = *fs* +
fixes *subst-v* :: *'a::fs* ⇒ *x* ⇒ *v* ⇒ *'a::fs* (‹-[-::=-]$_v$› [*1000*,*50*,*50*] *1000*)
assumes *fresh-subst-v-if*: *y* ♯ (*subst-v a x v*) ⟷ (*atom x* ♯ *a* ∧ *y* ♯ *a*) ∨ (*y* ♯ *v* ∧ (*y* ♯ *a* ∨ *y* = *atom x*))
and *forget-subst-v*[*simp*]: *atom x* ♯ *a* ⟹ *subst-v a x v* = *a*
and *subst-v-id*[*simp*]: *subst-v a x* (*V-var x*) = *a*
and *eqvt*[*simp*,*eqvt*]: (*p*::*perm*) • (*subst-v a x v*) = (*subst-v* (*p* • *a*) (*p* •*x*) (*p* •*v*))
and *flip-subst-v*[*simp*]: *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) • *c*) = *c*[*z*::=[*x*]^v]$_v$
and *subst-v-simple-commute*[*simp*]: *atom x* ♯ *c* ⟹(*c*[*z*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ = *c*[*z*::=*b*]$_v$
begin

lemma *subst-v-flip-eq-one*:
fixes *z1*::*x* **and** *z2*::*x* **and** *x1*::*x* **and** *x2*::*x*
assumes [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
and *atom x1* ♯ (*z1*,*z2*,*c1*,*c2*)
shows (*c1*[*z1*::=[*x1*]^v]$_v$) = (*c2*[*z2*::=[*x1*]^v]$_v$)
proof −
have (*c1*[*z1*::=[*x1*]^v]$_v$) = (*x1* ↔ *z1*) • *c1* **using** *assms flip-subst-v* **by** *auto*
moreover have (*c2*[*z2*::=[*x1*]^v]$_v$) = (*x1* ↔ *z2*) • *c2* **using** *assms flip-subst-v* **by** *auto*
ultimately show *?thesis* **using** *Abs1-eq-iff-all*(*3*)[*of z1 c1 z2 c2 z1*] *assms*
by (*metis Abs1-eq-iff-fresh*(*3*) *flip-commute*)
qed

lemma *subst-v-flip-eq-two*:
fixes *z1*::*x* **and** *z2*::*x* **and** *x1*::*x* **and** *x2*::*x*
assumes [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
shows (*c1*[*z1*::=*b*]$_v$) = (*c2*[*z2*::=*b*]$_v$)
proof −
obtain *x*::*x* **where** ∗:*atom x* ♯ (*z1*,*z2*,*c1*,*c2*) **using** *obtain-fresh* **by** *metis*
hence (*c1*[*z1*::=[*x*]^v]$_v$) = (*c2*[*z2*::=[*x*]^v]$_v$) **using** *subst-v-flip-eq-one*[*OF assms*, *of x*] **by** *metis*
hence (*c1*[*z1*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ = (*c2*[*z2*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ **by** *auto*

thus *?thesis* **using** *subst-v-simple-commute* ∗ *fresh-prod4* **by** *metis*
qed

lemma *subst-v-flip-eq-three*:
assumes $[[atom\ z1]]lst.\ c1 = [[atom\ z1']]lst.\ c1'$ **and** $atom\ x \sharp c1$ **and** $atom\ x' \sharp (x,z1,z1', c1, c1')$
shows $(x \leftrightarrow x') \cdot (c1[z1::=[x]^v]_v) = c1'[z1'::=[x']^v]_v$
proof −
have $atom\ x' \sharp c1[z1::=[x]^v]_v$ **using** *assms fresh-subst-v-if* **by** *simp*
hence $(x \leftrightarrow x') \cdot (c1[z1::=[x]^v]_v) = c1[z1::=[x]^v]_v[x::=[x']^v]_v$ **using** *flip-subst-v*[*of x′ c1*$[z1::=[x]^v]_v$ *x*] *flip-commute* **by** *metis*
also have ... $= c1[z1::=[x']^v]_v$ **using** *subst-v-simple-commute fresh-prod4 assms* **by** *auto*
also have ... $= c1'[z1'::=[x']^v]_v$ **using** *subst-v-flip-eq-one*[*of z1 c1 z1′ c1′ x′*] **using** *assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed

end

3.2 Values

nominal-function
subst-vv :: $v \Rightarrow x \Rightarrow v \Rightarrow v$ **where**
subst-vv (*V-lit l*) *x v* = *V-lit l*
| *subst-vv* (*V-var y*) *x v* = (*if x = y then v else V-var y*)
| *subst-vv* (*V-cons tyid c v′*) *x v* = *V-cons tyid c* (*subst-vv v′ x v*)
| *subst-vv* (*V-consp tyid c b v′*) *x v* = *V-consp tyid c b* (*subst-vv v′ x v*)
| *subst-vv* (*V-pair v1 v2*) *x v* = *V-pair* (*subst-vv v1 x v*) (*subst-vv v2 x v*)
by(*auto simp*: *eqvt-def subst-vv-graph-aux-def*, *metis v.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-vv-abbrev :: $v \Rightarrow x \Rightarrow v \Rightarrow v$ (‹-[-::=-]$_{vv}$› [*1000,50,50*] *1000*)
where
$v[x::=v']_{vv} \equiv$ *subst-vv v x v′*

lemma *fresh-subst-vv-if* [*simp*]:
$j \sharp t[i::=x]_{vv} = ((atom\ i \sharp t \wedge j \sharp t) \vee (j \sharp x \wedge (j \sharp t \vee j = atom\ i)))$
using *supp-l-empty* **apply** (*induct t rule*: *v.induct*,*auto simp add*: *subst-vv.simps fresh-def*, *auto*)
by (*simp add*: *supp-at-base* |*metis b.supp supp-b-empty*)+

lemma *forget-subst-vv* [*simp*]: $atom\ a \sharp tm \Longrightarrow tm[a::=x]_{vv} = tm$
by (*induct tm rule*: *v.induct*) (*simp-all add*: *fresh-at-base*)

lemma *subst-vv-id* [*simp*]: $tm[a::=V\text{-}var\ a]_{vv} = tm$
by (*induct tm rule*: *v.induct*) *simp-all*

lemma *subst-vv-commute* [*simp*]:
$atom\ j \sharp tm \Longrightarrow tm[i::=t]_{vv}[j::=u]_{vv} = tm[i::=t[j::=u]_{vv}]_{vv}$
by (*induct tm rule*: *v.induct*) (*auto simp*: *fresh-Pair*)

lemma *subst-vv-commute-full* [*simp*]:
$atom\ j \sharp t \Longrightarrow atom\ i \sharp u \Longrightarrow i \neq j \Longrightarrow tm[i::=t]_{vv}[j::=u]_{vv} = tm[j::=u]_{vv}[i::=t]_{vv}$
by (*induct tm rule*: *v.induct*) *auto*

lemma *subst-vv-var-flip*[*simp*]:
fixes *v*::*v*
assumes *atom y* ♯ *v*
shows $(y \leftrightarrow x) \cdot v = v\ [x::=V\text{-}var\ y]_{vv}$
using *assms* **apply**(*induct v rule*:*v.induct*)
apply *auto*
using *l.fresh l.perm-simps l.strong-exhaust supp-l-empty permute-pure permute-list.simps fresh-def flip-fresh-fresh* **apply** *fastforce*
using *permute-pure* **apply** *blast*+
done

instantiation *v* :: *has-subst-v*
begin

definition
subst-v = *subst-vv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*v*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-vv-if*[*of j t i x*] *subst-v-v-def* **by** *metis*

fix *a*::*x* **and** *tm*::*v* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-vv subst-v-v-def* **by** *simp*

fix *a*::*x* **and** *tm*::*v*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-vv-id subst-v-v-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*v*
show *p* · *subst-v t1 x1 v* = *subst-v* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
using *subst-v-v-def* **by** *simp*

fix *x*::*x* **and** *c*::*v* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ $((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$
using *subst-v-v-def* **by** *simp*

fix *x*::*x* **and** *c*::*v* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ $c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v$
using *subst-v-v-def* **by** *simp*
qed

end

3.3 Expressions

nominal-function *subst-ev* :: *e* ⇒ *x* ⇒ *v* ⇒ *e* **where**
subst-ev ((*AE-val v′*)) *x v* = ((*AE-val* (*subst-vv v′ x v*)))
| *subst-ev* ((*AE-app f v′*)) *x v* = ((*AE-app f* (*subst-vv v′ x v*)))
| *subst-ev* ((*AE-appP f b v′*)) *x v* = ((*AE-appP f b* (*subst-vv v′ x v*)))
| *subst-ev* ((*AE-op opp v1 v2*)) *x v* = ((*AE-op opp* (*subst-vv v1 x v*) (*subst-vv v2 x v*)))

```
| subst-ev  [#1 v']^e x v = [#1 (subst-vv v' x v )]^e
| subst-ev  [#2 v']^e x v = [#2 (subst-vv v' x v )]^e
| subst-ev  ( (AE-mvar u)) x v = AE-mvar u
| subst-ev  [| v' |]^e x v = [| (subst-vv  v' x v ) |]^e
| subst-ev  ( AE-concat v1 v2) x v = AE-concat (subst-vv v1 x v ) (subst-vv v2 x v )
| subst-ev  ( AE-split v1 v2) x v = AE-split (subst-vv v1 x v ) (subst-vv v2 x v )
  by(simp add: eqvt-def subst-ev-graph-aux-def,auto)(meson e.strong-exhaust)

nominal-termination (eqvt) by lexicographic-order

abbreviation
  subst-ev-abbrev :: e ⇒ x ⇒ v ⇒ e (‹-[-::=-]_ev› [1000,50,50] 500)
  where
    e[x::=v']_ev  ≡ subst-ev e x v'

lemma size-subst-ev [simp]: size ( subst-ev A i x) = size A
  apply (nominal-induct A avoiding: i x rule: e.strong-induct)
  by auto

lemma forget-subst-ev [simp]: atom a ♯ A ⟹ subst-ev A a x  = A
  apply (nominal-induct A avoiding: a x rule: e.strong-induct)
  by (auto simp: fresh-at-base)

lemma subst-ev-id [simp]: subst-ev A a (V-var a)  = A
  by (nominal-induct A avoiding: a rule: e.strong-induct) (auto simp: fresh-at-base)

lemma fresh-subst-ev-if [simp]:
  j ♯ (subst-ev A i x ) = ((atom i ♯ A ∧ j ♯ A) ∨ (j ♯ x ∧ (j ♯ A ∨ j = atom i)))
  apply (induct A rule: e.induct)
  unfolding subst-ev.simps fresh-subst-vv-if apply auto+
  using pure-fresh fresh-opp-all apply metis+
  done

lemma subst-ev-commute [simp]:
  atom j ♯ A ⟹ (A[i::=t]_ev)[j::=u]_ev = A[i::=t[j::=u]_vv]_ev
  by (nominal-induct A avoiding: i j t u rule: e.strong-induct) (auto simp: fresh-at-base)

lemma subst-ev-var-flip[simp]:
  fixes e::e and y::x and x::x
  assumes atom y ♯ e
  shows (y ↔ x) • e = e [x::=V-var y]_ev
  using assms apply(nominal-induct e rule:e.strong-induct)
          apply (simp add: subst-v-v-def)
       apply (metis (mono-tags, lifting) b.eq-iff b.perm-simps e.fresh e.perm-simps flip-b-id subst-ev.simps
subst-vv-var-flip)
      apply (metis (mono-tags, lifting) b.eq-iff b.perm-simps e.fresh e.perm-simps flip-b-id subst-ev.simps
subst-vv-var-flip)
  subgoal for x
    apply (rule-tac y=x in  opp.strong-exhaust)
    using  subst-vv-var-flip flip-def by (simp add: flip-def permute-pure)+
  using  subst-vv-var-flip flip-def by (simp add: flip-def permute-pure)+
```

lemma *subst-ev-flip*:
  fixes *e*::*e* **and** *ea*::*e* **and** *c*::*x*
  assumes *atom c* ♯ (*e*, *ea*) **and** *atom c* ♯ (*x*, *xa*, *e*, *ea*) **and** $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
  shows $e[x{::=}v']_{ev} = ea[xa{::=}v']_{ev}$
proof −
  have $e[x{::=}v']_{ev} = (e[x{::=}V\text{-}var\ c]_{ev})[c{::=}v']_{ev}$ **using** *subst-ev-commute assms* **by** *simp*
  also have ... $= ((c \leftrightarrow x) \cdot e)[c{::=}v']_{ev}$ **using** *subst-ev-var-flip assms* **by** *simp*
  also have ... $= ((c \leftrightarrow xa) \cdot ea)[c{::=}v']_{ev}$ **using** *assms flip-commute* **by** *metis*
  also have ... $= ea[xa{::=}v']_{ev}$ **using** *subst-ev-var-flip assms* **by** *simp*
  finally show *?thesis* **by** *auto*
qed

lemma *subst-ev-var*[*simp*]:
  $(AE\text{-}val\ (V\text{-}var\ x))[x{::=}[z]^v]_{ev} = AE\text{-}val\ (V\text{-}var\ z)$
  by *auto*

instantiation *e* :: *has-subst-v*
begin

definition
  subst-v = *subst-ev*

instance proof
  fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*e*
  show $(j \sharp subst\text{-}v\ t\ i\ x) = ((atom\ i \sharp t \wedge j \sharp t) \vee (j \sharp x \wedge (j \sharp t \vee j = atom\ i)))$
    using *fresh-subst-ev-if*[*of j t i x*] *subst-v-e-def* **by** *metis*

  fix *a*::*x* **and** *tm*::*e* **and** *x*::*v*
  show $atom\ a \sharp tm \Longrightarrow subst\text{-}v\ tm\ a\ x = tm$
    using *forget-subst-ev subst-v-e-def* **by** *simp*

  fix *a*::*x* **and** *tm*::*e*
  show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-ev-id subst-v-e-def* **by** *simp*

  fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*e*
  show $p \cdot subst\text{-}v\ t1\ x1\ v = subst\text{-}v\ (p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$
    using *subst-ev-commute subst-v-e-def* **by** *simp*

  fix *x*::*x* **and** *c*::*e* **and** *z*::*x*
  show $atom\ x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c) = c[z{::=}[x]^v]_v$
    using *subst-v-e-def* **by** *simp*

  fix *x*::*x* **and** *c*::*e* **and** *z*::*x*
  show $atom\ x \sharp c \Longrightarrow c[z{::=}[x]^v]_v[x{::=}v]_v = c[z{::=}v]_v$
    using *subst-v-e-def* **by** *simp*
qed
end

lemma *subst-ev-commute-full*:
  fixes *e*::*e* **and** *w*::*v* **and** *v*::*v*
  assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** $x \neq z$
  shows $subst\text{-}ev\ (e[z{::=}w]_{ev})\ x\ v = subst\text{-}ev\ (e[x{::=}v]_{ev})\ z\ w$

using *assms* **by**(*nominal-induct e rule*: *e.strong-induct*,*simp*+)

lemma *subst-ev-v-flip1* [*simp*]:
fixes *e*::*e*
assumes *atom z1* ♯ (*z*,*e*) **and** *atom z1'* ♯ (*z*,*e*)
shows(*z1* ↔ *z1'*) • *e*[*z*::=*v*]$_{ev}$ = *e*[*z*::= ((*z1* ↔ *z1'*) • *v*)]$_{ev}$
using *assms* **proof**(*nominal-induct e rule*:*e.strong-induct*)
qed (*simp add*: *flip-def fresh-Pair swap-fresh-fresh*)+

3.4 Expressions in Constraints

nominal-function *subst-cev* :: *ce* ⇒ *x* ⇒ *v* ⇒ *ce* **where**
subst-cev ((*CE-val v'*)) *x v* = ((*CE-val* (*subst-vv v' x v*)))
| *subst-cev* ((*CE-op opp v1 v2*)) *x v* = ((*CE-op opp* (*subst-cev v1 x v*) (*subst-cev v2 x v*)))
| *subst-cev* ((*CE-fst v'*)) *x v* = *CE-fst* (*subst-cev v' x v*)
| *subst-cev* ((*CE-snd v'*)) *x v* = *CE-snd* (*subst-cev v' x v*)
| *subst-cev* ((*CE-len v'*)) *x v* = *CE-len* (*subst-cev v' x v*)
| *subst-cev* (*CE-concat v1 v2*) *x v* = *CE-concat* (*subst-cev v1 x v*) (*subst-cev v2 x v*)
apply (*simp add*: *eqvt-def subst-cev-graph-aux-def*,*auto*)
by (*meson ce.strong-exhaust*)

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-cev-abbrev :: *ce* ⇒ *x* ⇒ *v* ⇒ *ce* (‹-[-::=-]$_{cev}$› [*1000*,*50*,*50*] *500*)
where
e[*x*::=*v'*]$_{cev}$ ≡ *subst-cev e x v'*

lemma *size-subst-cev* [*simp*]: *size* (*subst-cev A i x*) = *size A*
by (*nominal-induct A avoiding*: *i x rule*: *ce.strong-induct*,*auto*)

lemma *forget-subst-cev* [*simp*]: *atom a* ♯ *A* ⟹ *subst-cev A a x* = *A*
by (*nominal-induct A avoiding*: *a x rule*: *ce.strong-induct*, *auto simp*: *fresh-at-base*)

lemma *subst-cev-id* [*simp*]: *subst-cev A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: *ce.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-cev-if* [*simp*]:
j ♯ (*subst-cev A i x*) = ((*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*)))
proof(*nominal-induct A avoiding*: *i x rule*: *ce.strong-induct*)
case (*CE-op opp v1 v2*)
then show *?case* **using** *fresh-subst-vv-if subst-ev.simps e.supp pure-fresh opp.fresh fresh-e-opp*
using *fresh-opp-all* **by** *auto*
qed(*auto*)+

lemma *subst-cev-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-cev* (*subst-cev A i t*) *j u*) = *subst-cev A i* (*subst-vv t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *ce.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-cev-var-flip*[*simp*]:
fixes *e*::*ce* **and** *y*::*x* **and** *x*::*x*

assumes *atom y* ♯ *e*
shows $(y \leftrightarrow x) \cdot e = e\ [x{::=}V\text{-}var\ y]_{cev}$
using *assms* **proof**(*nominal-induct e rule*:*ce.strong-induct*)
case (*CE-val v*)
then show *?case* **using** *subst-vv-var-flip* **by** *auto*
next
case (*CE-op opp v1 v2*)
hence *yf*: *atom y* ♯ *v1* ∧ *atom y* ♯ *v2* **using** *ce.fresh* **by** *blast*
have $(y \leftrightarrow x) \cdot (CE\text{-}op\ opp\ v1\ v2\ ) = CE\text{-}op\ ((y \leftrightarrow x) \cdot opp)\ (\ (y \leftrightarrow x) \cdot v1\ )\ (\ (y \leftrightarrow x) \cdot v2)$
using *opp.perm-simps ce.perm-simps permute-pure ce.fresh opp.strong-exhaust* **by** *presburger*
also have ... $= CE\text{-}op\ ((y \leftrightarrow x) \cdot opp)\ (v1[x{::=}V\text{-}var\ y]_{cev})\ (v2\ [x{::=}V\text{-}var\ y]_{cev})$ **using** *yf*
by (*simp add*: *CE-op.hyps*(*1*) *CE-op.hyps*(*2*))
finally show *?case* **using** *subst-cev.simps opp.perm-simps opp.strong-exhaust*
by (*metis* (*full-types*))
qed((*auto simp add*: *permute-pure subst-vv-var-flip*)+)

lemma *subst-cev-flip*:
fixes *e*::*ce* **and** *ea*::*ce* **and** *c*::*x*
assumes *atom c* ♯ (*e*, *ea*) **and** *atom c* ♯ (*x*, *xa*, *e*, *ea*) **and** $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
shows $e[x{::=}v']_{cev} = ea[xa{::=}v']_{cev}$
proof −
have $e[x{::=}v']_{cev} = (e[x{::=}V\text{-}var\ c]_{cev})[c{::=}v']_{cev}$ **using** *subst-ev-commute assms* **by** *simp*
also have ... $= ((c \leftrightarrow x) \cdot e)[c{::=}v']_{cev}$ **using** *subst-ev-var-flip assms* **by** *simp*
also have ... $= ((c \leftrightarrow xa) \cdot ea)[c{::=}v']_{cev}$ **using** *assms flip-commute* **by** *metis*
also have ... $= ea[xa{::=}v']_{cev}$ **using** *subst-ev-var-flip assms* **by** *simp*
finally show *?thesis* **by** *auto*
qed

lemma *subst-cev-var*[*simp*]:
fixes *z*::*x* **and** *x*::*x*
shows $[[x]^v]^{ce}\ [x{::=}[z]^v]_{cev} = [[z]^v]^{ce}$
by *auto*

instantiation *ce* :: *has-subst-v*
begin

definition
subst-v = *subst-cev*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*ce*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-cev-if*[*of j t i x*] *subst-v-ce-def* **by** *metis*

fix *a*::*x* **and** *tm*::*ce* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-cev subst-v-ce-def* **by** *simp*

fix *a*::*x* **and** *tm*::*ce*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-cev-id subst-v-ce-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*ce*

show *p* • *subst-v t1 x1 v* = *subst-v* (*p* • *t1*) (*p* • *x1*) (*p* • *v*)
using *subst-cev-commute subst-v-ce-def* **by** *simp*

fix *x*::*x* **and** *c*::*ce* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) • *c*) = *c* $[z::=V\text{-}var\ x]_v$
using *subst-v-ce-def* **by** *simp*

fix *x*::*x* **and** *c*::*ce* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ *c* $[z::=V\text{-}var\ x]_v[x::=v]_v = c[z::=v]_v$
using *subst-v-ce-def* **by** *simp*
qed

end

lemma *subst-cev-commute-full*:
fixes *e*::*ce* **and** *w*::*v* **and** *v*::*v*
assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** *x* ≠ *z*
shows *subst-cev* ($e[z::=w]_{cev}$) *x v* = *subst-cev* ($e[x::=v]_{cev}$) *z w*
using *assms* **by**(*nominal-induct e rule*: *ce.strong-induct*,*simp*+)

lemma *subst-cev-v-flip1*[*simp*]:
fixes *e*::*ce*
assumes *atom z1* ♯ (*z*,*e*) **and** *atom z1'* ♯ (*z*,*e*)
shows(*z1* ↔ *z1'*) • $e[z::=v]_{cev}$ = $e[z::= ((z1 \leftrightarrow z1') \cdot v)]_{cev}$
using *assms* **apply**(*nominal-induct e rule*:*ce.strong-induct*)
by (*simp add*: *flip-def fresh-Pair swap-fresh-fresh*)+

3.5 Constraints

nominal-function *subst-cv* :: *c* ⇒ *x* ⇒ *v* ⇒ *c* **where**
subst-cv (*C-true*) *x v* = *C-true*
| *subst-cv* (*C-false*) *x v* = *C-false*
| *subst-cv* (*C-conj c1 c2*) *x v* = *C-conj* (*subst-cv c1 x v*) (*subst-cv c2 x v*)
| *subst-cv* (*C-disj c1 c2*) *x v* = *C-disj* (*subst-cv c1 x v*) (*subst-cv c2 x v*)
| *subst-cv* (*C-imp c1 c2*) *x v* = *C-imp* (*subst-cv c1 x v*) (*subst-cv c2 x v*)
| *subst-cv* (*e1* == *e2*) *x v* = ((*subst-cev e1 x v*) == (*subst-cev e2 x v*))
| *subst-cv* (*C-not c*) *x v* = *C-not* (*subst-cv c x v*)
apply (*simp add*: *eqvt-def subst-cv-graph-aux-def*,*auto*)
using *c.strong-exhaust* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-cv-abbrev :: *c* ⇒ *x* ⇒ *v* ⇒ *c* (‹$\text{-}[\text{-}::=\text{-}]_{cv}$› [*1000*,*50*,*50*] *1000*)
where
$c[x::=v']_{cv}$ ≡ *subst-cv c x v'*

lemma *size-subst-cv* [*simp*]: *size* (*subst-cv A i x*) = *size A*
by (*nominal-induct A avoiding*: *i x rule*: *c.strong-induct*,*auto*)

lemma *forget-subst-cv* [*simp*]: *atom a* ♯ *A* ⟹ *subst-cv A a x* = *A*
by (*nominal-induct A avoiding*: *a x rule*: *c.strong-induct*, *auto simp*: *fresh-at-base*)

lemma *subst-cv-id* [*simp*]: *subst-cv A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-cv-if* [*simp*]:
$j \sharp$ (*subst-cv A i x*) $\longleftrightarrow$ (*atom i* $\sharp$ *A* $\wedge$ *j* $\sharp$ *A*) $\vee$ (*j* $\sharp$ *x* $\wedge$ (*j* $\sharp$ *A* $\vee$ *j* = *atom i*))
by (*nominal-induct A avoiding*: *i x rule*: *c.strong-induct*, (*auto simp add*: *pure-fresh*)+)

lemma *subst-cv-commute* [*simp*]:
atom j $\sharp$ *A* $\Longrightarrow$ (*subst-cv* (*subst-cv A i t*) *j u*) = *subst-cv A i* (*subst-vv t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *let-s-size* [*simp*]: *size s* $\leq$ *size* (*AS-let x e s*)
apply (*nominal-induct s avoiding*: *e x rule*: *s-branch-s-branch-list.strong-induct*(*1*))
apply *auto*
done

lemma *subst-cv-var-flip*[*simp*]:
fixes *c*::*c*
assumes *atom y* $\sharp$ *c*
shows $(y \leftrightarrow x) \cdot c = c[x{::=}V\text{-}var\ y]_{cv}$
using *assms* **by**(*nominal-induct c rule*:*c.strong-induct*,(*simp add*: *flip-subst-v subst-v-ce-def*)+)

instantiation *c* :: *has-subst-v*
begin

definition
subst-v = *subst-cv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*c*
show (*j* $\sharp$ *subst-v t i x*) = ((*atom i* $\sharp$ *t* $\wedge$ *j* $\sharp$ *t*) $\vee$ (*j* $\sharp$ *x* $\wedge$ (*j* $\sharp$ *t* $\vee$ *j* = *atom i*)))
using *fresh-subst-cv-if*[*of j t i x*] *subst-v-c-def* **by** *metis*

fix *a*::*x* **and** *tm*::*c* **and** *x*::*v*
show *atom a* $\sharp$ *tm* $\Longrightarrow$ *subst-v tm a x* = *tm*
using *forget-subst-cv subst-v-c-def* **by** *simp*

fix *a*::*x* **and** *tm*::*c*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-cv-id subst-v-c-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*c*
show *p* $\cdot$ *subst-v t1 x1 v* = *subst-v* (*p* $\cdot$ *t1*) (*p* $\cdot$ *x1*) (*p* $\cdot$ *v*)
using *subst-cv-commute subst-v-c-def* **by** *simp*

fix *x*::*x* **and** *c*::*c* **and** *z*::*x*
show *atom x* $\sharp$ *c* $\Longrightarrow$ $((x \leftrightarrow z) \cdot c) = c[z{::=}[x]^v]_v$
using *subst-cv-var-flip subst-v-c-def* **by** *simp*

fix *x*::*x* **and** *c*::*c* **and** *z*::*x*
show *atom x* $\sharp$ *c* $\Longrightarrow$ $c[z{::=}[x]^v]_v[x{::=}v]_v = c[z{::=}v]_v$
using *subst-cv-var-flip subst-v-c-def* **by** *simp*

qed

end

lemma *subst-cv-var-flip1*[*simp*]:
  fixes *c*::*c*
  assumes *atom y* ♯ *c*
  shows (*x* ↔ *y*) · *c* = *c*[*x*::=*V-var y*]$_{cv}$
  using *subst-cv-var-flip flip-commute*
  by (*metis assms*)

lemma *subst-cv-v-flip3*[*simp*]:
  fixes *c*::*c*
  assumes *atom z1* ♯ *c* **and** *atom z1'* ♯ *c*
  shows(*z1* ↔ *z1'*) · *c*[*z*::=[*z1*]^v]$_{cv}$ = *c*[*z*::=[*z1'*]^v]$_{cv}$
proof −
  consider *z1'* = *z* | *z1* = *z* | *atom z1* ♯ *z* ∧ *atom z1'* ♯ *z* **by** *force*
  then show *?thesis* **proof**(*cases*)
    case *1*
    then show *?thesis* **using** *1 assms* **by** *auto*
  next
    case *2*
    then show *?thesis* **using** *2 assms* **by** *auto*
  next
    case *3*
    then show *?thesis* **using** *assms* **by** *auto*
  qed
qed

lemma *subst-cv-v-flip*[*simp*]:
  fixes *c*::*c*
  assumes *atom x* ♯ *c*
  shows ((*x* ↔ *z*) · *c*)[*x*::=*v*]$_{cv}$ = *c* [*z*::=*v*]$_{cv}$
  using *assms subst-v-c-def* **by** *auto*

lemma *subst-cv-commute-full*:
  fixes *c*::*c*
  assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** *x*≠*z*
  shows (*c*[*z*::=*w*]$_{cv}$)[*x*::=*v*]$_{cv}$ = (*c*[*x*::=*v*]$_{cv}$)[*z*::=*w*]$_{cv}$
  using *assms* **proof**(*nominal-induct c rule*: *c.strong-induct*)
  case (*C-eq e1 e2*)
  then show *?case* **using** *subst-cev-commute-full* **by** *simp*
qed(*force+*)

lemma *subst-cv-eq*[*simp*]:
  assumes *atom z1* ♯ *e1*
  shows (*CE-val* (*V-var z1*) == *e1*)[*z1*::=[*x*]^v]$_{cv}$ = (*CE-val* (*V-var x*) == *e1*) (**is** *?A* = *?B*)
proof −
  have *?A* = (((*CE-val* (*V-var z1*))[*z1*::=[*x*]^v]$_{cev}$) == *e1*) **using** *subst-cv.simps assms* **by** *simp*
  thus *?thesis* **by** *simp*
qed

3.6 Variable Context

The idea of this substitution is to remove x from the context. We really want to add the condition that x is fresh in v but this causes problems with proofs.

nominal-function *subst-gv* :: Γ ⇒ *x* ⇒ *v* ⇒ Γ **where**
  subst-gv GNil x v = *GNil*
| *subst-gv* ((*y*,*b*,*c*) $\#_\Gamma$ Γ) *x v* = (*if x* = *y then* Γ *else* ((*y*,*b*,*c*[*x*::=*v*]$_{cv}$)$\#_\Gamma$ (*subst-gv* Γ *x v*)))
proof(*goal-cases*)
  case *1*
  then show *?case* **by**(*simp add*: *eqvt-def subst-gv-graph-aux-def*)
next
  case (*3 P x*)
  then show *?case* **by** (*metis neq-GNil-conv prod-cases3*)
qed(*fast*+)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
  subst-gv-abbrev :: Γ ⇒ *x* ⇒ *v* ⇒ Γ (‹-[-::=-]$_{\Gamma v}$› [*1000*,*50*,*50*] *1000*)
  where
    g[*x*::=*v*]$_{\Gamma v}$ ≡ *subst-gv g x v*

lemma *size-subst-gv* [*simp*]: *size* (*subst-gv G i x*) ≤ *size G*
  by (*induct G*,*auto*)

lemma *forget-subst-gv* [*simp*]: *atom a* ♯ *G* ⟹ *subst-gv G a x* = *G*
  apply (*induct G* ,*auto*)
  using *fresh-GCons fresh-PairD*(*1*) *not-self-fresh* **apply** *blast*
   apply (*simp add*: *fresh-GCons*)+
  done

lemma *fresh-subst-gv*: *atom a* ♯ *G* ⟹ *atom a* ♯ *v* ⟹ *atom a* ♯ *subst-gv G x v*
proof(*induct G*)
  case *GNil*
  then show *?case* **by** *auto*
next
  case (*GCons xbc G*)
  obtain *x'* **and** *b'* **and** *c'* **where** *xbc*: *xbc* = (*x'*,*b'*,*c'*) **using** *prod-cases3* **by** *blast*
  show *?case* **proof**(*cases x*=*x'*)
    case *True*
    have *atom a* ♯ *G* **using** *GCons fresh-GCons* **by** *blast*
    thus *?thesis* **using** *subst-gv.simps*(*2*)[*of x' b' c' G*] *GCons xbc True* **by** *presburger*
  next
    case *False*
    then show *?thesis* **using** *subst-gv.simps*(*2*)[*of x' b' c' G*] *GCons xbc False fresh-GCons* **by** *simp*
  qed
qed

lemma *subst-gv-flip*:
  fixes *x*::*x* **and** *xa*::*x* **and** *z*::*x* **and** *c*::*c* **and** *b*::*b* **and** Γ::Γ
  assumes *atom xa* ♯ ((*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ) **and** *atom xa* ♯ Γ **and** *atom x* ♯ Γ **and** *atom x* ♯ (*z*, *c*) **and** *atom xa* ♯ (*z*, *c*)
  shows (*x* ↔ *xa*) • ((*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ) = (*xa*, *b*, *c*[*z*::=*V-var xa*]$_{cv}$) $\#_\Gamma$ Γ

proof −
have $(x \leftrightarrow xa) \cdot ((x, b, c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma) = (((x \leftrightarrow xa) \cdot x, b, (x \leftrightarrow xa) \cdot c[z::=[x]^v]_{cv}) \#_\Gamma ((x \leftrightarrow xa) \cdot \Gamma))$
using *subst Cons-eqvt flip-fresh-fresh* **using** *G-cons-flip* **by** *simp*
also have ... $= ((xa, b, (x \leftrightarrow xa) \cdot c[z::=[x]^v]_{cv}) \#_\Gamma ((x \leftrightarrow xa) \cdot \Gamma))$ **using** *assms* **by** *fastforce*
also have ... $= ((xa, b, c[z::=V\text{-}var\ xa]_{cv}) \#_\Gamma ((x \leftrightarrow xa) \cdot \Gamma))$ **using** *assms subst-cv-var-flip* **by** *fastforce*
also have ... $= ((xa, b, c[z::=V\text{-}var\ xa]_{cv}) \#_\Gamma \Gamma)$ **using** *assms flip-fresh-fresh* **by** *blast*
finally show *?thesis* **by** *simp*
qed

3.7 Types

nominal-function *subst-tv* :: $\tau \Rightarrow x \Rightarrow v \Rightarrow \tau$ **where**
atom $z \sharp (x,v) \Longrightarrow$ *subst-tv* $\{\!\mid z : b \mid c \mid\!\}\ x\ v = \{\!\mid z : b \mid c[x::=v]_{cv} \mid\!\}$
apply (*simp add*: *eqvt-def subst-tv-graph-aux-def*)
apply *auto*
subgoal for *P a aa b*
apply(*rule-tac* y=a **and** c=(aa,b) **in** *τ.strong-exhaust*)
by (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
proof −
fix $z :: x$ **and** $c :: c$ **and** $za :: x$ **and** $xa :: x$ **and** $va :: v$ **and** $ca :: c$ **and** $cb :: x$
assume *a1*: *atom* $za \sharp va$ **and** *a2*: *atom* $z \sharp va$ **and** *a3*: $\forall cb.$ *atom* $cb \sharp c \wedge$ *atom* $cb \sharp ca \longrightarrow cb \neq z \wedge cb \neq za \longrightarrow c[z::=V\text{-}var\ cb]_{cv} = ca[za::=V\text{-}var\ cb]_{cv}$
assume *a4*: *atom* $cb \sharp c$ **and** *a5*: *atom* $cb \sharp ca$ **and** *a6*: $cb \neq z$ **and** *a7*: $cb \neq za$ **and** *atom* $cb \sharp va$ **and** *a8*: $za \neq xa$ **and** *a9*: $z \neq xa$
assume *a10*:$cb \neq xa$
note *assms* = *a10 a9 a8 a7 a6 a5 a4 a3 a2 a1*

have $c[z::=V\text{-}var\ cb]_{cv} = ca[za::=V\text{-}var\ cb]_{cv}$ **using** *assms* **by** *auto*
hence $c[z::=V\text{-}var\ cb]_{cv}[xa::=va]_{cv} = ca[za::=V\text{-}var\ cb]_{cv}[xa::=va]_{cv}$ **by** *simp*
moreover have $c[z::=V\text{-}var\ cb]_{cv}[xa::=va]_{cv} = c[xa::=va]_{cv}[z::=V\text{-}var\ cb]_{cv}$ **using** *subst-cv-commute-full*[*of z va xa V-var cb*] *assms fresh-def v.supp* **by** *fastforce*
moreover have $ca[za::=V\text{-}var\ cb]_{cv}[xa::=va]_{cv} = ca[xa::=va]_{cv}[za::=V\text{-}var\ cb]_{cv}$
using *subst-cv-commute-full*[*of za va xa V-var cb*] *assms fresh-def v.supp* **by** *fastforce*

ultimately show $c[xa::=va]_{cv}[z::=V\text{-}var\ cb]_{cv} = ca[xa::=va]_{cv}[za::=V\text{-}var\ cb]_{cv}$ **by** *simp*
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-tv-abbrev :: $\tau \Rightarrow x \Rightarrow v \Rightarrow \tau$ (‹-[-::=-]$_{\tau v}$› [*1000*,*50*,*50*] *1000*)
where
$t[x::=v]_{\tau v} \equiv$ *subst-tv t x v*

lemma *size-subst-tv* [*simp*]: *size* (*subst-tv A i x*) = *size A*
proof (*nominal-induct A avoiding*: *i x rule*: *τ.strong-induct*)
case (*T-refined-type* x' b' c')
then show *?case* **by** *auto*
qed

lemma *forget-subst-tv* [*simp*]: *atom a* ♯ *A* ⟹ *subst-tv A a x* = *A*
apply (*nominal-induct A avoiding*: *a x rule*: *τ.strong-induct*)
apply(*auto simp*: *fresh-at-base*)
done

lemma *subst-tv-id* [*simp*]: *subst-tv A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: *τ.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-tv-if* [*simp*]:
j ♯ (*subst-tv A i x*) ⟷ (*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*))
apply (*nominal-induct A avoiding*: *i x rule*: *τ.strong-induct*)
using *fresh-def supp-b-empty x-fresh-b* **by** *auto*

lemma *subst-tv-commute* [*simp*]:
atom y ♯ *τ* ⟹ $(\tau[x::= t]_{\tau v})[y::=v]_{\tau v} = \tau[x::= t[y::=v]_{vv}]_{\tau v}$
by (*nominal-induct τ avoiding*: *x y t v rule*: *τ.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-tv-var-flip* [*simp*]:
fixes *x*::*x* **and** *xa*::*x* **and** *τ*::*τ*
assumes *atom xa* ♯ *τ*
shows (*x* ↔ *xa*) · *τ* = $\tau[x::=V\text{-}var\ xa]_{\tau v}$
proof −
obtain *z*::*x* **and** *b* **and** *c* **where** *zbc*: *atom z* ♯ (*x*,*xa*, *V-var xa*) ∧ *τ* = ⦃ *z* : *b* | *c* ⦄
using *obtain-fresh-z* **by** (*metis prod.inject subst-tv.cases*)
hence *atom xa* ∉ *supp c* − { *atom z* } **using** *τ.supp*[*of z b c*] *fresh-def supp-b-empty assms*
by *auto*
moreover have *xa* ≠ *z* **using** *zbc fresh-prod3* **by** *force*
ultimately have *xaf*: *atom xa* ♯ *c* **using** *fresh-def* **by** *auto*
have (*x* ↔ *xa*) · *τ* = ⦃ *z* : *b* | (*x* ↔ *xa*) · *c* ⦄
by (*metis τ.perm-simps empty-iff flip-at-base-simps*(*3*) *flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*) *fresh-def not-self-fresh supp-b-empty v.fresh*(*2*) *zbc*)
also have ... = ⦃ *z* : *b* | $c[x::=V\text{-}var\ xa]_{cv}$ ⦄ **using** *subst-cv-v-flip xaf*
by (*metis permute-flip-cancel permute-flip-cancel2 subst-cv-var-flip*)
finally show *?thesis* **using** *subst-tv.simps zbc*
using *fresh-PairD*(*1*) *not-self-fresh* **by** *force*
qed

instantiation *τ* :: *has-subst-v*
begin

definition
subst-v = *subst-tv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*τ*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))

proof(*nominal-induct t avoiding*: *i x rule*:*τ.strong-induct*)
case (*T-refined-type z b c*)
hence *j* ♯ ⦃ *z* : *b* | *c* ⦄$[i::=x]_v$ = *j* ♯ ⦃ *z* : *b* | $c[i::=x]_{cv}$ ⦄ **using** *subst-tv.simps subst-v-τ-def fresh-Pair* **by** *simp*

```
    also have ... = (atom i ♯ ⦃ z : b | c ⦄ ∧ j ♯ ⦃ z : b | c ⦄ ∨ j ♯ x ∧ (j ♯ ⦃ z : b | c ⦄ ∨ j = atom i))
      unfolding τ.fresh using subst-v-c-def fresh-subst-v-if
      using T-refined-type.hyps(1) T-refined-type.hyps(2) x-fresh-b by auto
    finally show ?case by auto
  qed

  fix a::x and tm::τ and x::v
  show atom a ♯ tm ⟹ subst-v tm a x = tm
    apply(nominal-induct tm avoiding: a x rule:τ.strong-induct)
    using subst-v-c-def forget-subst-v subst-tv.simps subst-v-τ-def fresh-Pair by simp

  fix a::x and tm::τ
  show subst-v tm a (V-var a) = tm
    apply(nominal-induct tm avoiding: a rule:τ.strong-induct)
    using subst-v-c-def forget-subst-v subst-tv.simps subst-v-τ-def fresh-Pair by simp

  fix p::perm and x1::x and v::v and t1::τ
  show p · subst-v t1 x1 v = subst-v (p · t1) (p · x1) (p · v)
    apply(nominal-induct tm avoiding: a x rule:τ.strong-induct)
    using subst-v-c-def forget-subst-v subst-tv.simps subst-v-τ-def fresh-Pair by simp

  fix x::x and c::τ and z::x
  show atom x ♯ c ⟹ ((x ↔ z) · c) = c[z::=[x]^v]_v
    apply(nominal-induct c avoiding: z x rule:τ.strong-induct)
    using subst-v-c-def flip-subst-v subst-tv.simps subst-v-τ-def fresh-Pair by auto

  fix x::x and c::τ and z::x
  show  atom x ♯ c ⟹ c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v
    apply(nominal-induct c avoiding:  x v z rule:τ.strong-induct)
    using subst-v-c-def  subst-tv.simps subst-v-τ-def fresh-Pair
    by (metis flip-commute subst-tv-commute subst-tv-var-flip subst-v-τ-def subst-vv.simps(2))
qed

end

lemma subst-tv-commute-full:
  fixes c::τ
  assumes atom z ♯ v and atom x ♯ w and x≠z
  shows (c[z::=w]_τv)[x::=v]_τv = (c[x::=v]_τv)[z::=w]_τv
  using assms proof(nominal-induct c avoiding: x v z w rule: τ.strong-induct)
  case (T-refined-type x1a x2a x3a)
  then show ?case using subst-cv-commute-full by simp
qed

lemma type-eq-subst-eq:
  fixes v::v and c1::c
  assumes ⦃ z1 : b1 | c1 ⦄ = ⦃ z2 : b2 | c2 ⦄
  shows c1[z1::=v]_cv = c2[z2::=v]_cv
  using subst-v-flip-eq-two[of z1 c1 z2 c2 v] τ.eq-iff assms subst-v-c-def by simp
```

Extract constraint from a type. We cannot just project out the constraint as this would mean alpha-equivalent types give different answers

nominal-function *c-of* :: $\tau \Rightarrow x \Rightarrow c$ **where**
atom z ♯ *x* ⟹ *c-of* (*T-refined-type z b c*) *x* = *c*[*z*::=[*x*]^v]$_{cv}$
proof(*goal-cases*)
case *1*
then show *?case* **using** *eqvt-def c-of-graph-aux-def* **by** *force*
next
case (*2 x y*)
then show *?case* **using** *eqvt-def c-of-graph-aux-def* **by** *force*
next
case (*3 P x*)
then obtain *x1*::τ **and** *x2*::*x* **where** ∗:*x* = (*x1*,*x2*) **by** *force*
obtain *z′* **and** *b′* **and** *c′* **where** *x1* = ⦃ *z′* : *b′* | *c′* ⦄ ∧ *atom z′* ♯ *x2* **using** *obtain-fresh-z* **by** *metis*
then show *?case* **using** *3* ∗ **by** *auto*
next
case (*4 z1 x1 b1 c1 z2 x2 b2 c2*)
then show *?case* **using** *subst-v-flip-eq-two τ.eq-iff* **by** (*metis prod.inject type-eq-subst-eq*)
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *c-of-eq*:
shows *c-of* ⦃ *x* : *b* | *c* ⦄ *x* = *c*
proof(*nominal-induct* ⦃ *x* : *b* | *c* ⦄ *avoiding*: *x rule*: *τ.strong-induct*)
case (*T-refined-type x′ c′*)
moreover hence *c-of* ⦃ *x′* : *b* | *c′* ⦄ *x* = *c′*[*x′*::=*V-var x*]$_{cv}$ **using** *c-of.simps* **by** *auto*
moreover have ⦃ *x′* : *b* | *c′* ⦄ = ⦃ *x* : *b* | *c* ⦄ **using** *T-refined-type τ.eq-iff* **by** *metis*
moreover have *c′*[*x′*::=*V-var x*]$_{cv}$ = *c* **using** *T-refined-type Abs1-eq-iff flip-subst-v subst-v-c-def*
by (*metis subst-cv-id*)
ultimately show *?case* **by** *auto*
qed

lemma *obtain-fresh-z-c-of*:
fixes *t*::′*b*::*fs*
obtains *z* **where** *atom z* ♯ *t* ∧ τ = ⦃ *z* : *b-of* τ | *c-of* τ *z* ⦄
proof −
obtain *z* **and** *c* **where** *atom z* ♯ *t* ∧ τ = ⦃ *z* : *b-of* τ | *c* ⦄ **using** *obtain-fresh-z2* **by** *metis*
moreover hence *c* = *c-of* τ *z* **using** *c-of.simps* **using** *c-of-eq* **by** *metis*
ultimately show *?thesis*
using *that* **by** *auto*
qed

lemma *c-of-fresh*:
fixes *x*::*x*
assumes *atom x* ♯ (*t*,*z*)
shows *atom x* ♯ *c-of t z*
proof −
obtain *z′* **and** *c′* **where** *z*:*t* = ⦃ *z′* : *b-of t* | *c′* ⦄ ∧ *atom z′* ♯ (*x*,*z*) **using** *obtain-fresh-z-c-of* **by** *metis*
hence ∗:*c-of t z* = *c′*[*z′*::=*V-var z*]$_{cv}$ **using** *c-of.simps fresh-Pair* **by** *metis*
have (*atom x* ♯ *c′* ∨ *atom x* ∈ *set* [*atom z′*]) ∧ *atom x* ♯ *b-of t* **using** *τ.fresh assms z fresh-Pair* **by** *metis*
hence *atom x* ♯ *c′* **using** *fresh-Pair z fresh-at-base*(*2*) **by** *fastforce*
moreover have *atom x* ♯ *V-var z* **using** *assms fresh-Pair v.fresh* **by** *metis*

ultimately show *?thesis* **using** *assms fresh-subst-v-if*[*of atom x c′ z′ V-var z*] *subst-v-c-def* ∗ **by** *metis*
qed

lemma *c-of-switch*:
fixes *z*::*x*
assumes *atom z* ♯ *t*
shows $(c\text{-}of\ t\ z)[z::=V\text{-}var\ x]_{cv} = c\text{-}of\ t\ x$
proof −
obtain *z′* **and** *c′* **where** *z*:*t* = ⦃ *z′* : *b-of t* | *c′* ⦄ ∧ *atom z′* ♯ (*x*,*z*) **using** *obtain-fresh-z-c-of* **by** *metis*
hence (*atom z* ♯ *c′* ∨ *atom z* ∈ *set* [*atom z′*]) ∧ *atom z* ♯ *b-of t* **using** *τ.fresh*[*of atom z z′ b-of t c′*] *assms* **by** *metis*
moreover have *atom z* ∉ *set* [*atom z′*] **using** *z fresh-Pair* **by** *force*
ultimately have ∗∗:*atom z* ♯ *c′* **using** *fresh-Pair z fresh-at-base*(*2*) **by** *metis*

have $(c\text{-}of\ t\ z)[z::=V\text{-}var\ x]_{cv} = c'[z'::=V\text{-}var\ z]_{cv}[z::=V\text{-}var\ x]_{cv}$ **using** *c-of.simps fresh-Pair z* **by** *metis*
also have ... $= c'[z'::=V\text{-}var\ x]_{cv}$ **using** *subst-v-simple-commute subst-v-c-def assms c-of.simps z* ∗∗ **by** *metis*
finally show *?thesis* **using** *c-of.simps*[*of z′ x b-of t c′*] *fresh-Pair z* **by** *metis*
qed

lemma *type-eq-subst-eq1*:
fixes *v*::*v* **and** *c1*::*c*
assumes ⦃ *z1* : *b1* | *c1* ⦄ = (⦃ *z2* : *b2* | *c2* ⦄) **and** *atom z1* ♯ *c2*
shows $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **and** *b1*=*b2* **and** *c1* = (*z1* ↔ *z2*) • *c2*
proof −
show $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **using** *type-eq-subst-eq assms* **by** *blast*
show *b1*=*b2* **using** *τ.eq-iff assms* **by** *blast*
have *z1* = *z2* ∧ *c1* = *c2* ∨ *z1* ≠ *z2* ∧ *c1* = (*z1* ↔ *z2*) • *c2* ∧ *atom z1* ♯ *c2*
using *τ.eq-iff Abs1-eq-iff*[*of z1 c1 z2 c2*] *assms* **by** *blast*
thus *c1* = (*z1* ↔ *z2*) • *c2* **by** *auto*
qed

lemma *type-eq-subst-eq2*:
fixes *v*::*v* **and** *c1*::*c*
assumes ⦃ *z1* : *b1* | *c1* ⦄ = (⦃ *z2* : *b2* | *c2* ⦄)
shows $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **and** *b1*=*b2* **and** [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
proof −
show $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **using** *type-eq-subst-eq assms* **by** *blast*
show *b1*=*b2* **using** *τ.eq-iff assms* **by** *blast*
show [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
using *τ.eq-iff assms* **by** *auto*
qed

lemma *type-eq-subst-eq3*:
fixes *v*::*v* **and** *c1*::*c*
assumes ⦃ *z1* : *b1* | *c1* ⦄ = (⦃ *z2* : *b2* | *c2* ⦄) **and** *atom z1* ♯ *c2*
shows $c1 = c2[z2::=V\text{-}var\ z1]_{cv}$ **and** *b1*=*b2*
using *type-eq-subst-eq1 assms subst-v-c-def*
by (*metis subst-cv-var-flip*)+

lemma *type-eq-flip*:
assumes *atom x* $\sharp$ *c*
shows $\{\!\mid z : b \mid c \mid\!\} = \{\!\mid x : b \mid (x \leftrightarrow z) \cdot c \mid\!\}$
using τ*.eq-iff Abs1-eq-iff assms*
by (*metis* (*no-types*, *lifting*) *flip-fresh-fresh*)

lemma *c-of-true*:
c-of $\{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}$ *x* = *C-true*
proof(*nominal-induct* $\{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}$ *avoiding*: *x rule*:τ*.strong-induct*)
case (*T-refined-type x1a x3a*)
hence $\{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\} = \{\!\mid x1a : B\text{-}bool \mid x3a \mid\!\}$ **using** τ*.eq-iff* **by** *metis*
then show *?case* **using** *subst-cv.simps c-of.simps T-refined-type*
type-eq-subst-eq3
by (*metis type-eq-subst-eq*)
qed

lemma *type-eq-subst*:
assumes *atom x* $\sharp$ *c*
shows $\{\!\mid z : b \mid c \mid\!\} = \{\!\mid x : b \mid c[z::=[x]^v]_{cv} \mid\!\}$
using τ*.eq-iff Abs1-eq-iff assms*
using *subst-cv-var-flip type-eq-flip* **by** *auto*

lemma *type-e-subst-fresh*:
fixes *x*::*x* **and** *z*::*x*
assumes *atom z* $\sharp$ (*x*,*v*) **and** *atom x* $\sharp$ *e*
shows $\{\!\mid z : b \mid CE\text{-}val\ (V\text{-}var\ z) == e \mid\!\}[x::=v]_{\tau v} = \{\!\mid z : b \mid CE\text{-}val\ (V\text{-}var\ z) == e \mid\!\}$
using *assms subst-tv.simps subst-cv.simps forget-subst-cev* **by** *simp*

lemma *type-v-subst-fresh*:
fixes *x*::*x* **and** *z*::*x*
assumes *atom z* $\sharp$ (*x*,*v*) **and** *atom x* $\sharp$ *v'*
shows $\{\!\mid z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v' \mid\!\}[x::=v]_{\tau v} = \{\!\mid z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v' \mid\!\}$
using *assms subst-tv.simps subst-cv.simps* **by** *simp*

lemma *subst-tbase-eq*:
b-of τ = *b-of* $\tau[x::=v]_{\tau v}$
proof −
obtain *z* **and** *b* **and** *c* **where** *zbc*: $\tau = \{\!\mid z{:}b|c \mid\!\} \wedge$ *atom z* $\sharp$ (*x*,*v*) **using** τ*.exhaust*
by (*metis prod.inject subst-tv.cases*)
hence *b-of* $\{\!\mid z{:}b|c \mid\!\}$ = *b-of* $\{\!\mid z{:}b|c \mid\!\}[x::=v]_{\tau v}$ **using** *subst-tv.simps* **by** *simp*
thus *?thesis* **using** *zbc* **by** *blast*
qed

lemma *subst-tv-if*:
assumes *atom z1* $\sharp$ (*x*,*v*) **and** *atom z'* $\sharp$ (*x*,*v*)
shows $\{\!\mid z1 : b \mid CE\text{-}val\ (v'[x::=v]_{vv}) == CE\text{-}val\ (V\text{-}lit\ l)\ \ IMP\ \ (c'[x::=v]_{cv})[z'::=[z1]^v]_{cv} \mid\!\} =$
$\{\!\mid z1 : b \mid CE\text{-}val\ v' == CE\text{-}val\ (V\text{-}lit\ l)\ \ IMP\ \ c'[z'::=[z1]^v]_{cv} \mid\!\}[x::=v]_{\tau v}$
using *subst-cv-commute-full*[*of z' v x V-var z1 c'*] *subst-tv.simps subst-vv.simps*(*1*) *subst-ev.simps subst-cv.simps assms*
by *simp*

lemma *subst-tv-tid*:
  assumes *atom za* ♯ (*x*,*v*)
  shows ⦃ *za* : *B-id tid* | *TRUE* ⦄ = ⦃ *za* : *B-id tid* | *TRUE* ⦄$[x::=v]_{\tau v}$
  using *assms subst-tv.simps subst-cv.simps* **by** *presburger*

lemma *b-of-subst*:
  b-of $(\tau[x::=v]_{\tau v})$ = *b-of* τ
proof –
  obtain *z b c* **where** ∗:τ = ⦃ *z* : *b* | *c* ⦄ ∧ *atom z* ♯ (*x*,*v*) **using** *obtain-fresh-z* **by** *metis*
  thus *?thesis* **using** *subst-tv.simps* ∗ **by** *auto*
qed

lemma *subst-tv-flip*:
  assumes $\tau'[x::=v]_{\tau v} = \tau$ **and** *atom x* ♯ (*v*,τ) **and** *atom x′* ♯ (*v*,τ)
  shows $((x' \leftrightarrow x) \cdot \tau')[x'::=v]_{\tau v} = \tau$
proof –
  have (*x′* ↔ *x*) · *v* = *v* ∧ (*x′* ↔ *x*) · τ = τ **using** *assms flip-fresh-fresh* **by** *auto*
  thus *?thesis* **using** *subst-tv.eqvt*[*of* (*x′* ↔ *x*) τ′ *x v*] *assms* **by** *auto*
qed

lemma *subst-cv-true*:
  ⦃ *z* : *B-id tid* | *TRUE* ⦄ = ⦃ *z* : *B-id tid* | *TRUE* ⦄$[x::=v]_{\tau v}$
proof –
  obtain *za*::*x* **where** *atom za* ♯ (*x*,*v*) **using** *obtain-fresh* **by** *auto*
  hence ⦃ *z* : *B-id tid* | *TRUE* ⦄ = ⦃ *za*: *B-id tid* | *TRUE* ⦄ **using** τ*.eq-iff Abs1-eq-iff* **by** *fastforce*
  moreover have ⦃ *za* : *B-id tid* | *TRUE* ⦄ = ⦃ *za* : *B-id tid* | *TRUE* ⦄$[x::=v]_{\tau v}$
    using *subst-cv.simps subst-tv.simps* **by** (*simp add*: ‹*atom za* ♯ (*x*, *v*)›)
  ultimately show *?thesis* **by** *argo*
qed

lemma *t-eq-supp*:
  assumes (⦃ *z* : *b* | *c* ⦄) = (⦃ *z1* : *b1* | *c1* ⦄)
  shows *supp c* – { *atom z* } = *supp c1* – { *atom z1* }
proof –
  have *supp c* – { *atom z* } ∪ *supp b* = *supp c1* – { *atom z1* } ∪ *supp b1* **using** τ*.supp assms*
    by (*metis list.set*(*1*) *list.simps*(*15*) *sup-bot.right-neutral supp-b-empty*)
  moreover have *supp b* = *supp b1* **using** *assms* τ*.eq-iff* **by** *simp*
  moreover have *atom z1* ∉ *supp b1* ∧ *atom z* ∉ *supp b* **using** *supp-b-empty* **by** *simp*
  ultimately show *?thesis*
    by (*metis* τ*.eq-iff* τ*.supp assms b.supp*(*1*) *list.set*(*1*) *list.set*(*2*) *sup-bot.right-neutral*)
qed

lemma *fresh-t-eq*:
  fixes *x*::*x*
  assumes (⦃ *z* : *b* | *c* ⦄) = (⦃ *zz* : *b* | *cc* ⦄) **and** *atom x* ♯ *c* **and** *x* ≠ *zz*
  shows *atom x* ♯ *cc*
proof –
  have *supp c* – { *atom z* } ∪ *supp b* = *supp cc* – { *atom zz* } ∪ *supp b* **using** τ*.supp assms*
    by (*metis list.set*(*1*) *list.simps*(*15*) *sup-bot.right-neutral supp-b-empty*)
  moreover have *atom x* ∉ *supp c* **using** *assms fresh-def* **by** *blast*
  ultimately have *atom x* ∉ *supp cc* – { *atom zz* } ∪ *supp b* **by** *force*

hence *atom x* ∉ *supp cc* **using** *assms* **by** *simp*
thus *?thesis* **using** *fresh-def* **by** *auto*
qed

3.8 Mutable Variable Context

nominal-function *subst-dv* :: Δ ⇒ *x* ⇒ *v* ⇒ Δ **where**
subst-dv DNil x v = *DNil*
| *subst-dv* ((*u*,*t*) $\#_{\Delta}$ Δ) *x v* = ((*u*,$t[x::=v]_{\tau v}$) $\#_{\Delta}$ (*subst-dv* Δ *x v*))
apply (*simp add*: *eqvt-def subst-dv-graph-aux-def*,*auto*)
using *delete-aux.elims* **by** (*metis* Δ.*exhaust surj-pair*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-dv-abbrev :: Δ ⇒ *x* ⇒ *v* ⇒ Δ (‹-[-::=-]$_{\Delta v}$› [*1000*,*50*,*50*] *1000*)
where
Δ$[x::=v]_{\Delta v}$ ≡ *subst-dv* Δ *x v*

nominal-function *dmap* :: (*u*∗τ ⇒ *u*∗τ) ⇒ Δ ⇒ Δ **where**
dmap f DNil = *DNil*
| *dmap f* ((*u*,*t*)$\#_{\Delta}$Δ) = (*f* (*u*,*t*) $\#_{\Delta}$ (*dmap f* Δ))
apply (*simp add*: *eqvt-def dmap-graph-aux-def*,*auto*)
using *delete-aux.elims* **by** (*metis* Δ.*exhaust surj-pair*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *subst-dv-iff*:
Δ$[x::=v]_{\Delta v}$ = *dmap* (λ(*u*,*t*). (*u*, $t[x::=v]_{\tau v}$)) Δ
by(*induct* Δ, *auto*)

lemma *size-subst-dv* [*simp*]: *size* (*subst-dv G i x*) ≤ *size G*
by (*induct G*,*auto*)

lemma *forget-subst-dv* [*simp*]: *atom a* ♯ *G* ⟹ *subst-dv G a x* = *G*
apply (*induct G* ,*auto*)
using *fresh-DCons fresh-PairD*(*1*) *not-self-fresh* **apply** *fastforce*
apply (*simp add*: *fresh-DCons*)+
done

lemma *subst-dv-member*:
assumes (*u*,τ) ∈ *setD* Δ
shows (*u*, $\tau[x::=v]_{\tau v}$) ∈ *setD* (Δ$[x::=v]_{\Delta v}$)
using *assms* **by**(*induct* Δ *rule*: Δ-*induct*,*auto*)

lemma *fresh-subst-dv*:
fixes *x*::*x*
assumes *atom xa* ♯ Δ **and** *atom xa* ♯ *v*
shows *atom xa* ♯Δ$[x::=v]_{\Delta v}$
using *assms* **proof**(*induct* Δ *rule*:Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u t* Δ)

then show *?case* **using** *subst-dv.simps subst-v-τ-def fresh-DCons fresh-Pair* **by** *simp*
qed

lemma *fresh-subst-dv-if*:
fixes $j::atom$ **and** $i::x$ **and** $x::v$ **and** $t::\Delta$
assumes $j \sharp t \wedge j \sharp x$
shows $(j \sharp \textit{subst-dv } t\ i\ x)$
using *assms* **proof**(*induct t rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-gv.simps fresh-GNil* **by** *auto*
next
case $(DCons\ u'\ t'\ D')$
then show *?case* **unfolding** *subst-dv.simps* **using** *fresh-DCons fresh-subst-tv-if fresh-Pair* **by** *metis*
qed

3.9 Statements

Using ideas from proofs at top of AFP/Launchbury/Substitution.thy. Subproofs borrowed from there; hence the apply style proofs.

nominal-function (*default case-sum* (λ*x. Inl undefined*) (*case-sum* (λ*x. Inl undefined*) (λ*x. Inr undefined*)))
subst-sv :: *s* ⇒ *x* ⇒ *v* ⇒ *s*
and *subst-branchv* :: *branch-s* ⇒ *x* ⇒ *v* ⇒ *branch-s*
and *subst-branchlv* :: *branch-list* ⇒ *x* ⇒ *v* ⇒ *branch-list* **where**
subst-sv ((*AS-val v'*)) *x v* = (*AS-val* (*subst-vv v' x v*))
| *atom y* ♯ (*x*,*v*) ⟹ *subst-sv* (*AS-let y e s*) *x v* = (*AS-let y* ($e[x::=v]_{ev}$) (*subst-sv s x v*))
| *atom y* ♯ (*x*,*v*) ⟹ *subst-sv* (*AS-let2 y t s1 s2*) *x v* = (*AS-let2 y* ($t[x::=v]_{\tau v}$) (*subst-sv s1 x v*) (*subst-sv s2 x v*))
| *subst-sv* (*AS-match v' cs*) *x v* = *AS-match* ($v'[x::=v]_{vv}$) (*subst-branchlv cs x v*)
| *subst-sv* (*AS-assign y v'*) *x v* = *AS-assign y* (*subst-vv v' x v*)
| *subst-sv* ((*AS-if v' s1 s2*)) *x v* = (*AS-if* (*subst-vv v' x v*) (*subst-sv s1 x v*) (*subst-sv s2 x v*))
| *atom u* ♯ (*x*,*v*) ⟹ *subst-sv* (*AS-var u τ v' s*) *x v* = *AS-var u* (*subst-tv τ x v*) (*subst-vv v' x v*) (*subst-sv s x v*)
| *subst-sv* (*AS-while s1 s2*) *x v* = *AS-while* (*subst-sv s1 x v*) (*subst-sv s2 x v*)
| *subst-sv* (*AS-seq s1 s2*) *x v* = *AS-seq* (*subst-sv s1 x v*) (*subst-sv s2 x v*)
| *subst-sv* (*AS-assert c s*) *x v* = *AS-assert* (*subst-cv c x v*) (*subst-sv s x v*)
| *atom x1* ♯ (*x*,*v*) ⟹ *subst-branchv* (*AS-branch dc x1 s1*) *x v* = *AS-branch dc x1* (*subst-sv s1 x v*)

| *subst-branchlv* (*AS-final cs*) *x v* = *AS-final* (*subst-branchv cs x v*)
| *subst-branchlv* (*AS-cons cs css*) *x v* = *AS-cons* (*subst-branchv cs x v*) (*subst-branchlv css x v*)
apply (*auto*,*simp add*: *eqvt-def subst-sv-subst-branchv-subst-branchlv-graph-aux-def*)
proof(*goal-cases*)

have *eqvt-at-proj*: ⋀ *s xa va* . *eqvt-at subst-sv-subst-branchv-subst-branchlv-sumC* (*Inl* (*s*, *xa*, *va*)) ⟹

eqvt-at (λ*a. projl* (*subst-sv-subst-branchv-subst-branchlv-sumC* (*Inl a*))) (*s*, *xa*, *va*)
apply(*simp add*: *eqvt-at-def*)
apply(*rule*)
apply(*subst Projl-permute*)
apply(*thin-tac* -)+
apply (*simp add*: *subst-sv-subst-branchv-subst-branchlv-sumC-def*)

```
    apply (simp add: THE-default-def)
    apply (case-tac Ex1 (subst-sv-subst-branchv-subst-branchlv-graph (Inl (s,xa,va))))
     apply simp
     apply(auto)[1]
     apply (erule-tac x=x in allE)
     apply simp
     apply(cases rule: subst-sv-subst-branchv-subst-branchlv-graph.cases)
                 apply(assumption)
                apply(rule-tac x=Sum-Type.projl x in exI,clarify,rule the1-equality,blast,simp (no-asm)
only: sum.sel)+
       apply blast +

   apply(simp)+
   done

  {
   case (1 P x')
   then show ?case proof(cases x')
     case (Inl a) thus P
     proof(cases a)
       case (fields aa bb cc)
       thus P using Inl 1 s-branch-s-branch-list.strong-exhaust fresh-star-insert by metis
     qed
   next
     case (Inr b) thus P
     proof(cases b)
       case (Inl a) thus P proof(cases a)
         case (fields aa bb cc)
          then show ?thesis  using Inr Inl 1 s-branch-s-branch-list.strong-exhaust fresh-star-insert by
metis
       qed
     next
       case Inr2: (Inr b) thus P proof(cases b)
         case (fields aa bb cc)
         then show ?thesis  using Inr Inr2 1 s-branch-s-branch-list.strong-exhaust fresh-star-insert by
metis
       qed
     qed
   qed
  next
   case (2 y s ya xa va sa c)
   thus ?case using eqvt-triple eqvt-at-proj by blast
  next
   case (3 y s2 ya xa va s1a s2a c)
   thus ?case using eqvt-triple eqvt-at-proj by blast
  next
   case (4 u xa va s ua sa c)
   moreover have atom u ♯ (xa, va) ∧ atom ua ♯ (xa, va)
     using fresh-Pair u-fresh-xv by auto
   ultimately show ?case using eqvt-triple[of u xa va ua s sa]  subst-sv-def eqvt-at-proj by metis
  next
   case (5 x1 s1 x1a xa va s1a c)
```

thus *?case* **using** *eqvt-triple eqvt-at-proj* **by** *blast*
}
qed
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-sv-abbrev :: *s* ⇒ *x* ⇒ *v* ⇒ *s* (‹-[-::=-]$_{sv}$› [*1000*,*50*,*50*] *1000*)
where
s[*x*::=*v*]$_{sv}$ ≡ *subst-sv s x v*

abbreviation
subst-branchv-abbrev :: *branch-s* ⇒ *x* ⇒ *v* ⇒ *branch-s* (‹-[-::=-]$_{sv}$› [*1000*,*50*,*50*] *1000*)
where
s[*x*::=*v*]$_{sv}$ ≡ *subst-branchv s x v*

lemma *size-subst-sv* [*simp*]: *size* (*subst-sv A i x*) = *size A* **and** *size* (*subst-branchv B i x*) = *size B* **and** *size* (*subst-branchlv C i x*) = *size C*
by(*nominal-induct A* **and** *B* **and** *C avoiding*: *i x rule*: *s-branch-s-branch-list.strong-induct*,*auto*)

lemma *forget-subst-sv* [*simp*]: **shows** *atom a* ♯ *A* ⟹ *subst-sv A a x* = *A* **and** *atom a* ♯ *B* ⟹ *subst-branchv B a x* = *B* **and** *atom a* ♯ *C* ⟹ *subst-branchlv C a x* = *C*
by (*nominal-induct A* **and** *B* **and** *C avoiding*: *a x rule*: *s-branch-s-branch-list.strong-induct*,*auto simp*: *fresh-at-base*)

lemma *subst-sv-id* [*simp*]: *subst-sv A a* (*V-var a*) = *A* **and** *subst-branchv B a* (*V-var a*) = *B* **and** *subst-branchlv C a* (*V-var a*) = *C*
proof(*nominal-induct A* **and** *B* **and** *C avoiding*: *a rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-let x option e s*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *fresh-Pair not-None-eq subst-ev-id subst-sv.simps*(*2*) *subst-sv.simps*(*3*) *subst-tv-id v.fresh*(*2*))
next
case (*AS-match v branch-s*)
then show *?case* **using** *fresh-Pair not-None-eq subst-ev-id subst-sv.simps subst-sv.simps subst-tv-id v.fresh subst-vv-id*
by *metis*
qed(*auto*)+

lemma *fresh-subst-sv-if-rl*:
shows
(*atom x* ♯ *s* ∧ *j* ♯ *s*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *s* ∨ *j* = *atom x*)) ⟹ *j* ♯ (*subst-sv s x v*) **and**
(*atom x* ♯ *cs* ∧ *j* ♯ *cs*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *cs* ∨ *j* = *atom x*)) ⟹ *j* ♯ (*subst-branchv cs x v*) **and**
(*atom x* ♯ *css* ∧ *j* ♯ *css*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *css* ∨ *j* = *atom x*)) ⟹ *j* ♯ (*subst-branchlv css x v*)
apply(*nominal-induct s* **and** *cs* **and** *css avoiding*: *v x rule*: *s-branch-s-branch-list.strong-induct*)
using *pure-fresh* **by** *force*+

lemma *fresh-subst-sv-if-lr*:
shows *j* ♯ (*subst-sv s x v*) ⟹ (*atom x* ♯ *s* ∧ *j* ♯ *s*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *s* ∨ *j* = *atom x*)) **and**
j ♯ (*subst-branchv cs x v*) ⟹ (*atom x* ♯ *cs* ∧ *j* ♯ *cs*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *cs* ∨ *j* = *atom x*)) **and**
j ♯ (*subst-branchlv css x v*) ⟹ (*atom x* ♯ *css* ∧ *j* ♯ *css*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *css* ∨ *j* = *atom x*))
proof(*nominal-induct s* **and** *cs* **and** *css avoiding*: *v x rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-branch list x s*)

then show *?case* **using** *s-branch-s-branch-list.fresh fresh-Pair list.distinct*(*1*) *list.set-cases pure-fresh set-ConsD subst-branchv.simps* **by** *metis*
next
case (*AS-let y e s'*)
thus *?case* **proof**(*cases atom x* ♯ (*AS-let y e s'*))
case *True*
hence *subst-sv* (*AS-let y e s'*) *x v* = (*AS-let y e s'*) **using** *forget-subst-sv* **by** *simp*
hence *j* ♯ (*AS-let y e s'*) **using** *AS-let* **by** *argo*
then show *?thesis* **using** *True* **by** *blast*
next
case *False*
have *subst-sv* (*AS-let y e s'*) *x v* = *AS-let y* ($e[x::=v]_{ev}$) ($s'[x::=v]_{sv}$) **using** *subst-sv.simps*(*2*) *AS-let* **by** *force*
hence ((j ♯ $s'[x::=v]_{sv}$ ∨ *j* ∈ *set* [*atom y*]) ∧ *j* ♯ *None* ∧ *j* ♯ $e[x::=v]_{ev}$) **using** *s-branch-s-branch-list.fresh AS-let*
by (*simp add*: *fresh-None*)
then show *?thesis* **using** *AS-let fresh-None fresh-subst-ev-if list.discI list.set-cases s-branch-s-branch-list.fresh set-ConsD*
by *metis*
qed
next
case (*AS-let2 y τ s1 s2*)
thus *?case* **proof**(*cases atom x* ♯ (*AS-let2 y τ s1 s2*))
case *True*
hence *subst-sv* (*AS-let2 y τ s1 s2*) *x v* = (*AS-let2 y τ s1 s2*) **using** *forget-subst-sv* **by** *simp*
hence *j* ♯ (*AS-let2 y τ s1 s2*) **using** *AS-let2* **by** *argo*
then show *?thesis* **using** *True* **by** *blast*
next
case *False*
have *subst-sv* (*AS-let2 y τ s1 s2*) *x v* = *AS-let2 y* ($\tau[x::=v]_{\tau v}$) ($s1[x::=v]_{sv}$) ($s2[x::=v]_{sv}$) **using** *subst-sv.simps AS-let2* **by** *force*
then show *?thesis* **using** *AS-let2*
fresh-subst-tv-if list.discI list.set-cases s-branch-s-branch-list.fresh(*4*) *set-ConsD* **by** *auto*
qed
qed(*auto*)+

lemma *fresh-subst-sv-if* [*simp*]:
fixes *x*::*x* **and** *v*::*v*
shows *j* ♯ (*subst-sv s x v*) ⟷ (*atom x* ♯ *s* ∧ *j* ♯ *s*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *s* ∨ *j* = *atom x*)) **and**
j ♯ (*subst-branchv cs x v*) ⟷ (*atom x* ♯ *cs* ∧ *j* ♯ *cs*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *cs* ∨ *j* = *atom x*))
using *fresh-subst-sv-if-lr fresh-subst-sv-if-rl* **by** *metis*+

lemma *subst-sv-commute* [*simp*]:
fixes *A*::*s* **and** *t*::*v* **and** *j*::*x* **and** *i*::*x*
shows *atom j* ♯ *A* ⟹ (*subst-sv* (*subst-sv A i t*) *j u*) = *subst-sv A i* (*subst-vv t j u*) **and**
atom j ♯ *B* ⟹ (*subst-branchv* (*subst-branchv B i t*) *j u*) = *subst-branchv B i* (*subst-vv t j u*) **and**
atom j ♯ *C* ⟹ (*subst-branchlv* (*subst-branchlv C i t*) *j u*) = *subst-branchlv C i* (*subst-vv t j u*)
apply(*nominal-induct A* **and** *B* **and** *C avoiding*: *i j t u rule*: *s-branch-s-branch-list.strong-induct*)
by(*auto simp*: *fresh-at-base*)

lemma *c-eq-perm*:
assumes ((*atom z*) ⇌ (*atom z'*)) · *c* = *c'* **and** *atom z'* ♯ *c*

shows ⦃ *z* : *b* | *c* ⦄ = ⦃ *z'* : *b* | *c'* ⦄
using *τ.eq-iff Abs1-eq-iff(3)*
by (*metis Nominal2-Base.swap-commute assms(1) assms(2) flip-def swap-fresh-fresh*)

lemma *subst-sv-flip*:
fixes *s::s* **and** *sa::s* **and** *v'::v*
assumes *atom c* ♯ (*s*, *sa*) **and** *atom c* ♯ (*v'*,*x*, *xa*, *s*, *sa*) *atom x* ♯ *v'* **and** *atom xa* ♯ *v'* **and** (*x* ↔ *c*) • *s* = (*xa* ↔ *c*) • *sa*
shows $s[x::=v']_{sv}$ = $sa[xa::=v']_{sv}$
proof −
have *atom x* ♯ ($s[x::=v']_{sv}$) **and** *xafr*: *atom xa* ♯ ($sa[xa::=v']_{sv}$)
and *atom c* ♯ ($s[x::=v']_{sv}$, $sa[xa::=v']_{sv}$) **using** *assms* **using** *fresh-subst-sv-if assms* **by**(*blast+ ,force*)

hence $s[x::=v']_{sv}$ = (*x* ↔ *c*) • ($s[x::=v']_{sv}$) **by** (*simp add*: *flip-fresh-fresh fresh-Pair*)
also have ... = ((*x* ↔ *c*) • *s*)[((*x* ↔ *c*) • *x*) ::= ((*x* ↔ *c*) • *v'*) $]_{sv}$ **using** *subst-sv-subst-branchv-subst-branchlv.eqvt* **by** *blast*
also have ... = ((*xa* ↔ *c*) • *sa*)[((*x* ↔ *c*) • *x*) ::= ((*x* ↔ *c*) • *v'*) $]_{sv}$ **using** *assms* **by** *presburger*
also have ... = ((*xa* ↔ *c*) • *sa*)[((*xa* ↔ *c*) • *xa*) ::= ((*xa* ↔ *c*) • *v'*) $]_{sv}$ **using** *assms*
by (*metis flip-at-simps(1) flip-fresh-fresh fresh-PairD(1)*)
also have ... = (*xa* ↔ *c*) • ($sa[xa::=v']_{sv}$) **using** *subst-sv-subst-branchv-subst-branchlv.eqvt* **by** *presburger*
also have ... = $sa[xa::=v']_{sv}$ **using** *xafr assms* **by** (*simp add*: *flip-fresh-fresh fresh-Pair*)
finally show *?thesis* **by** *simp*
qed

lemma *if-type-eq*:
fixes Γ::Γ **and** *v::v* **and** *z1::x*
assumes *atom z1'* ♯ (*v*, *ca*, (*x*, *b*, *c*) $\#_\Gamma$ Γ, (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1]^v]_{cv}$)) **and** *atom z1* ♯ *v*
and *atom z1* ♯ (*za*,*ca*) **and** *atom z1'* ♯ (*za*,*ca*)
shows (⦃ *z1'* : *ba* | *CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1']^v]_{cv}$ ⦄) = ⦃ *z1* : *ba* | *CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1]^v]_{cv}$ ⦄
proof −
have *atom z1'* ♯ (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1]^v]_{cv}$) **using** *assms fresh-prod4* **by** *blast*
moreover hence (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1']^v]_{cv}$) = (*z1'* ↔ *z1*) • (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1]^v]_{cv}$)
proof −
have (*z1'* ↔ *z1*) • (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1]^v]_{cv}$) = ((*z1'* ↔ *z1*) • (*CE-val v* == *CE-val* (*V-lit ll*)) *IMP* ((*z1'* ↔ *z1*) • $ca[za::=[z1]^v]_{cv}$))
by *auto*
also have ... = ((*CE-val v* == *CE-val* (*V-lit ll*)) *IMP* ((*z1'* ↔ *z1*) • $ca[za::=[z1]^v]_{cv}$))
using ‹*atom z1* ♯ *v*› *assms*
by (*metis* (*mono-tags*) ‹*atom z1'* ♯ (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1]^v]_{cv}$)› *c.fresh(6) c.fresh(7) ce.fresh(1) flip-at-simps(2) flip-fresh-fresh fresh-at-base-permute-iff fresh-def supp-l-empty v.fresh(1)*)
also have ... = ((*CE-val v* == *CE-val* (*V-lit ll*)) *IMP* ($ca[za::=[z1']^v]_{cv}$))
using *assms* **by** *fastforce*
finally show *?thesis* **by** *auto*
qed
ultimately show *?thesis*
using *τ.eq-iff Abs1-eq-iff(3)[of z1' CE-val v* == *CE-val* (*V-lit ll*) *IMP* $ca[za::=[z1']^v]_{cv}$

z1 CE-val v == *CE-val* (*V-lit ll*) *IMP ca*[*za*::=[*z1*]^v]$_{cv}$] **by** *blast*
qed

lemma *subst-sv-var-flip*:
fixes *x*::*x* **and** *s*::*s* **and** *z*::*x*
shows *atom x* ♯ *s* ⟹ ((*x* ↔ *z*) · *s*) = *s*[*z*::=[*x*]^v]$_{sv}$ **and**
atom x ♯ *cs* ⟹ ((*x* ↔ *z*) · *cs*) = *subst-branchv cs z* [*x*]^v **and**
atom x ♯ *css* ⟹ ((*x* ↔ *z*) · *css*) = *subst-branchlv css z* [*x*]^v
apply(*nominal-induct s* **and** *cs* **and** *css avoiding*: *z x rule*: *s-branch-s-branch-list.strong-induct*)
using [[*simproc del*: *alpha-lst*]]
apply (*auto*)
using *subst-tv-var-flip flip-fresh-fresh v.fresh s-branch-s-branch-list.fresh*
subst-v-τ-def subst-v-v-def subst-vv-var-flip subst-v-e-def subst-ev-var-flip pure-fresh **apply** *auto*
defer *1*
using *x-fresh-u* **apply** *blast*
defer *1*
using *x-fresh-u* **apply** *blast*
defer *1*
using *x-fresh-u Abs1-eq-iff*′(*3*) *flip-fresh-fresh*
apply (*simp add*: *subst-v-c-def*)
using *x-fresh-u Abs1-eq-iff*′(*3*) *flip-fresh-fresh*
by (*simp add*: *flip-fresh-fresh*)

instantiation *s* :: *has-subst-v*
begin

definition
subst-v = *subst-sv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*s*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-sv-if subst-v-s-def* **by** *auto*

fix *a*::*x* **and** *tm*::*s* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-sv subst-v-s-def* **by** *simp*

fix *a*::*x* **and** *tm*::*s*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-sv-id subst-v-s-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*s*
show *p* · *subst-v t1 x1 v* = *subst-v* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
using *subst-sv-commute subst-v-s-def* **by** *simp*

fix *x*::*x* **and** *c*::*s* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) · *c*) = *c*[*z*::=[*x*]^v]$_v$
using *subst-sv-var-flip subst-v-s-def* **by** *simp*

fix *x*::*x* **and** *c*::*s* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ *c*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_v$ = *c*[*z*::=*v*]$_v$
using *subst-sv-var-flip subst-v-s-def* **by** *simp*

qed
end

3.10 Type Definition

nominal-function *subst-ft-v* :: *fun-typ* ⇒ *x* ⇒ *v* ⇒ *fun-typ* **where**
atom z ♯ (*x*,*v*) ⟹ *subst-ft-v* (*AF-fun-typ z b c t* (*s*::*s*)) *x v* = *AF-fun-typ z b* $c[x::=v]_{cv}$ $t[x::=v]_{\tau v}$ $s[x::=v]_{sv}$
apply(*simp add*: *eqvt-def subst-ft-v-graph-aux-def*)
apply(*simp add*:*fun-typ.strong-exhaust*)
apply(*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)

proof(*goal-cases*)
case (*1 z xa va c t s za ca ta sa cb*)
hence $c[z::=[\ cb\ ]^v]_{cv} = ca[za::=[\ cb\ ]^v]_{cv}$
by (*metis flip-commute subst-cv-var-flip*)
hence $c[z::=[\ cb\ ]^v]_{cv}[xa::=va]_{cv} = ca[za::=[\ cb\ ]^v]_{cv}[xa::=va]_{cv}$ **by** *auto*
then show *?case* **using** *subst-cv-commute atom-eq-iff fresh-atom fresh-atom-at-base subst-cv-commute-full v.fresh*
using *1 subst-cv-var-flip flip-commute* **by** *metis*
next
case (*2 z xa va c t s za ca ta sa cb*)
hence $t[z::=[\ cb\ ]^v]_{\tau v} = ta[za::=[\ cb\ ]^v]_{\tau v}$ **by** *metis*
hence $t[z::=[\ cb\ ]^v]_{\tau v}[xa::=va]_{\tau v} = ta[za::=[\ cb\ ]^v]_{\tau v}[xa::=va]_{\tau v}$ **by** *auto*
then show *?case* **using** *subst-tv-commute-full 2*
by (*metis atom-eq-iff fresh-atom fresh-atom-at-base v.fresh*(*2*))
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *subst-ftq-v* :: *fun-typ-q* ⇒ *x* ⇒ *v* ⇒ *fun-typ-q* **where**
atom bv ♯ (*x*,*v*) ⟹ *subst-ftq-v* (*AF-fun-typ-some bv ft*) *x v* = (*AF-fun-typ-some bv* (*subst-ft-v ft x v*))
| *subst-ftq-v* (*AF-fun-typ-none ft*) *x v* = (*AF-fun-typ-none* (*subst-ft-v ft x v*))
apply(*simp add*: *eqvt-def subst-ftq-v-graph-aux-def*)
apply(*simp add*:*fun-typ-q.strong-exhaust*)
apply(*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ-q.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
proof(*goal-cases*)
case (*1 bv ft bva fta xa va c*)
then show *?case* **using** *subst-ft-v.simps* **by** (*simp add*: *flip-fresh-fresh*)
qed
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *size-subst-ft*[*simp*]: *size* (*subst-ft-v A x v*) = *size A*
by(*nominal-induct A avoiding*: *x v rule*: *fun-typ.strong-induct*,*auto*)

lemma *forget-subst-ft* [*simp*]: **shows** *atom x* ♯ *A* ⟹ *subst-ft-v A x a* = *A*
by (*nominal-induct A avoiding*: *a x rule*: *fun-typ.strong-induct*,*auto simp*: *fresh-at-base*)

lemma *subst-ft-id* [*simp*]: *subst-ft-v A a* (*V-var a*) = *A*
by(*nominal-induct A avoiding*: *a rule*: *fun-typ.strong-induct*,*auto*)

instantiation *fun-typ* :: *has-subst-v*
begin

definition
subst-v = *subst-ft-v*

instance proof

fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*fun-typ*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ.strong-induct*)
apply(*simp only*: *subst-v-fun-typ-def subst-ft-v.simps*)
using *fun-typ.fresh fresh-subst-v-if* **apply** *simp*
by *auto*

fix *a*::*x* **and** *tm*::*fun-typ* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
proof(*nominal-induct tm avoiding*: *a x rule*:*fun-typ.strong-induct*)
case (*AF-fun-typ x1a x2a x3a x4a x5a*)
then show *?case* **unfolding** *subst-ft-v.simps subst-v-fun-typ-def fun-typ.fresh* **using** *forget-subst-v subst-ft-v.simps subst-v-c-def forget-subst-sv subst-v-τ-def* **by** *fastforce*
qed

fix *a*::*x* **and** *tm*::*fun-typ*
show *subst-v tm a* (*V-var a*) = *tm*
proof(*nominal-induct tm avoiding*: *a x rule*:*fun-typ.strong-induct*)
case (*AF-fun-typ x1a x2a x3a x4a x5a*)
then show *?case* **unfolding** *subst-ft-v.simps subst-v-fun-typ-def fun-typ.fresh* **using** *forget-subst-v subst-ft-v.simps subst-v-c-def forget-subst-sv subst-v-τ-def* **by** *fastforce*
qed

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*fun-typ*
show *p* • *subst-v t1 x1 v* = *subst-v* (*p* • *t1*) (*p* • *x1*) (*p* • *v*)
proof(*nominal-induct t1 avoiding*: *x1 v rule*:*fun-typ.strong-induct*)
case (*AF-fun-typ x1a x2a x3a x4a x5a*)
then show *?case* **unfolding** *subst-ft-v.simps subst-v-fun-typ-def fun-typ.fresh* **using** *forget-subst-v subst-ft-v.simps subst-v-c-def forget-subst-sv subst-v-τ-def* **by** *fastforce*
qed

fix *x*::*x* **and** *c*::*fun-typ* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) • *c*) = $c[z{::=}[x]^v]_v$
apply(*nominal-induct c avoiding*: *x z rule*:*fun-typ.strong-induct*)
by (*auto simp add*: *subst-v-c-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-def*)

fix *x*::*x* **and** *c*::*fun-typ* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ $c[z{::=}[x]^v]_v[x{::=}v]_v = c[z{::=}v]_v$
apply(*nominal-induct c avoiding*: *z x v rule*:*fun-typ.strong-induct*)
apply *auto*
by (*auto simp add*: *subst-v-c-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-def*)

qed
end

instantiation *fun-typ-q* :: *has-subst-v*
begin

definition
subst-v = *subst-ftq-v*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*fun-typ-q*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ-q.strong-induct*,*auto*)
apply(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)
by (*metis* (*no-types*) *fresh-subst-v-if subst-v-fun-typ-def*)+

fix *i*::*x* **and** *t*::*fun-typ-q* **and** *x*::*v*
show *atom i* ♯ *t* ⟹ *subst-v t i x* = *t*
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *i*::*x* **and** *t*::*fun-typ-q*
show *subst-v t i* (*V-var i*) = *t* **using** *subst-cv-id subst-v-fun-typ-def*
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*fun-typ-q*
show *p* · *subst-v t1 x1 v* = *subst-v* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
apply(*nominal-induct t1 avoiding*: *v x1 rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *x*::*x* **and** *c*::*fun-typ-q* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) · *c*) = $c[z::=[x]^v]_v$
apply(*nominal-induct c avoiding*: *x z rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *x*::*x* **and** *c*::*fun-typ-q* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ $c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v$
apply(*nominal-induct c avoiding*: *z x v rule*:*fun-typ-q.strong-induct*,*auto*)
apply(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)
by (*metis subst-v-fun-typ-def flip-bv-x-cancel subst-ft-v.eqvt subst-v-simple-commute v.perm-simps*)+
qed

end

3.11 Variable Context

lemma *subst-dv-fst-eq*:
  fst ' *setD* ($\Delta[x::=v]_{\Delta v}$) = *fst* ' *setD* Δ
  by(*induct* Δ *rule*: Δ-*induct*,*simp*,*force*)

lemma *subst-gv-member-iff*:
  fixes $x'::x$ **and** $x::x$ **and** $v::v$ **and** $c'::c$
  assumes $(x',b',c') \in$ *toSet* Γ **and** *atom* $x \notin$ *atom-dom* Γ
  shows $(x',b',c'[x::=v]_{cv}) \in$ *toSet* $\Gamma[x::=v]_{\Gamma v}$
proof −
  have $x' \neq x$ **using** *assms fresh-dom-free2* **by** *metis*
  then show *?thesis* **using** *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
    case *GNil*
    then show *?case* **by** *auto*
  next
    case (*GCons x1 b1 c1* Γ')
    show *?case* **proof**(*cases* $(x',b',c') = (x1,b1,c1)$)
      case *True*
    hence $((x1, b1, c1) \#_\Gamma \Gamma')[x::=v]_{\Gamma v} = ((x1, b1, c1[x::=v]_{cv}) \#_\Gamma (\Gamma'[x::=v]_{\Gamma v}))$ **using** *subst-gv.simps* ‹$x' \neq x$› **by** *auto*
      then show *?thesis* **using** *True* **by** *auto*
    next
      case *False*
      have $x1 \neq x$ **using** *fresh-def fresh-GCons fresh-Pair supp-at-base GCons fresh-dom-free2* **by** *auto*
      hence $(x', b', c') \in$ *toSet* Γ' **using** *GCons False toSet.simps* **by** *auto*
       moreover have *atom* $x \notin$ *atom-dom* Γ' **using** *fresh-GCons GCons dom.simps toSet.simps* **by** *simp*
      ultimately have $(x', b', c'[x::=v]_{cv}) \in$ *toSet* $\Gamma'[x::=v]_{\Gamma v}$ **using** *GCons* **by** *auto*
      hence $(x', b', c'[x::=v]_{cv}) \in$ *toSet* $((x1, b1, c1[x::=v]_{cv}) \#_\Gamma (\Gamma'[x::=v]_{\Gamma v}))$ **by** *auto*
      then show *?thesis* **using** *subst-gv.simps* ‹$x1 \neq x$› **by** *auto*
    qed
  qed
qed

lemma *fresh-subst-gv-if*:
  fixes $j::atom$ **and** $i::x$ **and** $x::v$ **and** $t::\Gamma$
  assumes $j \sharp t \wedge j \sharp x$
  shows ($j \sharp$ *subst-gv t i x*)
  using *assms* **proof**(*induct t rule*: Γ-*induct*)
  case *GNil*
  then show *?case* **using** *subst-gv.simps fresh-GNil* **by** *auto*
next
  case (*GCons* x' b' c' Γ')
  then show *?case* **unfolding** *subst-gv.simps* **using** *fresh-GCons fresh-subst-cv-if* **by** *auto*
qed

3.12 Lookup

lemma *set-GConsD*: $y \in$ *toSet* $(x \#_\Gamma xs) \Longrightarrow y=x \vee y \in$ *toSet* xs
  by *auto*

lemma *subst-g-assoc-cons*:
assumes $x \neq x'$
shows $(((x', b', c') \#_\Gamma \Gamma')[x::=v]_{\Gamma v} @ G) = ((x', b', c'[x::=v]_{cv}) \#_\Gamma ((\Gamma'[x::=v]_{\Gamma v}) @ G))$
using *subst-gv.simps append-g.simps assms* **by** *auto*

end

Chapter 4

Basic Type Variable Substitution

4.1 Class

```
class has-subst-b = fs +
  fixes subst-b :: 'a::fs ⇒ bv ⇒ b ⇒ 'a::fs (‹-[-::=-]b› [1000,50,50] 1000)

assumes fresh-subst-if:  j ♯ (t[i::=x]b ) ⟷ (atom i ♯ t ∧ j ♯ t) ∨ (j ♯ x ∧ (j ♯ t ∨ j = atom i))
  and    forget-subst[simp]:  atom a ♯ tm ⟹ tm[a::=x]b  = tm
  and    subst-id[simp]:      tm[a::=(B-var a)]b  = tm
  and    eqvt[simp,eqvt]:          (p::perm) · (subst-b t1 x1 v ) = (subst-b (p ·t1) (p ·x1) (p ·v) )
  and    flip-subst[simp]:    atom bv ♯ c ⟹ ((bv ↔ z) · c) = c[z::=B-var bv]b
  and    flip-subst-subst[simp]: atom bv ♯ c ⟹ ((bv ↔ z) · c)[bv::=v]b = c[z::=v]b
begin

lemmas flip-subst-b = flip-subst-subst

lemma subst-b-simple-commute:
  fixes x::bv
  assumes atom x ♯ c
  shows (c[z::=B-var x]b)[x::=b]b = c[z::=b]b
proof −
  have (c[z::=B-var x]b)[x::=b]b = (( x ↔ z) · c)[x::=b]b using flip-subst assms by simp
  thus ?thesis using flip-subst-subst assms by simp
qed

lemma subst-b-flip-eq-one:
  fixes z1::bv and z2::bv and x1::bv and x2::bv
  assumes [[atom z1]]lst. c1 = [[atom z2]]lst. c2
    and atom x1 ♯ (z1,z2,c1,c2)
  shows (c1[z1::=B-var x1]b) = (c2[z2::=B-var x1]b)
proof −
  have (c1[z1::=B-var x1]b) = (x1 ↔ z1) · c1 using assms flip-subst by auto
  moreover have  (c2[z2::=B-var x1]b) = (x1 ↔ z2) · c2 using assms flip-subst by auto
  ultimately show ?thesis using Abs1-eq-iff-all(3)[of z1 c1 z2 c2 z1]  assms
    by (metis Abs1-eq-iff-fresh(3) flip-commute)
qed

lemma subst-b-flip-eq-two:
```

fixes *z1*::*bv* **and** *z2*::*bv* **and** *x1*::*bv* **and** *x2*::*bv*
assumes [[*atom z1*]]*lst. c1* = [[*atom z2*]]*lst. c2*
shows (*c1*[*z1*::=*b*]$_b$) = (*c2*[*z2*::=*b*]$_b$)
proof −
obtain *x*::*bv* **where** ∗:*atom x* ♯ (*z1*,*z2*,*c1*,*c2*) **using** *obtain-fresh* **by** *metis*
hence (*c1*[*z1*::=*B-var x*]$_b$) = (*c2*[*z2*::=*B-var x*]$_b$) **using** *subst-b-flip-eq-one*[*OF assms*, *of x*] **by** *metis*
hence (*c1*[*z1*::=*B-var x*]$_b$)[*x*::=*b*]$_b$ = (*c2*[*z2*::=*B-var x*]$_b$)[*x*::=*b*]$_b$ **by** *auto*
thus *?thesis* **using** *subst-b-simple-commute* ∗ *fresh-prod4* **by** *metis*
qed

lemma *subst-b-fresh-x*:
fixes *tm*::′*a*::*fs* **and** *x*::*x*
shows *atom x* ♯ *tm* = *atom x* ♯ *tm*[*bv*::=*b*′]$_b$
using *fresh-subst-if*[*of atom x tm bv b*′] **using** *x-fresh-b* **by** *auto*

lemma *subst-b-x-flip*[*simp*]:
fixes *x*′::*x* **and** *x*::*x* **and** *bv*::*bv*
shows ((*x*′ ↔ *x*) · *tm*)[*bv*::=*b*′]$_b$ = (*x*′ ↔ *x*) · (*tm*[*bv*::=*b*′]$_b$)
proof −
have (*x*′ ↔ *x*) · *bv* = *bv* **using** *pure-supp flip-fresh-fresh* **by** *force*
moreover have (*x*′ ↔ *x*) · *b*′ = *b*′ **using** *x-fresh-b flip-fresh-fresh* **by** *auto*
ultimately show *?thesis* **using** *eqvt* **by** *simp*
qed

end

4.2 Base Type

nominal-function *subst-bb* :: *b* ⇒ *bv* ⇒ *b* ⇒ *b* **where**
subst-bb (*B-var bv2*) *bv1 b* = (*if bv1* = *bv2 then b else* (*B-var bv2*))
| *subst-bb B-int bv1 b* = *B-int*
| *subst-bb B-bool bv1 b* = *B-bool*
| *subst-bb* (*B-id s*) *bv1 b* = *B-id s*
| *subst-bb* (*B-pair b1 b2*) *bv1 b* = *B-pair* (*subst-bb b1 bv1 b*) (*subst-bb b2 bv1 b*)
| *subst-bb B-unit bv1 b* = *B-unit*
| *subst-bb B-bitvec bv1 b* = *B-bitvec*
| *subst-bb* (*B-app s b2*) *bv1 b* = *B-app s* (*subst-bb b2 bv1 b*)
apply (*simp add*: *eqvt-def subst-bb-graph-aux-def*)
apply (*simp add*: *eqvt-def subst-bb-graph-aux-def*)
by (*auto*,*meson b.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-bb-abbrev :: *b* ⇒ *bv* ⇒ *b* ⇒ *b* (‹-[-::=-]$_{bb}$› [*1000*,*50*,*50*] *1000*)
where
b[*bv*::=*b*′]$_{bb}$ ≡ *subst-bb b bv b*′

instantiation *b* :: *has-subst-b*
begin
definition *subst-b* = *subst-bb*

instance proof

```
  fix j::atom and i::bv and  x::b and t::b
  show  j ♯ subst-b t i x = (atom i ♯ t ∧ j ♯ t ∨ j ♯ x ∧ (j ♯ t ∨ j = atom i))
  proof (induct t rule: b.induct)
    case (B-id x)
    then show ?case using subst-bb.simps fresh-def pure-fresh subst-b-b-def by auto
  next
    case (B-var x)
    then show ?case using subst-bb.simps fresh-def pure-fresh subst-b-b-def by auto
  next
    case (B-app x1 x2)
    then show ?case using subst-bb.simps fresh-def pure-fresh subst-b-b-def by auto
  qed(auto simp add: subst-bb.simps fresh-def pure-fresh subst-b-b-def)+

  fix a::bv and tm::b and x::b
  show atom a ♯ tm ⟹ tm[a::=x]_b = tm
    by (induct tm rule: b.induct, auto simp add: fresh-at-base subst-bb.simps subst-b-b-def)

  fix a::bv and tm::b
  show subst-b tm a (B-var a) = tm using subst-bb.simps  subst-b-b-def
    by (induct tm rule: b.induct, auto simp add: fresh-at-base subst-bb.simps subst-b-b-def)

  fix p::perm and x1::bv and v::b and t1::b
  show p · subst-b t1 x1 v  = subst-b  (p · t1) (p · x1) (p · v)
    by (induct tm rule: b.induct, auto simp add: fresh-at-base subst-bb.simps subst-b-b-def)

  fix  bv::bv and c::b and z::bv
  show atom bv ♯ c ⟹ ((bv ↔ z) · c) = c[z::=B-var bv]_b
    by (induct c rule: b.induct,  (auto simp add: fresh-at-base subst-bb.simps subst-b-b-def permute-pure
pure-supp )+)

  fix bv::bv and c::b and z::bv and v::b
  show atom bv ♯ c ⟹ ((bv ↔ z) · c)[bv::=v]_b = c[z::=v]_b
    by (induct c rule: b.induct,  (auto simp add: fresh-at-base subst-bb.simps subst-b-b-def permute-pure
pure-supp )+)
qed
end

lemma subst-bb-inject:
  assumes b1 = b2[bv::=b]_bb and b2 ≠ B-var bv
  shows
    b1 = B-int ⟹ b2 = B-int and
    b1 = B-bool ⟹ b2 = B-bool and
    b1 = B-unit ⟹ b2 = B-unit and
    b1 = B-bitvec ⟹ b2 = B-bitvec and
    b1 = B-pair b11 b12 ⟹ (∃ b11' b12' . b11 = b11'[bv::=b]_bb ∧ b12 = b12'[bv::=b]_bb ∧ b2 = B-pair
b11' b12') and
    b1 = B-var bv' ⟹ b2 = B-var bv' and
    b1 = B-id tyid  ⟹ b2 = B-id tyid and
    b1 = B-app tyid b11 ⟹ (∃ b11'. b11= b11'[bv::=b]_bb ∧ b2 = B-app tyid b11')
  using assms by(nominal-induct b2 rule:b.strong-induct,auto+)

lemma flip-b-subst4:
```

fixes *b1*::*b* **and** *bv1*::*bv* **and** *c*::*bv* **and** *b*::*b*
assumes *atom c* ♯ (*b1*,*bv1*)
shows $b1[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot b1)[c ::= b]_{bb}$
using *assms* **proof**(*nominal-induct b1 rule*: *b.strong-induct*)
case *B-int*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case *B-bool*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case (*B-id x*)
hence *atom bv1* ♯ *x* ∧ *atom c* ♯ *x* **using** *fresh-def pure-supp* **by** *auto*
hence ((*bv1* ↔ *c*) · *B-id x*) = *B-id x* **using** *fresh-Pair b.fresh*(*3*) *flip-fresh-fresh b.perm-simps fresh-def pure-supp* **by** *metis*
then show *?case* **using** *subst-bb.simps* **by** *simp*
next
case (*B-pair x1 x2*)
hence $x1[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot x1)[c::=b]_{bb}$ **using** *b.perm-simps*(*4*) *b.fresh*(*4*) *fresh-Pair* **by** *metis*
moreover have $x2[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot x2)[c::=b]_{bb}$ **using** *b.perm-simps*(*4*) *b.fresh*(*4*) *fresh-Pair B-pair* **by** *metis*
ultimately show *?case* **using** *subst-bb.simps*(*5*) *b.perm-simps*(*4*) *b.fresh*(*4*) *fresh-Pair* **by** *auto*
next
case *B-unit*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case *B-bitvec*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case (*B-var x*)
then show *?case* **proof**(*cases x*=*bv1*)
case *True*
then show *?thesis* **using** *B-var subst-bb.simps b.perm-simps* **by** *simp*
next
case *False*
moreover have *x*≠*c* **using** *B-var b.fresh fresh-def supp-at-base fresh-Pair* **by** *fastforce*
ultimately show *?thesis* **using** *B-var subst-bb.simps*(*1*) *b.perm-simps*(*7*) **by** *simp*
qed
next
case (*B-app x1 x2*)
hence $x2[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot x2)[c::=b]_{bb}$ **using** *b.perm-simps b.fresh fresh-Pair* **by** *metis*
thus *?case* **using** *subst-bb.simps b.perm-simps b.fresh fresh-Pair B-app*
by (*simp add*: *permute-pure*)
qed

lemma *subst-bb-flip-sym*:
fixes *b1*::*b* **and** *b2*::*b*
assumes *atom c* ♯ *b* **and** *atom c* ♯ (*bv1*,*bv2*, *b1*, *b2*) **and** (*bv1* ↔ *c*) · *b1* = (*bv2* ↔ *c*) · *b2*
shows $b1[bv1::=b]_{bb} = b2[bv2::=b]_{bb}$
using *assms flip-b-subst4*[*of c b1 bv1 b*] *flip-b-subst4*[*of c b2 bv2 b*] *fresh-prod4 fresh-Pair* **by** *simp*

4.3 Value

nominal-function *subst-vb* :: *v* ⇒ *bv* ⇒ *b* ⇒ *v* **where**
subst-vb (*V-lit l*) *x v* = *V-lit l*
| *subst-vb* (*V-var y*) *x v* = *V-var y*
| *subst-vb* (*V-cons tyid c v′*) *x v* = *V-cons tyid c* (*subst-vb v′ x v*)
| *subst-vb* (*V-consp tyid c b v′*) *x v* = *V-consp tyid c* (*subst-bb b x v*) (*subst-vb v′ x v*)
| *subst-vb* (*V-pair v1 v2*) *x v* = *V-pair* (*subst-vb v1 x v*) (*subst-vb v2 x v*)
apply (*simp add*: *eqvt-def subst-vb-graph-aux-def*)
apply *auto*
using *v.strong-exhaust* **by** *meson*
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-vb-abbrev :: *v* ⇒ *bv* ⇒ *b* ⇒ *v* (‹-[-::=-]$_{vb}$› [*1000*,*50*,*50*] *500*)
where
$e[bv::=b]_{vb}$ ≡ *subst-vb e bv b*

instantiation *v* :: *has-subst-b*
begin
definition *subst-b* = *subst-vb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*v*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*induct t rule*: *v.induct*)
case (*V-lit l*)
have *j* ♯ *subst-b* (*V-lit l*) *i x* = *j* ♯ (*V-lit l*) **using** *subst-vb.simps fresh-def pure-fresh subst-b-v-def v.supp v.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-b-v-def* **by** *auto*
also have ... = *True* **using** *fresh-at-base v.fresh l.fresh supp-l-empty fresh-def* **by** *metis*
moreover have (*atom i* ♯ (*V-lit l*) ∧ *j* ♯ (*V-lit l*) ∨ *j* ♯ *x* ∧ (*j* ♯ (*V-lit l*) ∨ *j* = *atom i*)) = *True* **using** *fresh-at-base v.fresh l.fresh supp-l-empty fresh-def* **by** *metis*
ultimately show *?case* **by** *simp*
next
case (*V-var y*)
then show *?case* **using** *subst-b-v-def subst-vb.simps pure-fresh* **by** *force*
next
case (*V-pair x1a x2a*)
then show *?case* **using** *subst-b-v-def subst-vb.simps* **by** *auto*
next
case (*V-cons x1a x2a x3*)
then show *?case* **using** *V-cons subst-b-v-def subst-vb.simps pure-fresh* **by** *force*
next
case (*V-consp x1a x2a x3 x4*)
then show *?case* **using** *subst-b-v-def subst-vb.simps pure-fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-b-v-def* **by** *fastforce*
qed

fix *a*::*bv* **and** *tm*::*v* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
apply(*induct tm rule*: *v.induct*)
apply(*auto simp add*: *fresh-at-base subst-vb.simps subst-b-v-def*)
using *has-subst-b-class.fresh-subst-if subst-b-b-def e.fresh*

using *has-subst-b-class.forget-subst* **by** *fastforce*

fix *a*::*bv* **and** *tm*::*v*
show *subst-b tm a* (*B-var a*) = *tm* **using** *subst-bb.simps subst-b-b-def*
apply (*induct tm rule*: *v.induct*)
apply(*auto simp add*: *fresh-at-base subst-vb.simps subst-b-v-def*)
using *has-subst-b-class.fresh-subst-if subst-b-b-def e.fresh*
using *has-subst-b-class.subst-id* **by** *metis*

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*v*
show *p* · *subst-b t1 x1 v* = *subst-b* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
apply(*induct tm rule*: *v.induct*)
apply(*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)
using *has-subst-b-class.eqvt subst-b-b-def e.fresh*
using *has-subst-b-class.eqvt*
by (*simp add*: *subst-b-v-def*)+

fix *bv*::*bv* **and** *c*::*v* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*induct c rule*: *v.induct*, (*auto simp add*: *fresh-at-base subst-vb.simps subst-b-v-def permute-pure pure-supp*)+)
apply (*metis flip-fresh-fresh flip-l-eq permute-flip-cancel2*)
using *fresh-at-base flip-fresh-fresh*[*of bv x z*]
apply (*simp add*: *flip-fresh-fresh*)
using *subst-b-b-def* **by** *argo*

fix *bv*::*bv* **and** *c*::*v* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
apply (*induct c rule*: *v.induct*, (*auto simp add*: *fresh-at-base subst-vb.simps subst-b-v-def permute-pure pure-supp*)+)
apply (*metis flip-fresh-fresh flip-l-eq permute-flip-cancel2*)
using *fresh-at-base flip-fresh-fresh*[*of bv x z*]
apply (*simp add*: *flip-fresh-fresh*)
using *subst-b-b-def flip-subst-subst* **by** *fastforce*

qed
end

4.4 Constraints Expressions

nominal-function *subst-ceb* :: *ce* ⇒ *bv* ⇒ *b* ⇒ *ce* **where**
subst-ceb ((*CE-val v′*)) *bv b* = (*CE-val* (*subst-vb v′ bv b*))
| *subst-ceb* ((*CE-op opp v1 v2*)) *bv b* = ((*CE-op opp* (*subst-ceb v1 bv b*)(*subst-ceb v2 bv b*)))
| *subst-ceb* ((*CE-fst v′*)) *bv b* = *CE-fst* (*subst-ceb v′ bv b*)
| *subst-ceb* ((*CE-snd v′*)) *bv b* = *CE-snd* (*subst-ceb v′ bv b*)
| *subst-ceb* ((*CE-len v′*)) *bv b* = *CE-len* (*subst-ceb v′ bv b*)
| *subst-ceb* (*CE-concat v1 v2*) *bv b* = *CE-concat* (*subst-ceb v1 bv b*) (*subst-ceb v2 bv b*)
apply (*simp add*: *eqvt-def subst-ceb-graph-aux-def*)
apply *auto*
by (*meson ce.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-ceb-abbrev :: *ce* ⇒ *bv* ⇒ *b* ⇒ *ce* (‹-[-::=-]$_{ceb}$› [*1000*,*50*,*50*] *500*)
where
ce[*bv*::=*b*]$_{ceb}$ ≡ *subst-ceb ce bv b*

instantiation *ce* :: *has-subst-b*
begin
definition *subst-b* = *subst-ceb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*ce*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*induct t rule*: *ce.induct*)
case (*CE-val v*)
then show *?case* **using** *subst-ceb.simps fresh-def pure-fresh subst-b-ce-def ce.supp v.supp ce.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-b-v-def*
by *metis*
next
case (*CE-op opp v1 v2*)

have (*j* ♯ *v1*[*i*::=*x*]$_{ceb}$ ∧ *j* ♯ *v2*[*i*::=*x*]$_{ceb}$) = ((*atom i* ♯ *v1* ∧ *atom i* ♯ *v2*) ∧ *j* ♯ *v1* ∧ *j* ♯ *v2* ∨ *j* ♯ *x* ∧ (*j* ♯ *v1* ∧ *j* ♯ *v2* ∨ *j* = *atom i*))
using *has-subst-b-class.fresh-subst-if subst-b-v-def*
using *CE-op.hyps*(*1*) *CE-op.hyps*(*2*) *subst-b-ce-def* **by** *auto*

thus *?case* **unfolding** *subst-ceb.simps subst-b-ce-def ce.fresh*
using *fresh-def pure-fresh opp.fresh subst-b-v-def opp.exhaust fresh-e-opp-all*
by (*metis* (*full-types*))
next
case (*CE-concat x1a x2*)
then show *?case* **using** *subst-ceb.simps subst-b-ce-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def ce.fresh* **by** *force*
next
case (*CE-fst x*)
then show *?case* **using** *subst-ceb.simps subst-b-ce-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def ce.fresh* **by** *metis*

next
case (*CE-snd x*)
then show *?case* **using** *subst-ceb.simps subst-b-ce-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def ce.fresh* **by** *metis*
next
case (*CE-len x*)
then show *?case* **using** *subst-ceb.simps subst-b-ce-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def ce.fresh* **by** *metis*
qed

fix *a*::*bv* **and** *tm*::*ce* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
apply(*induct tm rule*: *ce.induct*)
apply(*auto simp add*: *fresh-at-base subst-ceb.simps subst-b-ce-def*)
using *has-subst-b-class.fresh-subst-if subst-b-b-def e.fresh*

using *has-subst-b-class.forget-subst subst-b-v-def* **apply** *metis+*
done

fix *a::bv* **and** *tm::ce*
show *subst-b tm a (B-var a) = tm* **using** *subst-bb.simps subst-b-b-def*
apply (*induct tm rule*: *ce.induct*)
apply(*auto simp add*: *fresh-at-base subst-ceb.simps subst-b-ce-def*)
using *has-subst-b-class.fresh-subst-if subst-b-b-def e.fresh*
using *has-subst-b-class.subst-id subst-b-v-def* **apply** *metis+*
done

fix *p::perm* **and** *x1::bv* **and** *v::b* **and** *t1::ce*
show *p* $\cdot$ *subst-b t1 x1 v* = *subst-b* (*p* $\cdot$ *t1*) (*p* $\cdot$ *x1*) (*p* $\cdot$ *v*)
apply(*induct tm rule*: *ce.induct*)
apply(*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)
using *has-subst-b-class.eqvt subst-b-b-def ce.fresh*
using *has-subst-b-class.eqvt*
by (*simp add*: *subst-b-ce-def*)+

fix *bv::bv* **and** *c::ce* **and** *z::bv*
show *atom bv* $\sharp$ *c* $\Longrightarrow$ ((*bv* $\leftrightarrow$ *z*) $\cdot$ *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*induct c rule*: *ce.induct*, (*auto simp add*: *fresh-at-base subst-ceb.simps subst-b-ce-def permute-pure pure-supp*)+)
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def* **apply** *metis*
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def*
by (*simp add*: *flip-fresh-fresh fresh-opp-all*)

fix *bv::bv* **and** *c::ce* **and** *z::bv* **and** *v::b*
show *atom bv* $\sharp$ *c* $\Longrightarrow$ ((*bv* $\leftrightarrow$ *z*) $\cdot$ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
proof (*induct c rule*: *ce.induct*)
case (*CE-val x*)
then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next
case (*CE-op x1a x2 x3*)
then show *?case* **unfolding** *subst-ceb.simps subst-b-ce-def ce.perm-simps* **using** *flip-subst-subst subst-b-v-def opp.perm-simps opp.strong-exhaust*
by (*metis* (*full-types*) *ce.fresh*(*2*))
next
case (*CE-concat x1a x2*)
then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next
case (*CE-fst x*)
then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next
case (*CE-snd x*)
then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next

case (*CE-len x*)
then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fast-force*
qed
qed
end

4.5 Constraints

nominal-function *subst-cb* :: *c* ⇒ *bv* ⇒ *b* ⇒ *c* **where**
subst-cb (*C-true*) *x v* = *C-true*
| *subst-cb* (*C-false*) *x v* = *C-false*
| *subst-cb* (*C-conj c1 c2*) *x v* = *C-conj* (*subst-cb c1 x v*) (*subst-cb c2 x v*)
| *subst-cb* (*C-disj c1 c2*) *x v* = *C-disj* (*subst-cb c1 x v*) (*subst-cb c2 x v*)
| *subst-cb* (*C-imp c1 c2*) *x v* = *C-imp* (*subst-cb c1 x v*) (*subst-cb c2 x v*)
| *subst-cb* (*C-eq e1 e2*) *x v* = *C-eq* (*subst-ceb e1 x v*) (*subst-ceb e2 x v*)
| *subst-cb* (*C-not c*) *x v* = *C-not* (*subst-cb c x v*)
apply (*simp add*: *eqvt-def subst-cb-graph-aux-def*)
apply *auto*
using *c.strong-exhaust* **apply** *metis*
done
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-cb-abbrev :: *c* ⇒ *bv* ⇒ *b* ⇒ *c* (‹-[-::=-]$_{cb}$› [*1000*,*50*,*50*] *500*)
where
c[*bv*::=*b*]$_{cb}$ ≡ *subst-cb c bv b*

instantiation *c* :: *has-subst-b*
begin
definition *subst-b* = *subst-cb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*c*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
by (*induct t rule*: *c.induct*, *unfold subst-cb.simps subst-b-c-def c.fresh*,
(*metis has-subst-b-class.fresh-subst-if subst-b-ce-def c.fresh*)+
)

fix *a*::*bv* **and** *tm*::*c* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
by(*induct tm rule*: *c.induct*, *unfold subst-cb.simps subst-b-c-def c.fresh*,
(*metis has-subst-b-class.forget-subst subst-b-ce-def*)+)

fix *a*::*bv* **and** *tm*::*c*
show *subst-b tm a* (*B-var a*) = *tm* **using** *subst-bb.simps subst-b-c-def*
by(*induct tm rule*: *c.induct*, *unfold subst-cb.simps subst-b-c-def c.fresh*,
(*metis has-subst-b-class.subst-id subst-b-ce-def*)+)

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*c*
show *p* · *subst-b t1 x1 v* = *subst-b* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
apply(*induct tm rule*: *c.induct*,*unfold subst-cb.simps subst-b-c-def c.fresh*)

by(*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)

fix *bv*::*bv* **and** *c*::*c* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = $c[z{::=}B\text{-}var\ bv]_b$
apply (*induct c rule*: *c.induct*, (*auto simp add*: *fresh-at-base subst-cb.simps subst-b-c-def permute-pure pure-supp*)+)
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def* **apply** *metis*
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def*
apply (*metis opp.perm-simps*(*2*) *opp.strong-exhaust*)+
done

fix *bv*::*bv* **and** *c*::*c* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ $((bv \leftrightarrow z) \cdot c)[bv{::=}v]_b = c[z{::=}v]_b$
apply (*induct c rule*: *c.induct*, (*auto simp add*: *fresh-at-base subst-cb.simps subst-b-c-def permute-pure pure-supp*)+)
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def*
using *flip-subst-subst* **apply** *fastforce*
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def*
opp.perm-simps(*2*) *opp.strong-exhaust*
proof −
fix *x1a* :: *ce* **and** *x2* :: *ce*
assume *a1*: *atom bv* ♯ *x2*
then have $((bv \leftrightarrow z) \cdot x2)[bv{::=}v]_b = x2[z{::=}v]_b$
by (*metis flip-subst-subst*)
then show $x2[z{::=}B\text{-}var\ bv]_b[bv{::=}v]_{ceb} = x2[z{::=}v]_{ceb}$
using *a1* **by** (*simp add*: *subst-b-ce-def*)
qed

qed
end

4.6 Types

nominal-function *subst-tb* :: τ ⇒ *bv* ⇒ *b* ⇒ τ **where**
subst-tb (⦃ *z* : *b2* | *c* ⦄) *bv1 b1* = ⦃ *z* : $b2[bv1{::=}b1]_{bb}$ | $c[bv1{::=}b1]_{cb}$ ⦄
proof(*goal-cases*)
case *1*
then show *?case* **using** *eqvt-def subst-tb-graph-aux-def* **by** *force*
next
case (*2 x y*)
then show *?case* **by** *auto*
next
case (*3 P x*)
then show *?case* **using** *eqvt-def subst-tb-graph-aux-def* τ*.strong-exhaust*
by (*metis b-of.cases prod-cases3*)
next
case (*4 z′ b2′ c′ bv1′ b1′ z b2 c bv1 b1*)
show *?case* **unfolding** τ*.eq-iff* **proof**
have ∗:[[*atom z′*]]*lst. c′* = [[*atom z*]]*lst. c* **using** τ*.eq-iff 4* **by** *auto*
show [[*atom z′*]]*lst.* $c'[bv1'{::=}b1']_{cb}$ = [[*atom z*]]*lst.* $c[bv1{::=}b1]_{cb}$
proof(*subst Abs1-eq-iff-all*(*3*),*rule*,*rule*,*rule*)

fix *ca::x*
assume *atom ca* ♯ *z* **and** *1:atom ca* ♯ $(z', z, c'[bv1'::=b1']_{cb}, c[bv1::=b1]_{cb})$
hence *2:atom ca* ♯ (c',c) **using** *fresh-subst-if subst-b-c-def fresh-Pair fresh-prod4 fresh-at-base subst-b-fresh-x* **by** *metis*
hence $(z' \leftrightarrow ca) \cdot c' = (z \leftrightarrow ca) \cdot c$ **using** *1 2 ∗ Abs1-eq-iff-all(3)* **by** *auto*
hence $((z' \leftrightarrow ca) \cdot c')[bv1'::=b1']_{cb} = ((z \leftrightarrow ca) \cdot c)[bv1'::=b1']_{cb}$ **by** *auto*
hence $(z' \leftrightarrow ca) \cdot c'[(z' \leftrightarrow ca) \cdot bv1'::=(z' \leftrightarrow ca) \cdot b1']_{cb} = (z \leftrightarrow ca) \cdot c[(z \leftrightarrow ca) \cdot bv1'::=(z \leftrightarrow ca) \cdot b1']_{cb}$ **by** *auto*
thus $(z' \leftrightarrow ca) \cdot c'[bv1'::=b1']_{cb} = (z \leftrightarrow ca) \cdot c[bv1::=b1]_{cb}$ **using** *4 flip-x-b-cancel* **by** *simp*
qed
show $b2'[bv1'::=b1']_{bb} = b2[bv1::=b1]_{bb}$ **using** *4* **by** *simp*
qed
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-tb-abbrev :: $\tau \Rightarrow bv \Rightarrow b \Rightarrow \tau$ (‹-[-::=-]$_{\tau b}$› [*1000,50,50*] *1000*)
where
$t[bv::=b']_{\tau b} \equiv$ *subst-tb t bv b′*

instantiation τ :: *has-subst-b*
begin
definition *subst-b* = *subst-tb*

instance proof
fix *j::atom* **and** *i::bv* **and** *x::b* **and** *t::τ*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*nominal-induct t avoiding*: *i x j rule*: *τ.strong-induct*)
case (*T-refined-type z b c*)
then show *?case*
unfolding *subst-b-τ-def subst-tb.simps τ.fresh*
using *fresh-subst-if*[*of j b i x*] *subst-b-b-def subst-b-c-def*
by (*metis has-subst-b-class.fresh-subst-if list.distinct*(*1*) *list.set-cases not-self-fresh set-ConsD*)
qed

fix *a::bv* **and** *tm::τ* **and** *x::b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
proof (*nominal-induct tm avoiding*: *a x rule*: *τ.strong-induct*)
case (*T-refined-type xx bb cc*)
moreover hence *atom a* ♯ *bb* ∧ *atom a* ♯ *cc* **using** *τ.fresh* **by** *auto*
ultimately show *?case*
unfolding *subst-b-τ-def subst-tb.simps*
using *forget-subst subst-b-b-def subst-b-c-def forget-subst τ.fresh* **by** *metis*
qed

fix *a::bv* **and** *tm::τ*
show *subst-b tm a* (*B-var a*) = *tm*
proof (*nominal-induct tm rule*: *τ.strong-induct*)
case (*T-refined-type xx bb cc*)
thus *?case*
unfolding *subst-b-τ-def subst-tb.simps*

using *subst-id subst-b-b-def subst-b-c-def* **by** *metis*
qed

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::τ
show *p* · *subst-b t1 x1 v* = *subst-b* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
by (*induct tm rule*: τ*.induct*, *auto simp add*: *fresh-at-base subst-tb.simps subst-b-*τ*-def subst-bb.simps subst-b-b-def*)

fix *bv*::*bv* **and** *c*::τ **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*induct c rule*: τ*.induct*, (*auto simp add*: *fresh-at-base subst-ceb.simps subst-b-ce-def permute-pure pure-supp*)+)
using *flip-fresh-fresh permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-c-def subst-b-b-def*
by (*simp add*: *flip-fresh-fresh subst-b-*τ*-def*)

fix *bv*::*bv* **and** *c*::τ **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
proof (*induct c rule*: τ*.induct*)
case (*T-refined-type x1a x2a x3a*)
hence *atom bv* ♯ *x2a* ∧ *atom bv* ♯ *x3a* ∧ *atom bv* ♯ *x1a* **using** *fresh-at-base* τ*.fresh* **by** *simp*
then show *?case*
unfolding *subst-tb.simps subst-b-*τ*-def* τ*.perm-simps*
using *fresh-at-base flip-fresh-fresh*[*of bv x1a z*] *flip-subst-subst subst-b-b-def subst-b-c-def T-refined-type*

proof −
have *atom z* ♯ *x1a*
by (*metis b.fresh*(*7*) *fresh-at-base*(*2*) *x-fresh-b*)
then show ⦃ (*bv* ↔ *z*) · *x1a* : ((*bv* ↔ *z*) · *x2a*)[*bv*::=*v*]$_{bb}$ | ((*bv* ↔ *z*) · *x3a*)[*bv*::=*v*]$_{cb}$ ⦄ = ⦃ *x1a* : *x2a*[*z*::=*v*]$_{bb}$ | *x3a*[*z*::=*v*]$_{cb}$ ⦄
by (*metis* ‹⟦*atom bv* ♯ *x1a*; *atom z* ♯ *x1a*⟧ ⟹ (*bv* ↔ *z*) · *x1a* = *x1a*› ‹*atom bv* ♯ *x2a* ∧ *atom bv* ♯ *x3a* ∧ *atom bv* ♯ *x1a*› *flip-subst-subst subst-b-b-def subst-b-c-def*)
qed
qed

qed
end

lemma *subst-bb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-bb* (*subst-bb A i t*) *j u*) = *subst-bb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *b.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-vb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-vb* (*subst-vb A i t*)) *j u* = *subst-vb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *v.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-ceb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-ceb* (*subst-ceb A i t*)) *j u* = *subst-ceb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *ce.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-cb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-cb* (*subst-cb A i t*)) *j u* = *subst-cb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-tb-commute* [*simp*]:
  atom j ♯ *A* ⟹ (*subst-tb* (*subst-tb A i t*)) *j u* = *subst-tb A i* (*subst-bb t j u*)
proof (*nominal-induct A avoiding*: *i j t u rule*: *τ.strong-induct*)
  case (*T-refined-type z b c*)
  then show *?case* **using** *subst-tb.simps subst-bb-commute subst-cb-commute* **by** *simp*
qed

4.7 Expressions

nominal-function *subst-eb* :: *e* ⇒ *bv* ⇒ *b* ⇒ *e* **where**
  subst-eb ((*AE-val v′*)) *bv b* = (*AE-val* (*subst-vb v′ bv b*))
| *subst-eb* ((*AE-app f v′*)) *bv b* = ((*AE-app f* (*subst-vb v′ bv b*)))
| *subst-eb* ((*AE-appP f b′ v′*)) *bv b* = ((*AE-appP f* (*b′*[*bv*::=*b*]$_{bb}$) (*subst-vb v′ bv b*)))
| *subst-eb* ((*AE-op opp v1 v2*)) *bv b* = ((*AE-op opp* (*subst-vb v1 bv b*) (*subst-vb v2 bv b*)))
| *subst-eb* ((*AE-fst v′*)) *bv b* = *AE-fst* (*subst-vb v′ bv b*)
| *subst-eb* ((*AE-snd v′*)) *bv b* = *AE-snd* (*subst-vb v′ bv b*)
| *subst-eb* ((*AE-mvar u*)) *bv b* = *AE-mvar u*
| *subst-eb* ((*AE-len v′*)) *bv b* = *AE-len* (*subst-vb v′ bv b*)
| *subst-eb* (*AE-concat v1 v2*) *bv b* = *AE-concat* (*subst-vb v1 bv b*) (*subst-vb v2 bv b*)
| *subst-eb* (*AE-split v1 v2*) *bv b* = *AE-split* (*subst-vb v1 bv b*) (*subst-vb v2 bv b*)
          apply (*simp add*: *eqvt-def subst-eb-graph-aux-def*)
          apply *auto*
  by (*meson e.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
  subst-eb-abbrev :: *e* ⇒ *bv* ⇒ *b* ⇒ *e* (‹-[-::=-]$_{eb}$› [*1000*,*50*,*50*] *500*)
  where
    e[*bv*::=*b*]$_{eb}$ ≡ *subst-eb e bv b*

instantiation *e* :: *has-subst-b*
begin
definition *subst-b* = *subst-eb*

instance proof
  fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*e*
  show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
  proof (*induct t rule*: *e.induct*)
    case (*AE-val v*)
    then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp*
      e.fresh has-subst-b-class.fresh-subst-if subst-b-e-def subst-b-v-def
      by *metis*
  next
    case (*AE-app f v*)
    then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def*
      e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def
      by (*metis* (*mono-tags*, *opaque-lifting*) *e.fresh*(*2*))
  next
    case (*AE-appP f b′ v*)
    then show *?case* **unfolding** *subst-eb.simps subst-b-e-def e.fresh* **using**
      fresh-def pure-fresh subst-b-e-def e.supp v.supp

e.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-vb-def **by** (*metis subst-b-v-def*)
next
case (*AE-op opp v1 v2*)
then show *?case* **unfolding** *subst-eb.simps subst-b-e-def e.fresh* **using**
fresh-def pure-fresh subst-b-e-def e.supp v.supp fresh-e-opp-all
e.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-vb-def **by** (*metis subst-b-v-def*)
next
case (*AE-concat x1a x2*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp*
has-subst-b-class.fresh-subst-if subst-b-v-def
by (*metis subst-vb.simps*(*5*))
next
case (*AE-split x1a x2*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp*
has-subst-b-class.fresh-subst-if subst-b-v-def
by (*metis subst-vb.simps*(*5*))
next
case (*AE-fst x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def* **by** *metis*
next
case (*AE-snd x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp* **using** *has-subst-b-class.fresh-su*
subst-b-v-def **by** *metis*
next
case (*AE-mvar x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp* **by** *auto*
next
case (*AE-len x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp* **using**
has-subst-b-class.fresh-subst-if subst-b-v-def **by** *metis*
qed

fix *a*::*bv* **and** *tm*::*e* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
apply(*induct tm rule*: *e.induct*)
apply(*auto simp add*: *fresh-at-base subst-eb.simps subst-b-e-def*)
using *has-subst-b-class.fresh-subst-if subst-b-b-def e.fresh*
using *has-subst-b-class.forget-subst subst-b-v-def* **apply** *metis*+
done

fix *a*::*bv* **and** *tm*::*e*
show *subst-b tm a* (*B-var a*) = *tm* **using** *subst-bb.simps subst-b-b-def*
apply (*induct tm rule*: *e.induct*)
apply(*auto simp add*: *fresh-at-base subst-eb.simps subst-b-e-def*)
using *has-subst-b-class.fresh-subst-if subst-b-b-def e.fresh*
using *has-subst-b-class.subst-id subst-b-v-def* **apply** *metis*+
done

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*e*
show *p* · *subst-b t1 x1 v* = *subst-b* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
apply(*induct tm rule*: *e.induct*)

apply(*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)
using *has-subst-b-class.eqvt subst-b-b-def e.fresh*
using *has-subst-b-class.eqvt*
by (*simp add*: *subst-b-e-def*)+

fix *bv*::*bv* **and** *c*::*e* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*induct c rule*: *e.induct*)
apply(*auto simp add*: *fresh-at-base subst-eb.simps subst-b-e-def subst-b-v-def permute-pure pure-supp*)
using *flip-fresh-fresh permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def subst-b-b-def flip-fresh-fresh subst-b-τ-def* **apply** *metis*
apply (*metis* (*full-types*) *opp.perm-simps opp.strong-exhaust*)
done

fix *bv*::*bv* **and** *c*::*e* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
apply (*induct c rule*: *e.induct*)
apply(*auto simp add*: *fresh-at-base subst-eb.simps subst-b-e-def subst-b-v-def permute-pure pure-supp*)
using *flip-fresh-fresh permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def subst-b-b-def flip-fresh-fresh subst-b-τ-def* **apply** *simp*

apply (*metis* (*full-types*) *opp.perm-simps opp.strong-exhaust*)
done
qed
end

4.8 Statements

nominal-function (*default case-sum* (*λx. Inl undefined*) (*case-sum* (*λx. Inl undefined*) (*λx. Inr undefined*)))
subst-sb :: *s* ⇒ *bv* ⇒ *b* ⇒ *s*
and *subst-branchb* :: *branch-s* ⇒ *bv* ⇒ *b* ⇒ *branch-s*
and *subst-branchlb* :: *branch-list* ⇒ *bv* ⇒ *b* ⇒ *branch-list*
where
subst-sb (*AS-val v'*) *bv b* = (*AS-val* (*subst-vb v' bv b*))
| *subst-sb* (*AS-let y e s*) *bv b* = (*AS-let y* (*e*[*bv*::=*b*]$_{eb}$) (*subst-sb s bv b*))
| *subst-sb* (*AS-let2 y t s1 s2*) *bv b* = (*AS-let2 y* (*subst-tb t bv b*) (*subst-sb s1 bv b*) (*subst-sb s2 bv b*))
| *subst-sb* (*AS-match v' cs*) *bv b* = *AS-match* (*subst-vb v' bv b*) (*subst-branchlb cs bv b*)
| *subst-sb* (*AS-assign y v'*) *bv b* = *AS-assign y* (*subst-vb v' bv b*)
| *subst-sb* (*AS-if v' s1 s2*) *bv b* = (*AS-if* (*subst-vb v' bv b*) (*subst-sb s1 bv b*) (*subst-sb s2 bv b*))
| *subst-sb* (*AS-var u τ v' s*) *bv b* = *AS-var u* (*subst-tb τ bv b*) (*subst-vb v' bv b*) (*subst-sb s bv b*)
| *subst-sb* (*AS-while s1 s2*) *bv b* = *AS-while* (*subst-sb s1 bv b*) (*subst-sb s2 bv b*)
| *subst-sb* (*AS-seq s1 s2*) *bv b* = *AS-seq* (*subst-sb s1 bv b*) (*subst-sb s2 bv b*)
| *subst-sb* (*AS-assert c s*) *bv b* = *AS-assert* (*subst-cb c bv b*) (*subst-sb s bv b*)

| *subst-branchb* (*AS-branch dc x1 s'*) *bv b* = *AS-branch dc x1* (*subst-sb s' bv b*)

| *subst-branchlb* (*AS-final sb*) *bv b* = *AS-final* (*subst-branchb sb bv b*)
| *subst-branchlb* (*AS-cons sb ssb*) *bv b* = *AS-cons* (*subst-branchb sb bv b*) (*subst-branchlb ssb bv b*)

apply (*simp add*: *eqvt-def subst-sb-subst-branchb-subst-branchlb-graph-aux-def*)

apply (*auto*,*metis s-branch-s-branch-list.exhaust s-branch-s-branch-list.exhaust*(*2*) *old.sum.exhaust surj-pair*)

proof(*goal-cases*)

have *eqvt-at-proj*: $\bigwedge$ *s xa va* . *eqvt-at subst-sb-subst-branchb-subst-branchlb-sumC* (*Inl* (*s*, *xa*, *va*)) $\Longrightarrow$

eqvt-at (*λa*. *projl* (*subst-sb-subst-branchb-subst-branchlb-sumC* (*Inl a*))) (*s*, *xa*, *va*)
apply(*simp only*: *eqvt-at-def*)
apply(*rule*)
apply(*subst Projl-permute*)
apply(*thin-tac* -)+
apply(*simp add*: *subst-sb-subst-branchb-subst-branchlb-sumC-def*)
apply(*simp add*: *THE-default-def*)
apply(*case-tac Ex1* (*subst-sb-subst-branchb-subst-branchlb-graph* (*Inl* (*s*,*xa*,*va*))))
apply *simp*
apply(*auto*)[*1*]
apply(*erule-tac x*=*x* **in** *allE*)
apply *simp*
apply(*cases rule*: *subst-sb-subst-branchb-subst-branchlb-graph.cases*)
apply(*assumption*)
apply(*rule-tac x*=*Sum-Type.projl x* **in** *exI*,*clarify*,*rule the1-equality*,*blast*,*simp* (*no-asm*) *only*: *sum.sel*)+
apply(*blast*)+
apply(*simp*)+
done
{
case (*1 y s bv b ya sa c*)
moreover have *atom y* ♯ (*bv*, *b*) ∧ *atom ya* ♯ (*bv*, *b*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*

ultimately show *?case*
using *eqvt-triple eqvt-at-proj* **by** *metis*
next
case (*2 y s1 s2 bv b ya s2a c*)
moreover have *atom y* ♯ (*bv*, *b*) ∧ *atom ya* ♯ (*bv*, *b*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
ultimately show *?case*
using *eqvt-triple eqvt-at-proj* **by** *metis*
next
case (*3 u s bv b ua sa c*)
moreover have *atom u* ♯ (*bv*, *b*) ∧ *atom ua* ♯ (*bv*, *b*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
ultimately show *?case* **using** *eqvt-triple eqvt-at-proj* **by** *metis*
next
case (*4 x1 s′ bv b x1a s′a c*)
moreover have *atom x1* ♯ (*bv*, *b*) ∧ *atom x1a* ♯ (*bv*, *b*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
ultimately show *?case* **using** *eqvt-triple eqvt-at-proj* **by** *metis*
}
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-sb-abbrev :: *s* ⇒ *bv* ⇒ *b* ⇒ *s* (‹-[-::=-]$_{sb}$› [*1000*,*50*,*50*] *1000*)
where
b[*bv*::=*b'*]$_{sb}$ ≡ *subst-sb b bv b'*

lemma *fresh-subst-sb-if* [*simp*]:
(*j* ♯ (*subst-sb A i x*)) = ((*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*))) **and**
(*j* ♯ (*subst-branchb B i x*)) = ((*atom i* ♯ *B* ∧ *j* ♯ *B*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *B* ∨ *j* = *atom i*))) **and**
(*j* ♯ (*subst-branchlb C i x*)) = ((*atom i* ♯ *C* ∧ *j* ♯ *C*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *C* ∨ *j* = *atom i*)))
proof (*nominal-induct A* **and** *B* **and** *C avoiding*: *i x rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-branch x1 x2 x3*)
have (*j* ♯ *subst-branchb* (*AS-branch x1 x2 x3*) *i x*) = (*j* ♯ (*AS-branch x1 x2* (*subst-sb x3 i x*))) **by** *auto*
also have ... = ((*j* ♯ *x3*[*i*::=*x*]$_{sb}$ ∨ *j* ∈ *set* [*atom x2*]) ∧ *j* ♯ *x1*) **using** *s-branch-s-branch-list.fresh* **by** *auto*
also have ... = ((*atom i* ♯ *AS-branch x1 x2 x3* ∧ *j* ♯ *AS-branch x1 x2 x3*) ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-branch x1 x2 x3* ∨ *j* = *atom i*))
using *subst-branchb.simps*(*1*) *s-branch-s-branch-list.fresh*(*1*) *fresh-at-base has-subst-b-class.fresh-subst-if list.distinct list.set-cases set-ConsD subst-b-τ-def*
v.fresh AS-branch
proof −
have *f1*: ∀ *cs b. atom* (*b*::*bv*) ♯ (*cs*::*char list*) **using** *pure-fresh* **by** *auto*

then have *j* ♯ *x* ∧ *atom i* = *j* ⟶ ((*j* ♯ *x3*[*i*::=*x*]$_{sb}$ ∨ *j* ∈ *set* [*atom x2*]) ∧ *j* ♯ *x1*) = (*atom i* ♯ *AS-branch x1 x2 x3* ∧ *j* ♯ *AS-branch x1 x2 x3* ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-branch x1 x2 x3* ∨ *j* = *atom i*))
by (*metis* (*full-types*) *AS-branch.hyps*(*3*))
then have *j* ♯ *x* ⟶ ((*j* ♯ *x3*[*i*::=*x*]$_{sb}$ ∨ *j* ∈ *set* [*atom x2*]) ∧ *j* ♯ *x1*) = (*atom i* ♯ *AS-branch x1 x2 x3* ∧ *j* ♯ *AS-branch x1 x2 x3* ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-branch x1 x2 x3* ∨ *j* = *atom i*))
using *AS-branch.hyps s-branch-s-branch-list.fresh* **by** *metis*
moreover
{ assume ¬ *j* ♯ *x*
have *?thesis*
using *f1 AS-branch.hyps*(*2*) *AS-branch.hyps*(*3*) **by** *force* **}**
ultimately show *?thesis*
by *satx*
qed
finally show *?case* **by** *auto*
next
case (*AS-cons cs css i x*)
show *?case*
unfolding *subst-branchlb.simps s-branch-s-branch-list.fresh*
using *AS-cons* **by** *auto*
next
case (*AS-val xx*)
then show *?case* **using** *subst-sb.simps*(*1*) *s-branch-s-branch-list.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-b-v-def* **by** *metis*
next
case (*AS-let x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh fresh-at-base has-subst-b-class.fresh-subst-if*

list.distinct list.set-cases set-ConsD subst-b-e-def
by *fastforce*
next
case (*AS-let2 x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh fresh-at-base has-subst-b-class.fresh-subst-if list.distinct list.set-cases set-ConsD subst-b-τ-def*
by *fastforce*
next
case (*AS-if x1 x2 x3*)
then show *?case* **unfolding** *subst-sb.simps s-branch-s-branch-list.fresh* **using** *has-subst-b-class.fresh-subst-if subst-b-v-def* **by** *metis*
next
case (*AS-var u t v s*)

have $(((atom\ i \sharp s \wedge j \sharp s \vee j \sharp x \wedge (j \sharp s \vee j = atom\ i)) \vee j \in set\ [atom\ u]) \wedge j \sharp t[i::=x]_{\tau b} \wedge j \sharp v[i::=x]_{vb}) =$
$(((atom\ i \sharp s \wedge j \sharp s \vee j \sharp x \wedge (j \sharp s \vee j = atom\ i)) \vee j \in set\ [atom\ u]) \wedge$
$((atom\ i \sharp t \wedge j \sharp t \vee j \sharp x \wedge (j \sharp t \vee j = atom\ i))) \wedge$
$((atom\ i \sharp v \wedge j \sharp v \vee j \sharp x \wedge (j \sharp v \vee j = atom\ i))))$
using *has-subst-b-class.fresh-subst-if subst-b-v-def subst-b-τ-def* **by** *metis*
also have ... = $(((atom\ i \sharp s \vee atom\ i \in set\ [atom\ u]) \wedge atom\ i \sharp t \wedge atom\ i \sharp v) \wedge$
$(j \sharp s \vee j \in set\ [atom\ u]) \wedge j \sharp t \wedge j \sharp v \vee j \sharp x \wedge ((j \sharp s \vee j \in set\ [atom\ u]) \wedge j \sharp t \wedge j \sharp v \vee j = atom\ i))$
using *u-fresh-b* **by** *auto*
finally show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh AS-var*
by *simp*
next
case (*AS-assign u v*)
then show *?case* **unfolding** *subst-sb.simps s-branch-s-branch-list.fresh* **using** *has-subst-b-class.fresh-subst-if subst-b-v-def* **by** *force*
next
case (*AS-match v cs*)
have $j \sharp (AS\text{-}match\ v\ cs)[i::=x]_{sb} = j \sharp (AS\text{-}match\ (subst\text{-}vb\ v\ i\ x)\ (subst\text{-}branchlb\ cs\ i\ x\ ))$ **using** *subst-sb.simps* **by** *auto*
also have ... = $(j \sharp (subst\text{-}vb\ v\ i\ x) \wedge j \sharp (subst\text{-}branchlb\ cs\ i\ x\ ))$ **using** *s-branch-s-branch-list.fresh* **by** *simp*
also have ... = $(j \sharp (subst\text{-}vb\ v\ i\ x) \wedge ((atom\ i \sharp cs \wedge j \sharp cs) \vee j \sharp x \wedge (j \sharp cs \vee j = atom\ i)))$ **using** *AS-match[of i x]* **by** *auto*
also have ... = $(atom\ i \sharp AS\text{-}match\ v\ cs \wedge j \sharp AS\text{-}match\ v\ cs \vee j \sharp x \wedge (j \sharp AS\text{-}match\ v\ cs \vee j = atom\ i))$
by (*metis* (*no-types*) *s-branch-s-branch-list.fresh has-subst-b-class.fresh-subst-if subst-b-v-def*)
finally show *?case* **by** *auto*
next
case (*AS-while x1 x2*)
then show *?case* **by** *auto*
next
case (*AS-seq x1 x2*)
then show *?case* **by** *auto*
next
case (*AS-assert x1 x2*)
then show *?case* **unfolding** *subst-sb.simps s-branch-s-branch-list.fresh*
using *fresh-at-base has-subst-b-class.fresh-subst-if list.distinct list.set-cases set-ConsD subst-b-e-def*

by (*metis subst-b-c-def*)
qed(*auto+*)

lemma
forget-subst-sb[*simp*]: *atom a* ♯ *A* ⟹ *subst-sb A a x* = *A* **and**
forget-subst-branchb [*simp*]: *atom a* ♯ *B* ⟹ *subst-branchb B a x* = *B* **and**
forget-subst-branchlb[*simp*]: *atom a* ♯ *C* ⟹ *subst-branchlb C a x* = *C*
proof (*nominal-induct A* **and** *B* **and** *C avoiding*: *a x rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-let x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-let2 x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-τ-def* **by** *force*
next
case (*AS-var x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **using** *subst-b-τ-def*
proof −
have *f1*: (*atom a* ♯ *x4* ∨ *atom a* ∈ *set* [*atom x1*]) ∧ *atom a* ♯ *x2* ∧ *atom a* ♯ *x3*
using *AS-var.prems s-branch-s-branch-list.fresh* **by** *simp*
then have *atom a* ♯ *x4*
by (*metis* (*no-types*) *Nominal−Utils.fresh-star-singleton AS-var.hyps*(*1*) *empty-set fresh-star-def list.simps*(*15*) *not-self-fresh*)
then show *?thesis*
using *f1* **by** (*metis AS-var.hyps*(*3*) *has-subst-b-class.forget-subst subst-b-τ-def subst-b-v-def subst-sb.simps*(*7*))

qed
next
case (*AS-branch x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-cons x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-val x*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-if x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-assign x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-match x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst*

subst-b-v-def **by** *force*
next
  case (*AS-while x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
  case (*AS-seq x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
  case (*AS-assert c s*)
  then show *?case* **unfolding** *subst-sb.simps* **using**
    s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def subst-b-c-def subst-cb.simps **by** *force*
qed(*auto+*)

lemma *subst-sb-id*: *subst-sb A a* (*B-var a*) = *A* **and**
  subst-branchb-id [*simp*]: *subst-branchb B a* (*B-var a*) = *B* **and**
  subst-branchlb-id: *subst-branchlb C a* (*B-var a*) = *C*
proof(*nominal-induct A* **and** *B* **and** *C avoiding*: *a rule*: *s-branch-s-branch-list.strong-induct*)
  case (*AS-branch x1 x2 x3*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def*
   by *simp*
next
  case (*AS-cons x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *simp*
next
  case (*AS-val x*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
  case (*AS-if x1 x2 x3*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
  case (*AS-assign x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
  case (*AS-match x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
  case (*AS-while x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
  case (*AS-seq x1 x2*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*

next
  case (*AS-let x1 x2 x3*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.subst-id*
by *metis*
next
  case (*AS-let2 x1 x2 x3 x4*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id*
by *metis*
next
  case (*AS-var x1 x2 x3 x4*)
 then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
  case (*AS-assert c s*)
 then show *?case* **unfolding** *subst-sb.simps* **using** *s-branch-s-branch-list.fresh subst-b-c-def has-subst-b-class.subst-id*
by *metis*
qed (*auto*)

lemma *flip-subst-s*:
  fixes *bv*::*bv* **and** *s*::*s* **and** *cs*::*branch-s* **and** *z*::*bv*
  shows $atom\ bv \sharp s \Longrightarrow ((bv \leftrightarrow z) \cdot s) = s[z::=B\text{-}var\ bv]_{sb}$ **and**
    $atom\ bv \sharp cs \Longrightarrow ((bv \leftrightarrow z) \cdot cs) = subst\text{-}branchb\ \ cs\ z\ (B\text{-}var\ bv)$ **and**
    $atom\ bv \sharp css \Longrightarrow ((bv \leftrightarrow z) \cdot css) = subst\text{-}branchlb\ css\ z\ (B\text{-}var\ bv)$

proof(*nominal-induct s* **and** *cs* **and** *css* *rule*: *s-branch-s-branch-list.strong-induct*)
  case (*AS-branch x1 x2 x3*)
  hence $((bv \leftrightarrow z) \cdot x1) = x1$ **using** *pure-fresh fresh-at-base flip-fresh-fresh* **by** *metis*
  moreover have $((bv \leftrightarrow z) \cdot x2) = x2$ **using** *fresh-at-base flip-fresh-fresh*[*of bv x2 z*] *AS-branch* **by** *auto*
   ultimately show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *s-branch-s-branch-list.fresh*(*1*) *AS-branch* **by** *auto*
next
  case (*AS-cons x1 x2*)
  hence $((bv \leftrightarrow z) \cdot x1) =$ *subst-branchb x1 z* (*B-var bv*) **using** *pure-fresh fresh-at-base flip-fresh-fresh s-branch-s-branch-list.fresh*(*13*) **by** *metis*
 moreover have $((bv \leftrightarrow z) \cdot x2) =$ *subst-branchlb x2 z* (*B-var bv*) **using** *fresh-at-base flip-fresh-fresh*[*of bv x2 z*] *AS-cons s-branch-s-branch-list.fresh* **by** *metis*
   ultimately show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *s-branch-s-branch-list.fresh*(*1*) *AS-cons* **by** *auto*
next
  case (*AS-val x*)
  then show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *flip-subst subst-b-v-def* **by** *simp*
next
  case (*AS-let x1 x2 x3*)
  moreover hence $((bv \leftrightarrow z) \cdot x1) = x1$ **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
  ultimately show *?case*
    unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
    using *flip-subst subst-b-e-def s-branch-s-branch-list.fresh* **by** *auto*
next
  case (*AS-let2 x1 x2 x3 x4*)
  moreover hence $((bv \leftrightarrow z) \cdot x1) = x1$ **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*

ultimately show *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst s-branch-s-branch-list.fresh*(*5*) *subst-b-τ-def* **by** *auto*
next
case (*AS-if x1 x2 x3*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-var x1 x2 x3 x4*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def subst-b-τ-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-assign x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-match x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-while x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-seq x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-assert x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-c-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *simp*
qed(*auto*)

lemma *flip-subst-subst-s*:
fixes *bv*::*bv* **and** *s*::*s* **and** *cs*::*branch-s* **and** *z*::*bv*
shows $atom\ bv \sharp s \Longrightarrow ((bv \leftrightarrow z) \cdot s)[bv{::=}v]_{sb} = s[z{::=}v]_{sb}$ **and**
$atom\ bv \sharp cs \Longrightarrow subst\text{-}branchb\ ((bv \leftrightarrow z) \cdot cs)\ bv\ v = subst\text{-}branchb\ cs\ z\ v$ **and**
$atom\ bv \sharp css \Longrightarrow subst\text{-}branchlb\ ((bv \leftrightarrow z) \cdot css)\ bv\ v = subst\text{-}branchlb\ css\ z\ v$
proof(*nominal-induct s* **and** *cs* **and** *css rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-branch x1 x2 x3*)
hence $((bv \leftrightarrow z) \cdot x1) = x1$ **using** *pure-fresh fresh-at-base flip-fresh-fresh* **by** *metis*
moreover have $((bv \leftrightarrow z) \cdot x2) = x2$ **using** *fresh-at-base flip-fresh-fresh*[*of bv x2 z*] *AS-branch* **by** *auto*

ultimately show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *s-branch-s-branch-list.fresh*(*1*) *AS-branch* **by** *auto*
next
case (*AS-cons x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-branchlb.simps*
using *s-branch-s-branch-list.fresh*(*1*) *AS-cons* **by** *auto*

next
case (*AS-val x*)
then show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *flip-subst subst-b-v-def* **by** *simp*
next
case (*AS-let x1 x2 x3*)
moreover hence $((bv \leftrightarrow z) \cdot x1) = x1$ **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
ultimately show *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst-subst subst-b-e-def s-branch-s-branch-list.fresh* **by** *force*
next
case (*AS-let2 x1 x2 x3 x4*)
moreover hence $((bv \leftrightarrow z) \cdot x1) = x1$ **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
ultimately show *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst s-branch-s-branch-list.fresh*(*5*) *subst-b-τ-def* **by** *auto*
next
case (*AS-if x1 x2 x3*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-var x1 x2 x3 x4*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def subst-b-τ-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-assign x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-match x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-while x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next

case (*AS-seq x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-assert x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-c-def s-branch-s-branch-list.fresh* **by** *auto*
qed(*auto*)

instantiation *s* :: *has-subst-b*
begin
definition $subst\text{-}b = (\lambda s\ bv\ b.\ subst\text{-}sb\ s\ bv\ b)$

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*s*
show $j \sharp subst\text{-}b\ t\ i\ x = ((atom\ i \sharp t \wedge j \sharp t) \vee (j \sharp x \wedge (j \sharp t \vee j = atom\ i)))$
using *fresh-subst-sb-if subst-b-s-def* **by** *metis*

fix *a*::*bv* **and** *tm*::*s* **and** *x*::*b*
show $atom\ a \sharp tm \Longrightarrow subst\text{-}b\ tm\ a\ x = tm$ **using** *subst-b-s-def forget-subst-sb* **by** *metis*

fix *a*::*bv* **and** *tm*::*s*
show $subst\text{-}b\ tm\ a\ (B\text{-}var\ a) = tm$ **using** *subst-b-s-def subst-sb-id* **by** *metis*

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*s*
show $p \cdot subst\text{-}b\ t1\ x1\ v = subst\text{-}b\ (p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$ **using** *subst-b-s-def subst-sb-subst-branchb-subst-branchlb.eqvt*
by *metis*

fix *bv*::*bv* **and** *c*::*s* **and** *z*::*bv*
show $atom\ bv \sharp c \Longrightarrow ((bv \leftrightarrow z) \cdot c) = c[z::=B\text{-}var\ bv]_b$
using *subst-b-s-def flip-subst-s* **by** *metis*

fix *bv*::*bv* **and** *c*::*s* **and** *z*::*bv* **and** *v*::*b*
show $atom\ bv \sharp c \Longrightarrow ((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$
using *flip-subst-subst-s subst-b-s-def* **by** *metis*
qed
end

4.9 Function Type

nominal-function *subst-ft-b* :: *fun-typ* ⇒ *bv* ⇒ *b* ⇒ *fun-typ* **where**
$subst\text{-}ft\text{-}b\ (\ AF\text{-}fun\text{-}typ\ z\ b\ c\ t\ (s::s))\ x\ v = AF\text{-}fun\text{-}typ\ z\ (subst\text{-}bb\ b\ x\ v)\ (subst\text{-}cb\ c\ x\ v)\ t[x::=v]_{\tau b}$
$s[x::=v]_{sb}$
apply(*simp add*: *eqvt-def subst-ft-b-graph-aux-def*)
apply(*simp add*:*fun-typ.strong-exhaust*,*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*a*,*b*) **in** *fun-typ.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
done

nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *subst-ftq-b* :: *fun-typ-q* ⇒ *bv* ⇒ *b* ⇒ *fun-typ-q* **where**
atom bv ♯ (*x*,*v*) ⟹ *subst-ftq-b* (*AF-fun-typ-some bv ft*) *x v* = (*AF-fun-typ-some bv* (*subst-ft-b ft x v*))

| *subst-ftq-b* (*AF-fun-typ-none ft*) *x v* = (*AF-fun-typ-none* (*subst-ft-b ft x v*))
apply(*simp add*: *eqvt-def subst-ftq-b-graph-aux-def*)
apply(*simp add*:*fun-typ-q.strong-exhaust*,*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ-q.strong-exhaust*)
by (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

instantiation *fun-typ* :: *has-subst-b*
begin
definition *subst-b* = *subst-ft-b*

Note: Using just simp in the second apply unpacks and gives a single goal whereas auto gives 43 non-intuitive goals. These goals are easier to solve and tedious, however they that it clear if a mistake is made in the definition of the function. For example, I saw that one of the goals was going through with metis and the next wasn't. It turned out the definition of the function itself was wrong

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*fun-typ*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-def*)
by(*metis fresh-subst-if subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def*)+

fix *a*::*bv* **and** *tm*::*fun-typ* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
apply (*nominal-induct tm avoiding*: *a x rule*: *fun-typ.strong-induct*)
apply(*simp add*: *subst-b-fun-typ-def Abs1-eq-iff′*)
using *subst-b-b-def subst-b-fun-typ-def subst-b-τ-def subst-b-c-def subst-b-s-def*
forget-subst fresh-at-base list.set-cases neq-Nil-conv set-ConsD
subst-ft-b.simps **by** *metis*

fix *a*::*bv* **and** *tm*::*fun-typ*
show *subst-b tm a* (*B-var a*) = *tm*
apply (*nominal-induct tm rule*: *fun-typ.strong-induct*)
apply(*simp add*: *subst-b-fun-typ-def Abs1-eq-iff′*,*auto*)
using *subst-b-b-def subst-b-fun-typ-def subst-b-τ-def subst-b-c-def subst-b-s-def*
forget-subst fresh-at-base list.set-cases neq-Nil-conv set-ConsD
subst-ft-b.simps
by (*metis has-subst-b-class.subst-id*)+

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*fun-typ*
show *p* · *subst-b t1 x1 v* = *subst-b* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
apply (*nominal-induct t1 avoiding*: *x1 v rule*: *fun-typ.strong-induct*)
by(*auto simp add*: *subst-b-fun-typ-def Abs1-eq-iff′ fun-typ.perm-simps*)

fix *bv*::*bv* **and** *c*::*fun-typ* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*nominal-induct c avoiding*: *z bv rule*: *fun-typ.strong-induct*)

by(*auto simp add*: *subst-b-fun-typ-def Abs1-eq-iff′ fun-typ.perm-simps subst-b-b-def subst-b-c-def subst-b-τ-def subst-b-s-def*)

fix *bv*::*bv* **and** *c*::*fun-typ* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
apply (*nominal-induct c avoiding*: *bv v z rule*: *fun-typ.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-def Abs1-eq-iff′ fun-typ.perm-simps subst-b-b-def subst-b-c-def subst-b-τ-def subst-b-s-def flip-subst-subst flip-subst*)
using *subst-b-fun-typ-def Abs1-eq-iff′ fun-typ.perm-simps subst-b-b-def subst-b-c-def subst-b-τ-def subst-b-s-def flip-subst-subst flip-subst*
using *flip-subst-s*(*1*) *flip-subst-subst-s*(*1*) **by** *auto*
qed
end

instantiation *fun-typ-q* :: *has-subst-b*
begin
definition *subst-b* = *subst-ftq-b*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*fun-typ-q*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
apply (*nominal-induct t avoiding*: *i x j rule*: *fun-typ-q.strong-induct*,*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps*)
using *fresh-subst-if subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def*
apply *metis+*
done

fix *a*::*bv* **and** *t*::*fun-typ-q* **and** *x*::*b*
show *atom a* ♯ *t* ⟹ *subst-b t a x* = *t*
apply (*nominal-induct t avoiding*: *a x rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff′*)
using *forget-subst subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def eqvt* **by** *metis+*

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t*::*fun-typ-q*
show *p* · *subst-b t x1 v* = *subst-b* (*p* · *t*) (*p* · *x1*) (*p* · *v*)
apply (*nominal-induct t avoiding*: *x1 v rule*: *fun-typ-q.strong-induct*)
by(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff′*)

fix *a*::*bv* **and** *tm*::*fun-typ-q*
show *subst-b tm a* (*B-var a*) = *tm*
apply (*nominal-induct tm avoiding*: *a rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff′*)
using *subst-id subst-b-b-def subst-b-fun-typ-def subst-b-τ-def subst-b-c-def subst-b-s-def forget-subst fresh-at-base list.set-cases neq-Nil-conv set-ConsD subst-ft-b.simps* **by** *metis+*

fix *bv*::*bv* **and** *c*::*fun-typ-q* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*nominal-induct c avoiding*: *z bv rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff′*)
using *forget-subst subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def*

eqvt **by** *metis+*

fix *bv*::*bv* **and** *c*::*fun-typ-q* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ $((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$
apply (*nominal-induct c avoiding*: *z v bv rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff'*)
using *flip-subst flip-subst-subst forget-subst subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def eqvt* **by** *metis+*

qed
end

4.10 Contexts

4.10.1 Immutable Variables

nominal-function *subst-gb* :: Γ ⇒ *bv* ⇒ *b* ⇒ Γ **where**
subst-gb GNil - - = *GNil*
| *subst-gb* $((y,b',c)\#_\Gamma\Gamma)$ *bv b* = $((y,b'[bv::=b]_{bb},c[bv::=b]_{cb})\#_\Gamma$ (*subst-gb* Γ *bv b*))
apply (*simp add*: *eqvt-def subst-gb-graph-aux-def*)+
apply *auto*
apply (*insert* Γ.*exhaust neq-GNil-conv*, *force*)
done
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-gb-abbrev :: Γ ⇒ *bv* ⇒ *b* ⇒ Γ (‹-[-::=-]$_{\Gamma b}$› [*1000*,*50*,*50*] *1000*)
where
$g[bv::=b']_{\Gamma b}$ ≡ *subst-gb g bv b'*

instantiation Γ :: *has-subst-b*
begin
definition *subst-b* = *subst-gb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::Γ
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof(*induct t rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *fresh-GNil subst-gb.simps fresh-def pure-fresh subst-b-Γ-def has-subst-b-class.fresh-subst-if fresh-GNil fresh-GCons* **by** *metis*
next
case (*GCons x' b' c'* Γ')
have ∗: *atom i* ♯ *x'* **using** *fresh-at-base* **by** *simp*

have *j* ♯ *subst-b* ((*x'*, *b'*, *c'*) $\#_\Gamma$ Γ') *i x* = *j* ♯ ((*x'*, $b'[i::=x]_{bb}$, $c'[i::=x]_{cb}$) $\#_\Gamma$ (*subst-b* Γ' *i x*)) **using** *subst-gb.simps subst-b-Γ-def* **by** *auto*
also have ... = (*j* ♯ ((*x'*, $b'[i::=x]_{bb}$, $c'[i::=x]_{cb}$)) ∧ (*j* ♯ (*subst-b* Γ' *i x*))) **using** *fresh-GCons* **by** *auto*
also have ... = (((*j* ♯ *x'*) ∧ (*j* ♯ $b'[i::=x]_{bb}$) ∧ (*j* ♯ $c'[i::=x]_{cb}$)) ∧ (*j* ♯ (*subst-b* Γ' *i x*))) **by** *auto*
also have ... = (((*j* ♯ *x'*) ∧ ((*atom i* ♯ *b'* ∧ *j* ♯ *b'* ∨ *j* ♯ *x* ∧ (*j* ♯ *b'* ∨ *j* = *atom i*))) ∧
((*atom i* ♯ *c'* ∧ *j* ♯ *c'* ∨ *j* ♯ *x* ∧ (*j* ♯ *c'* ∨ *j* = *atom i*))) ∧

$((atom\ i \sharp \Gamma' \wedge j \sharp \Gamma' \vee j \sharp x \wedge (j \sharp \Gamma' \vee j = atom\ i)))))$
using *fresh-subst-if*[*of j b' i x*] *fresh-subst-if*[*of j c' i x*] *GCons subst-b-b-def subst-b-c-def* **by** *simp*
also have ... $= ((atom\ i \sharp (x', b', c') \#_\Gamma \Gamma' \wedge j \sharp (x', b', c') \#_\Gamma \Gamma') \vee (j \sharp x \wedge (j \sharp (x', b', c') \#_\Gamma \Gamma' \vee j = atom\ i)))$ **using** $*$ *fresh-GCons fresh-prod3* **by** *metis*

finally show *?case* **by** *auto*
qed

fix $a::bv$ **and** $tm::\Gamma$ **and** $x::b$
show $atom\ a \sharp tm \Longrightarrow subst\text{-}b\ tm\ a\ x = tm$
proof (*induct tm rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gb.simps subst-b-*Γ*-def* **by** *auto*
next
case $(GCons\ x'\ b'\ c'\ \Gamma')$
have $*{:}b'[a::=x]_{bb} = b' \wedge c'[a::=x]_{cb} = c'$ **using** *GCons fresh-GCons*[*of atom a*] *fresh-prod3*[*of atom a*] *has-subst-b-class.forget-subst subst-b-b-def subst-b-c-def* **by** *metis*
have $subst\text{-}b\ ((x', b', c') \#_\Gamma \Gamma')\ a\ x = ((x', b'[a::=x]_{bb}, c'[a::=x]_{cb}) \#_\Gamma (subst\text{-}b\ \Gamma'\ a\ x))$ **using** *subst-b-*Γ*-def subst-gb.simps* **by** *auto*
also have ... $= ((x', b', c') \#_\Gamma \Gamma')$ **using** $*$ *GCons fresh-GCons*[*of atom a*] **by** *auto*
finally show *?case* **using** *has-subst-b-class.forget-subst fresh-GCons fresh-prod3 GCons subst-b-*Γ*-def has-subst-b-class.forget-subst*[*of a b' x*] *fresh-prod3*[*of atom a*] **by** *argo*
qed

fix $a::bv$ **and** $tm::\Gamma$
show $subst\text{-}b\ tm\ a\ (B\text{-}var\ a) = tm$
proof(*induct tm rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gb.simps subst-b-*Γ*-def* **by** *auto*
next
case $(GCons\ x'\ b'\ c'\ \Gamma')$
then show *?case* **using** *has-subst-b-class.subst-id subst-b-*Γ*-def subst-b-b-def subst-b-c-def subst-gb.simps* **by** *metis*
qed

fix $p::perm$ **and** $x1::bv$ **and** $v::b$ **and** $t1::\Gamma$
show $p \cdot subst\text{-}b\ t1\ x1\ v = subst\text{-}b\ (p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$
proof (*induct tm rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-b-*Γ*-def subst-gb.simps* **by** *simp*
next
case $(GCons\ x'\ b'\ c'\ \Gamma')$
then show *?case* **using** *subst-b-*Γ*-def subst-gb.simps has-subst-b-class.eqvt* **by** *argo*
qed

fix $bv::bv$ **and** $c::\Gamma$ **and** $z::bv$
show $atom\ bv \sharp c \Longrightarrow ((bv \leftrightarrow z) \cdot c) = c[z::=B\text{-}var\ bv]_b$
proof (*induct c rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-b-*Γ*-def subst-gb.simps* **by** *auto*
next
case $(GCons\ x\ b\ c\ \Gamma')$

have $*$:$(bv \leftrightarrow z) \cdot (x, b, c) = (x, (bv \leftrightarrow z) \cdot b, (bv \leftrightarrow z) \cdot c)$ **using** *flip-bv-x-cancel* **by** *auto*
then show *?case*
unfolding *subst-gb.simps subst-b-Γ-def permute-Γ.simps* $*$
using *GCons subst-b-Γ-def subst-gb.simps flip-subst subst-b-b-def subst-b-c-def fresh-GCons* **by** *auto*
qed

fix $bv::bv$ **and** $c::\Gamma$ **and** $z::bv$ **and** $v::b$
show *atom bv* $\sharp$ $c \Longrightarrow ((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$
proof (*induct c rule*: Γ*-induct*)
case *GNil*
then show *?case* **using** *subst-b-Γ-def subst-gb.simps* **by** *auto*
next
case (*GCons x b c* Γ′)
have $*$:$(bv \leftrightarrow z) \cdot (x, b, c) = (x, (bv \leftrightarrow z) \cdot b, (bv \leftrightarrow z) \cdot c)$ **using** *flip-bv-x-cancel* **by** *auto*
then show *?case*
unfolding *subst-gb.simps subst-b-Γ-def permute-Γ.simps* $*$
using *GCons subst-b-Γ-def subst-gb.simps flip-subst subst-b-b-def subst-b-c-def fresh-GCons* **by** *auto*
qed
qed
end

lemma *subst-b-base-for-lit*:
$(base\text{-}for\text{-}lit\ l)[bv::=b]_{bb} = base\text{-}for\text{-}lit\ l$
using *base-for-lit.simps l.strong-exhaust*
by (*metis subst-bb.simps*(*2*) *subst-bb.simps*(*3*) *subst-bb.simps*(*6*) *subst-bb.simps*(*7*))

lemma *subst-b-lookup*:
assumes *Some* $(b, c) = lookup\ \Gamma\ x$
shows *Some* $(b[bv::=b']_{bb}, c[bv::=b']_{cb}) = lookup\ \Gamma[bv::=b']_{\Gamma b}\ x$
using *assms* **by**(*induct* Γ *rule*: Γ*-induct*, *auto*)

lemma *subst-g-b-x-fresh*:
fixes $x::x$ **and** $b::b$ **and** $\Gamma::\Gamma$ **and** $bv::bv$
assumes *atom x* $\sharp$ Γ
shows *atom x* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$
using *subst-b-fresh-x subst-b-Γ-def assms* **by** *metis*

4.10.2 Mutable Variables

nominal-function *subst-db* :: $\Delta \Rightarrow bv \Rightarrow b \Rightarrow \Delta$ **where**
subst-db $[]_\Delta$ - - $= []_\Delta$
| *subst-db* $((u,t) \#_\Delta \Delta)\ bv\ b = ((u,t[bv::=b]_{\tau b}) \#_\Delta (subst\text{-}db\ \Delta\ bv\ b))$
apply (*simp add*: *eqvt-def subst-db-graph-aux-def*,*auto*)
using *list.exhaust delete-aux.elims*
using *neq-DNil-conv* **by** *fastforce*
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-db-abbrev :: $\Delta \Rightarrow bv \Rightarrow b \Rightarrow \Delta$ (‹-[-::=-]$_{\Delta b}$› [*1000*,*50*,*50*] *1000*)
where
$\Delta[bv::=b]_{\Delta b} \equiv subst\text{-}db\ \Delta\ bv\ b$

instantiation Δ :: *has-subst-b*

begin
definition *subst-b* = *subst-db*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::Δ
show $j \sharp$ *subst-b t i x* = (*atom i* $\sharp$ *t* ∧ *j* $\sharp$ *t* ∨ *j* $\sharp$ *x* ∧ (*j* $\sharp$ *t* ∨ *j* = *atom i*))
proof(*induct t rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *fresh-DNil subst-db.simps fresh-def pure-fresh subst-b-Δ-def has-subst-b-class.fresh-subst-if fresh-DNil fresh-DCons* **by** *metis*
next
case (*DCons u t* Γ′)
have *j* ♯ *subst-b* ((*u*, *t*) $\#_\Delta$ Γ′) *i x* = *j* ♯ ((*u*, $t[i{::=}x]_{\tau b}$) $\#_\Delta$ (*subst-b* Γ′ *i x*)) **using** *subst-db.simps subst-b-Δ-def* **by** *auto*
also have ... = (*j* ♯ ((*u*, $t[i{::=}x]_{\tau b}$)) ∧ (*j* ♯ (*subst-b* Γ′ *i x*))) **using** *fresh-DCons* **by** *auto*
also have ... = (((*j* ♯ *u*) ∧ (*j* ♯ $t[i{::=}x]_{\tau b}$)) ∧ (*j* ♯ (*subst-b* Γ′ *i x*))) **by** *auto*
also have ... = ((*j* ♯ *u*) ∧ ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))) ∧ (*atom i* ♯ Γ′ ∧ *j* ♯ Γ′ ∨ *j* ♯ *x* ∧ (*j* ♯ Γ′ ∨ *j* = *atom i*)))
using *has-subst-b-class.fresh-subst-if*[*of j t i x*] *subst-b-τ-def DCons subst-b-Δ-def* **by** *auto*
also have ... = (*atom i* ♯ (*u*, *t*) $\#_\Delta$ Γ′ ∧ *j* ♯ (*u*, *t*) $\#_\Delta$ Γ′ ∨ *j* ♯ *x* ∧ (*j* ♯ (*u*, *t*) $\#_\Delta$ Γ′ ∨ *j* = *atom i*))
using *DCons subst-db.simps*(*2*) *has-subst-b-class.fresh-subst-if fresh-DCons subst-b-Δ-def pure-fresh fresh-at-base* **by** *auto*
finally show *?case* **by** *auto*
qed

fix *a*::*bv* **and** *tm*::Δ **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
proof (*induct tm rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-db.simps subst-b-Δ-def* **by** *auto*
next
case (*DCons u t* Γ′)
have ∗:$t[a{::=}x]_{\tau b}$ = *t* **using** *DCons fresh-DCons*[*of atom a*] *fresh-prod2*[*of atom a*] *has-subst-b-class.forget-subst subst-b-τ-def* **by** *metis*
have *subst-b* ((*u*,*t*) $\#_\Delta$ Γ′) *a x* = ((*u*,$t[a{::=}x]_{\tau b}$) $\#_\Delta$ (*subst-b* Γ′ *a x*)) **using** *subst-b-Δ-def subst-db.simps* **by** *auto*
also have ... = ((*u*, *t*) $\#_\Delta$ Γ′) **using** ∗ *DCons fresh-DCons*[*of atom a*] **by** *auto*
finally show *?case* **using**
has-subst-b-class.forget-subst fresh-DCons fresh-prod3
DCons subst-b-Δ-def has-subst-b-class.forget-subst[*of a t x*] *fresh-prod3*[*of atom a*] **by** *argo*
qed

fix *a*::*bv* **and** *tm*::Δ
show *subst-b tm a* (*B-var a*) = *tm*
proof(*induct tm rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-db.simps subst-b-Δ-def* **by** *auto*
next
case (*DCons u t* Γ′)
then show *?case* **using** *has-subst-b-class.subst-id subst-b-Δ-def subst-b-τ-def subst-db.simps* **by** *metis*
qed

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::Δ
show *p* · *subst-b t1 x1 v* = *subst-b* (*p* · *t1*) (*p* · *x1*) (*p* · *v*)
proof (*induct tm rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-b-*Δ*-def subst-db.simps* **by** *simp*
next
case (*DCons x′ b′ Γ′*)
then show *?case* **by** *argo*
qed

fix *bv*::*bv* **and** *c*::Δ **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) · *c*) = $c[z{::=}B\text{-}var\ bv]_b$
proof (*induct c rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-b-*Δ*-def subst-db.simps* **by** *auto*
next
case (*DCons u t′*)
then show *?case*
unfolding *subst-db.simps subst-b-*Δ*-def permute-*Δ*.simps*
using *DCons subst-b-*Δ*-def subst-db.simps flip-subst subst-b-*τ*-def flip-fresh-fresh fresh-at-base fresh-DCons flip-bv-u-cancel* **by** *simp*
qed

fix *bv*::*bv* **and** *c*::Δ **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ $((bv \leftrightarrow z) \cdot c)[bv{::=}v]_b = c[z{::=}v]_b$
proof (*induct c rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-b-*Δ*-def subst-db.simps* **by** *auto*
next
case (*DCons u t′*)
then show *?case*
unfolding *subst-db.simps subst-b-*Δ*-def permute-*Δ*.simps*
using *DCons subst-b-*Δ*-def subst-db.simps flip-subst subst-b-*τ*-def flip-fresh-fresh fresh-at-base fresh-DCons flip-bv-u-cancel* **by** *simp*
qed
qed
end

lemma *subst-d-b-member*:
assumes (*u*, τ) ∈ *setD* Δ
shows (*u*, $\tau[bv{::=}b]_{\tau b}$) ∈ *setD* $\Delta[bv{::=}b]_{\Delta b}$
using *assms* **by** (*induct* Δ,*auto*)

lemmas *ms-fresh-all* = *e.fresh s-branch-s-branch-list.fresh* τ*.fresh c.fresh ce.fresh v.fresh l.fresh fresh-at-base opp.fresh pure-fresh ms-fresh*

lemmas *fresh-intros*[*intro*] = *fresh-GNil x-not-in-b-set x-not-in-u-atoms x-fresh-b u-not-in-x-atoms bv-not-in-x-atoms u-not-in-b-atoms*

lemmas *subst-b-simps* = *subst-tb.simps subst-cb.simps subst-ceb.simps subst-vb.simps subst-bb.simps subst-eb.simps subst-branchb.simps subst-sb.simps*

lemma *subst-d-b-x-fresh*:
fixes $x::x$ **and** $b::b$ **and** $\Delta::\Delta$ **and** $bv::bv$
assumes $atom\ x\ \sharp\ \Delta$
shows $atom\ x\ \sharp\ \Delta[bv::=b]_{\Delta b}$
using *subst-b-fresh-x subst-b-*Δ*-def assms* **by** *metis*

lemma *subst-b-fresh-x*:
fixes $x::x$
shows $atom\ x\ \sharp\ v \Longrightarrow atom\ x\ \sharp\ v[bv::=b']_{vb}$ **and**
$atom\ x\ \sharp\ ce \Longrightarrow atom\ x\ \sharp\ ce[bv::=b']_{ceb}$ **and**
$atom\ x\ \sharp\ e \Longrightarrow atom\ x\ \sharp\ e[bv::=b']_{eb}$ **and**
$atom\ x\ \sharp\ c \Longrightarrow atom\ x\ \sharp\ c[bv::=b']_{cb}$ **and**
$atom\ x\ \sharp\ t \Longrightarrow atom\ x\ \sharp\ t[bv::=b']_{\tau b}$ **and**
$atom\ x\ \sharp\ d \Longrightarrow atom\ x\ \sharp\ d[bv::=b']_{\Delta b}$ **and**
$atom\ x\ \sharp\ g \Longrightarrow atom\ x\ \sharp\ g[bv::=b']_{\Gamma b}$ **and**
$atom\ x\ \sharp\ s \Longrightarrow atom\ x\ \sharp\ s[bv::=b']_{sb}$
using *fresh-subst-if x-fresh-b subst-b-v-def subst-b-ce-def subst-b-e-def subst-b-c-def subst-b-*τ*-def subst-b-s-def subst-g-b-x-fresh subst-d-b-x-fresh*
by *metis+*

lemma *subst-b-fresh-u-cls*:
fixes $tm::'a::has\text{-}subst\text{-}b$ **and** $x::u$
shows $atom\ x\ \sharp\ tm = atom\ x\ \sharp\ tm[bv::=b']_b$
using *fresh-subst-if*[*of atom x tm bv b'*] **using** *u-fresh-b* **by** *auto*

lemma *subst-g-b-u-fresh*:
fixes $x::u$ **and** $b::b$ **and** $\Gamma::\Gamma$ **and** $bv::bv$
assumes $atom\ x\ \sharp\ \Gamma$
shows $atom\ x\ \sharp\ \Gamma[bv::=b]_{\Gamma b}$
using *subst-b-fresh-u-cls subst-b-*Γ*-def assms* **by** *metis*

lemma *subst-d-b-u-fresh*:
fixes $x::u$ **and** $b::b$ **and** $\Gamma::\Delta$ **and** $bv::bv$
assumes $atom\ x\ \sharp\ \Gamma$
shows $atom\ x\ \sharp\ \Gamma[bv::=b]_{\Delta b}$
using *subst-b-fresh-u-cls subst-b-*Δ*-def assms* **by** *metis*

lemma *subst-b-fresh-u*:
fixes $x::u$
shows $atom\ x\ \sharp\ v \Longrightarrow atom\ x\ \sharp\ v[bv::=b']_{vb}$ **and**
$atom\ x\ \sharp\ ce \Longrightarrow atom\ x\ \sharp\ ce[bv::=b']_{ceb}$ **and**
$atom\ x\ \sharp\ e \Longrightarrow atom\ x\ \sharp\ e[bv::=b']_{eb}$ **and**
$atom\ x\ \sharp\ c \Longrightarrow atom\ x\ \sharp\ c[bv::=b']_{cb}$ **and**
$atom\ x\ \sharp\ t \Longrightarrow atom\ x\ \sharp\ t[bv::=b']_{\tau b}$ **and**
$atom\ x\ \sharp\ d \Longrightarrow atom\ x\ \sharp\ d[bv::=b']_{\Delta b}$ **and**
$atom\ x\ \sharp\ g \Longrightarrow atom\ x\ \sharp\ g[bv::=b']_{\Gamma b}$ **and**
$atom\ x\ \sharp\ s \Longrightarrow atom\ x\ \sharp\ s[bv::=b']_{sb}$
using *fresh-subst-if u-fresh-b subst-b-v-def subst-b-ce-def subst-b-e-def subst-b-c-def subst-b-*τ*-def subst-b-s-def subst-g-b-u-fresh subst-d-b-u-fresh*
by *metis+*

lemma *subst-db-u-fresh*:
  fixes $u::u$ **and** $b::b$ **and** $D::\Delta$
  assumes *atom u* $\sharp$ *D*
  shows *atom u* $\sharp$ $D[bv::=b]_{\Delta b}$
  using *assms* **proof**(*induct D rule*: Δ-*induct*)
  case *DNil*
  then show *?case* **by** *auto*
next
  case (*DCons u′ t′ D′*)
  then show *?case* **using** *subst-db.simps fresh-def fresh-DCons fresh-subst-if subst-b-τ-def*
    by (*metis fresh-Pair u-not-in-b-atoms*)
qed

lemma *flip-bt-subst4*:
  fixes $t::\tau$ **and** $bv::bv$
  assumes *atom bv* $\sharp$ *t*
  shows $t[bv'::=b]_{\tau b} = ((bv' \leftrightarrow bv) \cdot t)[bv::=b]_{\tau b}$
  using *flip-subst-subst*[*OF assms*,*of bv′ b*]
  by (*simp add*: *flip-commute subst-b-τ-def*)

lemma *subst-bt-flip-sym*:
  fixes $t1::\tau$ **and** $t2::\tau$
  assumes *atom bv* $\sharp$ *b* **and** *atom bv* $\sharp$ (*bv1*, *bv2*, *t1*, *t2*) **and** $(bv1 \leftrightarrow bv) \cdot t1 = (bv2 \leftrightarrow bv) \cdot t2$
  shows $t1[bv1::=b]_{\tau b} = t2[bv2::=b]_{\tau b}$
  using *assms flip-bt-subst4*[*of bv t1 bv1 b*] *flip-bt-subst4 fresh-prod4 fresh-Pair* **by** *metis*

end

Chapter 5

Wellformed Terms

We require that expressions and values are well-formed. This includes a notion of well-sortedness. We identify a sort with a basic type and define the judgement as two clusters of mutually recursive inductive predicates. Some of the proofs are across all of the predicates and although they seemed at first to be daunting, they have all worked out well.

named-theorems *ms-wb Facts for helping with well−sortedness*

5.1 Definitions

inductive *wfV* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow b \Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - : - › [50,50,50] 50) **and**
wfC :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow c \Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - › [50,50] 50) **and**
wfG :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow$ *bool* (‹ - ; - $\vdash_{wf}$ - › [50,50] 50) **and**
wfT :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - › [50,50] 50) **and**
wfTs :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow$ (*string*$*\tau$) *list* $\Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - › [50,50] 50) **and**
wfTh :: $\Theta \Rightarrow$ *bool* (‹ $\vdash_{wf}$ - › [50] 50) **and**
wfB :: $\Theta \Rightarrow \mathcal{B} \Rightarrow b \Rightarrow$ *bool* (‹ - ; - $\vdash_{wf}$ - › [50,50] 50) **and**
wfCE :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow ce \Rightarrow b \Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - : - › [50,50,50] 50) **and**
wfTD :: $\Theta \Rightarrow$ *type-def* $\Rightarrow$ *bool* (‹ - $\vdash_{wf}$ - › [50,50] 50)
where

wfB-intI: $\vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf}$ *B-int*
| *wfB-boolI*: $\vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf}$ *B-bool*
| *wfB-unitI*: $\vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf}$ *B-unit*
| *wfB-bitvecI*: $\vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf}$ *B-bitvec*
| *wfB-pairI*: $[\![\Theta; \mathcal{B} \vdash_{wf} b1 ; \Theta; \mathcal{B} \vdash_{wf} b2]\!] \Longrightarrow \Theta; \mathcal{B} \vdash_{wf}$ *B-pair b1 b2*

| *wfB-consI*: $[\![$
$\vdash_{wf} \Theta$;
(*AF-typedef s dclist*) $\in$ *set* Θ
$]\!] \Longrightarrow$
$\Theta; \mathcal{B} \vdash_{wf}$ *B-id s*

| *wfB-appI*: $[\![$
$\vdash_{wf} \Theta$;
$\Theta; \mathcal{B} \vdash_{wf} b$;
(*AF-typedef-poly s bv dclist*) $\in$ *set* Θ
$]\!] \Longrightarrow$

$\Theta;\ \mathcal{B} \vdash_{wf}$ *B-app s b*

| *wfV-varI*: $[\![\ \Theta;\ \mathcal{B} \vdash_{wf} \Gamma\ ;$ *Some* $(b,c) =$ *lookup* $\Gamma\ x\ ]\!] \Longrightarrow \Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *V-var x* : *b*
| *wfV-litI*: $\Theta;\ \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *V-lit l* : *base-for-lit l*

| *wfV-pairI*: $[\![$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v1* : *b1* ;
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v2* : *b2*
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ (*V-pair v1 v2*) : *B-pair b1 b2*

| *wfV-consI*: $[\![$
AF-typedef s dclist $\in$ *set* Θ;
(*dc*, $\{\!|\ x : b'\ \mid c\ |\!\}$) $\in$ *set dclist* ;
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} v : b'$
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *V-cons s dc v* : *B-id s*

| *wfV-conspI*: $[\![$
AF-typedef-poly s bv dclist $\in$ *set* Θ;
(*dc*, $\{\!|\ x : b'\ \mid c\ |\!\}$) $\in$ *set dclist* ;
$\Theta\ ;\ \mathcal{B} \vdash_{wf} b$;
atom bv $\sharp$ $(\Theta, \mathcal{B}, \Gamma, b, v)$;
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} v : b'[bv::=b]_{bb}$
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *V-consp s dc b v* : *B-app s b*

| *wfCE-valI* : $[\![$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} v : b$
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-val v* : *b*

| *wfCE-plusI*: $[\![$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v1* : *B-int*;
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v2* : *B-int*
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-op Plus v1 v2* : *B-int*

| *wfCE-leqI*:$[\![$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v1* : *B-int*;
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v2* : *B-int*
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-op LEq v1 v2* : *B-bool*

| *wfCE-eqI*:$[\![$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v1* : *b*;
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v2* : *b*
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-op Eq v1 v2* : *B-bool*

| *wfCE-fstI*: $[\![$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *v1* : *B-pair b1 b2*

$]\!] \Longrightarrow$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{CE-fst v1} : \textit{b1}$

$|$ *wfCE-sndI*: $[\![$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{v1} : \textit{B-pair b1 b2}$
$]\!] \Longrightarrow$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{CE-snd v1} : \textit{b2}$

$|$ *wfCE-concatI*: $[\![$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{v1} : \textit{B-bitvec}$;
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{v2} : \textit{B-bitvec}$
$]\!] \Longrightarrow$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{CE-concat v1 v2} : \textit{B-bitvec}$

$|$ *wfCE-lenI*: $[\![$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{v1} : \textit{B-bitvec}$
$]\!] \Longrightarrow$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{CE-len v1} : \textit{B-int}$

$|$ *wfTI* : $[\![$
$\quad \textit{atom } z \sharp (\Theta, \mathcal{B}, \Gamma)$;
$\quad \Theta; \mathcal{B} \vdash_{wf} b$;
$\quad \Theta; \mathcal{B} ; (z,b,\textit{C-true}) \#_\Gamma \Gamma \vdash_{wf} c$
$]\!] \Longrightarrow$
$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!| z : b \mid c |\!\}$

$|$ *wfC-eqI*: $[\![$
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{e1} : b$;
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{e2} : b \;]\!] \Longrightarrow$
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-eq e1 e2}$
$|$ *wfC-trueI*: $\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-true}$
$|$ *wfC-falseI*: $\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-false}$

$|$ *wfC-conjI*: $[\![\Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1} ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c2}]\!] \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-conj c1 c2}$
$|$ *wfC-disjI*: $[\![\Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1} ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c2}]\!] \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-disj c1 c2}$
$|$ *wfC-notI*: $[\![\Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1}]\!] \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-not c1}$
$|$ *wfC-impI*: $[\![\Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1}$;
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c2}]\!] \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-imp c1 c2}$

$|$ *wfG-nilI*: $\vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \textit{GNil}$
$|$ *wfG-cons1I*: $[\![\; c \notin \{ \textit{TRUE}, \textit{FALSE} \}$;
$\qquad \Theta; \mathcal{B} \vdash_{wf} \Gamma$;
$\qquad \textit{atom } x \sharp \Gamma$;
$\qquad \Theta ; \mathcal{B} ; (x,b,\textit{C-true})\#_\Gamma\Gamma \vdash_{wf} c$; $\textit{wfB } \Theta\ \mathcal{B}\ b$
$\qquad]\!] \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} ((x,b,c)\#_\Gamma\Gamma)$
$|$ *wfG-cons2I*: $[\![\; c \in \{ \textit{TRUE}, \textit{FALSE} \}$;
$\qquad \Theta; \mathcal{B} \vdash_{wf} \Gamma$;
$\qquad \textit{atom } x \sharp \Gamma$;
$\qquad \textit{wfB } \Theta\ \mathcal{B}\ b$
$\qquad]\!] \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} ((x,b,c)\#_\Gamma\Gamma)$

$|$ *wfTh-emptyI*: $\vdash_{wf} []$

| *wfTh-consI*: ⟦
    $(name\text{-}of\text{-}type\ tdef) \notin name\text{-}of\text{-}type\ `\ set\ \Theta$;
    $\vdash_{wf} \Theta$;
    $\Theta \vdash_{wf}\ tdef$ ⟧ $\Longrightarrow\ \vdash_{wf} tdef\#\Theta$

| *wfTD-simpleI*: ⟦
    Θ ; {||} ; $GNil \vdash_{wf} lst$
  ⟧ $\Longrightarrow$
    $\Theta \vdash_{wf} (AF\text{-}typedef\ s\ lst\ )$

| *wfTD-poly*: ⟦
    Θ ; {|bv|} ; $GNil \vdash_{wf} lst$
  ⟧ $\Longrightarrow$
  $\Theta \vdash_{wf} (AF\text{-}typedef\text{-}poly\ s\ bv\ lst)$

| *wfTs-nil*: $\Theta;\ \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} []::(string*\tau)\ list$
| *wfTs-cons*: ⟦ $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \tau$;
    $dc \notin fst\ `\ set\ ts$;
    $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} ts::(string*\tau)\ list$ ⟧ $\Longrightarrow \Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} ((dc,\tau)\#ts)$

inductive-cases *wfC-elims*:
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}true$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}false$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}eq\ e1\ e2$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}conj\ c1\ c2$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}disj\ c1\ c2$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}not\ c1$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ C\text{-}imp\ c1\ c2$

inductive-cases *wfV-elims*:
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ V\text{-}var\ x : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ V\text{-}lit\ l : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ V\text{-}pair\ v1\ v2 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ V\text{-}cons\ tyid\ dc\ v : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ V\text{-}consp\ tyid\ dc\ b\ v : b'$

inductive-cases *wfCE-elims*:
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}val\ v : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}op\ Plus\ v1\ v2 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}op\ LEq\ v1\ v2 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}fst\ v1 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}snd\ v1 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}concat\ v1\ v2 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}len\ v1 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}op\ opp\ v1\ v2 : b$
  $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}\ CE\text{-}op\ Eq\ v1\ v2 : b$

inductive-cases *wfT-elims*:
  $\Theta;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf}\ \tau::\tau$
  $\Theta;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf}$ ⦃ $z : b \mid c$ ⦄

inductive-cases *wfG-elims*:
Θ; $\mathcal{B}$ $\vdash_{wf}$ *GNil*
Θ; $\mathcal{B}$ $\vdash_{wf}$ $(x,b,c)\#_\Gamma\Gamma$
Θ; $\mathcal{B}$ $\vdash_{wf}$ $(x,b,TRUE)\#_\Gamma\Gamma$
Θ; $\mathcal{B}$ $\vdash_{wf}$ $(x,b,FALSE)\#_\Gamma\Gamma$

inductive-cases *wfTh-elims*:
$\vdash_{wf}$ []
$\vdash_{wf}$ *td*#Π

inductive-cases *wfTD-elims*:
$\Theta \vdash_{wf}$ (*AF-typedef s lst*)
$\Theta \vdash_{wf}$ (*AF-typedef-poly s bv lst*)

inductive-cases *wfTs-elims*:
Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ ([]::((*string*$*\tau$) *list*))
Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ ((*t*#*ts*)::((*string*$*\tau$) *list*))

inductive-cases *wfB-elims*:
Θ; $\mathcal{B}$ $\vdash_{wf}$ *B-pair b1 b2*
Θ; $\mathcal{B}$ $\vdash_{wf}$ *B-id s*
Θ; $\mathcal{B}$ $\vdash_{wf}$ *B-app s b*

equivariance *wfV*

This setup of 'avoiding' is not complete and for some of lemmas, such as weakening, do it the hard way

nominal-inductive *wfV*
avoids *wfV-conspI*: *bv* | *wfTI*: *z*
proof(*goal-cases*)
case (*1 s bv dclist* Θ *dc x b′ c* $\mathcal{B}$ *b* Γ *v*)

moreover hence *atom bv* $\sharp$ *V-consp s dc b v* **using** *v.fresh fresh-prodN pure-fresh* **by** *metis*
moreover have *atom bv* $\sharp$ *B-app s b* **using** *b.fresh fresh-prodN pure-fresh 1* **by** *metis*
ultimately show *?case* **using** *b.fresh v.fresh pure-fresh fresh-star-def fresh-prodN* **by** *fastforce*
next
case (*2 s bv dclist* Θ *dc x b′ c* $\mathcal{B}$ *b* Γ *v*)
then show *?case* **by** *auto*
next
case (*3 z* Γ Θ $\mathcal{B}$ *b c*)
then show *?case* **using** τ*.fresh fresh-star-def fresh-prodN* **by** *fastforce*
next
case (*4 z* Γ Θ $\mathcal{B}$ *b c*)
then show *?case* **by** *auto*
qed

inductive
wfE :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow b \Rightarrow$ *bool* (‹ - ; - ; - ; - ; - $\vdash_{wf}$ - : - › [*50*,*50*,*50*] *50*) **and**
wfS :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow b \Rightarrow$ *bool* (‹ - ; - ; - ; - ; - $\vdash_{wf}$ - : - › [*50*,*50*,*50*] *50*) **and**
wfCS :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* $\Rightarrow$ *string* $\Rightarrow \tau \Rightarrow$ *branch-s* $\Rightarrow b \Rightarrow$ *bool* (‹ - ; - ; - ; - ; - ; - ; - ; - $\vdash_{wf}$ - : - › [*50*,*50*,*50*,*50*,*50*] *50*) **and**
wfCSS :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* $\Rightarrow$ (*string* $*$ τ) *list* $\Rightarrow$ *branch-list* $\Rightarrow b \Rightarrow$ *bool* (‹ - ; - ; - ; -

$; - ; - ; - \vdash_{wf} - : -$ › [50,50,50,50,50] 50) **and**
wfPhi :: $\Theta \Rightarrow \Phi \Rightarrow$ *bool* (‹ - $\vdash_{wf}$ - › [50,50] 50) **and**
wfD :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - › [50,50] 50) **and**
wfFTQ :: $\Theta \Rightarrow \Phi \Rightarrow$ *fun-typ-q* $\Rightarrow$ *bool* (‹ - ; - $\vdash_{wf}$ - › [50] 50) **and**
wfFT :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow$ *fun-typ* $\Rightarrow$ *bool* (‹ - ; - ; - $\vdash_{wf}$ - › [50] 50) **where**

wfE-valI : ⟦
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$
⟧ $\Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf}$ *AE-val v* : *b*

| *wfE-plusI*: ⟦
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1$: *B-int*;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2$: *B-int*
⟧ $\Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf}$ *AE-op Plus v1 v2* : *B-int*

| *wfE-leqI*:⟦
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1$: *B-int*;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2$: *B-int*
⟧ $\Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf}$ *AE-op LEq v1 v2* : *B-bool*

| *wfE-eqI*:⟦
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1$: *b*;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v2$: *b*;
b ∈ { *B-bool*, *B-int*, *B-unit* }
⟧ $\Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf}$ *AE-op Eq v1 v2* : *B-bool*

| *wfE-fstI*: ⟦
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1$: *B-pair b1 b2*
⟧ $\Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf}$ *AE-fst v1* : *b1*

| *wfE-sndI*: ⟦
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v1$: *B-pair b1 b2*
⟧ $\Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf}$ *AE-snd v1* : *b2*

| *wfE-concatI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}bitvec}$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v2} : \mathit{B\text{-}bitvec}$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}concat}\ \mathit{v1}\ \mathit{v2} : \mathit{B\text{-}bitvec}$

| *wfE-splitI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}bitvec}$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v2} : \mathit{B\text{-}int}$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}split}\ \mathit{v1}\ \mathit{v2} : \mathit{B\text{-}pair}\ \mathit{B\text{-}bitvec}\ \mathit{B\text{-}bitvec}$

| *wfE-lenI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}bitvec}$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}len}\ \mathit{v1} : \mathit{B\text{-}int}$

| *wfE-appI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\mathit{Some}\ (\mathit{AF\text{-}fundef}\ f\ (\mathit{AF\text{-}fun\text{-}typ\text{-}none}\ (\mathit{AF\text{-}fun\text{-}typ}\ x\ b\ c\ \tau\ s))) = \mathit{lookup\text{-}fun}\ \Phi\ f$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}app}\ f\ v : \mathit{b\text{-}of}\ \tau$

| *wfE-appPI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta; \mathcal{B} \vdash_{wf} b'$;
$\mathit{atom}\ \mathit{bv} \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, b', v, (\mathit{b\text{-}of}\ \tau)[\mathit{bv}::=b']_b)$;
$\mathit{Some}\ (\mathit{AF\text{-}fundef}\ f\ (\mathit{AF\text{-}fun\text{-}typ\text{-}some}\ \mathit{bv}\ (\mathit{AF\text{-}fun\text{-}typ}\ x\ b\ c\ \tau\ s))) = \mathit{lookup\text{-}fun}\ \Phi\ f$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : (b[\mathit{bv}::=b']_b)$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} (\mathit{AE\text{-}appP}\ f\ b'\ v) : ((\mathit{b\text{-}of}\ \tau)[\mathit{bv}::=b']_b)$

| *wfE-mvarI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$(u,\tau) \in \mathit{setD}\ \Delta$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}mvar}\ u : \mathit{b\text{-}of}\ \tau$

| *wfS-valI*: $\llbracket$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$

$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} (\textit{AS-val } v) : b$

| *wfS-letI*: $\llbracket$
$\textit{wfE}\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ \ e\ b'\ ;$
$\Theta ; \Phi ; \mathcal{B} ; (x{,}b'{,}\textit{C-true}) \#_{\Gamma}\ \Gamma ; \Delta \vdash_{wf} s : b;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta ;$
$\textit{atom } x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, e, b)$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} \textit{LET } x = e \textit{ IN } s : b$

| *wfS-assertI*: $\llbracket$
$\Theta ; \Phi ; \mathcal{B} ; (x{,}\textit{B-bool}{,}c) \#_{\Gamma}\ \Gamma ; \Delta \vdash_{wf} s : b;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} c ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta ;$
$\textit{atom } x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s)$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} \textit{ASSERT } c \textit{ IN } s : b$

| *wfS-let2I*: $\llbracket$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta\ \vdash_{wf} s1 : \textit{b-of } \tau\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \tau;$
$\Theta ; \Phi ; \mathcal{B} ; (x{,}\textit{b-of } \tau{,}\textit{C-true}) \#_{\Gamma}\ \Gamma ; \Delta \vdash_{wf} s2 : b ;$
$\textit{atom } x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, s1, b{,}\tau)$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} \textit{LET } x : \tau = s1 \textit{ IN } s2 : b$

| *wfS-ifI*: $\llbracket$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} v\ : \textit{B-bool};$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} s1 : b ;$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} s2 : b$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} \textit{IF } v \textit{ THEN } s1 \textit{ ELSE } s2 : b$

| *wfS-varI* : $\llbracket$
$\textit{wfT}\ \Theta\ \mathcal{B}\ \Gamma\ \ \tau\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} v\ : \textit{b-of } \tau;$
$\textit{atom } u \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \tau, v, b);$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ;\ (u{,}\tau) \#_{\Delta} \Delta \vdash_{wf} s : b$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} \textit{VAR } u : \tau = v \textit{ IN } s : b$

| *wfS-assignI*: $\llbracket$
$(u{,}\tau) \in \textit{setD}\ \Delta\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta ;$
$\Theta \vdash_{wf} \Phi ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} v\ : \textit{b-of } \tau$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} u ::= v : \textit{B-unit}$

| *wfS-whileI*: $\llbracket$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash_{wf} s1 : \textit{B-bool}\ ;$

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \textit{s2} : b$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \textit{WHILE s1 DO} \{ \textit{ s2 } \} : b$

| *wfS-seqI*: $\llbracket$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \textit{s1} : \textit{B-unit}$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \textit{s2} : b$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \textit{s1} ;; \textit{s2} : b$

| *wfS-matchI*: $\llbracket$
wfV Θ $\mathcal{B}$ Γ *v* (*B-id tid*) ;
(*AF-typedef tid dclist*) $\in$ *set* Θ;
wfD Θ $\mathcal{B}$ Γ Δ ;
$\Theta \vdash_{wf} \Phi$;
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; \textit{tid} ; \textit{dclist} \vdash_{wf} \textit{cs} : b$
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \textit{AS-match v cs} : b$

| *wfS-branchI*: $\llbracket$
$\Theta ; \Phi ; \mathcal{B} ; (x, \textit{b-of } \tau, \textit{C-true}) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b$;
atom x $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ, Δ, Γ,τ);
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; \textit{tid} ; \textit{dc} ; \tau \vdash_{wf} \textit{dc } x \Rightarrow s : b$

| *wfS-finalI*: $\llbracket$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; \textit{tid} ; \textit{dc} ; t \vdash_{wf} \textit{cs} : b$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; \textit{tid} ; [(\textit{dc},t)] \vdash_{wf} \textit{AS-final cs} : b$

| *wfS-cons*: $\llbracket$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; \textit{tid} ; \textit{dc} ; t \vdash_{wf} \textit{cs} : b$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; \textit{tid} ; \textit{dclist} \vdash_{wf} \textit{css} : b$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; \textit{tid} ; (\textit{dc},t)\#\textit{dclist} \vdash_{wf} \textit{AS-cons cs css} : b$

| *wfD-emptyI*: $\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} []_{\Delta}$

| *wfD-cons*: $\llbracket$
$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta{::}\Delta$;
$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau$;
$u \notin \textit{fst} ` \textit{setD } \Delta$
$\rrbracket \Longrightarrow$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ((u,\tau) \#_{\Delta} \Delta)$

| *wfPhi-emptyI*: $\vdash_{wf} \Theta \Longrightarrow \Theta \vdash_{wf} []$

| *wfPhi-consI*: $\llbracket$
$f \notin \textit{name-of-fun} ` \textit{set } \Phi$;
$\Theta ; \Phi \vdash_{wf} \textit{ft}$;
$\Theta \vdash_{wf} \Phi$

$\rrbracket \Longrightarrow$
  $\Theta \ \vdash_{wf}$ ((*AF-fundef f ft*)#Φ)

| *wfFTNone*: Θ ; Φ ; {||} $\vdash_{wf}$ *ft* $\Longrightarrow$ Θ ; $\Phi \vdash_{wf}$ *AF-fun-typ-none ft*
| *wfFTSome*: Θ ; Φ ; {| *bv* |} $\vdash_{wf}$ *ft* $\Longrightarrow$ Θ ; $\Phi \vdash_{wf}$ *AF-fun-typ-some bv ft*

| *wfFTI*: $\llbracket$
  Θ ; $B \ \vdash_{wf} \ b$;
  supp s $\subseteq$ {*atom x*} $\cup$ *supp B* ;
  supp c $\subseteq$ { *atom x* } ;
  Θ ; B ; (x,b,c) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ;
  $\Theta \vdash_{wf} \Phi$
$\rrbracket \Longrightarrow$
  Θ ; Φ ; $B \vdash_{wf}$ (*AF-fun-typ x b c* τ *s*)

inductive-cases *wfE-elims*:
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-val v* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-op Plus v1 v2* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-op LEq v1 v2* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-fst v1* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-snd v1* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-concat v1 v2* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-len v1* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-op opp v1 v2* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-app f v*: *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-appP f* b' *v*: *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-mvar u* : *b*
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-op Eq v1 v2* : *b*

inductive-cases *wfCS-elims*:
  Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ (*cs*::*branch-s*) : *b*
  Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dc* $\vdash_{wf}$ (*cs*::*branch-list*) : *b*

inductive-cases *wfPhi-elims*:
  $\Theta \ \vdash_{wf}$ []
  $\Theta \ \vdash_{wf}$ ((*AF-fundef f ft*)#Π)
  $\Theta \ \vdash_{wf}$ (*fd*#Φ::Φ)

declare[[*simproc del*: *alpha-lst*]]

inductive-cases *wfFTQ-elims*:
  Θ ; $\Phi \ \vdash_{wf}$ *AF-fun-typ-none ft*
  Θ ; $\Phi \ \vdash_{wf}$ *AF-fun-typ-some bv ft*
  Θ ; $\Phi \ \vdash_{wf}$ *AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)

inductive-cases *wfFT-elims*:
  Θ ; Φ ; $\mathcal{B} \ \vdash_{wf}$ *AF-fun-typ x b c* τ *s*

declare[[*simproc add*: *alpha-lst*]]

inductive-cases *wfD-elims*:
  Π ; $\mathcal{B}$; (Γ::Γ) $\vdash_{wf}$ $[]_\Delta$

Π ; $\mathcal{B}$; $(\Gamma::\Gamma) \vdash_{wf} (u,\tau) \#_{\Delta} \Delta::\Delta$

equivariance *wfE*

nominal-inductive *wfE*
avoids *wfE-appPI*: *bv* | *wfS-varI*: *u* | *wfS-letI*: *x* | *wfS-let2I*: *x* | *wfS-branchI*: *x* | *wfS-assertI*: *x*

proof(*goal-cases*)
case (*1* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv* *v* τ *f* *x* *b* *c* *s*)
moreover hence *atom bv* ♯ *AE-appP f b' v* **using** *pure-fresh fresh-prodN e.fresh* **by** *auto*
ultimately show *?case* **using** *fresh-star-def* **by** *fastforce*
next
case (*2* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv* *v* τ *f* *x* *b* *c* *s*)
then show *?case* **by** *auto*
next
case (*3* Φ Θ $\mathcal{B}$ Γ Δ *e* *b'* *x* *s* *b*)
moreover hence *atom x* ♯ *LET x = e IN s* **using** *fresh-prodN* **by** *auto*
ultimately show *?case* **using** *fresh-prodN fresh-star-def* **by** *fastforce*
next
case (*4* Φ Θ $\mathcal{B}$ Γ Δ *e* *b'* *x* *s* *b*)
then show *?case* **by** *auto*
next
case (*5* Θ Φ $\mathcal{B}$ *x* *c* Γ Δ *s* *b*)
hence *atom x* ♯ *ASSERT c IN s* **using** *s-branch-s-branch-list.fresh* **by** *auto*
then show *?case* **using** *fresh-prodN fresh-star-def 5* **by** *fastforce*
next
case (*6* Θ Φ $\mathcal{B}$ *x* *c* Γ Δ *s* *b*)
then show *?case* **by** *auto*
next
case (*7* Φ Θ $\mathcal{B}$ Γ Δ *s1* τ *x* *s2* *b*)
hence *atom x* ♯ τ $\wedge$ *atom x* ♯ *s1* **using** *fresh-prodN* **by** *metis*
moreover hence *atom x* ♯ *LET x* : τ = *s1 IN s2*
using *s-branch-s-branch-list.fresh*(*3*)[*of atom x x* τ *s1 s2*] *fresh-prodN* **by** *simp*
ultimately show *?case* **using** *fresh-prodN fresh-star-def 7* **by** *fastforce*
next
case (*8* Φ Θ $\mathcal{B}$ Γ Δ *s1* τ *x* *s2* *b*)
then show *?case* **by** *auto*
next
case (*9* Θ $\mathcal{B}$ Γ τ *v* *u* Φ Δ *b* *s*)
moreover hence *atom u* ♯ *AS-var u* τ *v s* **using** *fresh-prodN s-branch-s-branch-list.fresh* **by** *simp*
ultimately show *?case* **using** *fresh-star-def fresh-prodN s-branch-s-branch-list.fresh* **by** *fastforce*
next
case (*10* Θ $\mathcal{B}$ Γ τ *v* *u* Φ Δ *b* *s*)
then show *?case* **by** *auto*
next
case (*11* Φ Θ $\mathcal{B}$ *x* τ Γ Δ *s* *b* *tid* *dc*)
moreover have *atom x* ♯ (*dc x* ⇒ *s*) **using** *pure-fresh s-branch-s-branch-list.fresh* **by** *auto*
ultimately show *?case* **using** *fresh-prodN fresh-star-def pure-fresh* **by** *fastforce*
next
case (*12* Φ Θ $\mathcal{B}$ *x* τ Γ Δ *s* *b* *tid* *dc*)
then show *?case* **by** *auto*
qed

inductive *wfVDs* :: *var-def list* ⇒ *bool* **where**

wfVDs-nilI: *wfVDs* []

| *wfVDs-consI*: ⟦
  atom u ♯ *ts*;
  wfV ([]::Θ) {||} *GNil* *v* (*b-of* τ);
  wfT ([]::Θ) {||} *GNil* τ;
  wfVDs ts
⟧ ⟹ *wfVDs* ((*AV-def u* τ *v*)#*ts*)

equivariance *wfVDs*
nominal-inductive *wfVDs* **.**

end

hide-const *Syntax.dom*

Chapter 6

Refinement Constraint Logic

Semantics for the logic we use in the refinement constraints. It is a multi-sorted, quantifier free logic with polymorphic datatypes and linear arithmetic. We could have modelled by using one of the encodings to FOL however we wanted to explore using a more direct model.

6.1 Evaluation and Satisfiability

6.1.1 Valuation

Refinement constraint logic values. SUt is a value for the uninterpreted sort that corresponds to basic type variables. For now we only need one of these universes. We wrap an smt_val inside it during a process we call 'boxing' which is introduced in the RCLogicL theory

nominal-datatype *rcl-val = SBitvec bit list | SNum int | SBool bool | SPair rcl-val rcl-val |*
SCons tyid string rcl-val | SConsp tyid string b rcl-val |
SUnit | SUt rcl-val

RCL sorts. Represent our domains. The universe is the union of all of the these. S_Ut is the single uninterpreted sort. These map almost directly to basic type but we have them to distinguish syntax (basic types) and semantics (RCL sorts)

nominal-datatype *rcl-sort = S-bool | S-int | S-unit | S-pair rcl-sort rcl-sort | S-id tyid | S-app tyid rcl-sort | S-bitvec | S-ut*

type-synonym *valuation = (x,rcl-val) map*

type-synonym *type-valuation = (bv,rcl-sort) map*

Well-sortedness for RCL values

inductive *wfRCV*:: Θ ⇒ *rcl-val* ⇒ *b* ⇒ *bool* (‹ - ⊢ - : -› [*50*,*50*] *50*) **where**
wfRCV-BBitvecI: *P* ⊢ (*SBitvec bv*) : *B-bitvec*
| *wfRCV-BIntI*: *P* ⊢ (*SNum n*) : *B-int*
| *wfRCV-BBoolI*: *P* ⊢ (*SBool b*) : *B-bool*
| *wfRCV-BPairI*: ⟦ *P* ⊢ *s1* : *b1* ; *P* ⊢ *s2* : *b2* ⟧ ⟹ *P* ⊢ (*SPair s1 s2*) : (*B-pair b1 b2*)
| *wfRCV-BConsI*: ⟦ *AF-typedef s dclist* ∈ *set* Θ;
(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* ;
Θ ⊢ *s1* : *b* ⟧ ⟹ Θ ⊢(*SCons s dc s1*) : (*B-id s*)
| *wfRCV-BConsPI*:⟦ *AF-typedef-poly s bv dclist* ∈ *set* Θ;

$(dc, \{\!|\ x : b \ \mid c\ |\!\}) \in set\ dclist$;
atom bv $\sharp$ (Θ, *SConsp s dc b′ s1*, *B-app s b′*);
$\Theta \vdash s1 : b[bv::=b']_{bb}$ $]\!] \Longrightarrow \Theta \vdash$(*SConsp s dc b′ s1*) : (*B-app s b′*)
| *wfRCV-BUnitI*: $P \vdash$ *SUnit* : *B-unit*
| *wfRCV-BVarI*: $P \vdash$ (*SUt n*) : (*B-var bv*)
equivariance *wfRCV*
nominal-inductive *wfRCV*
avoids *wfRCV-BConsPI*: *bv*
proof(*goal-cases*)
case (*1 s bv dclist* Θ *dc x b c b′ s1*)
then show *?case* **using** *fresh-star-def* **by** *auto*
next
case (*2 s bv dclist* Θ *dc x b c s1 b′*)
then show *?case* **by** *auto*
qed

inductive-cases *wfRCV-elims* :
wfRCV P s B-bitvec
wfRCV P s (*B-pair b1 b2*)
wfRCV P s (*B-int*)
wfRCV P s (*B-bool*)
wfRCV P s (*B-id ss*)
wfRCV P s (*B-var bv*)
wfRCV P s (*B-unit*)
wfRCV P s (*B-app tyid b*)
wfRCV P (*SBitvec bv*) *b*
wfRCV P (*SNum n*) *b*
wfRCV P (*SBool n*) *b*
wfRCV P (*SPair s1 s2*) *b*
wfRCV P (*SCons s dc s1*) *b*
wfRCV P (*SConsp s dc b′ s1*) *b*
wfRCV P SUnit b
wfRCV P (*SUt s1*) *b*

Sometimes we want to assert $P \vdash s \sim b[bv=b']$ and we want to know what b is however substitution is not injective so we can't write this in terms of *wfRCV*. So we define a relation that makes the components of the substitution explicit.

inductive *wfRCV-subst*:: $\Theta \Rightarrow$ *rcl-val* $\Rightarrow b \Rightarrow (bv*b)$ *option* $\Rightarrow$ *bool* **where**
wfRCV-subst-BBitvecI: *wfRCV-subst P* (*SBitvec bv*) *B-bitvec sub*
| *wfRCV-subst-BIntI*: *wfRCV-subst P* (*SNum n*) *B-int sub*
| *wfRCV-subst-BBoolI*: *wfRCV-subst P* (*SBool b*) *B-bool sub*
| *wfRCV-subst-BPairI*: $[\![$ *wfRCV-subst P s1 b1 sub* ; *wfRCV-subst P s2 b2 sub* $]\!] \Longrightarrow$ *wfRCV-subst P* (*SPair s1 s2*) (*B-pair b1 b2*) *sub*
| *wfRCV-subst-BConsI*: $[\![$ *AF-typedef s dclist* $\in$ *set* Θ;
$(dc, \{\!|\ x : b \ \mid c\ |\!\}) \in set\ dclist$;
wfRCV-subst Θ *s1 b None* $]\!] \Longrightarrow$ *wfRCV-subst* Θ (*SCons s dc s1*) (*B-id s*) *sub*
| *wfRCV-subst-BConspI*: $[\![$ *AF-typedef-poly s bv dclist* $\in$ *set* Θ;
$(dc, \{\!|\ x : b \ \mid c\ |\!\}) \in set\ dclist$;
wfRCV-subst Θ *s1* $(b[bv::=b']_{bb})$ *sub* $]\!] \Longrightarrow$ *wfRCV-subst* Θ (*SConsp s dc b′ s1*) (*B-app s b′*) *sub*
| *wfRCV-subst-BUnitI*: *wfRCV-subst P SUnit B-unit sub*
| *wfRCV-subst-BVar1I*: *bvar* $\neq$ *bv* $\Longrightarrow$ *wfRCV-subst P* (*SUt n*) (*B-var bv*) (*Some* (*bvar*,*bin*))
| *wfRCV-subst-BVar2I*: $[\![$ *bvar* = *bv*; *wfRCV-subst P s bin None* $]\!] \Longrightarrow$ *wfRCV-subst P s* (*B-var bv*)

(*Some* (*bvar*,*bin*))
| *wfRCV-subst-BVar3I*: *wfRCV-subst P* (*SUt n*) (*B-var bv*) *None*
equivariance *wfRCV-subst*
nominal-inductive *wfRCV-subst* **.**

6.1.2 Evaluation base-types

inductive *eval-b* :: *type-valuation* ⇒ *b* ⇒ *rcl-sort* ⇒ *bool* (‹- ⟦ - ⟧ ~ - ›) **where**
v ⟦ *B-bool* ⟧ ~ *S-bool*
| *v* ⟦ *B-int* ⟧ ~ *S-int*
| *Some s* = *v bv* ⟹ *v* ⟦ *B-var bv* ⟧ ~ *s*
equivariance *eval-b*
nominal-inductive *eval-b* **.**

6.1.3 Wellformed vvaluations

definition *wfI* :: Θ ⇒ Γ ⇒ *valuation* ⇒ *bool* (‹ - ; - ⊢ -›) **where**
Θ ; Γ ⊢ *i* = (∀ (*x*,*b*,*c*) ∈ *toSet* Γ. ∃ *s*. *Some s* = *i x* ∧ Θ ⊢ *s* : *b*)

6.1.4 Evaluating Terms

nominal-function *eval-l* :: *l* ⇒ *rcl-val* (‹⟦ - ⟧ ›) **where**
⟦ *L-true* ⟧ = *SBool True*
| ⟦ *L-false* ⟧ = *SBool False*
| ⟦ *L-num n* ⟧ = *SNum n*
| ⟦ *L-unit* ⟧ = *SUnit*
| ⟦ *L-bitvec n* ⟧ = *SBitvec n*
apply(*auto simp*: *eqvt-def eval-l-graph-aux-def*)
by (*metis l.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *eval-v* :: *valuation* ⇒ *v* ⇒ *rcl-val* ⇒ *bool* (‹- ⟦ - ⟧ ~ - ›) **where**
eval-v-litI: *i* ⟦ *V-lit l* ⟧ ~ ⟦ *l* ⟧
| *eval-v-varI*: *Some sv* = *i x* ⟹ *i* ⟦ *V-var x* ⟧ ~ *sv*
| *eval-v-pairI*: ⟦ *i* ⟦ *v1* ⟧ ~ *s1* ; *i* ⟦ *v2* ⟧ ~ *s2* ⟧ ⟹ *i* ⟦ *V-pair v1 v2* ⟧ ~ *SPair s1 s2*
| *eval-v-consI*: *i* ⟦ *v* ⟧ ~ *s* ⟹ *i* ⟦ *V-cons tyid dc v* ⟧ ~ *SCons tyid dc s*
| *eval-v-conspI*: *i* ⟦ *v* ⟧ ~ *s* ⟹ *i* ⟦ *V-consp tyid dc b v* ⟧ ~ *SConsp tyid dc b s*
equivariance *eval-v*
nominal-inductive *eval-v* **.**

inductive-cases *eval-v-elims*:
i ⟦ *V-lit l* ⟧ ~ *s*
i ⟦ *V-var x* ⟧ ~ *s*
i ⟦ *V-pair v1 v2* ⟧ ~ *s*
i ⟦ *V-cons tyid dc v* ⟧ ~ *s*
i ⟦ *V-consp tyid dc b v* ⟧ ~ *s*

inductive *eval-e* :: *valuation* ⇒ *ce* ⇒ *rcl-val* ⇒ *bool* (‹- ⟦ - ⟧ ~ - ›) **where**
eval-e-valI: *i* ⟦ *v* ⟧ ~ *sv* ⟹ *i* ⟦ *CE-val v* ⟧ ~ *sv*
| *eval-e-plusI*: ⟦ *i* ⟦ *v1* ⟧ ~ *SNum n1*; *i* ⟦ *v2* ⟧ ~ *SNum n2* ⟧ ⟹ *i* ⟦ (*CE-op Plus v1 v2*) ⟧ ~ (*SNum* (*n1*+*n2*))
| *eval-e-leqI*: ⟦ *i* ⟦ *v1* ⟧ ~ (*SNum n1*); *i* ⟦ *v2* ⟧ ~ (*SNum n2*) ⟧ ⟹ *i* ⟦ (*CE-op LEq v1 v2*) ⟧ ~ (*SBool* (*n1* ≤ *n2*))

| *eval-e-eqI*: ⟦ *i* ⟦ *v1* ⟧ ~ *s1*; *i* ⟦ *v2* ⟧ ~ *s2* ⟧ ⟹ *i* ⟦ (*CE-op Eq v1 v2*) ⟧ ~ (*SBool* (*s1* = *s2*))
| *eval-e-fstI*: ⟦ *i* ⟦ *v* ⟧ ~ *SPair v1 v2* ⟧ ⟹ *i* ⟦ (*CE-fst v*) ⟧ ~ *v1*
| *eval-e-sndI*: ⟦ *i* ⟦ *v* ⟧ ~ *SPair v1 v2* ⟧ ⟹ *i* ⟦ (*CE-snd v*) ⟧ ~ *v2*
| *eval-e-concatI*:⟦ *i* ⟦ *v1* ⟧ ~ (*SBitvec bv1*); *i* ⟦ *v2* ⟧ ~ (*SBitvec bv2*) ⟧ ⟹ *i* ⟦ (*CE-concat v1 v2*) ⟧ ~ (*SBitvec* (*bv1*@*bv2*))
| *eval-e-lenI*:⟦ *i* ⟦ *v* ⟧ ~ (*SBitvec bv*) ⟧ ⟹ *i* ⟦ (*CE-len v*) ⟧ ~ (*SNum* (*int* (*List.length bv*)))
equivariance *eval-e*
nominal-inductive *eval-e* **.**

inductive-cases *eval-e-elims*:
  i ⟦ (*CE-val v*) ⟧ ~ *s*
  i ⟦ (*CE-op Plus v1 v2*) ⟧ ~ *s*
  i ⟦ (*CE-op LEq v1 v2*) ⟧ ~ *s*
  i ⟦ (*CE-op Eq v1 v2*) ⟧ ~ *s*
  i ⟦ (*CE-fst v*) ⟧ ~ *s*
  i ⟦ (*CE-snd v*) ⟧ ~ *s*
  i ⟦ (*CE-concat v1 v2*) ⟧ ~ *s*
  i ⟦ (*CE-len v*) ⟧ ~ *s*

inductive *eval-c* :: *valuation* ⇒ *c* ⇒ *bool* ⇒ *bool* (‹ - ⟦ - ⟧ ~ - ›) **where**
  eval-c-trueI: *i* ⟦ *C-true* ⟧ ~ *True*
| *eval-c-falseI*:*i* ⟦ *C-false* ⟧ ~ *False*
| *eval-c-conjI*: ⟦ *i* ⟦ *c1* ⟧ ~ *b1* ; *i* ⟦ *c2* ⟧ ~ *b2* ⟧ ⟹ *i* ⟦ (*C-conj c1 c2*) ⟧ ~ (*b1* ∧ *b2*)
| *eval-c-disjI*: ⟦ *i* ⟦ *c1* ⟧ ~ *b1* ; *i* ⟦ *c2* ⟧ ~ *b2* ⟧ ⟹ *i* ⟦ (*C-disj c1 c2*) ⟧ ~ (*b1* ∨ *b2*)
| *eval-c-impI*:⟦ *i* ⟦ *c1* ⟧ ~ *b1* ; *i* ⟦ *c2* ⟧ ~ *b2* ⟧ ⟹ *i* ⟦ (*C-imp c1 c2*) ⟧ ~ (*b1* ⟶ *b2*)
| *eval-c-notI*:⟦ *i* ⟦ *c* ⟧ ~ *b* ⟧ ⟹ *i* ⟦ (*C-not c*) ⟧ ~ (¬ *b*)
| *eval-c-eqI*:⟦ *i* ⟦ *e1* ⟧ ~ *sv1*; *i* ⟦ *e2* ⟧ ~ *sv2* ⟧ ⟹ *i* ⟦ (*C-eq e1 e2*) ⟧ ~ (*sv1*=*sv2*)
equivariance *eval-c*
nominal-inductive *eval-c* **.**

inductive-cases *eval-c-elims*:
  i ⟦ *C-true* ⟧ ~ *True*
  i ⟦ *C-false* ⟧ ~ *False*
  i ⟦ (*C-conj c1 c2*)⟧ ~ *s*
  i ⟦ (*C-disj c1 c2*)⟧ ~ *s*
  i ⟦ (*C-imp c1 c2*)⟧ ~ *s*
  i ⟦ (*C-not c*) ⟧ ~ *s*
  i ⟦ (*C-eq e1 e2*)⟧ ~ *s*
  i ⟦ *C-true* ⟧ ~ *s*
  i ⟦ *C-false* ⟧ ~ *s*

6.1.5 Satisfiability

inductive *is-satis* :: *valuation* ⇒ *c* ⇒ *bool* (‹ - ⊨ - ›) **where**
  i ⟦ *c* ⟧ ~ *True* ⟹ *i* ⊨ *c*
equivariance *is-satis*
nominal-inductive *is-satis* **.**

nominal-function *is-satis-g* :: *valuation* ⇒ Γ ⇒ *bool* (‹ - ⊨ - ›) **where**
  i ⊨ *GNil* = *True*
| *i* ⊨ ((*x*,*b*,*c*) $\#_\Gamma$ *G*) = (*i* ⊨ *c* ∧ *i* ⊨ *G*)
    apply(*auto simp*: *eqvt-def is-satis-g-graph-aux-def*)
  by (*metis* Γ.*exhaust old.prod.exhaust*)

nominal-termination (*eqvt*) **by** *lexicographic-order*

6.2 Validity

nominal-function *valid* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow c \Rightarrow bool$ (‹- ; - ; - ⊨ - › [*50*, *50*] *50*) **where**
P ; B ; $G \models c = (\ (P\ ;\ B\ ;\ G \vdash_{wf} c) \wedge (\forall i.\ (P\ ;\ G \vdash i) \wedge\ i \models G \longrightarrow i \models c))$
by (*auto simp*: *eqvt-def wfI-def valid-graph-aux-def*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

6.3 Lemmas

Lemmas needed for Examples

lemma *valid-trueI* [*intro*]:
fixes G::Γ
assumes P ; $B \vdash_{wf} G$
shows P ; B ; $G \models$ *C-true*
proof −
have $\forall i.\ i \models$ *C-true* **using** *is-satis.simps eval-c-trueI* **by** *simp*
moreover have P ; B ; $G \vdash_{wf}$ *C-true* **using** *wfC-trueI assms* **by** *simp*
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed

end

Chapter 7

Syntax Lemmas

7.1 Support, lookup and contexts

lemma *supp-v-tau* [*simp*]:
  assumes *atom z ♯ v*
  shows *supp* ({| *z* : *b* | *CE-val* (*V-var z*) == *CE-val v* |}) = *supp v* ∪ *supp b*
  using *assms τ.supp c.supp ce.supp*
  by (*simp add*: *fresh-def supp-at-base*)

lemma *supp-v-var-tau* [*simp*]:
  assumes *z* ≠ *x*
  shows *supp* ({| *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-var x*) |}) = { *atom x* } ∪ *supp b*
  using *supp-v-tau assms*
  using *supp-at-base* **by** *fastforce*

Sometimes we need to work with a version of a binder where the variable is fresh in something else, such as a bigger context. I think these could be generated automatically

lemma *obtain-fresh-fun-def*:
  fixes *t*::′*b*::*fs*
  shows ∃ *y*::*x*. *atom y* ♯ (*s*,*c*,*τ*,*t*) ∧ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)) = *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ y b* ((*y* ↔ *x*) • *c*) ((*y* ↔ *x*) • *τ*) ((*y* ↔ *x*) • *s*))))
proof −
  obtain *y*::*x* **where** *y*: *atom y* ♯ (*s*,*c*,*τ*,*t*) **using** *obtain-fresh* **by** *blast*
  moreover have *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ y b* ((*y* ↔ *x*) • *c*) ((*y* ↔ *x*) • *τ*) ((*y* ↔ *x*) • *s*))) = (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)))
  proof(*cases x*=*y*)
    case *True*
    then show *?thesis* **using** *fun-def.eq-iff Abs1-eq-iff*(*3*) *flip-commute flip-fresh-fresh fresh-PairD* **by** *auto*
  next
    case *False*

    have (*AF-fun-typ y b* ((*y* ↔ *x*) • *c*) ((*y* ↔ *x*) • *τ*) ((*y* ↔ *x*) • *s*)) =(*AF-fun-typ x b c τ s*) **proof**(*subst fun-typ.eq-iff*, *subst Abs1-eq-iff*(*3*))
      show ‹(*y* = *x* ∧ (((*y* ↔ *x*) • *c*, (*y* ↔ *x*) • *τ*), (*y* ↔ *x*) • *s*) = ((*c*, *τ*), *s*) ∨
        y ≠ *x* ∧ (((*y* ↔ *x*) • *c*, (*y* ↔ *x*) • *τ*), (*y* ↔ *x*) • *s*) = (*y* ↔ *x*) • ((*c*, *τ*), *s*) ∧ *atom y* ♯ ((*c*, *τ*), *s*)) ∧
        b = *b*› **using** *False flip-commute flip-fresh-fresh fresh-PairD y* **by** *auto*

qed
thus *?thesis* **by** *metis*
qed
ultimately show *?thesis* **using** *y fresh-Pair* **by** *metis*
qed

lemma *lookup-fun-member*:
assumes *Some* (*AF-fundef f ft*) = *lookup-fun* Φ *f*
shows *AF-fundef f ft* ∈ *set* Φ
using *assms* **proof** (*induct* Φ)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a* Φ)
then show *?case* **using** *lookup-fun.simps*
by (*metis fun-def.exhaust insert-iff list.simps*(*15*) *option.inject*)
qed

lemma *rig-dom-eq*:
dom (*G*[*x* ⟼ *c*]) = *dom G*
proof(*induct G rule*: Γ.*induct*)
case *GNil*
then show *?case* **using** *replace-in-g.simps* **by** *presburger*
next
case (*GCons xbc* Γ′)
obtain *x′* **and** *b′* **and** *c′* **where** *xbc*: *xbc*=(*x′*,*b′*,*c′*) **using** *prod-cases3* **by** *blast*
then show *?case* **using** *replace-in-g.simps GCons* **by** *simp*
qed

lemma *lookup-in-rig-eq*:
assumes *Some* (*b*,*c*) = *lookup* Γ *x*
shows *Some* (*b*,*c′*) = *lookup* (Γ[*x*⟼*c′*]) *x*
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x b c* Γ′)
then show *?case* **using** *replace-in-g.simps lookup.simps* **by** *auto*
qed

lemma *lookup-in-rig-neq*:
assumes *Some* (*b*,*c*) = *lookup* Γ *y* **and** *x*≠*y*
shows *Some* (*b*,*c*) = *lookup* (Γ[*x*⟼*c′*]) *y*
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x′ b′ c′* Γ′)
then show *?case* **using** *replace-in-g.simps lookup.simps* **by** *auto*
qed

lemma *lookup-in-rig*:

assumes *Some* (*b*,*c*) = *lookup* Γ *y*
shows ∃ *c''*. *Some* (*b*,*c''*) = *lookup* (Γ[*x*⟼*c'*]) *y*
proof(*cases x*=*y*)
case *True*
then show *?thesis* **using** *lookup-in-rig-eq* **using** *assms* **by** *blast*
next
case *False*
then show *?thesis* **using** *lookup-in-rig-neq* **using** *assms* **by** *blast*
qed

lemma *lookup-inside*[*simp*]:
assumes *x* ∉ *fst* ' *toSet* Γ'
shows *Some* (*b1*,*c1*) = *lookup* (Γ'@(*x*,*b1*,*c1*)#$_\Gamma$Γ) *x*
using *assms* **by**(*induct* Γ',*auto*)

lemma *lookup-inside2*:
assumes *Some* (*b1*,*c1*) = *lookup* (Γ'@((*x*,*b0*,*c0*)#$_\Gamma$Γ)) *y* **and** *x*≠*y*
shows *Some* (*b1*,*c1*) = *lookup* (Γ'@((*x*,*b0*,*c0'*)#$_\Gamma$Γ)) *y*
using *assms* **by**(*induct* Γ' *rule*: Γ.*induct*,*auto*+)

fun *tail*:: *'a list* ⇒ *'a list* **where**
tail [] = []
| *tail* (*x*#*xs*) = *xs*

lemma *lookup-options*:
assumes *Some* (*b*,*c*) = *lookup* (*xt*#$_\Gamma$*G*) *x*
shows ((*x*,*b*,*c*) = *xt*) ∨ (*Some* (*b*,*c*) = *lookup G x*)
by (*metis assms lookup.simps*(*2*) *option.inject surj-pair*)

lemma *lookup-x*:
assumes *Some* (*b*,*c*) = *lookup G x*
shows *x* ∈ *fst* ' *toSet G*
using *assms*
by(*induct G rule*: Γ.*induct* ,*auto*+)

lemma *GCons-eq-appendI*:
fixes *xs1*::Γ
shows [| *x* #$_\Gamma$ *xs1* = *ys*; *xs* = *xs1* @ *zs* |] ==> *x* #$_\Gamma$ *xs* = *ys* @ *zs*
by (*drule sym*) *simp*

lemma *split-G*: *x* : *toSet xs* ⟹ ∃ *ys zs*. *xs* = *ys* @ *x* #$_\Gamma$ *zs*
proof (*induct xs*)
case *GNil* **thus** *?case* **by** *simp*
next
case *GCons* **thus** *?case* **using** *GCons-eq-appendI*
by (*metis Un-iff append-g.simps*(*1*) *singletonD toSet.simps*(*2*))
qed

lemma *lookup-not-empty*:
assumes *Some* τ = *lookup G x*
shows *G* ≠ *GNil*
using *assms* **by** *auto*

lemma *lookup-in-g*:
assumes *Some* (*b*,*c*) = *lookup* Γ *x*
shows (*x*,*b*,*c*) ∈ *toSet* Γ
using *assms* **apply**(*induct* Γ, *simp*)
using *lookup-options* **by** *fastforce*

lemma *lookup-split*:
fixes Γ::Γ
assumes *Some* (*b*,*c*) = *lookup* Γ *x*
shows ∃ *G G′* . Γ = *G′*@(*x*,*b*,*c*)$\#_\Gamma$*G*
by (*meson assms*(*1*) *lookup-in-g split-G*)

lemma *toSet-splitU*[*simp*]:
(*x′*,*b′*,*c′*) ∈ *toSet* (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) ⟷ (*x′*,*b′*,*c′*) ∈ (*toSet* Γ′ ∪ {(*x*, *b*, *c*)} ∪ *toSet* Γ)
using *append-g-toSetU toSet.simps* **by** *auto*

lemma *toSet-splitP*[*simp*]:
(∀ (*x′*, *b′*, *c′*)∈*toSet* (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ). *P x′ b′ c′*) ⟷ (∀ (*x′*, *b′*, *c′*)∈*toSet* Γ′. *P x′ b′ c′*) ∧ *P x b c* ∧ (∀ (*x′*, *b′*, *c′*)∈*toSet* Γ. *P x′ b′ c′*) (**is** *?A* ⟷ *?B*)
using *toSet-splitU* **by** *force*

lemma *lookup-restrict*:
assumes *Some* (*b′*,*c′*) = *lookup* (Γ′@(*x*,*b*,*c*)$\#_\Gamma$Γ) *y* **and** *x* ≠ *y*
shows *Some* (*b′*,*c′*) = *lookup* (Γ′@Γ) *y*
using *assms* **proof**(*induct* Γ′ *rule*:Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ′)
then show *?case* **by** *auto*
qed

lemma *supp-list-member*:
fixes *x*::′*a*::*fs* **and** *l*::′*a list*
assumes *x* ∈ *set l*
shows *supp x* ⊆ *supp l*
using *assms* **apply**(*induct l*, *auto*)
using *supp-Cons* **by** *auto*

lemma *GNil-append*:
assumes *GNil* = *G1*@*G2*
shows *G1* = *GNil* ∧ *G2* = *GNil*
proof(*rule ccontr*)
assume ¬ (*G1* = *GNil* ∧ *G2* = *GNil*)
hence *G1*@*G2* ≠ *GNil* **using** *append-g.simps* **by** (*metis* Γ.*distinct*(*1*) Γ.*exhaust*)
thus *False* **using** *assms* **by** *auto*
qed

lemma *GCons-eq-append-conv*:
fixes *xs*::Γ
shows *x*$\#_\Gamma$*xs* = *ys*@*zs* = (*ys* = *GNil* ∧ *x*$\#_\Gamma$*xs* = *zs* ∨ (∃ *ys′*. *x*$\#_\Gamma$*ys′* = *ys* ∧ *xs* = *ys′*@*zs*))

by(*cases ys*) *auto*

lemma *dclist-distinct-unique*:
assumes (*dc*, *const*) ∈ *set dclist2* **and** (*cons*, *const1*) ∈ *set dclist2* **and** *dc*=*cons* **and** *distinct* (*List.map fst dclist2*)
shows (*const*) = *const1*
proof −
have (*cons*, *const*) = (*dc*, *const1*)
using *assms* **by** (*metis* (*no-types*, *lifting*) *assms*(*3*) *assms*(*4*) *distinct.simps*(*1*) *distinct.simps*(*2*) *empty-iff insert-iff list.set*(*1*) *list.simps*(*15*) *list.simps*(*8*) *list.simps*(*9*) *map-of-eq-Some-iff*)
thus *?thesis* **by** *auto*
qed

lemma *fresh-d-fst-d*:
assumes *atom u* ♯ δ
shows *u* ∉ *fst* ‘ *set* δ
using *assms* **proof**(*induct* δ)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons ut* δ′)
obtain *u*′ **and** *t*′ **where** ∗:*ut* = (*u*′,*t*′) **by** *fastforce*
hence *atom u* ♯ *ut* ∧ *atom u* ♯ δ′ **using** *fresh-Cons Cons* **by** *auto*
moreover hence *atom u* ♯ *fst ut* **using** ∗ *fresh-Pair*[*of atom u u*′ *t*′] *Cons* **by** *auto*
ultimately show *?case* **using** *Cons* **by** *auto*
qed

lemma *bv-not-in-bset-supp*:
fixes *bv*::*bv*
assumes *bv* |∉| *B*
shows *atom bv* ∉ *supp B*
proof −
have ∗:*supp B* = *fset* (*fimage atom B*)
by (*metis fimage.rep-eq finite-fset supp-finite-set-at-base supp-fset*)
thus *?thesis* **using** *assms*
by *fastforce*
qed

lemma *u-fresh-d*:
assumes *atom u* ♯ *D*
shows *u* ∉ *fst* ‘ *setD D*
using *assms* **proof**(*induct D rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u*′ *t*′ Δ′)
then show *?case* **unfolding** *setD.simps*
using *fresh-DCons fresh-Pair* **by** (*simp add*: *fresh-Pair fresh-at-base*(*2*))
qed

7.2 Type Definitions

lemma *exist-fresh-bv*:
fixes *tm*::*'a*::*fs*
shows ∃ *bva2 dclist2*. *AF-typedef-poly tyid bva dclist* = *AF-typedef-poly tyid bva2 dclist2* ∧
atom bva2 ♯ *tm*
proof −
obtain *bva2*::*bv* **where** ∗:*atom bva2* ♯ (*bva*, *dclist*,*tyid*,*tm*) **using** *obtain-fresh* **by** *metis*
moreover hence *bva2* ≠ *bva* **using** *fresh-at-base* **by** *auto*
moreover have *dclist* = (*bva* ↔ *bva2*) · (*bva2* ↔ *bva*) · *dclist* **by** *simp*
moreover have *atom bva* ♯ (*bva2* ↔ *bva*) · *dclist* **proof** −
have *atom bva2* ♯ *dclist* **using** ∗ *fresh-prodN* **by** *auto*
hence *atom* ((*bva2* ↔ *bva*) · *bva2*) ♯ (*bva2* ↔ *bva*) · *dclist* **using** *fresh-eqvt True-eqvt*
proof −
have (*bva2* ↔ *bva*) · *atom bva2* ♯ (*bva2* ↔ *bva*) · *dclist*
by (*metis True-eqvt* ‹*atom bva2* ♯ *dclist*› *fresh-eqvt*)
then show *?thesis*
by *simp*
qed
thus *?thesis* **by** *auto*
qed
ultimately have *AF-typedef-poly tyid bva dclist* = *AF-typedef-poly tyid bva2* ((*bva2* ↔ *bva*) · *dclist*)

unfolding *type-def.eq-iff Abs1-eq-iff* **by** *metis*
thus *?thesis* **using** ∗ *fresh-prodN* **by** *metis*
qed

lemma *obtain-fresh-bv*:
fixes *tm*::*'a*::*fs*
obtains *bva2*::*bv* **and** *dclist2* **where** *AF-typedef-poly tyid bva dclist* = *AF-typedef-poly tyid bva2 dclist2* ∧
atom bva2 ♯ *tm*
using *exist-fresh-bv* **by** *metis*

7.3 Function Definitions

lemma *fun-typ-flip*:
fixes *bv1*::*bv* **and** *c*::*bv*
shows (*bv1* ↔ *c*) · *AF-fun-typ x1 b1 c1 τ1 s1* = *AF-fun-typ x1* ((*bv1* ↔ *c*) · *b1*) ((*bv1* ↔ *c*) · *c1*) ((*bv1* ↔ *c*) · *τ1*) ((*bv1* ↔ *c*) · *s1*)
using *fun-typ.perm-simps flip-fresh-fresh supp-at-base fresh-def*
flip-fresh-fresh fresh-def supp-at-base
by (*simp add*: *flip-fresh-fresh*)

lemma *fun-def-eq*:
assumes *AF-fundef fa* (*AF-fun-typ-none* (*AF-fun-typ xa ba ca τa sa*)) = *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*))
shows *f* = *fa* **and** *b* = *ba* **and** [[*atom xa*]]*lst*. *sa* = [[*atom x*]]*lst*. *s* **and** [[*atom xa*]]*lst*. *τa* = [[*atom x*]]*lst*. *τ* **and**
[[*atom xa*]]*lst*. *ca* = [[*atom x*]]*lst*. *c*
using *fun-def.eq-iff fun-typ-q.eq-iff fun-typ.eq-iff lst-snd lst-fst* **using** *assms* **apply** *metis*
using *fun-def.eq-iff fun-typ-q.eq-iff fun-typ.eq-iff lst-snd lst-fst* **using** *assms* **apply** *metis*

proof −
have ([[*atom xa*]]*lst*. ((*ca*, τ*a*), *sa*) = [[*atom x*]]*lst*. ((*c*, τ), *s*)) **using** *assms fun-def.eq-iff fun-typ-q.eq-iff fun-typ.eq-iff* **by** *auto*
thus [[*atom xa*]]*lst*. *sa* = [[*atom x*]]*lst*. *s* **and** [[*atom xa*]]*lst*. τ*a* = [[*atom x*]]*lst*. τ **and** [[*atom xa*]]*lst*. *ca* = [[*atom x*]]*lst*. *c* **using** *lst-snd lst-fst* **by** *metis+*
qed

lemma *fun-arg-unique-aux*:
assumes *AF-fun-typ x1 b1 c1* τ*1' s1'* = *AF-fun-typ x2 b2 c2* τ*2' s2'*
shows ⦃ *x1* : *b1* | *c1* ⦄ = ⦃ *x2* : *b2* | *c2*⦄
proof −
have ([[*atom x1*]]*lst*. *c1* = [[*atom x2*]]*lst*. *c2*) **using** *fun-def-eq assms* **by** *metis*
moreover have *b1* = *b2* **using** *fun-typ.eq-iff assms* **by** *metis*
ultimately show *?thesis* **using** τ*.eq-iff* **by** *fast*
qed

lemma *fresh-x-neq*:
fixes *x*::*x* **and** *y*::*x*
shows *atom x* ♯ *y* = (*x* ≠ *y*)
using *fresh-at-base fresh-def* **by** *auto*

lemma *obtain-fresh-z3*:
fixes *tm*::*'b*::*fs*
obtains *z*::*x* **where** ⦃ *x* : *b* | *c* ⦄ = ⦃ *z* : *b* | *c*[*x*::=*V-var z*]$_{cv}$ ⦄ ∧ *atom z* ♯ *tm* ∧ *atom z* ♯ (*x*,*c*)
proof −
obtain *z*::*x* **and** *c'*::*c* **where** *z*:⦃ *x* : *b* | *c* ⦄ = ⦃ *z* : *b* | *c'* ⦄ ∧ *atom z* ♯ (*tm*,*x*,*c*) **using** *obtain-fresh-z2 b-of.simps* **by** *metis*
hence *c'* = *c*[*x*::=*V-var z*]$_{cv}$ **proof** −
have ([[*atom z*]]*lst*. *c'* = [[*atom x*]]*lst*. *c*) **using** *z* τ*.eq-iff* **by** *metis*
hence *c'* = (*z* ↔ *x*) · *c* **using** *Abs1-eq-iff*[*of z c' x c*] *fresh-x-neq fresh-prodN* **by** *fastforce*
also have ... = *c*[*x*::=*V-var z*]$_{cv}$
using *subst-v-c-def flip-subst-v*[*of z c x*] *z fresh-prod3* **by** *metis*
finally show *?thesis* **by** *auto*
qed
thus *?thesis* **using** *z fresh-prodN that* **by** *metis*
qed

lemma *u-fresh-v*:
fixes *u*::*u* **and** *t*::*v*
shows *atom u* ♯ *t*
by(*nominal-induct t rule*:*v.strong-induct*,*auto*)

lemma *u-fresh-ce*:
fixes *u*::*u* **and** *t*::*ce*
shows *atom u* ♯ *t*
apply(*nominal-induct t rule*:*ce.strong-induct*)
using *u-fresh-v pure-fresh*
apply (*auto simp add*: *opp.fresh ce.fresh opp.fresh opp.exhaust*)
unfolding *ce.fresh opp.fresh opp.exhaust* **by** (*simp add*: *fresh-opp-all*)

lemma *u-fresh-c*:
fixes *u*::*u* **and** *t*::*c*

shows *atom u* ♯ *t*
by(*nominal-induct t rule*:*c.strong-induct*,*auto simp add*: *c.fresh u-fresh-ce*)

lemma *u-fresh-g*:
fixes *u*::*u* **and** *t*::Γ
shows *atom u* ♯ *t*
by(*induct t rule*:Γ-*induct*, *auto simp add*: *u-fresh-b u-fresh-c fresh-GCons fresh-GNil*)

lemma *u-fresh-t*:
fixes *u*::*u* **and** *t*::τ
shows *atom u* ♯ *t*
by(*nominal-induct t rule*:τ.*strong-induct*,*auto simp add*: τ.*fresh u-fresh-c u-fresh-b*)

lemma *b-of-c-of-eq*:
assumes *atom z* ♯ τ
shows ⦃ *z* : *b-of* τ | *c-of* τ *z* ⦄ = τ
using *assms* **proof**(*nominal-induct* τ *avoiding*: *z rule*: τ.*strong-induct*)
case (*T-refined-type x1a x2a x3a*)

hence ⦃ *z* : *b-of* ⦃ *x1a* : *x2a* | *x3a* ⦄ | *c-of* ⦃ *x1a* : *x2a* | *x3a* ⦄ *z* ⦄ = ⦃ *z* : *x2a* | *x3a*[*x1a*::=*V-var* $z]_{cv}$ ⦄
using *b-of.simps c-of.simps c-of-eq* **by** *auto*
moreover have ⦃ *z* : *x2a* | *x3a*[*x1a*::=*V-var* $z]_{cv}$ ⦄ = ⦃ *x1a* : *x2a* | *x3a* ⦄ **using** *T-refined-type* τ.*fresh* **by** *auto*
ultimately show *?case* **by** *auto*
qed

lemma *fresh-d-not-in*:
assumes *atom u2* ♯ Δ′
shows *u2* ∉ *fst* ‘ *setD* Δ′
using *assms* **proof**(*induct* Δ′ *rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *simp*
next
case (*DCons u t* Δ′)
hence ∗: *atom u2* ♯ Δ′ ∧ *atom u2* ♯ (*u*,*t*)
by (*simp add*: *fresh-def supp-DCons*)
hence *u2* ∉ *fst* ‘ *setD* Δ′ **using** *DCons* **by** *auto*
moreover have *u2* ≠ *u* **using** ∗ *fresh-Pair*
by (*metis eq-fst-iff not-self-fresh*)
ultimately show *?case* **by** *simp*
qed

end

Chapter 8

Wellformedness Lemmas

8.1 Prelude

lemma *b-of-subst-bb-commute*:
$(b\text{-}of\ (\tau[bv::=b]_{\tau b})) = (b\text{-}of\ \tau)[bv::=b]_{bb}$
proof −
obtain z' **and** b' **and** c' **where** $\tau = \{\!\mid z' : b' \mid c' \mid\!\}$ **using** *obtain-fresh-z* **by** *metis*
moreover hence $(b\text{-}of\ (\tau[bv::=b]_{\tau b})) = b\text{-}of\ \{\!\mid z' : b'[bv::=b]_{bb} \mid c' \mid\!\}$ **using** *subst-tb.simps* **by** *simp*
ultimately show *?thesis* **using** *subst-tv.simps subst-tb.simps* **by** *simp*
qed

lemmas *wf-intros = wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.intros wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfF*

lemmas *freshers = fresh-prodN b.fresh c.fresh v.fresh ce.fresh fresh-GCons fresh-GNil fresh-at-base*

8.2 Strong Elimination

Inversion/elimination for well-formed polymorphic constructors

lemma *wf-strong-elim*:
fixes $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(string*\tau)$ *list*
and $\Delta::\Delta$ **and** $b::b$ **and** *ftq::fun-typ-q* **and** *ft::fun-typ* **and** *ce::ce* **and** *td::type-def* **and** $s::s$
and $tm::'a::fs$
and *cs::branch-s* **and** *css::branch-list* **and** $\Theta::\Theta$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (V\text{-}consp\ tyid\ dc\ b\ v) : b'' \Longrightarrow (\exists\ bv\ dclist\ x\ b'\ c.\ b'' = B\text{-}app\ tyid\ b\ \wedge$
$AF\text{-}typedef\text{-}poly\ tyid\ bv\ dclist \in set\ \Theta\ \wedge$
$(dc, \{\!\mid x : b' \mid c \mid\!\}) \in set\ dclist\ \wedge$
$\Theta; \mathcal{B} \vdash_{wf} b\ \wedge atom\ bv \sharp (\Theta, \mathcal{B}, \Gamma, b, v) \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb} \wedge atom\ bv \sharp tm)$
and
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow True$ **and**
$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow True$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow True$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow True$ **and**
$\vdash_{wf} \Theta \Longrightarrow True$ **and**
$\Theta; \mathcal{B} \vdash_{wf} b \Longrightarrow True$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b' \Longrightarrow True$ **and**
$\Theta \vdash_{wf} td \Longrightarrow True$
proof(*nominal-induct*

V-consp tyid dc b v b'' **and** *c* **and** Γ **and** *τ* **and** *ts* **and** Θ **and** *b* **and** *b'* **and** *td*
avoiding: *tm*

rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-conspI bv dclist* Θ *x b' c* 𝓑 Γ)
then show *?case* **by** *force*
qed(*auto*+)

8.3 Context Extension

definition *wfExt* :: Θ ⇒ 𝓑 ⇒ Γ ⇒ Γ ⇒ *bool* (‹ - ; - $\vdash_{wf}$ - < - › [*50*,*50*,*50*] *50*) **where**
wfExt T B G1 G2 = (*wfG T B G2* ∧ *wfG T B G1* ∧ *toSet G1* ⊆ *toSet G2*)

8.4 Context

lemma *wfG-cons*[*ms-wb*]:
fixes Γ::Γ
assumes *P*; 𝓑 $\vdash_{wf}$ (*z*,*b*,*c*) #$_{\Gamma}$Γ
shows *P*; 𝓑 $\vdash_{wf}$ Γ ∧ *atom z* ♯ Γ ∧ *wfB P* 𝓑 *b*
using *wfG-elims*(*2*)[*OF assms*] **by** *metis*

lemma *wfG-cons2*[*ms-wb*]:
fixes Γ::Γ
assumes *P*; 𝓑 $\vdash_{wf}$ *zbc* #$_{\Gamma}$Γ
shows *P*; 𝓑 $\vdash_{wf}$ Γ
proof −
obtain *z* **and** *b* **and** *c* **where** *zbc*: *zbc*=(*z*,*b*,*c*) **using** *prod-cases3* **by** *blast*
hence *P*; 𝓑 $\vdash_{wf}$ (*z*,*b*,*c*) #$_{\Gamma}$Γ **using** *assms* **by** *auto*
thus *?thesis* **using** *zbc wfG-cons assms* **by** *simp*
qed

lemma *wf-g-unique*:
fixes Γ::Γ
assumes Θ; 𝓑 $\vdash_{wf}$ Γ **and** (*x*,*b*,*c*) ∈ *toSet* Γ **and** (*x*,*b'*,*c'*) ∈ *toSet* Γ
shows *b*=*b'* ∧ *c*=*c'*
using *assms* **proof**(*induct* Γ *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *simp*
next
case (*GCons a* Γ)
consider (*x*,*b*,*c*)=*a* ∧ (*x*,*b'*,*c'*)=*a* | (*x*,*b*,*c*)=*a* ∧ (*x*,*b'*,*c'*)≠*a* | (*x*,*b*,*c*)≠*a* ∧ (*x*,*b'*,*c'*)=*a* | (*x*,*b*,*c*)≠ *a* ∧ (*x*,*b'*,*c'*)≠*a* **by** *blast*
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **by** *auto*
next
case *2*
hence *atom x* ♯ Γ **using** *wfG-elims*(*2*) *GCons* **by** *blast*
moreover have (*x*,*b'*,*c'*) ∈ *toSet* Γ **using** *GCons 2* **by** *force*
ultimately show *?thesis* **using** *forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem* Γ.*distinct subst-gv.simps 2 GCons* **by** *metis*

next
case *3*
hence *atom x* $\sharp$ Γ **using** *wfG-elims*(*2*) *GCons* **by** *blast*
moreover have (*x*,*b*,*c*) ∈ *toSet* Γ **using** *GCons 3* **by** *force*
ultimately show *?thesis*
using *forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem* Γ.*distinct subst-gv.simps 3 GCons* **by** *metis*
next
case *4*
then obtain x'' **and** b'' **and** c''::*c* **where** *xbc*: $a=(x'',b'',c'')$
using *prod-cases3* **by** *blast*
hence Θ; $\mathcal{B} \vdash_{wf}$ $((x'',b'',c'')\ \ \#_\Gamma\Gamma)$ **using** *GCons wfG-elims* **by** *blast*
hence Θ; $\mathcal{B} \vdash_{wf}$ Γ ∧ (*x*, *b*, *c*) ∈ *toSet* Γ ∧ (*x*, b', c') ∈ *toSet* Γ **using** *GCons wfG-elims 4 xbc prod-cases3 set-GConsD* **using** *forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem* Γ.*distinct subst-gv.simps 4 GCons* **by** *meson*
thus *?thesis* **using** *GCons* **by** *auto*
qed
qed

lemma *lookup-if1*:
fixes Γ::Γ
assumes Θ; $\mathcal{B} \vdash_{wf}$ Γ **and** *Some* (*b*,*c*) = *lookup* Γ *x*
shows (*x*,*b*,*c*) ∈ *toSet* Γ ∧ (∀ b' c'. (*x*,b',c') ∈ *toSet* Γ ⟶ b'=*b* ∧ c'=*c*)
using *assms* **proof**(*induct* Γ *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc* Γ)
then obtain x' **and** b' **and** c'::*c* **where** *xbc*: *xbc*=(x',b',c')
using *prod-cases3* **by** *blast*
then show *?case* **using** *wf-g-unique GCons lookup-in-g xbc*
lookup.simps set-GConsD wfG.cases
insertE insert-is-Un toSet.simps wfG-elims **by** *metis*
qed

lemma *lookup-if2*:
assumes *wfG P* $\mathcal{B}$ Γ **and** (*x*,*b*,*c*) ∈ *toSet* Γ ∧ (∀ b' c'. (*x*,b',c') ∈ *toSet* Γ ⟶ b'=*b* ∧ c'=*c*)
shows *Some* (*b*,*c*) = *lookup* Γ *x*
using *assms* **proof**(*induct* Γ *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc* Γ)
then obtain x' **and** b' **and** c'::*c* **where** *xbc*: *xbc*=(x',b',c')
using *prod-cases3* **by** *blast*
then show *?case* **proof**(*cases x*=x')
case *True*
then show *?thesis* **using** *lookup.simps GCons xbc* **by** *simp*
next
case *False*
then show *?thesis* **using** *lookup.simps GCons xbc toSet.simps Un-iff set-GConsD wfG-cons2*
by (*metis* (*full-types*) *Un-iff set-GConsD toSet.simps*(*2*) *wfG-cons2*)

qed
qed

lemma *lookup-iff*:
fixes Θ::Θ **and** Γ::Γ
assumes Θ; $\mathcal{B} \vdash_{wf} \Gamma$
shows *Some* (b,c) = *lookup* Γ $x \longleftrightarrow (x,b,c) \in$ *toSet* $\Gamma \wedge (\forall b'\ c'.\ (x,b',c') \in$ *toSet* $\Gamma \longrightarrow b'=b \wedge c'=c)$
using *assms lookup-if1 lookup-if2* **by** *meson*

lemma *wfG-lookup-wf*:
fixes Θ::Θ **and** Γ::Γ **and** b::b **and** $\mathcal{B}$::$\mathcal{B}$
assumes Θ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *Some* (b,c) = *lookup* Γ x
shows Θ; $\mathcal{B} \vdash_{wf} b$
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* x' b' c' Γ')
then show *?case* **proof**(*cases* $x=x'$)
case *True*
then show *?thesis* **using** *lookup.simps wfG-elims*(*2*) *GCons* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *lookup.simps wfG-elims*(*2*) *GCons* **by** *fastforce*
qed
qed

lemma *wfG-unique*:
fixes Γ::Γ
assumes *wfG B* Θ $((x,\ b,\ c)\quad \#_\Gamma\ \Gamma)$ **and** $(x1,b1,c1) \in$ *toSet* $((x,\ b,\ c)\quad \#_\Gamma\ \Gamma)$ **and** $x1=x$
shows $b1 = b \wedge c1 = c$
proof −
have $(x,\ b,\ c) \in$ *toSet* $((x,\ b,\ c)\quad \#_\Gamma\ \Gamma)$ **by** *simp*
thus *?thesis* **using** *wf-g-unique assms* **by** *blast*
qed

lemma *wfG-unique-full*:
fixes Γ::Γ
assumes *wfG* Θ *B* $(\Gamma'@(x,\ b,\ c)\quad \#_\Gamma\ \Gamma)$ **and** $(x1,b1,c1) \in$ *toSet* $(\Gamma'@(x,\ b,\ c)\quad \#_\Gamma\ \Gamma)$ **and** $x1=x$
shows $b1 = b \wedge c1 = c$
proof −
have $(x,\ b,\ c) \in$ *toSet* $(\Gamma'@(x,\ b,\ c)\quad \#_\Gamma\ \Gamma)$ **by** *simp*
thus *?thesis* **using** *wf-g-unique assms* **by** *blast*
qed

8.5 Converting between wb forms

We cannot prove wfB properties here for expressions and statements as need some more facts about Φ context which we can prove without this lemma. Trying to cram everything into a single large mutually recursive lemma is not a good idea

lemma *wfX-wfY1*:

fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows *wfV-wf*: Θ; *B*; Γ $\vdash_{wf}$ *v* : *b* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ $\vdash_{wf}$ Θ **and**
wfC-wf: Θ; *B*; Γ $\vdash_{wf}$ *c* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ $\vdash_{wf}$ Θ **and**
wfG-wf :Θ; *B* $\vdash_{wf}$ Γ ⟹ $\vdash_{wf}$ Θ **and**
wfT-wf: Θ; *B*; Γ $\vdash_{wf}$ τ ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ $\vdash_{wf}$ Θ ∧ Θ; *B* $\vdash_{wf}$ *b-of* τ **and**
wfTs-wf:Θ; *B*; Γ $\vdash_{wf}$ *ts* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ $\vdash_{wf}$ Θ **and**
$\vdash_{wf}$ Θ ⟹ *True* **and**
wfB-wf: Θ; *B* $\vdash_{wf}$ *b* ⟹ $\vdash_{wf}$ Θ **and**
wfCE-wf: Θ; *B*; Γ $\vdash_{wf}$ *ce* : *b* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ $\vdash_{wf}$ Θ **and**
wfTD-wf: Θ $\vdash_{wf}$ *td* ⟹ $\vdash_{wf}$ Θ
proof(*induct rule*:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)

case (*wfV-varI* Θ *B* Γ *b c x*)
hence (*x*,*b*,*c*) ∈ *toSet* Γ **using** *lookup-iff lookup-in-g* **by** *presburger*
hence *b* ∈ *fst‘snd‘toSet* Γ **by** *force*
hence *wfB* Θ *B b* **using** *wfV-varI* **using** *wfG-lookup-wf* **by** *auto*
then show *?case* **using** *wfV-varI wfV-elims wf-intros* **by** *metis*
next
case (*wfV-litI* Θ *B* Γ *l*)
moreover have *wfTh* Θ **using** *wfV-litI* **by** *metis*
ultimately show *?case* **using** *wf-intros base-for-lit.simps l.exhaust* **by** *metis*
next
case (*wfV-pairI* Θ *B* Γ *v1 b1 v2 b2*)
then show *?case* **using** *wfB-pairI* **by** *simp*
next
case (*wfV-consI s dclist* Θ *dc x b c B* Γ *v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTI z* Γ Θ *B b c*)
then show *?case* **using** *wf-intros b-of.simps wfG-cons2* **by** *metis*
qed(*auto*)

lemma *wfX-wfY2*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows
wfE-wf: Θ; Φ; *B*; Γ; Δ $\vdash_{wf}$ *e* : *b* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ Θ; *B*; Γ $\vdash_{wf}$ Δ ∧ $\vdash_{wf}$ Θ ∧ Θ $\vdash_{wf}$ Φ **and**
wfS-wf: Θ; Φ; *B*; Γ; Δ $\vdash_{wf}$ *s* : *b* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ Θ; *B*; Γ $\vdash_{wf}$ Δ ∧ $\vdash_{wf}$ Θ ∧ Θ $\vdash_{wf}$ Φ **and**
Θ; Φ; *B*; Γ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ Θ; *B*; Γ $\vdash_{wf}$ Δ ∧ $\vdash_{wf}$ Θ ∧ Θ $\vdash_{wf}$ Φ **and**
Θ; Φ; *B*; Γ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ Θ; *B*; Γ $\vdash_{wf}$ Δ ∧ $\vdash_{wf}$ Θ ∧ Θ $\vdash_{wf}$ Φ **and**
wfPhi-wf: Θ $\vdash_{wf}$ (Φ::Φ) ⟹ $\vdash_{wf}$ Θ **and**
wfD-wf: Θ; *B*; Γ $\vdash_{wf}$ Δ ⟹ Θ; *B* $\vdash_{wf}$ Γ ∧ $\vdash_{wf}$ Θ **and**
wfFTQ-wf: Θ ; Φ $\vdash_{wf}$ *ftq* ⟹ Θ $\vdash_{wf}$ Φ ∧ $\vdash_{wf}$ Θ **and**
wfFT-wf: Θ ; Φ ; *B* $\vdash_{wf}$ *ft* ⟹ Θ $\vdash_{wf}$ Φ ∧ $\vdash_{wf}$ Θ
proof(*induct rule*:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfS-varI* Θ *B* Γ τ *v u* Δ Φ *s b*)
then show *?case* **using** *wfD-elims* **by** *auto*

```
next
  case (wfS-assignI u τ Δ Θ ℬ Γ Φ v)
  then show ?case using wf-intros by metis
next
  case (wfD-emptyI Θ ℬ Γ)
  then show ?case using wfX-wfY1 by auto
next
  case (wfS-assertI Θ Φ ℬ x c Γ Δ s b)
  then have Θ; ℬ ⊢wf Γ using wfX-wfY1 by auto
  moreover have Θ; ℬ; Γ ⊢wf Δ using wfS-assertI by auto
  moreover have ⊢wf Θ ∧ Θ ⊢wf Φ using wfS-assertI by auto
  ultimately show ?case by auto
qed(auto)

lemmas wfX-wfY=wfX-wfY1 wfX-wfY2

lemma setD-ConsD:
  ut ∈ setD (ut′ #Δ D) = (ut = ut′ ∨ ut ∈ setD D)
proof(induct D rule: Δ-induct)
  case DNil
  then show ?case by auto
next
  case (DCons u′ t′ x2)
  then show ?case using setD.simps by auto
qed

lemma wfD-wfT:
  fixes Δ::Δ and τ::τ
  assumes Θ; ℬ; Γ ⊢wf Δ
  shows ∀(u,τ) ∈ setD Δ. Θ; ℬ; Γ ⊢wf τ
using assms proof(induct Δ rule: Δ-induct)
  case DNil
  then show ?case by auto
next
  case (DCons u′ t′ x2)
  then show ?case using wfD-elims DCons setD-ConsD
    by (metis case-prodI2 set-ConsD)
qed

lemma subst-b-lookup-d:
  assumes u ∉ fst ‘ setD Δ
  shows  u ∉ fst ‘ setD Δ[bv::=b]Δb
using assms proof(induct Δ rule: Δ-induct)
  case DNil
  then show ?case by auto
next
  case (DCons u′ t′  x2)
  hence u≠u′ using DCons by simp
  show ?case using DCons subst-db.simps by simp
qed

lemma wfG-cons-splitI:
```

fixes Φ::Φ **and** Γ::Γ
assumes $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ **and** *atom x* $\sharp$ Γ **and** *wfB* Θ $\mathcal{B}$ *b* **and**
$c \in \{ \; TRUE, \; FALSE \; \} \longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma$ **and**
$c \notin \{ \; TRUE, \; FALSE \; \} \longrightarrow \Theta \; ;\mathcal{B} \; ; \; (x,b,C\text{-}true) \; \#_{\Gamma}\Gamma \vdash_{wf} c$
shows $\Theta; \mathcal{B} \vdash_{wf} ((x,b,c) \; \#_{\Gamma}\Gamma)$
using *wfG-cons1I wfG-cons2I assms* **by** *metis*

lemma *wfG-consI*:
fixes Φ::Φ **and** Γ::Γ **and** *c*::*c*
assumes $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ **and** *atom x* $\sharp$ Γ **and** *wfB* Θ $\mathcal{B}$ *b* **and**
$\Theta \; ; \mathcal{B} \; ; (x,b,C\text{-}true) \; \#_{\Gamma}\Gamma \vdash_{wf} c$
shows $\Theta \; ; \mathcal{B} \vdash_{wf} ((x,b,c) \; \#_{\Gamma}\Gamma)$
using *wfG-cons1I wfG-cons2I wfG-cons-splitI wfC-trueI assms* **by** *metis*

lemma *wfG-elim2*:
fixes *c*::*c*
assumes *wfG P* $\mathcal{B}$ $((x,b,c) \; \#_{\Gamma}\Gamma)$
shows $P; \mathcal{B} \; ; (x, \; b, \; TRUE) \; \#_{\Gamma} \; \Gamma \; \vdash_{wf} c \wedge$ *wfB P* $\mathcal{B}$ *b*
proof(*cases c* $\in \{TRUE,FALSE\}$)
case *True*
have $P; \mathcal{B} \vdash_{wf} \Gamma \; \wedge$ *atom x* $\sharp$ $\Gamma \wedge$ *wfB P* $\mathcal{B}$ *b* **using** *wfG-elims*(*2*)[*OF assms*] **by** *auto*
hence $P; \mathcal{B} \vdash_{wf} ((x,b,TRUE) \; \#_{\Gamma}\Gamma) \wedge$ *wfB P* $\mathcal{B}$ *b* **using** *wfG-cons2I* **by** *auto*
thus *?thesis* **using** *wfC-trueI wfC-falseI True* **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-elims*(*2*)[*OF assms*] **by** *auto*
qed

lemma *wfG-cons-wfC*:
fixes Γ::Γ **and** *c*::*c*
assumes $\Theta \; ; \; B \; \vdash_{wf} (x, \; b, \; c) \quad \#_{\Gamma} \; \Gamma$
shows $\Theta \; ; \; B \; ; ((x, \; b, \; TRUE) \quad \#_{\Gamma} \; \Gamma) \vdash_{wf} c$
using *assms wfG-elim2* **by** *auto*

lemma *wfG-wfB*:
assumes *wfG P* $\mathcal{B}$ Γ **and** *b* $\in$ *fst'snd'toSet* Γ
shows *wfB P* $\mathcal{B}$ *b*
using *assms* **proof**(*induct* Γ *rule*:Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* x' b' c' Γ')
show *?case* **proof**(*cases* $b{=}b'$)
case *True*
then show *?thesis* **using** *wfG-elim2 GCons* **by** *auto*
next
case *False*
hence *b* $\in$ *fst'snd'toSet* Γ' **using** *GCons* **by** *auto*
moreover have *wfG P* $\mathcal{B}$ Γ' **using** *wfG-cons GCons* **by** *auto*
ultimately show *?thesis* **using** *GCons* **by** *auto*
qed
qed

lemma *wfG-cons-TRUE*:
fixes Γ::Γ **and** *b*::*b*
assumes *P*; ℬ $\vdash_{wf}$ Γ **and** *atom z* ♯ Γ **and** *P*; ℬ $\vdash_{wf}$ *b*
shows *P* ; ℬ $\vdash_{wf}$ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ
using *wfG-cons2I wfG-wfB assms* **by** *simp*

lemma *wfG-cons-TRUE2*:
assumes *P*; ℬ $\vdash_{wf}$ (*z*,*b*,*c*) $\#_\Gamma$Γ **and** *atom z* ♯ Γ
shows *P*; ℬ $\vdash_{wf}$ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ
using *wfG-cons wfG-cons2I assms* **by** *simp*

lemma *wfG-suffix*:
fixes Γ::Γ
assumes *wfG P* ℬ (Γ′@Γ)
shows *wfG P* ℬ Γ
using *assms* **proof**(*induct* Γ′ *rule*: Γ*-induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x b c* Γ′)
hence *P*; ℬ $\vdash_{wf}$ Γ′ @ Γ **using** *wfG-elims* **by** *auto*
then show *?case* **using** *GCons wfG-elims* **by** *auto*
qed

lemma *wfV-wfCE*:
fixes *v*::*v*
assumes Θ; ℬ; Γ $\vdash_{wf}$ *v* : *b*
shows Θ ; ℬ ; Γ $\vdash_{wf}$ *CE-val v* : *b*
proof −
have Θ $\vdash_{wf}$ ([]::Φ) **using** *wfPhi-emptyI wfV-wf wfG-wf assms* **by** *metis*
moreover have Θ; ℬ; Γ $\vdash_{wf}$ $[]_\Delta$ **using** *wfD-emptyI wfV-wf wfG-wf assms* **by** *metis*
ultimately show *?thesis* **using** *wfCE-valI assms* **by** *auto*
qed

8.6 Support

lemma *wf-supp1*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css* ::*branch-list*

shows *wfV-supp*: Θ; ℬ; Γ $\vdash_{wf}$ *v* : *b* ⟹ *supp v* ⊆ *atom-dom* Γ ∪ *supp* ℬ **and**
wfC-supp: Θ; ℬ; Γ $\vdash_{wf}$ *c* ⟹ *supp c* ⊆ *atom-dom* Γ ∪ *supp* ℬ **and**
wfG-supp: Θ; ℬ $\vdash_{wf}$ Γ ⟹ *atom-dom* Γ ⊆ *supp* Γ **and**
wfT-supp: Θ; ℬ; Γ $\vdash_{wf}$ τ ⟹ *supp* τ ⊆ *atom-dom* Γ ∪ *supp* ℬ **and**
wfTs-supp: Θ; ℬ; Γ $\vdash_{wf}$ *ts* ⟹ *supp ts* ⊆ *atom-dom* Γ ∪ *supp* ℬ **and**
wfTh-supp: $\vdash_{wf}$ Θ ⟹ *supp* Θ = {} **and**
wfB-supp: Θ; ℬ $\vdash_{wf}$ *b* ⟹ *supp b* ⊆ *supp* ℬ **and**
wfCE-supp: Θ; ℬ; Γ $\vdash_{wf}$ *ce* : *b* ⟹ *supp ce* ⊆ *atom-dom* Γ ∪ *supp* ℬ **and**
wfTD-supp: Θ $\vdash_{wf}$ *td* ⟹ *supp td* ⊆ {}
proof(*induct rule*:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)

case (*wfB-consI* Θ *s dclist* $\mathcal{B}$)
then show *?case* **by**(*auto simp add*: *b.supp pure-supp*)
next
case (*wfB-appI* Θ $\mathcal{B}$ *b s bv dclist*)
then show *?case* **by**(*auto simp add*: *b.supp pure-supp*)
next
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c x*)
then show *?case* **using** *v.supp wfV-elims*
empty-subsetI insert-subset supp-at-base
fresh-dom-free2 lookup-if1
by (*metis sup.coboundedI1*)
next
case (*wfV-litI* Θ $\mathcal{B}$ Γ *l*)
then show *?case* **using** *supp-l-empty v.supp* **by** *simp*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
then show *?case* **using** *v.supp wfV-elims* **by** (*metis Un-subset-iff*)
next
case (*wfV-consI s dclist* Θ *dc x b c* $\mathcal{B}$ Γ *v*)
then show *?case* **using** *v.supp wfV-elims*
Un-commute b.supp sup-bot.right-neutral supp-b-empty pure-supp **by** *metis*
next
case (*wfV-conspI typid bv dclist* Θ *dc x b′ c* $\mathcal{B}$ Γ *v b*)
then show *?case* **unfolding** *v.supp*
using *wfV-elims*
Un-commute b.supp sup-bot.right-neutral supp-b-empty pure-supp
by (*simp add*: *Un-commute pure-supp sup.coboundedI1*)
next
case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
hence *supp e1* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *c.supp wfC-elims*
image-empty list.set(*1*) *sup-bot.right-neutral* **by** (*metis IntI UnE empty-iff subsetCE subsetI*)
moreover have *supp e2* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *c.supp wfC-elims*
image-empty list.set(*1*) *sup-bot.right-neutral IntI UnE empty-iff subsetCE subsetI*
by (*metis wfC-eqI.hyps*(*4*))
ultimately show *?case* **using** *c.supp* **by** *auto*
next
case (*wfG-cons1I c* Θ $\mathcal{B}$ Γ *x b*)
then show *?case* **using** *atom-dom.simps dom-supp-g supp-GCons* **by** *metis*
next
case (*wfG-cons2I c* Θ $\mathcal{B}$ Γ *x b*)
then show *?case* **using** *atom-dom.simps dom-supp-g supp-GCons* **by** *metis*
next
case *wfTh-emptyI*
then show *?case* **by** (*simp add*: *supp-Nil*)
next
case (*wfTh-consI* Θ *lst*)
then show *?case* **using** *supp-Cons* **by** *fast*
next
case (*wfTD-simpleI* Θ *lst s*)
then have *supp* (*AF-typedef s lst*) = *supp lst* ∪ *supp s* **using** *type-def.supp* **by** *auto*
then show *?case* **using** *wfTD-simpleI pure-supp*
by (*simp add*: *pure-supp supp-Cons supp-at-base*)

next
  case (*wfTD-poly* Θ *bv lst s*)
  then have *supp* (*AF-typedef-poly s bv lst*) = *supp lst* − { *atom bv* } ∪ *supp s* **using** *type-def.supp* **by** *auto*
  then show *?case* **using** *wfTD-poly pure-supp*
    by (*simp add*: *pure-supp supp-Cons supp-at-base*)
next
  case (*wfTs-nil* Θ *B* Γ)
  then show *?case* **using** *supp-Nil* **by** *auto*
next
  case (*wfTs-cons* Θ *B* Γ τ *dc ts*)
  then show *?case* **using** *supp-Cons supp-Pair pure-supp*[*of dc*] **by** *blast*
next
  case (*wfCE-valI* Θ *B* Γ *v b*)
  thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
  case (*wfCE-plusI* Θ *B* Γ *v1 v2*)
  hence *supp* (*CE-op Plus v1 v2*) ⊆ *atom-dom* Γ ∪ *supp B* **using** *ce.supp pure-supp*
    by (*simp add*: *wfCE-plusI opp.supp*)
  then show *?case* **using** *ce.supp wfCE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
  case (*wfCE-leqI* Θ *B* Γ *v1 v2*)
  hence *supp* (*CE-op LEq v1 v2*) ⊆ *atom-dom* Γ ∪ *supp B* **using** *ce.supp pure-supp*
    by (*simp add*: *wfCE-plusI opp.supp*)
  then show *?case* **using** *ce.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
  case (*wfCE-eqI* Θ *B* Γ *v1 b v2*)
  hence *supp* (*CE-op Eq v1 v2*) ⊆ *atom-dom* Γ ∪ *supp B* **using** *ce.supp pure-supp*
    by (*simp add*: *wfCE-eqI opp.supp*)
  then show *?case* **using** *ce.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
  case (*wfCE-fstI* Θ *B* Γ *v1 b1 b2*)
  thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
  case (*wfCE-sndI* Θ *B* Γ *v1 b1 b2*)
 thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
  case (*wfCE-concatI* Θ *B* Γ *v1 v2*)
  thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
  case (*wfCE-lenI* Θ *B* Γ *v1*)
  thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
   case (*wfTI z* Θ *B* Γ *b c*)
  hence *supp c* ⊆ *supp z* ∪ *atom-dom* Γ ∪ *supp B* **using** *supp-at-base dom-cons* **by** *metis*
  moreover have *supp b* ⊆ *supp B* **using** *wfTI* **by** *auto*
  ultimately have *supp* ⦃ *z* : *b* | *c* ⦄ ⊆ *atom-dom* Γ ∪ *supp B* **using** *τ.supp supp-at-base* **by** *force*
  thus *?case* **by** *auto*
qed(*auto*)

lemma *wf-supp2*:
  fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and**

ts::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and**
ft::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css* ::*branch-list*
shows
wfE-supp: Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *e* : *b* ⟹ (*supp e* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ ∪ *atom* ' *fst* ' *setD* Δ)
and
wfS-supp: Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *s* : *b* ⟹ *supp s* ⊆ *atom-dom* Γ ∪ *atom* ' *fst* ' *setD* Δ ∪ *supp* $\mathcal{B}$
and
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* ⟹ *supp cs* ⊆ *atom-dom* Γ ∪ *atom* ' *fst* ' *setD* Δ ∪ *supp* $\mathcal{B}$ **and**
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* ⟹ *supp css* ⊆ *atom-dom* Γ ∪ *atom* ' *fst* ' *setD* Δ ∪ *supp* $\mathcal{B}$ **and**
wfPhi-supp: Θ $\vdash_{wf}$ (Φ::Φ) ⟹ *supp* Φ = {} **and**
wfD-supp: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ ⟹ *supp* Δ ⊆ *atom*'*fst*'(*setD* Δ) ∪ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* ⟹ *supp ftq* = {} **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* ⟹ *supp ft* ⊆ *supp* $\mathcal{B}$
proof(*induct rule*:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
hence *supp* (*AE-val v*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp wf-supp1* **by** *simp*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-op Plus v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$
using *wfE-plusI opp.supp wf-supp1 e.supp pure-supp Un-least*
by (*metis sup-bot.left-neutral*)

then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-op LEq v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp Un-least*
sup-bot.left-neutral **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b v2*)
hence *supp* (*AE-op Eq v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp Un-least*
sup-bot.left-neutral **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
hence *supp* (*AE-fst v1*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp sup-bot.left-neutral* **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
hence *supp* (*AE-snd v1*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp wfE-plusI opp.supp wf-supp1* **by** (*metis Un-least*)
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-concat v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp*
wfE-plusI opp.supp wf-supp1 **by** (*metis Un-least*)
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next

case (*wfE-splitI* Θ Φ *ℬ* Γ Δ *v1 v2*)
hence *supp* (*AE-split v1 v2*) ⊆ *atom-dom* Γ ∪ *supp ℬ* **using** *e.supp pure-supp wfE-plusI opp.supp wf-supp1* **by** (*metis Un-least*)
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-lenI* Θ Φ *ℬ* Γ Δ *v1*)
hence *supp* (*AE-len v1*) ⊆ *atom-dom* Γ ∪ *supp ℬ* **using** *e.supp pure-supp* **using** *e.supp pure-supp sup-bot.left-neutral* **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-appI* Θ Φ *ℬ* Γ Δ *f x b c* τ *s v*)
then obtain *b* **where** Θ; *ℬ*; Γ $\vdash_{wf}$ *v* : *b* **using** *wfE-elims* **by** *metis*
hence *supp v* ⊆ *atom-dom* Γ ∪ *supp ℬ* **using** *wfE-appI wf-supp1* **by** *metis*
hence *supp* (*AE-app f v*) ⊆ *atom-dom* Γ ∪ *supp ℬ* **using** *e.supp pure-supp* **by** *fast*
then show *?case* **using** *e.supp*(*2*) *UnCI subsetCE subsetI wfE-appI* **using** *b.supp*(*3*) *pure-supp x-not-in-b-set* **by** *metis*
next
case (*wfE-appPI* Θ Φ *ℬ* Γ Δ *b′ bv v* τ *f xa ba ca s*)
then obtain *b* **where** Θ; *ℬ*; Γ $\vdash_{wf}$ *v* : (*b*[*bv*::=*b′*]$_b$) **using** *wfE-elims* **by** *metis*
hence *supp v* ⊆ *atom-dom* Γ ∪ *supp ℬ* **using** *wfE-appPI wf-supp1* **by** *auto*
moreover have *supp b′* ⊆ *supp ℬ* **using** *wf-supp1*(*7*) *wfE-appPI* **by** *simp*
ultimately show *?case* **unfolding** *e.supp* **using** *wfE-appPI pure-supp* **by** *fast*
next
case (*wfE-mvarI* Θ Φ *ℬ* Γ Δ *u* τ)
then obtain τ **where** (*u*,τ) ∈ *setD* Δ **using** *wfE-elims*(*10*) **by** *metis*
hence *atom u* ∈ *atom‘fst‘setD* Δ **by** *force*
hence *supp* (*AE-mvar u*) ⊆ *atom‘fst‘setD* Δ **using** *e.supp* **by** (*simp add*: *supp-at-base*)
thus *?case* **using** *UnCI subsetCE subsetI e.supp wfE-mvarI supp-at-base subsetCE supp-at-base u-not-in-b-set*

by (*simp add*: *supp-at-base*)
next
case (*wfS-valI* Θ Φ *ℬ* Γ *v b* Δ)
then show *?case* **using** *wf-supp1*
by (*metis s-branch-s-branch-list.supp*(*1*) *sup.coboundedI2 sup-assoc sup-commute*)
next
case (*wfS-letI* Θ Φ *ℬ* Γ Δ *e b′ x s b*)
then show *?case* **by** *auto*
next
case (*wfS-let2I* Θ Φ *ℬ* Γ Δ *s1* τ *x s2 b*)
then show *?case* **unfolding** *s-branch-s-branch-list.supp* (*3*) **using** *wf-supp1*(*4*)[*OF wfS-let2I*(*3*)] **by** *auto*
next
case (*wfS-ifI* Θ *ℬ* Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wf-supp1*(*1*)[*OF wfS-ifI*(*1*)] **by** *auto*
next
case (*wfS-varI* Θ *ℬ* Γ τ *v u* Δ Φ *s b*)
then show *?case* **using** *wf-supp1*(*1*)[*OF wfS-varI*(*2*)] *wf-supp1*(*4*)[*OF wfS-varI*(*1*)] **by** *auto*
next
next
case (*wfS-assignI u* τ Δ Θ *ℬ* Γ Φ *v*)
hence *supp u* ⊆ *atom* ‘ *fst* ‘ *setD* Δ **proof**(*induct* Δ *rule*:Δ*-induct*)

case *DNil*
then show *?case* **by** *auto*
next
case $(DCons\ u'\ t'\ \Delta')$
show *?case* **proof**$(cases\ u{=}u')$
case *True*
then show *?thesis* **using** *toSet.simps DCons supp-at-base* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *toSet.simps DCons supp-at-base wfS-assignI*
by (*metis empty-subsetI fstI image-eqI insert-subset*)
qed
qed
then show *?case* **using** *s-branch-s-branch-list.supp*(*8*) *wfS-assignI wf-supp1*(*1*)[*OF wfS-assignI*(*6*)] **by** *auto*
next
case $(wfS\text{-}matchI\ \Theta\ \mathcal{B}\ \Gamma\ v\ tid\ dclist\ \Delta\ \Phi\ cs\ b)$
then show *?case* **using** *wf-supp1*(*1*)[*OF wfS-matchI*(*1*)] **by** *auto*
next
case $(wfS\text{-}branchI\ \Theta\ \Phi\ \mathcal{B}\ x\ \tau\ \Gamma\ \Delta\ s\ b\ tid\ dc)$
moreover have $supp\ s \subseteq supp\ x \cup atom\text{-}dom\ \Gamma \cup atom\ \text{`}\ fst\ \text{`}\ setD\ \Delta \cup supp\ \mathcal{B}$
using *dom-cons supp-at-base wfS-branchI* **by** *auto*
moreover hence $supp\ s - set\ [atom\ x] \subseteq atom\text{-}dom\ \Gamma \cup atom\ \text{`}\ fst\ \text{`}\ setD\ \Delta \cup supp\ \mathcal{B}$ **using** *supp-at-base* **by** *force*
ultimately have
$(supp\ s - set\ [atom\ x]) \cup (supp\ dc\ ) \subseteq atom\text{-}dom\ \Gamma \cup atom\ \text{`}\ fst\ \text{`}\ setD\ \Delta \cup supp\ \mathcal{B}$
by (*simp add*: *pure-supp*)
thus *?case* **using** *s-branch-s-branch-list.supp*(*2*) **by** *auto*
next
case $(wfD\text{-}emptyI\ \Theta\ \mathcal{B}\ \Gamma)$
then show *?case* **using** *supp-DNil* **by** *auto*
next
case $(wfD\text{-}cons\ \Theta\ \mathcal{B}\ \Gamma\ \Delta\ \tau\ u)$
have $supp\ ((u, \tau)\ \#_{\Delta}\ \Delta) = supp\ u \cup supp\ \tau \cup supp\ \Delta$ **using** *supp-DCons supp-Pair* **by** *metis*
also have $\ldots \subseteq\ supp\ u \cup atom\ \text{`}\ fst\ \text{`}\ setD\ \Delta \cup atom\text{-}dom\ \Gamma \cup supp\ \mathcal{B}$
using *wfD-cons wf-supp1*(*4*)[*OF wfD-cons*(*3*)] **by** *auto*
also have $\ldots \subseteq atom\ \text{`}\ fst\ \text{`}\ setD\ ((u, \tau)\ \#_{\Delta}\ \Delta) \cup atom\text{-}dom\ \Gamma \cup supp\ \mathcal{B}$ **using** *supp-at-base* **by** *auto*
finally show *?case* **by** *auto*
next
case $(wfPhi\text{-}emptyI\ \Theta)$
then show *?case* **using** *supp-Nil* **by** *auto*
next
case $(wfPhi\text{-}consI\ f\ \Theta\ \Phi\ ft)$
then show *?case* **using** *fun-def.supp*
by (*simp add*: *pure-supp supp-Cons*)
next
case $(wfFTI\ \Theta\ B'\ b\ s\ x\ c\ \tau\ \Phi)$
have $supp\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s) = supp\ c \cup (supp\ \tau \cup supp\ s) - set\ [atom\ x] \cup supp\ b$ **using** *fun-typ.supp* **by** *auto*
thus *?case* **using** *wfFTI wf-supp1*
proof −
have *f1*: $supp\ \tau \subseteq \{atom\ x\} \cup atom\text{-}dom\ GNil \cup supp\ B'$

using *dom-cons wfFTI.hyps wf-supp1(4)* **by** *blast*
have *supp b* ⊆ *supp B'*
using *wfFTI.hyps(1) wf-supp1(7)* **by** *blast*
then show *?thesis*
using *f1* ‹*supp (AF-fun-typ x b c τ s) = supp c ∪ (supp τ ∪ supp s) − set [atom x] ∪ supp b*›
wfFTI.hyps(4) wfFTI.hyps **by** *auto*
qed
next
case (*wfFTNone* Θ Φ *ft*)
then show *?case* **by** (*simp add*: *fun-typ-q.supp(2)*)
next
case (*wfFTSome* Θ Φ *bv ft*)
then show *?case* **using** *fun-typ-q.supp*
by (*simp add*: *supp-at-base*)
next
case (*wfS-assertI* Θ Φ *ℬ x c* Γ Δ *s b*)
then have *supp c* ⊆ *atom-dom* Γ ∪ *atom* ‘ *fst* ‘ *setD* Δ ∪ *supp ℬ* **using** *wf-supp1*
by (*metis Un-assoc Un-commute le-supI2*)
moreover have *supp s* ⊆ *atom-dom* Γ ∪ *atom* ‘ *fst* ‘ *setD* Δ ∪ *supp ℬ* **proof**
fix *z*
assume ∗:*z* ∈ *supp s*
have ∗∗:*atom x* ∉ *supp s* **using** *wfS-assertI fresh-prodN fresh-def* **by** *metis*
have *z* ∈ *atom-dom* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) ∪ *atom* ‘ *fst* ‘ *setD* Δ ∪ *supp ℬ* **using** *wfS-assertI* ∗ **by** *blast*
have *z* ∈ *atom-dom* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) ⟹ *z* ∈ *atom-dom* Γ **using** ∗ ∗∗ **by** *auto*
thus *z* ∈ *atom-dom* Γ ∪ *atom* ‘ *fst* ‘ *setD* Δ ∪ *supp ℬ* **using** ∗ ∗∗
using ‹*z* ∈ *atom-dom* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) ∪ *atom* ‘ *fst* ‘ *setD* Δ ∪ *supp ℬ*› **by** *blast*
qed
ultimately show *?case* **by** *auto*
qed(*auto*)

lemmas *wf-supp* = *wf-supp1 wf-supp2*

lemma *wfV-supp-nil*:
fixes *v*::*v*
assumes *P* ; {||} ; *GNil* $\vdash_{wf}$ *v* : *b*
shows *supp v* = {}
using *wfV-supp*[*of P* {||} *GNil v b*] *dom.simps toSet.simps*
using *assms* **by** *auto*

lemma *wfT-TRUE-aux*:
assumes *wfG P ℬ* Γ **and** *atom z* ♯ (*P*, *ℬ*, Γ) **and** *wfB P ℬ b*
shows *wfT P ℬ* Γ (⦃ *z* : *b* | *TRUE* ⦄)
proof (*rule*)
show ‹ *atom z* ♯ (*P*, *ℬ*, Γ)› **using** *assms* **by** *auto*
show ‹ *P*; *ℬ* $\vdash_{wf}$ *b* › **using** *assms* **by** *auto*
show ‹ *P* ;*ℬ* ; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *TRUE* › **using** *wfG-cons2I wfC-trueI assms* **by** *auto*
qed

lemma *wfT-TRUE*:
assumes *wfG P ℬ* Γ **and** *wfB P ℬ b*
shows *wfT P ℬ* Γ (⦃ *z* : *b* | *TRUE* ⦄)

proof −
  obtain *z′*::*x* **where** ∗:*atom z′* ♯ (*P*, *ℬ*, Γ) **using** *obtain-fresh* **by** *metis*
  hence ⦃ *z* : *b* | *TRUE* ⦄ = ⦃ *z′* : *b* | *TRUE* ⦄ **by** *auto*
  thus *?thesis* **using** *wfT-TRUE-aux assms* ∗ **by** *metis*
qed

lemma *phi-flip-eq*:
  assumes *wfPhi T P*
  shows (*x* ↔ *xa*) · *P* = *P*
  using *wfPhi-supp*[*OF assms*] *flip-fresh-fresh fresh-def* **by** *blast*

lemma *wfC-supp-cons*:
  fixes *c′*::*c* **and** *G*::Γ
  assumes *P*; *ℬ* ; (*x′*, *b′* , *TRUE*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c′*
  shows *supp c′* ⊆ *atom-dom G* ∪ *supp x′* ∪ *supp ℬ* **and** *supp c′* ⊆ *supp G* ∪ *supp x′* ∪ *supp ℬ*
proof −
  show *supp c′* ⊆ *atom-dom G* ∪ *supp x′* ∪ *supp ℬ*
    using *wfC-supp*[*OF assms*] *dom-cons supp-at-base* **by** *blast*
  moreover have *atom-dom G* ⊆ *supp G*
    by (*meson assms wfC-wf wfG-cons wfG-supp*)
  ultimately show *supp c′* ⊆ *supp G* ∪ *supp x′* ∪ *supp ℬ* **using** *wfG-supp assms wfG-cons wfC-wf* **by**
fast
qed

lemma *wfG-dom-supp*:
  fixes *x*::*x*
  assumes *wfG P ℬ G*
  shows *atom x* ∈ *atom-dom G* ⟷ *atom x* ∈ *supp G*
using *assms* **proof**(*induct G rule*: Γ-*induct*)
  case *GNil*
  then show *?case* **using** *dom.simps supp-of-atom-list*
    using *supp-GNil* **by** *auto*
next
  case (*GCons x′ b′ c′ G*)

  show *?case* **proof**(*cases x′* = *x*)
    case *True*
    then show *?thesis* **using** *dom.simps supp-of-atom-list supp-at-base*
      using *supp-GCons* **by** *auto*
  next
    case *False*
    have (*atom x* ∈ *atom-dom* ((*x′*, *b′*, *c′*) $\#_\Gamma$ *G*)) = (*atom x* ∈ *atom-dom G*) **using** *atom-dom.simps*
False **by** *simp*
    also have ... = (*atom x* ∈ *supp G*) **using** *GCons wfG-elims* **by** *metis*
    also have ... = (*atom x* ∈ (*supp* (*x′*, *b′*, *c′*) ∪ *supp G*)) **proof**
      show *atom x* ∈ *supp G* ⟹ *atom x* ∈ *supp* (*x′*, *b′*, *c′*) ∪ *supp G* **by** *auto*
      assume *atom x* ∈ *supp* (*x′*, *b′*, *c′*) ∪ *supp G*
      then consider *atom x* ∈ *supp* (*x′*, *b′*, *c′*) | *atom x* ∈ *supp G* **by** *auto*
      then show *atom x* ∈ *supp G* **proof**(*cases*)
        case *1*
        assume *atom x* ∈ *supp* (*x′*, *b′*, *c′*)
        hence *atom x* ∈ *supp c′* **using** *supp-triple False supp-b-empty supp-at-base* **by** *force*

moreover have P; $\mathcal{B}$; $(x', b', TRUE)$ $\#_\Gamma G \vdash_{wf} c'$ **using** *wfG-elim2 GCons* **by** *simp*
moreover hence *supp* $c' \subseteq$ *supp* $G \cup$ *supp* $x' \cup$ *supp* $\mathcal{B}$ **using** *wfC-supp-cons* **by** *auto*
ultimately have *atom* $x \in$ *supp* $G \cup$ *supp* x' **using** *x-not-in-b-set* **by** *auto*
then show *?thesis* **using** *False supp-at-base* **by** (*simp add*: *supp-at-base*)
next
case *2*
then show *?thesis* **by** *simp*
qed
qed
also have ... = (*atom* $x \in$ *supp* $((x', b', c')$ $\#_\Gamma$ $G))$ **using** *supp-at-base False supp-GCons* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *wfG-atoms-supp-eq* :
fixes *x*::*x*
assumes *wfG P* $\mathcal{B}$ *G*
shows *atom* $x \in$ *atom-dom* $G \longleftrightarrow$ *atom* $x \in$ *supp* G
using *wfG-dom-supp assms* **by** *auto*

lemma *beta-flip-eq*:
fixes *x*::*x* **and** *xa*::*x* **and** $\mathcal{B}$::$\mathcal{B}$
shows $(x \leftrightarrow xa) \cdot \mathcal{B} = \mathcal{B}$
proof −
have *atom* $x \sharp \mathcal{B} \wedge$ *atom* $xa \sharp \mathcal{B}$ **using** *x-not-in-b-set fresh-def supp-set* **by** *metis*
thus *?thesis* **by** (*simp add*: *flip-fresh-fresh fresh-def*)
qed

lemma *theta-flip-eq2*:
assumes $\vdash_{wf} \Theta$
shows $(z \leftrightarrow za) \cdot \Theta = \Theta$
proof −
have *supp* $\Theta = \{\}$ **using** *wfTh-supp assms* **by** *simp*
thus *?thesis*
by (*simp add*: *flip-fresh-fresh fresh-def*)
qed

lemma *theta-flip-eq*:
assumes *wfTh* Θ
shows $(x \leftrightarrow xa) \cdot \Theta = \Theta$
using *wfTh-supp flip-fresh-fresh fresh-def*
by (*simp add*: *assms theta-flip-eq2*)

lemma *wfT-wfC*:
fixes *c*::*c*
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!\!\{ z : b \mid c \}\!\!\}$ **and** *atom* $z \sharp \Gamma$
shows Θ; $\mathcal{B}$; $(z,b,TRUE)$ $\#_\Gamma\Gamma \vdash_{wf} c$
proof −
obtain *za ba ca* **where** $*$:$\{\!\!\{ z : b \mid c \}\!\!\} = \{\!\!\{ za : ba \mid ca \}\!\!\} \wedge$ *atom* $za \sharp (\Theta,\mathcal{B},\Gamma) \wedge$ Θ; $\mathcal{B}$; $(za, ba, TRUE)$ $\#_\Gamma \Gamma \vdash_{wf} ca$
using *wfT-elims*[*OF assms*(*1*)] **by** *metis*

hence *c1*: [[*atom z*]]*lst. c* = [[*atom za*]]*lst. ca* **using** *τ.eq-iff* **by** *meson*
show *?thesis* **proof**(*cases z=za*)
case *True*
hence *ca* = *c* **using** *c1* **by** (*simp add*: *Abs1-eq-iff*(*3*))
then show *?thesis* **using** ∗ *True* **by** *simp*
next
case *False*
have $\vdash_{wf}$ Θ **using** *wfT-wf wfG-wf assms* **by** *metis*
moreover have *atom za* ♯ Γ **using** ∗ *fresh-prodN* **by** *auto*
ultimately have Θ; $\mathcal{B}$; (*z* ↔ *za*) • (*za*, *ba*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ (*z* ↔ *za*) • *ca*
using *wfC.eqvt theta-flip-eq2 beta-flip-eq* ∗ *GCons-eqvt assms flip-fresh-fresh* **by** *metis*
moreover have *atom z* ♯ *ca*
proof −
have *supp ca* ⊆ *atom-dom* Γ ∪ { *atom za* } ∪ *supp* $\mathcal{B}$ **using** ∗ *wfC-supp atom-dom.simps toSet.simps*
by *fastforce*
moreover have *atom z* ∉ *atom-dom* Γ **using** *assms fresh-def wfT-wf wfG-dom-supp wfC-supp*
by *metis*
moreover hence *atom z* ∉ *atom-dom* Γ ∪ { *atom za* } **using** *False* **by** *simp*
moreover have *atom z* ∉ *supp* $\mathcal{B}$ **using** *x-not-in-b-set* **by** *simp*
ultimately show *?thesis* **using** *fresh-def False* **by** *fast*
qed
moreover hence (*z* ↔ *za*) • *ca* = *c* **using** *type-eq-subst-eq1*(*3*) ∗ **by** *metis*
ultimately show *?thesis* **using** *assms G-cons-flip-fresh* ∗ **by** *auto*
qed
qed

lemma *u-not-in-dom-g*:
fixes *u*::*u*
shows *atom u* ∉ *atom-dom G*
using *toSet.simps atom-dom.simps u-not-in-x-atoms* **by** *auto*

lemma *bv-not-in-dom-g*:
fixes *bv*::*bv*
shows *atom bv* ∉ *atom-dom G*
using *toSet.simps atom-dom.simps u-not-in-x-atoms* **by** *auto*

An important lemma that confirms that Γ does not rely on mutable variables

lemma *u-not-in-g*:
fixes *u*::*u*
assumes *wfG* Θ *B G*
shows *atom u* ∉ *supp G*
using *assms* **proof**(*induct G rule*: Γ*-induct*)
case *GNil*
then show *?case* **using** *supp-GNil fresh-def*
using *fresh-set-empty* **by** *fastforce*
next
case (*GCons x b c* Γ′)
moreover hence *atom u* ∉ *supp b* **using**
wfB-supp wfC-supp u-not-in-x-atoms wfG-elims wfX-wfY **by** *auto*
moreover hence *atom u* ∉ *supp x* **using** *u-not-in-x-atoms supp-at-base* **by** *blast*
moreover hence *atom u* ∉ *supp c* **proof** −
have Θ ; *B* ; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ′ $\vdash_{wf}$ *c* **using** *wfG-cons-wfC GCons* **by** *simp*

hence *supp c* ⊆ *atom-dom* ((*x*, *b*, *TRUE*) #$_\Gamma$ Γ′) ∪ *supp B* **using** *wfC-supp* **by** *blast*
thus *?thesis* **using** *u-not-in-dom-g u-not-in-b-atoms*
using *u-not-in-b-set* **by** *auto*
qed
ultimately have *atom u* ∉ *supp* (*x*,*b*,*c*) **using** *supp-Pair* **by** *simp*
thus *?case* **using** *supp-GCons GCons wfG-elims* **by** *blast*
qed

An important lemma that confirms that types only depend on immutable variables

lemma *u-not-in-t*:
fixes *u*::*u*
assumes *wfT* Θ *B G* τ
shows *atom u* ∉ *supp* τ
proof −
have *supp* τ ⊆ *atom-dom G* ∪ *supp B* **using** *wfT-supp assms* **by** *auto*
thus *?thesis* **using** *u-not-in-dom-g u-not-in-b-set* **by** *blast*
qed

lemma *wfT-supp-c*:
fixes *ℬ*::*ℬ* **and** *z*::*x*
assumes *wfT P ℬ* Γ (⦃ *z* : *b* | *c* ⦄)
shows *supp c* − { *atom z* } ⊆ *atom-dom* Γ ∪ *supp ℬ*
using *wf-supp* τ.*supp assms*
by (*metis Un-subset-iff empty-set list.simps*(*15*))

lemma *wfG-wfC*[*ms-wb*]:
assumes *wfG P ℬ* ((*x*,*b*,*c*) #$_\Gamma$Γ)
shows *wfC P ℬ* ((*x*,*b*,*TRUE*) #$_\Gamma$Γ) *c*
using *assms* **proof**(*cases c* ∈ {*TRUE*,*FALSE*})
case *True*
have *atom x* ♯ Γ ∧ *wfG P ℬ* Γ ∧ *wfB P ℬ b* **using** *wfG-cons assms* **by** *auto*
hence *wfG P ℬ* ((*x*,*b*,*TRUE*) #$_\Gamma$Γ) **using** *wfG-cons2I* **by** *auto*
then show *?thesis* **using** *wfC-trueI wfC-falseI True* **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-elims assms* **by** *blast*
qed

lemma *wfT-wf-cons*:
assumes *wfT P ℬ* Γ ⦃ *z* : *b* | *c* ⦄ **and** *atom z* ♯ Γ
shows *wfG P ℬ* ((*z*,*b*,*c*) #$_\Gamma$Γ)
using *assms* **proof**(*cases c* ∈ { *TRUE*,*FALSE* })
case *True*
then show *?thesis* **using** *wfT-wfC wfC-wf wfG-wfB wfG-cons2I assms wfT-wf* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *wfT-wfC wfC-wf wfG-wfB wfG-cons1I wfT-wf wfT-wfC assms* **by** *fastforce*
qed

lemma *wfV-b-fresh*:
fixes *b*::*b* **and** *v*::*v* **and** *bv*::*bv*
assumes Θ; *ℬ*; Γ $\vdash_{wf}$ *v*: *b* **and** *bv* |∉| *ℬ*

shows *atom bv* ♯ *v*
using *wfV-supp bv-not-in-dom-g fresh-def assms bv-not-in-bset-supp* **by** *blast*

lemma *wfCE-b-fresh*:
fixes *b*::*b* **and** *ce*::*ce* **and** *bv*::*bv*
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce*: *b* **and** *bv* |∉| $\mathcal{B}$
shows *atom bv* ♯ *ce*
using *bv-not-in-dom-g fresh-def assms bv-not-in-bset-supp wf-supp1*(*8*) **by** *fast*

8.7 Freshness

lemma *wfG-fresh-x*:
fixes Γ::Γ **and** *z*::*x*
assumes Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** *atom z* ♯ Γ
shows *atom z* ♯ (Θ,$\mathcal{B}$, Γ)
unfolding *fresh-prodN* **apply**(*intro conjI*)
using *wf-supp1 wfX-wfY assms fresh-def x-not-in-b-set* **by**(*metis empty-iff*)+

lemma *wfG-wfT*:
assumes *wfG P* $\mathcal{B}$ ((*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) $\#_\Gamma$ *G*) **and** *atom x* ♯ *c*
shows *P*; $\mathcal{B}$; *G* $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄
proof −
have *P*; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c*[*z*::=*V-var x*]$_{cv}$ ∧ *wfB P* $\mathcal{B}$ *b* **using** *assms*
using *wfG-elim2* **by** *auto*
moreover have *atom x* ♯ (*P* ,$\mathcal{B}$,*G*) **using** *wfG-elims assms wfG-fresh-x* **by** *metis*
ultimately have *wfT P* $\mathcal{B}$ *G* ⦃ *x* : *b* | *c*[*z*::=*V-var x*]$_{cv}$ ⦄ **using** *wfTI assms* **by** *metis*
moreover have ⦃ *x* : *b* | *c*[*z*::=*V-var x*]$_{cv}$ ⦄ = ⦃ *z* : *b* | *c* ⦄ **using** *type-eq-subst* ‹*atom x* ♯ *c*› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *wfT-wfT-if*:
assumes *wfT* Θ $\mathcal{B}$ Γ (⦃ *z2* : *b* | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP* *c*[*z*::=*V-var z2*]$_{cv}$ ⦄)
and *atom z2* ♯ (*c*,Γ)
shows *wfT* Θ $\mathcal{B}$ Γ ⦃ *z* : *b* | *c* ⦄
proof −
have ∗: *atom z2* ♯ (Θ, $\mathcal{B}$, Γ) **using** *wfG-fresh-x wfX-wfY assms fresh-Pair* **by** *metis*
have *wfB* Θ $\mathcal{B}$ *b* **using** *assms wfT-elims* **by** *metis*
have Θ; $\mathcal{B}$; (*GCons* (*z2*,*b*,*TRUE*) Γ) $\vdash_{wf}$ (*CE-val v* == *CE-val* (*V-lit L-false*) *IMP* *c*[*z*::=*V-var z2*]$_{cv}$) **using** *wfT-wfC assms fresh-Pair* **by** *auto*
hence Θ; $\mathcal{B}$; ((*z2*,*b*,*TRUE*) $\#_\Gamma$Γ) $\vdash_{wf}$ *c*[*z*::=*V-var z2*]$_{cv}$ **using** *wfC-elims* **by** *metis*
hence *wfT* Θ $\mathcal{B}$ Γ (⦃ *z2* : *b* | *c*[*z*::=*V-var z2*]$_{cv}$⦄) **using** *assms fresh-Pair wfTI* ‹*wfB* Θ $\mathcal{B}$ *b*› ∗ **by** *auto*
moreover have ⦃ *z* : *b* | *c* ⦄ = ⦃ *z2* : *b* | *c*[*z*::=*V-var z2*]$_{cv}$ ⦄ **using** *type-eq-subst assms fresh-Pair* **by** *auto*
ultimately show *?thesis* **using** *wfTI assms* **by** *argo*
qed

lemma *wfT-fresh-c*:
fixes *x*::*x*
assumes *wfT P* $\mathcal{B}$ Γ ⦃ *z* : *b* | *c* ⦄ **and** *atom x* ♯ Γ **and** *x* ≠ *z*
shows *atom x* ♯ *c*
proof(*rule ccontr*)

assume ¬ *atom x* ♯ *c*
hence ∗:*atom x* ∈ *supp c* **using** *fresh-def* **by** *auto*
moreover have *supp c* − *set* [*atom z*] ∪ *supp b* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$
using *assms wfT-supp τ.supp* **by** *blast*
moreover hence *atom x* ∈ *supp c* − *set* [*atom z*] **using** *assms* ∗ **by** *auto*
ultimately have *atom x* ∈ *atom-dom* Γ **using** *x-not-in-b-set* **by** *auto*
thus *False* **using** *assms wfG-atoms-supp-eq wfT-wf fresh-def* **by** *metis*
qed

lemma *wfG-x-fresh* [*simp*]:
fixes *x*::*x*
assumes *wfG P* $\mathcal{B}$ *G*
shows *atom x* ∉ *atom-dom G* ⟷ *atom x* ♯ *G*
using *wfG-atoms-supp-eq assms fresh-def* **by** *metis*

lemma *wfD-x-fresh*:
fixes *x*::*x*
assumes *atom x* ♯ Γ **and** *wfD P B* Γ Δ
shows *atom x* ♯ Δ
using *assms* **proof**(*induct* Δ *rule*: Δ*-induct*)
case *DNil*
then show *?case* **using** *supp-DNil fresh-def* **by** *auto*
next
case (*DCons u′ t′* Δ′)
have *wfg*: *wfG P B* Γ **using** *wfD-wf DCons* **by** *blast*
hence *wfd*: *wfD P B* Γ Δ′ **using** *wfD-elims DCons* **by** *blast*
have *supp t′* ⊆ *atom-dom* Γ ∪ *supp B* **using** *wfT-supp DCons wfD-elims* **by** *metis*
moreover have *atom x* ∉ *atom-dom* Γ **using** *DCons*(*2*) *fresh-def wfG-supp wfg* **by** *blast*
ultimately have *atom x* ♯ *t′* **using** *fresh-def DCons wfG-supp wfg x-not-in-b-set* **by** *blast*
moreover have *atom x* ♯ *u′* **using** *supp-at-base fresh-def* **by** *fastforce*
ultimately have *atom x* ♯ (*u′*,*t′*) **using** *supp-Pair* **by** *fastforce*
thus *?case* **using** *DCons fresh-DCons wfd* **by** *fast*
qed

lemma *wfG-fresh-x2*:
fixes Γ::Γ **and** *z*::*x* **and** Δ::Δ **and** Φ::Φ
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ **and** Θ $\vdash_{wf}$ Φ **and** *atom z* ♯ Γ
shows *atom z* ♯ (Θ,Φ,$\mathcal{B}$, Γ,Δ)
unfolding *fresh-prodN* **apply**(*intro conjI*)
using *wfG-fresh-x assms fresh-prod3 wfX-wfY* **apply** *metis*
using *wf-supp2*(*5*) *assms fresh-def* **apply** *blast*
using *assms wfG-fresh-x wfX-wfY fresh-prod3* **apply** *metis*
using *assms wfG-fresh-x wfX-wfY fresh-prod3* **apply** *metis*
using *wf-supp2*(*6*) *assms fresh-def wfD-x-fresh* **by** *metis*

lemma *wfV-x-fresh*:
fixes *v*::*v* **and** *b*::*b* **and** Γ::Γ **and** *x*::*x*
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* **and** *atom x* ♯ Γ
shows *atom x* ♯ *v*
proof −
have *supp v* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *assms wfV-supp* **by** *auto*
moreover have *atom x* ∉ *atom-dom* Γ **using** *fresh-def assms*

dom.simps subsetCE wfG-elims wfG-supp **by** (*metis dom-supp-g*)
moreover have *atom x* $\notin$ *supp* $\mathcal{B}$ **using** *x-not-in-b-set* **by** *auto*
ultimately show *?thesis* **using** *fresh-def* **by** *fast*
qed

lemma *wfE-x-fresh*:
fixes *e*::*e* **and** *b*::*b* **and** Γ::Γ **and** Δ::Δ **and** Φ::Φ **and** *x*::*x*
assumes Θ; Φ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b* **and** *atom x* ♯ Γ
shows *atom x* ♯ *e*
proof −
have *wfG* Θ $\mathcal{B}$ Γ **using** *assms wfE-wf* **by** *auto*
hence *supp e* $\subseteq$ *atom-dom* Γ $\cup$ *supp* $\mathcal{B}$ $\cup$ *atom'fst'setD* Δ **using** *wfE-supp dom.simps assms* **by** *auto*
moreover have *atom x* $\notin$ *atom-dom* Γ **using** *fresh-def assms*
dom.simps subsetCE ‹*wfG* Θ $\mathcal{B}$ Γ› *wfG-supp* **by** (*metis dom-supp-g*)
moreover have *atom x* $\notin$ *atom'fst'setD* Δ **by** *auto*
ultimately show *?thesis* **using** *fresh-def x-not-in-b-set* **by** *fast*
qed

lemma *wfT-x-fresh*:
fixes τ::τ **and** Γ::Γ **and** *x*::*x*
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ **and** *atom x* ♯ Γ
shows *atom x* ♯ τ
proof −
have *wfG* Θ $\mathcal{B}$ Γ **using** *assms wfX-wfY* **by** *auto*
hence *supp* τ $\subseteq$ *atom-dom* Γ $\cup$ *supp* $\mathcal{B}$ **using** *wfT-supp dom.simps assms* **by** *auto*
moreover have *atom x* $\notin$ *atom-dom* Γ **using** *fresh-def assms*
dom.simps subsetCE ‹*wfG* Θ $\mathcal{B}$ Γ› *wfG-supp* **by** (*metis dom-supp-g*)
moreover have *atom x* $\notin$ *supp* $\mathcal{B}$ **using** *x-not-in-b-set* **by** *simp*
ultimately show *?thesis* **using** *fresh-def* **by** *fast*
qed

lemma *wfS-x-fresh*:
fixes *s*::*s* **and** Δ::Δ **and** *x*::*x*
assumes Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *s* : *b* **and** *atom x* ♯ Γ
shows *atom x* ♯ *s*
proof −
have *supp s* $\subseteq$ *atom-dom* Γ $\cup$ *atom* ' *fst* ' *setD* Δ $\cup$ *supp* $\mathcal{B}$ **using** *wf-supp assms* **by** *metis*
moreover have *atom x* $\notin$ *atom* ' *fst* ' *setD* Δ **by** *auto*
moreover have *atom x* $\notin$ *atom-dom* Γ **using** *assms fresh-def wfG-dom-supp wfX-wfY* **by** *metis*
moreover have *atom x* $\notin$ *supp* $\mathcal{B}$ **using** *supp-b-empty supp-fset*
by (*simp add*: *x-not-in-b-set*)
ultimately show *?thesis* **using** *fresh-def* **by** *fast*
qed

lemma *wfTh-fresh*:
fixes *x*
assumes *wfTh T*
shows *atom x* ♯ *T*
using *wf-supp1 assms fresh-def* **by** *fastforce*

lemmas *wfTh-x-fresh* = *wfTh-fresh*

lemma *wfPhi-fresh*:
fixes *x*
assumes *wfPhi T P*
shows *atom x* ♯ *P*
using *wf-supp assms fresh-def* **by** *fastforce*

lemmas *wfPhi-x-fresh* = *wfPhi-fresh*
lemmas *wb-x-fresh* = *wfTh-x-fresh wfPhi-x-fresh wfD-x-fresh wfT-x-fresh wfV-x-fresh*

lemma *wfG-inside-fresh*[*ms-fresh*]:
fixes Γ::Γ **and** *x*::*x*
assumes *wfG P B* (Γ′@((*x*,*b*,*c*) #$_Γ$Γ))
shows *atom x* ∉ *atom-dom* Γ′
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
moreover hence *atom x* ∉ *atom* ‘ *fst* ‘({(*x1*,*b1*,*c1*)}) **proof** −
have ∗: *P*; *B* $\vdash_{wf}$ (Γ*1* @ (*x*, *b*, *c*) #$_Γ$ Γ) **using** *wfG-elims append-g.simps GCons* **by** *metis*
have *atom x1* ♯ (Γ*1* @ (*x*, *b*, *c*) #$_Γ$ Γ) **using** *GCons wfG-elims append-g.simps* **by** *metis*
hence *atom x1* ∉ *atom-dom* (Γ*1* @ (*x*, *b*, *c*) #$_Γ$ Γ) **using** *wfG-dom-supp fresh-def* ∗ **by** *metis*
thus *?thesis* **by** *auto*
qed
ultimately show *?case* **using** *append-g.simps atom-dom.simps toSet.simps wfG-elims dom.simps*
by (*metis image-insert insert-iff insert-is-Un*)
qed

lemma *wfG-inside-x-in-atom-dom*:
fixes *c*::*c* **and** *x*::*x* **and** Γ::Γ
shows *atom x* ∈ *atom-dom* (Γ′@ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_Γ$ Γ)
by(*induct* Γ′ *rule*: Γ-*induct*, (*simp add*: *toSet.simps atom-dom.simps*)+)

lemma *wfG-inside-x-neq*:
fixes *c*::*c* **and** *x*::*x* **and** Γ::Γ **and** *G*::Γ **and** *xa*::*x*
assumes *G*=(Γ′@ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_Γ$ Γ) **and** *atom xa* ♯ *G* **and** Θ; *B* $\vdash_{wf}$ *G*
shows *xa* ≠ *x*
proof −
have *atom xa* ∉ *atom-dom G* **using** *fresh-def wfG-atoms-supp-eq assms* **by** *metis*
moreover have *atom x* ∈ *atom-dom G* **using** *wfG-inside-x-in-atom-dom assms* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *wfG-inside-x-fresh*:
fixes *c*::*c* **and** *x*::*x* **and** Γ::Γ **and** *G*::Γ **and** *xa*::*x*
assumes *G*=(Γ′@ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_Γ$ Γ) **and** *atom xa* ♯ *G* **and** Θ; *B* $\vdash_{wf}$ *G*
shows *atom xa* ♯ *x*
using *fresh-def supp-at-base wfG-inside-x-neq assms* **by** *auto*

lemma *wfT-nil-supp*:
fixes *t*::τ
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ *t*

shows *supp t* = {}
using *wfT-supp atom-dom.simps assms toSet.simps* **by** *force*

8.8 Misc

lemma *wfG-cons-append*:
fixes b'::b
assumes Θ; $\mathcal{B} \vdash_{wf}$ ((x', b', c') $\#_\Gamma$ Γ') @ (x, b, c) $\#_\Gamma$ Γ
shows Θ; $\mathcal{B} \vdash_{wf}$ (Γ' @ (x, b, c) $\#_\Gamma$ Γ) $\wedge$ *atom* x' ♯ (Γ' @ (x, b, c) $\#_\Gamma$ Γ) $\wedge$ Θ; $\mathcal{B} \vdash_{wf}$ b' $\wedge$ $x' \neq x$
proof −
have ((x', b', c') $\#_\Gamma$ Γ') @ (x, b, c) $\#_\Gamma$ Γ = (x', b', c') $\#_\Gamma$ (Γ' @ (x, b, c) $\#_\Gamma$ Γ) **using** *append-g.simps* **by** *auto*
hence ∗:Θ; $\mathcal{B} \vdash_{wf}$ (Γ' @ (x, b, c) $\#_\Gamma$ Γ) $\wedge$ *atom* x' ♯ (Γ' @ (x, b, c) $\#_\Gamma$ Γ) $\wedge$ Θ; $\mathcal{B} \vdash_{wf}$ b' **using** *assms wfG-cons* **by** *metis*
moreover have *atom* x' ♯ x **proof**(*rule wfG-inside-x-fresh*[*of* (Γ' @ (x, b, c) $\#_\Gamma$ Γ)])
show Γ' @ (x, b, c) $\#_\Gamma$ Γ = Γ' @ (x, b, c[x::=*V-var* x]$_{cv}$) $\#_\Gamma$ Γ **by** *simp*
show *atom* x' ♯ Γ' @ (x, b, c) $\#_\Gamma$ Γ **using** ∗ **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (x, b, c) $\#_\Gamma$ Γ **using** ∗ **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *flip-u-eq*:
fixes u::u **and** u'::u **and** Θ::Θ **and** τ::τ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ τ
shows ($u \leftrightarrow u'$) $\cdot$ τ = τ **and** ($u \leftrightarrow u'$) $\cdot$ Γ = Γ **and** ($u \leftrightarrow u'$) $\cdot$ Θ = Θ **and** ($u \leftrightarrow u'$) $\cdot$ $\mathcal{B}$ = $\mathcal{B}$
proof −
show ($u \leftrightarrow u'$) $\cdot$ τ = τ **using** *wfT-supp flip-fresh-fresh*
by (*metis assms*(*1*) *fresh-def u-not-in-t*)
show ($u \leftrightarrow u'$) $\cdot$ Γ = Γ **using** *u-not-in-g wfX-wfY assms flip-fresh-fresh fresh-def* **by** *metis*
show ($u \leftrightarrow u'$) $\cdot$ Θ = Θ **using** *theta-flip-eq assms wfX-wfY* **by** *metis*
show ($u \leftrightarrow u'$) $\cdot$ $\mathcal{B}$ = $\mathcal{B}$ **using** *u-not-in-b-set flip-fresh-fresh fresh-def* **by** *metis*
qed

lemma *wfT-wf-cons-flip*:
fixes c::c **and** x::x
assumes *wfT P* $\mathcal{B}$ Γ ⦃ z : b | c ⦄ **and** *atom* x ♯ (c,Γ)
shows *wfG P* $\mathcal{B}$ ((x,b,c[z::=*V-var* x]$_{cv}$) $\#_\Gamma$$\Gamma$)
proof −
have ⦃ x : b | c[z::=*V-var* x]$_{cv}$ ⦄ = ⦃ z : b | c ⦄ **using** *assms freshers type-eq-subst* **by** *metis*
hence ∗:*wfT P* $\mathcal{B}$ Γ ⦃ x : b | c[z::=*V-var* x]$_{cv}$ ⦄ **using** *assms* **by** *metis*
show *?thesis* **proof**(*rule wfG-consI*)
show ‹ P; $\mathcal{B}$ $\vdash_{wf}$ Γ › **using** *assms wfT-wf* **by** *auto*
show ‹*atom* x ♯ Γ› **using** *assms* **by** *auto*
show ‹ P; $\mathcal{B}$ $\vdash_{wf}$ b › **using** *assms wfX-wfY b-of.simps* **by** *metis*
show ‹ P; $\mathcal{B}$; (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ c[z::=*V-var* x]$_{cv}$ › **using** *wfT-wfC* ∗ *assms fresh-Pair* **by** *metis*
qed
qed

8.9 Context Strengthening

We can remove an entry for a variable from the context if the variable doesn't appear in the term and the variable is not used later in the context or any other context

lemma *fresh-restrict*:
fixes y::$'a$::*at-base* **and** Γ::Γ
assumes *atom* y ♯ $(\Gamma' \ @\ (x,\ b,\ c) \quad \#_\Gamma\ \Gamma)$
shows *atom* y ♯ $(\Gamma'@\Gamma)$
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *fresh-GCons fresh-GNil* **by** *auto*
next
case (*GCons* x' b' c' Γ′′)
then show *?case* **using** *fresh-GCons fresh-GNil* **by** *auto*
qed

lemma *wf-restrict1*:
fixes Γ::Γ **and** Γ′::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*∗τ) *list* **and** Δ::Δ **and** s::s **and** b::b **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows $\Theta;\ \mathcal{B};\ \Gamma\ \vdash_{wf}\ v : b \implies \Gamma{=}\Gamma_1@((x,b',c')\ \ \#_\Gamma\Gamma_2) \implies$ *atom* x ♯ $v \implies$ *atom* x ♯ $\Gamma_1 \implies \Theta;\ \mathcal{B};\ \Gamma_1@\Gamma_2 \vdash_{wf}\ v : b$ **and**

$\Theta;\ \mathcal{B};\ \Gamma\ \vdash_{wf}\ c \implies \Gamma{=}\Gamma_1@((x,b',c')\ \ \#_\Gamma\Gamma_2) \implies$ *atom* x ♯ $c \implies$ *atom* x ♯ $\Gamma_1 \implies \Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1@\Gamma_2\ \vdash_{wf}\ c$ **and**
$\Theta;\ \mathcal{B} \vdash_{wf} \Gamma \implies \Gamma{=}\Gamma_1@((x,b',c')\ \ \#_\Gamma\Gamma_2) \implies$ *atom* x ♯ $\Gamma_1 \implies \Theta;\ \mathcal{B} \vdash_{wf} \Gamma_1@\Gamma_2$ **and**
$\Theta;\ \mathcal{B};\ \Gamma\ \vdash_{wf}\ \tau \implies \Gamma{=}\Gamma_1@((x,b',c')\ \ \#_\Gamma\Gamma_2) \implies$ *atom* x ♯ $\tau \implies$ *atom* x ♯ $\Gamma_1 \implies \Theta;\ \mathcal{B};\ \Gamma_1@\Gamma_2 \vdash_{wf}\ \tau$ **and**
$\Theta;\ \mathcal{B};\ \Gamma\ \vdash_{wf}\ ts \implies$ *True* **and**
$\vdash_{wf}\ \Theta \implies$ *True* **and**

$\Theta;\ \mathcal{B} \vdash_{wf}\ b \implies$ *True* **and**

$\Theta;\ \mathcal{B};\ \Gamma\ \vdash_{wf}\ ce : b \implies \Gamma{=}\Gamma_1@((x,b',c')\ \ \#_\Gamma\Gamma_2) \implies$ *atom* x ♯ $ce \implies$ *atom* x ♯ $\Gamma_1 \implies \Theta;\ \mathcal{B};\ \Gamma_1@\Gamma_2\ \vdash_{wf}\ ce : b$ **and**
$\Theta\ \vdash_{wf}\ td \implies$ *True*
proof(*induct arbitrary*: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ b c y)
hence $y{\neq}x$ **using** *v.fresh* **by** *auto*
hence *Some* $(b,\ c)$ = *lookup* $(\Gamma_1@\Gamma_2)$ y **using** *lookup-restrict wfV-varI* **by** *metis*
then show *?case* **using** *wfV-varI wf-intros* **by** *metis*
next
case (*wfV-litI* Θ Γ l)
then show *?case* **using** *e.fresh wf-intros* **by** *metis*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
show *?case* **proof**
show $\Theta;\ \mathcal{B};\ \Gamma_1\ @\ \Gamma_2 \vdash_{wf}\ v1\ :\ b1$ **using** *wfV-pairI* **by** *auto*
show $\Theta;\ \mathcal{B};\ \Gamma_1\ @\ \Gamma_2 \vdash_{wf}\ v2\ :\ b2$ **using** *wfV-pairI* **by** *auto*
qed

next
case (*wfV-consI s dclist* Θ *dc x b c* $\mathcal{B}$ Γ *v*)
show *?case* **proof**
show *AF-typedef s dclist* ∈ *set* Θ **using** *wfV-consI* **by** *auto*
show (*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* **using** *wfV-consI* **by** *auto*
show Θ; $\mathcal{B}$; Γ_1 @ Γ_2 $\vdash_{wf}$ *v* : *b* **using** *wfV-consI* **by** *auto*
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x b′ c* $\mathcal{B}$ *b* Γ *v*)
show *?case* **proof**
show *AF-typedef-poly s bv dclist* ∈ *set* Θ **using** *wfV-conspI* **by** *auto*
show (*dc*, ⦃ *x* : *b′* | *c* ⦄) ∈ *set dclist* **using** *wfV-conspI* **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *wfV-conspI* **by** *auto*
show Θ; $\mathcal{B}$; Γ_1 @ Γ_2 $\vdash_{wf}$ *v* : *b′*[*bv*::=*b*]$_{bb}$ **using** *wfV-conspI* **by** *auto*
show *atom bv* ♯ (Θ, $\mathcal{B}$, Γ_1 @ Γ_2, *b*, *v*) **unfolding** *fresh-prodN fresh-append-g* **using** *wfV-conspI fresh-prodN fresh-GCons fresh-append-g* **by** *metis*
qed
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-eqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
hence $x \neq z$ **using** *wfTI*
fresh-GCons fresh-prodN fresh-PairD(*1*) *fresh-gamma-append not-self-fresh* **by** *metis*
show *?case* **proof**
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ_1 @ Γ_2)› **using** *wfTI fresh-restrict*[*of z*] **using** *wfG-fresh-x wfX-wfY wfTI fresh-prodN* **by** *metis*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfTI* **by** *auto*
have Θ; $\mathcal{B}$; ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1) @ Γ_2 $\vdash_{wf}$ *c* **proof**(*rule wfTI*(*5*)[*of* (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1])
show ‹(*z*, *b*, *TRUE*) $\#_\Gamma$ Γ = ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1) @ (*x*, *b′*, *c′*) $\#_\Gamma$ Γ_2› **using** *wfTI* **by** *auto*
show ‹*atom x* ♯ *c*› **using** *wfTI τ.fresh* ‹$x \neq z$› **by** *auto*

show ‹*atom x* ♯ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1› **using** *wfTI* ‹$x \neq z$› *fresh-GCons* **by** *simp*
qed
thus ‹ Θ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1 @ Γ_2 $\vdash_{wf}$ *c* › **by** *auto*
qed
next
case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
show *?case* **proof**
show Θ; $\mathcal{B}$; Γ_1 @ Γ_2 $\vdash_{wf}$ *e1* : *b* **using** *wfC-eqI c.fresh fresh-Nil* **by** *auto*
show Θ; $\mathcal{B}$; Γ_1 @ Γ_2 $\vdash_{wf}$ *e2* : *b* **using** *wfC-eqI c.fresh fresh-Nil* **by** *auto*
qed
next
case (*wfC-trueI* Θ Γ)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-falseI* Θ Γ)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-conjI* Θ Γ *c1 c2*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-disjI* Θ Γ *c1 c2*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-notI* Θ Γ *c1*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-impI* Θ Γ *c1 c2*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfG-nilI* Θ)
then show *?case* **using** *wfV-varI wf-intros*
by (*meson GNil-append* Γ.*simps*(*3*))
next
case (*wfG-cons1I c1* Θ $\mathcal{B}$ *G x1 b1*)
show *?case* **proof**(*cases* Γ_1=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons1I wfG-consI* **by** *auto*
next
case *False*
then obtain *G′*::Γ **where** ∗:(*x1*, *b1*, *c1*) $\#_\Gamma$ *G′* = Γ_1 **using** *GCons-eq-append-conv wfG-cons1I* **by** *auto*
hence ∗∗:*G*=*G′* @ (*x*, *b′*, *c′*) $\#_\Gamma$ Γ_2 **using** *wfG-cons1I* **by** *auto*

have Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x1*, *b1*, *c1*) $\#_\Gamma$ (*G′* @ Γ_2) **proof**(*rule Wellformed.wfG-cons1I*)
show ‹*c1* ∉ {*TRUE*, *FALSE*}› **using** *wfG-cons1I* **by** *auto*
show ‹*atom x1* ♯ *G′* @ Γ_2› **using** *wfG-cons1I*(*4*) ∗∗ *fresh-restrict* **by** *metis*
have *atom x* ♯ *G′* **using** *wfG-cons1I* ∗ **using** *fresh-GCons* **by** *blast*
thus ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *G′* @ Γ_2 › **using** *wfG-cons1I*(*3*)[*of G′*] ∗∗ **by** *auto*
have *atom x* ♯ *c1* ∧ *atom x* ♯ (*x1*, *b1*, *TRUE*) $\#_\Gamma$ *G′* **using** *fresh-GCons* ‹*atom x* ♯ Γ_1› ∗ **by** *auto*
thus ‹ Θ; $\mathcal{B}$; (*x1*, *b1*, *TRUE*) $\#_\Gamma$ *G′* @ Γ_2 $\vdash_{wf}$ *c1* › **using** *wfG-cons1I*(*6*)[*of* (*x1*, *b1*, *TRUE*) $\#_\Gamma$ *G′*] ∗∗ ∗ *wfG-cons1I* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfG-cons1I* **by** *auto*

qed
thus *?thesis* **using** $*$ **by** *auto*
qed
next
case (*wfG-cons2I c1* Θ $\mathcal{B}$ *G x1 b1*)
show *?case* **proof**(*cases* Γ_1=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons2I wfG-consI* **by** *auto*
next
case *False*
then obtain G'::Γ **where** $*$:(*x1*, *b1*, *c1*) $\#_\Gamma$ $G' = \Gamma_1$ **using** *GCons-eq-append-conv wfG-cons2I* **by** *auto*
hence $**$:G=G' @ (x, b', c') $\#_\Gamma$ Γ_2 **using** *wfG-cons2I* **by** *auto*

have Θ; $\mathcal{B} \vdash_{wf}$ (*x1*, *b1*, *c1*) $\#_\Gamma$ (G' @ Γ_2) **proof**(*rule Wellformed.wfG-cons2I*)
show ‹*c1* $\in$ {*TRUE*, *FALSE*}› **using** *wfG-cons2I* **by** *auto*
show ‹*atom x1* $\sharp$ G' @ Γ_2› **using** *wfG-cons2I* $**$ *fresh-restrict* **by** *metis*
have *atom x* $\sharp$ G' **using** *wfG-cons2I* $*$ **using** *fresh-GCons* **by** *blast*
thus ‹ Θ; $\mathcal{B} \vdash_{wf}$ G' @ Γ_2 › **using** *wfG-cons2I* $**$ **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfG-cons2I* **by** *auto*
qed
thus *?thesis* **using** $*$ **by** *auto*
qed
qed(*auto*)+

lemma *wf-restrict2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ; Φ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *e* : *b* $\Longrightarrow$ Γ=Γ_1@((*x*,b',c') $\#_\Gamma\Gamma_2$) $\Longrightarrow$ *atom x* $\sharp$ *e* $\Longrightarrow$ *atom x* $\sharp$ Γ_1 $\Longrightarrow$ *atom x* $\sharp$ Δ $\Longrightarrow$ Θ; Φ; $\mathcal{B}$; Γ_1@Γ_2 ; $\Delta \vdash_{wf}$ *e* : *b* **and**
Θ; Φ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s* : *b* $\Longrightarrow$ *True* **and**
Θ; Φ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow$ *True* **and**
Θ; Φ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow$ *True* **and**
$\Theta \vdash_{wf}$ (Φ::Φ) $\Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\Longrightarrow$ Γ=Γ_1@((*x*,b',c') $\#_\Gamma\Gamma_2$) $\Longrightarrow$ *atom x* $\sharp$ Γ_1 $\Longrightarrow$ *atom x* $\sharp$ Δ $\Longrightarrow$ Θ; $\mathcal{B}$; Γ_1@Γ_2 $\vdash_{wf}$ Δ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\Longrightarrow$ *True*

proof(*induct* *arbitrary*: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfE-valI* Θ Φ Γ Δ *v b*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-plusI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-leqI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next

case (*wfE-eqI* Θ Φ Γ Δ *v1 b v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-fstI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-sndI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-concatI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-splitI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-lenI* Θ Φ Γ Δ *v1*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-appI* Θ Φ Γ Δ *f x b c τ s′ v*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-appPI* Θ Φ *B* Γ Δ *b′ bv v τ f x b c s*)
show *?case* **proof**
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *wfE-appPI* **by** *auto*
show ‹ $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} \Delta$ › **using** *wfE-appPI* **by** *auto*
show ‹ $\Theta; \mathcal{B} \vdash_{wf} b'$ › **using** *wfE-appPI* **by** *auto*

have *atom bv* ♯ $\Gamma_1 @ \Gamma_2$ **using** *wfE-appPI fresh-prodN fresh-restrict* **by** *metis*
thus ‹*atom bv* ♯ (Φ, Θ, *B*, $\Gamma_1 @ \Gamma_2$, Δ, *b′*, *v*, (*b-of τ*)[*bv*::=*b′*]$_b$)›
using *wfE-appPI fresh-prodN* **by** *auto*

show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c τ s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
show ‹ $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v : b[bv::=b']_b$ › **using** *wfE-appPI wf-restrict1* **by** *auto*
qed
next
case (*wfE-mvarI* Θ Φ Γ Δ *u τ*)
then show *?case* **using** *e.fresh wf-intros* **by** *metis*
next
case (*wfD-emptyI* Θ Γ)
then show *?case* **using** *c.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfD-cons* Θ *B* Γ Δ *τ u*)
show *?case* **proof**
show $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} \Delta$ **using** *wfD-cons fresh-DCons* **by** *metis*
show $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} \tau$ **using** *wfD-cons fresh-DCons fresh-Pair wf-restrict1* **by** *metis*
show *u* ∉ *fst* ‘ *setD* Δ **using** *wfD-cons* **by** *auto*
qed
next
case (*wfFTNone* Θ *ft*)
then show *?case* **by** *auto*
next

case (*wfFTSome* Θ *bv ft*)
then show *?case* **by** *auto*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
then show *?case* **by** *auto*
qed(*auto*)+

lemmas *wf-restrict=wf-restrict1 wf-restrict2*

lemma *wfT-restrict2*:
fixes τ::τ
assumes *wfT* Θ *ℬ* ((*x*, *b*, *c*) $\#_\Gamma$ Γ) τ **and** *atom x* ♯ τ
shows Θ; *ℬ*; Γ $\vdash_{wf}$ τ
using *wf-restrict1*(*4*)[*of* Θ *ℬ* ((*x*, *b*, *c*) $\#_\Gamma$ Γ) τ *GNil x b c* Γ] *assms fresh-GNil append-g.simps* **by** *auto*

lemma *wfG-intros2*:
assumes *wfC P ℬ* ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) *c*
shows *wfG P ℬ* ((*x*,*b*,*c*) $\#_\Gamma$Γ)
proof −
have *wfG P ℬ* ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) **using** *wfC-wf assms* **by** *auto*
hence ∗:*wfG P ℬ* Γ ∧ *atom x* ♯ Γ ∧ *wfB P ℬ b* **using** *wfG-elims* **by** *metis*
show *?thesis* **using** *assms* **proof**(*cases c* ∈ {*TRUE*,*FALSE*})
case *True*
then show *?thesis* **using** *wfG-cons2I* ∗ **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-cons1I* ∗ *assms* **by** *auto*
qed
qed

8.10 Type Definitions

lemma *wf-theta-weakening1*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ℬ* :: *ℬ* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list* **and** *t*::τ

shows Θ; *ℬ*; Γ $\vdash_{wf}$ *v* : *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* ; Γ $\vdash_{wf}$ *v* : *b* **and**
Θ; *ℬ*; Γ $\vdash_{wf}$ *c* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* ; Γ $\vdash_{wf}$ *c* **and**
Θ; *ℬ* $\vdash_{wf}$ Γ ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* $\vdash_{wf}$ Γ **and**
Θ; *ℬ*; Γ $\vdash_{wf}$ τ ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* ; Γ $\vdash_{wf}$ τ **and**
Θ; *ℬ*; Γ $\vdash_{wf}$ *ts* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* ; Γ $\vdash_{wf}$ *ts* **and**
$\vdash_{wf}$ *P* ⟹ *True* **and**
Θ; *ℬ* $\vdash_{wf}$ *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* $\vdash_{wf}$ *b* **and**
Θ; *ℬ*; Γ $\vdash_{wf}$ *ce* : *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *ℬ* ; Γ $\vdash_{wf}$ *ce* : *b* **and**
Θ $\vdash_{wf}$ *td* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ $\vdash_{wf}$ *td*
proof(*nominal-induct b* **and** *c* **and** Γ **and** τ **and** *ts* **and** *P* **and** *b* **and** *b* **and** *td*
avoiding: Θ′
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-consI s dclist* Θ *dc x b c ℬ* Γ *v*)
show *?case* **proof**

show ‹*AF-typedef s dclist* ∈ *set* Θ′› **using** *wfV-consI* **by** *auto*
show ‹(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist*› **using** *wfV-consI* **by** *auto*
show ‹ Θ′ ; *B* ; Γ $\vdash_{wf}$ *v* : *b* › **using** *wfV-consI* **by** *auto*
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x b′ c B b* Γ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ′› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, ⦃ *x* : *b′* | *c* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹Θ′ ; *B* ; Γ $\vdash_{wf}$ *v* : $b'[bv::=b]_{bb}$ › **using** *wfV-conspI* **by** *auto*
show Θ′ ; *B* $\vdash_{wf}$ *b* **using** *wfV-conspI* **by** *auto*
show *atom bv* ♯ (Θ′, *B*, Γ, *b*, *v*) **using** *wfV-conspI fresh-prodN* **by** *auto*
qed
next
case (*wfTI z* Θ *B* Γ *b c*)
thus *?case* **using** *Wellformed.wfTI* **by** *auto*
next
case (*wfB-consI* Θ *s dclist*)
show *?case* **proof**
show ‹ $\vdash_{wf}$ Θ′ › **using** *wfB-consI* **by** *auto*
show ‹*AF-typedef s dclist* ∈ *set* Θ′› **using** *wfB-consI* **by** *auto*
qed
next
case (*wfB-appI* Θ *B b s bv dclist*)
show *?case* **proof**
show ‹ $\vdash_{wf}$ Θ′ › **using** *wfB-appI* **by** *auto*
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ′› **using** *wfB-appI* **by** *auto*
show Θ′ ; *B* $\vdash_{wf}$ *b* **using** *wfB-appI* **by** *simp*
qed
qed(*metis wf-intros*)+

lemma *wf-theta-weakening2*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *B* :: *B* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *t*::τ
shows
Θ; Φ; *B*; Γ ; Δ $\vdash_{wf}$ *e* : *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; Φ ; *B* ; Γ ; Δ $\vdash_{wf}$ *e* : *b* **and**
Θ; Φ; *B*; Γ ; Δ $\vdash_{wf}$ *s* : *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; Φ ; *B* ; Γ ; Δ $\vdash_{wf}$ *s* : *b* **and**
Θ; Φ; *B*; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; Φ ; *B* ; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* **and**
Θ; Φ; *B*; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; Φ ; *B* ; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* **and**
Θ $\vdash_{wf}$ (Φ::Φ) ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ $\vdash_{wf}$ (Φ::Φ) **and**
Θ; *B*; Γ $\vdash_{wf}$ Δ ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; *B* ; Γ $\vdash_{wf}$ Δ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; Φ $\vdash_{wf}$ *ftq* **and**
Θ ; Φ ; *B* $\vdash_{wf}$ *ft* ⟹ $\vdash_{wf}$ Θ′ ⟹ *set* Θ ⊆ *set* Θ′ ⟹ Θ′ ; Φ ; *B* $\vdash_{wf}$ *ft*

proof(*nominal-induct b* **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Θ′
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-appPI* Θ Φ *B* Γ Δ *b′ bv v* τ *f x b c s*)
show *?case* **proof**

show ‹ Θ' $\vdash_{wf}$ Φ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$ $\vdash_{wf}$ b' › **using** *wfE-appPI wf-theta-weakening1* **by** *auto*
show ‹*atom bv* ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, b', v, (b-*of* τ)[bv::=b']$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ v : b[bv::=b']$_b$ › **using** *wfE-appPI wf-theta-weakening1* **by** *auto*
qed
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
show *?case* **proof**
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ v : *B-id tid* › **using** *wfS-matchI wf-theta-weakening1* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ'› **using** *wfS-matchI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *wfS-matchI* **by** *auto*
show ‹ Θ' $\vdash_{wf}$ Φ › **using** *wfS-matchI* **by** *auto*
show ‹Θ'; Φ; $\mathcal{B}$; Γ; Δ; *tid*; *dclist* $\vdash_{wf}$ *cs* : *b* › **using** *wfS-matchI* **by** *auto*
qed
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
show *?case* **proof**
show ‹ Θ' ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ › **using** *wfS-varI wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ v : b-*of* τ › **using** *wfS-varI wf-theta-weakening1* **by** *auto*
show ‹*atom u* ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, τ, v, b)› **using** *wfS-varI* **by** *auto*
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; (u, τ) $\#_{\Delta}$ $\Delta \vdash_{wf}$ s : b › **using** *wfS-varI* **by** *auto*
qed
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b' x s b*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ e : b' › **using** *wfS-letI* **by** *auto*
show ‹ Θ' ; Φ ; $\mathcal{B}$; (x, b', *TRUE*) $\#_{\Gamma}$ Γ ; $\Delta \vdash_{wf}$ s : b › **using** *wfS-letI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ › **using** *wfS-letI* **by** *auto*
show ‹*atom x* ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, e, b)› **using** *wfS-letI* **by** *auto*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* : b-*of* τ › **using** *wfS-let2I* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ › **using** *wfS-let2I wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; Φ ; $\mathcal{B}$; (x, b-*of* τ, *TRUE*) $\#_{\Gamma}$ Γ ; $\Delta \vdash_{wf}$ *s2* : b › **using** *wfS-let2I* **by** *auto*
show ‹*atom x* ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, *s1*, b, τ)› **using** *wfS-let2I* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; (x, b-*of* τ, *TRUE*) $\#_{\Gamma}$ Γ ; $\Delta \vdash_{wf}$ s : b › **using** *wfS-branchI* **by** *auto*
show ‹*atom x* ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, Γ, τ)› **using** *wfS-branchI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfPhi-consI f* Φ Θ *ft*)
show *?case* **proof**
show *f* ∉ *name-of-fun* ‘ *set* Φ **using** *wfPhi-consI* **by** *auto*

show Θ' ; $\Phi \vdash_{wf}$ *ft* **using** *wfPhi-consI* **by** *auto*
show $\Theta' \vdash_{wf} \Phi$ **using** *wfPhi-consI* **by** *auto*
qed
next
case (*wfFTNone* Θ *ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTSome* Θ *bv ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
thus *?case* **using** *Wellformed.wfFTI wf-theta-weakening1* **by** *simp*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$ › **using** *wfS-assertI wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} c$ › **using** *wfS-assertI wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-assertI wf-theta-weakening1* **by** *auto*
have *atom x* $\sharp$ Θ' **using** *wf-supp*(*6*)[*OF* ‹$\vdash_{wf}$ Θ' ›] *fresh-def* **by** *auto*
thus ‹*atom x* $\sharp$ (Φ, Θ', $\mathcal{B}$, Γ, Δ, *c*, *b*, *s*)› **using** *wfS-assertI fresh-prodN fresh-def* **by** *simp*
qed
qed(*metis wf-intros wf-theta-weakening1*)+

lemmas *wf-theta-weakening* = *wf-theta-weakening1 wf-theta-weakening2*

lemma *lookup-wfTD*:
fixes *td*::*type-def*
assumes *td* $\in$ *set* Θ **and** $\vdash_{wf}$ Θ
shows $\Theta \vdash_{wf}$ *td*
using *assms* **proof**(*induct* Θ)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons td'* Θ')
then consider *td* = *td'* | *td* $\in$ *set* Θ' **by** *auto*
then have $\Theta' \vdash_{wf}$ *td* **proof**(*cases*)
case *1*
then show *?thesis* **using** *Cons* **using** *wfTh-elims* **by** *auto*
next
case *2*
then show *?thesis* **using** *Cons* **using** *wfTh-elims* **by** *auto*
qed
then show *?case* **using** *wf-theta-weakening Cons* **by** (*meson set-subset-Cons*)
qed

8.10.1 Simple

lemma *wfTh-dclist-unique*:
assumes *wfTh* Θ **and** *AF-typedef tid dclist1* $\in$ *set* Θ **and** *AF-typedef tid dclist2* $\in$ *set* Θ
shows *dclist1* = *dclist2*
using *assms* **proof**(*induct* Θ *rule*: Θ-*induct*)
case *TNil*

then show *?case* **by** *auto*
next
case (*AF-typedef tid′ dclist′* Θ′)
then show *?case* **using** *wfTh-elims*
by (*metis image-eqI name-of-type.simps*(*1*) *set-ConsD type-def.eq-iff*(*1*))
next
case (*AF-typedef-poly tid bv dclist* Θ′)
then show *?case* **using** *wfTh-elims* **by** *auto*
qed

lemma *wfTs-ctor-unique*:
fixes *dclist*::(*string*∗τ) *list*
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ *dclist* **and** (*c*, *t1*) ∈ *set dclist* **and** (*c*,*t2*) ∈ *set dclist*
shows *t1* = *t2*
using *assms* **proof**(*induct dclist rule*: *list.inducts*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
consider *x1* = (*c*,*t1*) | *x1* = (*c*,*t2*) | *x1* ≠ (*c*,*t1*) ∧ *x1* ≠ (*c*,*t2*) **by** *auto*
thus *?case* **proof**(*cases*)
case *1*
then show *?thesis* **using** *Cons wfTs-elims set-ConsD*
by (*metis fst-conv image-eqI prod.inject*)
next
case *2*
then show *?thesis* **using** *Cons wfTs-elims set-ConsD*
by (*metis fst-conv image-eqI prod.inject*)
next
case *3*
then show *?thesis* **using** *Cons wfTs-elims* **by** (*metis set-ConsD*)
qed
qed

lemma *wfTD-ctor-unique*:
assumes Θ $\vdash_{wf}$ (*AF-typedef tid dclist*) **and** (*c*, *t1*) ∈ *set dclist* **and** (*c*,*t2*) ∈ *set dclist*
shows *t1* = *t2*
using *wfTD-elims wfTs-elims assms wfTs-ctor-unique* **by** *metis*

lemma *wfTh-ctor-unique*:
assumes *wfTh* Θ **and** *AF-typedef tid dclist* ∈ *set* Θ **and** (*c*, *t1*) ∈ *set dclist* **and** (*c*,*t2*) ∈ *set dclist*

shows *t1* = *t2*
using *lookup-wfTD wfTD-ctor-unique assms* **by** *metis*

lemma *wfTs-supp-t*:
fixes *dclist*::(*string*∗τ) *list*
assumes (*c*,*t*) ∈ *set dclist* **and** Θ ; *B* ; *GNil* $\vdash_{wf}$ *dclist*
shows *supp t* ⊆ *supp B*
using *assms* **proof**(*induct dclist arbitrary*: *c t rule*:*list.induct*)
case *Nil*
then show *?case* **by** *auto*

next
  case (*Cons ct dclist'*)
  then consider *ct* = (*c*,*t*) | (*c*,*t*) ∈ *set dclist'* **by** *auto*
  then show *?case* **proof**(*cases*)
    case *1*
    then have Θ ; *B* ; *GNil* $\vdash_{wf}$ *t* **using** *Cons wfTs-elims* **by** *blast*
    thus *?thesis* **using** *wfT-supp atom-dom.simps* **by** *force*
  next
    case *2*
    then show *?thesis* **using** *Cons wfTs-elims* **by** *metis*
  qed
qed

lemma *wfTh-lookup-supp-empty*:
  fixes *t*::τ
  assumes *AF-typedef tid dclist* ∈ *set* Θ **and** (*c*,*t*) ∈ *set dclist* **and** $\vdash_{wf}$ Θ
  shows *supp t* = {}
proof −
  have Θ ; {||} ; *GNil* $\vdash_{wf}$ *dclist* **using** *assms lookup-wfTD wfTD-elims* **by** *metis*
  thus *?thesis* **using** *wfTs-supp-t assms* **by** *force*
qed

lemma *wfTh-supp-b*:
  assumes *AF-typedef tid dclist* ∈ *set* Θ **and** (*dc*,⦃ *z* : *b* | *c* ⦄) ∈ *set dclist* **and** $\vdash_{wf}$ Θ
  shows *supp b* = {}
  using *assms wfTh-lookup-supp-empty τ.supp* **by** *blast*

lemma *wfTh-b-eq-iff*:
  fixes *bva1*::*bv* **and** *bva2*::*bv* **and** *dc*::*string*
  assumes (*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist1* **and** (*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2* **and**
   wfTs P {|*bva1*|} *GNil dclist1* **and** *wfTs P* {|*bva2*|} *GNil dclist2*
  [[*atom bva1*]]*lst*.*dclist1* = [[*atom bva2*]]*lst*.*dclist2*
 shows [[*atom bva1*]]*lst*. (*dc*,⦃ *x1* : *b1* | *c1* ⦄) = [[*atom bva2*]]*lst*. (*dc*,⦃ *x2* : *b2* | *c2* ⦄)
using *assms* **proof**(*induct dclist1 arbitrary*: *dclist2*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons dct1' dclist1'*)
  show *?case* **proof**(*cases dclist2* = [])
    case *True*
    then show *?thesis* **using** *Cons* **by** *auto*
  next
    case *False*
    then obtain *dct2'* **and** *dclist2'* **where** *cons*:*dct2'* # *dclist2'* = *dclist2* **using** *list.exhaust* **by** *metis*
     hence ∗:[[*atom bva1*]]*lst*. *dclist1'* = [[*atom bva2*]]*lst*. *dclist2'* ∧ [[*atom bva1*]]*lst*. *dct1'* = [[*atom bva2*]]*lst*. *dct2'*
      using *Cons lst-head-cons Cons cons* **by** *metis*
    hence ∗∗: *fst dct1'* = *fst dct2'* **using** *lst-fst*[*THEN lst-pure*]
     by (*metis* (*no-types*) ‹[[*atom bva1*]]*lst*. *dclist1'* = [[*atom bva2*]]*lst*. *dclist2'* ∧ [[*atom bva1*]]*lst*. *dct1'* = [[*atom bva2*]]*lst*. *dct2'*›
          ‹⋀*x2 x1 t2' t2a t2 t1*. [[*atom x1*]]*lst*. (*t1*, *t2a*) = [[*atom x2*]]*lst*. (*t2*, *t2'*) ⟹ *t1* = *t2*› *fst-conv surj-pair*)

show *?thesis* **proof**(*cases fst dct1′ = dc*)
case *True*
have *dc* ∉ *fst ‘ set dclist1′* **using** *wfTs-elims Cons* **by** (*metis True fstI*)
hence *1*:(*dc*, ⦃ *x1* : *b1* | *c1* ⦄) = *dct1′* **using** *Cons* **by** (*metis fstI image-iff set-ConsD*)
have *dc* ∉ *fst ‘ set dclist2′* **using** *wfTs-elims Cons cons*
by (*metis* ∗∗ *True fstI*)
hence *2*:(*dc*, ⦃ *x2* : *b2* | *c2* ⦄) = *dct2′* **using** *Cons cons* **by** (*metis fst-conv image-eqI set-ConsD*)
then show *?thesis* **using** *Cons* ∗ *1 2* **by** *blast*
next
case *False*
hence *fst dct2′* ≠ *dc* **using** ∗∗ **by** *auto*
hence (*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist1′* ∧ (*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2′* **using** *Cons cons False*
by (*metis fstI set-ConsD*)
moreover have [[*atom bva1*]]*lst*. *dclist1′* = [[*atom bva2*]]*lst*. *dclist2′* **using** ∗ *False* **by** *metis*
ultimately show *?thesis* **using** *Cons* ∗∗ ∗
using *cons wfTs-elims*(*2*) **by** *blast*
qed
qed
qed

8.10.2 Polymorphic

lemma *wfTh-wfTs-poly*:
fixes *dclist*::(*string* ∗ τ) *list*
assumes *AF-typedef-poly tyid bva dclist* ∈ *set P* **and** $\vdash_{wf}$ *P*
shows *P* ; {|*bva*|} ; *GNil* $\vdash_{wf}$ *dclist*
proof −
have ∗:*P* $\vdash_{wf}$ *AF-typedef-poly tyid bva dclist* **using** *lookup-wfTD assms* **by** *simp*

obtain *bv lst* **where** ∗:*P* ; {|*bv*|} ; *GNil* $\vdash_{wf}$ *lst* ∧
(∀ *c*. *atom c* ♯ (*dclist*, *lst*) ⟶ *atom c* ♯ (*bva*, *bv*, *dclist*, *lst*) ⟶ (*bva* ↔ *c*) · *dclist* = (*bv* ↔ *c*) · *lst*)
using *wfTD-elims*(*2*)[*OF* ∗] **by** *metis*

obtain *c*::*bv* **where** ∗∗:*atom c* ♯ ((*dclist*, *lst*),(*bva*, *bv*, *dclist*, *lst*)) **using** *obtain-fresh* **by** *metis*
have *P* ; {|*bv*|} ; *GNil* $\vdash_{wf}$ *lst* **using** ∗ **by** *metis*
hence *wfTs* ((*bv* ↔ *c*) · *P*) ((*bv* ↔ *c*) · {|*bv*|}) ((*bv* ↔ *c*) · *GNil*) ((*bv* ↔ *c*) · *lst*) **using** ∗∗ *wfTs.eqvt* **by** *metis*
hence *wfTs P* {|*c*|} *GNil* ((*bva* ↔ *c*) · *dclist*) **using** ∗ *theta-flip-eq fresh-GNil assms*
proof −
have ∀ *b ba*. (*ba*::*bv* ↔ *b*) · *P* = *P* **by** (*metis* ‹$\vdash_{wf}$ *P*› *theta-flip-eq*)
then show *?thesis*
using ∗ ∗∗ ‹(*bv* ↔ *c*) · *P* ; (*bv* ↔ *c*) · {|*bv*|} ; (*bv* ↔ *c*) · *GNil* $\vdash_{wf}$ (*bv* ↔ *c*) · *lst*› **by** *fastforce*
qed
hence *wfTs* ((*bva* ↔ *c*) · *P*) ((*bva* ↔ *c*) · {|*bva*|}) ((*bva* ↔ *c*) · *GNil*) ((*bva* ↔ *c*) · *dclist*)
using *wfTs.eqvt fresh-GNil*
by (*simp add*: *assms*(*2*) *theta-flip-eq2*)

thus *?thesis* **using** *wfTs.eqvt permute-flip-cancel* **by** *metis*
qed

lemma *wfTh-dclist-poly-unique*:

assumes *wfTh* Θ **and** *AF-typedef-poly tid bva dclist1* ∈ *set* Θ **and** *AF-typedef-poly tid bva2 dclist2* ∈ *set* Θ
shows [[*atom bva*]]*lst. dclist1* = [[*atom bva2*]]*lst.dclist2*
using *assms* **proof**(*induct* Θ *rule*: Θ-*induct*)
case *TNil*
then show *?case* **by** *auto*
next
case (*AF-typedef tid′ dclist′* Θ′)
then show *?case* **using** *wfTh-elims* **by** *auto*
next
case (*AF-typedef-poly tid bv dclist* Θ′)
then show *?case* **using** *wfTh-elims image-eqI name-of-type.simps set-ConsD type-def.eq-iff*
by (*metis Abs1-eq*(*3*))
qed

lemma *wfTh-poly-lookup-supp*:
fixes *t*::τ
assumes *AF-typedef-poly tid bv dclist* ∈ *set* Θ **and** (*c*,*t*) ∈ *set dclist* **and** $\vdash_{wf}$ Θ
shows *supp t* ⊆ {*atom bv*}
proof −
have *supp dclist* ⊆ {*atom bv*} **using** *assms lookup-wfTD wf-supp1 type-def.supp*
by (*metis Diff-single-insert Un-subset-iff list.simps*(*15*) *supp-Nil supp-of-atom-list*)
then show *?thesis* **using** *assms*(*2*) **proof**(*induct dclist*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a dclist*)
then show *?case* **using** *supp-Pair supp-Cons*
by (*metis* (*mono-tags*, *opaque-lifting*) *Un-empty-left Un-empty-right pure-supp subset-Un-eq subset-singletonD supp-list-member*)
qed
qed

lemma *wfTh-poly-supp-b*:
assumes *AF-typedef-poly tid bv dclist* ∈ *set* Θ **and** (*dc*,⦃ *z* : *b* | *c* ⦄) ∈ *set dclist* **and** $\vdash_{wf}$ Θ
shows *supp b* ⊆ {*atom bv*}
using *assms wfTh-poly-lookup-supp* τ*.supp* **by** *force*

lemma *subst-g-inside*:
fixes *x*::*x* **and** *c*::*c* **and** Γ::Γ **and** Γ′::Γ
assumes *wfG P B* (Γ′ @ (*x*, *b*, $c[z::=V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)
shows (Γ′ @ (*x*, *b*, $c[z::=V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)$[x::=v]_{\Gamma v}$ = (Γ′$[x::=v]_{\Gamma v}$@Γ)
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gb.simps* **by** *simp*
next
case (*GCons x′ b′ c′ G*)
hence *wfg*:*wfG P B* (*G* @ (*x*, *b*, $c[z::=V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) ∧ *atom x′* ♯ (*G* @ (*x*, *b*, $c[z::=V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) **using** *wfG-elims*(*2*)
using *GCons.prems append-g.simps* **by** *metis*
hence *atom x* ∉ *atom-dom* ((*x′*, *b′*, *c′*) $\#_\Gamma$ *G*) **using** *GCons wfG-inside-fresh* **by** *fast*
hence *x*≠*x′*

using *GCons append-Cons wfG-inside-fresh atom-dom.simps toSet.simps* **by** *simp*
hence $((GCons\ (x', b', c')\ G)\ @\ (GCons\ (x, b, c[z::=V\text{-}var\ x]_{cv})\ \Gamma))[x::=v]_{\Gamma v} =$
$(GCons\ (x', b', c')\ (G\ @\ (GCons\ (x, b, c[z::=V\text{-}var\ x]_{cv})\ \Gamma)))[x::=v]_{\Gamma v}$ **by** *auto*
also have ... = $GCons\ (x', b', c'[x::=v]_{cv})\ ((G\ @\ (GCons\ (x, b, c[z::=V\text{-}var\ x]_{cv})\ \Gamma))[x::=v]_{\Gamma v})$
using *subst-gv.simps* ‹$x\neq x'$› **by** *simp*
also have ... = $(x', b', c'[x::=v]_{cv})\ \#_\Gamma\ (G[x::=v]_{\Gamma v}\ @\ \Gamma)$ **using** *GCons wfg* **by** *blast*
also have ... = $((x', b', c')\ \#_\Gamma\ G)[x::=v]_{\Gamma v}\ @\ \Gamma$ **using** *subst-gv.simps* ‹$x\neq x'$› **by** *simp*
finally show *?case* **by** *auto*
qed

lemma *wfTh-td-eq*:
assumes *td1* ∈ *set* (*td2* # *P*) **and** *wfTh* (*td2* # *P*) **and** *name-of-type td1* = *name-of-type td2*
shows *td1* = *td2*
proof(*rule ccontr*)
assume *as*: *td1* ≠ *td2*
have *name-of-type td2* ∉ *name-of-type* ‘ *set P* **using** *wfTh-elims*(*2*)[*OF assms*(*2*)] **by** *metis*
moreover have *td1* ∈ *set P* **using** *assms as* **by** *simp*
ultimately have *name-of-type td1* ≠ *name-of-type td2*
by (*metis rev-image-eqI*)
thus *False* **using** *assms* **by** *auto*
qed

lemma *wfTh-td-unique*:
assumes *td1* ∈ *set P* **and** *td2* ∈ *set P* **and** *wfTh P* **and** *name-of-type td1* = *name-of-type td2*
shows *td1* = *td2*
using *assms* **proof**(*induct P rule*: *list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons td* Θ′)
consider *td* = *td1* | *td* = *td2* | *td* ≠ *td1* ∧ *td* ≠ *td2* **by** *auto*
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **using** *Cons wfTh-elims wfTh-td-eq* **by** *metis*
next
case *2*
then show *?thesis* **using** *Cons wfTh-elims wfTh-td-eq* **by** *metis*
next
case *3*
then show *?thesis* **using** *Cons wfTh-elims* **by** *auto*
qed
qed

lemma *wfTs-distinct*:
fixes *dclist*::(*string* ∗ τ) *list*
assumes $\Theta\ ;\ B\ ;\ GNil \vdash_{wf}\ dclist$
shows *distinct* (*map fst dclist*)
using *assms* **proof**(*induct dclist rule*: *list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)

then show *?case*
by (*metis Cons.hyps Cons.prems distinct.simps*(*2*) *fst-conv list.set-map list.simps*(*9*) *wfTs-elims*(*2*))

qed

lemma *wfTh-dclist-distinct*:
assumes *AF-typedef s dclist* ∈ *set P* **and** *wfTh P*
shows *distinct* (*map fst dclist*)
proof −
have *wfTD P* (*AF-typedef s dclist*) **using** *assms lookup-wfTD* **by** *auto*
hence *wfTs P* {||} *GNil dclist* **using** *wfTD-elims* **by** *metis*
thus *?thesis* **using** *wfTs-distinct* **by** *metis*
qed

lemma *wfTh-dc-t-unique2*:
assumes *AF-typedef s dclist′* ∈ *set P* **and** (*dc*,*tc′*) ∈ *set dclist′* **and** *AF-typedef s dclist* ∈ *set P* **and** *wfTh P* **and**
(*dc*, *tc*) ∈ *set dclist*
shows *tc*= *tc′*
proof −
have *dclist* = *dclist′* **using** *assms wfTh-td-unique name-of-type.simps* **by** *force*
moreover have *distinct* (*map fst dclist*) **using** *wfTh-dclist-distinct assms* **by** *auto*
ultimately show *?thesis* **using** *assms*
by (*meson eq-key-imp-eq-value*)
qed

lemma *wfTh-dc-t-unique*:
assumes *AF-typedef s dclist′* ∈ *set P* **and** (*dc*, ⦃ *x′* : *b′* | *c′* ⦄) ∈ *set dclist′* **and** *AF-typedef s dclist* ∈ *set P* **and** *wfTh P* **and**
(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist*
shows ⦃ *x′* : *b′* | *c′* ⦄= ⦃ *x* : *b* | *c* ⦄
using *assms wfTh-dc-t-unique2* **by** *metis*

lemma *wfTs-wfT*:
fixes *dclist*::(*string* ∗*τ*) *list* **and** *t*::*τ*
assumes Θ; *B*; *GNil* $\vdash_{wf}$ *dclist* **and** (*dc*,*t*) ∈ *set dclist*
shows Θ; *B*; *GNil* $\vdash_{wf}$ *t*
using *assms* **proof**(*induct dclist rule*:*list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
thus *?case* **using** *wfTs-elims*(*2*)[*OF Cons*(*2*)] **by** *auto*
qed

lemma *wfTh-wfT*:
fixes *t*::*τ*
assumes *wfTh P* **and** *AF-typedef tid dclist* ∈ *set P* **and** (*dc*,*t*) ∈ *set dclist*
shows *P* ; {||} ; *GNil* $\vdash_{wf}$ *t*
proof −
have *P* $\vdash_{wf}$ *AF-typedef tid dclist* **using** *lookup-wfTD assms* **by** *auto*
hence *P* ; {||} ; *GNil* $\vdash_{wf}$ *dclist* **using** *wfTD-elims* **by** *auto*

thus *?thesis* **using** *wfTs-wfT assms* **by** *auto*
qed

lemma *td-lookup-eq-iff*:
fixes *dc* :: *string* **and** *bva1*::*bv* **and** *bva2*::*bv*
assumes [[*atom bva1*]]*lst. dclist1* = [[*atom bva2*]]*lst. dclist2* **and** (*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist1*
shows ∃ *x2 b2 c2*. (*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2*
using *assms* **proof**(*induct dclist1 arbitrary*: *dclist2*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons dct1′ dclist1′*)
then obtain *dct2′* **and** *dclist2′* **where** *cons*:*dct2′* # *dclist2′* = *dclist2* **using** *lst-head-cons-neq-nil*[*OF Cons*(*2*)] *list.exhaust* **by** *metis*
hence ∗:[[*atom bva1*]]*lst. dclist1′* = [[*atom bva2*]]*lst. dclist2′* ∧ [[*atom bva1*]]*lst. dct1′* = [[*atom bva2*]]*lst. dct2′*
using *Cons lst-head-cons Cons cons* **by** *metis*
show *?case* **proof**(*cases dc*=*fst dct1′*)
case *True*
hence *dc* = *fst dct2′* **using** ∗ *lst-fst*[*THEN lst-pure*]
proof −
show *?thesis*
by (*metis* (*no-types*) *local*.∗ *True* ‹⋀*x2 x1 t2′ t2a t2 t1*. [[*atom x1*]]*lst*. (*t1*, *t2a*) = [[*atom x2*]]*lst*. (*t2*, *t2′*) ⟹ *t1* = *t2*› *prod.exhaust-sel*)
qed
obtain *x2 b2* **and** *c2* **where** *snd dct2′* = ⦃ *x2* : *b2* | *c2* ⦄ **using** *obtain-fresh-z* **by** *metis*
hence (*dc*, ⦃ *x2* : *b2* | *c2* ⦄) = *dct2′* **using** ‹*dc* = *fst dct2′*›
by (*metis prod.exhaust-sel*)
then show *?thesis* **using** *cons* **by** *force*
next
case *False*
hence (*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist1′* **using** *Cons* **by** *auto*
then show *?thesis* **using** *Cons*
by (*metis local*.∗ *cons list.set-intros*(*2*))
qed
qed

lemma *lst-t-b-eq-iff*:
fixes *bva1*::*bv* **and** *bva2*::*bv*
assumes [[*atom bva1*]]*lst*. ⦃ *x1* : *b1* | *c1* ⦄ = [[*atom bva2*]]*lst*. ⦃ *x2* : *b2* | *c2* ⦄
shows [[*atom bva1*]]*lst. b1* = [[*atom bva2*]]*lst.b2*
proof(*subst Abs1-eq-iff-all*(*3*)[*of bva1 b1 bva2 b2*],*rule*,*rule*,*rule*)
fix *c*::*bv*
assume *atom c* ♯ (⦃ *x1* : *b1* | *c1* ⦄ , ⦃ *x2* : *b2* | *c2* ⦄) **and** *atom c* ♯ (*bva1*, *bva2*, *b1*, *b2*)

show (*bva1* ↔ *c*) · *b1* = (*bva2* ↔ *c*) · *b2* **using** *assms Abs1-eq-iff*(*3*) *assms*
by (*metis Abs1-eq-iff-fresh*(*3*) ‹*atom c* ♯ (*bva1*, *bva2*, *b1*, *b2*)› *τ.fresh τ.perm-simps type-eq-subst-eq2*(*2*))
qed

lemma *wfTh-typedef-poly-b-eq-iff*:
assumes *AF-typedef-poly tyid bva1 dclist1* ∈ *set P* **and** (*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist1*
and *AF-typedef-poly tyid bva2 dclist2* ∈ *set P* **and** (*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2* **and** $\vdash_{wf}$ *P*

shows *b1*[*bva1*::=*b*]$_{bb}$ = *b2*[*bva2*::=*b*]$_{bb}$
proof −
have [[*atom bva1*]]*lst*. *dclist1* = [[*atom bva2*]]*lst*.*dclist2* **using** *assms wfTh-dclist-poly-unique* **by** *metis*
hence [[*atom bva1*]]*lst*. (*dc*,⦃ *x1* : *b1* | *c1* ⦄) = [[*atom bva2*]]*lst*. (*dc*,⦃ *x2* : *b2* | *c2* ⦄) **using** *wfTh-b-eq-iff assms wfTh-wfTs-poly* **by** *metis*
hence [[*atom bva1*]]*lst*. ⦃ *x1* : *b1* | *c1* ⦄ = [[*atom bva2*]]*lst*. ⦃ *x2* : *b2* | *c2* ⦄ **using** *lst-snd* **by** *metis*
hence [[*atom bva1*]]*lst*. *b1* = [[*atom bva2*]]*lst*.*b2* **using** *lst-t-b-eq-iff* **by** *metis*
thus *?thesis* **using** *subst-b-flip-eq-two subst-b-b-def* **by** *metis*
qed

8.11 Equivariance Lemmas

lemma *x-not-in-u-set*[*simp*]:
fixes *x*::*x* **and** *us*::*u fset*
shows *atom x* ∉ *supp us*
by(*induct us*,*auto*, *simp add*: *supp-finsert supp-at-base*)

lemma *wfS-flip-eq*:
fixes *s1*::*s* **and** *x1*::*x* **and** *s2*::*s* **and** *x2*::*x* **and** Δ::Δ
assumes [[*atom x1*]]*lst*. *s1* = [[*atom x2*]]*lst*. *s2* **and** [[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2* **and** [[*atom x1*]]*lst*. *c1* = [[*atom x2*]]*lst*. *c2* **and** *atom x2* ♯ Γ **and**
Θ; *ℬ*; Γ $\vdash_{wf}$ Δ **and**
Θ ; Φ ; *ℬ* ; (*x1*, *b*, *c1*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s1* : *b-of t1*
shows Θ ; Φ ; *ℬ* ; (*x2*, *b*, *c2*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s2* : *b-of t2*
proof(*cases x1*=*x2*)
case *True*
hence *s1* = *s2* ∧ *t1* = *t2* ∧ *c1* = *c2* **using** *assms Abs1-eq-iff* **by** *metis*
then show *?thesis* **using** *assms True* **by** *simp*
next
case *False*
have $\vdash_{wf}$ Θ ∧ Θ $\vdash_{wf}$ Φ ∧ Θ; *ℬ*; Γ $\vdash_{wf}$ Δ **using** *wfX-wfY assms* **by** *metis*
moreover have *atom x1* ♯ Γ **using** *wfX-wfY wfG-elims assms* **by** *metis*
moreover hence *atom x1* ♯ Δ ∧ *atom x2* ♯ Δ **using** *wfD-x-fresh assms* **by** *auto*
ultimately have Θ ; Φ ; *ℬ* ; (*x2* ↔ *x1*) • ((*x1*, *b*, *c1*) $\#_\Gamma$ Γ) ; Δ $\vdash_{wf}$ (*x2* ↔ *x1*) • *s1* : (*x2* ↔ *x1*) • *b-of t1*
using *wfS.eqvt theta-flip-eq phi-flip-eq assms flip-base-eq beta-flip-eq flip-fresh-fresh supp-b-empty* **by** *metis*
hence Θ ; Φ ; *ℬ* ; ((*x2*, *b*, (*x2* ↔ *x1*) • *c1*) $\#_\Gamma$ ((*x2* ↔ *x1*) • Γ)) ; Δ $\vdash_{wf}$ (*x2* ↔ *x1*) • *s1* : *b-of* ((*x2* ↔ *x2*) • *t1*) **by** *fastforce*
thus *?thesis* **using** *assms Abs1-eq-iff*
proof −
have *f1*: *x2* = *x1* ∧ *t2* = *t1* ∨ *x2* ≠ *x1* ∧ *t2* = (*x2* ↔ *x1*) • *t1* ∧ *atom x2* ♯ *t1*
by (*metis* (*full-types*) *Abs1-eq-iff*(*3*) ‹[[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2*›)
then have *x2* ≠ *x1* ∧ *s2* = (*x2* ↔ *x1*) • *s1* ∧ *atom x2* ♯ *s1* ⟶ *b-of t2* = (*x2* ↔ *x1*) • *b-of t1*
by (*metis b-of.eqvt*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *Abs1-eq-iff*(*3*) *G-cons-flip-fresh3* ‹[[*atom x1*]]*lst*. *c1* = [[*atom x2*]]*lst*. *c2*› ‹[[*atom x1*]]*lst*. *s1* = [[*atom x2*]]*lst*. *s2*› ‹Θ ; Φ ; *ℬ* ; (*x1*, *b*, *c1*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s1* : *b-of t1*› ‹Θ ; Φ ; *ℬ* ; (*x2* ↔ *x1*) • ((*x1*, *b*, *c1*) $\#_\Gamma$ Γ) ; Δ $\vdash_{wf}$ (*x2* ↔ *x1*) • *s1* : (*x2* ↔ *x1*) • *b-of t1*› ‹*atom x1* ♯ Γ› ‹*atom x2* ♯ Γ›)
qed
qed

8.12 Lookup

lemma *wf-not-in-prefix*:
  assumes Θ ; $B \vdash_{wf}$ $(\Gamma'@(x,b1,c1)\ \#_\Gamma\Gamma)$
  shows $x \notin$ *fst* ‘ *toSet* Γ'
using *assms* **proof**(*induct* Γ' *rule*: Γ.*induct*)
  case *GNil*
  then show *?case* **by** *simp*
next
  case (*GCons xbc* Γ')
  then obtain x' **and** b' **and** $c'::c$ **where** *xbc*: $xbc=(x',b',c')$
    using *prod-cases3* **by** *blast*
  hence $*$:$(xbc\ \ \#_\Gamma\ \Gamma')$ @ $(x,\ b1,\ c1)\ \ \#_\Gamma\ \Gamma$ = $((x',b',c')\ \#_\Gamma(\Gamma'@\ ((x,\ b1,\ c1)\ \ \#_\Gamma\ \Gamma)))$ **by** *simp*
  hence *atom* $x' \sharp$ $(\Gamma'@(x,b1,c1)\ \#_\Gamma\Gamma)$ **using** *wfG-elims*(*2*) *GCons* **by** *metis*

  moreover have Θ ; $B \vdash_{wf}$ $(\Gamma'$ @ $(x,\ b1,\ c1)\ \ \#_\Gamma\ \Gamma)$ **using** *GCons wfG-elims* $*$ **by** *metis*
  ultimately have *atom* $x' \notin$ *atom-dom* $(\Gamma'@(x,b1,c1)\ \#_\Gamma\Gamma)$ **using** *wfG-dom-supp GCons append-g.simps xbc fresh-def* **by** *fast*
  hence $x' \neq x$ **using** *GCons fresh-GCons xbc* **by** *fastforce*
  then show *?case* **using** *GCons xbc toSet.simps*
    using *Un-commute* ‹Θ ; $B \vdash_{wf}$ Γ' @ $(x,\ b1,\ c1)\ \ \#_\Gamma\ \Gamma$› *atom-dom.simps* **by** *auto*
qed

lemma *lookup-inside-wf*[*simp*]:
  assumes Θ ; $B \vdash_{wf}$ $(\Gamma'@(x,b1,c1)\ \#_\Gamma\Gamma)$
  shows *Some* $(b1,c1)$ = *lookup* $(\Gamma'@(x,b1,c1)\ \#_\Gamma\Gamma)$ x
  using *wf-not-in-prefix lookup-inside assms* **by** *fast*

lemma *lookup-weakening*:
  fixes Θ::Θ **and** Γ::Γ **and** Γ'::Γ
  assumes *Some* (b,c) = *lookup* Γ x **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** Θ; $\mathcal{B} \vdash_{wf}$ Γ' **and** Θ; $\mathcal{B} \vdash_{wf}$ Γ
  shows *Some* (b,c) = *lookup* Γ' x
proof −
  have $(x,b,c) \in$ *toSet* Γ $\wedge$ $(\forall\, b'\ c'.\ (x,b',c') \in$ *toSet* $\Gamma \longrightarrow b'=b \wedge c'=c)$ **using** *assms lookup-iff toSet.simps* **by** *force*
  hence $(x,b,c) \in$ *toSet* Γ' **using** *assms* **by** *auto*
  moreover have $(\forall\, b'\ c'.\ (x,b',c') \in$ *toSet* $\Gamma' \longrightarrow b'=b \wedge c'=c)$ **using** *assms wf-g-unique*
    using *calculation* **by** *auto*
  ultimately show *?thesis* **using** *lookup-iff*
    using *assms*(*3*) **by** *blast*
qed

lemma *wfPhi-lookup-fun-unique*:
  fixes Φ::Φ
  assumes $\Theta \vdash_{wf}$ Φ **and** *AF-fundef f fd* $\in$ *set* Φ
  shows *Some* (*AF-fundef f fd*) = *lookup-fun* Φ *f*
using *assms* **proof**(*induct* Φ *rule*: *list.induct*)
  case *Nil*
  then show *?case* **using** *lookup-fun.simps* **by** *simp*
next
  case (*Cons a* Φ')
  then obtain f' **and** fd' **where** *a*:a = *AF-fundef* f' fd' **using** *fun-def.exhaust* **by** *auto*
  have *wf*: $\Theta \vdash_{wf}$ $\Phi' \wedge f' \notin$ *name-of-fun* ‘ *set* Φ' **using** *wfPhi-elims Cons a* **by** *metis*

then show *?case* **using** *Cons lookup-fun.simps* **using** *Cons lookup-fun.simps wf a*
by (*metis image-eqI name-of-fun.simps set-ConsD*)
qed

lemma *lookup-fun-weakening*:
fixes Φ'::Φ
assumes *Some fd* = *lookup-fun* Φ *f* **and** *set* $\Phi \subseteq$ *set* Φ' **and** $\Theta \vdash_{wf} \Phi'$
shows *Some fd* = *lookup-fun* Φ' *f*
using *assms* **proof**(*induct* Φ)
case *Nil*
then show *?case* **using** *lookup-fun.simps* **by** *simp*
next
case (*Cons a* Φ'')
then obtain *f'* **and** *fd'* **where** *a*: *a* = *AF-fundef f' fd'* **using** *fun-def.exhaust* **by** *auto*
then show *?case* **proof**(*cases f*=*f'*)
case *True*
then show *?thesis* **using** *lookup-fun.simps Cons wfPhi-lookup-fun-unique a*
by (*metis lookup-fun-member subset-iff*)
next
case *False*
then show *?thesis* **using** *lookup-fun.simps Cons*
using ‹*a* = *AF-fundef f' fd'*› **by** *auto*
qed
qed

lemma *fundef-poly-fresh-bv*:
assumes *atom bv2* ♯ (*bv1*,*b1*,*c1*,τ*1*,*s1*)
shows ∗ : (*AF-fun-typ-some bv2* (*AF-fun-typ x1* ((*bv1*↔*bv2*) · *b1*) ((*bv1*↔*bv2*) ·*c1*) ((*bv1*↔*bv2*) · τ*1*) ((*bv1*↔*bv2*) · *s1*)) = (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s1*)))
(**is** (*AF-fun-typ-some ?bv ?fun-typ* = *AF-fun-typ-some ?bva ?fun-typa*))
proof −
have *1*:*atom bv2* ∉ *set* [*atom x1*] **using** *bv-not-in-x-atoms* **by** *simp*
have *2*:*bv1* ≠ *bv2* **using** *assms* **by** *auto*
have *3*:(*bv2* ↔ *bv1*) · *x1* = *x1* **using** *pure-fresh flip-fresh-fresh*
by (*simp add*: *flip-fresh-fresh*)
have *AF-fun-typ x1* ((*bv1* ↔ *bv2*) · *b1*) ((*bv1* ↔ *bv2*) · *c1*) ((*bv1* ↔ *bv2*) · τ*1*) ((*bv1* ↔ *bv2*) · *s1*) = (*bv2* ↔ *bv1*) · *AF-fun-typ x1 b1 c1* τ*1 s1*
using *1 2 3 assms* **by** (*simp add*: *flip-commute*)
moreover have (*atom bv2* ♯ *c1* ∧ *atom bv2* ♯ τ*1* ∧ *atom bv2* ♯ *s1* ∨ *atom bv2* ∈ *set* [*atom x1*]) ∧ *atom bv2* ♯ *b1*
using *1 2 3 assms fresh-prod5* **by** *metis*
ultimately show *?thesis* **unfolding** *fun-typ-q.eq-iff Abs1-eq-iff*(*3*) *fun-typ.fresh 1 2* **by** *fastforce*
qed

It is possible to collapse some of the easy to prove inductive cases into a single proof at the qed line but this makes it fragile under change. For example, changing the lemma statement might make one of the previously trivial cases non-trivial and so the collapsing needs to be unpacked. Is there a way to find which case has failed in the qed line?

lemma *wb-b-weakening1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*

shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} v : b$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} c$ **and**
$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' \vdash_{wf} \Gamma$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} \tau$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} ts$ **and**
$\vdash_{wf} P \Longrightarrow$ *True* **and**
wfB $\Theta\ \mathcal{B}\ b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow$ *wfB* $\Theta\ \mathcal{B}'\ b$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} ce : b$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
proof(*nominal-induct* b **and** c **and** Γ **and** τ **and** ts **and** P **and** b **and** b **and** td
avoiding: $\mathcal{B}'$
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-conspI s bv dclist* Θ *dc x* b' *c* $\mathcal{B}$ *b* Γ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ› **using** *wfV-conspI* **by** *metis*
show ‹(*dc*, $\{\!|\ x : b' \mid c\ |\!\}$) $\in$ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}' \vdash_{wf} b$ › **using** *wfV-conspI* **by** *auto*
show ‹*atom bv* $\sharp$ (Θ, $\mathcal{B}'$, Γ, b, v)› **using** *fresh-prodN wfV-conspI* **by** *auto*
thus ‹ $\Theta; \mathcal{B}' ; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb}$ › **using** *wfV-conspI* **by** *simp*
qed
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
show *?case* **proof**
show *atom z* $\sharp$ (Θ, $\mathcal{B}'$, Γ) **using** *wfTI* **by** *auto*
show $\Theta; \mathcal{B}' \vdash_{wf} b$ **using** *wfTI* **by** *auto*
show $\Theta; \mathcal{B}'$; (z, b, *TRUE*) $\#_\Gamma\ \Gamma \vdash_{wf} c$ **using** *wfTI* **by** *auto*
qed
qed((*auto simp add*: *wf-intros* | *metis wf-intros*)+)

lemma *wb-b-weakening2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*

shows
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} e : b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta \vdash_{wf} e : b$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} s : b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta \vdash_{wf} s : b$ **and**
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b$ **and**
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow$ *True* **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} \Delta$ **and**
$\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow$ *True* **and**
$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' \vdash_{wf} ft$
proof(*nominal-induct* b **and** b **and** b **and** b **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: $\mathcal{B}'$
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next

case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b v2*)
then show *?case* **using** *wf-intros wb-b-weakening1*
by *meson*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *Wellformed.wfE-fstI wb-b-weakening1* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f ft v*)
then show *?case* **using** *wf-intros* **using** *wb-b-weakening1* **by** *meson*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$*1* Γ Δ *b′ bv1 v1 τ1 f1 x1 b1 c1 s1*)

have Θ ; Φ ; $\mathcal{B}'$; Γ ; Δ $\vdash_{wf}$ *AE-appP f1 b′ v1* : (*b-of τ1*)[*bv1*::=*b′*]$_b$
proof
show Θ $\vdash_{wf}$ Φ **using** *wfE-appPI* **by** *auto*
show Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ Δ **using** *wfE-appPI* **by** *auto*
show Θ; $\mathcal{B}'$ $\vdash_{wf}$ *b′* **using** *wfE-appPI wb-b-weakening1* **by** *auto*
thus *atom bv1* ♯ (Φ, Θ, $\mathcal{B}'$, Γ, Δ, *b′*, *v1*, (*b-of τ1*)[*bv1*::=*b′*]$_b$)
using *wfE-appPI fresh-prodN* **by** *auto*

show *Some* (*AF-fundef f1* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1 s1*))) = *lookup-fun* Φ *f1*
using *wfE-appPI* **by** *auto*
show Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *v1* : *b1*[*bv1*::=*b′*]$_b$ **using** *wfE-appPI wb-b-weakening1* **by** *auto*
qed
then show *?case* **by** *auto*
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u τ*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b′ x s b*)
show *?case* **proof**

show ‹ Θ ; Φ ; $\mathcal{B}'$; Γ ; Δ $\vdash_{wf}$ *e* : *b'* › **using** *wfS-letI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}'$; (*x*, *b'*, *TRUE*) $\#_{\Gamma}$ Γ ; Δ $\vdash_{wf}$ *s* : *b* › **using** *wfS-letI* **by** *auto*
show ‹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ Δ › **using** *wfS-letI* **by** *auto*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}'$, Γ, Δ, *e*, *b*)› **using** *wfS-letI* **by** *auto*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x* *s2* *b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-let2I* **by** *simp*
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1* *b* *s2*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-ifI* **by** *simp*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v* *u* Δ Φ *s* *b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-varI* **by** *simp*
next
case (*wfS-assignI* *u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-assignI* **by** *simp*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1* *s2* *b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-whileI* **by** *simp*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1* *s2* *b*)
then show *?case* **using** *Wellformed.wfS-seqI* **by** *metis*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v* *tid* *dclist* Δ Φ *cs* *b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-matchI* **by** *metis*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s* *b* *tid* *dc*)
then show *?case* **using** *Wellformed.wfS-branchI* **by** *auto*
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid* *dclist'* *cs* *b* *dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid* *dclist'* *cs* *b* *css* *dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfPhi-consI* *f* Θ Φ *ft*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfFTSome* Θ *bv* *ft*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfFTI* Θ *B* *b* *s* *x* *c* τ Φ)

show *?case* **proof**
show Θ; $\mathcal{B}'$ $\vdash_{wf}$ *b* **using** *wfFTI wb-b-weakening1* **by** *auto*

show *supp c* ⊆ {*atom x*} **using** *wfFTI wb-b-weakening1* **by** *auto*
show Θ; $\mathcal{B}'$; (*x*, *b*, *c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfFTI wb-b-weakening1* **by** *auto*
show Θ $\vdash_{wf}$ Φ **using** *wfFTI wb-b-weakening1* **by** *auto*
from ‹ *B* |⊆| $\mathcal{B}'$› **have** *supp B* ⊆ *supp* $\mathcal{B}'$ **proof**(*induct B*)
case *empty*
then show *?case* **by** *auto*
next
case (*insert x B*)
then show *?case*
by (*metis fsubset-funion-eq subset-Un-eq supp-union-fset*)
qed
thus *supp s* ⊆ {*atom x*} ∪ *supp* $\mathcal{B}'$ **using** *wfFTI* **by** *auto*
qed
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}'$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s* : *b* › **using** *wb-b-weakening1 wfS-assertI* **by** *simp*
show ‹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *c* › **using** *wb-b-weakening1 wfS-assertI* **by** *simp*
show ‹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ Δ › **using** *wb-b-weakening1 wfS-assertI* **by** *simp*
have *atom x* ♯ $\mathcal{B}'$ **using** *x-not-in-b-set fresh-def* **by** *metis*
thus ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}'$, Γ, Δ, *c*, *b*, *s*)› **using** *wfS-assertI fresh-prodN* **by** *simp*
qed
qed(*auto*)

lemmas *wb-b-weakening* = *wb-b-weakening1 wb-b-weakening2*

lemma *wfG-b-weakening*:
fixes Γ::Γ
assumes $\mathcal{B}$ |⊆| $\mathcal{B}'$ **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ
shows Θ; $\mathcal{B}'$ $\vdash_{wf}$ Γ
using *wb-b-weakening assms* **by** *auto*

lemma *wfT-b-weakening*:
fixes Γ::Γ **and** Θ::Θ **and** τ::τ
assumes $\mathcal{B}$ |⊆| $\mathcal{B}'$ **and** Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ
shows Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ τ
using *wb-b-weakening assms* **by** *auto*

lemma *wfB-subst-wfB*:
fixes τ::τ **and** *b'*::*b* **and** *b*::*b*
assumes Θ ; {|*bv*|} $\vdash_{wf}$ *b* **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ *b'*
shows Θ; $\mathcal{B}$ $\vdash_{wf}$ $b[bv::=b']_{bb}$
using *assms* **proof**(*nominal-induct b rule*:*b.strong-induct*)
case *B-int*
hence Θ ; {||} $\vdash_{wf}$ *B-int* **using** *wfB-intI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case *B-bool*
hence Θ ; {||} $\vdash_{wf}$ *B-bool* **using** *wfB-boolI wfX-wfY* **by** *fast*

then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case (*B-id x*)
hence Θ; $\mathcal{B}$ $\vdash_{wf}$ (*B-id x*) **using** *wfB-consI wfB-elims wfX-wfY* **by** *metis*
then show *?case* **using** *subst-bb.simps*(*4*) **by** *auto*
next
case (*B-pair x1 x2*)
then show *?case* **using** *subst-bb.simps*
by (*metis wfB-elims*(*1*) *wfB-pairI*)
next
case *B-unit*
hence Θ ; {||} $\vdash_{wf}$ *B-unit* **using** *wfB-unitI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case *B-bitvec*
hence Θ ; {||} $\vdash_{wf}$ *B-bitvec* **using** *wfB-bitvecI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case (*B-var x*)
then show *?case*
proof −
have *False*
using *B-var.prems*(*1*) *wfB.cases* **by** *fastforce*
then show *?thesis* **by** *metis*
qed
next
case (*B-app s b*)
then obtain *bv′ dclist* **where** ∗:*AF-typedef-poly s bv′ dclist* ∈ *set* Θ ∧ Θ ; {|*bv*|} $\vdash_{wf}$ *b* **using** *wfB-elims* **by** *metis*
show *?case* **unfolding** *subst-b-simps* **proof**
show $\vdash_{wf}$ Θ **using** *B-app wfX-wfY* **by** *metis*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b*[*bv*::=*b′*]$_{bb}$ **using** ∗ *B-app forget-subst wfB-supp fresh-def*
by (*metis ex-in-conv subset-empty subst-b-b-def supp-empty-fset*)
show *AF-typedef-poly s bv′ dclist* ∈ *set* Θ **using** ∗ **by** *auto*
qed
qed

lemma *wfT-subst-wfB*:
fixes τ::τ **and** *b′*::*b*
assumes Θ ; {|*bv*|} ; (*x*, *b*, *c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ *b′*
shows Θ; $\mathcal{B}$ $\vdash_{wf}$ (*b-of* τ)[*bv*::=*b′*]$_{bb}$
proof −
obtain *b* **where** Θ ; {|*bv*|} $\vdash_{wf}$ *b* ∧ *b-of* τ = *b* **using** *wfT-elims b-of.simps assms* **by** *metis*
thus *?thesis* **using** *wfB-subst-wfB assms* **by** *auto*
qed

lemma *wfG-cons-unique*:
assumes (*x1*,*b1*,*c1*) ∈ *toSet* (((*x*,*b*,*c*) $\#_\Gamma$Γ)) **and** *wfG* Θ $\mathcal{B}$ (((*x*,*b*,*c*) $\#_\Gamma$Γ)) **and** *x* = *x1*
shows *b1* = *b* ∧ *c1* = *c*
proof −
have *x1* ∉ *fst* ‘ *toSet* Γ
proof −

have *atom x1* $\sharp$ Γ **using** *assms wfG-cons* **by** *metis*
then show *?thesis*
using *fresh-gamma-elem*
by (*metis assms*(*2*) *atom-dom.simps dom.simps rev-image-eqI wfG-cons2 wfG-x-fresh*)
qed
thus *?thesis* **using** *assms* **by** *force*
qed

lemma *wfG-member-unique*:
assumes $(x1,b1,c1) \in toSet\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $wfG\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $x = x1$
shows $b1 = b \wedge c1 = c$
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *wfG-suffix wfG-cons-unique append-g.simps* **by** *metis*
next
case $(GCons\ x'\ b'\ c'\ \Gamma')$
moreover hence $(x1,\ b1,\ c1) \in toSet\ (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)$ **using** *wf-not-in-prefix* **by** *fastforce*
ultimately show *?case* **using** *wfG-cons* **by** *fastforce*
qed

8.13 Function Definitions

lemma *wb-phi-weakening*:
fixes $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(string*\tau)\ list$ **and** $\Delta::\Delta$ **and** $s::s$ **and** $\mathcal{B}::\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list* **and** $\Phi::\Phi$
shows
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma\ ;\ \Delta \vdash_{wf}\ e : b \Longrightarrow \Theta\ \vdash_{wf}\ \Phi' \Longrightarrow set\ \Phi\ \subseteq set\ \Phi' \Longrightarrow \Theta\ ;\ \Phi'\ ;\ \mathcal{B}\ ;\ \ \Gamma\ ;\ \Delta \vdash_{wf}\ e : b$
and
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma\ ;\ \Delta \vdash_{wf}\ s : b \Longrightarrow \Theta\ \vdash_{wf}\ \Phi' \Longrightarrow set\ \Phi\ \subseteq set\ \Phi' \Longrightarrow \Theta\ ;\ \Phi'\ ;\ \mathcal{B}\ ;\ \ \Gamma\ ;\ \Delta \vdash_{wf}\ s : b$
and
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ \ ;\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dc\ ;\ t \vdash_{wf}\ cs : b \Longrightarrow \Theta\ \vdash_{wf}\ \Phi' \Longrightarrow set\ \Phi\ \subseteq set\ \Phi' \Longrightarrow\ \ \Theta\ ;\ \Phi'\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dc\ ;\ t\ \vdash_{wf}\ cs : b$ **and**
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ \ ;\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dclist \vdash_{wf}\ css : b \Longrightarrow \Theta\ \vdash_{wf}\ \Phi' \Longrightarrow set\ \Phi\ \subseteq set\ \Phi' \Longrightarrow\ \ \Theta\ ;\ \Phi'\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dclist \vdash_{wf}\ css : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow True$ **and**
$\Theta;\ \mathcal{B};\ \Gamma\ \ \vdash_{wf}\ \Delta \Longrightarrow True$ **and**
$\Theta\ ;\ \Phi\ \ \vdash_{wf}\ ftq \Longrightarrow \Theta\ \vdash_{wf}\ \Phi' \Longrightarrow set\ \Phi\ \subseteq set\ \Phi' \Longrightarrow \Theta\ ;\ \Phi'\ \vdash_{wf}\ ftq$ **and**
$\Theta\ ;\ \Phi\ ;\ \mathcal{B} \vdash_{wf}\ ft \Longrightarrow\ \ \Theta\ \vdash_{wf}\ \Phi' \Longrightarrow set\ \Phi\ \subseteq set\ \Phi' \Longrightarrow \Theta\ ;\ \Phi'\ ;\ \mathcal{B} \vdash_{wf}\ ft$
proof(*nominal-induct*
b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Φ'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case $(wfE\text{-}valI\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ v\ b)$
then show *?case* **using** *wf-intros* **by** *metis*
next
case $(wfE\text{-}plusI\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ v1\ v2)$
then show *?case* **using** *wf-intros* **by** *metis*
next
case $(wfE\text{-}leqI\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ v1\ v2)$
then show *?case* **using** *wf-intros* **by** *metis*
next

case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
then show *?case* **using** *wf-intros lookup-fun-weakening* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b′ bv v* τ *f x b c s*)
show *?case* **proof**
show ‹ $\Theta \vdash_{wf} \Phi'$ › **using** *wfE-appPI* **by** *auto*
show ‹ $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$ › **using** *wfE-appPI* **by** *auto*
show ‹ $\Theta; \mathcal{B} \vdash_{wf} b'$ › **using** *wfE-appPI* **by** *auto*
show ‹*atom bv* ♯ (Φ′, Θ, $\mathcal{B}$, Γ, Δ, *b′*, *v*, (*b-of* τ)[*bv*::=*b′*]$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ′ *f*›
using *wfE-appPI lookup-fun-weakening* **by** *metis*
show ‹ $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b[bv::=b']_b$ › **using** *wfE-appPI* **by** *auto*
qed
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b′ x s b*)
then show *?case* **using** *Wellformed.wfS-letI* **by** *fastforce*
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1 b′ x s2 b*)
then show *?case* **using** *Wellformed.wfS-let2I* **by** *fastforce*
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
show *?case* **proof**
show ‹ $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$ › **using** *wfS-varI* **by** *simp*
show ‹ $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-}of\ \tau$ › **using** *wfS-varI* **by** *simp*
show ‹*atom u* ♯ (Φ′, Θ, $\mathcal{B}$, Γ, Δ, τ, *v*, *b*)› **using** *wfS-varI* **by** *simp*

show ‹ Θ ; Φ' ; $\mathcal{B}$; Γ ; (u, τ) $\#_\Delta$ $\Delta \vdash_{wf} s : b$ › **using** *wfS-varI* **by** *simp*
qed
next
case (*wfS-assignI u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
then show *?case* **using** *Wellformed.wfS-branchI* **by** *fastforce*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**
show ‹ Θ ; Φ' ; $\mathcal{B}$; $(x, B\text{-}bool, c)$ $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$ › **using** *wfS-assertI* **by** *auto*
next
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf} c$ › **using** *wfS-assertI* **by** *auto*
next
show ‹ Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-assertI* **by** *auto*
have *atom x* $\sharp$ Φ' **using** *wfS-assertI wfPhi-supp fresh-def* **by** *blast*
thus ‹*atom x* $\sharp$ $(\Phi', \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s)$› **using** *fresh-prodN wfS-assertI wfPhi-supp fresh-def* **by** *auto*
qed
next
case (*wfFTI* Θ *B b s x c* τ Φ)
show *?case* **proof**
show ‹ Θ ; B $\vdash_{wf} b$ › **using** *wfFTI* **by** *auto*
next
show ‹*supp c* $\subseteq$ {*atom x*}› **using** *wfFTI* **by** *auto*
next
show ‹ Θ ; B ; (x, b, c) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau$ › **using** *wfFTI* **by** *auto*
next
show ‹ Θ $\vdash_{wf} \Phi'$ › **using** *wfFTI* **by** *auto*
next
show ‹*supp s* $\subseteq$ {*atom x*} $\cup$ *supp B*› **using** *wfFTI* **by** *auto*
qed
qed(*auto*|*metis wf-intros*)+

lemma *wfT-fun-return-t*:
fixes $\tau a'$::τ **and** τ'::τ
assumes Θ; $\mathcal{B}$; (xa, b, ca) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau a'$ **and** (*AF-fun-typ x b c* τ' *s'*) = (*AF-fun-typ xa b ca* $\tau a'$ *sa'*)
shows Θ; $\mathcal{B}$; (x, b, c) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau'$
proof −
obtain *cb*::*x* **where** *xf*: *atom cb* $\sharp$ $(c, \tau', s', sa', \tau a', ca, x, xa)$ **using** *obtain-fresh* **by** *blast*

hence *atom cb* ♯ $(c, \tau', s', sa', \tau a', ca)$ ∧ *atom cb* ♯ $(x, xa, ((c, \tau'), s'), (ca, \tau a'), sa')$ **using** *fresh-prod6 fresh-prod4 fresh-prod8* **by** *auto*
hence $*$:$c[x::=V\text{-}var\ cb]_{cv} = ca[xa::=V\text{-}var\ cb]_{cv} \wedge \tau'[x::=V\text{-}var\ cb]_{\tau v} = \tau a'[xa::=V\text{-}var\ cb]_{\tau v}$ **using** *assms τ.eq-iff Abs1-eq-iff-all* **by** *auto*

have $**$: Θ; $\mathcal{B}$; $(xa \leftrightarrow cb) \cdot ((xa, b, ca)\ \#_\Gamma\ GNil) \vdash_{wf} (xa \leftrightarrow cb) \cdot \tau a'$ **using** *assms True-eqvt beta-flip-eq theta-flip-eq wfG-wf*
by (*metis GCons-eqvt GNil-eqvt wfT.eqvt wfT-wf*)

have Θ; $\mathcal{B}$; $(x \leftrightarrow cb) \cdot ((x, b, c)\ \#_\Gamma\ GNil) \vdash_{wf} (x \leftrightarrow cb) \cdot \tau'$ **proof** −
have $(xa \leftrightarrow cb) \cdot xa = (x \leftrightarrow cb) \cdot x$ **using** *xf* **by** *auto*
hence $(x \leftrightarrow cb) \cdot ((x, b, c)\ \#_\Gamma\ GNil) = (xa \leftrightarrow cb) \cdot ((xa, b, ca)\ \#_\Gamma\ GNil)$ **using** $*$ $**$ *xf G-cons-flip fresh-GNil* **by** *simp*
thus *?thesis* **using** $**$ $*$ *xf* **by** *simp*
qed
thus *?thesis* **using** *beta-flip-eq theta-flip-eq wfT-wf wfG-wf* $*$ $**$ *True-eqvt wfT.eqvt permute-flip-cancel* **by** *metis*
qed

lemma *wfFT-wf-aux*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Δ::Δ
assumes Θ ; Φ ; *B* $\vdash_{wf}$ (*AF-fun-typ x b c τ s*)
shows Θ ; *B* ; $(x,b,c)\ \#_\Gamma\ GNil \vdash_{wf} \tau$ ∧ Θ $\vdash_{wf}$ Φ ∧ *supp s* ⊆ { *atom x* } ∪ *supp B*
proof −
obtain *xa* **and** *ca* **and** *sa* **and** τ′ **where** $*$:Θ ; *B* $\vdash_{wf}$ *b* ∧ (Θ $\vdash_{wf}$ Φ) ∧
supp sa ⊆ {*atom xa*} ∪ *supp B* ∧ (Θ ; *B* ; $(xa, b, ca)\ \#_\Gamma\ GNil \vdash_{wf} \tau'$) ∧
AF-fun-typ x b c τ s = *AF-fun-typ xa b ca τ′ sa*
using *wfFT.simps*[*of* Θ Φ *B AF-fun-typ x b c τ s*] *assms* **by** *auto*

moreover hence $**$: (*AF-fun-typ x b c τ s*) = (*AF-fun-typ xa b ca τ′ sa*) **by** *simp*
ultimately have Θ ; *B* ; $(x,b,c)\ \#_\Gamma GNil \vdash_{wf} \tau$ **using** *wfT-fun-return-t* **by** *metis*
moreover have (Θ $\vdash_{wf}$ Φ) **using** $*$ **by** *auto*
moreover have *supp s* ⊆ { *atom x* } ∪ *supp B* **proof** −
have [[*atom x*]]*lst.s* = [[*atom xa*]]*lst.sa* **using** $**$ *fun-typ.eq-iff lst-fst lst-snd* **by** *metis*
thus *?thesis* **using** *lst-supp-subset* $*$ **by** *metis*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *wfFT-simple-wf*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Δ::Δ
assumes Θ ; Φ $\vdash_{wf}$ (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*))
shows Θ ; {||} ; $(x,b,c)\ \#_\Gamma GNil \vdash_{wf} \tau$ ∧ Θ $\vdash_{wf}$ Φ ∧ *supp s* ⊆ { *atom x* }
proof −
have $*$:Θ ; Φ ; {||} $\vdash_{wf}$ (*AF-fun-typ x b c τ s*) **using** *wfFTQ-elims assms* **by** *auto*
thus *?thesis* **using** *wfFT-wf-aux* **by** *force*
qed

lemma *wfFT-poly-wf*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ftq* :: *fun-typ-q* **and** *s*::*s* **and** Δ::Δ
assumes Θ ; Φ $\vdash_{wf}$ (*AF-fun-typ-some bv* (*AF-fun-typ x b c τ s*))
shows Θ ; {|*bv*|} ; $(x,b,c)\ \#_\Gamma GNil \vdash_{wf} \tau$ ∧ Θ $\vdash_{wf}$ Φ ∧ Θ ; Φ ; {|*bv*|} $\vdash_{wf}$ (*AF-fun-typ x b c τ s*)

proof −
obtain *bv1 ft1* **where** ∗:Θ ; Φ ; {|*bv1*|} $\vdash_{wf}$ *ft1* ∧ [[*atom bv1*]]*lst*. *ft1* = [[*atom bv*]]*lst*. *AF-fun-typ x b c τ s*
using *wfFTQ-elims*(*3*)[*OF assms*] **by** *metis*

show *?thesis* **proof**(*cases bv1* = *bv*)
case *True*
then show *?thesis* **using** ∗ *fun-typ-q.eq-iff Abs1-eq-iff* **by** (*metis* (*no-types*, *opaque-lifting*) *wfFT-wf-aux*)
next
case *False*
obtain *x1 b1 c1 t1 s1* **where** ∗∗: *ft1* = *AF-fun-typ x1 b1 c1 t1 s1* **using** *fun-typ.eq-iff*
by (*meson fun-typ.exhaust*)

hence *eqv*: (*bv* ↔ *bv1*) • *AF-fun-typ x1 b1 c1 t1 s1* = *AF-fun-typ x b c τ s* ∧ *atom bv1* ♯ *AF-fun-typ x b c τ s* **using**
Abs1-eq-iff(*3*) ∗ *False* **by** *metis*

have (*bv* ↔ *bv1*) • Θ ; (*bv* ↔ *bv1*) • Φ ; (*bv* ↔ *bv1*) • {|*bv1*|} $\vdash_{wf}$ (*bv* ↔ *bv1*) • *ft1* **using** *wfFT.eqvt* ∗ **by** *metis*
moreover have (*bv* ↔ *bv1*) • Φ = Φ **using** *phi-flip-eq wfX-wfY* ∗ **by** *metis*
moreover have (*bv* ↔ *bv1*) • Θ =Θ **using** *wfX-wfY* ∗ *theta-flip-eq2* **by** *metis*
moreover have (*bv* ↔ *bv1*) • *ft1* = *AF-fun-typ x b c τ s* **using** *eqv* ∗∗ **by** *metis*
ultimately have Θ ; Φ ; {|*bv*|} $\vdash_{wf}$ *AF-fun-typ x b c τ s* **by** *auto*
thus *?thesis* **using** *wfFT-wf-aux* **by** *auto*
qed
qed

lemma *wfFT-poly-wfT*:
fixes *τ*::*τ* **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes Θ ; Φ $\vdash_{wf}$ (*AF-fun-typ-some bv* (*AF-fun-typ x b c τ s*))
shows Θ ; {| *bv* |} ; (*x*,*b*,*c*) $\#_{\Gamma}$ *GNil* $\vdash_{wf}$ *τ*
using *wfFT-poly-wf assms* **by** *simp*

lemma *b-of-supp*:
supp (*b-of t*) ⊆ *supp t*
proof(*nominal-induct t rule*:*τ*.*strong-induct*)
case (*T-refined-type x b c*)
then show *?case* **by** *auto*
qed

lemma *wfPhi-f-simple-wf*:
fixes *τ*::*τ* **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Φ′::Φ
assumes *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)) ∈ *set* Φ **and** Θ $\vdash_{wf}$ Φ **and** *set* Φ ⊆ *set* Φ′ **and** Θ $\vdash_{wf}$ Φ′
shows Θ ; {||} ; (*x*,*b*,*c*) $\#_{\Gamma}$ *GNil* $\vdash_{wf}$ *τ* ∧ Θ $\vdash_{wf}$ Φ ∧ *supp s* ⊆ { *atom x* }
using *assms* **proof**(*induct* Φ *rule*: Φ-*induct*)
case *PNil*
then show *?case* **by** *auto*
next
case (*PConsSome f1 bv x1 b1 c1 τ1 s′* Φ′′)
hence *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)) ∈ *set* Φ′′ **by** *auto*

moreover have $\Theta \vdash_{wf} \Phi'' \wedge$ *set* $\Phi'' \subseteq$ *set* Φ' **using** *wfPhi-elims(3) PConsSome* **by** *auto*
ultimately show *?case* **using** *PConsSome wfPhi-elims wfFT-simple-wf* **by** *auto*
next
case (*PConsNone* f' x' b' c' τ' s' Φ'')
show *?case* **proof**(*cases* $f=f'$)
case *True*
have *AF-fun-typ-none* (*AF-fun-typ* x' b' c' τ' s') = *AF-fun-typ-none* (*AF-fun-typ* x b c τ s)
by (*metis PConsNone.prems(1) PConsNone.prems(2) True fun-def.eq-iff image-eqI name-of-fun.simps set-ConsD wfPhi-elims(2)*)
hence $*$:Θ ; $\Phi'' \vdash_{wf}$ *AF-fun-typ-none* (*AF-fun-typ* x b c τ s) **using** *wfPhi-elims(2)[OF PConsNone(3)]* **by** *metis*
hence Θ ; Φ'' ; $\{||\} \vdash_{wf}$ (*AF-fun-typ* x b c τ s) **using** *wfFTQ-elims(1)* **by** *metis*
thus *?thesis* **using** *wfFT-simple-wf[OF $*$] wb-phi-weakening PConsNone* **by** *force*
next
case *False*
hence *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x b c τ s)) $\in$ *set* Φ'' **using** *PConsNone* **by** *simp*
moreover have $\Theta \vdash_{wf} \Phi'' \wedge$ *set* $\Phi'' \subseteq$ *set* Φ' **using** *wfPhi-elims(3) PConsNone* **by** *auto*
ultimately show *?thesis* **using** *PConsNone wfPhi-elims wfFT-simple-wf* **by** *auto*
qed
qed

lemma *wfPhi-f-simple-wfT*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x b c τ s))) = *lookup-fun* Φ f **and** $\Theta \vdash_{wf} \Phi$
shows Θ ; $\{||\}$; (x,b,c) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau$
using *wfPhi-f-simple-wf assms* **using** *lookup-fun-member* **by** *blast*

lemma *wfPhi-f-simple-supp-b*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x b c τ s))) = *lookup-fun* Φ f **and** $\Theta \vdash_{wf} \Phi$
shows *supp* $b = \{\}$
proof $-$
have Θ ; $\{||\}$; (x,b,c) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau$ **using** *wfPhi-f-simple-wfT assms* **by** *auto*
thus *?thesis* **using** *wfT-wf wfG-cons wfB-supp* **by** *fastforce*
qed

lemma *wfPhi-f-simple-supp-t*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x b c τ s))) = *lookup-fun* Φ f **and** $\Theta \vdash_{wf} \Phi$
shows *supp* $\tau \subseteq \{$ *atom* x $\}$
using *wfPhi-f-simple-wfT wfT-supp assms* **by** *fastforce*

lemma *wfPhi-f-simple-supp-c*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x b c τ s))) = *lookup-fun* Φ f **and** $\Theta \vdash_{wf} \Phi$
shows *supp* $c \subseteq \{$ *atom* x $\}$
proof $-$
have Θ ; $\{||\}$; (x,b,c) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau$ **using** *wfPhi-f-simple-wfT assms* **by** *auto*

thus *?thesis* **using** *wfG-wfC wfC-supp wfT-wf* **by** *fastforce*
qed

lemma *wfPhi-f-simple-supp-s*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** $\Theta \vdash_{wf} \Phi$
shows *supp s* $\subseteq$ {*atom x*}
proof −
have *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **using** *lookup-fun-member assms* **by** *auto*
hence *supp s* $\subseteq$ { *atom x* } **using** *wfPhi-f-simple-wf assms* **by** *blast*
thus *?thesis* **using** *wf-supp*(*3*) *atom-dom.simps toSet.simps x-not-in-u-set x-not-in-b-set setD.simps*
using *wf-supp2*(*2*) **by** *fastforce*
qed

lemma *wfPhi-f-poly-wf*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Φ'::Φ
assumes *AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **and** $\Theta \vdash_{wf} \Phi$ **and** *set* $\Phi \subseteq$ *set* Φ' **and** $\Theta \vdash_{wf} \Phi'$
shows Θ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi' \wedge \Theta$; Φ' ; {|*bv*|} $\vdash_{wf}$ (*AF-fun-typ x b c* τ *s*)
using *assms* **proof**(*induct* Φ *rule*: Φ-*induct*)
case *PNil*
then show *?case* **by** *auto*
next
case (*PConsNone f x b c* τ *s'* Φ'')
moreover have $\Theta \vdash_{wf} \Phi'' \wedge$ *set* $\Phi'' \subseteq$ *set* Φ' **using** *wfPhi-elims*(*3*) *PConsNone* **by** *auto*
ultimately show *?case* **using** *PConsNone wfPhi-elims wfFT-poly-wf* **by** *auto*
next
case (*PConsSome f1 bv1 x1 b1 c1* τ*1 s1* Φ'')
show *?case* **proof**(*cases f*=*f1*)
case *True*
have *AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s1*) = *AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)
by (*metis PConsSome.prems*(*1*) *PConsSome.prems*(*2*) *True fun-def.eq-iff list.set-intros*(*1*) *option.inject wfPhi-lookup-fun-unique*)
hence $*$:Θ ; Φ'' $\vdash_{wf}$ *AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*) **using** *wfPhi-elims PConsSome* **by** *metis*
thus *?thesis* **using** *wfFT-poly-wf* $*$ *wb-phi-weakening PConsSome*
by (*meson set-subset-Cons*)
next
case *False*
hence *AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ'' **using** *PConsSome*
by (*meson fun-def.eq-iff set-ConsD*)
moreover have $\Theta \vdash_{wf} \Phi'' \wedge$ *set* $\Phi'' \subseteq$ *set* Φ' **using** *wfPhi-elims*(*3*) *PConsSome*
by (*meson dual-order.trans set-subset-Cons*)
ultimately show *?thesis* **using** *PConsSome wfPhi-elims wfFT-poly-wf*
by *blast*
qed
qed

lemma *wfPhi-f-poly-wfT*:

fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows Θ ; {| *bv* |} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ
using *assms* **proof**(*induct* Φ *rule*: Φ-*induct*)
case *PNil*
then show *?case* **by** *auto*
next
case (*PConsSome f1 bv1 x1 b1 c1* τ*1 s'* Φ')
then show *?case* **proof**(*cases f1=f*)
case *True*
hence *lookup-fun* (*AF-fundef f1* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s'*)) # Φ') *f* = *Some* (*AF-fundef f1* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s'*))) **using**
lookup-fun.simps **using** *PConsSome.prems* **by** *simp*
then show *?thesis* **using** *PConsSome.prems wfPhi-elims wfFT-poly-wfT*
by (*metis option.inject*)
next
case *False*
then show *?thesis* **using** *PConsSome* **using** *lookup-fun.simps*
using *wfPhi-elims*(*3*) **by** *auto*
qed
next
case (*PConsNone f' x' b' c'* τ' *s'* Φ')
then show *?case* **proof**(*cases f'=f*)
case *True*
then have *:Θ ; Φ' $\vdash_{wf}$ *AF-fun-typ-none* (*AF-fun-typ x' b' c'* τ' *s'*) **using** *lookup-fun.simps PConsNone wfPhi-elims* **by** *metis*
thus *?thesis* **using** *PConsNone wfFT-poly-wfT wfPhi-elims lookup-fun.simps*
by (*metis fun-def.eq-iff fun-typ-q.distinct*(*1*) *option.inject*)
next
case *False*
thus *?thesis* **using** *PConsNone wfPhi-elims*
by (*metis False lookup-fun.simps*(*2*))
qed
qed

lemma *wfPhi-f-poly-supp-b*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp b* $\subseteq$ *supp bv*
proof −
have Θ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-poly-wfT assms* **by** *auto*
thus *?thesis* **using** *wfT-wf wfG-cons wfB-supp* **by** *fastforce*
qed

lemma *wfPhi-f-poly-supp-t*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp* τ $\subseteq$ { *atom x* , *atom bv* }
using *wfPhi-f-poly-wfT*[*OF assms*, *THEN wfT-supp*] *atom-dom.simps supp-at-base* **by** *auto*

lemma *wfPhi-f-poly-supp-b-of-t*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp* (*b-of* τ) $\subseteq$ { *atom bv* }
proof −
have *atom x* $\notin$ *supp* (*b-of* τ) **using** *x-fresh-b* **by** *auto*
moreover have *supp* (*b-of* τ) $\subseteq$ { *atom x* , *atom bv* } **using** *wfPhi-f-poly-supp-t*
using *supp-at-base b-of.simps wfPhi-f-poly-supp-t* τ*.supp b-of-supp assms* **by** *fast*
ultimately show *?thesis* **by** *blast*
qed

lemma *wfPhi-f-poly-supp-c*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp c* $\subseteq$ { *atom x*, *atom bv* }
proof −
have Θ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-poly-wfT assms* **by** *auto*
thus *?thesis* **using** *wfG-wfC wfC-supp wfT-wf*
using *supp-at-base* **by** *fastforce*
qed

lemma *wfPhi-f-poly-supp-s*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp s* $\subseteq$ {*atom x*, *atom bv*}
proof −

have *AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **using** *lookup-fun-member assms* **by** *auto*
hence $*$:Θ ; Φ ; {|*bv*|} $\vdash_{wf}$ (*AF-fun-typ x b c* τ *s*) **using** *assms wfPhi-f-poly-wf* **by** *simp*

thus *?thesis* **using** *wfFT-wf-aux*[*OF* $*$] **using** *supp-at-base* **by** *auto*
qed

lemmas *wfPhi-f-supp* = *wfPhi-f-poly-supp-b wfPhi-f-simple-supp-b wfPhi-f-poly-supp-c wfPhi-f-simple-supp-t wfPhi-f-poly-supp-t wfPhi-f-simple-supp-t wfPhi-f-poly-wfT wfPhi-f-simple-wfT wfPhi-f-poly-supp-s wfPhi-f-simple-supp-s*

lemma *fun-typ-eq-ret-unique*:
assumes (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*) = (*AF-fun-typ x2 b2 c2* $\tau 2'$ *s2'*)
shows $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$
proof −
have [[*atom x1*]]*lst*. $\tau 1'$ = [[*atom x2*]]*lst*. $\tau 2'$ **using** *assms lst-fst fun-typ.eq-iff lst-snd* **by** *metis*
thus *?thesis* **using** *subst-v-flip-eq-two*[*of x1* $\tau 1'$ *x2* $\tau 2'$ *v*] *subst-v-*τ*-def* **by** *metis*
qed

lemma *fun-typ-eq-body-unique*:

fixes *v::v* **and** *x1::x* **and** *x2::x* **and** *s1'::s* **and** *s2'::s*
assumes (*AF-fun-typ x1 b1 c1 τ1' s1'*) = (*AF-fun-typ x2 b2 c2 τ2' s2'*)
shows $s1'[x1::=v]_{sv} = s2'[x2::=v]_{sv}$
proof −
have [[*atom x1*]]*lst. s1'* = [[*atom x2*]]*lst. s2'* **using** *assms lst-fst fun-typ.eq-iff lst-snd* **by** *metis*
thus *?thesis* **using** *subst-v-flip-eq-two*[*of x1 s1' x2 s2' v*] *subst-v-s-def* **by** *metis*
qed

lemma *fun-ret-unique*:
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) = *lookup-fun* Φ *f*
shows $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$
proof −
have ∗: (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) **using** *option.inject assms* **by** *metis*
thus *?thesis* **using** *fun-typ-eq-ret-unique fun-def.eq-iff fun-typ-q.eq-iff* **by** *metis*
qed

lemma *fun-poly-arg-unique*:
fixes *bv1::bv* **and** *bv2::bv* **and** *b::b* **and** *τ1::τ* **and** *τ2::τ*
assumes [[*atom bv1*]]*lst.* (*AF-fun-typ x1 b1 c1 τ1 s1*) = [[*atom bv2*]]*lst.* (*AF-fun-typ x2 b2 c2 τ2 s2*) (**is** [[*atom ?x*]]*lst. ?a* = [[*atom ?y*]]*lst. ?b*)
shows $\{\!| \; x1 : b1[bv1::=b]_{bb} \mid c1[bv1::=b]_{cb} \; |\!\} = \{\!| \; x2 : b2[bv2::=b]_{bb} \mid c2[bv2::=b]_{cb} \; |\!\}$
proof −
obtain *c::bv* **where** ∗:*atom c* ♯ (*b,b1,b2,c1,c2*) ∧ *atom c* ♯ (*bv1, bv2, AF-fun-typ x1 b1 c1 τ1 s1, AF-fun-typ x2 b2 c2 τ2 s2*) **using** *obtain-fresh fresh-Pair* **by** *metis*
hence (*bv1* ↔ *c*) · *AF-fun-typ x1 b1 c1 τ1 s1* = (*bv2* ↔ *c*) · *AF-fun-typ x2 b2 c2 τ2 s2* **using** *Abs1-eq-iff-all*(*3*)[*of ?x ?a ?y ?b*] *assms* **by** *metis*
hence *AF-fun-typ x1* ((*bv1* ↔ *c*) · *b1*) ((*bv1* ↔ *c*) · *c1*) ((*bv1* ↔ *c*) · *τ1*) ((*bv1* ↔ *c*) · *s1*) = *AF-fun-typ x2* ((*bv2* ↔ *c*) · *b2*) ((*bv2* ↔ *c*) · *c2*) ((*bv2* ↔ *c*) · *τ2*) ((*bv2* ↔ *c*) · *s2*)
using *fun-typ-flip* **by** *metis*
hence ∗∗:⦃ *x1* :((*bv1* ↔ *c*) · *b1*) | ((*bv1* ↔ *c*) · *c1*) ⦄ = ⦃ *x2* : ((*bv2* ↔ *c*) · *b2*) | ((*bv2* ↔ *c*) · *c2*) ⦄ (**is** ⦃ *x1* : *?b1* | *?c1* ⦄ = ⦃ *x2* : *?b2* | *?c2* ⦄) **using** *fun-arg-unique-aux* **by** *metis*
hence ⦃ *x1* :((*bv1* ↔ *c*) · *b1*) | ((*bv1* ↔ *c*) · *c1*) ⦄$[c::=b]_{\tau b}$ = ⦃ *x2* : ((*bv2* ↔ *c*) · *b2*) | ((*bv2* ↔ *c*) · *c2*) ⦄$[c::=b]_{\tau b}$ **by** *metis*
hence ⦃ *x1* :((*bv1* ↔ *c*) · *b1*)$[c::=b]_{bb}$ | ((*bv1* ↔ *c*) · *c1*)$[c::=b]_{cb}$ ⦄ = ⦃ *x2* : ((*bv2* ↔ *c*) · *b2*)$[c::=b]_{bb}$ | ((*bv2* ↔ *c*) · *c2*)$[c::=b]_{cb}$ ⦄ **using** *subst-tb.simps* **by** *metis*
thus *?thesis* **using** ∗ *flip-subst-subst subst-b-c-def subst-b-b-def fresh-prodN flip-commute* **by** *metis*
qed

lemma *fun-poly-ret-unique*:
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) = *lookup-fun* Φ *f*
shows $\tau 1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau 2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$
proof −
have ∗: (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) **using** *option.inject assms* **by** *metis*
hence *AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*) = *AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*)
(**is** *AF-fun-typ-some bv1 ?ft1* = *AF-fun-typ-some bv2 ?ft2*) **using** *fun-def.eq-iff* **by** *metis*
hence ∗∗:[[*atom bv1*]]*lst. ?ft1* = [[*atom bv2*]]*lst. ?ft2* **using** *fun-typ-q.eq-iff*(*1*) **by** *metis*

hence ∗:*subst-ft-b ?ft1 bv1 b* = *subst-ft-b ?ft2 bv2 b* **using** *subst-b-flip-eq-two subst-b-fun-typ-def* **by** *metis*
have $[[atom\ x1]]lst.\ \tau 1'[bv1::=b]_{\tau b} = [[atom\ x2]]lst.\ \tau 2'[bv2::=b]_{\tau b}$
apply(*rule lst-snd*[*of - c1*$[bv1::=b]_{cb}$ *- - c2*$[bv2::=b]_{cb}$])
apply(*rule lst-fst*[*of - - s1'*$[bv1::=b]_{sb}$ *- - s2'*$[bv2::=b]_{sb}$])
using ∗ *subst-ft-b.simps fun-typ.eq-iff* **by** *metis*
thus *?thesis* **using** *subst-v-flip-eq-two subst-v-τ-def* **by** *metis*
qed

lemma *fun-poly-body-unique*:
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) = *lookup-fun* Φ *f*
shows $s1'[bv1::=b]_{sb}[x1::=v]_{sv} = s2'[bv2::=b]_{sb}[x2::=v]_{sv}$
proof −
have ∗: (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*)))
using *option.inject assms* **by** *metis*
hence *AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*) = *AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*)
(**is** *AF-fun-typ-some bv1 ?ft1* = *AF-fun-typ-some bv2 ?ft2*) **using** *fun-def.eq-iff* **by** *metis*
hence ∗∗:[[*atom bv1*]]*lst. ?ft1* = [[*atom bv2*]]*lst. ?ft2* **using** *fun-typ-q.eq-iff*(*1*) **by** *metis*

hence ∗:*subst-ft-b ?ft1 bv1 b* = *subst-ft-b ?ft2 bv2 b* **using** *subst-b-flip-eq-two subst-b-fun-typ-def* **by** *metis*
have $[[atom\ x1]]lst.\ s1'[bv1::=b]_{sb} = [[atom\ x2]]lst.\ s2'[bv2::=b]_{sb}$
using *lst-snd lst-fst subst-ft-b.simps fun-typ.eq-iff*
by (*metis local.*∗)

thus *?thesis* **using** *subst-v-flip-eq-two subst-v-s-def* **by** *metis*
qed

lemma *funtyp-eq-iff-equalities*:
fixes *s'*::*s* **and** *s*::*s*
assumes [[*atom x'*]]*lst.* ((*c'*, *τ'*), *s'*) = [[*atom x*]]*lst.* ((*c*, *τ*), *s*)
shows $\{\!\!\{ x' : b \mid c' \}\!\!\} = \{\!\!\{ x : b \mid c \}\!\!\} \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge \tau'[x'::=v]_{\tau v} = \tau[x::=v]_{\tau v}$
proof −
have [[*atom x'*]]*lst. s'* = [[*atom x*]]*lst. s* **and** [[*atom x'*]]*lst. τ'* = [[*atom x*]]*lst. τ* **and**
[[*atom x'*]]*lst. c'* = [[*atom x*]]*lst. c* **using** *lst-snd lst-fst assms* **by** *metis+*
thus *?thesis* **using** *subst-v-flip-eq-two τ.eq-iff*
by (*metis assms fun-typ.eq-iff fun-typ-eq-body-unique fun-typ-eq-ret-unique*)
qed

8.14 Weakening

lemma *wfX-wfB1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** *τ*::*τ* **and** *ts*::(*string*∗*τ*) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** ℬ::ℬ **and** Φ::Φ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows *wfV-wfB*: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} b$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow$ *True* **and**
$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow$ *True* **and**
wfT-wfB: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow \Theta; \mathcal{B} \vdash_{wf}$ *b-of τ* **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow$ *True* **and**
$\vdash_{wf} \Theta \Longrightarrow$ *True* **and**
$\Theta; \mathcal{B} \vdash_{wf} b \Longrightarrow$ *True* **and**
wfCE-wfB: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} b$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
proof(*induct rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c x*)
hence (*x*,*b*,*c*) ∈ *toSet* Γ **using** *lookup-iff wfV-wf* **using** *lookup-in-g* **by** *presburger*
hence *b* ∈ *fst‘snd‘toSet* Γ **by** *force*
hence *wfB* Θ $\mathcal{B}$ *b* **using** *wfG-wfB wfV-varI* **by** *metis*
then show *?case* **using** *wfV-elims wfG-wf wf-intros* **by** *metis*
next
case (*wfV-litI* Θ Γ *l*)
moreover have *wfTh* Θ **using** *wfV-wf wfG-wf wfV-litI* **by** *metis*
ultimately show *?case* **using** *wfV-wf wfG-wf wf-intros base-for-lit.simps l.exhaust* **by** *metis*
next
case (*wfV-pairI* Θ Γ *v1 b1 v2 b2*)
then show *?case* **using** *wfG-wf wf-intros* **by** *metis*
next
case (*wfV-consI s dclist* Θ *dc x b c B* Γ *v*)
then show *?case*
using *wfV-wf wfG-wf wfB-consI* **by** *metis*
next
case (*wfV-conspI s bv dclist* Θ *dc x b′ c* $\mathcal{B}$ *b* Γ *v*)
then show *?case*
using *wfV-wf wfG-wf* **using** *wfB-appI* **by** *metis*
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
then show *?case* **using** *wfB-elims* **by** *auto*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wfB-elims* **by** *auto*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wfV-wf wfG-wf wf-intros wfX-wfY* **by** *metis*
next
case (*wfCE-eqI* Θ $\mathcal{B}$ Γ *v1 b v2*)
then show *?case* **using** *wfV-wf wfG-wf wf-intros wfX-wfY* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wfB-elims* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wfB-elims* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wfB-elims* **by** *auto*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *wfV-wf wfG-wf wf-intros wfX-wfY* **by** *metis*
qed(*auto* | *metis wfV-wf wfG-wf wf-intros*)+

lemma *wfX-wfB2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** $\mathcal{B}$::$\mathcal{B}$ **and** Φ::Φ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows
wfE-wfB: $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} b$ **and**
wfS-wfB: $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} b$ **and**
wfCS-wfB: $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta$; *tid* ; *dc* ; $t \vdash_{wf} cs : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} b$ **and**
wfCSS-wfB: $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta$; *tid* ; *dclist* $\vdash_{wf} css : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} b$ **and**
$\Theta \vdash_{wf} \Phi \Longrightarrow$ *True* **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \Longrightarrow$ *True* **and**
$\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow$ *True* **and**
$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' \vdash_{wf} ft$
proof(*induct rule*:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b v2*)
then show *?case* **using** *wfB-boolI wfX-wfY* **by** *metis*
next
case (*wfE-fstI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-sndI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wfB-elims wfX-wfB1*
using *wfB-pairI* **by** *auto*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wfB-elims wfX-wfB1*
using *wfB-intI wfX-wfY1*(*1*) **by** *auto*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
hence $\Theta; \mathcal{B};(x,b,c) \#_{\Gamma}$ *GNil* $\vdash_{wf} \tau$ **using** *wfPhi-f-simple-wfT wfT-b-weakening* **by** *fast*
then show *?case* **using** *b-of.simps* **using** *wfT-b-weakening*
by (*metis b-of.cases bot.extremum wfT-elims*(*2*))
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ b' *bv v* τ *f x b c s*)
hence Θ ; {| *bv* |} ;$(x,b,c) \#_{\Gamma}$ *GNil* $\vdash_{wf} \tau$ **using** *wfPhi-f-poly-wfT wfX-wfY* **by** *blast*
then show *?case* **using** *wfE-appPI b-of.simps* **using** *wfT-b-weakening wfT-elims wfT-subst-wfB subst-b-b-def* **by** *metis*
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
hence $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$ **using** *wfD-wfT* **by** *fast*

then show *?case* **using** *wfT-elims b-of.simps* **by** *metis*
next
case (*wfFTNone* Θ *ft*)
then show *?case* **by** *auto*
next
case (*wfFTSome* Θ *bv ft*)
then show *?case* **by** *auto*
next
case (*wfS-valI* Θ Φ *B* Γ *v b* Δ)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-letI* Θ Φ *B* Γ Δ *e b' x s b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-let2I* Θ Φ *B* Γ Δ *s1* τ *x s2 b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-ifI* Θ *B* Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-varI* Θ *B* Γ τ *v u* Φ Δ *b s*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-assignI u* τ Δ Θ *B* Γ Φ *v*)
then show *?case* **using** *wfX-wfB1*
using *wfB-unitI wfX-wfY2*(*5*) **by** *auto*
next
case (*wfS-whileI* Θ Φ *B* Γ Δ *s1 s2 b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-seqI* Θ Φ *B* Γ Δ *s1 s2 b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-matchI* Θ *B* Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-branchI* Θ Φ *B x* τ Γ Δ *s b tid dc*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-finalI* Θ Φ *B* Γ Δ *tid dc t cs b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-cons* Θ Φ *B* Γ Δ *tid dc t cs b dclist css*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfD-emptyI* Θ *B* Γ)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfD-cons* Θ *B* Γ Δ τ *u*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfPhi-emptyI* Θ)

then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfPhi-consI f* Θ Φ *ft*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
then show *?case* **using** *wfX-wfB1*
by (*meson Wellformed.wfFTI wb-b-weakening2*(*8*))
qed(*metis wfV-wf wfG-wf wf-intros wfX-wfB1*)

lemmas *wfX-wfB* = *wfX-wfB1 wfX-wfB2*

lemma *wf-weakening1*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*

shows *wfV-weakening*: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ *v* : *b* **and**
wfC-weakening: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ *c* **and**
Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ⟹ *True* **and**
wfT-weakening: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ τ **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* ⟹ *True* **and**
$\vdash_{wf}$ *P* ⟹ *True* **and**
wfB-weakening: *wfB* Θ $\mathcal{B}$ *b* ⟹ $\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ *wfB* Θ $\mathcal{B}$ *b* **and**
wfCE-weakening: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ *ce* : *b* **and**
Θ $\vdash_{wf}$ *td* ⟹ *True*
proof(*nominal-induct*
b **and** *c* **and** Γ **and** τ **and** *ts* **and** *P* **and** *b* **and** *b* **and** *td*
avoiding: Γ′
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c x*)
hence *Some* (*b*, *c*) = *lookup* Γ′ *x* **using** *lookup-weakening* **by** *metis*
then show *?case* **using** *Wellformed.wfV-varI wfV-varI* **by** *metis*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
show *?case* **proof**
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′)› **using** *wfTI* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfTI* **by** *auto*
have ∗:*toSet* ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ) ⊆ *toSet* ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′) **using** *toSet.simps wfTI* **by** *auto*
thus ‹ Θ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′ $\vdash_{wf}$ *c* › **using** *wfTI*(*8*)[*OF* - ∗] *wfTI wfX-wfY*
by (*simp add*: *wfG-cons-TRUE*)
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x b′ c* $\mathcal{B}$ *b* Γ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, ⦃ *x* : *b′* | *c* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfV-conspI* **by** *auto*

show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ′, *b*, *v*)› **using** *wfV-conspI* **by** *simp*
show ‹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ *v* : *b*′[*bv*::=*b*]$_{bb}$ › **using** *wfV-conspI* **by** *auto*
qed
qed(*metis wf-intros*)+

lemma *wf-weakening2*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows
wfE-weakening: Θ; Φ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; Φ; $\mathcal{B}$; Γ′ ; Δ $\vdash_{wf}$ *e* : *b* **and**
wfS-weakening: Θ; Φ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *s* : *b* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; Φ; $\mathcal{B}$; Γ′ ; Δ $\vdash_{wf}$ *s* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; Φ; $\mathcal{B}$; Γ′ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; Φ; $\mathcal{B}$; Γ′ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* **and**
Θ $\vdash_{wf}$ (Φ::Φ) ⟹ *True* **and**
wfD-weakning: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ Δ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* ⟹ *True* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* ⟹ *True*
proof(*nominal-induct*
b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Γ′
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b*′ *bv v* τ *f x b c s*)
show *?case* **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ Δ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b*′ › **using** *wfE-appPI* **by** *auto*
show ‹*atom bv* ♯ (Φ, Θ, $\mathcal{B}$, Γ′, Δ, *b*′, *v*, (*b-of* τ)[*bv*::=*b*′]$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ *v* : *b*[*bv*::=*b*′]$_b$ › **using** *wfE-appPI wf-weakening1* **by** *auto*
qed
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b*′ *x s b*)
show *?case* **proof**(*rule*)
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ $\vdash_{wf}$ *e* : *b*′ › **using** *wfS-letI* **by** *auto*
have *toSet* ((*x*, *b*′, *TRUE*) $\#_\Gamma$ Γ) ⊆ *toSet* ((*x*, *b*′, *TRUE*) $\#_\Gamma$ Γ′) **using** *wfS-letI* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b*′, *TRUE*) $\#_\Gamma$ Γ′ ; Δ $\vdash_{wf}$ *s* : *b* › **using** *wfS-letI* **by** (*meson wfG-cons wfG-cons-TRUE wfS-wf*)
show ‹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ Δ › **using** *wfS-letI* **by** *auto*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ′, Δ, *e*, *b*)› **using** *wfS-letI* **by** *auto*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ $\vdash_{wf}$ *s1* : *b-of* τ › **using** *wfS-let2I* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ τ › **using** *wfS-let2I wf-weakening1* **by** *auto*

have *toSet* $((x, \textit{b-of } \tau, \textit{TRUE}) \ \#_\Gamma \ \Gamma) \subseteq$ *toSet* $((x, \textit{b-of } \tau, \textit{TRUE}) \ \#_\Gamma \ \Gamma')$ **using** *wfS-let2I* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; $(x, \textit{b-of } \tau, \textit{TRUE}) \ \#_\Gamma \ \Gamma'$; $\Delta \vdash_{wf} s2 : b$ › **using** *wfS-let2I* **by** (*meson wfG-cons wfG-cons-TRUE wfS-wf*)
show ‹*atom* $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, s1, b, \tau)$› **using** *wfS-let2I* **by** *auto*
qed
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ v u Φ Δ b s)
show *?case* **proof**
show Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \tau$ **using** *wfS-varI wf-weakening1* **by** *auto*
show Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} v : \textit{b-of } \tau$ **using** *wfS-varI wf-weakening1* **by** *auto*
show *atom* $u \sharp (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, \tau, v, b)$ **using** *wfS-varI* **by** *auto*
show Θ ; Φ ; $\mathcal{B}$; Γ' ; $(u, \tau) \ \#_\Delta \ \Delta \vdash_{wf} s : b$ **using** *wfS-varI* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ x τ Γ Δ s b *tid dc*)
show *?case* **proof**
have *toSet* $((x, \textit{b-of } \tau, \textit{TRUE}) \ \#_\Gamma \ \Gamma) \subseteq$ *toSet* $((x, \textit{b-of } \tau, \textit{TRUE}) \ \#_\Gamma \ \Gamma')$ **using** *wfS-branchI* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; $(x, \textit{b-of } \tau, \textit{TRUE}) \ \#_\Gamma \ \Gamma'$; $\Delta \vdash_{wf} s : b$ › **using** *wfS-branchI* **by** (*meson wfG-cons wfG-cons-TRUE wfS-wf*)
show ‹*atom* $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, \Gamma', \tau)$› **using** *wfS-branchI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b css dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ x c Γ Δ s b)
show *?case* **proof**(*rule*)
show ‹ Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} c$ › **using** *wfS-assertI wf-weakening1* **by** *auto*
have Θ; $\mathcal{B} \vdash_{wf} (x, \textit{B-bool}, c) \ \#_\Gamma \ \Gamma'$ **proof**(*rule wfG-consI*)
show ‹ Θ; $\mathcal{B} \vdash_{wf} \Gamma'$ › **using** *wfS-assertI* **by** *auto*
show ‹*atom* $x \sharp \Gamma'$› **using** *wfS-assertI* **by** *auto*
show ‹ Θ; $\mathcal{B} \vdash_{wf}$ *B-bool* › **using** *wfS-assertI wfB-boolI wfX-wfY* **by** *metis*
have Θ; $\mathcal{B} \vdash_{wf} (x, \textit{B-bool}, \textit{TRUE}) \ \#_\Gamma \ \Gamma'$ **proof**
show $(\textit{TRUE}) \in \{\textit{TRUE}, \textit{FALSE}\}$ **by** *auto*
show ‹ Θ; $\mathcal{B} \vdash_{wf} \Gamma'$ › **using** *wfS-assertI* **by** *auto*
show ‹*atom* $x \sharp \Gamma'$› **using** *wfS-assertI* **by** *auto*
show ‹ Θ; $\mathcal{B} \vdash_{wf}$ *B-bool* › **using** *wfS-assertI wfB-boolI wfX-wfY* **by** *metis*
qed
thus ‹ Θ; $\mathcal{B}$; $(x, \textit{B-bool}, \textit{TRUE}) \ \#_\Gamma \ \Gamma' \vdash_{wf} c$ ›
using *wf-weakening1*(*2*)[*OF* ‹ Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} c$ › ‹ Θ; $\mathcal{B} \vdash_{wf} (x, \textit{B-bool}, \textit{TRUE}) \ \#_\Gamma \ \Gamma'$ ›] **by** *force*
qed
thus ‹ Θ; Φ; $\mathcal{B}$; $(x, \textit{B-bool}, c) \ \#_\Gamma \ \Gamma'$; $\Delta \vdash_{wf} s : b$ › **using** *wfS-assertI* **by** *fastforce*
show ‹ Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$ › **using** *wfS-assertI* **by** *auto*
show ‹*atom* $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, c, b, s)$› **using** *wfS-assertI* **by** *auto*
qed

qed(*metis wf-intros wf-weakening1*)+

lemmas *wf-weakening* = *wf-weakening1 wf-weakening2*

lemma *wfV-weakening-cons*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *c*::*c*
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* **and** *atom y* ♯ Γ **and** Θ; $\mathcal{B}$; ((*y*,*b*′,*TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *c*
shows Θ; $\mathcal{B}$; (*y*,*b*′,*c*) $\#_\Gamma$Γ $\vdash_{wf}$ *v* : *b*
proof −
have *wfG* Θ $\mathcal{B}$ ((*y*,*b*′,*c*) $\#_\Gamma$Γ) **using** *wfG-intros2 assms* **by** *auto*
moreover have *toSet* Γ ⊆ *toSet* ((*y*,*b*′,*c*) $\#_\Gamma$Γ) **using** *toSet.simps* **by** *auto*
ultimately show *?thesis* **using** *wf-weakening* **using** *assms*(*1*) **by** *blast*
qed

lemma *wfG-cons-weakening*:
fixes Γ′::Γ
assumes Θ; $\mathcal{B}$ $\vdash_{wf}$ ((*x*, *b*, *c*) $\#_\Gamma$ Γ) **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ **and** *toSet* Γ ⊆ *toSet* Γ′ **and** *atom x* ♯ Γ′
shows Θ; $\mathcal{B}$ $\vdash_{wf}$ ((*x*, *b*, *c*) $\#_\Gamma$ Γ′)
proof(*cases c* ∈ {*TRUE*,*FALSE*})
case *True*
then show *?thesis* **using** *wfG-wfB wfG-cons2I assms* **by** *auto*
next
case *False*
hence ∗:Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ∧ *atom x* ♯ Γ ∧ Θ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c*
using *wfG-elims*(*2*)[*OF assms*(*1*)] **by** *auto*
have *a1*:Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ′ **using** *wfG-wfB wfG-cons2I assms* **by** *simp*
moreover have *a2*:*toSet* ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) ⊆ *toSet* ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ′) **using** *toSet.simps assms* **by** *blast*
moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ′ **proof**
show (*TRUE*) ∈ {*TRUE*, *FALSE*} **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ **using** *assms* **by** *auto*
show *atom x* ♯ Γ′ **using** *assms* **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *assms wfG-elims* **by** *metis*
qed
hence Θ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ′ $\vdash_{wf}$ *c* **using** *wf-weakening a1 a2* ∗ **by** *auto*
then show *?thesis* **using** *wfG-cons1I*[*of c* Θ $\mathcal{B}$ Γ′ *x b*, *OF False*] *wfG-wfB assms* **by** *simp*
qed

lemma *wfT-weakening-aux*:
fixes Γ::Γ **and** Γ′::Γ **and** *c*::*c*
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄ **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ **and** *toSet* Γ ⊆ *toSet* Γ′ **and** *atom z* ♯ Γ′
shows Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄
proof
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′)›
using *wf-supp wfX-wfY assms fresh-prodN fresh-def x-not-in-b-set wfG-fresh-x* **by** *metis*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *assms wfT-elims* **by** *metis*
show ‹ Θ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′ $\vdash_{wf}$ *c* › **proof** −
have ∗:Θ; $\mathcal{B}$; (*z*,*b*,*TRUE*) $\#_\Gamma$Γ $\vdash_{wf}$ *c* **using** *wfT-wfC fresh-weakening assms* **by** *auto*
moreover have *a1*:Θ; $\mathcal{B}$ $\vdash_{wf}$ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′ **using** *wfG-cons2I assms* ‹Θ; $\mathcal{B}$ $\vdash_{wf}$ *b*› **by** *simp*
moreover have *a2*:*toSet* ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ) ⊆ *toSet* ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′) **using** *toSet.simps assms* **by** *blast*

moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ (z, b, *TRUE*) $\#_\Gamma$ Γ' **proof**
show (*TRUE*) $\in$ {*TRUE*, *FALSE*} **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ' **using** *assms* **by** *auto*
show *atom* z $\sharp$ Γ' **using** *assms* **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ b **using** *assms wfT-elims* **by** *metis*
qed
thus *?thesis* **using** *wf-weakening a1 a2* $*$ **by** *auto*
qed
qed

lemma *wfT-weakening-all*:
fixes Γ::Γ **and** Γ'::Γ **and** τ::τ
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ **and** Θ; $\mathcal{B}'$ $\vdash_{wf}$ Γ' **and** *toSet* Γ $\subseteq$ *toSet* Γ' **and** $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$
shows Θ; $\mathcal{B}'$; Γ' $\vdash_{wf}$ τ
using *wb-b-weakening assms wfT-weakening* **by** *metis*

lemma *wfT-weakening-nil*:
fixes Γ::Γ **and** Γ'::Γ **and** τ::τ
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ τ **and** Θ; $\mathcal{B}'$ $\vdash_{wf}$ Γ'
shows Θ; $\mathcal{B}'$; Γ' $\vdash_{wf}$ τ
using *wfT-weakening-all*
using *assms*(*1*) *assms*(*2*) *toSet.simps*(*1*) **by** *blast*

lemma *wfTh-wfT2*:
fixes x::x **and** v::v **and** τ::τ **and** G::Γ
assumes *wfTh* Θ **and** *AF-typedef s dclist* $\in$ *set* Θ **and**
(dc, τ) $\in$ *set dclist* **and** Θ ; B $\vdash_{wf}$ G
shows *supp* τ = {} **and** $\tau[x::=v]_{\tau v}$ = τ **and** *wfT* Θ B G τ
proof −
show *supp* τ = {} **proof**(*rule ccontr*)
assume *a1*: *supp* τ $\neq$ {}
have *supp* Θ $\neq$ {} **proof** −
obtain *dclist* **where** *dc*: *AF-typedef s dclist* $\in$ *set* Θ $\wedge$ (dc, τ) $\in$ *set dclist*
using *assms* **by** *auto*
hence *supp* (dc,τ) $\neq$ {}
using *a1* **by** (*simp add*: *supp-Pair*)
hence *supp dclist* $\neq$ {} **using** *dc supp-list-member* **by** *auto*
hence *supp* (*AF-typedef s dclist*) $\neq$ {} **using** *type-def.supp* **by** *auto*
thus *?thesis* **using** *supp-list-member dc* **by** *auto*
qed
thus *False* **using** *assms wfTh-supp* **by** *simp*
qed
thus $\tau[x::=v]_{\tau v}$ = τ **by** (*simp add*: *fresh-def*)
have *wfT* Θ {||} *GNil* τ **using** *assms wfTh-wfT* **by** *auto*
thus *wfT* Θ B G τ **using** *assms wfT-weakening-nil* **by** *simp*
qed

lemma *wf-d-weakening*:
fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** s::s **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows

$\Theta; \Phi; \mathcal{B}; \Gamma \ ; \Delta \vdash_{wf} e : b \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta; \Phi; \mathcal{B}; \ \Gamma \ ; \Delta' \vdash_{wf}$ $e : b$ **and**

$\Theta; \Phi; \mathcal{B}; \Gamma \ ; \Delta \vdash_{wf} s : b \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta; \Phi; \mathcal{B}; \ \Gamma \ ; \Delta' \vdash_{wf}$ $s : b$ **and**

$\Theta \ ; \Phi \ ; \mathcal{B} \ ; \Gamma \ ; \Delta \ ; tid \ ; dc \ ; t \vdash_{wf} cs : b \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta;$ $\Phi; \mathcal{B}; \Gamma \ ; \Delta' \ ; tid \ ; dc \ ; t \vdash_{wf} cs : b$ **and**

$\Theta \ ; \Phi \ ; \mathcal{B} \ ; \Gamma \ ; \Delta \ ; tid \ ; dclist \vdash_{wf} css : b \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta;$ $\Phi; \mathcal{B}; \Gamma \ ; \Delta' \ ; tid \ ; dclist \vdash_{wf} css : b$ **and**

$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow$ *True* **and**

$\Theta; \mathcal{B}; \Gamma \ \vdash_{wf} \Delta \Longrightarrow$ *True* **and**

$\Theta \ ; \Phi \ \ \vdash_{wf} ftq \Longrightarrow$ *True* **and**

$\Theta \ ; \Phi \ \ ; \mathcal{B} \vdash_{wf} ft \Longrightarrow$ *True*

proof(*nominal-induct*
  b **and** b **and** b **and** b **and** Φ **and** Δ **and** *ftq* **and** *ft*
  avoiding: Δ'
  rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
 case (*wfE-valI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ v\ b$)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-plusI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 v2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-leqI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 v2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-eqI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 b v2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-fstI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 b1 b2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-sndI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 b1 b2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-concatI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 v2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-splitI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1 v2*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-lenI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *v1*)
 then show *?case* **using** *wf-intros* **by** *metis*
next
 case (*wfE-appI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ f\ x\ b\ c\ \tau\ s\ v$)
 then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfE-appPI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ b'\ bv\ v\ \tau\ f\ x\ b\ c\ s$)
  show *?case* **proof**(*rule*, (*rule wfE-appPI*)+)
   show ‹*atom bv* $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ, Δ', b', v, (*b-of* τ)[*bv*::=*b'*]$_b$)› **using** *wfE-appPI* **by** *auto*
   show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
   show ‹ $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b[bv::=b']_b$ › **using** *wfE-appPI* **by** *auto*

```
  qed
next
  case (wfE-mvarI Θ Φ ℬ Γ Δ u τ)
  show ?case proof
    show ‹ Θ  ⊢wf Φ › using wfE-mvarI by auto
    show ‹ Θ; ℬ; Γ ⊢wf Δ' › using wfE-mvarI by auto
    show ‹(u, τ) ∈ setD Δ'› using wfE-mvarI by auto
  qed
next
  case (wfS-valI Θ Φ ℬ Γ v b Δ)
  then show ?case using wf-intros by metis
next
  case (wfS-letI Θ Φ ℬ Γ Δ e b' x s b)
  show ?case proof(rule)
    show ‹ Θ; Φ; ℬ; Γ; Δ' ⊢wf e : b' › using wfS-letI by auto
    have Θ; ℬ  ⊢wf  (x, b', TRUE)  #Γ Γ  using wfG-cons2I wfX-wfY wfS-letI by metis
    hence Θ; ℬ; (x, b', TRUE)  #Γ Γ ⊢wf Δ' using wf-weakening2(6) wfS-letI by force
    thus ‹ Θ ; Φ  ; ℬ ; (x, b', TRUE)  #Γ Γ ; Δ' ⊢wf s : b › using wfS-letI by metis
    show ‹ Θ; ℬ; Γ ⊢wf Δ' › using wfS-letI by auto
    show ‹atom x ♯ (Φ, Θ, ℬ, Γ, Δ', e, b)› using wfS-letI by auto
  qed
next
  case (wfS-assertI Θ Φ ℬ x c Γ Δ s b)
  show ?case proof
    have Θ; ℬ; (x, B-bool, c) #Γ Γ ⊢wf Δ' proof(rule  wf-weakening2(6))
      show ‹ Θ; ℬ; Γ ⊢wf Δ' › using wfS-assertI by auto
    next
      show ‹ Θ; ℬ  ⊢wf (x, B-bool, c) #Γ Γ › using wfS-assertI wfX-wfY by metis
    next
      show ‹toSet Γ ⊆ toSet ((x, B-bool, c) #Γ Γ)› using wfS-assertI by auto
    qed
    thus  ‹ Θ; Φ; ℬ; (x, B-bool, c) #Γ Γ ; Δ' ⊢wf s : b › using wfS-assertI wfX-wfY by metis
  next
    show ‹ Θ; ℬ; Γ   ⊢wf c › using wfS-assertI by auto
  next
    show ‹ Θ; ℬ; Γ ⊢wf Δ' › using wfS-assertI by auto
  next
    show ‹atom x ♯ (Φ, Θ, ℬ, Γ, Δ', c, b, s)› using wfS-assertI by auto
  qed
next
  case (wfS-let2I Θ Φ ℬ Γ Δ s1 τ x s2 b)
  show ?case proof
    show ‹ Θ; Φ; ℬ; Γ; Δ' ⊢wf s1 : b-of τ › using wfS-let2I by auto
    show ‹ Θ; ℬ; Γ   ⊢wf τ › using wfS-let2I by auto
    have Θ; ℬ  ⊢wf  (x, b-of τ, TRUE)  #Γ Γ  using wfG-cons2I wfX-wfY wfS-let2I by metis
    hence Θ; ℬ; (x, b-of τ, TRUE)  #Γ Γ ⊢wf Δ' using wf-weakening2(6) wfS-let2I by force
    thus  ‹ Θ ; Φ  ; ℬ ; (x, b-of τ, TRUE)  #Γ Γ ; Δ' ⊢wf s2 : b › using  wfS-let2I by metis
    show ‹atom x ♯ (Φ, Θ, ℬ, Γ, Δ', s1, b,τ)› using wfS-let2I by auto
  qed
next
  case (wfS-ifI Θ ℬ Γ v Φ Δ s1 b s2)
  then show ?case using wf-intros by metis
```

next
  case (*wfS-varI* Θ *B* Γ τ *v u* Φ Δ *b s*)
  show *?case* **proof**
    show ‹ Θ; *B*; Γ $\vdash_{wf}$ τ › **using** *wfS-varI* **by** *auto*
    show ‹ Θ; *B*; Γ $\vdash_{wf}$ *v* : *b-of* τ › **using** *wfS-varI* **by** *auto*
    show ‹*atom u* ♯ (Φ, Θ, *B*, Γ, Δ′, τ, *v*, *b*)› **using** *wfS-varI setD.simps* **by** *auto*
    have Θ; *B*; Γ $\vdash_{wf}$ (*u*, τ) $\#_{\Delta}$ Δ′ **using** *wfS-varI wfD-cons setD.simps u-fresh-d* **by** *metis*
    thus ‹ Θ ; Φ ; *B* ; Γ ; (*u*, τ) $\#_{\Delta}$ Δ′ $\vdash_{wf}$ *s* : *b* › **using** *wfS-varI setD.simps* **by** *blast*
  qed
next
  case (*wfS-assignI u* τ Δ Θ *B* Γ Φ *v*)
  show *?case* **proof**
    show ‹(*u*, τ) ∈ *setD* Δ′› **using** *wfS-assignI setD.simps* **by** *auto*
    show ‹ Θ; *B*; Γ $\vdash_{wf}$ Δ′ › **using** *wfS-assignI* **by** *auto*
    show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfS-assignI* **by** *auto*
    show ‹ Θ; *B*; Γ $\vdash_{wf}$ *v* : *b-of* τ › **using** *wfS-assignI* **by** *auto*
  qed
next
  case (*wfS-whileI* Θ Φ *B* Γ Δ *s1 s2 b*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfS-seqI* Θ Φ *B* Γ Δ *s1 s2 b*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfS-matchI* Θ *B* Γ *v tid dclist* Δ Φ *cs b*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfS-branchI* Θ Φ *B x* τ Γ Δ *s b tid dc*)
  show *?case* **proof**
    have Θ; *B* $\vdash_{wf}$ (*x*, *b-of* τ, *TRUE*) $\#_{\Gamma}$ Γ **using** *wfG-cons2I wfX-wfY wfS-branchI* **by** *metis*
    hence Θ; *B*; (*x*, *b-of* τ, *TRUE*) $\#_{\Gamma}$ Γ $\vdash_{wf}$ Δ′ **using** *wf-weakening2*(*6*) *wfS-branchI* **by** *force*
    thus ‹ Θ ; Φ ; *B* ; (*x*, *b-of* τ, *TRUE*) $\#_{\Gamma}$ Γ ; Δ′ $\vdash_{wf}$ *s* : *b* › **using** *wfS-branchI* **by** *simp*
    show ‹ *atom x* ♯ (Φ, Θ, *B*, Γ, Δ′, Γ, τ)› **using** *wfS-branchI* **by** *auto*
    show ‹ Θ; *B*; Γ $\vdash_{wf}$ Δ′ › **using** *wfS-branchI* **by** *auto*
  qed
next
  case (*wfS-finalI* Θ Φ *B* Γ Δ *tid dclist′ cs b dclist*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfS-cons* Θ Φ *B* Γ Δ *tid dclist′ cs b css dclist*)
  then show *?case* **using** *wf-intros* **by** *metis*
qed(*auto*+)

8.15 Useful well-formedness instances

Well-formedness for particular constructs that we will need later

lemma *wfC-e-eq*:
  fixes *ce*::*ce* **and** Γ::Γ
  assumes Θ ; *B* ; Γ $\vdash_{wf}$ *ce* : *b* **and** *atom x* ♯ Γ
  shows Θ ; *B* ; ((*x*, *b*, *TRUE*) $\#_{\Gamma}$ Γ) $\vdash_{wf}$ (*CE-val* (*V-var x*) == *ce*)
proof −

have Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *assms wfX-wfB* **by** *auto*
hence *wbg*: Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ **using** *wfX-wfY assms* **by** *auto*
show *?thesis* **proof**
show ∗:Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *CE-val* (*V-var x*) : *b*
proof(*rule*)
show Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *V-var x* : *b* **proof**
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ **using** *wfG-cons2I wfX-wfY assms* ‹Θ; $\mathcal{B}$ $\vdash_{wf}$ *b*› **by** *auto*
show *Some* (*b*, *TRUE*) = *lookup* ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) *x* **using** *lookup.simps* **by** *auto*
qed
qed
show Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *ce* : *b*
using *assms wf-weakening1*(*8*)[*OF assms*(*1*), *of* (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ] ∗ *toSet.simps wfX-wfY*
by (*metis Un-subset-iff equalityE*)
qed
qed

lemma *wfC-e-eq2*:
fixes *e1*::*ce* **and** *e2*::*ce*
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *e1* : *b* **and** Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *e2* : *b* **and** $\vdash_{wf}$ Θ **and** *atom x* ♯ Γ
shows Θ; $\mathcal{B}$; (*x*, *b*, (*CE-val* (*V-var x*)) == *e1*) $\#_\Gamma$ Γ $\vdash_{wf}$ (*CE-val* (*V-var x*)) == *e2*
proof(*rule wfC-eqI*)
have ∗: Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ Γ **proof**(*rule wfG-cons1I*)
show (*CE-val* (*V-var x*) == *e1*) ∉ {*TRUE*, *FALSE*} **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ **using** *assms wfX-wfY* **by** *metis*
show ∗:*atom x* ♯ Γ **using** *assms* **by** *auto*
show Θ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *CE-val* (*V-var x*) == *e1* **using** *wfC-e-eq assms* ∗ **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *assms wfX-wfB* **by** *auto*
qed
show Θ; $\mathcal{B}$; (*x*, *b*, *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ Γ $\vdash_{wf}$ *CE-val* (*V-var x*) : *b* **using** *assms* ∗ *wfCE-valI wfV-varI* **by** *auto*
show Θ; $\mathcal{B}$; (*x*, *b*, *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ Γ $\vdash_{wf}$ *e2* : *b* **proof**(*rule wf-weakening1*(*8*))
show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *e2* : *b* **using** *assms* **by** *auto*
show Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ Γ **using** ∗ **by** *auto*
show *toSet* Γ ⊆ *toSet* ((*x*, *b*, *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ Γ) **by** *auto*
qed
qed

lemma *wfT-wfT-if-rev*:
assumes *wfV P* $\mathcal{B}$ Γ *v* (*base-for-lit l*) **and** *wfT P* $\mathcal{B}$ Γ *t* **and** ‹*atom z1* ♯ Γ›
shows *wfT P* $\mathcal{B}$ Γ (⦃ *z1* : *b-of t* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t z1*) ⦄)
proof
show ‹ *P*; $\mathcal{B}$ $\vdash_{wf}$ *b-of t* › **using** *wfX-wfY assms* **by** *meson*
have *wfg*: *P*; $\mathcal{B}$ $\vdash_{wf}$ (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ **using** *assms wfV-wf wfG-cons2I wfX-wfY*
by (*meson wfG-cons-TRUE*)
show ‹ *P*; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ $[\ v\ ]^{ce}$ == $[\ [\ l\ ]^{v}\ ]^{ce}$ *IMP* *c-of t z1* › **proof**
show ∗: ‹ *P*; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ $[\ v\ ]^{ce}$ == $[\ [\ l\ ]^{v}\ ]^{ce}$ ›
proof(*rule wfC-eqI*[**where** *b*=*base-for-lit l*])
show *P*; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ $[\ v\ ]^{ce}$: *base-for-lit l*
using *assms wf-intros wf-weakening wfg* **by** (*meson wfV-weakening-cons*)
show *P*; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ $[\ [\ l\ ]^{v}\ ]^{ce}$: *base-for-lit l* **using** *wfg assms wf-intros wf-weakening wfV-weakening-cons* **by** *meson*
qed

have $t = \{\!\mid z1 : b\text{-}of\ t \mid c\text{-}of\ t\ z1 \mid\!\}$ **using** *c-of-eq*
using *assms(2) assms(3) b-of-c-of-eq wfT-x-fresh* **by** *auto*
thus ‹ $P; \mathcal{B}$; $(z1, b\text{-}of\ t, TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c\text{-}of\ t\ z1$ › **using** *wfT-wfC assms wfG-elims* ∗ **by** *simp*
qed
show ‹*atom z1* ♯ $(P, \mathcal{B}, \Gamma)$› **using** *assms wfG-fresh-x wfX-wfY* **by** *metis*
qed

lemma *wfT-eq-imp*:
fixes *zz*::*x* **and** *ll*::*l* **and** τ'::τ
assumes *base-for-lit ll = B-bool* **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ τ' **and**
Θ ; {||} $\vdash_{wf}$ $(x, b\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}, c\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}\ x)\ \#_\Gamma\ GNil$ **and** *atom zz* ♯ *x*
shows Θ ; {||} ; $(x, b\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}, c\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}\ x)\ \#_\Gamma$
$GNil\ \vdash_{wf} \{\!\mid zz : b\text{-}of\ \tau' \mid [\ [\ x\ ]^v\ ]^{ce}\ ==\ [\ [\ ll\ ]^v\ ]^{ce}\ \ IMP\ \ c\text{-}of\ \tau'\ zz\ \mid\!\}$
proof(*rule wfT-wfT-if-rev*)
show ‹ Θ ; {||} ; $(x, b\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}, c\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}\ x)\ \#_\Gamma\ GNil \vdash_{wf} [\ x\ ]^v$: *base-for-lit ll* ›
using *wfV-varI lookup.simps base-for-lit.simps assms* **by** *simp*
show ‹ Θ ; {||} ; $(x, b\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}, c\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}\ x)\ \#_\Gamma\ GNil\ \vdash_{wf}\ \tau'$ ›
using *wf-weakening assms toSet.simps* **by** *auto*
show ‹*atom zz* ♯ $(x, b\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}, c\text{-}of \{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}\ x)\ \#_\Gamma\ GNil$›
unfolding *fresh-GCons fresh-prod3 b-of.simps c-of-true*
using *x-fresh-b fresh-GNil c-of-true c.fresh assms* **by** *metis*
qed

lemma *wfC-v-eq*:
fixes *ce*::*ce* **and** Γ::Γ **and** *v*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma\ \vdash_{wf}\ v : b$ **and** *atom x* ♯ Γ
shows Θ ; $\mathcal{B}$; $((x, b, TRUE)\ \#_\Gamma\ \Gamma)\ \vdash_{wf}$ (*CE-val* (*V-var x*) == *CE-val v*)
using *wfC-e-eq wfCE-valI assms wfX-wfY* **by** *auto*

lemma *wfT-e-eq*:
fixes *ce*::*ce*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *ce* : *b* **and** *atom z* ♯ Γ
shows Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!\mid z : b \mid$ *CE-val* (*V-var z*) == *ce* $\mid\!\}$
proof
show Θ; $\mathcal{B} \vdash_{wf} b$ **using** *wfX-wfB assms* **by** *auto*
show *atom z* ♯ $(\Theta, \mathcal{B}, \Gamma)$ **using** *assms wfG-fresh-x wfX-wfY* **by** *metis*
show Θ ; $\mathcal{B}$; $(z, b, TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf}$ *CE-val* (*V-var z*) == *ce*
using *wfTI wfC-e-eq assms wfTI* **by** *auto*
qed

lemma *wfT-v-eq*:
assumes *wfB* Θ $\mathcal{B}$ *b* **and** *wfV* Θ $\mathcal{B}$ Γ *v b* **and** *atom z* ♯ Γ
shows *wfT* Θ $\mathcal{B}$ Γ $\{\!\mid z : b \mid$ *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)$\mid\!\}$
using *wfT-e-eq wfE-valI assms wfX-wfY*
by (*simp add*: *wfCE-valI*)

lemma *wfC-wfG*:
fixes Γ::Γ **and** *c*::*c* **and** *b*::*b*

assumes Θ ; *B* ; Γ $\vdash_{wf}$ *c* **and** Θ ; *B* $\vdash_{wf}$ *b* **and** *atom x* ♯ Γ
shows Θ ; *B* $\vdash_{wf}$ (*x*,*b*,*c*)$\#_\Gamma$ Γ
proof −
have Θ ; *B* $\vdash_{wf}$ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ **using** *wfG-cons2I assms wfX-wfY* **by** *fast*
hence Θ ; *B* ; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c* **using** *wfC-weakening assms* **by** *force*
thus *?thesis* **using** *wfG-consI assms wfX-wfY* **by** *metis*
qed

8.16 Replacing the constraint on a variable in a context

lemma *wfG-cons-fresh2*:
fixes Γ′::Γ
assumes *wfG P B* (((*x*′,*b*′,*c*′) $\#_\Gamma$ Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ))
shows *x*′≠*x*
proof −
have *atom x*′ ♯ (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ)
using *assms wfG-elims*(*2*) **by** *blast*
thus *?thesis*
using *fresh-gamma-append*[*of atom x*′ Γ′ (*x*, *b*, *c*) $\#_\Gamma$ Γ] *fresh-GCons fresh-prod3*[*of atom x*′ *x b c*]
by *auto*
qed

lemma *replace-in-g-inside*:
fixes Γ::Γ
assumes *wfG P B* (Γ′@((*x*,*b0*,*c0*′) $\#_\Gamma$Γ))
shows *replace-in-g* (Γ′@((*x*,*b0*,*c0*′) $\#_\Gamma$Γ)) *x c0* = (Γ′@((*x*,*b0*,*c0*) $\#_\Gamma$Γ))
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *replace-in-g.simps* **by** *auto*
next
case (*GCons x*′ *b*′ *c*′ Γ′′)
hence *P*; *B* $\vdash_{wf}$ ((*x*′, *b*′, *c*′) $\#_\Gamma$ (Γ′′@ (*x*, *b0*, *c0*′) $\#_\Gamma$ Γ)) **by** *simp*
hence *x* ≠ *x*′ **using** *wfG-cons-fresh2* **by** *metis*
then show *?case* **using** *replace-in-g.simps GCons* **by** (*simp add*: *wfG-cons*)
qed

lemma *wfG-supp-rig-eq*:
fixes Γ::Γ
assumes *wfG P B* (Γ′′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) **and** *wfG P B* (Γ′′ @ (*x*, *b0*, *c0*′) $\#_\Gamma$ Γ)
shows *supp* (Γ′′ @ (*x*, *b0*, *c0*′) $\#_\Gamma$ Γ) ∪ *supp B* = *supp* (Γ′′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) ∪ *supp B*
using *assms* **proof**(*induct* Γ′′)
case *GNil*
have *supp* (*GNil* @ (*x*, *b0*, *c0*′) $\#_\Gamma$ Γ) ∪ *supp B* = *supp* ((*x*, *b0*, *c0*′) $\#_\Gamma$ Γ) ∪ *supp B* **using** *supp-Cons supp-GNil* **by** *auto*
also have ... = *supp x* ∪ *supp b0* ∪ *supp c0*′ ∪ *supp* Γ ∪ *supp B* **using** *supp-GCons* **by** *auto*
also have ... = *supp x* ∪ *supp b0* ∪ *supp c0* ∪ *supp* Γ ∪ *supp B* **using** *GNil wfG-wfC*[*THEN wfC-supp-cons*(*2*)] **by** *fastforce*
also have ... = (*supp* ((*x*, *b0*, *c0*) $\#_\Gamma$ Γ)) ∪ *supp B* **using** *supp-GCons* **by** *auto*
finally have *supp* (*GNil* @ (*x*, *b0*, *c0*′) $\#_\Gamma$ Γ) ∪ *supp B* = *supp* (*GNil* @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) ∪ *supp B* **using** *supp-Cons supp-GNil* **by** *auto*
then show *?case* **using** *supp-GCons wfG-cons2* **by** *auto*
next

case (*GCons xbc* Γ*1*)
moreover have (*xbc* $\#_\Gamma$ Γ*1*) @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ = (*xbc* $\#_\Gamma$ (Γ*1* @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ)) **by** *simp*
moreover have (*xbc* $\#_\Gamma$ Γ*1*) @ (*x*, *b0*, *c0′*) $\#_\Gamma$ Γ = (*xbc* $\#_\Gamma$ (Γ*1* @ (*x*, *b0*, *c0′*) $\#_\Gamma$ Γ)) **by** *simp*
ultimately have (*P*; *B* $\vdash_{wf}$ Γ*1* @ ((*x*, *b0*, *c0*) $\#_\Gamma$ Γ)) ∧ *P*; *B* $\vdash_{wf}$ Γ*1* @ ((*x*, *b0*, *c0′*) $\#_\Gamma$ Γ) **using** *wfG-cons2* **by** *metis*
thus *?case* **using** *GCons supp-GCons* **by** *auto*
qed

lemma *fresh-replace-inside*[*ms-fresh*]:
fixes *y*::*x* **and** Γ::Γ
assumes *wfG P B* (Γ′′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **and** *wfG P B* (Γ′′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ)
shows *atom y* ♯ (Γ′′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) = *atom y* ♯ (Γ′′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ)
unfolding *fresh-def* **using** *wfG-supp-rig-eq assms x-not-in-b-set* **by** *fast*

lemma *wf-replace-inside1*:
fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** *c′′*::*c* **and** *c′*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *b′*::*b* **and** *b*::*b* **and** *s*::*s*
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows *wfV-replace-inside*: Θ; *B*; *G* $\vdash_{wf}$ *v* : *b′* ⟹ *G* = (Γ′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ) ⟹ Θ; *B*; ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) $\vdash_{wf}$ *c* ⟹ Θ ; *B* ; (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *v* : *b′* **and**
wfC-replace-inside: Θ; *B*; *G* $\vdash_{wf}$ *c′′* ⟹ *G* = (Γ′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ) ⟹ Θ; *B*; ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) $\vdash_{wf}$ *c* ⟹ Θ ; *B* ; (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *c′′* **and**
wfG-replace-inside: Θ; *B* $\vdash_{wf}$ *G* ⟹ *G* = (Γ′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ) ⟹ Θ; *B*; ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) $\vdash_{wf}$ *c* ⟹ Θ; *B* $\vdash_{wf}$ (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **and**
wfT-replace-inside: Θ; *B*; *G* $\vdash_{wf}$ τ ⟹ *G* = (Γ′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ) ⟹ Θ; *B*; ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) $\vdash_{wf}$ *c* ⟹ Θ ; *B* ; (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) $\vdash_{wf}$ τ **and**
Θ; *B*; Γ $\vdash_{wf}$ *ts* ⟹ *True* **and**
$\vdash_{wf}$ *P* ⟹ *True* **and**
Θ; *B* $\vdash_{wf}$ *b* ⟹ *True* **and**
wfCE-replace-inside: Θ ; *B* ; *G* $\vdash_{wf}$ *ce* : *b′* ⟹ *G* = (Γ′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ) ⟹ Θ; *B*; ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) $\vdash_{wf}$ *c* ⟹ Θ ; *B* ; (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *ce* : *b′* **and**
Θ $\vdash_{wf}$ *td* ⟹ *True*
proof(*nominal-induct*
b′ **and** *c′′* **and** *G* **and** τ **and** *ts* **and** *P* **and** *b* **and** *b′* **and** *td*
avoiding: Γ′ *c′*
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* Θ *B* Γ*2 b2 c2 x2*)
then show *?case* **using** *wf-intros* **by** (*metis lookup-in-rig-eq lookup-in-rig-neq replace-in-g-inside*)
next
case (*wfV-conspI s bv dclist* Θ *dc x1 b′ c1 B b1* Γ*1 v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, ⦃ *x1* : *b′* | *c1* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; *B* $\vdash_{wf}$ *b1* › **using** *wfV-conspI* **by** *auto*
show ∗: ‹ Θ; *B*; Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ *v* : *b′*[*bv*::=*b1*]$_{bb}$ › **using** *wfV-conspI* **by** *auto*
moreover have Θ; *B* $\vdash_{wf}$ Γ′ @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ **using** *wfV-wf wfV-conspI* **by** *simp*
ultimately have *atom bv* ♯ Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **unfolding** *fresh-def* **using** *wfV-wf wfG-supp-rig-eq wfV-conspI*
by (*metis Un-iff fresh-def*)
thus ‹*atom bv* ♯ (Θ, *B*, Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ, *b1*, *v*)›

unfolding *fresh-prodN* **using** *fresh-prodN wfV-conspI* **by** *metis*
qed
next
case (*wfTI z* Θ *ℬ G b1 c1*)
show *?case* **proof**
show ‹ Θ; *ℬ* $\vdash_{wf}$ *b1* › **using** *wfTI* **by** *auto*

have Θ; *ℬ* $\vdash_{wf}$ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-consI wfTI wfG-cons wfX-wfY* **by** *metis*
moreover hence ∗:*wfG* Θ *ℬ* (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **using** *wfX-wfY*
by (*metis append-g.simps*(*2*) *wfG-cons2 wfTI.hyps wfTI.prems*(*1*) *wfTI.prems*(*2*))
hence ‹*atom z* ♯ Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ›
using *fresh-replace-inside*[*of* Θ *ℬ* Γ′ *x b c* Γ *c′ z*,*OF* ∗] *wfTI wfX-wfY wfG-elims* **by** *metis*
thus ‹*atom z* ♯ (Θ, *ℬ*, Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ)› **using** *wfG-fresh-x*[*OF* ∗] **by** *auto*

have (*z*, *b1*, *TRUE*) $\#_\Gamma$ *G* = ((*z*, *b1*, *TRUE*) $\#_\Gamma$ Γ′) @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ
using *wfTI append-g.simps* **by** *metis*
thus ‹ Θ; *ℬ*; (*z*, *b1*, *TRUE*) $\#_\Gamma$ Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* ›
using *wfTI*(*9*)[*OF* - *wfTI*(*11*)] **by** *fastforce*
qed
next
case (*wfG-nilI* Θ)
hence *GNil* = (*x*, *b*, *c′*) $\#_\Gamma$ Γ **using** *append-g.simps* Γ*.distinct GNil-append* **by** *auto*
hence *False* **using** Γ*.distinct* **by** *auto*
then show *?case* **by** *auto*
next
case (*wfG-cons1I c1* Θ *ℬ G x1 b1*)
show *?case* **proof**(*cases* Γ′=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons1I wfG-consI* **by** *auto*
next
case *False*
then obtain *G′*::Γ **where** ∗:(*x1*, *b1*, *c1*) $\#_\Gamma$ *G′* = Γ′ **using** *wfG-cons1I wfG-cons1I*(*7*) *GCons-eq-append-conv* **by** *auto*
hence ∗∗: *G* = *G′* @ (*x*, *b*, *c′*) $\#_\Gamma$ Γ **using** *wfG-cons1I* **by** *auto*
hence Θ; *ℬ* $\vdash_{wf}$ *G′* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons1I* **by** *auto*
have Θ; *ℬ* $\vdash_{wf}$ (*x1*, *b1*, *c1*) $\#_\Gamma$ *G′* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **proof**(*rule Wellformed.wfG-cons1I*)
show *c1* ∉ {*TRUE*, *FALSE*} **using** *wfG-cons1I* **by** *auto*
show Θ; *ℬ* $\vdash_{wf}$ *G′* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons1I*(*3*)[*of G′*,*OF* ∗∗] *wfG-cons1I* **by** *auto*
show *atom x1* ♯ *G′* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons1I* ∗ ∗∗ *fresh-replace-inside* **by** *metis*
show Θ; *ℬ*; (*x1*, *b1*, *TRUE*) $\#_\Gamma$ *G′* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* **using** *wfG-cons1I*(*6*)[*of* (*x1*, *b1*, *TRUE*) $\#_\Gamma$ *G′*] *wfG-cons1I* ∗∗ **by** *auto*
show Θ; *ℬ* $\vdash_{wf}$ *b1* **using** *wfG-cons1I* **by** *auto*
qed
thus *?thesis* **using** ∗ **by** *auto*
qed
next
case (*wfG-cons2I c1* Θ *ℬ G x1 b1*)
show *?case* **proof**(*cases* Γ′=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons2I wfG-consI* **by** *auto*
next
case *False*

then obtain G'::Γ **where** $*$:$(x1,\ b1,\ c1)\ \ \#_\Gamma\ \ G' = \Gamma'$ **using** *wfG-cons2I GCons-eq-append-conv* **by** *auto*

hence $**$: $G = G'\ @\ (x,\ b,\ c')\ \ \#_\Gamma\ \Gamma$ **using** *wfG-cons2I* **by** *auto*

moreover have $\Theta;\ \mathcal{B} \vdash_{wf}\ G'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma$ **using** *wfG-cons2I* $*\ **$ **by** *auto*

moreover hence *atom x1* $\sharp\ G'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma$ **using** *wfG-cons2I* $*\ **$ *fresh-replace-inside* **by** *metis*

ultimately show *?thesis* **using** *Wellformed.wfG-cons2I*[*OF wfG-cons2I*(*1*), *of* $\Theta\ \mathcal{B}\ G'@\ (x,\ b,\ c)\ \#_\Gamma\ \Gamma$ *x1 b1*] *wfG-cons2I* $*\ **$ **by** *auto*

qed

qed(*metis wf-intros*)+

lemma *wf-replace-inside2*:

fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** c''::*c* **and** c'::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** b'::*b* **and** *b*::*b* **and** *s*::*s*

and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows

$\Theta\ ;\ \Phi\ ;\ \ \mathcal{B}\ ;\ G\ ;\ D\ \vdash_{wf}\ e\ :\ b' \Longrightarrow G =\ \ (\Gamma'\ @\ (x,\ b,\ c')\ \ \#_\Gamma\ \Gamma) \Longrightarrow \Theta;\ \mathcal{B};\ ((x,b,TRUE)\ \#_\Gamma\Gamma) \vdash_{wf}\ c \Longrightarrow \Theta\ ;\ \Phi\ ;\ \ \mathcal{B}\ ;\ (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma);\ D \vdash_{wf}\ e\ :\ b'$ **and**

$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma\ ;\ \Delta \vdash_{wf}\ s\ :\ b \Longrightarrow$ *True* **and**

$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dc\ ;\ t \vdash_{wf}\ cs\ :\ b \Longrightarrow$ *True* **and**

$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dclist \vdash_{wf}\ css\ :\ b \Longrightarrow$ *True* **and**

$\Theta \vdash_{wf}\ \Phi \Longrightarrow$ *True* **and**

$\Theta;\ \mathcal{B};\ G\ \ \vdash_{wf}\ \Delta \Longrightarrow\ \ G =\ \ (\Gamma'\ @\ (x,\ b,\ c')\ \ \#_\Gamma\ \Gamma) \Longrightarrow \Theta;\ \mathcal{B};\ ((x,b,TRUE)\ \#_\Gamma\Gamma) \vdash_{wf}\ c \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma) \vdash_{wf}\ \Delta$ **and**

$\Theta\ ;\ \Phi\ \ \ \vdash_{wf}\ ftq \Longrightarrow$ *True* **and**

$\Theta\ ;\ \Phi\ \ ;\ \mathcal{B} \vdash_{wf}\ ft \Longrightarrow$ *True*

proof(*nominal-induct*

b' **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*

avoiding: $\Gamma'\ c'$

rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-valI* **by** *auto*

next

case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-plusI* **by** *auto*

next

case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-leqI* **by** *auto*

next

case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b v2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-eqI* **by** *metis*

next

case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-fstI* **by** *metis*

next

case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-sndI* **by** *metis*

next

case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-concatI* **by** *auto*

next

case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)

then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-splitI* **by** *auto*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-lenI* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-appI* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ'' Δ *b' bv v* τ *f x1 b1 c1 s*)
show *?case* **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b'* › **using** *wfE-appPI* **by** *auto*
show ∗:‹ Θ; $\mathcal{B}$; Γ' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ *v* : *b1*[*bv*::=*b'*]$_b$ › **using** *wfE-appPI wf-replace-inside1* **by** *auto*

moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (*x*, *b*, *c'*) $\#_\Gamma$ Γ **using** *wfV-wf wfE-appPI* **by** *metis*
ultimately have *atom bv* ♯ Γ' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ
unfolding *fresh-def* **using** *wfV-wf wfG-supp-rig-eq wfE-appPI Un-iff fresh-def* **by** *metis*

thus ‹*atom bv* ♯ (Φ, Θ, $\mathcal{B}$, Γ' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ, Δ, *b'*, *v*, (*b-of* τ)[*bv*::=*b'*]$_b$)›
using *wfE-appPI fresh-prodN* **by** *metis*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x1 b1 c1* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
qed
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-mvarI* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI*
by (*simp add*: *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfD-cons*)
next
case (*wfFTNone* Θ Φ *ft*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI* **by** *metis*
next
case (*wfFTSome* Θ Φ *bv ft*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI* **by** *metis*
qed(*auto*)

lemmas *wf-replace-inside* = *wf-replace-inside1 wf-replace-inside2*

lemma *wfC-replace-cons*:
assumes *wfG P* $\mathcal{B}$ ((*x*,*b*,*c1*) $\#_\Gamma$Γ) **and** *wfC P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) $\#_\Gamma$Γ) *c2*
shows *wfC P* $\mathcal{B}$ ((*x*,*b*,*c1*) $\#_\Gamma$Γ) *c2*
proof −
have *wfC P* $\mathcal{B}$ (*GNil*@((*x*,*b*,*c1*) $\#_\Gamma$Γ)) *c2* **proof**(*rule wf-replace-inside1*(*2*))
show *P*; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c2* **using** *wfG-elim2 assms* **by** *auto*
show ‹(*x*, *b*, *TRUE*) $\#_\Gamma$ Γ = *GNil* @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ› **using** *append-g.simps* **by** *auto*

show ‹P; $\mathcal{B}$; (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* › **using** *wfG-elim2 assms* **by** *auto*
qed
thus *?thesis* **using** *append-g.simps* **by** *auto*
qed

lemma *wfC-refl*:
assumes *wfG* Θ $\mathcal{B}$ ((x, b', c') $\#_\Gamma\Gamma$)
shows *wfC* Θ $\mathcal{B}$ ((x, b', c') $\#_\Gamma\Gamma$) c'
using *wfG-wfC assms wfC-replace-cons* **by** *auto*

lemma *wfG-wfC-inside*:
assumes (x, b, c) $\in$ *toSet G* **and** *wfG* Θ *B G*
shows *wfC* Θ *B G c*
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* x' b' c' Γ')
then consider (*hd*) (x, b, c) = (x',b',c') | (*tail*) (x, b, c) $\in$ *toSet* Γ' **using** *toSet.simps* **by** *auto*
then show *?case* **proof**(*cases*)
case *hd*
then show *?thesis* **using** *GCons wf-weakening*
by (*metis wfC-replace-cons wfG-cons-wfC*)
next
case *tail*
then show *?thesis* **using** *GCons wf-weakening*
by (*metis insert-iff insert-is-Un subsetI toSet.simps*(*2*) *wfG-cons2*)
qed
qed

lemma *wfT-wf-cons3*:
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ z : b | c ⦄ **and** *atom y* $\sharp$ (c,Γ)
shows Θ; $\mathcal{B}$ $\vdash_{wf}$ (y, b, $c[z{::=}V\text{-}var\ y]_{cv}$) $\#_\Gamma$ Γ
proof −
have ⦃ z : b | c ⦄ = ⦃ y : b | (y $\leftrightarrow$ z) $\cdot$ c ⦄ **using** *type-eq-flip assms* **by** *auto*
moreover hence (y $\leftrightarrow$ z) $\cdot$ c = $c[z{::=}V\text{-}var\ y]_{cv}$ **using** *assms subst-v-c-def* **by** *auto*
ultimately have ⦃ z : b | c ⦄ = ⦃ y : b | $c[z{::=}V\text{-}var\ y]_{cv}$ ⦄ **by** *metis*
thus *?thesis* **using** *assms wfT-wf-cons*[*of* Θ $\mathcal{B}$ Γ *y b*] *fresh-Pair* **by** *metis*
qed

lemma *wfT-wfC-cons*:
assumes *wfT P* $\mathcal{B}$ Γ ⦃ *z1* : b | *c1* ⦄ **and** *wfT P* $\mathcal{B}$ Γ ⦃ *z2* : b | *c2* ⦄ **and** *atom x* $\sharp$ (*c1*,*c2*,Γ)
shows *wfC P* $\mathcal{B}$ ((x,b,$c1[z1{::=}V\text{-}var\ x]_v$) $\#_\Gamma\Gamma$) ($c2[z2{::=}V\text{-}var\ x]_v$) (**is** *wfC P* $\mathcal{B}$ *?G ?c*)
proof −
have *eq*: ⦃ *z2* : b | *c2* ⦄ = ⦃ x : b | $c2[z2{::=}V\text{-}var\ x]_{cv}$ ⦄ **using** *type-eq-subst assms fresh-prod3* **by** *simp*
have *eq2*: ⦃ *z1* : b | *c1* ⦄ = ⦃ x : b | $c1[z1{::=}V\text{-}var\ x]_{cv}$ ⦄ **using** *type-eq-subst assms fresh-prod3* **by** *simp*
moreover have *wfT P* $\mathcal{B}$ Γ ⦃ x : b | $c1[z1{::=}V\text{-}var\ x]_{cv}$ ⦄ **using** *assms eq2* **by** *auto*
moreover hence *wfG P* $\mathcal{B}$ ((x,b,$c1[z1{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma\Gamma$) **using** *wfT-wf-cons fresh-prod3 assms* **by** *auto*
moreover have *wfT P* $\mathcal{B}$ Γ ⦃ x : b | $c2[z2{::=}V\text{-}var\ x]_{cv}$ ⦄ **using** *assms eq* **by** *auto*

moreover hence *wfC P* $\mathcal{B}$ $((x,b,TRUE)\ \#_\Gamma\Gamma)$ $(c2[z2::=V\text{-}var\ x]_{cv})$ **using** *wfT-wfC assms fresh-prod3* **by** *simp*
ultimately show *?thesis* **using** *wfC-replace-cons subst-v-c-def* **by** *simp*
qed

lemma *wfT-wfC2*:
fixes $c::c$ **and** $x::x$
assumes $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \{\!|\ z : b \mid c\ |\!\}$ **and** *atom* $x \sharp \Gamma$
shows $\Theta;\ \mathcal{B};\ (x,b,TRUE)\#_\Gamma\Gamma \vdash_{wf}\ c[z::=[x]^v]_v$
proof(*cases x=z*)
case *True*
then show *?thesis* **using** *wfT-wfC assms* **by** *auto*
next
case *False*
hence *atom* $x \sharp c$ **using** *wfT-fresh-c assms* **by** *metis*
hence $\{\!|\ x : b\ \mid c[z::=[\ x\ ]^v]_v\ |\!\} = \{\!|\ z : b \mid c\ |\!\}$
using τ*.eq-iff Abs1-eq-iff(3)[of x* $c[z::=[\ x\ ]^v]_v\ z\ c$*]*
by (*metis flip-subst-v type-eq-flip*)
hence $\Theta;\ \mathcal{B};\ \Gamma\ \ \vdash_{wf} \{\!|\ x : b\ \mid c[z::=[\ x\ ]^v]_v\ |\!\}$ **using** *assms* **by** *metis*
thus *?thesis* **using** *wfT-wfC assms* **by** *auto*
qed

lemma *wfT-wfG*:
fixes $x::x$ **and** $\Gamma::\Gamma$ **and** $z::x$ **and** $c::c$ **and** $b::b$
assumes $\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \{\!|\ z : b \mid c\ |\!\}$ **and** *atom* $x \sharp \Gamma$
shows $\Theta;\ \mathcal{B}\ \vdash_{wf} (x,b,\ c[z::=[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma$
proof −
have $\Theta;\ \mathcal{B};\ (x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma\ \ \vdash_{wf}\ c[z::=[\ x\ ]^v]_v$ **using** *wfT-wfC2 assms* **by** *metis*
thus *?thesis* **using** *wfG-consI assms wfT-wfB b-of.simps wfX-wfY* **by** *metis*
qed

lemma *wfG-replace-inside2*:
fixes $\Gamma::\Gamma$
assumes *wfG P* $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c')\ \ \#_\Gamma\ \Gamma)$ **and** *wfG P* $\mathcal{B}$ $((x,b,c)\ \#_\Gamma\Gamma)$
shows *wfG P* $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)$
proof −
have *wfC P* $\mathcal{B}$ $((x,b,TRUE)\ \#_\Gamma\Gamma)$ c **using** *wfG-wfC assms* **by** *auto*
thus *?thesis* **using** *wf-replace-inside1(3)[OF assms(1)]* **by** *auto*
qed

lemma *wfG-replace-inside-full*:
fixes $\Gamma::\Gamma$
assumes *wfG P* $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c')\ \ \#_\Gamma\ \Gamma)$ **and** *wfG P* $\mathcal{B}$ $(\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$
shows *wfG P* $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)$
proof −
have *wfG P* $\mathcal{B}$ $((x,b,c)\ \#_\Gamma\Gamma)$ **using** *wfG-suffix assms* **by** *auto*
thus *?thesis* **using** *wfG-replace-inside assms* **by** *auto*
qed

lemma *wfT-replace-inside2*:
assumes *wfT* Θ $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c')\ \ \#_\Gamma\ \Gamma)$ t **and** *wfG* Θ $\mathcal{B}$ $(\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$
shows *wfT* Θ $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)$ t

proof −
  have *wfG* Θ *B* $(((x,b,c)\ \#_\Gamma\Gamma))$ **using** *wfG-suffix assms* **by** *auto*
  hence *wfC* Θ *B* $((x,b,TRUE)\ \#_\Gamma\Gamma)$ *c* **using** *wfG-wfC* **by** *auto*
  thus *?thesis* **using** *wf-replace-inside assms* **by** *metis*
qed

lemma *wfD-unique*:
  assumes *wfD P B* Γ Δ **and** $(u,\tau') \in$ *setD* Δ **and** $(u,\tau) \in$ *setD* Δ
  shows $\tau'=\tau$
using *assms* **proof**(*induct* Δ *rule*: Δ*-induct*)
  case *DNil*
  then show *?case* **by** *auto*
next
  case (*DCons u′ t′ D*)
  hence ∗: *wfD P B* Γ $((u',t')\ \#_\Delta\ D)$ **using** *Cons* **by** *auto*
  show *?case* **proof**(*cases* $u=u'$)
    case *True*
    then have $u \notin$ *fst ‘ setD D* **using** *wfD-elims* ∗ **by** *blast*
    then show *?thesis* **using** *DCons* **by** *force*
  next
    case *False*
    then show *?thesis* **using** *DCons wfD-elims* ∗ **by** (*metis fst-conv setD-ConsD*)
  qed
qed

lemma *replace-in-g-forget*:
  fixes *x*::*x*
  assumes *wfG P B G*
  shows *atom x* ∉ *atom-dom G* $\Longrightarrow$ $(G[x\longmapsto c]) = G$ **and**
  atom x ♯ *G* $\Longrightarrow$ $(G[x\longmapsto c]) = G$
proof −
  show *atom x* ∉ *atom-dom G* $\Longrightarrow$ $G[x\longmapsto c] = G$ **by** (*induct G rule*: Γ*-induct*,*auto*)
  thus *atom x* ♯ *G* $\Longrightarrow$ $(G[x\longmapsto c]) = G$ **using** *wfG-x-fresh assms* **by** *simp*
qed

lemma *replace-in-g-fresh-single*:
  fixes *G*::Γ **and** *x*::*x*
  assumes ‹Θ; *B* $\vdash_{wf}$ $G[x'\longmapsto c'']$› **and** *atom x* ♯ *G* **and** ‹Θ; *B* $\vdash_{wf}$ *G* ›
  shows *atom x* ♯ $G[x'\longmapsto c'']$
  using *rig-dom-eq wfG-dom-supp assms fresh-def atom-dom.simps dom.simps* **by** *metis*

8.17 Preservation of well-formedness under substitution

lemma *wfC-cons-switch*:
  fixes *c*::*c* **and** *c′*::*c*
  assumes Θ; *B*; $(x,\ b,\ c)\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c'$
  shows Θ; *B*; $(x,\ b,\ c')\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c$
proof −
  have ∗:Θ; *B* $\vdash_{wf}$ $(x,\ b,\ c)\ \#_\Gamma\ \Gamma$ **using** *wfC-wf assms* **by** *auto*
  hence *atom x* ♯ Γ ∧ *wfG* Θ *B* Γ ∧ Θ; *B* $\vdash_{wf}$ *b* **using** *wfG-cons* **by** *auto*
  hence Θ; *B*; $(x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ TRUE$ **using** *wfC-trueI wfG-cons2I* **by** *simp*
  hence Θ; *B*;$(x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c'$

using *wf-replace-inside1* (*2*)[*of* Θ $\mathcal{B}$ (*x*, *b*, *c*) $\#_\Gamma$ Γ *c'* *GNil x b c* Γ *TRUE*] *assms* **by** *auto*
hence *wfG* Θ $\mathcal{B}$ ((*x*,*b*,*c'*) $\#_\Gamma\Gamma$) **using** *wf-replace-inside1* (*3*)[*OF* ∗, *of GNil x b c* Γ *c'*] **by** *auto*
moreover have *wfC* Θ $\mathcal{B}$ ((*x*,*b*,*TRUE*) $\#_\Gamma\Gamma$) *c* **proof**(*cases c* ∈ { *TRUE*, *FALSE* })
case *True*
have Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ∧ *atom x* ♯ Γ ∧ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *wfG-elims*(*2*)[*OF* ∗] **by** *auto*
hence Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*,*b*,*TRUE*) $\#_\Gamma$ Γ **using** *wfG-cons-TRUE* **by** *auto*
then show *?thesis* **using** *wfC-trueI wfC-falseI True* **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-elims*(*2*)[*OF* ∗] **by** *auto*
qed
ultimately show *?thesis* **using** *wfC-replace-cons* **by** *auto*
qed

lemma *subst-g-inside-simple*:
fixes Γ_1::Γ **and** Γ_2::Γ
assumes *wfG P* $\mathcal{B}$ (Γ_1@((*x*,*b*,*c*) $\#_\Gamma\Gamma_2$))
shows (Γ_1@((*x*,*b*,*c*) $\#_\Gamma\Gamma_2$))[*x*::=*v*]$_{\Gamma v}$ = Γ_1[*x*::=*v*]$_{\Gamma v}$@Γ_2
using *assms* **proof**(*induct* Γ_1 *rule*: Γ*-induct*)
case *GNil*
then show *?case* **using** *subst-gv.simps* **by** *simp*
next
case (*GCons x' b' c' G*)
hence ∗:*P*; $\mathcal{B}$ $\vdash_{wf}$ (*x'*, *b'*, *c'*) $\#_\Gamma$ (*G* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ_2) **by** *auto*
hence *x*≠*x'*
using *GCons append-Cons wfG-cons-fresh2*[*OF* ∗] **by** *auto*
hence ((*GCons* (*x'*, *b'*, *c'*) *G*) @ (*GCons* (*x*, *b*, *c*) Γ_2))[*x*::=*v*]$_{\Gamma v}$ =
(*GCons* (*x'*, *b'*, *c'*) (*G* @ (*GCons* (*x*, *b*, *c*) Γ_2)))[*x*::=*v*]$_{\Gamma v}$ **by** *auto*
also have ... = *GCons* (*x'*, *b'*, *c'*[*x*::=*v*]$_{cv}$) ((*G* @ (*GCons* (*x*, *b*, *c*) Γ_2))[*x*::=*v*]$_{\Gamma v}$)
using *subst-gv.simps* ‹*x*≠*x'*› **by** *simp*
also have ... = (*x'*, *b'*, *c'*[*x*::=*v*]$_{cv}$) $\#_\Gamma$ (*G*[*x*::=*v*]$_{\Gamma v}$ @ Γ_2) **using** *GCons* ∗ *wfG-elims* **by** *metis*
also have ... = ((*x'*, *b'*, *c'*) $\#_\Gamma$ *G*)[*x*::=*v*]$_{\Gamma v}$ @ Γ_2 **using** *subst-gv.simps* ‹*x*≠*x'*› **by** *simp*
finally show *?case* **by** *blast*
qed

lemma *subst-c-TRUE-FALSE*:
fixes *c*::*c*
assumes *c* ∉ {*TRUE*,*FALSE*}
shows *c*[*x*::=*v*]$_{cv}$ ∉ {*TRUE*, *FALSE*}
using *assms* **by**(*nominal-induct c rule*:*c.strong-induct*,*auto simp add*: *subst-cv.simps*)

lemma *lookup-subst*:
assumes *Some* (*b*, *c*) = *lookup* Γ *x* **and** *x* ≠ *x'*
shows ∃ *c'*. *Some* (*b*,*c'*) = *lookup* Γ[*x'*::=*v*]$_{\Gamma v}$ *x*
using *assms* **proof**(*induct* Γ *rule*: Γ*-induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
then show *?case* **proof**(*cases x1*=*x'*)
case *True*
then show *?thesis* **using** *subst-gv.simps GCons* **by** *auto*

next
case *False*
hence $*$:((*x1*, *b1*, *c1*) $\#_\Gamma$ Γ*1*)[x'::=v]$_{\Gamma v}$ = ((*x1*, *b1*, *c1*[x'::=v]$_{cv}$) $\#_\Gamma$ Γ*1*[x'::=v]$_{\Gamma v}$) **using** *subst-gv.simps* **by** *auto*
then show *?thesis* **proof**(*cases x1*=*x*)
case *True*
then show *?thesis* **using** *lookup.simps* $*$
using *GCons.prems*(*1*) **by** *auto*
next
case *False*
then show *?thesis* **using** *lookup.simps* $*$
using *GCons.prems*(*1*) **by** (*simp add*: *GCons.hyps assms*(*2*))
qed
qed
qed

lemma *lookup-subst2*:
assumes *Some* (*b*, *c*) = *lookup* (Γ′@((x',b_1,*c0*[*z0*::=[x]^v]$_{cv}$)$\#_\Gamma$Γ)) x **and** $x \neq x'$ **and**
Θ; $\mathcal{B}$ $\vdash_{wf}$ (Γ′@((x',b_1,*c0*[*z0*::=[x]^v]$_{cv}$)$\#_\Gamma$Γ))
shows ∃ c'. *Some* (b,c') = *lookup* (Γ′[x'::=v]$_{\Gamma v}$@Γ) x
using *assms lookup-subst subst-g-inside* **by** *metis*

lemma *wf-subst1*:
fixes Γ::Γ **and** Γ′::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** b::b **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
shows *wfV-subst*: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ v : b ⟹ Γ=Γ_1@((x,b',c') $\#_\Gamma\Gamma_2$) ⟹ Θ; $\mathcal{B}$;Γ_2 $\vdash_{wf}$ v' : b' ⟹ Θ ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ v[x::=v']$_{vv}$: b **and**
wfC-subst: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ c ⟹ Γ=Γ_1@((x,b',c') $\#_\Gamma\Gamma_2$) ⟹ Θ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' ⟹ Θ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ c[x::=v']$_{cv}$ **and**
wfG-subst: Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ⟹ Γ=Γ_1@((x,b',c') $\#_\Gamma\Gamma_2$) ⟹ Θ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' ⟹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ[x::=v']$_{\Gamma v}$ **and**
wfT-subst: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ ⟹ Γ=Γ_1@((x,b',c') $\#_\Gamma\Gamma_2$) ⟹ Θ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' ⟹ Θ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ τ[x::=v']$_{\tau v}$ **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* ⟹ *True* **and**
$\vdash_{wf}$ Θ ⟹*True* **and**
Θ; $\mathcal{B}$ $\vdash_{wf}$ b ⟹ *True* **and**
wfCE-subst: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : b ⟹ Γ=Γ_1@((x,b',c') $\#_\Gamma\Gamma_2$) ⟹ Θ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' ⟹ Θ ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ *ce*[x::=v']$_{cev}$: b **and**
Θ $\vdash_{wf}$ *td* ⟹ *True*
proof(*nominal-induct*
b **and** c **and** Γ **and** τ **and** *ts* **and** Θ **and** b **and** b **and** *td*
avoiding: x v'
arbitrary: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b1 c1 x1*)

show *?case* **proof**(*cases x1*=*x*)
case *True*
hence (*V-var x1*)[x::=v']$_{vv}$ = v' **using** *subst-vv.simps* **by** *auto*
moreover have b' = *b1* **using** *wfV-varI True lookup-inside-wf*
by (*metis option.inject prod.inject*)

moreover have Θ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ v' : b' **using** *wfV-varI subst-g-inside-simple wf-weakening*

append-g-toSetU sup-ge2 wfV-wf **by** *metis*
ultimately show *?thesis* **by** *auto*
next
case *False*
hence (*V-var x1*)[x::=v']$_{vv}$ = (*V-var x1*) **using** *subst-vv.simps* **by** *auto*
moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ[x::=v']$_{\Gamma v}$ **using** *wfV-varI* **by** *simp*
moreover obtain *c1'* **where** *Some* (*b1*, *c1'*) = *lookup* Γ[x::=v']$_{\Gamma v}$ *x1* **using** *wfV-varI False lookup-subst* **by** *metis*
ultimately show *?thesis* **using** *Wellformed.wfV-varI*[*of* Θ $\mathcal{B}$ Γ[x::=v']$_{\Gamma v}$ *b1 c1' x1*] **by** *metis*
qed
next
case (*wfV-litI* Θ Γ *l*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfV-pairI* Θ Γ *v1 b1 v2 b2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfV-consI s dclist* Θ *dc x b c* Γ *v*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfV-conspI s bv dclist* Θ *dc x' b' c* $\mathcal{B}$ *b* Γ *va*)
show *?case* **unfolding** *subst-vv.simps* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **and** ‹(*dc*, ⦃ *x'* : *b'* | *c* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ;$\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfV-conspI* **by** *auto*
have *atom bv* ♯ Γ[x::=v']$_{\Gamma v}$ **using** *fresh-subst-gv-if wfV-conspI* **by** *metis*
moreover have *atom bv* ♯ *va*[x::=v']$_{vv}$ **using** *wfV-conspI fresh-subst-if* **by** *simp*
ultimately show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ[x::=v']$_{\Gamma v}$, *b*, *va*[x::=v']$_{vv}$)› **unfolding** *fresh-prodN* **using** *wfV-conspI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ *va*[x::=v']$_{vv}$: *b'*[*bv*::=*b*]$_{bb}$ › **using** *wfV-conspI* **by** *auto*
qed
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
have Θ; $\mathcal{B}$; Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ ⦃ *z* : *b* | *c*[x::=v']$_{cv}$ ⦄ **proof**
have ‹Θ; $\mathcal{B}$; ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ)[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ *c*[x::=v']$_{cv}$ ›
proof(*rule wfTI*(*9*))
show ‹(*z*, *b*, *TRUE*) $\#_\Gamma$ Γ = ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1) @ (*x*, *b'*, *c'*) $\#_\Gamma$ Γ_2› **using** *wfTI append-g.simps* **by** *simp*
show ‹ Θ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ *v'* : *b'* › **using** *wfTI* **by** *auto*
qed
thus ∗:‹Θ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ[x::=v']$_{\Gamma v}$ $\vdash_{wf}$ *c*[x::=v']$_{cv}$ ›
using *subst-gv.simps subst-cv.simps wfTI fresh-x-neq* **by** *auto*

have *atom z* ♯ Γ[x::=v']$_{\Gamma v}$ **using** *fresh-subst-gv-if wfTI* **by** *metis*
moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ[x::=v']$_{\Gamma v}$ **using** *wfTI wfX-wfY wfG-elims subst-gv.simps* ∗ **by** *metis*
ultimately show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ[x::=v']$_{\Gamma v}$)› **using** *wfG-fresh-x* **by** *metis*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfTI* **by** *auto*
qed
thus *?case* **using** *subst-tv.simps wfTI* **by** *auto*
next

case (*wfC-trueI* Θ Γ)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-falseI* Θ Γ)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
show *?case* **proof**(*subst subst-cv.simps,rule*)
show Θ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ *e1*$[x::=v']_{cev}$: *b* **using** *wfC-eqI subst-dv.simps* **by** *auto*
show Θ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ *e2*$[x::=v']_{cev}$: *b* **using** *wfC-eqI* **by** *auto*
qed
next
case (*wfC-conjI* Θ Γ *c1 c2*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-disjI* Θ Γ *c1 c2*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-notI* Θ Γ *c1*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-impI* Θ Γ *c1 c2*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfG-nilI* Θ)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfG-cons1I c* Θ $\mathcal{B}$ Γ *y b*)

show *?case* **proof**(*cases x=y*)
case *True*
hence $((y, b, c) \#_\Gamma \Gamma)[x::=v']_{\Gamma v} = \Gamma$ **using** *subst-gv.simps* **by** *auto*
moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ **using** *wfG-cons1I* **by** *auto*
ultimately show *?thesis* **by** *auto*
next
case *False*
have $\Gamma_1 \neq$ *GNil* **using** *wfG-cons1I False* **by** *auto*
then obtain *G* **where** $\Gamma_1 = (y, b, c) \#_\Gamma G$ **using** *GCons-eq-append-conv wfG-cons1I* **by** *auto*
hence $*{:}\Gamma = G$ @ $(x, b', c') \#_\Gamma \Gamma_2$ **using** *wfG-cons1I* **by** *auto*
hence $((y, b, c) \#_\Gamma \Gamma)[x::=v']_{\Gamma v} = (y, b, c[x::=v']_{cv}) \#_\Gamma \Gamma[x::=v']_{\Gamma v}$ **using** *subst-gv.simps False* **by** *auto*
moreover have Θ; $\mathcal{B}$ $\vdash_{wf}$ $(y, b, c[x::=v']_{cv}) \#_\Gamma \Gamma[x::=v']_{\Gamma v}$ **proof**(*rule Wellformed.wfG-cons1I*)
show ‹$c[x::=v']_{cv} \notin \{TRUE, FALSE\}$› **using** *wfG-cons1I subst-c-TRUE-FALSE* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ $\Gamma[x::=v']_{\Gamma v}$ › **using** *wfG-cons1I* ∗ **by** *auto*
have $\Gamma = (G$ @ $((x, b', c') \#_\Gamma GNil))$ @ Γ_2 **using** ∗ *append-g-assoc* **by** *auto*
hence *atom y* ♯ Γ_2 **using** *fresh-suffix* ‹*atom y* ♯ Γ› **by** *auto*
hence *atom y* ♯ v' **using** *wfG-cons1I wfV-x-fresh* **by** *metis*
thus ‹*atom y* ♯ $\Gamma[x::=v']_{\Gamma v}$› **using** *fresh-subst-gv wfG-cons1I* **by** *auto*
have $((y, b, TRUE) \#_\Gamma \Gamma)[x::=v']_{\Gamma v} = (y, b, TRUE) \#_\Gamma \Gamma[x::=v']_{\Gamma v}$ **using** *subst-gv.simps subst-cv.simps False* **by** *auto*
thus ‹ Θ; $\mathcal{B}$; $(y, b, TRUE) \#_\Gamma \Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $c[x::=v']_{cv}$ › **using** *wfG-cons1I(6)*[*of* $(y,b,TRUE) \#_\Gamma G$] ∗ *subst-gv.simps*

wfG-cons1I **by** *fastforce*
show Θ; $\mathcal{B} \vdash_{wf}$ *b* **using** *wfG-cons1I* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*wfG-cons2I c* Θ $\mathcal{B}$ Γ *y b*)
show *?case* **proof**(*cases x=y*)
case *True*
hence $((y,\ b,\ c)\ \#_\Gamma\ \Gamma)[x::=v']_{\Gamma v} = \Gamma$ **using** *subst-gv.simps* **by** *auto*
moreover have Θ; $\mathcal{B} \vdash_{wf} \Gamma$ **using** *wfG-cons2I* **by** *auto*
ultimately show *?thesis* **by** *auto*
next
case *False*
have $\Gamma_1 \neq$ *GNil* **using** *wfG-cons2I False* **by** *auto*
then obtain *G* **where** $\Gamma_1 = (y,\ b,\ c)\ \#_\Gamma\ G$ **using** *GCons-eq-append-conv wfG-cons2I* **by** *auto*
hence $*{:}\Gamma = G\ @\ (x,\ b',\ c')\ \#_\Gamma\ \Gamma_2$ **using** *wfG-cons2I* **by** *auto*
hence $((y,\ b,\ c)\ \#_\Gamma\ \Gamma)[x::=v']_{\Gamma v} =(y,\ b,\ c[x::=v']_{cv})\ \#_\Gamma \Gamma[x::=v']_{\Gamma v}$ **using** *subst-gv.simps False* **by** *auto*
moreover have Θ; $\mathcal{B} \vdash_{wf} (y,\ b,\ c[x::=v']_{cv})\ \#_\Gamma \Gamma[x::=v']_{\Gamma v}$ **proof**(*rule Wellformed.wfG-cons2I*)
show ‹$c[x::=v']_{cv} \in \{TRUE,\ FALSE\}$› **using** *subst-cv.simps wfG-cons2I* **by** *auto*
show ‹ Θ; $\mathcal{B} \vdash_{wf} \Gamma[x::=v']_{\Gamma v}$ › **using** *wfG-cons2I* $*$ **by** *auto*
have $\Gamma = (G\ @\ ((x,\ b',\ c')\ \#_\Gamma GNil))\ @\ \Gamma_2$ **using** $*$ *append-g-assoc* **by** *auto*
hence *atom y* $\sharp$ Γ_2 **using** *fresh-suffix wfG-cons2I* **by** *metis*
hence *atom y* $\sharp$ v' **using** *wfG-cons2I wfV-x-fresh* **by** *metis*
thus ‹*atom y* $\sharp$ $\Gamma[x::=v']_{\Gamma v}$› **using** *fresh-subst-gv wfG-cons2I* **by** *auto*
show Θ; $\mathcal{B} \vdash_{wf}$ *b* **using** *wfG-cons2I* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-eqI* Θ $\mathcal{B}$ Γ *v1 b v2*)
then show *?case* **unfolding** *subst-cev.simps*
using *Wellformed.wfCE-eqI* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *Wellformed.wfCE-fstI subst-cev.simps* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *subst-cev.simps wf-intros* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*

next
  case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
  then show *?case* **using** *subst-vv.simps* *wf-intros* **by** *auto*
qed(*metis subst-sv.simps wf-intros*)+

lemma *wf-subst2*:
  fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
  shows $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} e : b \implies \Gamma=\Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \implies \Theta; \mathcal{B} ; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v']_{\Gamma v} ; \Delta[x::=v']_{\Delta v} \vdash_{wf} e[x::=v']_{ev} : b$ **and**
  $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} s : b \implies \Gamma=\Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \implies \Theta ;\mathcal{B} ; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v']_{\Gamma v} ; \Delta[x::=v']_{\Delta v} \vdash_{wf} s[x::=v']_{sv} : b$ **and**
  $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \implies \Gamma=\Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta; \Phi; \mathcal{B}; \Gamma[x::=v']_{\Gamma v} ; \Delta[x::=v']_{\Delta v} ; tid ; dc ; t \vdash_{wf}$ *subst-branchv cs x v′* : *b* **and**
  $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \implies \Gamma=\Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta; \Phi; \mathcal{B}; \Gamma[x::=v']_{\Gamma v} ; \Delta[x::=v']_{\Delta v} ; tid ; dclist \vdash_{wf}$ *subst-branchlv css x v′* : *b* **and**
  $\Theta \vdash_{wf} (\Phi::\Phi) \implies$ *True* **and**
  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Gamma=\Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \implies \Theta; \mathcal{B} ; \Gamma_2 \vdash_{wf} v' : b' \implies \Theta ; \mathcal{B} ; \Gamma[x::=v']_{\Gamma v} \vdash_{wf} \Delta[x::=v']_{\Delta v}$ **and**
  $\Theta ; \Phi \vdash_{wf} ftq \implies$ *True* **and**
  $\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \implies$ *True*
proof(*nominal-induct*
    b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
    avoiding: *x v′*
    arbitrary: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
    rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
  case (*wfE-valI* Θ Γ *v b*)
  then show *?case* **using** *subst-vv.simps* *wf-intros wf-subst1*
    by (*metis subst-ev.simps*(*1*))
next
  case (*wfE-plusI* Θ Γ *v1 v2*)
  then show *?case* **using** *subst-vv.simps* *wf-intros wf-subst1* **by** *auto*
next
  case (*wfE-leqI* Θ Φ Γ Δ *v1 v2*)
  then show *?case*
    using *subst-vv.simps* *subst-ev.simps subst-ev.simps wf-subst1 Wellformed.wfE-leqI*
    by *auto*
next
  case (*wfE-eqI* Θ Φ Γ Δ *v1 b v2*)
  then show *?case*
    using *subst-vv.simps* *subst-ev.simps subst-ev.simps wf-subst1 Wellformed.wfE-eqI*
  proof −
    show *?thesis*
    by (*metis* (*no-types*) *subst-ev.simps*(*4*) *wfE-eqI.hyps*(*1*) *wfE-eqI.hyps*(*4*) *wfE-eqI.hyps*(*5*) *wfE-eqI.hyps*(*6*) *wfE-eqI.hyps*(*7*) *wfE-eqI.prems*(*1*) *wfE-eqI.prems*(*2*) *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-eqI* *wfV-subst*)
  qed
next
  case (*wfE-fstI* Θ Γ *v1 b1 b2*)
  then show *?case* **using** *subst-vv.simps subst-ev.simps wf-subst1 Wellformed.wfE-fstI*
  proof −

show *?thesis*
by (*metis* (*full-types*) *subst-ev.simps*(*5*) *wfE-fstI.hyps*(*1*) *wfE-fstI.hyps*(*4*) *wfE-fstI.hyps*(*5*) *wfE-fstI.prems*(*1*) *wfE-fstI.prems*(*2*) *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-fstI wf-subst1*(*1*))
qed
next
case (*wfE-sndI* Θ Γ *v1 b1 b2*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-sndI wf-subst1*(*1*))
next
case (*wfE-concatI* Θ Φ Γ Δ *v1 v2*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-concatI wf-subst1*(*1*))
next
case (*wfE-splitI* Θ Φ Γ Δ *v1 v2*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-splitI wf-subst1*(*1*))
next
case (*wfE-lenI* Θ Φ Γ Δ *v1*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-lenI wf-subst1*(*1*))
next
case (*wfE-appI* Θ Φ Γ Δ *f x b c* τ *s′ v*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-appI wf-subst1*(*1*))
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b′ bv1 v1* τ*1 f1 x1 b1 c1 s1*)
show *?case* **proof**(*subst subst-ev.simps*, *rule*)
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-appPI wfX-wfY* **by** *metis*
show $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$ **using** *wfE-appPI* **by** *auto*
show *Some* (*AF-fundef f1* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s1*))) = *lookup-fun* Φ *f1* **using** *wfE-appPI* **by** *auto*
show $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} v1[x::=v]_{vv} : b1[bv1::=b']_b$ **using** *wfE-appPI wf-subst1* **by** *auto*
show $\Theta; \mathcal{B} \vdash_{wf} b'$ **using** *wfE-appPI* **by** *auto*
have *atom bv1* ♯ $\Gamma[x::=v]_{\Gamma v}$ **using** *fresh-subst-gv-if wfE-appPI* **by** *metis*
moreover have *atom bv1* ♯ $v1[x::=v]_{vv}$ **using** *wfE-appPI fresh-subst-if* **by** *simp*
moreover have *atom bv1* ♯ $\Delta[x::=v]_{\Delta v}$ **using** *wfE-appPI fresh-subst-dv-if* **by** *simp*
ultimately show *atom bv1* ♯ $(\Phi, \Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, b', v1[x::=v]_{vv}, (b\text{-}of\ \tau 1)[bv1::=b']_b)$

using *wfE-appPI fresh-prodN* **by** *metis*
qed
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
have $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash_{wf} (AE\text{-}mvar\ u) : b\text{-}of\ \tau[x::=v]_{\tau v}$ **proof**
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-mvarI* **by** *auto*
show $\Theta; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$ **using** *wfE-mvarI* **by** *auto*
show $(u, \tau[x::=v]_{\tau v}) \in setD\ \Delta[x::=v]_{\Delta v}$ **using** *wfE-mvarI subst-dv-member* **by** *auto*
qed
thus *?case* **using** *subst-ev.simps b-of-subst* **by** *auto*
next
case (*wfD-emptyI* Θ Γ)
then show *?case* **using** *subst-dv.simps wf-intros wf-subst1* **by** *auto*
next

case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)
moreover hence $u \notin fst$ ' *setD* $\Delta[x::=v]_{\Delta v}$ **using** *subst-dv.simps subst-dv-iff* **using** *subst-dv-fst-eq* **by** *presburger*
ultimately show *?case* **using** *subst-dv.simps Wellformed.wfD-cons wf-subst1* **by** *auto*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **by** *auto*
next
case (*wfPhi-consI f* Θ Φ *ft*)
then show *?case* **by** *auto*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x2 c* Γ Δ *s b*)
show *?case* **unfolding** *subst-sv.simps* **proof**
show ‹ Θ; Φ; $\mathcal{B}$; (*x2*, *B-bool*, $c[x::=v]_{cv}$) $\#_\Gamma$ $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ $s[x::=v]_{sv}$: *b* ›
using *wfS-assertI*(*4*)[*of* (*x2*, *B-bool*, *c*) $\#_\Gamma$ Γ_1 *x*] *wfS-assertI* **by** *auto*

show ‹ Θ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash_{wf}$ $c[x::=v]_{cv}$ › **using** *wfS-assertI wf-subst1* **by** *auto*
show ‹ Θ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfS-assertI wf-subst1* **by** *auto*
show ‹*atom x2* ♯ (Φ, Θ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $c[x::=v]_{cv}$, *b*, $s[x::=v]_{sv}$)›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-assertI*)+
apply(*metis fresh-subst-gv-if wfS-assertI*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-assertI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-e-def wfS-assertI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-τ-def wfS-assertI*)
by(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-s-def wfS-assertI*)
qed
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b1 y s b2*)
have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ *LET y* = ($e[x::=v]_{ev}$) *IN* ($s[x::=v]_{sv}$) : *b2*
proof
show ‹ Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ $e[x::=v]_{ev}$: *b1* › **using** *wfS-letI* **by** *auto*
have ‹ Θ ; Φ ; $\mathcal{B}$; ((*y*, *b1*, *TRUE*) $\#_\Gamma$ Γ)$[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ $s[x::=v]_{sv}$: *b2* ›
using *wfS-letI*(*6*) *wfS-letI append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*y*, *b1*, *TRUE*) $\#_\Gamma$ $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ $s[x::=v]_{sv}$: *b2* ›
using *wfS-letI subst-gv.simps* **by** *auto*
show ‹ Θ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfS-letI* **by** *auto*
show ‹*atom y* ♯ (Φ, Θ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $e[x::=v]_{ev}$, *b2*)›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-letI*)+
apply(*metis fresh-subst-gv-if wfS-letI*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-letI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-e-def wfS-letI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-τ-def wfS-letI*)
done
qed
thus *?case* **using** *subst-sv.simps wfS-letI* **by** *auto*
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *y s2 b*)
have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ *LET y* : $\tau[x::=v]_{\tau v}$ = ($s1[x::=v]_{sv}$) *IN* ($s2[x::=v]_{sv}$) : *b*
proof

show ‹ Θ ; Φ ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$; Δ[x::=v]$_{\Delta v}$ $\vdash_{wf}$ *s1*[x::=v]$_{sv}$: *b-of* (τ[x::=v]$_{\tau v}$) › **using** *wfS-let2I b-of-subst* **by** *simp*
have ‹ Θ ; Φ ; $\mathcal{B}$; ((y, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ)[x::=v]$_{\Gamma v}$; Δ[x::=v]$_{\Delta v}$ $\vdash_{wf}$ *s2*[x::=v]$_{sv}$: b ›
using *wfS-let2I append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (y, *b-of* τ[x::=v]$_{\tau v}$, *TRUE*) $\#_\Gamma$ Γ[x::=v]$_{\Gamma v}$; Δ[x::=v]$_{\Delta v}$ $\vdash_{wf}$ *s2*[x::=v]$_{sv}$: b ›
using *wfS-let2I subst-gv.simps append-g.simps* **using** *b-of-subst* **by** *simp*
show ‹ Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ τ[x::=v]$_{\tau v}$ › **using** *wfS-let2I wf-subst1* **by** *metis*
show ‹*atom y* ♯ (Φ, Θ, $\mathcal{B}$, Γ[x::=v]$_{\Gamma v}$, Δ[x::=v]$_{\Delta v}$, *s1*[x::=v]$_{sv}$, b, τ[x::=v]$_{\tau v}$)›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-let2I*)+
apply(*metis fresh-subst-gv-if wfS-let2I*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-let2I*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-e-def wfS-let2I*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-τ-def wfS-let2I*)+
done
qed
thus *?case* **using** *subst-sv.simps*(*3*) *subst-tv.simps wfS-let2I* **by** *auto*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ v u Φ Δ b s)
show *?case* **proof**(*subst subst-sv.simps*, *auto simp add*: *u-fresh-xv*,*rule*)
show ‹ Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ τ[x::=v]$_{\tau v}$ › **using** *wfS-varI wf-subst1* **by** *auto*
have *b-of* (τ[x::=v]$_{\tau v}$) = *b-of* τ **using** *b-of-subst* **by** *auto*
thus ‹ Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ v[x::=v]$_{vv}$: *b-of* τ[x::=v]$_{\tau v}$ › **using** *wfS-varI wf-subst1* **by** *auto*
have ∗:*atom u* ♯ v′ **using** *wfV-supp wfS-varI fresh-def* **by** *metis*
show ‹*atom u* ♯ (Φ, Θ, $\mathcal{B}$, Γ[x::=v]$_{\Gamma v}$, Δ[x::=v]$_{\Delta v}$, τ[x::=v]$_{\tau v}$, v[x::=v]$_{vv}$, b)›
unfolding *fresh-prodN* **apply**(*auto simp add*: *wfS-varI*)
using *wfS-varI fresh-subst-gv* ∗ *fresh-subst-dv* **by** *metis*+
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$; (u, τ[x::=v]$_{\tau v}$) $\#_\Delta$ Δ[x::=v]$_{\Delta v}$ $\vdash_{wf}$ s[x::=v]$_{sv}$: b › **using** *wfS-varI* **by** *auto*
qed
next
case (*wfS-assignI* u τ Δ Θ $\mathcal{B}$ Γ Φ v)
show *?case* **proof**(*subst subst-sv.simps*, *rule wf-intros*)
show ‹(u, τ[x::=v]$_{\tau v}$) ∈ *setD* Δ[x::=v]$_{\Delta v}$› **using** *subst-dv-iff wfS-assignI* **using** *subst-dv-fst-eq*
using *subst-dv-member* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ Δ[x::=v]$_{\Delta v}$ › **using** *wfS-assignI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ v[x::=v]$_{vv}$: *b-of* τ[x::=v]$_{\tau v}$ › **using** *wfS-assignI b-of-subst wf-subst1* **by** *auto*
show Θ $\vdash_{wf}$ Φ **using** *wfS-assignI* **by** *auto*
qed
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ v *tid dclist* Δ Φ *cs* b)
show *?case* **proof**(*subst subst-sv.simps*, *rule wf-intros*)
show ‹ Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ v[x::=v]$_{vv}$: *B-id tid* › **using** *wfS-matchI wf-subst1* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ› **using** *wfS-matchI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$; Δ[x::=v]$_{\Delta v}$; *tid* ; *dclist* $\vdash_{wf}$ *subst-branchlv cs x v′* : b › **using** *wfS-matchI* **by** *simp*
show Θ; $\mathcal{B}$; Γ[x::=v]$_{\Gamma v}$ $\vdash_{wf}$ Δ[x::=v]$_{\Delta v}$ **using** *wfS-matchI* **by** *auto*
show Θ $\vdash_{wf}$ Φ **using** *wfS-matchI* **by** *auto*
qed
next

case (*wfS-branchI* Θ Φ ℬ *y* τ Γ Δ *s b tid dc*)
have Θ ; Φ ; ℬ ; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$; *tid* ; *dc* ; τ $\vdash_{wf}$ *dc y* ⇒ ($s[x::=v]_{sv}$) : *b*
proof
have ‹ Θ ; Φ ; ℬ ; ((*y*, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ)$[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ $s[x::=v]_{sv}$: *b* ›
using *wfS-branchI append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; ℬ ; (*y*, *b-of* τ, *TRUE*) $\#_\Gamma$ $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash_{wf}$ $s[x::=v]_{sv}$: *b* ›
using *subst-gv.simps b-of-subst wfS-branchI* **by** *simp*
show ‹*atom y* ♯ (Φ, Θ, ℬ, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $\Gamma[x::=v]_{\Gamma v}$, τ)›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-branchI*)+
apply(*metis fresh-subst-gv-if wfS-branchI*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-branchI*)
apply(*metis fresh-subst-gv-if wfS-branchI*)+
done
show ‹ Θ; ℬ; $\Gamma[x::=v]_{\Gamma v}$ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfS-branchI* **by** *auto*
qed
thus *?case* **using** *subst-branchv.simps wfS-branchI* **by** *auto*
next
case (*wfS-finalI* Θ Φ ℬ Γ Δ *tid dclist′ cs b dclist*)
then show *?case* **using** *subst-branchlv.simps wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ ℬ Γ Δ *tid dclist′ cs b css dclist*)
then show *?case* **using** *subst-branchlv.simps wf-intros* **by** *metis*

qed(*metis subst-sv.simps wf-subst1 wf-intros*)+

lemmas *wf-subst* = *wf-subst1 wf-subst2*

lemma *wfG-subst-wfV*:
assumes Θ; ℬ $\vdash_{wf}$ Γ′ @ (*x*, *b*, $c0[z0::=V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ **and** *wfV* Θ ℬ Γ *v b*
shows Θ; ℬ $\vdash_{wf}$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ
using *assms wf-subst subst-g-inside-simple* **by** *auto*

lemma *wfG-member-subst*:
assumes (*x1*,*b1*,*c1*) ∈ *toSet* (Γ′@Γ) **and** *wfG* Θ ℬ (Γ′@((*x*,*b*,*c*) $\#_\Gamma$Γ)) **and** *x* ≠ *x1*
shows ∃ *c1′*. (*x1*,*b1*,*c1′*) ∈ *toSet* (($\Gamma'[x::=v]_{\Gamma v}$)@Γ)
proof −
consider (*lhs*) (*x1*,*b1*,*c1*) ∈ *toSet* Γ′ | (*rhs*) (*x1*,*b1*,*c1*) ∈ *toSet* Γ **using** *append-g-toSetU assms* **by** *auto*
thus *?thesis* **proof**(*cases*)
case *lhs*
hence (*x1*,*b1*,$c1[x::=v]_{cv}$) ∈ *toSet* ($\Gamma'[x::=v]_{\Gamma v}$) **using** *wfG-inside-fresh*[*THEN subst-gv-member-iff*[*OF lhs*]] *assms* **by** *metis*
hence (*x1*,*b1*,$c1[x::=v]_{cv}$) ∈ *toSet* ($\Gamma'[x::=v]_{\Gamma v}$@Γ) **using** *append-g-toSetU* **by** *auto*
then show *?thesis* **by** *auto*
next
case *rhs*
hence (*x1*,*b1*,*c1*) ∈ *toSet* ($\Gamma'[x::=v]_{\Gamma v}$@Γ) **using** *append-g-toSetU* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

lemma *wfG-member-subst2*:
assumes $(x1,b1,c1) \in toSet\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $wfG\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $x \neq x1$
shows $\exists\ c1'.\ (x1,b1,c1') \in toSet\ ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$
proof −
consider (*lhs*) $(x1,b1,c1) \in toSet\ \Gamma'$ | (*rhs*) $(x1,b1,c1) \in toSet\ \Gamma$ **using** *append-g-toSetU assms* **by** *auto*
thus *?thesis* **proof**(*cases*)
case *lhs*
hence $(x1,b1,c1[x::=v]_{cv}) \in toSet\ (\Gamma'[x::=v]_{\Gamma v})$ **using** *wfG-inside-fresh*[*THEN subst-gv-member-iff*[*OF lhs*]] *assms* **by** *metis*
hence $(x1,b1,c1[x::=v]_{cv}) \in toSet\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)$ **using** *append-g-toSetU* **by** *auto*
then show *?thesis* **by** *auto*
next
case *rhs*
hence $(x1,b1,c1) \in toSet\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)$ **using** *append-g-toSetU* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

lemma *wbc-subst*:
fixes $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$
assumes $wfC\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b,c')\ \#_\Gamma\Gamma))\ c$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$
shows $\Theta; \mathcal{B}; ((\Gamma'[x::=v]_{\Gamma v})@\Gamma) \vdash_{wf} c[x::=v]_{cv}$
proof −
have $(\Gamma'@((x,b,c')\ \#_\Gamma\Gamma))[x::=v]_{\Gamma v} = ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ **using** *assms subst-g-inside-simple wfC-wf* **by** *metis*
thus *?thesis* **using** *wf-subst1*(*2*)[*OF assms*(*1*) - *assms*(*2*)] **by** *metis*
qed

lemma *wfG-inside-fresh-suffix*:
assumes $wfG\ P\ B\ (\Gamma'@(x,b,c)\ \#_\Gamma\Gamma)$
shows $atom\ x \sharp \Gamma$
proof −
have $wfG\ P\ B\ ((x,b,c)\ \#_\Gamma\Gamma)$ **using** *wfG-suffix assms* **by** *auto*
thus *?thesis* **using** *wfG-elims* **by** *metis*
qed

lemmas *wf-b-subst-lemmas* = *subst-eb.simps wf-intros*
forget-subst subst-b-b-def subst-b-v-def subst-b-ce-def fresh-e-opp-all subst-bb.simps wfV-b-fresh ms-fresh-all(*6*)

lemma *wf-b-subst1*:
fixes $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(string*\tau)\ list$ **and** $\Delta::\Delta$ **and** $b::b$ **and** $ftq::fun\text{-}typ\text{-}q$ **and** $ft::fun\text{-}typ$ **and** $s::s$ **and** $b'::b$ **and** $ce::ce$ **and** $td::type\text{-}def$
and $cs::branch\text{-}s$ **and** $css::branch\text{-}list$
shows $\Theta\ ;\ B'\ ;\ \Gamma \vdash_{wf} v : b' \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta\ ;\ B \vdash_{wf} b \Longrightarrow \Theta\ ;\ B\ ;\ \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb} : b'[bv::=b]_{bb}$ **and**
$\Theta\ ;\ B'\ ;\ \Gamma \vdash_{wf} c \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta\ ;\ B \vdash_{wf} b \Longrightarrow \Theta\ ;\ B\ ;\ \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} c[bv::=b]_{cb}$ **and**
$\Theta\ ;\ B' \vdash_{wf} \Gamma \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta\ ;\ B \vdash_{wf} b \Longrightarrow \Theta\ ;\ B \vdash_{wf} \Gamma[bv::=b]_{\Gamma b}$ **and**
$\Theta\ ;\ B'\ ;\ \Gamma \vdash_{wf} \tau \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta\ ;\ B \vdash_{wf} b \Longrightarrow \Theta\ ;\ B\ ;\ \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \tau[bv::=b]_{\tau b}$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow True$ **and**

$\vdash_{wf}$ Θ ⟹ *True* **and**
Θ ; B' $\vdash_{wf}$ b' ⟹ {|*bv*|} = B' ⟹ Θ ; B $\vdash_{wf}$ b ⟹ Θ ; B $\vdash_{wf}$ $b'[bv::=b]_{bb}$ **and**
Θ ; B' ; Γ $\vdash_{wf}$ *ce* : b' ⟹ {|*bv*|} = B' ⟹ Θ ; B $\vdash_{wf}$ b ⟹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $ce[bv::=b]_{ceb}$: $b'[bv::=b]_{bb}$ **and**
Θ $\vdash_{wf}$ *td* ⟹ *True*
proof(*nominal-induct*
b' **and** *c* **and** Γ **and** τ **and** *ts* **and** Θ **and** b' **and** b' **and** *td*
avoiding: *bv b B*
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfB-intI* Θ *ℬ*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-boolI* Θ *ℬ*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-unitI* Θ *ℬ*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-bitvecI* Θ *ℬ*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-pairI* Θ *ℬ b1 b2*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-consI* Θ *s dclist ℬ*)
then show *?case* **using** *subst-bb.simps Wellformed.wfB-consI* **by** *simp*
next
case (*wfB-appI* Θ *ba s bva dclist ℬ*)
then show *?case* **using** *subst-bb.simps Wellformed.wfB-appI forget-subst wfB-supp*
by (*metis bot.extremum-uniqueI ex-in-conv fresh-def subst-b-b-def supp-empty-fset*)
next
case (*wfV-varI* Θ *ℬ1* Γ *b1 c x*)
show *?case* **unfolding** *subst-vb.simps* **proof**
show Θ ; B $\vdash_{wf}$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *wfV-varI* **by** *auto*
show *Some* ($b1[bv::=b]_{bb}$, $c[bv::=b]_{cb}$) = *lookup* $\Gamma[bv::=b]_{\Gamma b}$ *x* **using** *subst-b-lookup wfV-varI* **by** *simp*
qed
next
case (*wfV-litI* Θ *ℬ* Γ *l*)
then show *?case* **using** *Wellformed.wfV-litI subst-b-base-for-lit* **by** *simp*
next
case (*wfV-pairI* Θ *ℬ1* Γ *v1 b1 v2 b2*)
show *?case* **unfolding** *subst-vb.simps* **proof**(*subst subst-bb.simps*,*rule*)
show Θ ; B ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $v1[bv::=b]_{vb}$: $b1[bv::=b]_{bb}$ **using** *wfV-pairI* **by** *simp*
show Θ ; B ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $v2[bv::=b]_{vb}$: $b2[bv::=b]_{bb}$ **using** *wfV-pairI* **by** *simp*
qed
next
case (*wfV-consI s dclist* Θ *dc x* b' *c* $ℬ'$ Γ *v*)
show *?case* **unfolding** *subst-vb.simps* **proof**(*subst subst-bb.simps*, *rule Wellformed.wfV-consI*)
show *1*:*AF-typedef s dclist* ∈ *set* Θ **using** *wfV-consI* **by** *auto*
show *2*:(*dc*, ⦃ *x* : b' | *c* ⦄) ∈ *set dclist* **using** *wfV-consI* **by** *auto*
have Θ ; B ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $v[bv::=b]_{vb}$: $b'[bv::=b]_{bb}$ **using** *wfV-consI* **by** *auto*

moreover hence *supp* $b' = \{\}$ **using** *1 2 wfTh-lookup-supp-empty τ.supp wfX-wfY* **by** *blast*
moreover hence $b'[bv::=b]_{bb} = b'$ **using** *forget-subst subst-bb-def fresh-def* **by** (*metis empty-iff subst-b-b-def*)
ultimately show Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb} : b'$ **using** *wfV-consI* **by** *simp*
qed
next
case (*wfV-conspI s bva dclist* Θ *dc x* b' *c* $\mathcal{B}'$ *ba* Γ *v*)
have $*$:*atom bv* ♯ b' **using** *wfTh-poly-supp-b*[*of s bva dclist* Θ *dc x* b' *c*] *fresh-def wfX-wfY* ‹*atom bva* ♯ *bv*›
by (*metis insert-iff not-self-fresh singleton-insert-inj-eq' subsetI subset-antisym wfV-conspI wfV-conspI.hyps*(*4*) *wfV-conspI.prems*(*2*))
show *?case* **unfolding** *subst-vb.simps subst-bb.simps* **proof**
show ‹*AF-typedef-poly s bva dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, ⦃ *x* : b' | *c* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
thus ‹ Θ ; B $\vdash_{wf}$ $ba[bv::=b]_{bb}$ › **using** *wfV-conspI* **by** *metis*
have *atom bva* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *fresh-subst-if subst-b-Γ-def wfV-conspI* **by** *metis*
moreover have *atom bva* ♯ $ba[bv::=b]_{bb}$ **using** *fresh-subst-if subst-b-b-def wfV-conspI* **by** *metis*
moreover have *atom bva* ♯ $v[bv::=b]_{vb}$ **using** *fresh-subst-if subst-b-v-def wfV-conspI* **by** *metis*
ultimately show ‹*atom bva* ♯ (Θ, B, $\Gamma[bv::=b]_{\Gamma b}$, $ba[bv::=b]_{bb}$, $v[bv::=b]_{vb}$)›
unfolding *fresh-prodN* **using** *wfV-conspI fresh-def supp-fset* **by** *auto*
show ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb} : b'[bva::=ba[bv::=b]_{bb}]_{bb}$ ›
using *wfV-conspI subst-bb-commute*[*of bv* b' *bva ba b*] $*$ *wfV-conspI* **by** *metis*
qed
next
case (*wfTI z* Θ $\mathcal{B}'$ Γ' b' *c*)
show *?case* **proof**(*subst subst-tb.simps*, *rule Wellformed.wfTI*)
show *atom z* ♯ (Θ, B, $\Gamma'[bv::=b]_{\Gamma b}$) **using** *wfTI subst-g-b-x-fresh* **by** *simp*
show Θ ; B $\vdash_{wf}$ $b'[bv::=b]_{bb}$ **using** *wfTI* **by** *auto*
show Θ ; B ; (z, $b'[bv::=b]_{bb}$, *TRUE*) $\#_{\Gamma}$ $\Gamma'[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $c[bv::=b]_{cb}$ **using** *wfTI* **by** *simp*
qed
next
case (*wfC-eqI* Θ $\mathcal{B}'$ Γ *e1* b' *e2*)
thus *?case* **using** *Wellformed.wfC-eqI subst-db.simps subst-cb.simps wfC-eqI* **by** *metis*
next
case (*wfG-nilI* Θ $\mathcal{B}'$)
then show *?case* **using** *Wellformed.wfG-nilI subst-gb.simps* **by** *simp*
next
case (*wfG-cons1I* c' Θ $\mathcal{B}'$ Γ' *x* b')
show *?case* **proof**(*subst subst-gb.simps*, *rule Wellformed.wfG-cons1I*)
show $c'[bv::=b]_{cb} \notin \{TRUE, FALSE\}$ **using** *wfG-cons1I*(*1*)
by(*nominal-induct* c' *rule*: *c.strong-induct*,*auto*+)
show Θ ; B $\vdash_{wf}$ $\Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons1I* **by** *auto*
show *atom x* ♯ $\Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons1I subst-g-b-x-fresh* **by** *auto*
show Θ ; B ; (x, $b'[bv::=b]_{bb}$, *TRUE*) $\#_{\Gamma}$ $\Gamma'[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $c'[bv::=b]_{cb}$ **using** *wfG-cons1I* **by** *auto*
show Θ ; B $\vdash_{wf}$ $b'[bv::=b]_{bb}$ **using** *wfG-cons1I* **by** *auto*
qed
next
case (*wfG-cons2I* c' Θ $\mathcal{B}'$ Γ' *x* b')
show *?case* **proof**(*subst subst-gb.simps*, *rule Wellformed.wfG-cons2I*)
show $c'[bv::=b]_{cb} \in \{TRUE, FALSE\}$ **using** *wfG-cons2I* **by** *auto*
show Θ ; B $\vdash_{wf}$ $\Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons2I* **by** *auto*

show *atom x* $\sharp$ $\Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons2I subst-g-b-x-fresh* **by** *auto*
show Θ ; B $\vdash_{wf}$ $b'[bv::=b]_{bb}$ **using** *wfG-cons2I* **by** *auto*
qed
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
then show *?case* **using** *subst-ceb.simps wf-intros wfX-wfY*
by (*metis wf-b-subst-lemmas wfCE-b-fresh*)
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY*
by *metis*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY*
by *metis*
next
case (*wfCE-eqI* Θ $\mathcal{B}$ Γ *v1 b v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY*
by *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case*
by (*metis* (*no-types*) *subst-bb.simps*(*5*) *subst-ceb.simps*(*3*) *wfCE-fstI.hyps*(*2*)
wfCE-fstI.prems(*1*) *wfCE-fstI.prems*(*2*) *Wellformed.wfCE-fstI*)
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case*
by (*metis* (*no-types*) *subst-bb.simps*(*5*) *subst-ceb.simps wfCE-sndI.hyps*(*2*)
wfCE-sndI wfCE-sndI.prems(*2*) *Wellformed.wfCE-sndI*)
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY wf-b-subst-lemmas wfCE-b-fresh*

proof −
show *?thesis*
using *wfCE-concatI.hyps*(*2*) *wfCE-concatI.hyps*(*4*) *wfCE-concatI.prems*(*1*) *wfCE-concatI.prems*(*2*)

Wellformed.wfCE-concatI **by** *auto*
qed
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY wf-b-subst-lemmas wfCE-b-fresh*
by *metis*
qed(*auto simp add*: *wf-intros*)

lemma *wf-b-subst2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *b*::*b*
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *s*::*s* **and** b'::*b* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; Φ ; B' ; Γ ; Δ $\vdash_{wf}$ e : b' $\Longrightarrow$ $\{|bv|\} = B'$ $\Longrightarrow$ Θ ; B $\vdash_{wf}$ b $\Longrightarrow$ Θ ; Φ ; B ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash_{wf}$ $e[bv::=b]_{eb}$: $b'[bv::=b]_{bb}$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ s : b $\Longrightarrow$ *True* **and**

Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow$ *True* **and**
Θ $\vdash_{wf}$ (Φ::Φ) $\Longrightarrow$ *True* **and**
Θ ; B' ; Γ $\vdash_{wf}$ Δ $\Longrightarrow$ $\{|bv|\} = B' \Longrightarrow \Theta$; B $\vdash_{wf}$ $b \Longrightarrow \Theta$; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$
and
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* $\Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** b **and** b **and** b **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: *bv b B*
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-valI* Θ' Φ' $\mathcal{B}'$ Γ' Δ' v' b')
then show *?case* **unfolding** *subst-vb.simps subst-eb.simps* **using** *wf-b-subst1*(*1*) *Wellformed.wfE-valI*
by *auto*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps*
using *wf-b-subst-lemmas wf-b-subst1*(*1*) *Wellformed.wfE-plusI*
proof –
have $\forall$ *b ba v g f ts.* ((*ts* ; *f* ; $g[bv::=ba]_{\Gamma b} \vdash_{wf}$ $v[bv::=ba]_{vb}$: $b[bv::=ba]_{bb}$) $\vee$ $\neg$ *ts* ; $\mathcal{B}$; *g* $\vdash_{wf}$ *v* :
b) $\vee$ $\neg$ *ts* ; *f* $\vdash_{wf}$ *ba*
using *wfE-plusI.prems*(*1*) *wf-b-subst1*(*1*) **by** *force*
then show Θ ; Φ ; B ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b} \vdash_{wf}$ [*plus* $v1[bv::=b]_{vb}$ $v2[bv::=b]_{vb}$ $]^e$:
$B\text{-}int[bv::=b]_{bb}$

by (*metis wfE-plusI.hyps*(*1*) *wfE-plusI.hyps*(*4*) *wfE-plusI.hyps*(*5*) *wfE-plusI.hyps*(*6*) *wfE-plusI.prems*(*1*)
wfE-plusI.prems(*2*) *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-plusI wf-b-subst-lemmas*(*86*))
qed
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps*
using *wf-b-subst-lemmas wf-b-subst1* *Wellformed.wfE-leqI*
proof –
have $\bigwedge$*ts f b ba g v.* $\neg$ (*ts* ; *f* $\vdash_{wf}$ *b*) $\vee$ $\neg$ (*ts* ; $\{|ba|\}$; *g* $\vdash_{wf}$ *v* : *B-int*) $\vee$ (*ts* ; *f* ; $g[ba::=b]_{\Gamma b} \vdash_{wf}$
$v[ba::=b]_{vb}$: *B-int*)
by (*metis wf-b-subst1*(*1*) *wf-b-subst-lemmas*(*86*))
then show Θ ; Φ ; B ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b} \vdash_{wf}$ [*leq* $v1[bv::=b]_{vb}$ $v2[bv::=b]_{vb}$ $]^e$: $B\text{-}bool[bv::=b]_{bb}$
by (*metis* (*no-types*) *wfE-leqI.hyps*(*1*) *wfE-leqI.hyps*(*4*) *wfE-leqI.hyps*(*5*) *wfE-leqI.hyps*(*6*) *wfE-leqI.prems*(*1*)
wfE-leqI.prems(*2*) *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-leqI wf-b-subst-lemmas*(*87*))
qed
next
case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 bb v2*)
show *?case* **unfolding** *subst-eb.simps subst-bb.simps* **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfX-wfY wfE-eqI* **by** *metis*
show ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ › **using** *wfX-wfY wfE-eqI* **by** *metis*
show ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$ $v1[bv::=b]_{vb}$: *bb* › **using** *subst-bb.simps wfE-eqI*
by (*metis* (*no-types*, *opaque-lifting*) *empty-iff insert-iff wf-b-subst1*(*1*))
show ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$ $v2[bv::=b]_{vb}$: *bb* › **using** *wfX-wfY wfE-eqI*
by (*metis insert-iff singleton-iff wf-b-subst1*(*1*) *wf-b-subst-lemmas*(*86*) *wf-b-subst-lemmas*(*87*)
wf-b-subst-lemmas(*90*))
show ‹*bb* $\in$ {*B-bool*, *B-int*, *B-unit*}› **using** *wfE-eqI* **by** *auto*
qed

next
  case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
   then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*84*) *wf-b-subst1*(*1*) *Wellformed.wfE-fstI*
    by (*metis wf-b-subst-lemmas*(*89*))
next
  case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
   then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-sndI*
  by (*metis wf-b-subst-lemmas*(*89*))
next
  case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-concatI*

  by (*metis wf-b-subst-lemmas*(*91*))
next
  case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
   then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-splitI*
    by (*metis wf-b-subst-lemmas*(*89*) *wf-b-subst-lemmas*(*91*))
next
  case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
   then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-lenI*
    by (*metis wf-b-subst-lemmas*(*91*) *wf-b-subst-lemmas*(*89*))
next
  case (*wfE-appI* Θ Φ $\mathcal{B}'$ Γ Δ *f x b' c τ s v*)
   hence *bf*: *atom bv* ♯ *b'* **using** *wfPhi-f-simple-wfT wfT-supp bv-not-in-dom-g wfPhi-f-simple-supp-b fresh-def* **by** *fast*
  hence *bseq*: $b'[bv::=b]_{bb} = b'$ **using** *subst-bb.simps wf-b-subst-lemmas* **by** *metis*
  have Θ ; Φ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b} \vdash_{wf}$ (*AE-app f* ($v[bv::=b]_{vb}$)) : (*b-of* ($\tau[bv::=b]_{\tau b}$))
  proof
    show Θ $\vdash_{wf}$ Φ **using** *wfE-appI* **by** *auto*
    show Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfE-appI* **by** *simp*
   have *atom bv* ♯ *τ* **using** *wfPhi-f-simple-wfT*[*OF wfE-appI*(*5*) *wfE-appI*(*1*), *THEN wfT-supp*] *bv-not-in-dom-g fresh-def* **by** *force*
    hence $\tau[bv::=b]_{\tau b} = \tau$ **using** *forget-subst subst-b-τ-def* **by** *metis*
     thus *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b' c* $\tau[bv::=b]_{\tau b}$ *s*))) = *lookup-fun* Φ *f* **using** *wfE-appI* **by** *simp*
    show Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb}$: *b'* **using** *wfE-appI bseq wf-b-subst1* **by** *metis*
  qed
  then show *?case* **using** *subst-eb.simps b-of-subst-bb-commute* **by** *simp*
next
  case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b' bv1 v1 τ1 f x1 b1 c1 s1*)
  then have ∗: *atom bv* ♯ *b1* **using** *wfPhi-f-supp*(*1*) *wfE-appPI*(*7*,*11*)
      by (*metis fresh-def fresh-finsert singleton-iff subsetD fresh-def supp-at-base wfE-appPI.hyps*(*1*))
   have Θ ; Φ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b} \vdash_{wf}$ *AE-appP f* $b'[bv::=b]_{bb}$ ($v1[bv::=b]_{vb}$) : (*b-of τ1*)$[bv1::=b'[bv::=b]_{bb}]_b$
  proof
    show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfE-appPI* **by** *auto*
    show ‹ Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ › **using** *wfE-appPI* **by** *auto*
    show ‹ Θ ; *B* $\vdash_{wf} b'[bv::=b]_{bb}$ › **using** *wfE-appPI wf-b-subst1* **by** *auto*

have *atom bv1* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *fresh-subst-if subst-b-Γ-def wfE-appPI* **by** *metis*
moreover have *atom bv1* ♯ $b'[bv::=b]_{bb}$ **using** *fresh-subst-if subst-b-b-def wfE-appPI* **by** *metis*
moreover have *atom bv1* ♯ $v1[bv::=b]_{vb}$ **using** *fresh-subst-if subst-b-v-def wfE-appPI* **by** *metis*
moreover have *atom bv1* ♯ $\Delta[bv::=b]_{\Delta b}$ **using** *fresh-subst-if subst-b-Δ-def wfE-appPI* **by** *metis*
moreover have *atom bv1* ♯ $(b\text{-}of\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_{bb}$ **using** *fresh-subst-if subst-b-b-def wfE-appPI* **by** *metis*
ultimately show *atom bv1* ♯ $(\Phi, \Theta, B, \Gamma[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, b'[bv::=b]_{bb}, v1[bv::=b]_{vb}, (b\text{-}of\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_b)$
using *wfE-appPI* **using** *fresh-def fresh-prodN subst-b-b-def* **by** *metis*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1 s1*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*

have ‹ $\Theta\ ;\ B\ ;\ \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v1[bv::=b]_{vb} : b1[bv1::=b']_b[bv::=b]_{bb}$ ›
using *wfE-appPI subst-b-b-def* ∗ *wf-b-subst1* **by** *metis*
thus ‹ $\Theta\ ;\ B\ ;\ \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v1[bv::=b]_{vb} : b1[bv1::=b'[bv::=b]_{bb}]_b$ ›
using *subst-bb-commute subst-b-b-def* ∗ **by** *auto*
qed
moreover have *atom bv* ♯ *b-of τ1* **proof** −
have *supp* (*b-of τ1*) ⊆ { *atom bv1* } **using** *wfPhi-f-poly-supp-b-of-t*
using *b-of.simps wfE-appPI wfPhi-f-supp*(*5*) **by** *simp*
thus *?thesis* **using** *wfE-appPI*
fresh-def fresh-finsert singleton-iff subsetD fresh-def supp-at-base wfE-appPI.hyps **by** *metis*
qed
ultimately show *?case* **using** *subst-eb.simps*(*3*) *subst-bb-commute subst-b-b-def* ∗ **by** *simp*
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}'$ Γ Δ *u τ*)

have $\Theta\ ;\ \Phi\ ;\ B\ ;\ subst\text{-}gb\ \Gamma\ bv\ b\ ;\ subst\text{-}db\ \Delta\ bv\ b \vdash_{wf} (AE\text{-}mvar\ u)[bv::=b]_{eb} : (b\text{-}of\ (\tau[bv::=b]_{\tau b}))$

proof(*subst subst-eb.simps*,*rule Wellformed.wfE-mvarI*)
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-mvarI* **by** *simp*
show $\Theta\ ;\ B\ ;\ \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfE-mvarI* **by** *metis*
show $(u, \tau[bv::=b]_{\tau b}) \in setD\ \Delta[bv::=b]_{\Delta b}$
using *wfE-mvarI subst-db.simps set-insert subst-d-b-member* **by** *simp*
qed
thus *?case* **using** *b-of-subst-bb-commute* **by** *auto*

next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}'$ Γ)
then show *?case* **using** *subst-db.simps Wellformed.wfD-emptyI wf-b-subst1* **by** *simp*
next
case (*wfD-cons* Θ $\mathcal{B}'$ Γ' Δ *τ u*)
show *?case* **proof**(*subst subst-db.simps*, *rule Wellformed.wfD-cons*)
show $\Theta\ ;\ B\ ;\ \Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfD-cons* **by** *auto*
show $\Theta\ ;\ B\ ;\ \Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} \tau[bv::=b]_{\tau b}$ **using** *wfD-cons wf-b-subst1* **by** *auto*
show $u \notin fst$ ‘ $setD\ \Delta[bv::=b]_{\Delta b}$ **using** *wfD-cons subst-b-lookup-d* **by** *metis*
qed
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)

show *?case* **by** *auto*
qed(*auto*)

lemmas *wf-b-subst* = *wf-b-subst1 wf-b-subst2*

lemma *wfT-subst-wfT*:
fixes τ::τ **and** b'::*b* **and** *bv*::*bv*
assumes Θ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **and** Θ ; *B* $\vdash_{wf}$ b'
shows Θ ; *B* ; (*x*,*b*[*bv*::=b']$_{bb}$,*c*[*bv*::=b']$_{cb}$) $\#_\Gamma$ *GNil* $\vdash_{wf}$ (τ[*bv*::=b']$_{\tau b}$)
proof −
have Θ ; *B* ; ((*x*,*b*,*c*) $\#_\Gamma$ *GNil*)[*bv*::=b']$_{\Gamma b}$ $\vdash_{wf}$ (τ[*bv*::=b']$_{\tau b}$)
using *wf-b-subst assms* **by** *metis*
thus *?thesis* **using** *subst-gb.simps wf-b-subst-lemmas wfCE-b-fresh* **by** *metis*
qed

lemma *wf-trans*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *s*::*s*
and *cs*::*branch-s* **and** *css*::*branch-list* **and** Θ::Θ
shows Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : b' $\Longrightarrow$ Γ = (*x*, *b*, *c2*) $\#_\Gamma$ *G* $\Longrightarrow$ Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c2* $\Longrightarrow$ Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_\Gamma$ *G* $\vdash_{wf}$ *v* : b' **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* $\Longrightarrow$ Γ = (*x*, *b*, *c2*) $\#_\Gamma$ *G* $\Longrightarrow$ Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c2* $\Longrightarrow$ Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c* **and**
Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ $\Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* $\Longrightarrow$ *True* **and**
$\vdash_{wf}$ Θ $\Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* $\Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : b' $\Longrightarrow$ Γ = (*x*, *b*, *c2*) $\#_\Gamma$ *G* $\Longrightarrow$ Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c2* $\Longrightarrow$ Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_\Gamma$ *G* $\vdash_{wf}$ *ce* : b' **and**
Θ $\vdash_{wf}$ *td* $\Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** *c* **and** Γ **and** τ **and** *ts* **and** Θ **and** *b* **and** b' **and** *td*
avoiding: *c1*
arbitrary: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ b' c' x')
have *wbg*: Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c1*) $\#_\Gamma$ *G* **using** *wfC-wf wfV-varI* **by** *simp*
show *?case* **proof**(*cases* *x*=x')
case *True*
have *Some* (b', *c1*) = *lookup* ((*x*, *b*, *c1*) $\#_\Gamma$ *G*) x' **using** *lookup.simps wfV-varI* **using** *True* **by** *auto*
then show *?thesis* **using** *Wellformed.wfV-varI wbg* **by** *simp*
next
case *False*
then have *Some* (b', c') = *lookup* ((*x*, *b*, *c1*) $\#_\Gamma$ *G*) x' **using** *lookup.simps wfV-varI*
by *simp*
then show *?thesis* **using** *Wellformed.wfV-varI wbg* **by** *simp*
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x1* b' *c* $\mathcal{B}$ *b1* Γ *v*)

show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, ⦃ *x1* : b' | *c* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹Θ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfV-conspI* **by** *auto*
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, (*x*, *b*, *c1*) $\#_{\Gamma}$ *G*, *b1*, *v*)› **unfolding** *fresh-prodN fresh-GCons* **using** *wfV-conspI fresh-prodN fresh-GCons* **by** *simp*
show ‹Θ; $\mathcal{B}$; (*x*, *b*, *c1*) $\#_{\Gamma}$ *G* $\vdash_{wf}$ *v* : $b'[bv::=b1]_{bb}$› **using** *wfV-conspI* **by** *auto*
qed
qed((*auto* | *metis wfC-wf wf-intros*) +)

end

Chapter 9

Type System

The MiniSail type system. We define subtyping judgement first and then typing judgement for the term forms

9.1 Subtyping

Subtyping is defined on top of refinement constraint logic (RCL). A subtyping check is converted into an RCL validity check.

inductive *subtype* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow$ *bool* (‹- ; - ; - ⊢ - $\lesssim$ -› [*50*, *50*, *50*] *50*) **where**
  subtype-baseI: ⟦
   atom $x \sharp (\Theta, \mathcal{B}, \Gamma, z, c, z', c')$;
   $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!| z : b \mid c |\!\}$;
   $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!| z' : b \mid c' |\!\}$;
   $\Theta; \mathcal{B}$; $(x, b, c[z::=[x]^v]_v) \#_\Gamma \Gamma \models c'[z'::=[x]^v]_v$
⟧ $\Longrightarrow$
   $\Theta; \mathcal{B}; \Gamma \vdash \{\!| z : b \mid c |\!\} \lesssim \{\!| z' : b \mid c' |\!\}$

equivariance *subtype*
nominal-inductive *subtype*
  avoids *subtype-baseI*: *x*
proof(*goal-cases*)
  case (*1* Θ $\mathcal{B}$ Γ *z b c z′ c′ x*)
  then show *?case* **using** *fresh-star-def 1* **by** *force*
next
  case (*2* Θ $\mathcal{B}$ Γ *z b c z′ c′ x*)
  then show *?case* **by** *auto*
qed

inductive-cases *subtype-elims*:
  $\Theta; \mathcal{B}; \Gamma \vdash \{\!| z : b \mid c |\!\} \lesssim \{\!| z' : b \mid c' |\!\}$
  $\Theta; \mathcal{B}; \Gamma \vdash \tau_1 \lesssim \tau_2$

9.2 Literals

The type synthesised has the constraint that z equates to the literal

inductive *infer-l* :: $l \Rightarrow \tau \Rightarrow bool$ (‹ ⊢ - ⇒ -› [*50*, *50*] *50*) **where**
infer-trueI: $\vdash$ *L-true* $\Rightarrow$ ⦃ z : *B-bool* | $[[z]^v]^{ce}$ == $[[L\text{-}true]^v]^{ce}$ ⦄
| *infer-falseI*: $\vdash$ *L-false* $\Rightarrow$ ⦃ z : *B-bool* | $[[z]^v]^{ce}$ == $[[L\text{-}false]^v]^{ce}$ ⦄
| *infer-natI*: $\vdash$ *L-num n* $\Rightarrow$ ⦃ z : *B-int* | $[[z]^v]^{ce}$ == $[[L\text{-}num\ n]^v]^{ce}$ ⦄
| *infer-unitI*: $\vdash$ *L-unit* $\Rightarrow$ ⦃ z : *B-unit* | $[[z]^v]^{ce}$ == $[[L\text{-}unit]^v]^{ce}$ ⦄
| *infer-bitvecI*: $\vdash$ *L-bitvec bv* $\Rightarrow$ ⦃ z : *B-bitvec* | $[[z]^v]^{ce}$ == $[[L\text{-}bitvec\ bv]^v]^{ce}$ ⦄

nominal-inductive *infer-l* .
equivariance *infer-l*

inductive-cases *infer-l-elims*[*elim!*]:
$\vdash$ *L-true* $\Rightarrow \tau$
$\vdash$ *L-false* $\Rightarrow \tau$
$\vdash$ *L-num n* $\Rightarrow \tau$
$\vdash$ *L-unit* $\Rightarrow \tau$
$\vdash$ *L-bitvec x* $\Rightarrow \tau$
$\vdash l \Rightarrow \tau$

lemma *infer-l-form2*[*simp*]:
shows $\exists z.\ \vdash l \Rightarrow$ (⦃ z : *base-for-lit l* | $[[z]^v]^{ce}$ == $[[l]^v]^{ce}$ ⦄)
proof (*nominal-induct l rule*: *l.strong-induct*)
case (*L-num x*)
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case *L-true*
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case *L-false*
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case *L-unit*
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case (*L-bitvec x*)
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
qed

9.3 Values

inductive *infer-v* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow bool$ (‹ - ; - ; - ⊢ - ⇒ -› [*50*, *50*, *50*] *50*) **where**

infer-v-varI: ⟦
Θ; $\mathcal{B} \vdash_{wf} \Gamma$;
Some (b,c) = *lookup* Γ x;
atom z ♯ x ; *atom* z ♯ $(\Theta, \mathcal{B}, \Gamma)$
⟧ $\Longrightarrow$
Θ; $\mathcal{B}$; $\Gamma \vdash [x]^v \Rightarrow$ ⦃ z : b | $[[z]^v]^{ce}$ == $[[x]^v]^{ce}$ ⦄

| *infer-v-litI*: ⟦
Θ; $\mathcal{B} \vdash_{wf} \Gamma$;
$\vdash l \Rightarrow \tau$
⟧ $\Longrightarrow$

$\Theta;\ \mathcal{B};\ \Gamma \vdash [l]^v \Rightarrow \tau$

| *infer-v-pairI*: $[\![$
$atom\ z \sharp (v1,\ v2);\ atom\ z \sharp\ (\Theta,\ \mathcal{B},\ \Gamma)$;
$\Theta;\ \mathcal{B};\ \Gamma \vdash (v1::v) \Rightarrow t1$;
$\Theta;\ \mathcal{B}\ ;\ \ \Gamma \vdash (v2::v) \Rightarrow t2$
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash\ V\text{-}pair\ v1\ v2 \Rightarrow (\{\!|\ z : B\text{-}pair\ (b\text{-}of\ t1)\ (b\text{-}of\ t2)\ \mid [[z]^v]^{ce} == [[v1,v2]^v]^{ce}\ |\!\})$

| *infer-v-consI*: $[\![$
$AF\text{-}typedef\ s\ dclist \in set\ \Theta$;
$(dc,\ tc) \in set\ dclist$;
$\Theta;\ \ \mathcal{B};\ \Gamma \vdash v \Rightarrow tv$;
$\Theta;\ \mathcal{B};\ \Gamma \vdash tv \lesssim tc$;
$atom\ z \sharp\ v$; $\ atom\ z \sharp (\Theta,\ \mathcal{B},\ \Gamma)$
$]\!] \Longrightarrow$
$\Theta;\ \ \mathcal{B};\ \Gamma\ \ \vdash V\text{-}cons\ s\ dc\ v \Rightarrow (\{\!|\ z : B\text{-}id\ s \mid\ \ [[z]^v]^{ce} == [\ V\text{-}cons\ s\ dc\ v\ ]^{ce}\ |\!\})$

| *infer-v-conspI*: $[\![$
$AF\text{-}typedef\text{-}poly\ s\ bv\ dclist \in set\ \Theta$;
$(dc,\ tc) \in set\ dclist$;
$\Theta;\ \mathcal{B};\ \Gamma \vdash v \Rightarrow tv$;
$\Theta;\ \mathcal{B};\ \Gamma \vdash tv \lesssim tc[bv::=b]_{\tau b}$;
$atom\ z \sharp (\Theta,\ \mathcal{B},\ \Gamma,\ v,\ b)$;
$atom\ bv \sharp (\Theta,\ \mathcal{B},\ \Gamma,\ v,\ b)$;
$\Theta;\ \mathcal{B} \vdash_{wf}\ b$
$]\!] \Longrightarrow$
$\Theta;\ \mathcal{B};\ \Gamma \vdash\ V\text{-}consp\ s\ dc\ b\ v \Rightarrow (\{\!|\ z : B\text{-}app\ s\ b \mid\ \ [[z]^v]^{ce} == (CE\text{-}val\ (V\text{-}consp\ s\ dc\ b\ v))\ |\!\})$

equivariance *infer-v*
nominal-inductive *infer-v*
avoids *infer-v-conspI*: *bv* **and** *z* | *infer-v-varI*: *z* | *infer-v-pairI*: *z* | *infer-v-consI*: *z*
proof(*goal-cases*)
case (*1* $\Theta\ \mathcal{B}\ \Gamma\ b\ c\ x\ z$)
hence $atom\ z \sharp \{\!|\ z : b\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ [\ x\ ]^v\ ]^{ce}\ |\!\}$ **using** *τ.fresh* **by** *simp*
then show *?case* **unfolding** *fresh-star-def* **using** *1* **by** *simp*
next
case (*2* $\Theta\ \mathcal{B}\ \Gamma\ b\ c\ x\ z$)
then show *?case* **by** *auto*
next
case (*3* $z\ v1\ v2\ \Theta\ \mathcal{B}\ \Gamma\ t1\ t2$)
hence $atom\ z \sharp \{\!|\ z : [\ b\text{-}of\ t1\ ,\ b\text{-}of\ t2\ ]^b\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ [\ v1\ ,\ v2\ ]^v\ ]^{ce}\ |\!\}$ **using** *τ.fresh* **by** *simp*
then show *?case* **unfolding** *fresh-star-def* **using** *3* **by** *simp*
next
case (*4* $z\ v1\ v2\ \Theta\ \mathcal{B}\ \Gamma\ t1\ t2$)
then show *?case* **by** *auto*
next
case (*5* $s\ dclist\ \Theta\ dc\ tc\ \mathcal{B}\ \Gamma\ v\ tv\ z$)
hence $atom\ z \sharp \{\!|\ z : B\text{-}id\ s\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ V\text{-}cons\ s\ dc\ v\ ]^{ce}\ |\!\}$ **using** *τ.fresh b.fresh pure-fresh* **by** *auto*
moreover have $atom\ z \sharp\ V\text{-}cons\ s\ dc\ v$ **using** *v.fresh 5* **using** *v.fresh fresh-prodN pure-fresh* **by** *metis*

then show *?case* **unfolding** *fresh-star-def* **using** *5* **by** *simp*
next
case (*6 s dclist* Θ *dc tc* *B* Γ *v tv z*)
then show *?case* **by** *auto*
next
case (*7 s bv dclist* Θ *dc tc* *B* Γ *v tv b z*)
hence *atom bv* ♯ *V-consp s dc b v* **using** *v.fresh fresh-prodN pure-fresh* **by** *metis*
moreover then have *atom bv* ♯ ⦃ *z* : *B-id s* | [[*z*]^{v}]^{ce} == [*V-consp s dc b v*]^{ce} ⦄
using *τ.fresh ce.fresh v.fresh* **by** *auto*
moreover have *atom z* ♯ *V-consp s dc b v* **using** *v.fresh fresh-prodN pure-fresh 7* **by** *metis*
moreover then have *atom z* ♯ ⦃ *z* : *B-id s* | [[*z*]^{v}]^{ce} == [*V-consp s dc b v*]^{ce} ⦄
using *τ.fresh ce.fresh v.fresh* **by** *auto*
ultimately show *?case* **using** *fresh-star-def 7* **by** *force*
next
case (*8 s bv dclist* Θ *dc tc* *B* Γ *v tv b z*)
then show *?case* **by** *auto*
qed

inductive-cases *infer-v-elims*[*elim*!]:
Θ; *B*; Γ ⊢ *V-var x* ⇒ τ
Θ; *B*; Γ ⊢ *V-lit l* ⇒ τ
Θ; *B*; Γ ⊢ *V-pair v1 v2* ⇒ τ
Θ; *B*; Γ ⊢ *V-cons s dc v* ⇒ τ
Θ; *B*; Γ ⊢ *V-pair v1 v2* ⇒ (⦃ *z* : *b* | *c* ⦄)
Θ; *B*; Γ ⊢ *V-pair v1 v2* ⇒ (⦃ *z* : [*b1* , *b2*]^{b} | [[*z*]^{v}]^{ce} == [[*v1*,*v2*]^{v}]^{ce} ⦄)
Θ; *B*; Γ ⊢ *V-consp s dc b v* ⇒ τ

inductive *check-v* :: Θ ⇒ *B* ⇒ Γ ⇒ *v* ⇒ τ ⇒ *bool* (‹- ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*] *50*) **where**
check-v-subtypeI: ⟦ Θ; *B*; Γ ⊢ τ*1* ≲ τ*2*; Θ; *B*; Γ ⊢ *v* ⇒ τ*1* ⟧ ⟹ Θ; *B* ; Γ ⊢ *v* ⇐ τ*2*
equivariance *check-v*
nominal-inductive *check-v* **.**

inductive-cases *check-v-elims*[*elim*!]:
Θ; *B* ; Γ ⊢ *v* ⇐ τ

9.4 Expressions

Type synthesis for expressions

inductive *infer-e* :: Θ ⇒ Φ ⇒ *B* ⇒ Γ ⇒ Δ ⇒ *e* ⇒ τ ⇒ *bool* (‹- ; - ; - ; - ; - ⊢ - ⇒ -› [*50*, *50*, *50*,*50*] *50*) **where**

infer-e-valI: ⟦
(Θ; *B*; Γ $\vdash_{wf}$ Δ) ;
(Θ $\vdash_{wf}$ (Φ::Φ)) ;
(Θ; *B*; Γ ⊢ *v* ⇒ τ) ⟧ ⟹
Θ; Φ; *B*; Γ; Δ ⊢ (*AE-val v*) ⇒ τ

| *infer-e-plusI*: ⟦
Θ; *B*; Γ $\vdash_{wf}$ Δ ;
Θ $\vdash_{wf}$ (Φ::Φ) ;
Θ; *B*; Γ ⊢ *v1* ⇒ ⦃ *z1* : *B-int* | *c1* ⦄ ;

$\Theta;\ \mathcal{B};\ \Gamma \vdash\ \ v2\ \Rightarrow\ \{\!|\ z2 : B\text{-}int \mid c2\ |\!\};$
$atom\ z3\ \sharp\ (AE\text{-}op\ Plus\ v1\ v2);\ atom\ z3\ \sharp\ \Gamma\ \rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AE\text{-}op\ Plus\ v1\ v2\ \Rightarrow\ \{\!|\ z3 : B\text{-}int \mid [[z3]^v]^{ce} == (CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$

$\mid$ *infer-e-leqI*: $\llbracket$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta;$
$\Theta \vdash_{wf} (\Phi{::}\Phi)\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v1\ \Rightarrow\ \{\!|\ z1 : B\text{-}int \mid c1\ |\!\}\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v2\ \Rightarrow\ \{\!|\ z2 : B\text{-}int \mid c2\ |\!\};$
$atom\ z3\ \sharp\ (AE\text{-}op\ LEq\ v1\ v2);\ atom\ z3\ \sharp\ \Gamma$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AE\text{-}op\ LEq\ v1\ v2\ \Rightarrow\ \{\!|\ z3 : B\text{-}bool \mid [[z3]^v]^{ce} == (CE\text{-}op\ LEq\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$

$\mid$ *infer-e-eqI*: $\llbracket$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta;$
$\Theta \vdash_{wf} (\Phi{::}\Phi)\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v1\ \Rightarrow\ \{\!|\ z1 : b \mid c1\ |\!\}\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v2\ \Rightarrow\ \{\!|\ z2 : b \mid c2\ |\!\};$
$atom\ z3\ \sharp\ (AE\text{-}op\ Eq\ v1\ v2);\ atom\ z3\ \sharp\ \Gamma\ \ ;$
$b \in \{\ B\text{-}bool,\ B\text{-}int,\ B\text{-}unit\ \}$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AE\text{-}op\ Eq\ v1\ v2\ \Rightarrow\ \{\!|\ z3 : B\text{-}bool \mid [[z3]^v]^{ce} == (CE\text{-}op\ Eq\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$

$\mid$ *infer-e-appI*: $\llbracket$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta\ ;$
$\Theta \vdash_{wf} (\Phi{::}\Phi)\ ;$
$Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v \Leftarrow \{\!|\ \ x : b \mid c\ |\!\};$
$atom\ x\ \sharp\ (\Theta,\ \Phi,\ \mathcal{B},\ \Gamma,\ \Delta, v\ ,\ \tau);$
$\tau'[x{::=}v]_v = \tau$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AE\text{-}app\ f\ v \Rightarrow \tau$

$\mid$ *infer-e-appPI*: $\llbracket$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta\ ;$
$\Theta \vdash_{wf} (\Phi{::}\Phi)\ ;$
$\Theta;\ \mathcal{B} \vdash_{wf} b'\ ;$
$Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v \Leftarrow \{\!|\ \ x : b[bv{::=}b']_b \mid c[bv{::=}b']_b\ |\!\};\ atom\ x\ \sharp\ \Gamma;$
$(\tau'[bv{::=}b']_b[x{::=}v]_v) = \tau\ ;$
$atom\ bv\ \sharp\ (\Theta,\ \Phi,\ \mathcal{B},\ \Gamma,\ \Delta,\ b',\ v,\ \tau)$
$\rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AE\text{-}appP\ f\ b'\ v \Rightarrow \tau$

$\mid$ *infer-e-fstI*: $\llbracket$
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf} \Delta\ ;$
$\Theta \vdash_{wf} (\Phi{::}\Phi)\ ;$
$\Theta;\ \mathcal{B};\ \Gamma \vdash v \Rightarrow \{\!|\ z' : [b1,b2]^b \mid c\ |\!\};$
$atom\ z\ \sharp\ AE\text{-}fst\ v\ ;\ atom\ z\ \sharp\ \Gamma\ \rrbracket \Longrightarrow$
$\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AE\text{-}fst\ v \Rightarrow \{\!|\ z : b1 \mid [[z]^v]^{ce} == ((CE\text{-}fst\ [v]^{ce}))\ |\!\}$

$\mid$ *infer-e-sndI*: $\llbracket$

Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ ;
$\Theta \vdash_{wf}$ (Φ::Φ) ;
Θ; $\mathcal{B}$; $\Gamma \vdash$ $v \Rightarrow$ ⦃ z' : [$b1$, $b2$]^b | c ⦄;
atom z ♯ *AE-snd* v ; *atom* z ♯ Γ ⟧ $\Longrightarrow$
Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash$ *AE-snd* $v \Rightarrow$ ⦃ z : $b2$ | [[z]^v]^{ce} == ((*CE-snd* [v]^{ce})) ⦄

| *infer-e-lenI*: ⟦
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ ;
$\Theta \vdash_{wf}$ (Φ::Φ) ;
Θ; $\mathcal{B}$; $\Gamma \vdash$ $v \Rightarrow$ ⦃ z' : *B-bitvec* | c ⦄;
atom z ♯ *AE-len* v ; *atom* z ♯ Γ⟧ $\Longrightarrow$
Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash$ *AE-len* $v \Rightarrow$ ⦃ z : *B-int* | [[z]^v]^{ce} == ((*CE-len* [v]^{ce})) ⦄

| *infer-e-mvarI*: ⟦
Θ; $\mathcal{B} \vdash_{wf}$ Γ ;
$\Theta \vdash_{wf}$ (Φ::Φ) ;
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ;
(u,τ) $\in$ *setD* Δ ⟧ $\Longrightarrow$
Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash$ *AE-mvar* $u \Rightarrow \tau$

| *infer-e-concatI*: ⟦
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ ;
$\Theta \vdash_{wf}$ (Φ::Φ) ;
Θ; $\mathcal{B}$; $\Gamma \vdash$ $v1 \Rightarrow$ ⦃ $z1$: *B-bitvec* | $c1$ ⦄ ;
Θ; $\mathcal{B}$; $\Gamma \vdash$ $v2 \Rightarrow$ ⦃ $z2$: *B-bitvec* | $c2$ ⦄;
atom $z3$ ♯ (*AE-concat* $v1$ $v2$); *atom* $z3$ ♯ Γ ⟧ $\Longrightarrow$
Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash$ *AE-concat* $v1$ $v2 \Rightarrow$ ⦃ $z3$: *B-bitvec* | [[$z3$]^v]^{ce} == (*CE-concat* [$v1$]^{ce} [$v2$]^{ce}) ⦄

| *infer-e-splitI*: ⟦
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ ;
$\Theta \vdash_{wf}$ (Φ::Φ);
Θ ; $\mathcal{B}$; $\Gamma \vdash$ $v1 \Rightarrow$ ⦃ $z1$: *B-bitvec* | $c1$ ⦄ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash$ $v2 \Leftarrow$ ⦃ $z2$: *B-int* | (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var* $z2$))) == (*CE-val* (*V-lit L-true*)) *AND*
(*CE-op LEq* (*CE-val* (*V-var* $z2$)) (*CE-len* (*CE-val* ($v1$)))) == (*CE-val* (*V-lit L-true*)) ⦄;
atom $z1$ ♯ (*AE-split* $v1$ $v2$); *atom* $z1$ ♯ Γ;
atom $z2$ ♯ (*AE-split* $v1$ $v2$); *atom* $z2$ ♯ Γ;
atom $z3$ ♯ (*AE-split* $v1$ $v2$); *atom* $z3$ ♯ Γ
⟧ $\Longrightarrow$
Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash$ (*AE-split* $v1$ $v2$) $\Rightarrow$ ⦃ $z3$: *B-pair B-bitvec B-bitvec* |
((*CE-val* $v1$) == (*CE-concat* (*CE-fst* (*CE-val* (*V-var* $z3$))) (*CE-snd* (*CE-val* (*V-var* $z3$)))))
AND (((*CE-len* (*CE-fst* (*CE-val* (*V-var* $z3$))))) == (*CE-val* ($v2$))) ⦄

equivariance *infer-e*
nominal-inductive *infer-e*
avoids *infer-e-appI*: x |*infer-e-appPI*: bv | *infer-e-splitI*: $z3$ **and** $z1$ **and** $z2$
proof(*goal-cases*)
case (*1* Θ $\mathcal{B}$ Γ Δ Φ f x b c τ' s' v τ)
moreover hence *atom* x ♯ [f v]^e **using** *fresh-prodN pure-fresh e.fresh* **by** *force*
ultimately show *?case* **unfolding** *fresh-star-def* **using** *fresh-prodN e.fresh pure-fresh* **by** *simp*

next
 case (*2* Θ *B* Γ Δ Φ *f x b c* τ′ *s*′ *v* τ)
 then show *?case* **by** *auto*
next
 case (*3* Θ *B* Γ Δ Φ *b*′ *f bv x b c* τ′ *s*′ *v* τ)
 moreover hence *atom bv* ♯ *AE-appP f b*′ *v* **using** *fresh-prodN pure-fresh e.fresh* **by** *force*
 ultimately show *?case* **unfolding** *fresh-star-def* **using** *fresh-prodN e.fresh pure-fresh fresh-Pair* **by** *auto*
next
 case (*4* Θ *B* Γ Δ Φ *b*′ *f bv x b c* τ′ *s*′ *v* τ)
 then show *?case* **by** *auto*
next
 case (*5* Θ *B* Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
 have *atom z3* ♯ ⦃ *z3* : [*B-bitvec* , *B-bitvec*]^{b} | [*v1*]^{ce} == [[#*1*[[*z3*]^{v}]^{ce}]^{ce} @@ [#*2*[[*z3*]^{v}]^{ce}]^{ce}]^{ce} *AND* [| [#*1*[[*z3*]^{v}]^{ce}]^{ce} |]^{ce} == [*v2*]^{ce} ⦄
  using τ*.fresh* **by** *simp*
 then show *?case* **unfolding** *fresh-star-def fresh-prod7* **using** *wfG-fresh-x2 5* **by** *auto*
next
 case (*6* Θ *B* Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
 then show *?case* **by** *auto*
qed

inductive-cases *infer-e-elims*[*elim*!]:
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Plus v1 v2*) ⇒ ⦃ *z3* : *B-int* | [[*z3*]^{v}]^{ce} == (*CE-op Plus* [*v1*]^{ce} [*v2*]^{ce}) ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op LEq v1 v2*) ⇒ ⦃ *z3* : *B-bool* | [[*z3*]^{v}]^{ce} == (*CE-op LEq* [*v1*]^{ce} [*v2*]^{ce}) ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Plus v1 v2*) ⇒ ⦃ *z3* : *B-int* | *c* ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Plus v1 v2*) ⇒ ⦃ *z3* : *b* | *c* ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op LEq v1 v2*) ⇒ ⦃ *z3* : *b* | *c* ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-app f v*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-val v*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-fst v*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-snd v*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-mvar u*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Plus v1 v2*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op LEq v1 v2*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op LEq v1 v2*) ⇒ ⦃ *z3* : *B-bool* | *c* ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-app f v*) ⇒ τ[*x*::=*v*]$_{\tau v}$
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op opp v1 v2*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-len v*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-len v*) ⇒ ⦃ *z* : *B-int* | [[*z*]^{v}]^{ce} == ((*CE-len* [*v*]^{ce}))⦄
 Θ; Φ; *B*; Γ; Δ ⊢ *AE-concat v1 v2* ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ *AE-concat v1 v2* ⇒ (⦃ *z* : *b* | *c* ⦄)
 Θ; Φ; *B*; Γ; Δ ⊢ *AE-concat v1 v2* ⇒ (⦃ *z* : *B-bitvec* | [[*z*]^{v}]^{ce} == (*CE-concat* [*v1*]^{ce} [*v1*]^{ce}) ⦄)
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-appP f b v*) ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ *AE-split v1 v2* ⇒ τ
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Eq v1 v2*) ⇒ ⦃ *z3* : *b* | *c* ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Eq v1 v2*) ⇒ ⦃ *z3* : *B-bool* | *c* ⦄
 Θ; Φ; *B*; Γ; Δ ⊢ (*AE-op Eq v1 v2*) ⇒ τ
nominal-termination (*eqvt*) **by** *lexicographic-order*

9.5 Statements

inductive *check-s* :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow \tau \Rightarrow$ *bool* (‹ - ; - ; - ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*,*50*,*50*] *50*) **and**
check-branch-s :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* ⇒ *string* ⇒ $\tau \Rightarrow v \Rightarrow$ *branch-s* ⇒ $\tau \Rightarrow$ *bool* (‹ - ; - ; - ; - ; - ; - ; - ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*,*50*,*50*] *50*) **and**
check-branch-list :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* ⇒ (*string* ∗ τ) *list* ⇒ *v* ⇒ *branch-list* ⇒ τ ⇒ *bool* (‹ - ; - ; - ; - ; - ; - ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*,*50*,*50*] *50*) **where**

check-valI: ⟦
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau'$;
$\Theta; \mathcal{B}; \Gamma \vdash \tau' \lesssim \tau$ ⟧ ⟹
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash$ (*AS-val v*) ⇐ τ

| *check-letI*: ⟦
atom x ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *e*, τ);
atom z ♯ (*x*, Θ, Φ, $\mathcal{B}$, Γ, Δ, *e*, τ, *s*);
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash e \Rightarrow \{\!\!\{ z : b \mid c \}\!\!\}$;
Θ; Φ ; $\mathcal{B}$; ((*x*,*b*,*c*[*z*::=*V-var x*]$_v$)#$_\Gamma$Γ) ; Δ ⊢ *s* ⇐ τ
⟧ ⟹
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash$ (*AS-let x e s*) ⇐ τ

| *check-assertI*: ⟦
atom x ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *c*, τ, *s*);
Θ; Φ ; $\mathcal{B}$; ((*x*,*B-bool*,*c*)#$_\Gamma$Γ) ; Δ ⊢ *s* ⇐ τ ;
$\Theta; \mathcal{B}; \Gamma \models c$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$
⟧ ⟹
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash$ (*AS-assert c s*) ⇐ τ

|*check-branch-s-branchI*: ⟦
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\vdash_{wf} \Theta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$;
Θ ; {||} ; *GNil* $\vdash_{wf}$ *const*;
atom x ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *tid*, *cons* , *const*, *v*, τ);
Θ; Φ; $\mathcal{B}$; ((*x*,*b-of const*, ([*v*]^{ce} == [*V-cons tid cons* [*x*]^v]^{ce}) *AND* (*c-of const x*))#$_\Gamma$Γ) ; Δ ⊢ *s* ⇐ τ
⟧ ⟹
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *cons* ; *const* ; *v* ⊢ (*AS-branch cons x s*) ⇐ τ

|*check-branch-list-consI*: ⟦
Θ; Φ; $\mathcal{B}$; Γ; Δ; *tid*; *cons*; *const*; *v* ⊢ *cs* ⇐ τ ;
Θ; Φ; $\mathcal{B}$; Γ; Δ; *tid*; *dclist*; *v* ⊢ *css* ⇐ τ
⟧ ⟹
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; (*cons*,*const*)#*dclist* ; *v* ⊢ *AS-cons cs css* ⇐ τ

|*check-branch-list-finalI*: ⟦
Θ; Φ;$\mathcal{B}$; Γ; Δ; *tid* ; *cons* ; *const* ; *v* ⊢ *cs* ⇐ τ
⟧ ⟹
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; [(*cons*,*const*)] ; *v* ⊢ *AS-final cs* ⇐ τ

| *check-ifI*: $\llbracket$

$\quad$ *atom* $z \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, s1, s2, \tau)$;

$\quad (\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow (\{\!\!| z : B\text{-}bool \mid TRUE |\!\!\}))$;

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow (\{\!\!| z : b\text{-}of\ \tau \mid ([v]^{ce} == [[L\text{-}true]^{v}]^{ce})\ IMP\ (c\text{-}of\ \tau\ z) |\!\!\})$;

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow (\{\!\!| z : b\text{-}of\ \tau \mid ([v]^{ce} == [[L\text{-}false]^{v}]^{ce})\ IMP\ (c\text{-}of\ \tau\ z) |\!\!\})$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash IF\ v\ THEN\ s1\ ELSE\ s2 \Leftarrow \tau$

| *check-let2I*: $\llbracket$

$\quad$ *atom* $x \sharp (\Theta, \Phi, \mathcal{B}, G, \Delta, t, s1, \tau)$;

$\quad \Theta; \Phi ; \mathcal{B} ; G; \Delta \vdash s1 \Leftarrow t$;

$\quad \Theta; \Phi ; \mathcal{B} ; ((x, b\text{-}of\ t, c\text{-}of\ t\ x) \#_{\Gamma} G) ; \Delta \vdash s2 \Leftarrow \tau$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi ; \mathcal{B} ; G ; \Delta \vdash (LET\ x : t = s1\ IN\ s2) \Leftarrow \tau$

| *check-varI*: $\llbracket$

$\quad$ *atom* $u \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \tau', v, \tau)$;

$\quad \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau'$;

$\quad \Theta; \Phi; \mathcal{B}; \Gamma ; ((u,\tau') \#_{\Delta} \Delta) \vdash s \Leftarrow \tau$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (VAR\ u : \tau' = v\ IN\ s) \Leftarrow \tau$

| *check-assignI*: $\llbracket$

$\quad \Theta \vdash_{wf} \Phi$;

$\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;

$\quad (u,\tau) \in setD\ \Delta$;

$\quad \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$;

$\quad \Theta; \mathcal{B}; \Gamma \vdash (\{\!\!| z : B\text{-}unit \mid TRUE |\!\!\}) \lesssim \tau'$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (u ::= v) \Leftarrow \tau'$

| *check-whileI*: $\llbracket$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow \{\!\!| z : B\text{-}bool \mid TRUE |\!\!\}$;

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow \{\!\!| z : B\text{-}unit \mid TRUE |\!\!\}$;

$\quad \Theta; \mathcal{B}; \Gamma \vdash (\{\!\!| z : B\text{-}unit \mid TRUE |\!\!\}) \lesssim \tau'$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash WHILE\ s1\ DO\ \{\ s2\ \} \Leftarrow \tau'$

| *check-seqI*: $\llbracket$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow \{\!\!| z : B\text{-}unit \mid TRUE |\!\!\}$;

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow \tau$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 ;; s2 \Leftarrow \tau$

| *check-caseI*: $\llbracket$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid ; dclist ; v \vdash cs \Leftarrow \tau$;

$\quad (AF\text{-}typedef\ tid\ dclist) \in set\ \Theta$;

$\quad \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{\!\!| z : B\text{-}id\ tid \mid TRUE |\!\!\}$;

$\quad \vdash_{wf} \Theta$

$\rrbracket \Longrightarrow$

$\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}match\ v\ cs \Leftarrow \tau$

equivariance *check-s*

We only need avoidance for cases where a variable is added to a context

nominal-inductive *check-s*
  avoids *check-letI*: *x* **and** *z* | *check-branch-s-branchI*: *x* | *check-let2I*: *x* | *check-varI*: *u* | *check-ifI*: *z*
| *check-assertI*: *x*
proof(*goal-cases*)
  case (*1 x* Θ Φ *ℬ* Γ Δ *e τ z s b c*)
  hence *atom x* ♯ *AS-let x e s* **using** *s-branch-s-branch-list.fresh*(*2*) **by** *auto*
  moreover have *atom z* ♯ *AS-let x e s* **using** *s-branch-s-branch-list.fresh*(*2*) *1 fresh-prod8* **by** *auto*
  then show *?case* **using** *fresh-star-def 1* **by** *force*
next
  case (*3 x* Θ Φ *ℬ* Γ Δ *c τ s*)
  hence *atom x* ♯ *AS-assert c s* **using** *fresh-prodN s-branch-s-branch-list.fresh pure-fresh* **by** *auto*
  then show *?case* **using** *fresh-star-def 3* **by** *force*
next
  case (*5* Θ *ℬ* Γ Δ *τ const x* Φ *tid cons v s*)
  hence *atom x* ♯ *AS-branch cons x s* **using** *fresh-prodN s-branch-s-branch-list.fresh pure-fresh* **by** *auto*

  then show *?case* **using** *fresh-star-def 5* **by** *force*
next
  case (*7 z* Θ Φ *ℬ* Γ Δ *v s1 s2 τ*)
  hence *atom z* ♯ *AS-if v s1 s2* **using** *s-branch-s-branch-list.fresh* **by** *auto*
  then show *?case* **using** *7 fresh-prodN fresh-star-def* **by** *fastforce*
next
  case (*9 x* Θ Φ *ℬ G* Δ *t s1 τ s2*)
  hence *atom x* ♯ *AS-let2 x t s1 s2* **using** *s-branch-s-branch-list.fresh* **by** *auto*
  thus *?case* **using** *fresh-star-def 9* **by** *force*
next
  case (*11 u* Θ Φ *ℬ* Γ Δ *τ′ v τ s*)
  hence *atom u* ♯ *AS-var u τ′ v s* **using** *s-branch-s-branch-list.fresh* **by** *auto*
  then show *?case* **using** *fresh-star-def 11* **by** *force*

qed(*auto*+)

inductive-cases *check-s-elims*[*elim*!]:
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-val v* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-let x e s* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-if v s1 s2* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-let2 x t s1 s2* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-while s1 s2* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-var u t v s* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-seq s1 s2* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-assign u v* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-match v cs* ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ ⊢ *AS-assert c s* ⇐ *τ*

inductive-cases *check-branch-s-elims*[*elim*!]:
  Θ; Φ; *ℬ*; Γ; Δ; *tid* ; *dclist* ; *v* ⊢ (*AS-final cs*) ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ; *tid* ; *dclist* ; *v* ⊢ (*AS-cons cs css*) ⇐ *τ*
  Θ; Φ; *ℬ*; Γ; Δ; *tid* ; *cons* ; *const* ; *v* ⊢ (*AS-branch dc x s*) ⇐ *τ*

9.6 Programs

Type check function bodies

```
inductive check-funtyp :: Θ ⇒ Φ ⇒ ℬ ⇒ fun-typ ⇒ bool ( ‹ - ; - ; - ⊢ - › ) where
  check-funtypI: ⟦
  atom x ♯ (Θ, Φ, B , b );
  Θ; Φ ;  B ; ((x,b,c) #Γ GNil) ; []Δ ⊢ s ⇐ τ
⟧ ⟹
  Θ; Φ ; B ⊢ (AF-fun-typ x b c τ s)

equivariance check-funtyp
nominal-inductive check-funtyp
  avoids check-funtypI: x
proof(goal-cases)
  case (1 x Θ Φ B b c s τ  )
  hence atom x ♯ (AF-fun-typ x b c τ s)  using fun-def.fresh fun-typ-q.fresh fun-typ.fresh by simp
  then show ?case using fresh-star-def 1 fresh-prodN by fastforce
next
  case (2 Θ Φ x b c s τ f)
  then show ?case by auto
qed

inductive check-funtypq :: Θ ⇒ Φ ⇒ fun-typ-q ⇒ bool  ( ‹ - ; - ⊢ - › ) where
  check-fundefq-simpleI: ⟦
  Θ; Φ ; {||} ⊢ (AF-fun-typ x b c t s)
⟧ ⟹
  Θ; Φ ⊢ ((AF-fun-typ-none (AF-fun-typ x b c t s)))

|check-funtypq-polyI: ⟦
  atom bv ♯ (Θ, Φ, (AF-fun-typ x b c t s));
  Θ; Φ; {|bv|} ⊢ (AF-fun-typ x b c t s)
⟧ ⟹
  Θ; Φ ⊢ (AF-fun-typ-some bv (AF-fun-typ x b c t s))

equivariance check-funtypq
nominal-inductive check-funtypq
  avoids check-funtypq-polyI: bv
proof(goal-cases)
  case (1 bv Θ Φ x b c t s )
   hence atom bv ♯ (AF-fun-typ-some bv (AF-fun-typ x b c t s))  using fun-def.fresh fun-typ-q.fresh
fun-typ.fresh by simp
  thus ?case using fresh-star-def 1 fresh-prodN by fastforce
next
  case (2 bv Θ Φ ft )
  then show ?case by auto
qed

inductive check-fundef :: Θ ⇒ Φ ⇒ fun-def ⇒ bool ( ‹ - ; - ⊢ - › ) where
  check-fundefI: ⟦
  Θ; Φ ⊢ ft
⟧ ⟹
  Θ; Φ ⊢ (AF-fundef f ft)
```

equivariance *check-fundef*
nominal-inductive *check-fundef* **.**

Temporarily remove this simproc as it produces untidy eliminations

declare[[*simproc del*: *alpha-lst*]]

inductive-cases *check-funtyp-elims*[*elim*!]:
  check-funtyp Θ Φ *B ft*

inductive-cases *check-funtypq-elims*[*elim*!]:
  check-funtypq Θ Φ (*AF-fun-typ-none* (*AF-fun-typ* *x* *b* *c* τ *s*))
  check-funtypq Θ Φ (*AF-fun-typ-some* *bv* (*AF-fun-typ* *x* *b* *c* τ *s*))

inductive-cases *check-fundef-elims*[*elim*!]:
  check-fundef Θ Φ (*AF-fundef f ftq*)

declare[[*simproc add*: *alpha-lst*]]

nominal-function Δ-*of* :: *var-def list* ⇒ Δ **where**
  Δ-*of* [] = *DNil*
| Δ-*of* ((*AV-def u t v*)#*vs*) = (*u*,*t*) $\#_\Delta$ (Δ-*of vs*)
  apply *auto*
  using *eqvt-def* Δ-*of-graph-aux-def neq-Nil-conv old.prod.exhaust* **apply** *force*
  using *eqvt-def* Δ-*of-graph-aux-def neq-Nil-conv old.prod.exhaust*
  by (*metis var-def.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *check-prog* :: *p* ⇒ τ ⇒ *bool* (‹⊢ - ⇐ -›) **where**
  ⟦
   Θ; Φ; {||}; *GNil* ; Δ-*of* $\mathcal{G}$ ⊢ *s* ⇐ τ
⟧ ⟹ ⊢ (*AP-prog* Θ Φ $\mathcal{G}$ *s*) ⇐ τ

inductive-cases *check-prog-elims*[*elim*!]:
  ⊢ (*AP-prog* Θ Φ $\mathcal{G}$ *s*) ⇐ τ

end

Chapter 10

Operational Semantics

Here we define the operational semantics in terms of a small-step reduction relation.

10.1 Reduction Rules

The store for mutable variables

type-synonym $\delta = (u*v)$ *list*

nominal-function *update-d* :: $\delta \Rightarrow u \Rightarrow v \Rightarrow \delta$ **where**
update-d [] - - = []
| *update-d* $((u',v')\#\delta)$ u v = (*if* $u = u'$ *then* $((u,v)\#\delta)$ *else* $((u',v')\#$ (*update-d* δ u v)))
by(*auto*,*simp add*: *eqvt-def update-d-graph-aux-def* ,*metis neq-Nil-conv old.prod.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

Relates constructor to the branch in the case and binding variable and statement

inductive *find-branch* :: *dc* ⇒ *branch-list* ⇒ *branch-s* ⇒ *bool* **where**
find-branch-finalI: $dc' = dc$ ⟹ *find-branch* dc' (*AS-final* (*AS-branch dc x s*)) (*AS-branch dc x s*)
| *find-branch-branch-eqI*: $dc' = dc$ ⟹ *find-branch* dc' (*AS-cons* (*AS-branch dc x s*) *css*) (*AS-branch dc x s*)
| *find-branch-branch-neqI*: ⟦ $dc \neq dc'$; *find-branch* dc' *css cs* ⟧ ⟹ *find-branch* dc' (*AS-cons* (*AS-branch dc x s*) *css*) *cs*
equivariance *find-branch*
nominal-inductive *find-branch* **.**

inductive-cases *find-branch-elims*[*elim*!]:
find-branch dc (*AS-final* cs') *cs*
find-branch dc (*AS-cons* cs' *css*) *cs*

nominal-function *lookup-branch* :: *dc* ⇒ *branch-list* ⇒ *branch-s option* **where**
lookup-branch dc (*AS-final* (*AS-branch* dc' *x s*)) = (*if* $dc = dc'$ *then* (*Some* (*AS-branch* dc' *x s*)) *else None*)
| *lookup-branch dc* (*AS-cons* (*AS-branch* dc' *x s*) *css*) = (*if* $dc = dc'$ *then* (*Some* (*AS-branch* dc' *x s*)) *else lookup-branch dc css*)
apply(*auto*,*simp add*: *eqvt-def lookup-branch-graph-aux-def*)
by(*metis neq-Nil-conv old.prod.exhaust s-branch-s-branch-list.strong-exhaust*)

nominal-termination (*eqvt*) **by** *lexicographic-order*

Reduction rules

inductive *reduce-stmt* :: Φ ⇒ δ ⇒ s ⇒ δ ⇒ s ⇒ *bool* (‹ - ⊢ ⟨ - , -⟩ ⟶ ⟨ - , -⟩› [*50*, *50*, *50*] *50*)
where
reduce-if-trueI: Φ ⊢ ⟨δ, *AS-if* $[L\text{-}true]^v$ *s1 s2*⟩ ⟶ ⟨δ, *s1*⟩
| *reduce-if-falseI*: Φ ⊢ ⟨δ, *AS-if* $[L\text{-}false]^v$ *s1 s2*⟩ ⟶ ⟨δ, *s2*⟩
| *reduce-let-valI*: Φ ⊢ ⟨δ, *AS-let x* (*AE-val v*) *s*⟩ ⟶ ⟨δ, $s[x::=v]_{sv}$⟩
| *reduce-let-plusI*: Φ ⊢ ⟨δ, *AS-let x* (*AE-op Plus* ((*V-lit* (*L-num n1*))) ((*V-lit* (*L-num n2*)))) *s*⟩ ⟶
⟨δ, *AS-let x* (*AE-val* (*V-lit* (*L-num* (((*n1*)+(*n2*)))))) *s* ⟩
| *reduce-let-leqI*: *b* = (*if* (*n1* ≤ *n2*) *then L-true else L-false*) ⟹
Φ ⊢ ⟨δ, *AS-let x* ((*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)))) *s*⟩ ⟶
⟨δ, *AS-let x* (*AE-val* (*V-lit b*)) *s*⟩
| *reduce-let-eqI*: *b* = (*if* (*n1* = *n2*) *then L-true else L-false*) ⟹
Φ ⊢ ⟨δ, *AS-let x* ((*AE-op Eq* (*V-lit n1*) (*V-lit n2*))) *s*⟩ ⟶
⟨δ, *AS-let x* (*AE-val* (*V-lit b*)) *s*⟩
| *reduce-let-appI*: *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ z b c* τ *s*′))) = *lookup-fun* Φ *f* ⟹

Φ ⊢ ⟨δ, *AS-let x* ((*AE-app f v*)) *s*⟩ ⟶ ⟨δ, *AS-let2 x* $\tau[z::=v]_{\tau v}$ $s'[z::=v]_{sv}$ *s*⟩
| *reduce-let-appPI*: *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ z b c* τ *s*′))) = *lookup-fun* Φ *f* ⟹
Φ ⊢ ⟨δ, *AS-let x* ((*AE-appP f b*′ *v*)) *s*⟩ ⟶ ⟨δ, *AS-let2 x* $\tau[bv::=b']_{\tau b}[z::=v]_{\tau v}$ $s'[bv::=b']_{sb}[z::=v]_{sv}$ *s*⟩
| *reduce-let-fstI*: Φ ⊢ ⟨δ, *AS-let x* (*AE-fst* (*V-pair v1 v2*)) *s*⟩ ⟶ ⟨δ, *AS-let x* (*AE-val v1*) *s*⟩
| *reduce-let-sndI*: Φ ⊢ ⟨δ, *AS-let x* (*AE-snd* (*V-pair v1 v2*)) *s*⟩ ⟶ ⟨δ, *AS-let x* (*AE-val v2*) *s*⟩
| *reduce-let-concatI*: Φ ⊢ ⟨δ, *AS-let x* (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) *s*⟩ ⟶
⟨δ, *AS-let x* (*AE-val* (*V-lit* (*L-bitvec* (*v1*@*v2*)))) *s*⟩
| *reduce-let-splitI*: *split n v* (*v1* , *v2*) ⟹ Φ ⊢ ⟨δ, *AS-let x* (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) *s*⟩ ⟶
⟨δ, *AS-let x* (*AE-val* (*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)))) *s*⟩
| *reduce-let-lenI*: Φ ⊢ ⟨δ, *AS-let x* (*AE-len* (*V-lit* (*L-bitvec v*))) *s*⟩ ⟶
⟨δ, *AS-let x* (*AE-val* (*V-lit* (*L-num* (*int* (*List.length v*))))) *s*⟩
| *reduce-let-mvar*: (*u*,*v*) ∈ *set* δ ⟹ Φ ⊢ ⟨δ, *AS-let x* (*AE-mvar u*) *s*⟩ ⟶ ⟨δ, *AS-let x* (*AE-val v*) *s* ⟩
| *reduce-assert1I*: Φ ⊢ ⟨δ, *AS-assert c* (*AS-val v*)⟩ ⟶ ⟨δ, *AS-val v*⟩
| *reduce-assert2I*: Φ ⊢ ⟨δ, *s* ⟩ ⟶ ⟨ δ′, *s*′⟩ ⟹ Φ ⊢ ⟨δ, *AS-assert c s*⟩ ⟶ ⟨ δ′, *AS-assert c s*′⟩
| *reduce-varI*: *atom u* ♯ δ ⟹ Φ ⊢ ⟨δ, *AS-var u* τ *v s* ⟩ ⟶ ⟨ ((*u*,*v*)#δ) , *s*⟩
| *reduce-assignI*: Φ ⊢ ⟨δ, *AS-assign u v* ⟩ ⟶ ⟨ *update-d* δ *u v* , *AS-val* (*V-lit L-unit*)⟩
| *reduce-seq1I*: Φ ⊢ ⟨δ, *AS-seq* (*AS-val* (*V-lit L-unit*)) *s* ⟩ ⟶ ⟨δ, *s*⟩
| *reduce-seq2I*: ⟦*s1* ≠ *AS-val v* ; Φ ⊢ ⟨δ, *s1*⟩ ⟶ ⟨ δ′, *s1*′⟩ ⟧ ⟹
Φ ⊢ ⟨δ, *AS-seq s1 s2*⟩ ⟶ ⟨ δ′, *AS-seq s1*′ *s2*⟩
| *reduce-let2-valI*: Φ ⊢ ⟨δ, *AS-let2 x t* (*AS-val v*) *s*⟩ ⟶ ⟨δ, $s[x::=v]_{sv}$⟩
| *reduce-let2I*: Φ ⊢ ⟨δ, *s1* ⟩ ⟶ ⟨ δ′, *s1*′⟩ ⟹ Φ ⊢ ⟨δ, *AS-let2 x t s1 s2*⟩ ⟶ ⟨ δ′, *AS-let2 x t s1*′ *s2*⟩

| *reduce-caseI*: ⟦ *Some* (*AS-branch dc x*′ *s*′) = *lookup-branch dc cs* ⟧ ⟹ Φ ⊢ ⟨δ, *AS-match* (*V-cons tyid dc v*) *cs*⟩ ⟶ ⟨δ, $s'[x'::=v]_{sv}$⟩
| *reduce-whileI*: ⟦ *atom x* ♯ (*s1*,*s2*); *atom z* ♯ *x* ⟧ ⟹
Φ ⊢ ⟨δ, *AS-while s1 s2* ⟩ ⟶
⟨δ, *AS-let2 x* (⦃ *z* : *B-bool* | *TRUE* ⦄) *s1* (*AS-if* (*V-var x*) (*AS-seq s2* (*AS-while s1 s2*)) (*AS-val* (*V-lit L-unit*))) ⟩

equivariance *reduce-stmt*

nominal-inductive *reduce-stmt* **.**

inductive-cases *reduce-stmt-elims*[*elim*!]:
  Φ ⊢ ⟨δ, *AS-if* (*V-lit L-true*) *s1 s2*⟩ ⟶ ⟨δ , *s1*⟩
  Φ ⊢ ⟨δ, *AS-if* (*V-lit L-false*) *s1 s2*⟩ ⟶ ⟨δ,*s2*⟩
  Φ ⊢ ⟨δ, *AS-let x* (*AE-val v*) *s*⟩ ⟶ ⟨δ,*s*′⟩
  Φ ⊢ ⟨δ, *AS-let x* (*AE-op Plus* ((*V-lit* (*L-num n1*))) ((*V-lit* (*L-num n2*)))) *s*⟩ ⟶
    ⟨δ, *AS-let x* (*AE-val* (*V-lit* (*L-num* (((*n1*)+(*n2*)))))) *s* ⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)))) *s*⟩ ⟶ ⟨δ, *AS-let x* (*AE-val* (*V-lit b*)) *s*⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-app f v*)) *s*⟩ ⟶ ⟨δ, *AS-let2 x* τ (*subst-sv s*′ *x v*) *s*⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-len v*)) *s*⟩ ⟶ ⟨δ, *AS-let x v*′ *s*⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-concat v1 v2*)) *s*⟩ ⟶ ⟨δ, *AS-let x v*′ *s*⟩
  Φ ⊢ ⟨δ, *AS-seq s1 s2*⟩ ⟶ ⟨ δ′ ,*s*′⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-appP f b v*)) *s*⟩ ⟶ ⟨δ, *AS-let2 x* τ (*subst-sv s*′ *z v*) *s*⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-split v1 v2*)) *s*⟩ ⟶ ⟨δ, *AS-let x v*′ *s*⟩
  Φ ⊢ ⟨δ, *AS-assert c s* ⟩ ⟶ ⟨δ, *s*′⟩
  Φ ⊢ ⟨δ, *AS-let x* ((*AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*)))) *s*⟩ ⟶ ⟨δ, *AS-let x* (*AE-val* (*V-lit b*)) *s*⟩

inductive *reduce-stmt-many* :: Φ ⇒ δ ⇒ *s* ⇒ δ ⇒ *s* ⇒ *bool* (‹- ⊢ ⟨ - , -⟩ ⟶* ⟨ - , -⟩› [*50*, *50*, *50*] *50*) **where**
  reduce-stmt-many-oneI: Φ ⊢ ⟨δ, *s*⟩ ⟶ ⟨δ′, *s*′⟩ ⟹ Φ ⊢ ⟨δ , *s*⟩ ⟶* ⟨δ′, *s*′⟩
| *reduce-stmt-many-manyI*: ⟦ Φ ⊢ ⟨δ, *s*⟩ ⟶ ⟨δ′, *s*′⟩ ; Φ ⊢ ⟨δ′, *s*′⟩ ⟶* ⟨δ′′, *s*′′⟩ ⟧ ⟹ Φ ⊢ ⟨δ, *s*⟩ ⟶* ⟨δ′′, *s*′′⟩

nominal-function *convert-fds* :: *fun-def list* ⇒ (*f*∗*fun-def*) *list* **where**
  convert-fds [] = []
| *convert-fds* ((*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*)))#*fs*) = ((*f*,*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*)))#*convert-fds fs*)
| *convert-fds* ((*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)))#*fs*) = ((*f*,*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)))#*convert-fds fs*)
  apply(*auto*)
  apply (*simp add*: *eqvt-def convert-fds-graph-aux-def*)
  using *fun-def.exhaust fun-typ.exhaust fun-typ-q.exhaust neq-Nil-conv*
  by *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *convert-tds* :: *type-def list* ⇒ (*f*∗*type-def*) *list* **where**
  convert-tds [] = []
| *convert-tds* ((*AF-typedef s dclist*)#*fs*) = ((*s*,*AF-typedef s dclist*)#*convert-tds fs*)
| *convert-tds* ((*AF-typedef-poly s bv dclist*)#*fs*) = ((*s*,*AF-typedef-poly s bv dclist*)#*convert-tds fs*)
  apply(*auto*)
  apply (*simp add*: *eqvt-def convert-tds-graph-aux-def*)
  by (*metis type-def.exhaust neq-Nil-conv*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *reduce-prog* :: *p* ⇒ *v* ⇒ *bool* **where**
  ⟦ *reduce-stmt-many* Φ [] *s* δ (*AS-val v*) ⟧ ⟹ *reduce-prog* (*AP-prog* Θ Φ [] *s*) *v*

10.2 Reduction Typing

Checks that the store is consistent with Δ

inductive *delta-sim* :: Θ ⇒ δ ⇒ Δ ⇒ *bool* (‹- ⊢ - ∼ - › [*50*,*50*] *50*) **where**
delta-sim-nilI: Θ ⊢ [] ∼ $[]_{\Delta}$
| *delta-sim-consI*: ⟦ Θ ⊢ δ ∼ Δ ; Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ ; *u* ∉ *fst* ‘ *set* δ ⟧ ⟹ Θ ⊢ ((*u*,*v*)#δ) ∼ ((*u*,τ)$\#_{\Delta}$Δ)

equivariance *delta-sim*
nominal-inductive *delta-sim* **.**

inductive-cases *delta-sim-elims*[*elim*!]:
Θ ⊢ [] ∼ $[]_{\Delta}$
Θ ⊢ ((*u*,*v*)#*ds*) ∼ (*u*,τ) $\#_{\Delta}$ *D*
Θ ⊢ ((*u*,*v*)#*ds*) ∼ *D*

A typing judgement that combines typing of the statement, the store and the condition that definitions are well-typed

inductive *config-type* :: Θ ⇒ Φ ⇒ Δ ⇒ δ ⇒ *s* ⇒ τ ⇒ *bool* (‹- ; - ; - ⊢ ⟨ - , -⟩ ⇐ - › [*50*, *50*, *50*] *50*) **where**
config-typeI: ⟦ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s* ⇐ τ;
(∀ *fd* ∈ *set* Φ. Θ ; Φ ⊢ *fd*) ;
Θ ⊢ δ ∼ Δ ⟧
⟹ Θ ; Φ ; Δ ⊢ ⟨ δ , *s*⟩ ⇐ τ
equivariance *config-type*
nominal-inductive *config-type* **.**

inductive-cases *config-type-elims* [*elim*!]:
Θ ; Φ ; Δ ⊢ ⟨ δ , *s*⟩ ⇐ τ

nominal-function δ*-of* :: *var-def list* ⇒ δ **where**
δ*-of* [] = []
| δ*-of* ((*AV-def u t v*)#*vs*) = (*u*,*v*) # (δ*-of vs*)
apply *auto*
using *eqvt-def* δ*-of-graph-aux-def neq-Nil-conv old.prod.exhaust* **apply** *force*
using *eqvt-def* δ*-of-graph-aux-def neq-Nil-conv old.prod.exhaust*
by (*metis var-def.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *config-type-prog* :: *p* ⇒ τ ⇒ *bool* (‹ ⊢ ⟨ -⟩ ⇐ -›) **where**
⟦
Θ ; Φ ; Δ*-of* 𝒢 ⊢ ⟨ δ*-of* 𝒢 , *s*⟩ ⇐ τ
⟧ ⟹ ⊢ ⟨ *AP-prog* Θ Φ 𝒢 *s*⟩ ⇐ τ

inductive-cases *config-type-prog-elims* [*elim*!]:
⊢ ⟨ *AP-prog* Θ Φ 𝒢 *s*⟩ ⇐ τ

end
theory *SubstMethods*

imports *IVSubst WellformedL HOL−Eisbach.Eisbach-Tools*
begin

See Eisbach/Examples.thy as well as Eisbach User Manual.

Freshness for various substitution situations. It seems that if undirected and we throw all the facts at them to try to solve in one shot, the automatic methods are *sometimes* unable to handle the different variants, so some guidance is needed. First we split into subgoals using fresh_prodN and intro conjI

The 'add', for example, will be induction premises that will contain freshness facts or freshness conditions from prior obtains

Use different arguments for different things or just lump into one bucket

method *fresh-subst-mth-aux* **uses** *add* = (
  (*match* **conclusion in** *atom z* ♯ (Γ::Γ)[*x*::=*v*]$_{\Gamma v}$ **for** *z x v* Γ ⇒ ‹*auto simp add*: *fresh-subst-gv-if*[*of atom z* Γ *v x*] *add*›)
  | (*match* **conclusion in** *atom z* ♯ (*v'*::*v*)[*x*::=*v*]$_{vv}$ **for** *z x v v'* ⇒ ‹*auto simp add*: *v.fresh fresh-subst-v-if pure-fresh subst-v-v-def add*›)
  | (*match* **conclusion in** *atom z* ♯ (*ce*::*ce*)[*x*::=*v*]$_{cev}$ **for** *z x v ce* ⇒ ‹*auto simp add*: *fresh-subst-v-if subst-v-ce-def add*›)
  | (*match* **conclusion in** *atom z* ♯ (Δ::Δ)[*x*::=*v*]$_{\Delta v}$ **for** *z x v* Δ ⇒ ‹*auto simp add*: *fresh-subst-v-if fresh-subst-dv-if add*›)
  | (*match* **conclusion in** *atom z* ♯ Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ **for** *z x v* Γ' Γ ⇒ ‹*metis add* ›)
  | (*match* **conclusion in** *atom z* ♯ (τ::τ)[*x*::=*v*]$_{\tau v}$ **for** *z x v* τ ⇒ ‹*auto simp add*: *v.fresh fresh-subst-v-if pure-fresh subst-v-τ-def add*›)
  | (*match* **conclusion in** *atom z* ♯ ({||} :: *bv fset*) **for** *z* ⇒ ‹*auto simp add*: *fresh-empty-fset*›)

  | (*auto simp add*: *add x-fresh-b pure-fresh*)
)

method *fresh-mth* **uses** *add* = (
  (*unfold fresh-prodN*, *intro conjI*)?,
  (*fresh-subst-mth-aux add*: *add*)+)

notepad
begin
  fix Γ::Γ **and** *z*::*x* **and** *x*::*x* **and** *v*::*v* **and** Θ::Θ **and** *v'*::*v* **and** *w*::*x* **and** *tyid*::*string* **and** *dc*::*string* **and** *b*::*b* **and** *ce*::*ce* **and** *bv*::*bv*

  assume *as*:*atom z* ♯ (Γ,*v'*,Θ, *v*,*w*,*ce*) ∧ *atom bv* ♯ (Γ,*v'*,Θ, *v*,*w*,*ce*,*b*)

  have *atom z* ♯ Γ[*x*::=*v*]$_{\Gamma v}$
    by (*fresh-mth add*: *as*)

  hence *atom z* ♯ *v'*[*x*::=*v*]$_{vv}$
    by (*fresh-mth add*: *as*)

  hence *atom z* ♯ Γ
    by (*fresh-mth add*: *as*)

  hence *atom z* ♯ Θ
    by (*fresh-mth add*: *as*)

  hence *atom z* ♯ (*CE-val v* == *ce*)[*x*::=*v*]$_{cv}$
    using *as* **by** *auto*

  hence *atom bv* ♯ (*CE-val v* == *ce*)[*x*::=*v*]$_{cv}$

using *as* **by** *auto*

have *atom z* ♯ $(\Theta,\Gamma[x::=v]_{\Gamma v},v'[x::=v]_{vv},w,$ *V-pair v v*, *V-consp tyid dc b v*, $($*CE-val v* $==$ *ce*$)[x::=v]_{cv})$

by (*fresh-mth add*: *as*)

have *atom bv* ♯ $(\Theta,\Gamma[x::=v]_{\Gamma v},v'[x::=v]_{vv},w,$ *V-pair v v*, *V-consp tyid dc b v*)
by (*fresh-mth add*: *as*)

end

end

hide-const *Syntax.dom*

Chapter 11

Refinement Constraint Logic Lemmas

11.1 Lemmas

lemma *wfI-domi*:
  assumes Θ ; $\Gamma \vdash i$
  shows *fst* ' *toSet* $\Gamma \subseteq$ *dom i*
  using *wfI-def toSet.simps assms* **by** *fastforce*

lemma *wfI-lookup*:
  fixes G::Γ **and** b::b
  assumes *Some* (b,c) = *lookup G x* **and** P ; $G \vdash i$ **and** *Some* $s = i\ x$ **and** P ; $B \vdash_{wf} G$
  shows $P \vdash s : b$
proof −
  have $(x,b,c) \in$ *toSet G* **using** *lookup.simps assms*
    using *lookup-in-g* **by** *blast*
  then obtain s' **where** $*$:*Some* $s' = i\ x \wedge$ *wfRCV P* s' *b* **using** *wfI-def assms* **by** *auto*
  hence $s' = s$ **using** *assms* **by** (*metis option.inject*)
  thus *?thesis* **using** $*$ **by** *auto*
qed

lemma *wfI-restrict-weakening*:
  assumes *wfI* Θ Γ' i' **and** i = *restrict-map* i' (*fst* '*toSet* Γ) **and** *toSet* $\Gamma \subseteq$ *toSet* Γ'
  shows Θ ; $\Gamma \vdash i$
proof −
  { **fix** x
    assume $x \in$ *toSet* Γ
    have *case x of* $(x, b, c) \Rightarrow \exists s.$ *Some* $s = i\ x \wedge \Theta \vdash s : b$ **using** *assms wfI-def*
    proof −
      have *case x of* $(x, b, c) \Rightarrow \exists s.$ *Some* $s = i'\ x \wedge \Theta \vdash s : b$
        using ‹$x \in$ *toSet* Γ› *assms wfI-def* **by** *auto*
      then have $\exists s.$ *Some* $s = i$ (*fst x*) $\wedge$ *wfRCV* Θ s (*fst* (*snd x*))
        by (*simp add*: ‹$x \in$ *toSet* Γ› *assms*(*2*) *case-prod-unfold*)
      then show *?thesis*
        by (*simp add*: *case-prod-unfold*)
    qed
  }

thus *?thesis* **using** *wfI-def assms* **by** *auto*
qed

lemma *wfI-suffix*:
fixes $G::\Gamma$
assumes *wfI* P $(G'@G)$ i **and** P ; $B \vdash_{wf}$ G
shows P ; $G \vdash i$
using *wfI-def append-g.simps assms toSet.simps* **by** *simp*

lemma *wfI-replace-inside*:
assumes *wfI* Θ $(\Gamma' @ (x, b, c) \#_\Gamma \Gamma)$ i
shows *wfI* Θ $(\Gamma' @ (x, b, c') \#_\Gamma \Gamma)$ i
using *wfI-def toSet-splitP assms* **by** *simp*

11.2 Existence of evaluation

lemma *eval-l-base*:
$\Theta \vdash [\![l]\!] : (base\text{-}for\text{-}lit\ l)$
apply(*nominal-induct l rule:l.strong-induct*)
using *wfRCV.intros eval-l.simps base-for-lit.simps* **by** *auto+*

lemma *obtain-fresh-bv-dclist*:
fixes *tm::'a::fs*
assumes $(dc, \{\!| x : b \mid c |\!\}) \in set\ dclist$
obtains *bv1::bv* **and** *dclist1 x1 b1 c1* **where** *AF-typedef-poly tyid bv dclist = AF-typedef-poly tyid bv1 dclist1*
$\wedge$ $(dc, \{\!| x1 : b1 \mid c1 |\!\}) \in set\ dclist1 \wedge atom\ bv1 \sharp tm$
proof −
obtain *bv1 dclist1* **where** *AF-typedef-poly tyid bv dclist = AF-typedef-poly tyid bv1 dclist1* $\wedge$ *atom bv1* $\sharp$ *tm*
using *obtain-fresh-bv* **by** *metis*
moreover hence [[*atom bv*]]*lst. dclist* = [[*atom bv1*]]*lst. dclist1* **using** *type-def.eq-iff* **by** *metis*
moreover then obtain *x1 b1 c1* **where** $(dc, \{\!| x1 : b1 \mid c1 |\!\}) \in set\ dclist1$ **using** *td-lookup-eq-iff assms* **by** *metis*
ultimately show *?thesis* **using** *that* **by** *blast*
qed

lemma *obtain-fresh-bv-dclist-b-iff*:
fixes *tm::'a::fs*
assumes $(dc, \{\!| x : b \mid c |\!\}) \in set\ dclist$ **and** *AF-typedef-poly tyid bv dclist* $\in$ *set P* **and** $\vdash_{wf} P$
obtains *bv1::bv* **and** *dclist1 x1 b1 c1* **where** *AF-typedef-poly tyid bv dclist = AF-typedef-poly tyid bv1 dclist1*
$\wedge$ $(dc, \{\!| x1 : b1 \mid c1 |\!\}) \in set\ dclist1 \wedge atom\ bv1 \sharp tm \wedge b[bv::=b']_{bb}=b1[bv1::=b']_{bb}$
proof −
obtain *bv1 dclist1 x1 b1 c1* **where** $*$:*AF-typedef-poly tyid bv dclist = AF-typedef-poly tyid bv1 dclist1* $\wedge$ *atom bv1* $\sharp$ *tm*
$\wedge$ $(dc, \{\!| x1 : b1 \mid c1 |\!\}) \in set\ dclist1$ **using** *obtain-fresh-bv-dclist assms* **by** *metis*

hence *AF-typedef-poly tyid bv1 dclist1* $\in$ *set P* **using** *assms* **by** *metis*

hence $b[bv::=b']_{bb} = b1[bv1::=b']_{bb}$
using *wfTh-typedef-poly-b-eq-iff*[*OF assms(2) assms(1) - - assms(3),of bv1 dclist1 x1 b1 c1 b'*] $*$

by *metis*

from *this that* **show** *?thesis* **using** ∗ **by** *metis*
qed

lemma *eval-v-exist*:
fixes Γ::Γ **and** *v*::*v* **and** *b*::*b*
assumes *P* ; Γ ⊢ *i* **and** P ; B ; $\Gamma \vdash_{wf} v : b$
shows ∃ *s*. *i* ⟦ *v* ⟧ ~ *s* ∧ *P* ⊢ *s* : *b*
using *assms* **proof**(*nominal-induct v arbitrary*: *b rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *eval-l-base eval-v.intros eval-l.simps wfV-elims rcl-val.supp pure-supp* **by** *metis*
next
case (*V-var x*)
then obtain *c* **where** ∗:*Some* (*b*,*c*) = *lookup* Γ *x* **using** *wfV-elims* **by** *metis*
hence *x* ∈ *fst* ‘ *toSet* Γ **using** *lookup-x* **by** *blast*
hence *x* ∈ *dom i* **using** *wfI-domi* **using** *assms* **by** *blast*
then obtain *s* **where** *i x* = *Some s* **by** *auto*
moreover hence *P* ⊢ *s* : *b* **using** *wfRCV.intros wfI-lookup* ∗ *V-var*
by (*metis wfV-wf*)

ultimately show *?case* **using** *eval-v.intros rcl-val.supp b.supp* **by** *metis*
next
case (*V-pair v1 v2*)
then obtain *b1* **and** *b2* **where** ∗:P ; B ; $\Gamma \vdash_{wf} v1 : b1$ ∧ P ; B ; $\Gamma \vdash_{wf} v2 : b2$ ∧ *b* = *B-pair b1 b2* **using** *wfV-elims* **by** *metis*
then obtain *s1* **and** *s2* **where** *eval-v i v1 s1* ∧ *wfRCV P s1 b1* ∧ *eval-v i v2 s2* ∧ *wfRCV P s2 b2* **using** *V-pair* **by** *metis*
thus *?case* **using** *eval-v.intros wfRCV.intros* ∗ **by** *metis*
next
case (*V-cons tyid dc v*)
then obtain *s* **and** *b′*::*b* **and** *dclist* **and** *x*::*x* **and** *c*::*c* **where** (*wfV P B* Γ *v b′*) ∧ *i* ⟦ *v* ⟧ ~ *s* ∧ *P* ⊢ *s* : *b′* ∧ *b* = *B-id tyid* ∧
AF-typedef tyid dclist ∈ *set P* ∧ (*dc*, ⦃ *x* : *b′* | *c* ⦄) ∈ *set dclist* **using** *wfV-elims*(*4*) **by** *metis*
then show *?case* **using** *eval-v.intros*(*4*) *wfRCV.intros*(*5*) *V-cons* **by** *metis*
next
case (*V-consp tyid dc b′ v*)

obtain *b′a*::*b* **and** *bv* **and** *dclist* **and** *x*::*x* **and** *c*::*c* **where** ∗:(*wfV P B* Γ *v* $b'a[bv::=b']_{bb}$) ∧ *b* = *B-app tyid b′* ∧
AF-typedef-poly tyid bv dclist ∈ *set P* ∧ (*dc*, ⦃ *x* : *b′a* | *c* ⦄) ∈ *set dclist* ∧
atom bv ♯ (*P*, *B-app tyid b′*,*B*) **using** *wf-strong-elim*(*1*)[*OF V-consp*(*3*)] **by** *metis*

then obtain *s* **where** ∗∗:*i* ⟦ *v* ⟧ ~ *s* ∧ *P* ⊢ *s* : $b'a[bv::=b']_{bb}$ **using** *V-consp* **by** *auto*

have $\vdash_{wf} P$ **using** *wfX-wfY V-consp* **by** *metis*
then obtain *bv1*::*bv* **and** *dclist1 x1 b1 c1* **where** *3*:*AF-typedef-poly tyid bv dclist* = *AF-typedef-poly tyid bv1 dclist1*
∧ (*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist1* ∧ *atom bv1* ♯ (*P*, *SConsp tyid dc b′ s*, *B-app tyid b′*)
∧ $b'a[bv::=b']_{bb}$ = $b1[bv1::=b']_{bb}$
using ∗ *obtain-fresh-bv-dclist-b-iff* **by** *blast*

have *i* ⟦ *V-consp tyid dc b′ v* ⟧ ~ *SConsp tyid dc b′ s* **using** *eval-v.intros* **by** (*simp add*: ∗∗)

moreover have *P* ⊢ *SConsp tyid dc b′ s* : *B-app tyid b′* **proof**
 show ‹*AF-typedef-poly tyid bv1 dclist1* ∈ *set P*› **using** *3* ∗ **by** *metis*
 show ‹(*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist1*› **using** *3* **by** *auto*
 thus ‹*atom bv1* ♯ (*P*, *SConsp tyid dc b′ s*, *B-app tyid b′*)› **using** ∗ *3 fresh-prodN* **by** *metis*
 show ‹ *P* ⊢ *s* : $b1[bv1::=b']_{bb}$› **using** *3* ∗∗ **by** *auto*
qed

ultimately show *?case* **using** *eval-v.intros wfRCV.intros V-consp* ∗ **by** *auto*
qed

lemma *eval-v-uniqueness*:
 fixes *v*::*v*
 assumes *i* ⟦ *v* ⟧ ~ *s* **and** *i* ⟦ *v* ⟧ ~ *s′*
 shows *s*=*s′*
 using *assms* **proof**(*nominal-induct v arbitrary*: *s s′ rule*:*v.strong-induct*)
 case (*V-lit x*)
 then show *?case* **using** *eval-v-elims eval-l.simps* **by** *metis*
next
 case (*V-var x*)
 then show *?case* **using** *eval-v-elims* **by** (*metis option.inject*)
next
 case (*V-pair v1 v2*)
 obtain *s1* **and** *s2* **where** *s*:*i* ⟦ *v1* ⟧ ~ *s1* ∧ *i* ⟦ *v2* ⟧ ~ *s2* ∧ *s* = *SPair s1 s2* **using** *eval-v-elims V-pair* **by** *metis*
 obtain *s1′* **and** *s2′* **where** *s′*:*i* ⟦ *v1* ⟧ ~ *s1′* ∧ *i* ⟦ *v2* ⟧ ~ *s2′* ∧ *s′* = *SPair s1′ s2′* **using** *eval-v-elims V-pair* **by** *metis*
 then show *?case* **using** *eval-v-elims* **using** *V-pair s s′* **by** *auto*
next
 case (*V-cons tyid dc v1*)
 obtain *sv1* **where** *1*:*i* ⟦ *v1* ⟧ ~ *sv1* ∧ *s* = *SCons tyid dc sv1* **using** *eval-v-elims V-cons* **by** *metis*
 moreover obtain *sv2* **where** *2*:*i* ⟦ *v1* ⟧ ~ *sv2* ∧ *s′* = *SCons tyid dc sv2* **using** *eval-v-elims V-cons* **by** *metis*
 ultimately have *sv1* = *sv2* **using** *V-cons* **by** *auto*
 then show *?case* **using** *1 2* **by** *auto*
next
 case (*V-consp tyid dc b v1*)
 then show *?case* **using** *eval-v-elims* **by** *metis*

qed

lemma *eval-v-base*:
 fixes Γ::Γ **and** *v*::*v* **and** *b*::*b*
 assumes *P* ; Γ ⊢ *i* **and** *P* ; *B* ; Γ $\vdash_{wf}$ *v* : *b* **and** *i* ⟦ *v* ⟧ ~ *s*
 shows *P* ⊢ *s* : *b*
 using *eval-v-exist eval-v-uniqueness assms* **by** *metis*

lemma *eval-e-uniqueness*:
 fixes *e*::*ce*
 assumes *i* ⟦ *e* ⟧ ~ *s* **and** *i* ⟦ *e* ⟧ ~ *s′*

shows *s=s′*
using *assms* **proof**(*nominal-induct e arbitrary*: *s s′ rule*:*ce.strong-induct*)
case (*CE-val x*)
then show *?case* **using** *eval-v-uniqueness eval-e-elims* **by** *metis*
next
case (*CE-op opp x1 x2*)
consider *opp* = *Plus* | *opp* = *LEq* | *opp* = *Eq* **using** *opp.exhaust* **by** *metis*
thus *?case* **proof**(*cases*)
case *1*
hence *a1*:*eval-e i* (*CE-op Plus x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-op Plus x1 x2*) *s′* **using** *CE-op* **by** *auto*
then show *?thesis* **using** *eval-e-elims*(*2*)[*OF a1*] *eval-e-elims*(*2*)[*OF a2*]
CE-op eval-e-plusI
by (*metis rcl-val.eq-iff*(*2*))
next
case *2*
hence *a1*:*eval-e i* (*CE-op LEq x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-op LEq x1 x2*) *s′* **using** *CE-op* **by** *auto*
then show *?thesis* **using** *eval-v-uniqueness eval-e-elims*(*3*)[*OF a1*] *eval-e-elims*(*3*)[*OF a2*]
CE-op eval-e-plusI
by (*metis rcl-val.eq-iff*(*2*))
next
case *3*
hence *a1*:*eval-e i* (*CE-op Eq x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-op Eq x1 x2*) *s′* **using** *CE-op* **by** *auto*
then show *?thesis* **using** *eval-v-uniqueness eval-e-elims*(*4*)[*OF a1*] *eval-e-elims*(*4*)[*OF a2*]
CE-op eval-e-plusI
by (*metis rcl-val.eq-iff*(*2*))
qed
next
case (*CE-concat x1 x2*)
hence *a1*:*eval-e i* (*CE-concat x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-concat x1 x2*) *s′* **using** *CE-concat* **by** *auto*
show *?case* **using** *eval-e-elims*(*7*)[*OF a1*] *eval-e-elims*(*7*)[*OF a2*] *CE-concat eval-e-concatI rcl-val.eq-iff*

proof −
assume ⋀*P*. (⋀*bv1 bv2*. ⟦*s′* = *SBitvec* (*bv1* @ *bv2*); *i* ⟦ *x1* ⟧ ~ *SBitvec bv1* ; *i* ⟦ *x2* ⟧ ~ *SBitvec bv2* ⟧ ⟹ *P*) ⟹ *P*
obtain *bbs* :: *bit list* **and** *bbsa* :: *bit list* **where**
i ⟦ *x2* ⟧ ~ *SBitvec bbs* ∧ *i* ⟦ *x1* ⟧ ~ *SBitvec bbsa* ∧ *SBitvec* (*bbsa* @ *bbs*) = *s′*
by (*metis* ‹⋀*P*. (⋀*bv1 bv2*. ⟦*s′* = *SBitvec* (*bv1* @ *bv2*); *i* ⟦ *x1* ⟧ ~ *SBitvec bv1* ; *i* ⟦ *x2* ⟧ ~ *SBitvec bv2* ⟧ ⟹ *P*) ⟹ *P*›)
then have *s′* = *s*
by (*metis* (*no-types*) ‹⋀*P*. (⋀*bv1 bv2*. ⟦*s* = *SBitvec* (*bv1* @ *bv2*); *i* ⟦ *x1* ⟧ ~ *SBitvec bv1* ; *i* ⟦ *x2* ⟧ ~ *SBitvec bv2* ⟧ ⟹ *P*) ⟹ *P*› ‹⋀*s′ s*. ⟦*i* ⟦ *x1* ⟧ ~ *s* ; *i* ⟦ *x1* ⟧ ~ *s′* ⟧ ⟹ *s* = *s′*› ‹⋀*s′ s*. ⟦*i* ⟦ *x2* ⟧ ~ *s* ; *i* ⟦ *x2* ⟧ ~ *s′* ⟧ ⟹ *s* = *s′*› *rcl-val.eq-iff*(*1*))
then show *?thesis*
by *metis*
qed
next
case (*CE-fst x*)
then show *?case* **using** *eval-v-uniqueness* **by** (*meson eval-e-elims rcl-val.eq-iff*)
next
case (*CE-snd x*)
then show *?case* **using** *eval-v-uniqueness* **by** (*meson eval-e-elims rcl-val.eq-iff*)

next
  case (*CE-len x*)
  then show *?case* **using** *eval-e-elims rcl-val.eq-iff*
  proof −
    obtain *bbs* :: *rcl-val* ⇒ *ce* ⇒ (*x* ⇒ *rcl-val option*) ⇒ *bit list* **where**
      ∀ *x0 x1 x2*. (∃ *v3*. *x0* = *SNum* (*int* (*length v3*)) ∧ *x2* ⟦ *x1* ⟧ ~ *SBitvec v3*) = (*x0* = *SNum* (*int* (*length* (*bbs x0 x1 x2*))) ∧ *x2* ⟦ *x1* ⟧ ~ *SBitvec* (*bbs x0 x1 x2*))
      by *moura*
    then have ∀ *f c r*. ¬ *f* ⟦ [| *c* |]^{ce} ⟧ ~ *r* ∨ *r* = *SNum* (*int* (*length* (*bbs r c f*))) ∧ *f* ⟦ *c* ⟧ ~ *SBitvec* (*bbs r c f*)
      by (*meson eval-e-elims*(*8*))
    then show *?thesis*
      by (*metis* (*no-types*) *CE-len.hyps CE-len.prems*(*1*) *CE-len.prems*(*2*) *rcl-val.eq-iff*(*1*))
  qed

qed

lemma *wfV-eval-bitvec*:
  fixes *v*::*v*
  assumes *P* ; *B* ; Γ $\vdash_{wf}$ *v* : *B-bitvec* **and** *P* ; Γ ⊢ *i*
  shows ∃ *bv*. *eval-v i v* (*SBitvec bv*)
proof −
  obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *wfRCV P s B-bitvec* **using** *eval-v-exist assms* **by** *metis*
  moreover then obtain *bv* **where** *s* = *SBitvec bv* **using** *wfRCV-elims*(*1*)[*of P s*] **by** *metis*
  ultimately show *?thesis* **by** *metis*
qed

lemma *wfV-eval-pair*:
  fixes *v*::*v*
  assumes *P* ; *B* ; Γ $\vdash_{wf}$ *v* : *B-pair b1 b2* **and** *P* ; Γ ⊢ *i*
  shows ∃ *s1 s2*. *eval-v i v* (*SPair s1 s2*)
proof −
  obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *wfRCV P s* (*B-pair b1 b2*) **using** *eval-v-exist assms* **by** *metis*
  moreover then obtain *s1* **and** *s2* **where** *s* = *SPair s1 s2* **using** *wfRCV-elims*(*2*)[*of P s*] **by** *metis*
  ultimately show *?thesis* **by** *metis*
qed

lemma *wfV-eval-int*:
  fixes *v*::*v*
  assumes *P* ; *B* ; Γ $\vdash_{wf}$ *v* : *B-int* **and** *P* ; Γ ⊢ *i*
  shows ∃ *n*. *eval-v i v* (*SNum n*)
proof −
  obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *wfRCV P s* (*B-int*) **using** *eval-v-exist assms* **by** *metis*
  moreover then obtain *n* **where** *s* = *SNum n* **using** *wfRCV-elims*(*3*)[*of P s*] **by** *metis*
  ultimately show *?thesis* **by** *metis*
qed

Well sorted value with a well sorted valuation evaluates

lemma *wfI-wfV-eval-v*:
  fixes *v*::*v* **and** *b*::*b*
  assumes Θ ; *B* ; Γ $\vdash_{wf}$ *v* : *b* **and** *wfI* Θ Γ *i*
  shows ∃ *s*. *i* ⟦ *v* ⟧ ~ *s* ∧ Θ ⊢ *s* : *b*

using *eval-v-exist assms* **by** *auto*

lemma *wfI-wfCE-eval-e*:
fixes *e*::*ce* **and** *b*::*b*
assumes *wfCE P B G e b* **and** *P* ; *G* ⊢ *i*
shows ∃ *s*. *i* ⟦ *e* ⟧ ~ *s* ∧ *P* ⊢ *s* : *b*
using *assms* **proof**(*nominal-induct e arbitrary*: *b rule*: *ce.strong-induct*)
case (*CE-val v*)
obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *P* ⊢ *s* : *b* **using** *wfI-wfV-eval-v*[*of P B G v b i*] *assms wfCE-elims*(*1*) [*of P B G v b*] *CE-val* **by** *auto*
then show *?case* **using** *CE-val eval-e.intros*(*1*)[*of i v s*] **by** *auto*
next
case (*CE-op opp v1 v2*)

consider *opp* =*Plus* | *opp*=*LEq* | *opp*=*Eq* **using** *opp.exhaust* **by** *auto*

thus *?case* **proof**(*cases*)
case *1*
hence *wfCE P B G v1 B-int* ∧ *wfCE P B G v2 B-int* **using** *wfCE-elims*(*2*) *CE-op*

by *blast*
then obtain *s1* **and** *s2* **where** ∗: *eval-e i v1 s1* ∧ *wfRCV P s1 B-int* ∧ *eval-e i v2 s2* ∧ *wfRCV P s2 B-int*
using *wfI-wfV-eval-v CE-op* **by** *metis*
then obtain *n1* **and** *n2* **where** ∗∗:*s2*=*SNum n2* ∧ *s1* = *SNum n1* **using** *wfRCV-elims* **by** *meson*
hence *eval-e i* (*CE-op Plus v1 v2*) (*SNum* (*n1*+*n2*)) **using** *eval-e-plusI* ∗ ∗∗ **by** *simp*
moreover have *wfRCV P* (*SNum* (*n1*+*n2*)) *B-int* **using** *wfRCV.intros* **by** *auto*
ultimately show *?thesis* **using** *1*
using *CE-op.prems*(*1*) *wfCE-elims*(*2*) **by** *blast*
next
case *2*
hence *wfCE P B G v1 B-int* ∧ *wfCE P B G v2 B-int* **using** *wfCE-elims*(*3*) *CE-op*
by *blast*
then obtain *s1* **and** *s2* **where** ∗: *eval-e i v1 s1* ∧ *wfRCV P s1 B-int* ∧ *eval-e i v2 s2* ∧ *wfRCV P s2 B-int*
using *wfI-wfV-eval-v CE-op* **by** *metis*
then obtain *n1* **and** *n2* **where** ∗∗:*s2*=*SNum n2* ∧ *s1* = *SNum n1* **using** *wfRCV-elims* **by** *meson*
hence *eval-e i* (*CE-op LEq v1 v2*) (*SBool* (*n1* ≤ *n2*)) **using** *eval-e-leqI* ∗ ∗∗ **by** *simp*
moreover have *wfRCV P* (*SBool* (*n1*≤*n2*)) *B-bool* **using** *wfRCV.intros* **by** *auto*
ultimately show *?thesis* **using** *2*
using *CE-op.prems wfCE-elims* **by** *metis*
next
case *3*
then obtain *b2* **where** *wfCE P B G v1 b2* ∧ *wfCE P B G v2 b2* **using** *wfCE-elims*(*9*) *CE-op*
by *blast*
then obtain *s1* **and** *s2* **where** ∗: *eval-e i v1 s1* ∧ *wfRCV P s1 b2* ∧ *eval-e i v2 s2* ∧ *wfRCV P s2 b2*
using *wfI-wfV-eval-v CE-op* **by** *metis*
hence *eval-e i* (*CE-op Eq v1 v2*) (*SBool* (*s1* = *s2*)) **using** *eval-e-leqI* ∗
by (*simp add*: *eval-e-eqI*)
moreover have *wfRCV P* (*SBool* (*s1* = *s2*)) *B-bool* **using** *wfRCV.intros* **by** *auto*
ultimately show *?thesis* **using** *3*

```
      using CE-op.prems wfCE-elims by metis
  qed
next
  case (CE-concat v1 v2)
  then obtain s1 and s2 where *:b = B-bitvec ∧ eval-e i v1 s1 ∧ eval-e i v2 s2 ∧
      wfRCV P s1 B-bitvec ∧ wfRCV P s2 B-bitvec using
    CE-concat
    by (meson wfCE-elims(6))
  thus ?case using  eval-e-concatI wfRCV.intros(1) wfRCV-elims
  proof −
    obtain bbs :: type-def list ⇒ rcl-val ⇒ bit list where
      ∀ ts s. ¬ ts ⊢ s : B-bitvec ∨ s = SBitvec (bbs ts s)
      using wfRCV-elims(1) by moura
    then show ?thesis
      by (metis (no-types) local.∗ wfRCV-BBitvecI eval-e-concatI)
  qed
next
  case (CE-fst v1)
  thus ?case using  eval-e-fstI  wfRCV.intros wfCE-elims wfI-wfV-eval-v
    by (metis wfRCV-elims(2) rcl-val.eq-iff(4))
next
  case (CE-snd v1)
  thus ?case using  eval-e-sndI  wfRCV.intros wfCE-elims wfI-wfV-eval-v
    by (metis wfRCV-elims(2) rcl-val.eq-iff(4))
next
  case (CE-len x)
  thus ?case using  eval-e-lenI  wfRCV.intros wfCE-elims wfI-wfV-eval-v wfV-eval-bitvec
    by (metis wfRCV-elims(1))
qed

lemma eval-e-exist:
  fixes Γ::Γ and e::ce
  assumes P ; Γ  ⊢ i and P ;  B ; Γ  ⊢wf  e : b
  shows ∃ s. i ⟦ e ⟧ ~ s
  using assms proof(nominal-induct e arbitrary: b rule:ce.strong-induct)
  case (CE-val v)
  then show ?case using eval-v-exist wfCE-elims eval-e.intros by metis
next
  case (CE-op op v1 v2)

  show ?case proof(rule opp.exhaust)
    assume ‹op = Plus›
    hence P ;  B ; Γ ⊢wf v1 : B-int ∧ P ;  B ; Γ ⊢wf v2 : B-int ∧ b = B-int using wfCE-elims CE-op
by metis
     then obtain n1 and n2 where eval-e i v1 (SNum n1) ∧ eval-e i v2 (SNum n2) using CE-op
eval-v-exist wfV-eval-int
      by (metis wfI-wfCE-eval-e wfRCV-elims(3))
    then show ‹∃ a. eval-e i (CE-op op v1 v2) a› using eval-e-plusI[of i v1 - v2] ‹op=Plus› by auto
  next
    assume ‹op = LEq›
    hence P ;  B ; Γ ⊢wf v1 : B-int ∧ P ;  B ; Γ ⊢wf v2 : B-int ∧ b = B-bool using wfCE-elims CE-op
by metis
```

then obtain *n1* **and** *n2* **where** *eval-e i v1* (*SNum n1*) ∧ *eval-e i v2* (*SNum n2*) **using** *CE-op eval-v-exist wfV-eval-int*
by (*metis wfI-wfCE-eval-e wfRCV-elims*(*3*))
then show ‹∃ *a*. *eval-e i* (*CE-op op v1 v2*) *a*› **using** *eval-e-leqI*[*of i v1 - v2*] *eval-v-exist* ‹*op*=*LEq*› *CE-op* **by** *auto*
next
assume ‹*op* = *Eq*›
then obtain *b1* **where** P ; B ; $\Gamma \vdash_{wf}$ *v1* : *b1* ∧ P ; B ; $\Gamma \vdash_{wf}$ *v2* : *b1* ∧ *b* = *B-bool* **using** *wfCE-elims CE-op* **by** *metis*
then obtain *s1* **and** *s2* **where** *eval-e i v1 s1* ∧ *eval-e i v2 s2* **using** *CE-op eval-v-exist wfV-eval-int*

by (*metis wfI-wfCE-eval-e wfRCV-elims*(*3*))
then show ‹∃ *a*. *eval-e i* (*CE-op op v1 v2*) *a*› **using** *eval-e-eqI*[*of i v1 - v2*] *eval-v-exist* ‹*op*=*Eq*› *CE-op* **by** *auto*
qed
next
case (*CE-concat v1 v2*)
then obtain *bv1* **and** *bv2* **where** *eval-e i v1* (*SBitvec bv1*) ∧ *eval-e i v2* (*SBitvec bv2*)
using *wfV-eval-bitvec wfCE-elims*(*6*)
by (*meson eval-e-elims*(*7*) *wfI-wfCE-eval-e*)
then show *?case* **using** *eval-e.intros* **by** *metis*
next
case (*CE-fst ce*)
then obtain *b2* **where** *b*:P ; B ; $\Gamma \vdash_{wf}$ *ce* : *B-pair b b2* **using** *wfCE-elims* **by** *metis*
then obtain *s* **where** *s*:*i* ⟦ *ce* ⟧ ~ *s* **using** *CE-fst* **by** *auto*
then obtain *s1* **and** *s2* **where** *s* = (*SPair s1 s2*) **using** *eval-e-elims*(*4*) *CE-fst wfI-wfCE-eval-e*[*of P B Γ ce B-pair b b2 i*,*OF b*] *wfRCV-elims*(*2*)[*of P s b b2*]
by (*metis eval-e-uniqueness*)
then show *?case* **using** *s eval-e.intros* **by** *metis*
next
case (*CE-snd ce*)
then obtain *b1* **where** *b*:P ; B ; $\Gamma \vdash_{wf}$ *ce* : *B-pair b1 b* **using** *wfCE-elims* **by** *metis*
then obtain *s* **where** *s*:*i* ⟦ *ce* ⟧ ~ *s* **using** *CE-snd* **by** *auto*
then obtain *s1* **and** *s2* **where** *s* = (*SPair s1 s2*)
using *eval-e-elims*(*5*) *CE-snd wfI-wfCE-eval-e*[*of P B Γ ce B-pair b1 b i*,*OF b*] *wfRCV-elims*(*2*)[*of P s b b1*]
eval-e-uniqueness
by (*metis wfRCV-elims*(*2*))
then show *?case* **using** *s eval-e.intros* **by** *metis*
next
case (*CE-len v1*)
then obtain *bv1* **where** *eval-e i v1* (*SBitvec bv1*)
using *wfV-eval-bitvec CE-len wfCE-elims eval-e-uniqueness*
by (*metis eval-e-elims*(*7*) *wfCE-concatI wfI-wfCE-eval-e*)
then show *?case* **using** *eval-e.intros* **by** *metis*
qed

lemma *eval-c-exist*:
fixes Γ::Γ **and** *c*::*c*
assumes P ; $\Gamma \vdash i$ **and** P ; B ; $\Gamma \vdash_{wf}$ *c*
shows ∃ *s*. *i* ⟦ *c* ⟧ ~ *s*
using *assms* **proof**(*nominal-induct c rule*: *c.strong-induct*)

case *C-true*
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case *C-false*
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-conj c1 c2*)
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-disj x1 x2*)
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-not x*)
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-imp x1 x2*)
then show *?case* **using** *eval-c.intros eval-e-exist wfC-elims* **by** *metis*
next
case (*C-eq x1 x2*)
then show *?case* **using** *eval-c.intros eval-e-exist wfC-elims* **by** *metis*
qed

lemma *eval-c-uniqueness*:
fixes *c*::*c*
assumes i ⟦ c ⟧ ~ s **and** i ⟦ c ⟧ ~ s'
shows $s=s'$
using *assms* **proof**(*nominal-induct c arbitrary*: *s s′ rule*:*c.strong-induct*)
case *C-true*
then show *?case* **using** *eval-c-elims* **by** *metis*
next
case *C-false*
then show *?case* **using** *eval-c-elims* **by** *metis*
next
case (*C-conj x1 x2*)
then show *?case* **using** *eval-c-elims*(*3*) **by** (*metis* (*full-types*))
next
case (*C-disj x1 x2*)
then show *?case* **using** *eval-c-elims*(*4*) **by** (*metis* (*full-types*))
next
case (*C-not x*)
then show *?case* **using** *eval-c-elims*(*6*) **by** (*metis* (*full-types*))
next
case (*C-imp x1 x2*)
then show *?case* **using** *eval-c-elims*(*5*) **by** (*metis* (*full-types*))
next
case (*C-eq x1 x2*)
then show *?case* **using** *eval-e-uniqueness eval-c-elims*(*7*) **by** *metis*
qed

lemma *wfI-wfC-eval-c*:
fixes *c*::*c*
assumes *wfC P B G c* **and** P ; $G \vdash i$

shows $\exists s.\ i\ [\![\ c\ ]\!]\ \sim\ s$
using *assms* **proof**(*nominal-induct c rule*: *c.strong-induct*)
qed(*metis wfC-elims wfI-wfCE-eval-e eval-c.intros*)+

11.3 Satisfiability

lemma *satis-reflI*:
fixes *c*::*c*
assumes $i \models ((x,\ b,\ c)\ \#_\Gamma\ G)$
shows $i \models c$
using *assms* **by** *auto*

lemma *is-satis-mp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes $i \models$ (*c1 IMP c2*) **and** $i \models$ *c1*
shows $i \models$ *c2*
using *assms* **proof** −
have *eval-c i* (*c1 IMP c2*) *True* **using** *is-satis.simps* **using** *assms* **by** *blast*
then obtain *b1* **and** *b2* **where** *True* = (*b1* ⟶ *b2*) ∧ *eval-c i c1 b1* ∧ *eval-c i c2 b2*
using *eval-c-elims*(*5*) **by** *metis*
moreover have *eval-c i c1 True* **using** *is-satis.simps* **using** *assms* **by** *blast*
moreover have *b1* = *True* **using** *calculation eval-c-uniqueness* **by** *blast*
ultimately have *eval-c i c2 True* **by** *auto*
thus *?thesis* **using** *is-satis.intros* **by** *auto*
qed

lemma *is-satis-imp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes $i \models$ *c1* ⟶ $i \models$ *c2* **and** $i\ [\![\ c1\ ]\!]\ \sim\ b1$ **and** $i\ [\![\ c2\ ]\!]\ \sim\ b2$
shows $i \models$ (*c1 IMP c2*)
proof(*cases b1*)
case *True*
hence $i \models$ *c2* **using** *assms is-satis.simps* **by** *simp*
hence *b2* = *True* **using** *is-satis.simps assms*
using *eval-c-uniqueness* **by** *blast*
then show *?thesis* **using** *eval-c-impI is-satis.simps assms* **by** *force*
next
case *False*
then show *?thesis* **using** *assms eval-c-impI is-satis.simps* **by** *metis*
qed

lemma *is-satis-iff*:
$i \models G$ = (∀ *x b c*. (*x*,*b*,*c*) ∈ *toSet G* ⟶ $i \models c$)
by(*induct G*,*auto*)

lemma *is-satis-g-append*:
$i \models$ (*G1*@*G2*) = ($i \models$ *G1* ∧ $i \models$ *G2*)
using *is-satis-g.simps is-satis-iff* **by** *auto*

11.4 Substitution for Evaluation

lemma *eval-v-i-upd*:
fixes *v::v*
assumes *atom x ♯ v* **and** *i* ⟦ *v* ⟧ ~ *s'*
shows *eval-v* ((*i* (*x* ↦*s*))) *v s'*
using *assms* **proof**(*nominal-induct v arbitrary*: *s' rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **by** (*metis eval-v-elims*(*1*) *eval-v-litI*)
next
case (*V-var y*)
then obtain *s* **where** ∗: *Some s* = *i y* ∧ *s*=*s'* **using** *eval-v-elims* **by** *metis*
moreover have *x* ≠ *y* **using** ‹*atom x ♯ V-var y*› *v.supp* **by** *simp*
ultimately have (*i* (*x* ↦*s*)) *y* = *Some s*
by (*simp add*: ‹*Some s* = *i y* ∧ *s* = *s'*›)
then show *?case* **using** *eval-v-varI* ∗ ‹*x* ≠ *y*›
by (*simp add*: *eval-v.eval-v-varI*)
next
case (*V-pair v1 v2*)
hence *atom x ♯ v1* ∧ *atom x ♯ v2* **using** *v.supp* **by** *simp*
moreover obtain *s1* **and** *s2* **where** ∗: *eval-v i v1 s1* ∧ *eval-v i v2 s2* ∧ *s'* = *SPair s1 s2* **using** *eval-v-elims V-pair* **by** *metis*
ultimately have *eval-v* ((*i* (*x* ↦*s*))) *v1 s1* ∧ *eval-v* ((*i* (*x* ↦*s*))) *v2 s2* **using** *V-pair* **by** *blast*
thus *?case* **using** *eval-v-pairI* ∗ **by** *meson*
next
case (*V-cons tyid dc v1*)
hence *atom x ♯ v1* **using** *v.supp* **by** *simp*
moreover obtain *s1* **where** ∗: *eval-v i v1 s1* ∧ *s'* = *SCons tyid dc s1* **using** *eval-v-elims V-cons* **by** *metis*
ultimately have *eval-v* ((*i* (*x* ↦*s*))) *v1 s1* **using** *V-cons* **by** *blast*
thus *?case* **using** *eval-v-consI* ∗ **by** *meson*
next
case (*V-consp tyid dc b1 v1*)

hence *atom x ♯ v1* **using** *v.supp* **by** *simp*
moreover obtain *s1* **where** ∗: *eval-v i v1 s1* ∧ *s'* = *SConsp tyid dc b1 s1* **using** *eval-v-elims V-consp* **by** *metis*
ultimately have *eval-v* ((*i* (*x* ↦*s*))) *v1 s1* **using** *V-consp* **by** *blast*
thus *?case* **using** *eval-v-conspI* ∗ **by** *meson*
qed

lemma *eval-e-i-upd*:
fixes *e::ce*
assumes *i* ⟦ *e* ⟧ ~ *s'* **and** *atom x ♯ e*
shows (*i* (*x* ↦*s*)) ⟦ *e* ⟧ ~ *s'*
using *assms* **apply**(*induct rule*: *eval-e.induct*) **using** *eval-v-i-upd eval-e-elims*
by (*meson ce.fresh eval-e.intros*)+

lemma *eval-c-i-upd*:
fixes *c::c*
assumes *i* ⟦ *c* ⟧ ~ *s'* **and** *atom x ♯ c*
shows ((*i* (*x* ↦*s*))) ⟦ *c* ⟧ ~ *s'*
using *assms* **proof**(*induct rule*:*eval-c.induct*)

```
  case (eval-c-eqI i e1 sv1 e2 sv2)
  then show ?case using RCLogic.eval-c-eqI eval-e-i-upd c.fresh by metis
qed(simp add: eval-c.intros)+

lemma subst-v-eval-v[simp]:
  fixes v::v and v'::v
  assumes i ⟦ v ⟧ ~  s and i ⟦ (v'[x::=v]_vv) ⟧ ~ s'
  shows (i ( x ↦ s )) ⟦ v' ⟧ ~ s'
  using assms proof(nominal-induct v' arbitrary: s' rule:v.strong-induct)
  case (V-lit x)
  then show ?case using subst-vv.simps
    by (metis eval-v-elims(1) eval-v-litI)
next
  case (V-var x')
  then show ?case proof(cases x=x')
    case True
    hence (V-var x')[x::=v]_vv = v using subst-vv.simps by auto
    then show ?thesis using V-var eval-v-elims eval-v-varI eval-v-uniqueness True
      by (simp add: eval-v.eval-v-varI)
  next
    case False
    hence atom x ♯ (V-var x') by simp
    then show ?thesis using eval-v-i-upd False V-var by fastforce
  qed
next
  case (V-pair v1 v2)
  then obtain s1 and s2 where *:eval-v i (v1[x::=v]_vv) s1 ∧ eval-v i (v2[x::=v]_vv) s2 ∧ s' = SPair
s1 s2 using V-pair eval-v-elims subst-vv.simps by metis
  hence (i ( x ↦ s )) ⟦ v1 ⟧ ~ s1 ∧ (i ( x ↦ s )) ⟦ v2 ⟧ ~ s2 using V-pair by metis
  thus  ?case  using eval-v-pairI subst-vv.simps * V-pair by metis
next
  case (V-cons tyid dc v1)
  then obtain s1 where eval-v i (v1[x::=v]_vv) s1 using eval-v-elims subst-vv.simps by metis
  thus ?case  using eval-v-consI V-cons
    by (metis eval-v-elims subst-vv.simps)
next
  case (V-consp tyid dc b1 v1)

  then obtain s1 where *:eval-v i (v1[x::=v]_vv) s1 ∧ s' = SConsp tyid dc b1 s1 using eval-v-elims
subst-vv.simps by metis
  hence i ( x ↦ s ) ⟦ v1 ⟧ ~ s1 using V-consp by metis
  thus ?case  using * eval-v-conspI by metis
qed

lemma subst-e-eval-v[simp]:
  fixes y::x and e::ce and v::v and e'::ce
  assumes  i ⟦ e' ⟧ ~  s' and e'=(e[y::=v]_cev) and i ⟦ v ⟧ ~  s
  shows (i ( y ↦ s )) ⟦ e ⟧ ~ s'
  using assms proof(induct arbitrary: e rule: eval-e.induct)
  case (eval-e-valI i v1 sv)
  then obtain v1' where *:e = CE-val v1' ∧ v1 = v1'[y::=v]_vv
    using assms by(nominal-induct e rule:ce.strong-induct,simp+)
```

hence *eval-v i* ($v1'[y::=v]_{vv}$) *sv* **using** *eval-e-valI* **by** *simp*
hence *eval-v* (*i* (*y* ↦ *s*)) *v1' sv* **using** *subst-v-eval-v eval-e-valI* **by** *simp*
then show *?case* **using** *RCLogic.eval-e-valI* ∗ **by** *meson*
next
case (*eval-e-plusI i v1 n1 v2 n2*)
then obtain *v1'* **and** *v2'* **where** ∗:*e* = *CE-op Plus v1' v2'* ∧ *v1* = $v1'[y::=v]_{cev}$ ∧ *v2* = $v2'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) (*SNum n1*) ∧ *eval-e i* ($v2'[y::=v]_{cev}$) (*SNum n2*) **using** *eval-e-plusI* **by** *simp*
hence *eval-e* (*i* (*y* ↦ *s*)) *v1'* (*SNum n1*) ∧ *eval-e* (*i* (*y* ↦ *s*)) *v2'* (*SNum n2*) **using** *subst-v-eval-v eval-e-plusI*
using *local*.∗ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-plusI* ∗ **by** *meson*
next
case (*eval-e-leqI i v1 n1 v2 n2*)
then obtain *v1'* **and** *v2'* **where** ∗:*e* = *CE-op LEq v1' v2'* ∧ *v1* = $v1'[y::=v]_{cev}$ ∧ *v2* = $v2'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) (*SNum n1*) ∧ *eval-e i* ($v2'[y::=v]_{cev}$) (*SNum n2*) **using** *eval-e-leqI* **by** *simp*
hence *eval-e* (*i* (*y* ↦ *s*)) *v1'* (*SNum n1*) ∧ *eval-e* (*i* (*y* ↦ *s*)) *v2'* (*SNum n2*) **using** *subst-v-eval-v eval-e-leqI*
using ∗ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-leqI* ∗ **by** *meson*
next
case (*eval-e-eqI i v1 n1 v2 n2*)
then obtain *v1'* **and** *v2'* **where** ∗:*e* = *CE-op Eq v1' v2'* ∧ *v1* = $v1'[y::=v]_{cev}$ ∧ *v2* = $v2'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) *n1* ∧ *eval-e i* ($v2'[y::=v]_{cev}$) *n2* **using** *eval-e-eqI* **by** *simp*
hence *eval-e* (*i* (*y* ↦ *s*)) *v1' n1* ∧ *eval-e* (*i* (*y* ↦ *s*)) *v2' n2* **using** *subst-v-eval-v eval-e-eqI*
using ∗ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-eqI* ∗ **by** *meson*
next
case (*eval-e-fstI i v1 s1 s2*)
then obtain *v1'* **and** *v2'* **where** ∗:*e* = *CE-fst v1'* ∧ *v1* = $v1'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) (*SPair s1 s2*) **using** *eval-e-fstI* **by** *simp*
hence *eval-e* (*i* (*y* ↦ *s*)) *v1'* (*SPair s1 s2*) **using** *eval-e-fstI* ∗ **by** *metis*
then show *?case* **using** *RCLogic.eval-e-fstI* ∗ **by** *meson*
next
case (*eval-e-sndI i v1 s1 s2*)
then obtain *v1'* **and** *v2'* **where** ∗:*e* = *CE-snd v1'* ∧ *v1* = $v1'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) (*SPair s1 s2*) **using** *eval-e-sndI* **by** *simp*
hence *eval-e* (*i* (*y* ↦ *s*)) *v1'* (*SPair s1 s2*) **using** *subst-v-eval-v eval-e-sndI* ∗ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-sndI* ∗ **by** *meson*
next
case (*eval-e-concatI i v1 bv1 v2 bv2*)
then obtain *v1'* **and** *v2'* **where** ∗:*e* = *CE-concat v1' v2'* ∧ *v1* = $v1'[y::=v]_{cev}$ ∧ *v2* = $v2'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) (*SBitvec bv1*) ∧ *eval-e i* ($v2'[y::=v]_{cev}$) (*SBitvec bv2*) **using** *eval-e-concatI* **by** *simp*
moreover hence *eval-e* (*i* (*y* ↦ *s*)) *v1'* (*SBitvec bv1*) ∧ *eval-e* (*i* (*y* ↦ *s*)) *v2'* (*SBitvec bv2*)

using *subst-v-eval-v eval-e-concatI* ∗ **by** *blast*
ultimately show *?case* **using** *RCLogic.eval-e-concatI* ∗ *eval-v-uniqueness* **by** (*metis eval-e-concatI.hyps*(*1*))
next
case (*eval-e-lenI i v1 bv*)
then obtain *v1'* **where** ∗:$e = CE\text{-}len\ v1' \wedge v1 = v1'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* ($v1'[y::=v]_{cev}$) (*SBitvec bv*) **using** *eval-e-lenI* **by** *simp*
hence *eval-e* (*i* (*y* ↦ *s*)) *v1'* (*SBitvec bv*) **using** *subst-v-eval-v eval-e-lenI* ∗ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-lenI* ∗ **by** *meson*
qed

lemma *subst-c-eval-v*[*simp*]:
fixes *v*::*v* **and** *c* :: *c*
assumes *i* ⟦ *v* ⟧ ~ *s* **and** *i* ⟦ $c[x::=v]_{cv}$ ⟧ ~ *s1* **and**
(*i* (*x* ↦ *s*)) ⟦ *c* ⟧ ~ *s2*
shows *s1* = *s2*
using *assms* **proof**(*nominal-induct c arbitrary*: *s1 s2 rule*: *c.strong-induct*)
case *C-true*
hence *s1* = *True* ∧ *s2* = *True* **using** *eval-c-elims subst-cv.simps* **by** *auto*
then show *?case* **by** *auto*
next
case *C-false*
hence *s1* = *False* ∧ *s2* = *False* **using** *eval-c-elims subst-cv.simps* **by** *metis*
then show *?case* **by** *auto*
next
case (*C-conj c1 c2*)
hence ∗:*eval-c i* ($c1[x::=v]_{cv}$ *AND* $c2[x::=v]_{cv}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* ∧ *s12*)) ∧ *eval-c i* $c1[x::=v]_{cv}$ *s11* ∧ *eval-c i* $c2[x::=v]_{cv}$ *s12* **using**
eval-c-elims(*3*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ *eval-c* (*i* (*x* ↦ *s*)) *c2 s22* ∧ (*s2* = (*s21* ∧ *s22*)) **using**
eval-c-elims(*3*) *C-conj* **by** *metis*
ultimately show *?case* **using** *C-conj* **by** (*meson eval-c-elims*)
next
case (*C-disj c1 c2*)
hence ∗:*eval-c i* ($c1[x::=v]_{cv}$ *OR* $c2[x::=v]_{cv}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* ∨ *s12*)) ∧ *eval-c i* $c1[x::=v]_{cv}$ *s11* ∧ *eval-c i* $c2[x::=v]_{cv}$ *s12* **using**
eval-c-elims(*4*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ *eval-c* (*i* (*x* ↦ *s*)) *c2 s22* ∧ (*s2* = (*s21* ∨ *s22*)) **using**
eval-c-elims(*4*) *C-disj* **by** *metis*
ultimately show *?case* **using** *C-disj* **by** (*meson eval-c-elims*)
next
case (*C-not c1*)
then obtain *s11* **where** (*s1* = (¬ *s11*)) ∧ *eval-c i* $c1[x::=v]_{cv}$ *s11* **using**
eval-c-elims(*6*) **by** (*metis subst-cv.simps*(*7*))
moreover obtain *s21* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ (*s2* = (¬*s21*)) **using**
eval-c-elims(*6*) *C-not* **by** *metis*
ultimately show *?case* **using** *C-not* **by** (*meson eval-c-elims*)
next

case (*C-imp c1 c2*)
hence ∗:*eval-c i* (*c1*$[x::=v]_{cv}$ *IMP c2*$[x::=v]_{cv}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* ⟶ *s12*)) ∧ *eval-c i c1*$[x::=v]_{cv}$ *s11* ∧ *eval-c i c2*$[x::=v]_{cv}$ *s12* **using**
eval-c-elims(*5*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ *eval-c* (*i* (*x* ↦ *s*)) *c2 s22* ∧ (*s2* = (*s21* ⟶ *s22*)) **using**
eval-c-elims(*5*) *C-imp* **by** *metis*
ultimately show *?case* **using** *C-imp* **by** (*meson eval-c-elims*)
next
case (*C-eq e1 e2*)
hence ∗:*eval-c i* (*e1*$[x::=v]_{cev}$ == *e2*$[x::=v]_{cev}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* = *s12*)) ∧ *eval-e i* (*e1*$[x::=v]_{cev}$) *s11* ∧ *eval-e i* (*e2*$[x::=v]_{cev}$) *s12* **using**
eval-c-elims(*7*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-e* (*i* (*x* ↦ *s*)) *e1 s21* ∧ *eval-e* (*i* (*x* ↦ *s*)) *e2 s22* ∧ (*s2* = (*s21* = *s22*)) **using**
eval-c-elims(*7*) *C-eq* **by** *metis*
ultimately show *?case* **using** *C-eq subst-e-eval-v* **by** (*metis eval-e-uniqueness*)
qed

lemma *wfI-upd*:
assumes *wfI* Θ Γ *i* **and** *wfRCV* Θ *s b* **and** *wfG* Θ *B* ((*x*, *b*, *c*) $\#_\Gamma$ Γ)
shows *wfI* Θ ((*x*, *b*, *c*) $\#_\Gamma$ Γ) (*i*(*x* ↦ *s*))
proof(*subst wfI-def*,*rule*)
fix *xa*
assume *as*:*xa* ∈ *toSet* ((*x*, *b*, *c*) $\#_\Gamma$ Γ)

then obtain *x1*::*x* **and** *b1*::*b* **and** *c1*::*c* **where** *xa*: *xa* = (*x1*,*b1*,*c1*) **using** *toSet.simps*
using *prod-cases3* **by** *blast*

have ∃ *sa*. *Some sa* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *sa b1* **proof**(*cases x*=*x1*)
case *True*
hence *b*=*b1* **using** *as xa wfG-unique assms* **by** *metis*
hence *Some s* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *s b1* **using** *assms True* **by** *simp*
then show *?thesis* **by** *auto*
next
case *False*
hence (*x1*,*b1*,*c1*) ∈ *toSet* Γ **using** *xa as* **by** *auto*
then obtain *sa* **where** *Some sa* = *i x1* ∧ *wfRCV* Θ *sa b1* **using** *assms wfI-def as xa* **by** *auto*
hence *Some sa* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *sa b1* **using** *False* **by** *auto*
then show *?thesis* **by** *auto*
qed

thus *case xa of* (*xa*, *ba*, *ca*) ⇒ ∃ *sa*. *Some sa* = (*i*(*x* ↦ *s*)) *xa* ∧ *wfRCV* Θ *sa ba* **using** *xa* **by** *auto*
qed

lemma *wfI-upd-full*:
fixes *v*::*v*
assumes *wfI* Θ *G i* **and** *G* = ((Γ′$[x::=v]_{\Gamma v}$)@Γ) **and** *wfRCV* Θ *s b* **and** *wfG* Θ *B* (Γ′@((*x*,*b*,*c*)$\#_\Gamma$Γ)) **and** Θ ; *B* ; Γ $\vdash_{wf}$ *v* : *b*
shows *wfI* Θ (Γ′@((*x*,*b*,*c*)$\#_\Gamma$Γ)) (*i*(*x* ↦ *s*))

proof(*subst wfI-def*,*rule*)
fix *xa*
assume *as*:*xa* ∈ *toSet* (Γ′@((*x*,*b*,*c*)#$_Γ$Γ))

then obtain *x1*::*x* **and** *b1*::*b* **and** *c1*::*c* **where** *xa*: *xa* = (*x1*,*b1*,*c1*) **using** *toSet.simps*
using *prod-cases3* **by** *blast*

have ∃ *sa*. *Some sa* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *sa b1*
proof(*cases x*=*x1*)
case *True*
hence *b*=*b1* **using** *as xa wfG-unique-full assms* **by** *metis*
hence *Some s* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *s b1* **using** *assms True* **by** *simp*
then show *?thesis* **by** *auto*
next
case *False*
hence (*x1*,*b1*,*c1*) ∈ *toSet* (Γ′@Γ) **using** *as xa* **by** *auto*
then obtain *c1′* **where** (*x1*,*b1*,*c1′*) ∈ *toSet* (Γ′[*x*::=*v*]$_{Γv}$@Γ) **using** *xa as wfG-member-subst assms False* **by** *metis*
then obtain *sa* **where** *Some sa* = *i x1* ∧ *wfRCV* Θ *sa b1* **using** *assms wfI-def as xa* **by** *blast*
hence *Some sa* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *sa b1* **using** *False* **by** *auto*
then show *?thesis* **by** *auto*
qed

thus *case xa of* (*xa*, *ba*, *ca*) ⇒ ∃ *sa*. *Some sa* = (*i*(*x* ↦ *s*)) *xa* ∧ *wfRCV* Θ *sa ba* **using** *xa* **by** *auto*
qed

lemma *subst-c-satis*[*simp*]:
fixes *v*::*v*
assumes *i* ⟦ *v* ⟧ ~ *s* **and** *wfC* Θ *B* ((*x*,*b*,*c′*)#$_Γ$Γ) *c* **and** *wfI* Θ Γ *i* **and** Θ ; *B* ; Γ ⊢$_{wf}$ *v* : *b*
shows *i* ⊨ (*c*[*x*::=*v*]$_{cv}$) ⟷ (*i* (*x* ↦ *s*)) ⊨ *c*
proof −
have *wfI* Θ ((*x*, *b*, *c′*) #$_Γ$ Γ) (*i*(*x* ↦ *s*)) **using** *wfI-upd assms wfC-wf eval-v-base* **by** *blast*
then obtain *s1* **where** *s1*:*eval-c* (*i*(*x* ↦ *s*)) *c s1* **using** *eval-c-exist*[*of* Θ ((*x*,*b*,*c′*)#$_Γ$Γ) (*i* (*x* ↦ *s*)) *B c*] *assms* **by** *auto*

have Θ ; *B* ; Γ ⊢$_{wf}$ *c*[*x*::=*v*]$_{cv}$ **using** *wf-subst1*(*2*)[*OF assms*(*2*) - *assms*(*4*) , *of GNil x*] *subst-gv.simps* **by** *simp*
then obtain *s2* **where** *s2*:*eval-c i c*[*x*::=*v*]$_{cv}$ *s2* **using** *eval-c-exist*[*of* Θ Γ *i B c*[*x*::=*v*]$_{cv}$] *assms* **by** *auto*

show *?thesis* **using** *s1 s2 subst-c-eval-v*[*OF assms*(*1*) *s2 s1*] *is-satis.cases*
using *eval-c-uniqueness is-satis.simps* **by** *auto*
qed

Key theorem telling us we can replace a substitution with an update to the valuation

lemma *subst-c-satis-full*:
fixes *v*::*v* **and** Γ′::Γ
assumes *i* ⟦ *v* ⟧ ~ *s* **and** *wfC* Θ *B* (Γ′@((*x*,*b*,*c′*)#$_Γ$Γ)) *c* **and** *wfI* Θ ((Γ′[*x*::=*v*]$_{Γv}$)@Γ) *i* **and** Θ ; *B* ; Γ ⊢$_{wf}$ *v* : *b*
shows *i* ⊨ (*c*[*x*::=*v*]$_{cv}$) ⟷ (*i* (*x* ↦ *s*)) ⊨ *c*
proof −
have *wfI* Θ (Γ′@((*x*, *b*, *c′*)) #$_Γ$ Γ) (*i*(*x* ↦ *s*)) **using** *wfI-upd-full assms wfC-wf eval-v-base wfI-suffix*

wfI-def wfV-wf **by** *fast*
then obtain *s1* **where** *s1*:*eval-c* (*i*(*x* ↦ *s*)) *c s1* **using** *eval-c-exist*[*of* Θ (Γ′@(*x*,*b*,*c*′)$\#_\Gamma$Γ) (*i* (*x* ↦ *s*)) *B c*] *assms* **by** *auto*

have Θ ; *B* ; ((Γ′[*x*::=*v*]$_{\Gamma v}$)@Γ) $\vdash_{wf}$ *c*[*x*::=*v*]$_{cv}$ **using** *wbc-subst assms* **by** *auto*

then obtain *s2* **where** *s2*:*eval-c i c*[*x*::=*v*]$_{cv}$ *s2* **using** *eval-c-exist*[*of* Θ ((Γ′[*x*::=*v*]$_{\Gamma v}$)@Γ) *i B c*[*x*::=*v*]$_{cv}$] *assms* **by** *auto*

show *?thesis* **using** *s1 s2 subst-c-eval-v*[*OF assms*(*1*) *s2 s1*] *is-satis.cases*
using *eval-c-uniqueness is-satis.simps* **by** *auto*
qed

11.5 Validity

lemma *validI*[*intro*]:
fixes *c*::*c*
assumes *wfC P B G c* **and** ∀ *i*. *P* ; *G* ⊢ *i* ∧ *i* ⊨ *G* ⟶ *i* ⊨ *c*
shows *P* ; *B* ; *G* ⊨ *c*
using *assms valid.simps* **by** *presburger*

lemma *valid-g-wf*:
fixes *c*::*c* **and** *G*::Γ
assumes *P* ; *B* ; *G* ⊨ *c*
shows *P* ; *B* $\vdash_{wf}$ *G*
using *assms wfC-wf valid.simps* **by** *blast*

lemma *valid-reflI* [*intro*]:
fixes *b*::*b*
assumes *P* ; *B* ; ((*x*,*b*,*c1*)$\#_\Gamma$*G*) $\vdash_{wf}$ *c1* **and** *c1* = *c2*
shows *P* ; *B* ; ((*x*,*b*,*c1*)$\#_\Gamma$*G*) ⊨ *c2*
using *satis-reflI assms* **by** *simp*

11.5.1 Weakening and Strengthening

Adding to the domain of a valuation doesn't change the result

lemma *eval-v-weakening*:
fixes *c*::*v* **and** *B*::*bv fset*
assumes *i* = *i*′|‘ *d* **and** *supp c* ⊆ *atom* ‘ *d* ∪ *supp B* **and** *i* ⟦ *c* ⟧ ~ *s*
shows *i*′ ⟦ *c* ⟧ ~ *s*
using *assms* **proof**(*nominal-induct c arbitrary*:*s rule*: *v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *eval-v-elims eval-v-litI* **by** *metis*
next
case (*V-var x*)
have *atom x* ∈ *atom* ‘ *d* **using** *x-not-in-b-set*[*of x B*] *assms v.supp*(*2*) *supp-at-base*
proof −
show *?thesis*
by (*metis UnE V-var.prems*(*2*) ‹*atom x* ∉ *supp B*› *singletonI subset-iff supp-at-base v.supp*(*2*))
qed
moreover have *Some s* = *i x* **using** *assms eval-v-elims*(*2*)

```
    using V-var.prems(3) by blast
  hence Some s= i' x using assms insert-subset restrict-in
  proof -
    show ?thesis
      by (metis (no-types) ‹i = i' |` d› ‹Some s = i x› atom-eq-iff calculation imageE restrict-in)
  qed
  thus ?case using eval-v.eval-v-varI by simp

next
  case (V-pair v1 v2)
  then show ?case using eval-v-elims(3) eval-v-pairI v.supp
    by (metis assms le-sup-iff)
next
  case (V-cons dc v1)
  then show ?case using eval-v-elims(4) eval-v-consI v.supp
    by (metis assms le-sup-iff)
next
  case (V-consp tyid dc b1 v1)

  then obtain sv1 where *:i ⟦ v1 ⟧ ~ sv1 ∧ s = SConsp tyid dc b1 sv1 using eval-v-elims by metis
  hence i' ⟦ v1 ⟧ ~ sv1 using V-consp by auto
  then show ?case using * eval-v-conspI v.supp eval-v.simps  assms le-sup-iff by metis
qed

lemma eval-v-restrict:
  fixes c::v and B::bv fset
  assumes i = i' |` d and supp c ⊆ atom ` d ∪ supp B   and i' ⟦ c ⟧ ~ s
  shows i ⟦ c ⟧ ~ s
  using assms proof(nominal-induct c arbitrary:s rule: v.strong-induct)
  case (V-lit x)
  then show ?case using eval-v-elims eval-v-litI by metis
next
  case (V-var x)
  have atom x ∈ atom ` d using x-not-in-b-set[of x B] assms v.supp(2) supp-at-base
  proof -
    show ?thesis
      by (metis UnE V-var.prems(2) ‹atom x ∉ supp B› singletonI subset-iff supp-at-base v.supp(2))
  qed
  moreover have Some s = i' x using assms eval-v-elims(2)
    using V-var.prems(3) by blast
  hence Some s= i x using assms insert-subset restrict-in
  proof -
    show ?thesis
      by (metis (no-types) ‹i = i' |` d› ‹Some s = i' x› atom-eq-iff calculation imageE restrict-in)
  qed
  thus ?case using eval-v.eval-v-varI by simp
next
  case (V-pair v1 v2)
  then show ?case using eval-v-elims(3) eval-v-pairI v.supp
    by (metis assms le-sup-iff)
next
  case (V-cons dc v1)
```

then show *?case* using *eval-v-elims*(*4*) *eval-v-consI v.supp*
by (*metis assms le-sup-iff*)
next
case (*V-consp tyid dc b1 v1*)
then obtain *sv1* **where** ∗:*i′* ⟦ *v1* ⟧ ~ *sv1* ∧ *s* = *SConsp tyid dc b1 sv1* **using** *eval-v-elims* **by** *metis*
hence *i* ⟦ *v1* ⟧ ~ *sv1* **using** *V-consp* **by** *auto*
then show *?case* **using** ∗ *eval-v-conspI v.supp eval-v.simps assms le-sup-iff* **by** *metis*
qed

lemma *eval-e-weakening*:
fixes *e*::*ce* **and** *B*::*bv fset*
assumes *i* ⟦ *e* ⟧ ~ *s* **and** *i* = *i′* |‘ *d* **and** *supp e* ⊆ *atom* ‘ *d* ∪ *supp B*
shows *i′* ⟦ *e* ⟧ ~ *s*
using *assms* **proof**(*induct rule*: *eval-e.induct*)
case (*eval-e-valI i v sv*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
next
case (*eval-e-plusI i v1 n1 v2 n2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
next
case (*eval-e-leqI i v1 n1 v2 n2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
next
case (*eval-e-eqI i v1 n1 v2 n2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
next
case (*eval-e-fstI i v v1 v2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *metis*
next
case (*eval-e-sndI i v v1 v2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *metis*
next
case (*eval-e-concatI i v1 bv2 v2 bv1*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
next
case (*eval-e-lenI i v bv*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
qed

lemma *eval-e-restrict* :
fixes *e*::*ce* **and** *B*::*bv fset*
assumes *i′* ⟦ *e* ⟧ ~ *s* **and** *i* = *i′* |‘ *d* **and** *supp e* ⊆ *atom* ‘ *d* ∪ *supp B*
shows *i* ⟦ *e* ⟧ ~ *s*
using *assms* **proof**(*induct rule*: *eval-e.induct*)

```
  case (eval-e-valI i v sv)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by auto
next
  case (eval-e-plusI i v1 n1 v2 n2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by auto
next
  case (eval-e-leqI i v1 n1 v2 n2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by auto
next
  case (eval-e-eqI i v1 n1 v2 n2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by auto
next
  case (eval-e-fstI i v v1 v2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by metis
next
  case (eval-e-sndI i v v1 v2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by metis
next
  case (eval-e-concatI i v1 bv2 v2 bv1)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by auto
next
  case (eval-e-lenI i v bv)
  then show ?case using ce.supp eval-e.intros
    using eval-v-restrict by auto
qed

lemma eval-c-i-weakening:
  fixes c::c and B::bv fset
  assumes i ⟦ c ⟧ ~ s and i = i' |` d and supp c ⊆ atom ` d ∪ supp B
  shows i' ⟦ c ⟧ ~ s
  using assms proof(induct rule:eval-c.induct)
  case (eval-c-eqI i e1 sv1 e2 sv2)
  then show ?case using eval-c.intros eval-e-weakening by auto
qed(auto simp add: eval-c.intros)+

lemma eval-c-i-restrict:
  fixes c::c and B::bv fset
  assumes i' ⟦ c ⟧ ~ s and i = i' |` d and supp c ⊆ atom ` d ∪ supp B
  shows i ⟦ c ⟧ ~ s
  using assms proof(induct rule:eval-c.induct)
  case (eval-c-eqI i e1 sv1 e2 sv2)
  then show ?case using eval-c.intros eval-e-restrict by auto
qed(auto simp add: eval-c.intros)+

lemma is-satis-i-weakening:
```

fixes *c*::*c* **and** *B*::*bv fset*
assumes $i = i'$ |' *d* **and** *supp c* ⊆ *atom* ' *d* ∪ *supp B* **and** $i \models c$
shows $i' \models c$
using *is-satis.simps eval-c-i-weakening*[*OF* - *assms*(*1*) *assms*(*2*)]
using *assms*(*3*) **by** *auto*

lemma *is-satis-i-restrict*:
fixes *c*::*c* **and** *B*::*bv fset*
assumes $i = i'$ |' *d* **and** *supp c* ⊆ *atom* ' *d* ∪ *supp B* **and** $i' \models c$
shows $i \models c$
using *is-satis.simps eval-c-i-restrict*[*OF* - *assms*(*1*) *assms*(*2*)]
using *assms*(*3*) **by** *auto*

lemma *is-satis-g-restrict1*:
fixes Γ'::Γ **and** Γ::Γ
assumes *toSet* Γ ⊆ *toSet* Γ' **and** $i \models \Gamma'$
shows $i \models \Gamma$
using *assms* **proof**(*induct* Γ *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc G*)
obtain *x* **and** *b* **and** *c*::*c* **where** *xbc*: *xbc*=(*x*,*b*,*c*)
using *prod-cases3* **by** *blast*
hence $i \models G$ **using** *GCons* **by** *auto*
moreover have $i \models c$ **using** *GCons*
is-satis-iff toSet.simps subset-iff
using *xbc* **by** *blast*
ultimately show *?case* **using** *is-satis-g.simps GCons*
by (*simp add*: *xbc*)
qed

lemma *is-satis-g-restrict2*:
fixes Γ'::Γ **and** Γ::Γ
assumes $i \models \Gamma$ **and** $i' = i$ |' *d* **and** *atom-dom* Γ ⊆ *atom* ' *d* **and** $\Theta \ ; B \vdash_{wf} \Gamma$
shows $i' \models \Gamma$
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x b c G*)

hence $i' \models G$ **proof** −
have $i \models G$ **using** *GCons* **by** *simp*
moreover have *atom-dom G* ⊆ *atom* ' *d* **using** *GCons* **by** *simp*
ultimately show *?thesis* **using** *GCons wfG-cons2* **by** *blast*
qed

moreover have $i' \models c$ **proof** −
have $i \models c$ **using** *GCons* **by** *auto*
moreover have $\Theta \ ; B \ ; (x,\ b,\ TRUE) \#_{\Gamma} G \ \vdash_{wf} c$ **using** *wfG-wfC GCons* **by** *simp*
moreover hence *supp c* ⊆ *atom* ' *d* ∪ *supp B* **using** *wfC-supp GCons atom-dom-eq* **by** *blast*

ultimately show *?thesis* **using** *is-satis-i-restrict*[*of* i' *i d c*] *GCons* **by** *simp*
qed

ultimately show *?case* **by** *auto*
qed

lemma *is-satis-g-restrict*:
fixes $\Gamma'::\Gamma$ **and** $\Gamma::\Gamma$
assumes *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** $i' \models \Gamma'$ **and** $i = i' |` (fst ` toSet\ \Gamma)$ **and** Θ ; $B \vdash_{wf} \Gamma$
shows $i \models \Gamma$
using *assms is-satis-g-restrict1*[*OF assms*(*1*) *assms*(*2*)] *is-satis-g-restrict2*[*OF* - *assms*(*3*)] **by** *simp*

11.5.2 Updating valuation

lemma *is-satis-c-i-upd*:
fixes *c*::*c*
assumes *atom x* ♯ *c* **and** $i \models c$
shows $((i\ (\ x \mapsto s))) \models c$
using *assms eval-c-i-upd is-satis.simps* **by** *simp*

lemma *is-satis-g-i-upd*:
fixes *G*::Γ
assumes *atom x* ♯ *G* **and** $i \models G$
shows $((i\ (\ x \mapsto s))) \models G$
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* x' b' c' G')

hence ∗:*atom x* ♯ G' ∧ *atom x* ♯ c'
using *fresh-def fresh-GCons GCons* **by** *force*
moreover hence *is-satis* $((i\ (\ x \mapsto s)))\ c'$
using *is-satis-c-i-upd GCons is-satis-g.simps* **by** *auto*
moreover have *is-satis-g* $(i(x \mapsto s))\ G'$ **using** *GCons* ∗ **by** *fastforce*
ultimately show *?case*
using *GCons is-satis-g.simps*(*2*) **by** *metis*
qed

lemma *valid-weakening*:
assumes Θ ; B ; $\Gamma \models c$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** *wfG* Θ B Γ'
shows Θ ; B ; $\Gamma' \models c$
proof −
have *wfC* Θ B Γ *c* **using** *assms valid.simps* **by** *auto*
hence *sp*: *supp c* ⊆ *atom* `(*fst* `*toSet* Γ) ∪ *supp B* **using** *wfX-wfY wfG-elims*
using *atom-dom.simps dom.simps wf-supp*(*2*) **by** *metis*
have *wfg*: *wfG* Θ B Γ **using** *assms valid.simps wfC-wf* **by** *auto*

moreover have *a1*: ($\forall i.$ *wfI* $\Theta\ \Gamma'\ i \wedge i \models \Gamma' \longrightarrow i \models c$) **proof**(*rule allI*, *rule impI*)
fix *i*
assume *as*: *wfI* $\Theta\ \Gamma'\ i \wedge i \models \Gamma'$
hence *as1*: *fst* ` *toSet* Γ ⊆ *dom i* **using** *assms wfI-domi* **by** *blast*
obtain i' **where** *idash*: $i' =$ *restrict-map i* (*fst* `*toSet* Γ) **by** *blast*

hence *as2*: *dom* i' = (*fst* '*toSet* Γ) **using** *dom-restrict as1* **by** *auto*

have *id2*: Θ ; Γ ⊢ i' ∧ i' ⊨ Γ **proof** −
have *wfI* Θ Γ i' **using** *as2 wfI-restrict-weakening*[*of* Θ Γ′ *i* i' Γ] *as assms*
using *idash* **by** *blast*
moreover have i' ⊨ Γ **using** *is-satis-g-restrict*[*OF assms*(*2*)] *wfg as idash* **by** *auto*
ultimately show *?thesis* **using** *idash* **by** *auto*
qed
hence i' ⊨ *c* **using** *assms valid.simps* **by** *auto*
thus *i* ⊨ *c* **using** *assms valid.simps is-satis-i-weakening idash sp* **by** *blast*
qed
moreover have *wfC* Θ *B* Γ′ *c* **using** *wf-weakening assms valid.simps*
by (*meson wfg*)
ultimately show *?thesis* **using** *assms valid.simps* **by** *auto*
qed

lemma *is-satis-g-suffix*:
fixes *G*::Γ
assumes *i* ⊨ (*G′*@*G*)
shows *i* ⊨ *G*
using *assms* **proof**(*induct G′ rule*:Γ.*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc x2*)
obtain *x* **and** *b* **and** *c*::*c* **where** *xbc*: *xbc*=(*x*,*b*,*c*)
using *prod-cases3* **by** *blast*
hence *i* ⊨ (*xbc* $\#_\Gamma$ (*x2* @ *G*)) **using** *append-g.simps GCons* **by** *fastforce*
then show *?case* **using** *is-satis-g.simps GCons xbc* **by** *blast*
qed

lemma *wfG-inside-valid2*:
fixes *x*::*x* **and** Γ::Γ **and** *c0*::*c* **and** *c0′*::*c*
assumes *wfG* Θ *B* (Γ′@((*x*,*b0*,*c0′*)$\#_\Gamma$Γ)) **and**
Θ ; *B* ; Γ′@(*x*,*b0*,*c0*)$\#_\Gamma$Γ ⊨ *c0′*
shows *wfG* Θ *B* (Γ′@((*x*,*b0*,*c0*)$\#_\Gamma$Γ))
proof −
have *wfG* Θ *B* (Γ′@(*x*,*b0*,*c0*)$\#_\Gamma$Γ) **using** *valid.simps wfC-wf assms* **by** *auto*
thus *?thesis* **using** *wfG-replace-inside-full assms* **by** *auto*
qed

lemma *valid-trans*:
assumes Θ ; $\mathcal{B}$; Γ ⊨ *c0*[*z*::=*v*]$_v$ **and** Θ ; $\mathcal{B}$; (*z*,*b*,*c0*)$\#_\Gamma$Γ ⊨ *c1* **and** *atom z* ♯ Γ **and** *wfV* Θ $\mathcal{B}$ Γ *v b*
shows Θ ; $\mathcal{B}$; Γ ⊨ *c1*[*z*::=*v*]$_v$
proof −
have ∗:*wfC* Θ $\mathcal{B}$ ((*z*,*b*,*c0*)$\#_\Gamma$Γ) *c1* **using** *valid.simps assms* **by** *auto*
hence *wfC* Θ $\mathcal{B}$ Γ (*c1*[*z*::=*v*]$_v$) **using** *wf-subst1*(*2*)[*OF* ∗ , *of GNil*] *assms subst-gv.simps subst-v-c-def*
by *force*

moreover have ∀ *i*. *wfI* Θ Γ *i* ∧ *is-satis-g i* Γ ⟶ *is-satis i* (*c1*[*z*::=*v*]$_v$)
proof(*rule*,*rule*)

fix *i*
assume *as*: $wfI\ \Theta\ \Gamma\ \ i \wedge is\text{-}satis\text{-}g\ i\ \Gamma$
then obtain *sv* **where** *sv*: $eval\text{-}v\ i\ v\ sv \wedge wfRCV\ \Theta\ sv\ b$ **using** *eval-v-exist assms* **by** *metis*
hence $is\text{-}satis\ i\ (c0[z{::=}v]_v)$ **using** *assms valid.simps as* **by** *metis*
hence $is\text{-}satis\ (i(z \mapsto sv))\ \ c0$ **using** *subst-c-satis sv as assms valid.simps wfC-wf wfG-elim2 subst-v-c-def* **by** *metis*
moreover have $is\text{-}satis\text{-}g\ \ (i(z \mapsto sv))\ \Gamma$
using *is-satis-g-i-upd assms* **by** (*simp add*: *as*)
ultimately have $is\text{-}satis\text{-}g\ \ (i(z \mapsto sv))\ ((z,b,c0) \#_\Gamma \Gamma)$
using *is-satis-g.simps* **by** *simp*
moreover have $wfI\ \Theta\ ((z,b,c0) \#_\Gamma \Gamma)\ (i(z \mapsto sv))$ **using** *as wfI-upd sv assms valid.simps wfC-wf* **by** *metis*
ultimately have $is\text{-}satis\ (i(z \mapsto sv))\ c1$ **using** *assms valid.simps* **by** *auto*
thus $is\text{-}satis\ i\ (c1[z{::=}v]_v)$ **using** *subst-c-satis sv as assms valid.simps wfC-wf wfG-elim2 subst-v-c-def* **by** *metis*
qed

ultimately show *?thesis* **using** *valid.simps* **by** *auto*
qed

lemma *valid-trans-full*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ ((x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c2[z2{::=}V\text{-}var\ x]_v$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ ((x,\ b,\ c2[z2{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[z3{::=}V\text{-}var\ x]_v$
shows $\Theta\ ;\ \mathcal{B}\ ;\ ((x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[z3{::=}V\text{-}var\ x]_v$
unfolding *valid.simps* **proof**
show $\Theta\ ;\ \mathcal{B}\ ;\ (x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma\ \ \vdash_{wf}\ c3[z3{::=}V\text{-}var\ x]_v$ **using** *wf-trans valid.simps assms* **by** *metis*

show $\forall\, i.\ \ (\ wfI\ \ \Theta\ ((x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma)\ i \wedge\ \ (is\text{-}satis\text{-}g\ i\ ((x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma))\ \longrightarrow\ (is\text{-}satis\ i\ (c3[z3{::=}V\text{-}var\ x]_v))\ )$
proof(*rule*,*rule*)
fix *i*
assume *as*: $\Theta\ ;\ (x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma \vdash i \wedge\ \ i \models (x,\ b,\ c1[z1{::=}V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma$
have $i \models c2[z2{::=}V\text{-}var\ x]_v$ **using** *is-satis-g.simps as assms* **by** *simp*
moreover have $i \models \Gamma$ **using** *is-satis-g.simps as* **by** *simp*
ultimately show $i \models c3[z3{::=}V\text{-}var\ x]_v$ **using** *assms is-satis-g.simps valid.simps*
by (*metis append-g.simps*(*1*) *as wfI-replace-inside*)
qed
qed

lemma *eval-v-weakening-x*:
fixes *c*::*v*
assumes $i'\ [\![\ c\ ]\!]\ {\sim}\ s$ **and** $atom\ x \sharp c$ **and** $i = i'\ (x \mapsto s')$
shows $i\ [\![\ c\ ]\!]\ {\sim}\ s$
using *assms* **proof**(*induct rule*: *eval-v.induct*)
case (*eval-v-litI i l*)
then show *?case* **using** *eval-v.intros* **by** *auto*
next
case (*eval-v-varI sv i1 x1*)
hence $x \neq x1$ **using** *v.fresh fresh-at-base* **by** *auto*
hence $i\ x1 = Some\ sv$ **using** *eval-v-varI* **by** *simp*
then show *?case* **using** *eval-v.intros* **by** *auto*

next
  case (*eval-v-pairI i v1 s1 v2 s2*)
  then show *?case* **using** *eval-v.intros* **by** *auto*
next
  case (*eval-v-consI i v s tyid dc*)
  then show *?case* **using** *eval-v.intros* **by** *auto*
next
  case (*eval-v-conspI i v s tyid dc b*)
  then show *?case* **using** *eval-v.intros* **by** *auto*
qed

lemma *eval-e-weakening-x*:
  fixes *c*::*ce*
  assumes $i' ⟦ c ⟧ \sim s$ **and** *atom x* ♯ *c* **and** $i = i' (x \mapsto s')$
  shows $i ⟦ c ⟧ \sim s$
  using *assms* **proof**(*induct rule*: *eval-e.induct*)
  case (*eval-e-valI i v sv*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-plusI i v1 n1 v2 n2*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-leqI i v1 n1 v2 n2*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-eqI i v1 n1 v2 n2*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-fstI i v v1 v2*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-sndI i v v1 v2*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-concatI i v1 bv1 v2 bv2*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
  case (*eval-e-lenI i v bv*)
  then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
qed

lemma *eval-c-weakening-x*:
  fixes *c*::*c*
  assumes $i' ⟦ c ⟧ \sim s$ **and** *atom x* ♯ *c* **and** $i = i' (x \mapsto s')$
  shows $i ⟦ c ⟧ \sim s$
  using *assms* **proof**(*induct rule*: *eval-c.induct*)
  case (*eval-c-trueI i*)
  then show *?case* **using** *eval-c.intros* **by** *auto*
next
  case (*eval-c-falseI i*)
  then show *?case* **using** *eval-c.intros* **by** *auto*
next

case (*eval-c-conjI i c1 b1 c2 b2*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-disjI i c1 b1 c2 b2*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-impI i c1 b1 c2 b2*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-notI i c b*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-eqI i e1 sv1 e2 sv2*)
then show *?case* **using** *eval-e-weakening-x c.fresh eval-c.intros* **by** *metis*
qed

lemma *is-satis-weakening-x*:
fixes $c::c$
assumes $i' \models c$ **and** *atom* $x \sharp c$ **and** $i = i' \ (x \mapsto s)$
shows $i \models c$
using *eval-c-weakening-x assms is-satis.simps* **by** *simp*

lemma *is-satis-g-weakening-x*:
fixes $G::\Gamma$
assumes $i' \models G$ **and** *atom* $x \sharp G$ **and** $i = i' \ (x \mapsto s)$
shows $i \models G$
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* $x'\ b'\ c'\ \Gamma'$)
hence *atom* $x \sharp c'$ **using** *fresh-GCons fresh-prodN* **by** *simp*
moreover hence $i \models c'$ **using** *is-satis-weakening-x is-satis-g.simps*(*2*) *GCons* **by** *metis*
then show *?case* **using** *is-satis-g.simps*(*2*)[*of i* $x'\ b'\ c'\ \Gamma'$] *GCons fresh-GCons* **by** *simp*
qed

11.6 Base Type Substitution

The idea of boxing is to take an smt val and its base type and at nodes in the smt val that correspond to type variables we wrap them in an SUt smt val node. Another way of looking at it is that s' where the node for the base type variable is an 'any node'. It is needed to prove subst_b_valid - the base-type variable substitution lemma for validity.

The first *rcl-val* is the expanded form (has type with base-variables replaced with base-type terms) ; the second is its corresponding form

We only have one variable so we need to ensure that in all of the *bs-boxed-BVarI* cases, the s has the same base type.

For example is an SMT value is (SPair (SInt 1) (SBool true)) and it has sort (BPair (BVar x) BBool)[x::=BInt] then the boxed version is SPair (SUt (SInt 1)) (SBool true) and is has sort (BPair (BVar x) BBool). We need to do this so that we can obtain from a valuation i, that gives values like the first smt value, to a valuation i' that gives values like the second.

inductive *boxed-b* :: $\Theta \Rightarrow$ *rcl-val* $\Rightarrow$ *b* $\Rightarrow$ *bv* $\Rightarrow$ *b* $\Rightarrow$ *rcl-val* $\Rightarrow$ *bool* (‹ - ⊢ - ~ - [- ::= -] \ - › [*50*,*50*] *50*) **where**
boxed-b-BVar1I: ⟦ $bv = bv'$; *wfRCV P s b* ⟧ ⟹ *boxed-b P s* (*B-var bv'*) *bv b* (*SUt s*)
| *boxed-b-BVar2I*: ⟦ $bv \neq bv'$; *wfRCV P s* (*B-var bv'*) ⟧ ⟹ *boxed-b P s* (*B-var bv'*) *bv b s*
| *boxed-b-BIntI*:*wfRCV P s B-int* ⟹ *boxed-b P s B-int - - s*
| *boxed-b-BBoolI*:*wfRCV P s B-bool* ⟹ *boxed-b P s B-bool - - s*
| *boxed-b-BUnitI*:*wfRCV P s B-unit* ⟹ *boxed-b P s B-unit - - s*
| *boxed-b-BPairI*:⟦ *boxed-b P s1 b1 bv b s1'* ; *boxed-b P s2 b2 bv b s2'* ⟧ ⟹ *boxed-b P* (*SPair s1 s2*) (*B-pair b1 b2*) *bv b* (*SPair s1' s2'*)

| *boxed-b-BConsI*:⟦
AF-typedef tyid dclist ∈ *set P*;
(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* ;
boxed-b P s1 b bv b' s1'
⟧ ⟹
boxed-b P (*SCons tyid dc s1*) (*B-id tyid*) *bv b'* (*SCons tyid dc s1'*)

| *boxed-b-BConspI*:⟦ *AF-typedef-poly tyid bva dclist* ∈ *set P*;
atom bva ♯ (*b1*,*bv*,*b'*,*s1*,*s1'*);
(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* ;
boxed-b P s1 ($b[bva{::=}b1]_{bb}$) *bv b' s1'*
⟧ ⟹
boxed-b P (*SConsp tyid dc* $b1[bv{::=}b']_{bb}$ *s1*) (*B-app tyid b1*) *bv b'* (*SConsp tyid dc b1 s1'*)

| *boxed-b-Bbitvec*: *wfRCV P s B-bitvec* ⟹ *boxed-b P s B-bitvec bv b s*

equivariance *boxed-b*
nominal-inductive *boxed-b* **.**

inductive-cases *boxed-b-elims*:
boxed-b P s (*B-var bv*) *bv' b s'*
boxed-b P s B-int bv b s'
boxed-b P s B-bool bv b s'
boxed-b P s B-unit bv b s'
boxed-b P s (*B-pair b1 b2*) *bv b s'*
boxed-b P s (*B-id dc*) *bv b s'*
boxed-b P s B-bitvec bv b s'
boxed-b P s (*B-app dc b'*) *bv b s'*

lemma *boxed-b-wfRCV*:
assumes *boxed-b P s b bv b' s'* **and** $\vdash_{wf}$ *P*
shows *wfRCV P s* $b[bv{::=}b']_{bb}$ ∧ *wfRCV P s' b*
using *assms* **proof**(*induct rule*: *boxed-b.inducts*)
case (*boxed-b-BVar1I bv bv' P s b*)
then show *?case* **using** *wfRCV.intros* **by** *auto*
next
case (*boxed-b-BVar2I bv bv' P s*)
then show *?case* **using** *wfRCV.intros* **by** *auto*
next
case (*boxed-b-BPairI P s1 b1 bv b s1' s2 b2 s2'*)
then show *?case* **using** *wfRCV.intros rcl-val.supp* **by** *simp*

next
  case (*boxed-b-BConsI tyid dclist P dc x b c s1 bv b′ s1′*)
  hence *supp b* = {} **using** *wfTh-supp-b* **by** *metis*
  hence *b* [*bv* ::= *b′*]$_{bb}$ = *b* **using** *fresh-def subst-b-b-def forget-subst*[*of bv b b′*] **by** *auto*
  hence *P* ⊢ *SCons tyid dc s1* : (*B-id tyid*) **using** *wfRCV.intros rcl-val.supp subst-bb.simps boxed-b-BConsI* **by** *metis*
  moreover have *P* ⊢ *SCons tyid dc s1′* : *B-id tyid* **using** *boxed-b-BConsI*
    using *wfRCV.intros rcl-val.supp subst-bb.simps boxed-b-BConsI* **by** *metis*
  ultimately show *?case* **using** *subst-bb.simps* **by** *metis*
next
  case (*boxed-b-BConspI tyid bva dclist P b1 bv b′ s1 s1′ dc x b c*)

  obtain *bva2* **and** *dclist2* **where** ∗:*AF-typedef-poly tyid bva dclist* = *AF-typedef-poly tyid bva2 dclist2* ∧
        atom bva2 ♯ (*bv*,(*P*, *SConsp tyid dc b1*[*bv*::=*b′*]$_{bb}$ *s1*, *B-app tyid b1*[*bv*::=*b′*]$_{bb}$))
    using *obtain-fresh-bv* **by** *metis*

  then obtain *x2* **and** *b2* **and** *c2* **where** ∗∗:‹(*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2*›
    using *boxed-b-BConspI td-lookup-eq-iff type-def.eq-iff* **by** *metis*

  have *P* ⊢ *SConsp tyid dc b1*[*bv*::=*b′*]$_{bb}$ *s1* : (*B-app tyid b1*[*bv*::=*b′*]$_{bb}$) **proof**
    show *1*: ‹*AF-typedef-poly tyid bva2 dclist2* ∈ *set P*› **using** *boxed-b-BConspI* ∗ **by** *auto*
    show *2*: ‹(*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2*› **using** *boxed-b-BConspI* **using** ∗∗ **by** *simp*

    hence *atom bv* ♯ *b2* **proof** −
      have *supp b2* ⊆ { *atom bva2* } **using** *wfTh-poly-supp-b 1 2 boxed-b-BConspI* **by** *auto*
      moreover have *bv* ≠ *bva2* **using** ∗ *fresh-Pair fresh-at-base* **by** *metis*
      ultimately show *?thesis* **using** *fresh-def* **by** *force*
    qed
    moreover have *b*[*bva*::=*b1*]$_{bb}$ = *b2*[*bva2*::=*b1*]$_{bb}$ **using** *wfTh-typedef-poly-b-eq-iff* ∗ *2 boxed-b-BConspI* **by** *metis*
    ultimately show ‹ *P* ⊢ *s1* : *b2*[*bva2*::=*b1*[*bv*::=*b′*]$_{bb}$]$_{bb}$› **using** *boxed-b-BConspI subst-b-b-def subst-bb-commute* **by** *auto*
    show *atom bva2* ♯ (*P*, *SConsp tyid dc b1*[*bv*::=*b′*]$_{bb}$ *s1*, *B-app tyid b1*[*bv*::=*b′*]$_{bb}$) **using** ∗ *fresh-Pair* **by** *metis*
  qed

  moreover have *P* ⊢ *SConsp tyid dc b1 s1′* : *B-app tyid b1* **proof**
    show ‹*AF-typedef-poly tyid bva dclist* ∈ *set P*› **using** *boxed-b-BConspI* **by** *auto*
    show ‹(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist*› **using** *boxed-b-BConspI* **by** *auto*
    show ‹ *P* ⊢ *s1′* : *b*[*bva*::=*b1*]$_{bb}$› **using** *boxed-b-BConspI* **by** *auto*
    have *atom bva* ♯ *P* **using** *boxed-b-BConspI wfTh-fresh* **by** *metis*
    thus *atom bva* ♯ (*P*, *SConsp tyid dc b1 s1′*, *B-app tyid b1*) **using** *boxed-b-BConspI rcl-val.fresh b.fresh pure-fresh fresh-prodN* **by** *metis*
  qed

  ultimately show *?case* **using** *subst-bb.simps* **by** *simp*
qed(*auto*)+

lemma *subst-b-var*:
  assumes *B-var bv2* = *b*[*bv*::=*b′*]$_{bb}$
  shows (*b* = *B-var bv* ∧ *b′* = *B-var bv2*) ∨ (*b*=*B-var bv2* ∧ *bv* ≠ *bv2*)

using *assms* **by**(*nominal-induct b rule*: *b.strong-induct*,*auto*+)

Here the valuation i' is the conv wrap version of i. For every x in G, i' x is the conv wrap version of i x. The next lemma for a clearer explanation of what this is. i produces values of sort b[bv::=b'] and i' produces values of sort b

inductive *boxed-i* :: $\Theta \Rightarrow \Gamma \Rightarrow b \Rightarrow bv \Rightarrow valuation \Rightarrow valuation \Rightarrow bool$ (‹ - ; - ; - , - ⊢ - ≈ -› [*50*,*50*] *50*) **where**
boxed-i-GNilI: Θ ; *GNil* ; *b* , *bv* ⊢ *i* ≈ *i*
| *boxed-i-GConsI*: ⟦ *Some s* = *i x*; *boxed-b* Θ *s b bv b' s'* ; Θ ; Γ ; *b'* , *bv* ⊢ *i* ≈ *i'* ⟧ ⟹ Θ ; ((*x*,*b*,*c*)$\#_\Gamma\Gamma$) ; *b'* , *bv* ⊢ *i* ≈ (*i'*(*x* ↦ *s'*))
equivariance *boxed-i*
nominal-inductive *boxed-i* **.**

inductive-cases *boxed-i-elims*:
Θ ;*GNil* ; *b* , *bv* ⊢ *i* ≈ *i'*
Θ ; ((*x*,*b*,*c*)$\#_\Gamma\Gamma$) ; *b'* , *bv* ⊢ *i* ≈ *i'*

lemma *wfRCV-poly-elims*:
fixes *tm*::*'a*::*fs* **and** *b*::*b*
assumes *T* ⊢ *SConsp typid dc bdc s* : *b*
obtains *bva dclist x1 b1 c1* **where** *b* = *B-app typid bdc* ∧
AF-typedef-poly typid bva dclist ∈ *set T* ∧ (*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist* ∧ *T* ⊢ *s* : *b1*[*bva*::=*bdc*]$_{bb}$ ∧ *atom bva* ♯ *tm*
using *assms* **proof**(*nominal-induct SConsp typid dc bdc s b avoiding*: *tm rule*:*wfRCV*.*strong-induct*)
case (*wfRCV-BConsPI bv dclist* Θ *x b c*)
then show *?case* **by** *simp*
qed

lemma *boxed-b-ex*:
assumes *wfRCV T s b*[*bv*::=*b'*]$_{bb}$ **and** *wfTh T*
shows ∃ *s'*. *boxed-b T s b bv b' s'*
using *assms* **proof**(*nominal-induct s arbitrary*: *b rule*: *rcl-val*.*strong-induct*)
case (*SBitvec x*)
have ∗:*b*[*bv*::=*b'*]$_{bb}$ = *B-bitvec* **using** *wfRCV-elims*(*9*)[*OF SBitvec*(*1*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SBitvec x* : *B-bitvec* **using** *wfRCV*.*intros* **by** *simp*
moreover hence *b'* = *B-bitvec* **using** *True SBitvec subst-bb*.*simps* ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b*.*intros* *wfRCV*.*intros* **by** *metis*
next
case *False*
hence *b* = *B-bitvec* **using** *subst-bb-inject* ∗ **by** *metis*
then show *?thesis* **using** ∗ *SBitvec boxed-b*.*intros* **by** *metis*
qed
next
case (*SNum x*)
have ∗:*b*[*bv*::=*b'*]$_{bb}$ = *B-int* **using** *wfRCV-elims*(*10*)[*OF SNum*(*1*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SNum x* : *B-int* **using** *wfRCV*.*intros* **by** *simp*
moreover hence *b'* = *B-int* **using** *True SNum subst-bb*.*simps*(*1*) ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b-BVar1I* *wfRCV*.*intros* **by** *metis*

next
case *False*
hence *b* = *B-int* **using** *subst-bb-inject*(*1*) ∗ **by** *metis*
then show *?thesis* **using** ∗ *SNum boxed-b-BIntI* **by** *metis*
qed
next
case (*SBool x*)
have ∗:*b*[*bv*::=*b′*]$_{bb}$ = *B-bool* **using** *wfRCV-elims*(*11*)[*OF SBool*(*1*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SBool x* : *B-bool* **using** *wfRCV.intros* **by** *simp*
moreover hence *b′* = *B-bool* **using** *True SBool subst-bb.simps* ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
hence *b* = *B-bool* **using** *subst-bb-inject* ∗ **by** *metis*
then show *?thesis* **using** ∗ *SBool boxed-b.intros* **by** *metis*
qed
next
case (*SPair s1 s2*)
then obtain *b1* **and** *b2* **where** ∗:*b*[*bv*::=*b′*]$_{bb}$ = *B-pair b1 b2* ∧ *wfRCV T s1 b1* ∧ *wfRCV T s2 b2* **using** *wfRCV-elims*(*12*) **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SPair s1 s2* : *B-pair b1 b2* **using** *wfRCV.intros* ∗ **by** *simp*
moreover hence *b′* = *B-pair b1 b2* **using** *True SPair subst-bb.simps*(*1*) ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b-BVar1I* **by** *metis*
next
case *False*
then obtain *b1′* **and** *b2′* **where** *b* = *B-pair b1′ b2′* ∧ *b1*=*b1′*[*bv*::=*b′*]$_{bb}$ ∧ *b2*=*b2′*[*bv*::=*b′*]$_{bb}$ **using** *subst-bb-inject*(*5*)[*OF - False*] ∗ **by** *metis*
then show *?thesis* **using** ∗ *SPair boxed-b-BPairI* **by** *blast*
qed
next
case (*SCons tyid dc s1*)
have ∗:*b*[*bv*::=*b′*]$_{bb}$ = *B-id tyid* **using** *wfRCV-elims*(*13*)[*OF SCons*(*2*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SCons tyid dc s1* : *B-id tyid* **using** *wfRCV.intros*
using *local.*∗ *SCons.prems* **by** *auto*
moreover hence *b′* = *B-id tyid* **using** *True SCons subst-bb.simps*(*1*) ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b-BVar1I wfRCV.intros* **by** *metis*
next
case *False*
then obtain *b1′* **where** *beq*: *b* = *B-id tyid* **using** *subst-bb-inject* ∗ **by** *metis*
then obtain *b2 dclist x c* **where** ∗∗:*AF-typedef tyid dclist* ∈ *set T* ∧ (*dc*, ⦃ *x* : *b2* | *c* ⦄) ∈ *set dclist* ∧ *wfRCV T s1 b2* **using** *wfRCV-elims*(*13*) ∗ *SCons* **by** *metis*
then have *atom bv* ♯ *b2* **using** ‹*wfTh T*› *wfTh-lookup-supp-empty*[*of tyid dclist T dc* ⦃ *x* : *b2* | *c* ⦄] *τ.fresh fresh-def* **by** *auto*
then have *b2* = *b2*[*bv* ::= *b′*]$_{bb}$ **using** *forget-subst subst-b-b-def* **by** *metis*
then obtain *s1′* **where** *s1*:*T* ⊢ *s1* ~ *b2* [*bv* ::= *b′*] \ *s1′* **using** *SCons* ∗∗ **by** *metis*

have *T* ⊢ *SCons tyid dc s1* ~ (*B-id tyid*) [*bv* ::= *b′*] \ *SCons tyid dc s1′* **proof**(*rule boxed-b-BConsI*)
show *AF-typedef tyid dclist* ∈ *set T* **using** ∗∗ **by** *auto*
show (*dc*, ⦃ *x* : *b2* | *c* ⦄) ∈ *set dclist* **using** ∗∗ **by** *auto*
show *T* ⊢ *s1* ~ *b2* [*bv* ::= *b′*] \ *s1′* **using** *s1* ∗∗ **by** *auto*

qed
thus *?thesis* **using** *beq* **by** *metis*
qed
next
case (*SConsp typid dc bdc s*)

obtain *bva dclist x1 b1 c1* **where** ∗∗:$b[bv::=b']_{bb}$ = *B-app typid bdc* ∧
AF-typedef-poly typid bva dclist ∈ *set T* ∧ (*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist* ∧ *T* ⊢ *s* : $b1[bva::=bdc]_{bb}$
∧ *atom bva* ♯ *bv*
using *wfRCV-poly-elims*[*OF SConsp*(*2*)] **by** *metis*

then have ∗:*B-app typid bdc* = $b[bv::=b']_{bb}$ **using** *wfRCV-elims*(*14*)[*OF SConsp*(*2*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SConsp typid dc bdc s* : *B-app typid bdc* **using** *wfRCV.intros*
using *local*.∗ *SConsp.prems*(*1*) **by** *auto*
moreover hence *b′* = *B-app typid bdc* **using** *True SConsp subst-bb.simps* ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
then obtain *bdc′* **where** *bdc*: *b* = *B-app typid bdc′* ∧ *bdc* = $bdc'[bv::=b']_{bb}$ **using** ∗ *subst-bb-inject*(*8*)[*OF* ∗] **by** *metis*

have *atom bv* ♯ *b1* **proof** −
have *supp b1* ⊆ { *atom bva* } **using** *wfTh-poly-supp-b* ∗∗ *SConsp* **by** *metis*
moreover have *bv* ≠ *bva* **using** ∗∗ **by** *auto*
ultimately show *?thesis* **using** *fresh-def* **by** *force*
qed
have *T* ⊢ *s* : $b1[bva::=bdc]_{bb}$ **using** ∗∗ **by** *auto*
moreover have $b1[bva::=bdc']_{bb}[bv::=b']_{bb} = b1[bva::=bdc]_{bb}$ **using** *bdc subst-bb-commute* ‹*atom bv* ♯ *b1*› **by** *auto*
ultimately obtain *s′* **where** *s′*:*T* ⊢ *s* ~ $b1[bva::=bdc']_{bb}$ [*bv* ::= *b′*] \ *s′*
using *SConsp*(*1*)[*of* $b1[bva::=bdc']_{bb}$] *bdc SConsp* **by** *metis*
have *T* ⊢ *SConsp typid dc* $bdc'[bv::=b']_{bb}$ *s* ~ (*B-app typid bdc′*) [*bv* ::= *b′*] \ *SConsp typid dc bdc′ s′*
proof −

obtain *bva3* **and** *dclist3* **where** *3*:*AF-typedef-poly typid bva3 dclist3* = *AF-typedef-poly typid bva dclist* ∧
atom bva3 ♯ (*bdc′*, *bv*, *b′*, *s*, *s′*) **using** *obtain-fresh-bv* **by** *metis*
then obtain *x3 b3 c3* **where** *4*:(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3*
using *boxed-b-BConspI td-lookup-eq-iff type-def.eq-iff*
by (*metis* ∗∗)

show *?thesis* **proof**
show ‹*AF-typedef-poly typid bva3 dclist3* ∈ *set T*› **using** *3* ∗∗ **by** *metis*
show ‹*atom bva3* ♯ (*bdc′*, *bv*, *b′*, *s*, *s′*)› **using** *3* **by** *metis*

show *4*:‹(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3*› **using** *4* **by** *auto*
have $b3[bva3::=bdc']_{bb} = b1[bva::=bdc']_{bb}$ **proof**(*rule wfTh-typedef-poly-b-eq-iff*)
show ‹*AF-typedef-poly typid bva3 dclist3* ∈ *set T*› **using** *3* ∗∗ **by** *metis*
show ‹(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3*› **using** *4* **by** *auto*
show ‹*AF-typedef-poly typid bva dclist* ∈ *set T*› **using** ∗∗ **by** *auto*
show ‹(*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist*› **using** ∗∗ **by** *auto*
qed(*simp add*: ∗∗ *SConsp*)
thus ‹ $T \vdash s \sim b3[bva3::=bdc']_{bb}$ [*bv* ::= *b'*] \ *s'* › **using** *s'* **by** *auto*
qed
qed
then show *?thesis* **using** *bdc* **by** *auto*

qed
next
case *SUnit*
have ∗:$b[bv::=b']_{bb}$ = *B-unit* **using** *wfRCV-elims SUnit* **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have *T* ⊢ *SUnit* : *B-unit* **using** *wfRCV.intros* **by** *simp*
moreover hence *b'* = *B-unit* **using** *True SUnit subst-bb.simps* ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
hence *b* = *B-unit* **using** *subst-bb-inject* ∗ **by** *metis*
then show *?thesis* **using** ∗ *SUnit boxed-b.intros* **by** *metis*
qed
next
case (*SUt x*)
then obtain *bv'* **where** ∗:$b[bv::=b']_{bb}$= *B-var bv'* **using** *wfRCV-elims* **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
then show *?thesis* **using** *boxed-b-BVar1I*
using *local*.∗ *wfRCV-BVarI* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *boxed-b-BVar1I boxed-b-BVar2I*
using *local*.∗ *wfRCV-BVarI* **by** (*metis subst-b-var*)
qed
qed

lemma *boxed-i-ex*:
assumes *wfI T* $\Gamma[bv::=b]_{\Gamma b}$ *i* **and** *wfTh T*
shows ∃ *i'*. *T* ; Γ ; *b* , *bv* ⊢ *i* ≈ *i'*
using *assms* **proof**(*induct* Γ *arbitrary*: *i rule*:Γ-*induct*)
case *GNil*
then show *?case* **using** *boxed-i-GNilI* **by** *metis*
next
case (*GCons x' b' c'* Γ')
then obtain *s* **where** *1*:*Some s* = *i x'* ∧ *wfRCV T s* $b'[bv::=b]_{bb}$ **using** *wfI-def subst-gb.simps* **by** *auto*
then obtain *s'* **where** *2*: *boxed-b T s b' bv b s'* **using** *boxed-b-ex GCons* **by** *metis*
then obtain *i'* **where** *3*: *boxed-i T* Γ' *b bv i i'* **using** *GCons wfI-def subst-gb.simps* **by** *force*

have *boxed-i T* ((*x*′, *b*′, *c*′) $\#_\Gamma$ Γ′) *b bv i* (*i*′(*x*′ ↦ *s*′)) **proof**
  show *Some s* = *i x*′ **using** *1* **by** *auto*
  show *boxed-b T s b*′ *bv b s*′ **using** *2* **by** *auto*
  show *T* ; Γ′ ; *b* , *bv* ⊢ *i* ≈ *i*′ **using** *3* **by** *auto*
qed
thus *?case* **by** *auto*
qed

lemma *boxed-b-eq*:
assumes *boxed-b* Θ *s1 b bv b*′ *s1*′ **and** $\vdash_{wf}$ Θ
shows *wfTh* Θ ⟹ *boxed-b* Θ *s2 b bv b*′ *s2*′ ⟹ (*s1* = *s2*) = (*s1*′ = *s2*′)
using *assms* **proof**(*induct arbitrary*: *s2 s2*′ *rule*: *boxed-b.inducts*)
case (*boxed-b-BVar1I bv bv*′ *P s b*)
then show *?case*
  using *boxed-b-elims*(*1*) *rcl-val.eq-iff* **by** *metis*
next
case (*boxed-b-BVar2I bv bv*′ *P s b*)
then show *?case* **using** *boxed-b-elims*(*1*) **by** *metis*
next
case (*boxed-b-BIntI P s uu uv*)
hence *s2* = *s2*′ **using** *boxed-b-elims* **by** *metis*
then show *?case* **by** *auto*
next
case (*boxed-b-BBoolI P s uw ux*)
hence *s2* = *s2*′ **using** *boxed-b-elims* **by** *metis*
then show *?case* **by** *auto*
next
case (*boxed-b-BUnitI P s uy uz*)
hence *s2* = *s2*′ **using** *boxed-b-elims* **by** *metis*
then show *?case* **by** *auto*
next
case (*boxed-b-BPairI P s1 b1 bv b s1*′ *s2a b2 s2a*′)
then show *?case*
  by (*metis boxed-b-elims*(*5*) *rcl-val.eq-iff*(*4*))
next
case (*boxed-b-BConsI tyid dclist P dc x b c s1 bv b*′ *s1*′)
obtain *s22* **and** *s22*′ *dclist2 dc2 x2 b2 c2* **where** ∗:*s2* = *SCons tyid dc2 s22* ∧ *s2*′ = *SCons tyid dc2 s22*′ ∧ *boxed-b P s22 b2 bv b*′ *s22*′
  ∧ *AF-typedef tyid dclist2* ∈ *set P* ∧ (*dc2*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2* **using** *boxed-b-elims*(*6*)[*OF boxed-b-BConsI*(*6*)] **by** *metis*
show *?case* **proof**(*cases dc* = *dc2*)
  case *True*
  hence *b* = *b2* **using** *wfTh-ctor-unique τ.eq-iff wfTh-dclist-unique wf boxed-b-BConsI* ∗ **by** *metis*
  then show *?thesis* **using** *boxed-b-BConsI True* ∗ **by** *auto*
next
  case *False*
  then show *?thesis* **using** ∗ *boxed-b-BConsI* **by** *simp*
qed
next
case (*boxed-b-Bbitvec P s bv b*)
hence *s2* = *s2*′ **using** *boxed-b-elims* **by** *metis*
then show *?case* **by** *auto*

next
  case (*boxed-b-BConspI tyid bva dclist P b1 bv b′ s1 s1′ dc x b c*)
  obtain *bva2 s22 s22′ dclist2 dc2 x2 b2 c2* **where** ∗:
    $s2 = SConsp\ tyid\ dc2\ b1[bv{::=}b']_{bb}\ s22\ \wedge$
    $s2' = SConsp\ tyid\ dc2\ b1\ s22' \wedge$
    $boxed\text{-}b\ P\ s22\ b2[bva2{::=}b1]_{bb}\ \ bv\ b'\ s22'\ \wedge$
   AF-typedef-poly tyid bva2 dclist2 ∈ *set P* ∧ (*dc2*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2* **using** *boxed-b-elims*(*8*)[*OF boxed-b-BConspI*(*7*)] **by** *metis*
  show *?case* **proof**(*cases dc* = *dc2*)
    case *True*
    hence *AF-typedef-poly tyid bva2 dclist2* ∈ *set P* ∧ (*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist2* **using** ∗ **by** *auto*
     hence $b[bva{::=}b1]_{bb} = b2[bva2{::=}b1]_{bb}$ **using** *wfTh-typedef-poly-b-eq-iff*[*OF boxed-b-BConspI*(*1*) *boxed-b-BConspI*(*3*)] ∗ *boxed-b-BConspI* **by** *metis*
    then show *?thesis* **using** *boxed-b-BConspI True* ∗ **by** *auto*
  next
    case *False*
    then show *?thesis* **using** ∗ *boxed-b-BConspI* **by** *simp*
  qed
qed

lemma *bs-boxed-var*:
  assumes *boxed-i* Θ Γ *b′ bv i i′*
  shows *Some* (*b*,*c*) = *lookup* Γ *x* ⟹ *Some s* = *i x* ⟹ *Some s′* = *i′ x* ⟹ *boxed-b* Θ *s b bv b′ s′*
  using *assms* **proof**(*induct rule*: *boxed-i.inducts*)
  case (*boxed-i-GNilI T i*)
  then show *?case* **using** *lookup.simps* **by** *auto*
next
  case (*boxed-i-GConsI s i x1* Θ *b1 bv b′ s′* Γ *i′ c*)
  show *?case* **proof** (*cases x*=*x1*)
    case *True*
    then show *?thesis* **using** *boxed-i-GConsI*
       fun-upd-same lookup.simps(*2*) *option.inject prod.inject* **by** *metis*
  next
    case *False*
    then show *?thesis* **using** *boxed-i-GConsI*
       fun-upd-same lookup.simps option.inject prod.inject **by** *auto*
  qed
qed

lemma *eval-l-boxed-b*:
  assumes ⟦ *l* ⟧ = *s*
  shows *boxed-b* Θ *s* (*base-for-lit l*) *bv b′ s*
  using *assms* **proof**(*nominal-induct l arbitrary*: *s rule*:*l.strong-induct*)
qed(*auto simp add*: *boxed-b.intros wfRCV.intros*)+

lemma *boxed-i-eval-v-boxed-b*:
  fixes *v*::*v*
  assumes *boxed-i* Θ Γ *b′ bv i i′* **and** *i* ⟦ $v[bv{::=}b']_{vb}$ ⟧ ~ *s* **and** *i′* ⟦ *v* ⟧ ~ *s′* **and** *wfV* Θ *B* Γ *v b* **and** *wfI* Θ Γ *i′*
  shows *boxed-b* Θ *s b bv b′ s′*
  using *assms* **proof**(*nominal-induct v arbitrary*: *s s′ b rule*:*v.strong-induct*)

case (*V-lit l*)
hence $[\![\ l\ ]\!] = s \wedge [\![\ l\ ]\!] = s'$ **using** *eval-v-elims* **by** *auto*
moreover have b = *base-for-lit l* **using** *wfV-elims*(*2*) *V-lit* **by** *metis*
ultimately show *?case* **using** *V-lit* **using** *eval-l-boxed-b subst-b-base-for-lit* **by** *metis*
next
case (*V-var x*)
hence *Some* $s = i\ x \wedge$ *Some* $s' = i'\ x$ **using** *eval-v-elims subst-vb.simps* **by** *metis*
moreover obtain *c1* **where** *bc*:*Some* (*b*,*c1*) = *lookup* Γ *x* **using** *wfV-elims V-var* **by** *metis*
ultimately show *?case* **using** *bs-boxed-var V-var* **by** *metis*

next
case (*V-pair v1 v2*)
then obtain *b1* **and** *b2* **where** *b*:*b*=*B-pair b1 b2* **using** *wfV-elims subst-vb.simps* **by** *metis*
obtain *s1* **and** *s2* **where** *s*: *eval-v i* ($v1[bv{::=}b']_{vb}$) *s1* $\wedge$ *eval-v i* ($v2[bv{::=}b']_{vb}$) *s2* $\wedge$ *s* = *SPair s1 s2* **using** *eval-v-elims V-pair subst-vb.simps* **by** *metis*
obtain *s1'* **and** *s2'* **where** *s'*: *eval-v i' v1 s1'* $\wedge$ *eval-v i' v2 s2'* $\wedge$ *s'* = *SPair s1' s2'* **using** *eval-v-elims V-pair* **by** *metis*
have *boxed-b* Θ (*SPair s1 s2*) (*B-pair b1 b2*) *bv b'* (*SPair s1' s2'*) **proof**(*rule boxed-b-BPairI*)
show *boxed-b* Θ *s1 b1 bv b' s1'* **using** *V-pair eval-v-elims wfV-elims b s s' b.eq-iff* **by** *metis*
show *boxed-b* Θ *s2 b2 bv b' s2'* **using** *V-pair eval-v-elims wfV-elims b s s' b.eq-iff* **by** *metis*
qed
then show *?case* **using** *s s' b* **by** *auto*
next
case (*V-cons tyid dc v1*)

obtain *dclist x b1 c* **where** $*$: *b* = *B-id tyid* $\wedge$ *AF-typedef tyid dclist* $\in$ *set* $\Theta \wedge$ (*dc*, $\{\!\!\{\ x : b1\ \mid c\ \}\!\!\}$) $\in$ *set dclist* $\wedge\ \Theta\ ;\ B\ ;\ \Gamma \vdash_{wf} v1 : b1$
using *wfV-elims*(*4*)[*OF V-cons*(*5*)] *V-cons* **by** *metis*
obtain *s2* **where** *s2*: *s* = *SCons tyid dc s2* $\wedge$ *i* $[\![\ (v1[bv{::=}b']_{vb})\ ]\!] \sim s2$ **using** *eval-v-elims V-cons subst-vb.simps* **by** *metis*
obtain *s2'* **where** *s2'*: *s'* = *SCons tyid dc s2'* $\wedge$ *i'* $[\![\ v1\ ]\!] \sim s2'$ **using** *eval-v-elims V-cons* **by** *metis*
have *sp*: *supp* $\{\!\!\{\ x : b1\ \mid c\ \}\!\!\} = \{\}$ **using** *wfTh-lookup-supp-empty* $*$ *wfX-wfY* **by** *metis*

have *boxed-b* Θ (*SCons tyid dc s2*) (*B-id tyid*) *bv b'* (*SCons tyid dc s2'*)
proof(*rule boxed-b-BConsI*)
show *1*:*AF-typedef tyid dclist* $\in$ *set* Θ **using** $*$ **by** *auto*
show *2*:(*dc*, $\{\!\!\{\ x : b1\ \mid c\ \}\!\!\}$) $\in$ *set dclist* **using** $*$ **by** *auto*
have *bvf*:*atom bv* $\sharp$ *b1* **using** *sp* τ*.fresh fresh-def* **by** *auto*
show $\Theta \vdash s2 \sim b1\ [\ bv ::= b'\ ] \setminus s2'$ **using** *V-cons s2 s2'* $*$ **by** *metis*
qed
then show *?case* **using** $*$ *s2 s2'* **by** *simp*
next
case (*V-consp tyid dc b1 v1*)

obtain *bv2 dclist x2 b2 c2* **where** $*$: *b* = *B-app tyid b1* $\wedge$ *AF-typedef-poly tyid bv2 dclist* $\in$ *set* $\Theta \wedge$ (*dc*, $\{\!\!\{\ x2 : b2\ \mid c2\ \}\!\!\}$) $\in$ *set dclist* $\wedge\ \Theta\ ;\ B\ ;\ \Gamma \vdash_{wf} v1 : b2[bv2{::=}b1]_{bb}$
using *wf-strong-elim*(*1*)[*OF V-consp* (*5*)] **by** *metis*

obtain *s2* **where** *s2*: *s* = *SConsp tyid dc* $b1[bv{::=}b']_{bb}$ *s2* $\wedge$ *i* $[\![\ (v1[bv{::=}b']_{vb})\ ]\!] \sim s2$
using *eval-v-elims V-consp subst-vb.simps* **by** *metis*

obtain *s2'* **where** *s2'*: *s'* = *SConsp tyid dc b1 s2'* $\wedge$ *i'* $[\![\ v1\ ]\!] \sim s2'$

using *eval-v-elims V-consp* **by** *metis*

have $\vdash_{wf}$ Θ **using** *V-consp wfX-wfY* **by** *metis*
then obtain *bv3*::*bv* **and** *dclist3 x3 b3 c3* **where** ∗∗: *AF-typedef-poly tyid bv2 dclist* = *AF-typedef-poly tyid bv3 dclist3* ∧
(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3* ∧ *atom bv3* ♯ (*b1*, *bv*, *b'*, *s2*, *s2'*) ∧ *b2*[*bv2*::=*b1*]$_{bb}$ = *b3*[*bv3*::=*b1*]$_{bb}$
using ∗ *obtain-fresh-bv-dclist-b-iff*[**where** *tm*=(*b1*, *bv*, *b'*, *s2*, *s2'*)] **by** *metis*

have *boxed-b* Θ (*SConsp tyid dc b1*[*bv*::=*b'*]$_{bb}$ *s2*) (*B-app tyid b1*) *bv b'* (*SConsp tyid dc b1 s2'*)
proof(*rule boxed-b-BConspI*[*of tyid bv3 dclist3* Θ, **where** *x*=*x3* **and** *b*=*b3* **and** *c*=*c3*])
show *1*:*AF-typedef-poly tyid bv3 dclist3* ∈ *set* Θ **using** ∗ ∗∗ **by** *auto*
show *2*:(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3* **using** ∗∗ **by** *auto*
show *atom bv3* ♯ (*b1*, *bv*, *b'*, *s2*, *s2'*) **using** ∗∗ **by** *auto*
show Θ ⊢ *s2* ~ *b3*[*bv3*::=*b1*]$_{bb}$ [*bv* ::= *b'*] \ *s2'* **using** *V-consp s2 s2'* ∗ ∗∗ **by** *metis*
qed
then show *?case* **using** ∗ *s2 s2'* **by** *simp*
qed

lemma *boxed-b-eq-eq*:
assumes *boxed-b* Θ *n1 b1 bv b' n1'* **and** *boxed-b* Θ *n2 b1 bv b' n2'* **and** *s* = *SBool* (*n1* = *n2*) **and** $\vdash_{wf}$ Θ
s' = *SBool* (*n1'* = *n2'*)
shows *s*=*s'*
using *boxed-b-eq assms* **by** *auto*

lemma *boxed-i-eval-ce-boxed-b*:
fixes *e*::*ce*
assumes *i'* ⟦ *e* ⟧ ~ *s'* **and** *i* ⟦ *e*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *s* **and** *wfCE* Θ *B* Γ *e b* **and** *boxed-i* Θ Γ *b' bv i i'* **and** *wfI* Θ Γ *i'*
shows *boxed-b* Θ *s b bv b' s'*
using *assms* **proof**(*nominal-induct e arbitrary*: *s s' b b' rule*: *ce.strong-induct*)
case (*CE-val x*)
then show *?case* **using** *boxed-i-eval-v-boxed-b eval-e-elims wfCE-elims subst-ceb.simps* **by** *metis*
next
case (*CE-op opp v1 v2*)

show *?case* **proof**(*rule opp.exhaust*)
assume ‹*opp* = *Plus*›
have *1*:*wfCE* Θ *B* Γ *v1* (*B-int*) **using** *wfCE-elims CE-op* ‹*opp* = *Plus*› **by** *metis*
have *2*:*wfCE* Θ *B* Γ *v2* (*B-int*) **using** *wfCE-elims CE-op* ‹*opp* = *Plus*› **by** *metis*
have ∗:*b* = *B-int* **using** *CE-op wfCE-elims*
by (*metis* ‹*opp* = *plus*›)

obtain *n1* **and** *n2* **where** *n*:*s* = *SNum* (*n1* + *n2*) ∧ *i* ⟦ *v1*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SNum n1* ∧ *i* ⟦ *v2*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SNum n2* **using** *eval-e-elims CE-op subst-ceb.simps* ‹*opp* = *plus*› **by** *metis*
obtain *n1'* **and** *n2'* **where** *n'*:*s'* = *SNum* (*n1'* + *n2'*) ∧ *i'* ⟦ *v1* ⟧ ~ *SNum n1'* ∧ *i'* ⟦ *v2* ⟧ ~ *SNum n2'* **using** *eval-e-elims Plus CE-op* ‹*opp* = *plus*› **by** *metis*

have *boxed-b* Θ (*SNum n1*) *B-int bv b'* (*SNum n1'*) **using** *boxed-i-eval-v-boxed-b 1 2 n n' CE-op* ‹*opp* = *plus*› **by** *metis*
moreover have *boxed-b* Θ (*SNum n2*) *B-int bv b'* (*SNum n2'*) **using** *boxed-i-eval-v-boxed-b 1 2 n*

n′ CE-op **by** *metis*
ultimately have *s=s′* **using** *n′ n boxed-b-elims*(*2*)
by (*metis rcl-val.eq-iff* (*2*))
thus *?thesis* **using** ∗ *n n′ boxed-b-BIntI CE-op wfRCV.intros Plus* **by** *simp*
next
assume ‹*opp* = *LEq*›
have *1*:*wfCE* Θ *B* Γ *v1* (*B-int*) **using** *wfCE-elims CE-op* ‹*opp* = *LEq*› **by** *metis*
have *2*:*wfCE* Θ *B* Γ *v2* (*B-int*) **using** *wfCE-elims CE-op* ‹*opp* = *LEq*› **by** *metis*
hence ∗:*b* = *B-bool* **using** *CE-op wfCE-elims* ‹*opp* = *LEq*› **by** *metis*
obtain *n1* **and** *n2* **where** *n*:*s* = *SBool* (*n1* ≤ *n2*) ∧ *i* ⟦ $v1[bv::=b']_{ceb}$ ⟧ ~ *SNum n1* ∧ *i* ⟦ $v2[bv::=b']_{ceb}$ ⟧ ~ *SNum n2* **using** *eval-e-elims subst-ceb.simps CE-op* ‹*opp* = *LEq*› **by** *metis*
obtain *n1′* **and** *n2′* **where** *n′*:*s′* = *SBool* (*n1′* ≤ *n2′*) ∧ *i′* ⟦ *v1* ⟧ ~ *SNum n1′* ∧ *i′* ⟦ *v2* ⟧ ~ *SNum n2′* **using** *eval-e-elims CE-op* ‹*opp* = *LEq*› **by** *metis*

have *boxed-b* Θ (*SNum n1*) *B-int bv b′* (*SNum n1′*) **using** *boxed-i-eval-v-boxed-b 1 2 n n′ CE-op* **by** *metis*
moreover have *boxed-b* Θ (*SNum n2*) *B-int bv b′* (*SNum n2′*) **using** *boxed-i-eval-v-boxed-b 1 2 n n′ CE-op* **by** *metis*
ultimately have *s=s′* **using** *n′ n boxed-b-elims*(*2*)
by (*metis rcl-val.eq-iff* (*2*))
thus *?thesis* **using** ∗ *n n′ boxed-b-BBoolI CE-op wfRCV.intros* ‹*opp* = *LEq*› **by** *simp*
next
assume ‹*opp* = *Eq*›
obtain *b1* **where** *b1*:*wfCE* Θ *B* Γ *v1 b1* ∧ *wfCE* Θ *B* Γ *v2 b1* **using** *wfCE-elims CE-op* ‹*opp* = *Eq*› **by** *metis*

hence ∗:*b* = *B-bool* **using** *CE-op wfCE-elims* ‹*opp* = *Eq*› **by** *metis*
obtain *n1* **and** *n2* **where** *n*:*s* = *SBool* (*n1* = *n2*) ∧ *i* ⟦ $v1[bv::=b']_{ceb}$ ⟧ ~ *n1* ∧ *i* ⟦ $v2[bv::=b']_{ceb}$ ⟧ ~ *n2* **using** *eval-e-elims subst-ceb.simps CE-op* ‹*opp* = *Eq*› **by** *metis*
obtain *n1′* **and** *n2′* **where** *n′*:*s′* = *SBool* (*n1′* = *n2′*) ∧ *i′* ⟦ *v1* ⟧ ~ *n1′* ∧ *i′* ⟦ *v2* ⟧ ~ *n2′* **using** *eval-e-elims CE-op* ‹*opp* = *Eq*› **by** *metis*

have *boxed-b* Θ *n1 b1 bv b′ n1′* **using** *boxed-i-eval-v-boxed-b b1 n n′ CE-op* **by** *metis*
moreover have *boxed-b* Θ *n2 b1 bv b′ n2′* **using** *boxed-i-eval-v-boxed-b b1 n n′ CE-op* **by** *metis*
moreover have $\vdash_{wf}$ Θ **using** *b1 wfX-wfY* **by** *metis*
ultimately have *s=s′* **using** *n′ n boxed-b-elims*
boxed-b-eq-eq **by** *metis*
thus *?thesis* **using** ∗ *n n′ boxed-b-BBoolI CE-op wfRCV.intros* ‹*opp* = *Eq*› **by** *simp*
qed

next
case (*CE-concat v1 v2*)

obtain *bv1* **and** *bv2* **where** *s* : *s* = *SBitvec* (*bv1* @ *bv2*) ∧ (*i* ⟦ $v1[bv::=b']_{ceb}$ ⟧ ~ *SBitvec bv1*) ∧ *i* ⟦ $v2[bv::=b']_{ceb}$ ⟧ ~ *SBitvec bv2*
using *eval-e-elims*(*7*) *subst-ceb.simps CE-concat.prems*(*2*) *eval-e-elims*(*6*) *subst-ceb.simps*(*6*) **by** *metis*
obtain *bv1′* **and** *bv2′* **where** *s′* : *s′* = *SBitvec* (*bv1′* @ *bv2′*) ∧ *i′* ⟦ *v1* ⟧ ~ *SBitvec bv1′* ∧ *i′* ⟦ *v2* ⟧ ~ *SBitvec bv2′*
using *eval-e-elims*(*7*) *CE-concat* **by** *metis*

then show *?case* **using** *boxed-i-eval-v-boxed-b wfCE-elims s s′ CE-concat*

by (*metis CE-concat.prems*(*3*) *assms assms*(*5*) *wfRCV-BBitvecI boxed-b-Bbitvec boxed-b-elims*(*7*) *eval-e-concatI eval-e-uniqueness*)
next
case (*CE-fst ce*)
obtain *s2* **where** *1*:i ⟦ $ce[bv::=b']_{ceb}$ ⟧ ~ *SPair s s2* **using** *CE-fst eval-e-elims subst-ceb.simps* **by** *metis*
obtain *s2'* **where** *2*:i' ⟦ *ce* ⟧ ~ *SPair s' s2'* **using** *CE-fst eval-e-elims* **by** *metis*
obtain *b2* **where** *3*:*wfCE* Θ *B* Γ *ce* (*B-pair b b2*) **using** *wfCE-elims*(*4*) *CE-fst* **by** *metis*

have *boxed-b* Θ (*SPair s s2*) (*B-pair b b2*) *bv b'* (*SPair s' s2'*)
using *1 2 3 CE-fst boxed-i-eval-v-boxed-b boxed-b-BPairI* **by** *auto*
thus *?case* **using** *boxed-b-elims*(*5*) **by** *force*
next
case (*CE-snd v*)
obtain *s1* **where** *1*:i ⟦ $v[bv::=b']_{ceb}$ ⟧ ~ *SPair s1 s* **using** *CE-snd eval-e-elims subst-ceb.simps* **by** *metis*
obtain *s1'* **where** *2*:i' ⟦ *v* ⟧ ~ *SPair s1' s'* **using** *CE-snd eval-e-elims* **by** *metis*
obtain *b1* **where** *3*:*wfCE* Θ *B* Γ *v* (*B-pair b1 b*) **using** *wfCE-elims*(*5*) *CE-snd* **by** *metis*

have *boxed-b* Θ (*SPair s1 s*) (*B-pair b1 b*) *bv b'* (*SPair s1' s'*) **using** *1 2 3 CE-snd boxed-i-eval-v-boxed-b* **by** *simp*
thus *?case* **using** *boxed-b-elims*(*5*) **by** *force*
next
case (*CE-len v*)
obtain *s1* **where** *s*: i ⟦ $v[bv::=b']_{ceb}$ ⟧ ~ *SBitvec s1* **using** *CE-len eval-e-elims subst-ceb.simps* **by** *metis*
obtain *s1'* **where** *s'*: i' ⟦ *v* ⟧ ~ *SBitvec s1'* **using** *CE-len eval-e-elims* **by** *metis*

have Θ ; *B* ; Γ $\vdash_{wf}$ *v* : *B-bitvec* ∧ *b* = *B-int* **using** *wfCE-elims CE-len* **by** *metis*
then show *?case* **using** *boxed-i-eval-v-boxed-b s s' CE-len*
by (*metis boxed-b-BIntI boxed-b-elims*(*7*) *eval-e-lenI eval-e-uniqueness subst-ceb.simps*(*5*) *wfI-wfCE-eval-e*)
qed

lemma *eval-c-eq-bs-boxed*:
fixes *c*::*c*
assumes i ⟦ $c[bv::=b]_{cb}$ ⟧ ~ *s* **and** i' ⟦ *c* ⟧ ~ *s'* **and** *wfC* Θ *B* Γ *c* **and** *wfI* Θ Γ i' **and** Θ ; $\Gamma[bv::=b]_{\Gamma b}$ ⊢ *i*
and *boxed-i* Θ Γ *b bv i i'*
shows *s* = *s'*
using *assms* **proof**(*nominal-induct c arbitrary*: *s s'* *rule*:*c.strong-induct*)
case *C-true*
then show *?case* **using** *eval-c-elims subst-cb.simps* **by** *metis*
next
case *C-false*
then show *?case* **using** *eval-c-elims subst-cb.simps* **by** *metis*
next
case (*C-conj c1 c2*)
obtain *s1* **and** *s2* **where** *1*: *eval-c i* ($c1[bv::=b]_{cb}$) *s1* ∧ *eval-c i* ($c2[bv::=b]_{cb}$) *s2* ∧ *s* = (*s1*∧*s2*) **using** *C-conj eval-c-elims*(*3*) *subst-cb.simps*(*3*) **by** *metis*
obtain *s1'* **and** *s2'* **where** *2*:*eval-c i' c1 s1'* ∧ *eval-c i' c2 s2'* ∧ *s'* = (*s1'*∧*s2'*) **using** *C-conj eval-c-elims*(*3*) **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-conj* **by** *metis*

next
  case (*C-disj c1 c2*)

obtain *s1* **and** *s2* **where** *1*: *eval-c i* ($c1[bv::=b]_{cb}$) *s1* ∧ *eval-c i* ($c2[bv::=b]_{cb}$) *s2* ∧ *s* = (*s1*∨*s2*) **using** *C-disj eval-c-elims*(*4*) *subst-cb.simps*(*4*) **by** *metis*
obtain *s1'* **and** *s2'* **where** *2*:*eval-c i' c1 s1'* ∧ *eval-c i' c2 s2'* ∧ *s'* = (*s1'*∨*s2'*) **using** *C-disj eval-c-elims*(*4*) **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-disj* **by** *metis*
next
  case (*C-not c*)
obtain *s1*::*bool* **where** *1*: (*i* ⟦ $c[bv::=b]_{cb}$ ⟧ ~ *s1*) ∧ (*s* = (¬ *s1*)) **using** *C-not eval-c-elims*(*6*) *subst-cb.simps*(*7*) **by** *metis*
obtain *s1'*::*bool* **where** *2*: (*i'* ⟦ *c* ⟧ ~ *s1'*) ∧ (*s'* = (¬ *s1'*)) **using** *C-not eval-c-elims*(*6*) **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-not* **by** *metis*
next
  case (*C-imp c1 c2*)
obtain *s1* **and** *s2* **where** *1*: *eval-c i* ($c1[bv::=b]_{cb}$) *s1* ∧ *eval-c i* ($c2[bv::=b]_{cb}$) *s2* ∧ *s* = (*s1* ⟶ *s2*) **using** *C-imp eval-c-elims*(*5*) *subst-cb.simps*(*5*) **by** *metis*
obtain *s1'* **and** *s2'* **where** *2*:*eval-c i' c1 s1'* ∧ *eval-c i' c2 s2'* ∧ *s'* = (*s1'* ⟶ *s2'*) **using** *C-imp eval-c-elims*(*5*) **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-imp* **by** *metis*
next
  case (*C-eq e1 e2*)
obtain *be* **where** *be*: *wfCE* Θ *B* Γ *e1 be* ∧ *wfCE* Θ *B* Γ *e2 be* **using** *C-eq wfC-elims* **by** *metis*
obtain *s1* **and** *s2* **where** *1*: *eval-e i* ($e1[bv::=b]_{ceb}$) *s1* ∧ *eval-e i* ($e2[bv::=b]_{ceb}$) *s2* ∧ *s* = (*s1* = *s2*) **using** *C-eq eval-c-elims*(*7*) *subst-cb.simps*(*6*) **by** *metis*
obtain *s1'* **and** *s2'* **where** *2*:*eval-e i' e1 s1'* ∧ *eval-e i' e2 s2'* ∧ *s'* = (*s1'* = *s2'*) **using** *C-eq eval-c-elims*(*7*) **by** *metis*
have $\vdash_{wf}$ Θ **using** *C-eq wfX-wfY* **by** *metis*
moreover have Θ ; $\Gamma[bv::=b]_{\Gamma b}$ ⊢ *i* **using** *C-eq* **by** *auto*
ultimately show *?case* **using** *boxed-b-eq*[*of* Θ *s1 be bv b s1' s2 s2'*] *1 2 boxed-i-eval-ce-boxed-b C-eq wfC-elims subst-cb.simps 1 2 be* **by** *auto*
qed

lemma *is-satis-bs-boxed*:
  fixes *c*::*c*
  assumes *boxed-i* Θ Γ *b bv i i'* **and** *wfC* Θ *B* Γ *c* **and** *wfI* Θ $\Gamma[bv::=b]_{\Gamma b}$ *i* **and** Θ ; Γ ⊢ *i'*
    and (*i* ⊨ $c[bv::=b]_{cb}$)
  shows (*i'* ⊨ *c*)
proof −
  have *eval-c i* ($c[bv::=b]_{cb}$) *True* **using** *is-satis.simps assms* **by** *auto*
  moreover obtain *s* **where** *i'* ⟦ *c* ⟧ ~ *s* **using** *eval-c-exist assms* **by** *metis*
  ultimately show *?thesis* **using** *eval-c-eq-bs-boxed assms is-satis.simps* **by** *metis*
qed

lemma *is-satis-bs-boxed-rev*:
  fixes *c*::*c*
  assumes *boxed-i* Θ Γ *b bv i i'* **and** *wfC* Θ *B* Γ *c* **and** *wfI* Θ $\Gamma[bv::=b]_{\Gamma b}$ *i* **and** Θ ; Γ ⊢ *i'* **and** *wfC* Θ {||} $\Gamma[bv::=b]_{\Gamma b}$ ($c[bv::=b]_{cb}$)
    and (*i'* ⊨ *c*)
  shows (*i* ⊨ $c[bv::=b]_{cb}$)
proof −

have *eval-c* i' *c True* **using** *is-satis.simps assms* **by** *auto*
moreover obtain *s* **where** i ⟦ $c[bv{::=}b]_{cb}$ ⟧ ~ *s* **using** *eval-c-exist assms* **by** *metis*
ultimately show *?thesis* **using** *eval-c-eq-bs-boxed assms is-satis.simps* **by** *metis*
qed

lemma *bs-boxed-wfi-aux*:
fixes *b*::*b* **and** *bv*::*bv* **and** Θ::Θ **and** *B*::*ℬ*
assumes *boxed-i* Θ Γ *b bv i* i' **and** *wfI* Θ $\Gamma[bv{::=}b]_{\Gamma b}$ *i* **and** $\vdash_{wf}$ Θ **and** *wfG* Θ *B* Γ
shows Θ ; Γ ⊢ i'
using *assms* **proof**(*induct rule*: *boxed-i.inducts*)
case (*boxed-i-GNilI T i*)
then show *?case* **using** *wfI-def* **by** *auto*
next
case (*boxed-i-GConsI s i x1 T b1 bv b s' G i' c1*)
{
fix *x2 b2 c2*
assume *as* : (*x2*,*b2*,*c2*) ∈ *toSet* ((*x1*, *b1*, *c1*) $\#_\Gamma$ *G*)

then consider (*hd*) (*x2*,*b2*,*c2*) = (*x1*, *b1*, *c1*) | (*tail*) (*x2*,*b2*,*c2*) ∈ *toSet G* **using** *toSet.simps* **by** *auto*
hence ∃ *s*. *Some s* = (i'(*x1* ↦ s')) *x2* ∧ *wfRCV T s b2* **proof**(*cases*)
case *hd*
hence *b1*=*b2* **by** *auto*
moreover have (*x2*,$b2[bv{::=}b]_{bb}$,$c2[bv{::=}b]_{cb}$) ∈ *toSet* ((*x1*, *b1*, *c1*) $\#_\Gamma$ *G*)$[bv{::=}b]_{\Gamma b}$ **using** *hd subst-gb.simps* **by** *simp*
moreover hence *wfRCV T s* $b2[bv{::=}b]_{bb}$ **using** *wfI-def boxed-i-GConsI hd*
proof −
obtain *ss* :: *b* ⇒ *x* ⇒ (*x* ⇒ *rcl-val option*) ⇒ *type-def list* ⇒ *rcl-val* **where**
∀ *x1a x2a x3 x4*. (∃ *v5*. *Some v5* = *x3 x2a* ∧ *wfRCV x4 v5 x1a*) = (*Some* (*ss x1a x2a x3 x4*) = *x3 x2a* ∧ *wfRCV x4* (*ss x1a x2a x3 x4*) *x1a*)
by *moura*
then have *f1*: *Some* (*ss* $b2[bv{::=}b]_{bb}$ *x1 i T*) = *i x1* ∧ *wfRCV T* (*ss* $b2[bv{::=}b]_{bb}$ *x1 i T*) $b2[bv{::=}b]_{bb}$
using *boxed-i-GConsI.prems*(*1*) *hd wfI-def* **by** *auto*
then have *ss* $b2[bv{::=}b]_{bb}$ *x1 i T* = *s*
by (*metis* (*no-types*) *boxed-i-GConsI.hyps*(*1*) *option.inject*)
then show *?thesis*
using *f1* **by** *blast*
qed
ultimately have *wfRCV T* s' *b2* **using** *boxed-i-GConsI boxed-b-wfRCV* **by** *metis*

then show *?thesis* **using** *hd* **by** *simp*
next
case *tail*
hence *wfI T G* i' **using** *boxed-i-GConsI wfI-suffix wfG-suffix subst-gb.simps*
by (*metis* (*no-types*, *lifting*) *Un-iff toSet.simps*(*2*) *wfG-cons2 wfI-def*)
then show *?thesis* **using** *wfI-def*[*of T G* i'] *tail*
using *boxed-i-GConsI.prems*(*3*) *split-G wfG-cons-fresh2* **by** *fastforce*
qed
}
thus *?case* **using** *wfI-def* **by** *fast*

qed

lemma *is-satis-g-bs-boxed-aux*:
  fixes *G*::Γ
  assumes *boxed-i* Θ *G1 b bv i i′* **and** *wfI* Θ *G1*[*bv*::=*b*]$_{\Gamma b}$ *i* **and** *wfI* Θ *G1 i′* **and** *G1* = (*G2*@*G*) **and** *wfG* Θ *B G1*
    and (*i* ⊨ *G*[*bv*::=*b*]$_{\Gamma b}$)
  shows (*i′* ⊨ *G*)
  using *assms* **proof**(*induct G arbitrary*: *G2 rule*: Γ-*induct*)
  case *GNil*
  then show *?case* **by** *auto*
next
  case (*GCons x′ b′ c′* Γ′ *G2*)
  show *?case* **proof**(*subst is-satis-g.simps*,*rule*)
    have ∗:*wfC* Θ *B G1 c′* **using** *GCons wfG-wfC-inside* **by** *force*
    show *i′* ⊨ *c′* **using** *is-satis-bs-boxed*[*OF assms*(*1*) ∗] *GCons* **by** *auto*
    obtain *G3* **where** *G1* = *G3* @ Γ′ **using** *GCons append-g.simps*
      by (*metis append-g-assoc*)
    then show *i′* ⊨ Γ′ **using** *GCons append-g.simps* **by** *simp*
  qed
qed

lemma *is-satis-g-bs-boxed*:
  fixes *G*::Γ
  assumes *boxed-i* Θ *G b bv i i′* **and** *wfI* Θ *G*[*bv*::=*b*]$_{\Gamma b}$ *i* **and** *wfI* Θ *G i′* **and** *wfG* Θ *B G*
    and (*i* ⊨ *G*[*bv*::=*b*]$_{\Gamma b}$)
  shows (*i′* ⊨ *G*)
  using *is-satis-g-bs-boxed-aux assms*
  by (*metis* (*full-types*) *append-g.simps*(*1*))

lemma *subst-b-valid*:
  fixes *s*::*s* **and** *b*::*b*
  assumes Θ ; {||} ⊢$_{wf}$ *b* **and** *B* = {|*bv*|} **and** Θ ; {|*bv*|} ;Γ ⊨ *c*
  shows Θ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊨ *c*[*bv*::=*b*]$_{cb}$
proof(*rule validI*)

  show ∗∗:Θ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊢$_{wf}$ *c*[*bv*::=*b*]$_{cb}$ **using** *assms valid.simps wf-b-subst subst-gb.simps* **by** *metis*
  show ∀ *i*. (*wfI* Θ Γ[*bv*::=*b*]$_{\Gamma b}$ *i* ∧ *i* ⊨ Γ[*bv*::=*b*]$_{\Gamma b}$) ⟶ *i* ⊨ *c*[*bv*::=*b*]$_{cb}$
  proof(*rule*,*rule*)
    fix *i*
    assume ∗:*wfI* Θ Γ[*bv*::=*b*]$_{\Gamma b}$ *i* ∧ *i* ⊨ Γ[*bv*::=*b*]$_{\Gamma b}$

    obtain *i′* **where** *idash*: *boxed-i* Θ Γ *b bv i i′* **using** *boxed-i-ex wfX-wfY assms* ∗ **by** *fastforce*

    have *wfc*: Θ ; {|*bv*|} ; Γ ⊢$_{wf}$ *c* **using** *valid.simps assms* **by** *simp*
    have *wfg*: Θ ; {|*bv*|} ⊢$_{wf}$ Γ **using** *valid.simps wfX-wfY assms* **by** *metis*
    hence *wfi*: *wfI* Θ Γ *i′* **using** *idash* ∗ *bs-boxed-wfi-aux subst-gb.simps wfX-wfY* **by** *metis*
    moreover have *i′* ⊨ Γ **proof** (*rule is-satis-g-bs-boxed*[*OF idash*] *wfX-wfY*(*2*)[*OF wfc*])
      show *wfI* Θ Γ[*bv*::=*b*]$_{\Gamma b}$ *i* **using** *subst-gb.simps* ∗ **by** *simp*
      show *wfI* Θ Γ *i′* **using** *wfi* **by** *auto*
      show Θ ; *B* ⊢$_{wf}$ Γ **using** *wfg assms* **by** *auto*

show $i \models \Gamma[bv::=b]_{\Gamma b}$ **using** *subst-gb.simps* $*$ **by** *simp*
qed
ultimately have $ic{:}i' \models c$ **using** *assms valid-def* **using** *valid.simps* **by** *blast*

show $i \models c[bv::=b]_{cb}$ **proof**(*rule is-satis-bs-boxed-rev*)
show Θ ; Γ ; b , $bv \vdash i \approx i'$ **using** *idash* **by** *auto*
show Θ ; B ; $\Gamma \vdash_{wf} c$ **using** *wfc assms* **by** *auto*
show Θ ; $\Gamma[bv::=b]_{\Gamma b} \vdash i$ **using** *subst-gb.simps* $*$ **by** *simp*
show Θ ; $\Gamma \vdash i'$ **using** *wfi* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} c[bv::=b]_{cb}$ **using** $**$ **by** *auto*
show $i' \models c$ **using** *ic* **by** *auto*
qed

qed
qed

11.7 Expression Operator Lemmas

lemma *is-satis-len-imp*:
assumes $i \models$ (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*))))) (**is** *is-satis i ?c1*)
shows $i \models$ (*CE-val* (*V-var x*) == *CE-len* $[$*V-lit* (*L-bitvec v*)$]^{ce}$)
proof −
have $*$:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
then have *eval-e i* (*CE-val* (*V-lit* (*L-num* (*int* (*length v*))))) (*SNum* (*int* (*length v*)))
using *eval-e-elims*(*1*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros*(*1*) *eval-v-litI*)
hence *eval-e i* (*CE-val* (*V-var x*)) (*SNum* (*int* (*length v*))) **using** *eval-c-elims*(*7*)[*OF* $*$]
by (*metis eval-e-elims*(*1*) *eval-v-elims*(*1*))
moreover have *eval-e i* (*CE-len* $[$*V-lit* (*L-bitvec v*)$]^{ce}$) (*SNum* (*int* (*length v*)))
using *eval-e-elims*(*7*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros eval-v-litI*)
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*
qed

lemma *is-satis-plus-imp*:
assumes $i \models$ (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) (**is** *is-satis i ?c1*)
shows $i \models$ (*CE-val* (*V-var x*) == *CE-op Plus* ($[$*V-lit* (*L-num n1*)$]^{ce}$) ($[$*V-lit* (*L-num n2*)$]^{ce}$))
proof −
have $*$:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
then have *eval-e i* (*CE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) (*SNum* (*n1*+*n2*))
using *eval-e-elims*(*1*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros*(*1*) *eval-v-litI*)
hence *eval-e i* (*CE-val* (*V-var x*)) (*SNum* (*n1*+*n2*)) **using** *eval-c-elims*(*7*)[*OF* $*$]
by (*metis eval-e-elims*(*1*) *eval-v-elims*(*1*))
moreover have *eval-e i* (*CE-op Plus* ($[$*V-lit* (*L-num n1*)$]^{ce}$) ($[$*V-lit* (*L-num n2*)$]^{ce}$)) (*SNum* (*n1*+*n2*))
using *eval-e-elims*(*7*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros eval-v-litI*)
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*
qed

lemma *is-satis-leq-imp*:
assumes $i \models$ (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*if* (*n1* $\leq$ *n2*) *then L-true else L-false*))) (**is** *is-satis i ?c1*)
shows $i \models$ (*CE-val* (*V-var x*) == *CE-op LEq* $[$(*V-lit* (*L-num n1*))$]^{ce}$ $[$(*V-lit* (*L-num n2*))$]^{ce}$)
proof −
have $*$:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*

then have *eval-e i* (*CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*)))) (*SBool* (*n1*≤*n2*))
using *eval-e-elims*(*1*) *eval-v-elims eval-l.simps*
by (*metis* (*full-types*) *eval-e.intros*(*1*) *eval-v-litI*)
hence *eval-e i* (*CE-val* (*V-var x*)) (*SBool* (*n1*≤*n2*)) **using** *eval-c-elims*(*7*)[*OF* ∗]
by (*metis eval-e-elims*(*1*) *eval-v-elims*(*1*))
moreover have *eval-e i* (*CE-op LEq* [(*V-lit* (*L-num n1*))]^{ce} [(*V-lit* (*L-num n2*))]^{ce}) (*SBool* (*n1*≤*n2*))
using *eval-e-elims*(*3*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros eval-v-litI*)
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*
qed

lemma *eval-lit-inj*:
fixes *n1*::*l* **and** *n2*::*l*
assumes ⟦ *n1* ⟧ = *s* **and** ⟦ *n2* ⟧ = *s*
shows *n1*=*n2*
using *assms* **proof**(*nominal-induct s rule*: *rcl-val.strong-induct*)
case (*SBitvec x*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case (*SNum x*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case (*SBool x*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case (*SPair x1a x2a*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case (*SCons x1a x2a x3a*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case (*SConsp x1a x2a x3a x4*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case *SUnit*
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
next
case (*SUt x*)
then show *?case* **using** *eval-l.simps*
by (*metis l.strong-exhaust rcl-val.distinct rcl-val.eq-iff*)
qed

lemma *eval-e-lit-inj*:
fixes *n1*::*l* **and** *n2*::*l*
assumes *i* ⟦ [[*n1*]^{v}]^{ce} ⟧ ~ *s* **and** *i* ⟦ [[*n2*]^{v}]^{ce} ⟧ ~ *s*
shows *n1*=*n2*

using *eval-lit-inj assms eval-e-elims eval-v-elims* **by** *metis*

lemma *is-satis-eq-imp*:
assumes $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}val\ (V\text{-}lit\ (if\ (n1 = n2)\ then\ L\text{-}true\ else\ L\text{-}false)))$ (**is** *is-satis i ?c1*)
shows $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}op\ Eq\ [(V\text{-}lit\ (n1))]^{ce}\ [(V\text{-}lit\ (n2))]^{ce})$
proof −
have ∗:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
then have *eval-e i (CE-val (V-lit ((if (n1=n2) then L-true else L-false)))) (SBool (n1=n2))*
using *eval-e-elims(1) eval-v-elims eval-l.simps*
by (*metis (full-types) eval-e.intros(1) eval-v-litI*)
hence *eval-e i (CE-val (V-var x)) (SBool (n1=n2))* **using** *eval-c-elims(7)[OF ∗]*
by (*metis eval-e-elims(1) eval-v-elims(1)*)
moreover have $eval\text{-}e\ i\ (CE\text{-}op\ Eq\ [(V\text{-}lit\ (n1))]^{ce}\ [(V\text{-}lit\ (n2)\ )]^{ce})\ (SBool\ (n1=n2))$
proof −
obtain *s1* **and** *s2* **where** $*{:}i\ [\![\ [\ [\ n1\ ]^{v}\ ]^{ce}\ ]\!]\sim s1\ \wedge\ i\ [\![\ [\ [\ n2\ ]^{v}\ ]^{ce}\ ]\!]\sim s2$ **using** *eval-l.simps eval-e.intros eval-v-litI* **by** *metis*
moreover have *SBool (n1 = n2) = SBool (s1 = s2)* **proof**(*cases n1=n2*)
case *True*
then show *?thesis* **using** ∗
by (*simp add: calculation eval-e-uniqueness*)
next
case *False*
then show *?thesis* **using** ∗ *eval-e-lit-inj* **by** *auto*
qed
ultimately show *?thesis* **using** $eval\text{-}e\text{-}eqI[of\ i\ [(V\text{-}lit\ (n1))]^{ce}\ s1\ [(V\text{-}lit\ (n2))]^{ce}\ s2\ ]$ **by** *auto*
qed
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*
qed

lemma *valid-eq-e*:
assumes $\forall\ i\ s1\ s2.\ wfG\ P\ \mathcal{B}\ GNil \wedge wfI\ P\ GNil\ i \wedge eval\text{-}e\ i\ e1\ s1 \wedge eval\text{-}e\ i\ e2\ s2 \longrightarrow s1 = s2$
and *wfCE P* $\mathcal{B}$ *GNil e1 b* **and** *wfCE P* $\mathcal{B}$ *GNil e2 b*
shows $P\ ;\ \mathcal{B}\ ;\ (x,\ b\ ,\ CE\text{-}val\ (V\text{-}var\ x)\ ==\ e1\ )\#_{\Gamma}\ GNil \models\ CE\text{-}val\ (V\text{-}var\ x)\ ==\ e2$
unfolding *valid.simps*
proof(*intro conjI*)
show ‹ $P\ ;\ \mathcal{B}\ ;\ (x,\ b,\ [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e1\ )\ \#_{\Gamma}\ GNil\ \ \vdash_{wf}\ [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e2$ ›
using *assms wf-intros wfX-wfY b.eq-iff fresh-GNil wfC-e-eq2 wfV-elims* **by** *meson*
show ‹ $\forall\ i.\ ((P\ ;\ (x,\ b,\ [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e1\ )\ \#_{\Gamma}\ GNil \vdash i) \wedge\ (i \models (x,\ b,\ [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e1\ )\ \#_{\Gamma}\ GNil)\ \longrightarrow$
$(i \models [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e2))$ › **proof**(*rule+*)
fix *i*
assume $as{:}P\ ;\ (x,\ b,\ [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e1\ )\ \#_{\Gamma}\ GNil \vdash i \wedge\ i \models (x,\ b,\ [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e1\ )\ \#_{\Gamma}\ GNil$

have ∗: *P ; GNil ⊢ i* **using** *wfI-def* **by** *auto*

then obtain *s1* **where** *s1*:*eval-e i e1 s1* **using** *assms eval-e-exist* **by** *metis*
obtain *s2* **where** *s2*:*eval-e i e2 s2* **using** *assms eval-e-exist* ∗ **by** *metis*
moreover have *i x = Some s1* **proof** −
have $i \models [\ [\ x\ ]^{v}\ ]^{ce}\ ==\ e1$ **using** *as is-satis-g.simps* **by** *auto*
thus *?thesis* **using** *s1*
by (*metis eval-c-elims(7) eval-e-elims(1) eval-e-uniqueness eval-v-elims(2) is-satis.cases*)

qed
moreover have *s1* = *s2* **using** *s1 s2* ∗ *assms wfG-nilI wfX-wfY* **by** *metis*

ultimately show i ⟦ [[x]^v]^{ce} == *e2* ⟧ ~ *True*
using *eval-c.intros eval-e.intros eval-v.intros*
proof −
have i ⟦ *e2* ⟧ ~ *s1*
by (*metis* ‹*s1* = *s2*› *s2*)
then show *?thesis*
by (*metis* (*full-types*) ‹*i x* = *Some s1*› *eval-c-eqI eval-e-valI eval-v-varI*)
qed
qed
qed

lemma *valid-len*:
assumes $\vdash_{wf}$ Θ
shows Θ ; ℬ ; (x, *B-int*, [[x]^v]^{ce} == [[*L-num* (*int* (*length v*))]^v]^{ce}) $\#_\Gamma$ *GNil* ⊨ [[x]^v]^{ce} == *CE-len* [[*L-bitvec v*]^v]^{ce} (**is** Θ ; ℬ ; *?G* ⊨ *?c*)
proof −
have ∗:Θ $\vdash_{wf}$ ([]::Φ) ∧ Θ ; ℬ ; *GNil* $\vdash_{wf}$ $[]_\Delta$ **using** *assms wfG-nilI wfD-emptyI wfPhi-emptyI* **by** *auto*

moreover hence Θ ; ℬ ; *GNil* $\vdash_{wf}$ *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) : *B-int*
using *wfCE-valI* ∗ *wfV-litI base-for-lit.simps*
by (*metis wfE-valI wfX-wfY*)

moreover have Θ ; ℬ ; *GNil* $\vdash_{wf}$ *CE-len* [(*V-lit* (*L-bitvec v*))]^{ce} : *B-int*
using *wfE-valI* ∗ *wfV-litI base-for-lit.simps wfE-valI wfX-wfY wfCE-valI*
by (*metis wfCE-lenI*)
moreover have *atom x* ♯ *GNil* **by** *auto*
ultimately have Θ ; ℬ ; *?G* $\vdash_{wf}$ *?c* **using** *wfC-e-eq2 assms* **by** *simp*
moreover have (∀ *i*. *wfI* Θ *?G i* ∧ *is-satis-g i ?G* ⟶ *is-satis i ?c*) **using** *is-satis-len-imp* **by** *auto*
ultimately show *?thesis* **using** *valid.simps* **by** *auto*
qed

lemma *valid-arith-bop*:
assumes *wfG* Θ ℬ Γ **and** *opp* = *Plus* ∧ *ll* = (*L-num* (*n1*+*n2*)) ∨ (*opp* = *LEq* ∧ *ll* = (*if n1*≤*n2 then L-true else L-false*))
and (*opp* = *Plus* ⟶ *b* = *B-int*) ∧ (*opp* = *LEq* ⟶ *b* = *B-bool*) **and**
atom x ♯ Γ
shows Θ; ℬ ; (x, b, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*ll*)))) $\#_\Gamma$ Γ
⊨ (*CE-val* (*V-var x*) == *CE-op opp* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce})) (**is** Θ ; ℬ ; *?G* ⊨ *?c*)
proof −
have *wfC* Θ ℬ *?G ?c* **proof**(*rule wfC-e-eq2*)
show Θ ; ℬ ; Γ $\vdash_{wf}$ *CE-val* (*V-lit ll*) : *b* **using** *wfCE-valI wfV-litI assms base-for-lit.simps* **by** *metis*
show Θ ; ℬ ; Γ $\vdash_{wf}$ *CE-op opp* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce}) : *b*
using *wfCE-plusI wfCE-leqI wfCE-eqI wfV-litI wfCE-valI base-for-lit.simps assms* **by** *metis*
show $\vdash_{wf}$ Θ **using** *assms wfX-wfY* **by** *auto*
show *atom x* ♯ Γ **using** *assms* **by** *auto*
qed

moreover have ∀ *i*. *wfI* Θ *?G i* ∧ *is-satis-g i ?G* ⟶ *is-satis i ?c* **proof**(*rule allI* , *rule impI*)
fix *i*
assume *wfI* Θ *?G i* ∧ *is-satis-g i ?G*

hence *is-satis i* ((*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*ll*)))) **by** *auto*
thus *is-satis i* ((*CE-val* (*V-var x*) == *CE-op opp* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce})))
using *is-satis-plus-imp assms opp.exhaust is-satis-leq-imp* **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *metis*
qed

lemma *valid-eq-bop*:
assumes *wfG* Θ ℬ Γ **and** *atom x* ♯ Γ **and** *base-for-lit l1* = *base-for-lit l2*
shows Θ; ℬ ; (*x*, *B-bool*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*if l1* = *l2 then L-true else L-false*)))) $\#_\Gamma$ Γ
⊨ (*CE-val* (*V-var x*) == *CE-op Eq* ([*V-lit* (*l1*)]^{ce}) ([*V-lit* (*l2*)]^{ce})) (**is** Θ ; ℬ ; *?G* ⊨ *?c*)
proof −
let *?ll* = (*if l1* = *l2 then L-true else L-false*)
have *wfC* Θ ℬ *?G ?c* **proof**(*rule wfC-e-eq2*)
show Θ ; ℬ ; Γ $\vdash_{wf}$ *CE-val* (*V-lit ?ll*) : *B-bool* **using** *wfCE-valI wfV-litI assms base-for-lit.simps* **by** *metis*
show Θ ; ℬ ; Γ $\vdash_{wf}$ *CE-op Eq* ([*V-lit* (*l1*)]^{ce}) ([*V-lit* (*l2*)]^{ce}) : *B-bool*
using *wfCE-eqI wfCE-leqI wfCE-eqI wfV-litI wfCE-valI base-for-lit.simps assms* **by** *metis*
show $\vdash_{wf}$ Θ **using** *assms wfX-wfY* **by** *auto*
show *atom x* ♯ Γ **using** *assms* **by** *auto*
qed

moreover have ∀ *i*. *wfI* Θ *?G i* ∧ *is-satis-g i ?G* ⟶ *is-satis i ?c* **proof**(*rule allI* , *rule impI*)
fix *i*
assume *wfI* Θ *?G i* ∧ *is-satis-g i ?G*

hence *is-satis i* ((*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*?ll*)))) **by** *auto*
thus *is-satis i* ((*CE-val* (*V-var x*) == *CE-op Eq* ([*V-lit* (*l1*)]^{ce}) ([*V-lit* (*l2*)]^{ce})))
using *is-satis-eq-imp assms* **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *metis*
qed

lemma *valid-fst*:
fixes *x*::*x* **and** v_1::*v* **and** v_2::*v*
assumes *wfTh* Θ **and** *wfV* Θ ℬ *GNil* (*V-pair* v_1 v_2) (*B-pair* b_1 b_2)
shows Θ ; ℬ ; (*x*, b_1, $[[x]^v]^{ce}$ == $[v_1]^{ce}$) $\#_\Gamma$ *GNil* ⊨ $[[x]^v]^{ce}$ == $[\#1[[v_1,v_2]^v]^{ce}]^{ce}$
proof(*rule valid-eq-e*)
show ‹∀ *i s1 s2*. (Θ ; ℬ $\vdash_{wf}$ *GNil*) ∧ (Θ ; *GNil* ⊢ *i*) ∧ (*i* ⟦ [v_1]^{ce} ⟧ ~ *s1*) ∧ (*i* ⟦ [*#1*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*) ⟶ *s1* = *s2*›
proof(*rule*+)
fix *i s1 s2*
assume *as*:Θ ; ℬ $\vdash_{wf}$ *GNil* ∧ Θ ; *GNil* ⊢ *i* ∧ (*i* ⟦ [v_1]^{ce} ⟧ ~ *s1*) ∧ (*i* ⟦ [*#1*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*)
then obtain *s2′* **where** ∗:*i* ⟦ [v_1 , v_2]^v ⟧ ~ *SPair s2 s2′*
using *eval-e-elims*(*5*)[*of i* [[v_1 , v_2]^v]^{ce} *s2*] *eval-e-elims*

by *meson*
then have i ⟦ v_1 ⟧ ~ *s2* **using** *eval-v-elims(3)[OF ∗]* **by** *auto*
then show *s1* = *s2* **using** *eval-v-uniqueness as*
using *eval-e-uniqueness eval-e-valI* **by** *blast*
qed

show ‹ Θ ; ℬ ; *GNil* $\vdash_{wf}$ [v_1]^{ce} : b_1 › **using** *assms*
by (*metis b.eq-iff(4) wfV-elims(3) wfV-wfCE*)
show ‹ Θ ; ℬ ; *GNil* $\vdash_{wf}$ [*#1*[[v_1 , v_2]^v]^{ce}]^{ce} : b_1 › **using** *assms* **using** *wfCE-fstI*
using *wfCE-valI* **by** *blast*
qed

lemma *valid-snd*:
fixes *x*::*x* **and** v_1::*v* **and** v_2::*v*
assumes *wfTh* Θ **and** *wfV* Θ ℬ *GNil* (*V-pair* v_1 v_2) (*B-pair* b_1 b_2)
shows Θ ; ℬ ; (*x*, b_2, [[*x*]^v]^{ce} == [v_2]^{ce}) $\#_\Gamma$ *GNil* ⊨ [[*x*]^v]^{ce} == [*#2*[[v_1,v_2]^v]^{ce}]^{ce}
proof(*rule valid-eq-e*)
show ‹∀ *i s1 s2*. (Θ ; ℬ $\vdash_{wf}$ *GNil*) ∧ (Θ ; *GNil* ⊢ *i*) ∧ (*i* ⟦ [v_2]^{ce} ⟧ ~ *s1*) ∧
(*i* ⟦ [*#2*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*) ⟶ *s1* = *s2*›
proof(*rule+*)
fix *i s1 s2*
assume *as*:Θ ; ℬ $\vdash_{wf}$ *GNil* ∧ Θ ; *GNil* ⊢ *i* ∧ (*i* ⟦ [v_2]^{ce} ⟧ ~ *s1*) ∧ (*i* ⟦ [*#2*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧
~ *s2*)
then obtain *s2′* **where** ∗:*i* ⟦ [v_1 , v_2]^v ⟧ ~ *SPair s2′ s2*
using *eval-e-elims(5)[of i* [[v_1 , v_2]^v]^{ce} *s2] eval-e-elims*
by *meson*
then have i ⟦ v_2 ⟧ ~ *s2* **using** *eval-v-elims(3)[OF ∗]* **by** *auto*
then show *s1* = *s2* **using** *eval-v-uniqueness as*
using *eval-e-uniqueness eval-e-valI* **by** *blast*
qed

show ‹ Θ ; ℬ ; *GNil* $\vdash_{wf}$ [v_2]^{ce} : b_2 › **using** *assms*
by (*metis b.eq-iff wfV-elims wfV-wfCE*)
show ‹ Θ ; ℬ ; *GNil* $\vdash_{wf}$ [*#2*[[v_1 , v_2]^v]^{ce}]^{ce} : b_2 › **using** *assms* **using** *wfCE-sndI wfCE-valI* **by**
blast
qed

lemma *valid-concat*:
fixes *v1*::*bit list* **and** *v2*::*bit list*
assumes $\vdash_{wf}$ Π
shows Π ; ℬ ; (*x*, *B-bitvec*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-bitvec* (*v1*@ *v2*))))) $\#_\Gamma$ *GNil* ⊨
(*CE-val* (*V-var x*) == *CE-concat* ([*V-lit* (*L-bitvec v1*)]^{ce}) ([*V-lit* (*L-bitvec v2*)]^{ce}))
proof(*rule valid-eq-e*)
show ‹∀ *i s1 s2*. ((Π ; ℬ $\vdash_{wf}$ *GNil*) ∧ (Π ; *GNil* ⊢ *i*) ∧
(*i* ⟦ [[*L-bitvec* (*v1* @ *v2*)]^v]^{ce} ⟧ ~ *s1*) ∧ (*i* ⟦ [[[*L-bitvec v1*]^v]^{ce} @@ [[*L-bitvec v2*]^v]^{ce}]^{ce}
⟧ ~ *s2*) ⟶
s1 = *s2*)›
proof(*rule+*)
fix *i s1 s2*
assume *as*: (Π ; ℬ $\vdash_{wf}$ *GNil*) ∧ (Π ; *GNil* ⊢ *i*) ∧ (*i* ⟦ [[*L-bitvec* (*v1* @ *v2*)]^v]^{ce} ⟧ ~ *s1*) ∧
(*i* ⟦ [[[*L-bitvec v1*]^v]^{ce} @@ [[*L-bitvec v2*]^v]^{ce}]^{ce} ⟧ ~ *s2*)

hence ∗: i ⟦ [[[*L-bitvec v1* $]^v]^{ce}$ @@ [[*L-bitvec v2* $]^v]^{ce}]^{ce}$ ⟧ ~ *s2* **by** *auto*
obtain *bv1 bv2* **where** *s2*:*s2* = *SBitvec* (*bv1* @ *bv2*) ∧ *i* ⟦ [*L-bitvec v1* $]^v$ ⟧ ~ *SBitvec bv1* ∧ (*i* ⟦ [*L-bitvec v2* $]^v$ ⟧ ~ *SBitvec bv2*)
using *eval-e-elims*(*7*)[*OF* ∗] *eval-e-elims*(*1*) **by** *metis*
hence *v1* = *bv1* ∧ *v2* = *bv2* **using** *eval-v-elims*(*1*) *eval-l.simps*(*5*) **by** *force*
moreover then have *s1* = *SBitvec* (*bv1* @ *bv2*) **using** *s2* **using** *eval-v-elims*(*1*) *eval-l.simps*(*5*)
by (*metis as eval-e-elims*(*1*))

then show *s1* = *s2* **using** *s2* **by** *auto*
qed

show ‹ Π ; ℬ ; *GNil* $\vdash_{wf}$ [[*L-bitvec* (*v1* @ *v2*) $]^v$ $]^{ce}$: *B-bitvec* ›
by (*metis assms base-for-lit.simps*(*5*) *wfG-nilI wfV-litI wfV-wfCE*)
show ‹ Π ; ℬ ; *GNil* $\vdash_{wf}$ [[[*L-bitvec v1* $]^v]^{ce}$ @@ [[*L-bitvec v2* $]^v]^{ce}]^{ce}$: *B-bitvec* ›
by (*metis assms base-for-lit.simps*(*5*) *wfCE-concatI wfG-nilI wfV-litI wfCE-valI*)
qed

lemma *valid-ce-eq*:
fixes *ce*::*ce*
assumes Θ ; ℬ ; Γ $\vdash_{wf}$ *ce* : *b*
shows ‹Θ ; ℬ ; Γ ⊨ *ce* == *ce* ›
unfolding *valid.simps* **proof**
show ‹ Θ ; ℬ ; Γ $\vdash_{wf}$ *ce* == *ce* › **using** *assms wfC-eqI* **by** *auto*
show ‹∀ *i*. Θ ; Γ ⊢ *i* ∧ *i* ⊨ Γ ⟶ *i* ⊨ *ce* == *ce* › **proof**(*rule*+)
fix *i*
assume Θ ; Γ ⊢ *i* ∧ *i* ⊨ Γ
then obtain *s* **where** *i*⟦ *ce* ⟧ ~ *s* **using** *assms eval-e-exist* **by** *metis*
then show *i* ⟦ *ce* == *ce* ⟧ ~ *True* **using** *eval-c-eqI* **by** *metis*
qed
qed

lemma *valid-eq-imp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes Θ ; ℬ ; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1 IMP c2*
shows Θ ; ℬ ; (*x*, *b*, *c2*) $\#_\Gamma$ Γ ⊨ *c1* *IMP* *c2*
proof −
have ∀ *i*. (Θ ; (*x*, *b*, *c2*) $\#_\Gamma$ Γ ⊢ *i* ∧ *i* ⊨ (*x*, *b*, *c2*) $\#_\Gamma$ Γ) ⟶ *i* ⊨ (*c1* *IMP* *c2*)
proof(*rule*,*rule*)
fix *i*
assume *as*:Θ ; (*x*, *b*, *c2*) $\#_\Gamma$ Γ ⊢ *i* ∧ *i* ⊨ (*x*, *b*, *c2*) $\#_\Gamma$ Γ

have Θ ; ℬ ; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* **using** *wfC-elims assms* **by** *metis*

then obtain *sc* **where** *i* ⟦ *c1* ⟧ ~ *sc* **using** *eval-c-exist assms as* **by** *metis*
moreover have *i* ⟦ *c2* ⟧ ~ *True* **using** *as is-satis-g.simps is-satis.simps* **by** *auto*

ultimately have *i* ⟦ *c1* *IMP* *c2* ⟧ ~ *True* **using** *eval-c-impI* **by** *metis*

thus *i* ⊨ *c1* *IMP* *c2* **using** *is-satis.simps* **by** *auto*
qed
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *valid-range*:
assumes $0 \le n \wedge n \le m$ **and** $\vdash_{wf}\ \Theta$
shows Θ ; {||} ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_\Gamma$ *GNil* $\models$
(*C-eq* (*CE-op LEq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num m*)))) [[*L-true* $]^v$ $]^{ce}$) *AND*
(*C-eq* (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var x*))) [[*L-true* $]^v$ $]^{ce}$)
(**is** Θ ; {||} ; *?G* $\models$ *?c1 AND ?c2*)
proof(*rule validI*)
have *wfg*: Θ ; {||} $\vdash_{wf}$ (*x*, *B-int*, [[*x* $]^v$ $]^{ce}$ == [[*L-num n* $]^v$ $]^{ce}$) $\#_\Gamma$ *GNil*
using *assms base-for-lit.simps wfG-nilI wfV-litI fresh-GNil wfB-intI wfC-v-eq wfG-cons1I wfG-cons2I*
by *metis*

show Θ ; {||} ; *?G* $\vdash_{wf}$ *?c1 AND ?c2*
using *wfC-conjI wfC-eqI wfCE-leqI wfCE-valI wfV-varI wfg lookup.simps base-for-lit.simps wfV-litI wfB-intI wfB-boolI*
by *metis*

show $\forall$ *i*. Θ ; *?G* $\vdash$ *i* $\wedge$ *i* $\models$ *?G* $\longrightarrow$ *i* $\models$ *?c1 AND ?c2* **proof**(*rule*,*rule*)
fix *i*
assume *a*:Θ ; *?G* $\vdash$ *i* $\wedge$ *i* $\models$ *?G*
hence $*$:*i* ⟦ *V-var x* ⟧ ~ *SNum n*
proof −
obtain *sv* **where** *sv*: *i x* = *Some sv* $\wedge$ Θ $\vdash$ *sv* : *B-int* **using** *a wfI-def* **by** *force*
have *i* ⟦ (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*)))) ⟧ ~ *True*
using *a is-satis-g.simps*
using *is-satis.cases* **by** *blast*
hence *i x* = *Some*(*SNum n*) **using** *sv*
by (*metis eval-c-elims*(*7*) *eval-e-elims*(*1*) *eval-l.simps*(*3*) *eval-v-elims*(*1*) *eval-v-elims*(*2*))
thus *?thesis* **using** *eval-v-varI* **by** *auto*
qed

show *i* $\models$ *?c1 AND ?c2*
proof −
have *i* ⟦ *?c1* ⟧ ~ *True*
proof −
have *i* ⟦ [*leq* [[*x* $]^v$ $]^{ce}$ [[*L-num m* $]^v$ $]^{ce}$ $]^{ce}$⟧ ~ *SBool True*
using *eval-e-leqI assms eval-v-litI eval-l.simps* $*$
by (*metis* (*full-types*) *eval-e-valI*)
moreover have *i* ⟦ [[*L-true* $]^v$ $]^{ce}$ ⟧ ~ *SBool True*
using *eval-v-litI eval-e-valI eval-l.simps* **by** *metis*
ultimately show *?thesis* **using** *eval-c-eqI* **by** *metis*
qed

moreover have *i* ⟦ *?c2* ⟧ ~ *True*
proof −
have *i* ⟦ [*leq* [[*L-num 0* $]^v$ $]^{ce}$ [[*x* $]^v$ $]^{ce}$ $]^{ce}$ ⟧ ~ *SBool True*
using *eval-e-leqI assms eval-v-litI eval-l.simps* $*$
by (*metis* (*full-types*) *eval-e-valI*)
moreover have *i* ⟦ [[*L-true* $]^v$ $]^{ce}$ ⟧ ~ *SBool True*
using *eval-v-litI eval-e-valI eval-l.simps* **by** *metis*

ultimately show *?thesis* **using** *eval-c-eqI* **by** *metis*
qed
ultimately show *?thesis* **using** *eval-c-conjI is-satis.simps* **by** *metis*
qed
qed
qed

lemma *valid-range-length*:
fixes Γ::Γ
assumes $0 \leq n \wedge n \leq int\ (length\ v)$ **and** Θ ; {||} $\vdash_{wf}$ Γ **and** *atom x* ♯ Γ
shows Θ ; {||} ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_\Gamma$ Γ ⊨
(*C-eq* (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var x*))) [[*L-true* $]^v$ $]^{ce}$)
AND
(*C-eq* (*CE-op LEq* (*CE-val* (*V-var x*)) ([| [[*L-bitvec v* $]^v$ $]^{ce}$ $|]^{ce}$)) [[*L-true* $]^v$ $]^{ce}$)

(**is** Θ ; {||} ; *?G* ⊨ *?c1 AND ?c2*)
proof(*rule validI*)
have *wfg*: Θ ; {||} $\vdash_{wf}$ (*x*, *B-int*, [[*x* $]^v$ $]^{ce}$ == [[*L-num n* $]^v$ $]^{ce}$) $\#_\Gamma$ Γ **apply**(*rule wfG-cons1I*)
apply *simp*
using *assms* **apply** *simp+*
using *assms base-for-lit.simps wfG-nilI wfV-litI wfB-intI wfC-v-eq wfB-intI wfX-wfY assms* **by** *metis+*

show Θ ; {||} ; *?G* $\vdash_{wf}$ *?c1 AND ?c2*
using *wfC-conjI wfC-eqI wfCE-leqI wfCE-valI wfV-varI wfg lookup.simps base-for-lit.simps wfV-litI wfB-intI wfB-boolI*
by (*metis* (*full-types*) *wfCE-lenI*)

show ∀ *i*. Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G* ⟶ *i* ⊨ *?c1 AND ?c2* **proof**(*rule*,*rule*)
fix *i*
assume *a*:Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G*
hence ∗:*i* ⟦ *V-var x* ⟧ ~ *SNum n*
proof −
obtain *sv* **where** *sv*: *i x* = *Some sv* ∧ Θ ⊢ *sv* : *B-int* **using** *a wfI-def* **by** *force*
have *i* ⟦ (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*)))) ⟧ ~ *True*
using *a is-satis-g.simps*
using *is-satis.cases* **by** *blast*
hence *i x* = *Some*(*SNum n*) **using** *sv*
by (*metis eval-c-elims*(*7*) *eval-e-elims*(*1*) *eval-l.simps*(*3*) *eval-v-elims*(*1*) *eval-v-elims*(*2*))
thus *?thesis* **using** *eval-v-varI* **by** *auto*
qed

show *i* ⊨ *?c1 AND ?c2*
proof −
have *i* ⟦ *?c2* ⟧ ~ *True*
proof −
have *i* ⟦ [*leq* [[*x* $]^v$ $]^{ce}$ [| [[*L-bitvec v* $]^v$ $]^{ce}$ $|]^{ce}$ $]^{ce}$⟧ ~ *SBool True*
using *eval-e-leqI assms eval-v-litI eval-l.simps* ∗
by (*metis* (*full-types*) *eval-e-lenI eval-e-valI*)
moreover have *i* ⟦ [[*L-true* $]^v$ $]^{ce}$ ⟧ ~ *SBool True*
using *eval-v-litI eval-e-valI eval-l.simps* **by** *metis*
ultimately show *?thesis* **using** *eval-c-eqI* **by** *metis*

qed

moreover have *i* ⟦ *?c1* ⟧ ~ *True*
proof −
have *i* ⟦ [*leq* [[*L-num 0*]^v]^{ce} [[*x*]^v]^{ce}]^{ce} ⟧ ~ *SBool True*
using *eval-e-leqI assms eval-v-litI eval-l.simps* ∗
by (*metis* (*full-types*) *eval-e-valI*)
moreover have *i* ⟦ [[*L-true*]^v]^{ce} ⟧ ~ *SBool True*
using *eval-v-litI eval-e-valI eval-l.simps* **by** *metis*
ultimately show *?thesis* **using** *eval-c-eqI* **by** *metis*
qed
ultimately show *?thesis* **using** *eval-c-conjI is-satis.simps* **by** *metis*
qed
qed
qed

lemma *valid-range-length-inv-gnil*:
fixes Γ::Γ
assumes $\vdash_{wf}$ Θ
and Θ ; {||} ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_\Gamma$ *GNil* ⊨
(*C-eq* (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var x*))) [[*L-true*]^v]^{ce})
AND
(*C-eq* (*CE-op LEq* (*CE-val* (*V-var x*)) ([| [[*L-bitvec v*]^v]^{ce} |]^{ce})) [[*L-true*]^v]^{ce})

(**is** Θ ; {||} ; *?G* ⊨ *?c1 AND ?c2*)
shows *0* ≤ *n* ∧ *n* ≤ *int* (*length v*)
proof −
have ∗:∀ *i*. Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G* ⟶ *i* ⊨ *?c1 AND ?c2* **using** *assms valid.simps* **by** *simp*

obtain *i* **where** *i*: *i x* = *Some* (*SNum n*) **by** *auto*
have Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G* **proof**
show Θ ; *?G* ⊢ *i* **unfolding** *wfI-def* **using** *wfRCV-BIntI i* ∗ **by** *auto*
have *i* ⟦ ([[*x*]^v]^{ce} == [[*L-num n*]^v]^{ce}) ⟧ ~ *True*
using ∗ *eval-c.intros*(*7*) *eval-e.intros eval-v.intros eval-l.simps*
by (*metis* (*full-types*) *i*)
thus *i* ⊨ *?G* **unfolding** *is-satis-g.simps is-satis.simps* **by** *auto*
qed
hence ∗∗:*i* ⊨ *?c1 AND ?c2* **using** ∗ **by** *auto*

hence *1*: *i* ⟦ *?c1* ⟧ ~ *True* **using** *eval-c-elims*(*3*) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** (*sv1* = *sv2*) = *True* ∧ *i* ⟦ [*leq* [[*L-num 0*]^v]^{ce} [[*x*]^v]^{ce}]^{ce} ⟧
~ *sv1* ∧ *i* ⟦ [[*L-true*]^v]^{ce} ⟧ ~ *sv2*
using *eval-c-elims*(*7*) **by** *metis*
hence *sv1* = *SBool True* **using** *eval-e-elims eval-v-elims eval-l.simps i* **by** *metis*
obtain *n1* **and** *n2* **where** *SBool True* = *SBool* (*n1* ≤ *n2*) ∧ (*i* ⟦ [[*L-num 0*]^v]^{ce} ⟧ ~ *SNum n1*)
∧ (*i* ⟦ [[*x*]^v]^{ce} ⟧ ~ *SNum n2*)
using *eval-e-elims*(*3*)[*of i* [[*L-num 0*]^v]^{ce} [[*x*]^v]^{ce} *SBool True*]
using ‹(*sv1* = *sv2*) = *True* ∧ *i* ⟦ [*leq* [[*L-num 0*]^v]^{ce} [[*x*]^v]^{ce}]^{ce} ⟧ ~ *sv1* ∧ *i* ⟦ [[*L-true*]^v]^{ce}
⟧ ~ *sv2*› ‹*sv1* = *SBool True*› **by** *fastforce*
moreover hence *n1* = *0* **and** *n2* = *n* **using** *eval-e-elims eval-v-elims i*
apply (*metis eval-l.simps*(*3*) *rcl-val.eq-iff*(*2*))

using *eval-e-elims eval-v-elims i*
by (*metis calculation option.inject rcl-val.eq-iff*(*2*))
ultimately have *le1*: *0* ≤ *n* **by** *simp*

hence *2*: *i* ⟦ *?c2* ⟧ ~ *True* **using** ∗∗ *eval-c-elims*(*3*) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** *sv*: (*sv1* = *sv2*) = *True* ∧ *i* ⟦ [*leq* [[*x*]^{v}]^{ce} [| [[*L-bitvec v*]^{v}]^{ce} |]^{ce}]^{ce} ⟧ ~ *sv1* ∧ *i* ⟦ [[*L-true*]^{v}]^{ce} ⟧ ~ *sv2*
using *eval-c-elims*(*7*) **by** *metis*
hence *sv1* = *SBool True* **using** *eval-e-elims eval-v-elims eval-l.simps i* **by** *metis*
obtain *n1* **and** *n2* **where** ∗∗∗:*SBool True* = *SBool* (*n1* ≤ *n2*) ∧ (*i* ⟦ [[*x*]^{v}]^{ce} ⟧ ~ *SNum n1*) ∧ (*i* ⟦ [| [[*L-bitvec v*]^{v}]^{ce} |]^{ce} ⟧ ~ *SNum n2*)
using *eval-e-elims*(*3*)
using *sv* ‹*sv1* = *SBool True*› **by** *metis*
moreover hence *n1* = *n* **using** *eval-e-elims*(*1*)[*of i*] *eval-v-elims*(*2*)[*of i x SNum n1*] *i* **by** *auto*
moreover have *n2* = *int* (*length v*) **using** *eval-e-elims*(*8*) *eval-v-elims*(*1*) *eval-l.simps i*
by (*metis* ∗∗∗ *eval-e-elims*(*1*) *rcl-val.eq-iff*(*1*) *rcl-val.eq-iff*(*2*))
ultimately have *le2*: *n* ≤ *int* (*length v*) **by** *simp*

show *?thesis* **using** *le1 le2* **by** *auto*
qed

lemma *wfI-cons*:
fixes *i*::*valuation* **and** Γ::Γ
assumes *i′* ⊨ Γ **and** Θ ; Γ ⊢ *i′* **and** *i* = *i′* (*x* ↦ *s*) **and** Θ ⊢ *s* : *b* **and** *atom x* ♯ Γ
shows Θ ; (*x*,*b*,*c*) $\#_{\Gamma}$ Γ ⊢ *i*
unfolding *wfI-def* **proof** −
{
fix *x′ b′ c′*
assume (*x′*,*b′*,*c′*) ∈ *toSet* ((*x*, *b*, *c*) $\#_{\Gamma}$ Γ)
then consider (*x′*,*b′*,*c′*) = (*x*,*b*,*c*) | (*x′*,*b′*,*c′*) ∈ *toSet* Γ **using** *toSet.simps* **by** *auto*
then have ∃ *s*. *Some s* = *i x′* ∧ Θ ⊢ *s* : *b′* **proof**(*cases*)
case *1*
then show *?thesis* **using** *assms* **by** *auto*
next
case *2*
then obtain *s* **where** *s*:*Some s* = *i′ x′* ∧ Θ ⊢ *s* : *b′* **using** *assms wfI-def* **by** *auto*
moreover have *x′* ≠ *x* **using** *assms 2 fresh-dom-free* **by** *auto*
ultimately have *Some s* = *i x′* **using** *assms* **by** *auto*
then show *?thesis* **using** *s wfI-def* **by** *auto*
qed
}
thus ∀(*x*, *b*, *c*)∈*toSet* ((*x*, *b*, *c*) $\#_{\Gamma}$ Γ). ∃ *s*. *Some s* = *i x* ∧ Θ ⊢ *s* : *b* **by** *auto*
qed

lemma *valid-range-length-inv*:
fixes Γ::Γ
assumes Θ ; *B* $\vdash_{wf}$ Γ **and** *atom x* ♯ Γ **and** ∃ *i*. *i* ⊨ Γ ∧ Θ ; Γ ⊢ *i*
and Θ ; *B* ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_{\Gamma}$ Γ ⊨
(*C-eq* (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var x*))) [[*L-true*]^{v}]^{ce})
AND
(*C-eq* (*CE-op LEq* (*CE-val* (*V-var x*)) ([| [[*L-bitvec v*]^{v}]^{ce} |]^{ce})) [[*L-true*]^{v}]^{ce})

(**is** Θ ; *?B* ; *?G* ⊨ *?c1 AND ?c2*)
shows *0* ≤ *n* ∧ *n* ≤ *int* (*length v*)
proof −
have ∗:∀ *i*. Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G* ⟶ *i* ⊨ *?c1 AND ?c2* **using** *assms valid.simps* **by** *simp*

obtain *i′* **where** *idash*: *is-satis-g i′* Γ ∧ Θ ; Γ ⊢ *i′* **using** *assms* **by** *auto*
obtain *i* **where** *i*: *i* = *i′* (*x* ↦ *SNum n*) **by** *auto*
hence *ix*: *i x* = *Some* (*SNum n*) **by** *auto*
have Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G* **proof**
show Θ ; *?G* ⊢ *i* **using** *wfI-cons i idash ix wfRCV-BIntI assms* **by** *simp*

have ∗∗:*i* ⟦ ([[*x*]^{v}]^{ce} == [[*L-num n*]^{v}]^{ce}) ⟧ ~ *True*
using ∗ *eval-c.intros*(*7*) *eval-e.intros eval-v.intros eval-l.simps i*
by (*metis* (*full-types*) *ix*)

show *i* ⊨ *?G* **unfolding** *is-satis-g.simps* **proof**
show ‹ *i* ⊨ [[*x*]^{v}]^{ce} == [[*L-num n*]^{v}]^{ce} › **using** ∗∗ *is-satis.simps* **by** *auto*
show ‹ *i* ⊨ Γ › **using** *idash i assms is-satis-g-i-upd* **by** *metis*
qed
qed
hence ∗∗:*i* ⊨ *?c1 AND ?c2* **using** ∗ **by** *auto*

hence *1*: *i* ⟦ *?c1* ⟧ ~ *True* **using** *eval-c-elims*(*3*) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** (*sv1* = *sv2*) = *True* ∧ *i* ⟦ [*leq* [[*L-num 0*]^{v}]^{ce} [[*x*]^{v}]^{ce}]^{ce} ⟧ ~ *sv1* ∧ *i* ⟦ [[*L-true*]^{v}]^{ce} ⟧ ~ *sv2*
using *eval-c-elims*(*7*) **by** *metis*
hence *sv1* = *SBool True* **using** *eval-e-elims eval-v-elims eval-l.simps i* **by** *metis*
obtain *n1* **and** *n2* **where** *SBool True* = *SBool* (*n1* ≤ *n2*) ∧ (*i* ⟦ [[*L-num 0*]^{v}]^{ce} ⟧ ~ *SNum n1*) ∧ (*i* ⟦ [[*x*]^{v}]^{ce} ⟧ ~ *SNum n2*)
using *eval-e-elims*(*3*)[*of i* [[*L-num 0*]^{v}]^{ce} [[*x*]^{v}]^{ce} *SBool True*]
using ‹(*sv1* = *sv2*) = *True* ∧ *i* ⟦ [*leq* [[*L-num 0*]^{v}]^{ce} [[*x*]^{v}]^{ce}]^{ce} ⟧ ~ *sv1* ∧ *i* ⟦ [[*L-true*]^{v}]^{ce} ⟧ ~ *sv2*› ‹*sv1* = *SBool True*› **by** *fastforce*
moreover hence *n1* = *0* **and** *n2* = *n* **using** *eval-e-elims eval-v-elims i*
apply (*metis eval-l.simps*(*3*) *rcl-val.eq-iff*(*2*))
using *eval-e-elims eval-v-elims i*
calculation option.inject rcl-val.eq-iff(*2*)
by (*metis ix*)
ultimately have *le1*: *0* ≤ *n* **by** *simp*

hence *2*: *i* ⟦ *?c2* ⟧ ~ *True* **using** ∗∗ *eval-c-elims*(*3*) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** *sv*: (*sv1* = *sv2*) = *True* ∧ *i* ⟦ [*leq* [[*x*]^{v}]^{ce} [| [[*L-bitvec v*]^{v}]^{ce} |]^{ce}]^{ce} ⟧ ~ *sv1* ∧ *i* ⟦ [[*L-true*]^{v}]^{ce} ⟧ ~ *sv2*
using *eval-c-elims*(*7*) **by** *metis*
hence *sv1* = *SBool True* **using** *eval-e-elims eval-v-elims eval-l.simps i* **by** *metis*
obtain *n1* **and** *n2* **where** ∗∗∗:*SBool True* = *SBool* (*n1* ≤ *n2*) ∧ (*i* ⟦ [[*x*]^{v}]^{ce} ⟧ ~ *SNum n1*) ∧ (*i* ⟦ [| [[*L-bitvec v*]^{v}]^{ce} |]^{ce} ⟧ ~ *SNum n2*)
using *eval-e-elims*(*3*)
using *sv* ‹*sv1* = *SBool True*› **by** *metis*
moreover hence *n1* = *n* **using** *eval-e-elims*(*1*)[*of i*] *eval-v-elims*(*2*)[*of i x SNum n1*] *i* **by** *auto*

moreover have *n2* = *int* (*length v*) **using** *eval-e-elims*(*8*) *eval-v-elims*(*1*) *eval-l.simps i*
by (*metis* ∗∗∗ *eval-e-elims*(*1*) *rcl-val.eq-iff*(*1*) *rcl-val.eq-iff*(*2*))
ultimately have *le2*: *n* ≤ *int* (*length v*) **by** *simp*

show *?thesis* **using** *le1 le2* **by** *auto*
qed

lemma *eval-c-conj2I*[*intro*]:
assumes *i* ⟦ *c1* ⟧ ~ *True* **and** *i* ⟦ *c2* ⟧ ~ *True*
shows *i* ⟦ (*C-conj c1 c2*) ⟧ ~ *True*
using *assms eval-c-conjI* **by** *metis*

lemma *valid-split*:
assumes *split n v* (*v1*,*v2*) **and** $\vdash_{wf}$ Θ
shows Θ ; {||} ; (*z* , [*B-bitvec* , *B-bitvec* $]^b$, [[*z* $]^v$ $]^{ce}$ == [[[*L-bitvec v1* $]^v$, [*L-bitvec v2* $]^v$ $]^v$ $]^{ce}$) $\#_\Gamma$ *GNil*
⊨ ([[*L-bitvec v* $]^v$ $]^{ce}$ == [[*#1*[[*z* $]^v$ $]^{ce}]^{ce}$ @@ [*#2*[[*z* $]^v$ $]^{ce}]^{ce}$ $]^{ce}$) *AND* ([| [*#1*[[*z* $]^v$ $]^{ce}]^{ce}$ $|]^{ce}$ == [[*L-num n* $]^v$ $]^{ce}$)
(**is** Θ ; {||} ; *?G* ⊨ *?c1 AND ?c2*)
unfolding *valid.simps* **proof**

have *wfg*: Θ ; {||} $\vdash_{wf}$ (*z*, [*B-bitvec* , *B-bitvec* $]^b$, [[*z* $]^v$ $]^{ce}$ == [[[*L-bitvec v1* $]^v$, [*L-bitvec v2* $]^v$ $]^v$ $]^{ce}$) $\#_\Gamma$ *GNil*
using *wf-intros assms base-for-lit.simps fresh-GNil wfC-v-eq wfG-intros2* **by** *metis*

show Θ ; {||} ; *?G* $\vdash_{wf}$ *?c1 AND ?c2*
apply(*rule wfC-conjI*)
apply(*rule wfC-eqI*)
apply(*rule wfCE-valI*)
apply(*rule wfV-litI*)
using *wf-intros wfg lookup.simps base-for-lit.simps wfC-v-eq*
apply (*metis*)+
done

have *len*:*int* (*length v1*) = *n* **using** *assms split-length* **by** *auto*

show ∀ *i*. Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G* ⟶ *i* ⊨ (*?c1 AND ?c2*)
proof(*rule*,*rule*)
fix *i*
assume *a*:Θ ; *?G* ⊢ *i* ∧ *i* ⊨ *?G*
hence *i* ⟦ [[*z* $]^v$ $]^{ce}$ == [[[*L-bitvec v1* $]^v$, [*L-bitvec v2* $]^v$ $]^v$ $]^{ce}$ ⟧ ~ *True*
using *is-satis-g.simps is-satis.simps* **by** *simp*
then obtain *sv* **where** *i* ⟦ [[*z* $]^v$ $]^{ce}$ ⟧ ~ *sv* ∧ *i* ⟦ [[[*L-bitvec v1* $]^v$, [*L-bitvec v2* $]^v$ $]^v$ $]^{ce}$ ⟧ ~ *sv*
using *eval-c-elims* **by** *metis*
hence *i* ⟦ [[*z* $]^v$ $]^{ce}$ ⟧ ~ (*SPair* (*SBitvec v1*) (*SBitvec v2*)) **using** *eval-c-eqI eval-v.intros eval-l.simps*

by (*metis eval-e-elims*(*1*) *eval-v-uniqueness*)
hence *b*:*i z* = *Some* (*SPair* (*SBitvec v1*) (*SBitvec v2*)) **using** *a eval-e-elims eval-v-elims* **by** *metis*

have *v1*: *i* ⟦ [*#1*[[*z* $]^v$ $]^{ce}]^{ce}$ ⟧ ~ *SBitvec v1*
using *eval-e-fstI eval-e-valI eval-v-varI b* **by** *metis*
have *v2*: *i* ⟦ [*#2*[[*z* $]^v$ $]^{ce}]^{ce}$ ⟧ ~ *SBitvec v2*

using *eval-e-sndI eval-e-valI eval-v-varI b* **by** *metis*

have i ⟦ [[*L-bitvec v* $]^v$ $]^{ce}$ ⟧ ~ *SBitvec v* **using** *eval-e.intros eval-v.intros eval-l.simps* **by** *metis*
moreover have i ⟦ [[#*1*[[*z* $]^v$ $]^{ce}]^{ce}$ @@ [#*2*[[*z* $]^v$ $]^{ce}]^{ce}$ $]^{ce}$ ⟧ ~ *SBitvec v*
using *assms split-concat v1 v2 eval-e-concatI* **by** *metis*
moreover have i ⟦ [| [#*1*[[*z* $]^v$ $]^{ce}]^{ce}$ |$]^{ce}$ ⟧ ~ *SNum* (*int* (*length v1*))
using *v1 eval-e-lenI* **by** *auto*
moreover have i ⟦ [[*L-num n* $]^v$ $]^{ce}$ ⟧ ~ *SNum n* **using** *eval-e.intros eval-v.intros eval-l.simps* **by** *metis*
ultimately show i ⊨ *?c1 AND ?c2* **using** *is-satis.intros eval-c-conj2I eval-c-eqI len* **by** *metis*
qed
qed

lemma *is-satis-eq*:
assumes *wfI* Θ *G i* **and** *wfCE* Θ ℬ *G e b*
shows *is-satis i* (*e* == *e*)
proof(*rule*)
obtain *s* **where** *eval-e i e s* **using** *eval-e-exist assms* **by** *metis*
thus *eval-c i* (*e* == *e*) *True* **using** *eval-c-eqI* **by** *metis*
qed

lemma *is-satis-g-i-upd2*:
assumes *eval-v i v s* **and** *is-satis* ((*i* (*x* ↦ *s*))) *c0* **and** *atom x* ♯ *G* **and** *wfG* Θ ℬ (*G3*@((*x*,*b*,*c0*)$\#_\Gamma$*G*)) **and** *wfV* Θ ℬ *G v b* **and** *wfI* Θ (*G3*$[x::=v]_{\Gamma v}$@*G*) *i*
and *is-satis-g i* (*G3*$[x::=v]_{\Gamma v}$@*G*)
shows *is-satis-g* (*i* (*x* ↦ *s*)) (*G3*@((*x*,*b*,*c0*)$\#_\Gamma$*G*))
using *assms* **proof**(*induct G3 rule*: Γ*-induct*)
case *GNil*
hence *is-satis-g* (*i*(*x* ↦ *s*)) *G* **using** *is-satis-g-i-upd* **by** *auto*
then show *?case* **using** *GNil* **using** *is-satis-g.simps append-g.simps* **by** *metis*
next
case (*GCons x′ b′ c′* Γ′)
hence *x*≠*x′* **using** *wfG-cons-append* **by** *metis*
hence *is-satis-g i* (((*x′*, *b′*, *c′*$[x::=v]_{cv}$) $\#_\Gamma$ (Γ′$[x::=v]_{\Gamma v}$) @ *G*)) **using** *subst-gv.simps GCons* **by** *auto*
hence ∗:*is-satis i c′*$[x::=v]_{cv}$ ∧ *is-satis-g i* ((Γ′$[x::=v]_{\Gamma v}$) @ *G*) **using** *subst-gv.simps* **by** *auto*

have *is-satis-g* (*i*(*x* ↦ *s*)) ((*x′*, *b′*, *c′*) $\#_\Gamma$ (Γ′@ (*x*, *b*, *c0*) $\#_\Gamma$ *G*)) **proof**(*subst is-satis-g.simps*,*rule*)
show *is-satis* (*i*(*x* ↦ *s*)) *c′* **proof**(*subst subst-c-satis-full*[*symmetric*])
show ‹*eval-v i v s*› **using** *GCons* **by** *auto*
show ‹ Θ ; ℬ ; ((*x′*, *b′*, *c′*) $\#_\Gamma$Γ′)@(*x*, *b*, *c0*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c′* › **using** *GCons wfC-refl* **by** *auto*
show ‹*wfI* Θ ((((*x′*, *b′*, *c′*) $\#_\Gamma$ Γ′)$[x::=v]_{\Gamma v}$) @ *G*) *i*› **using** *GCons* **by** *auto*
show ‹ Θ ; ℬ ; *G* $\vdash_{wf}$ *v* : *b* › **using** *GCons* **by** *auto*
show ‹*is-satis i c′*$[x::=v]_{cv}$› **using** ∗ **by** *auto*
qed
show *is-satis-g* (*i*(*x* ↦ *s*)) (Γ′ @ (*x*, *b*, *c0*) $\#_\Gamma$ *G*) **proof**(*rule GCons*(*1*))
show ‹*eval-v i v s*› **using** *GCons* **by** *auto*
show ‹*is-satis* (*i*(*x* ↦ *s*)) *c0*› **using** *GCons* **by** *metis*
show ‹*atom x* ♯ *G*› **using** *GCons* **by** *auto*
show ‹ Θ ; ℬ$\vdash_{wf}$ Γ′ @ (*x*, *b*, *c0*) $\#_\Gamma$ *G* › **using** *GCons wfG-elims append-g.simps* **by** *metis*
show ‹*is-satis-g i* (Γ′$[x::=v]_{\Gamma v}$ @ *G*)› **using** ∗ **by** *auto*
show *wfI* Θ (Γ′$[x::=v]_{\Gamma v}$ @ *G*) *i* **using** *GCons wfI-def subst-g-assoc-cons* ‹*x*≠*x′*› **by** *auto*

show Θ ; $\mathcal{B}$; $G \vdash_{wf}$ v : b **using** *GCons* **by** *auto*
qed
qed
moreover have $((x', b', c') \#_\Gamma \Gamma' @ (x, b, c0) \#_\Gamma G) = (((x', b', c') \#_\Gamma \Gamma') @ (x, b, c0) \#_\Gamma G)$ **by** *auto*
ultimately show *?case* **using** *GCons* **by** *metis*
qed

end

Chapter 12

Typing Lemmas

12.1 Prelude

Needed as the typing elimination rules give us facts for an alpha-equivalent version of a term and so need to be able to 'jump back' to a typing judgement for the orginal term

lemma *τ-fresh-c*[*simp*]:
  assumes *atom x* $\sharp$ $\{\!|\ z : b \mid c\ |\!\}$ **and** *atom z* $\sharp$ *x*
  shows *atom x* $\sharp$ *c*
  using *τ.fresh assms fresh-at-base*
  by (*simp add*: *fresh-at-base*(*2*))

lemmas *subst-defs = subst-b-b-def subst-b-c-def subst-b-τ-def subst-v-v-def subst-v-c-def subst-v-τ-def*

lemma *wfT-wfT-if1*:
  assumes *wfT* Θ $\mathcal{B}$ Γ ($\{\!|\ z : $ *b-of t* $\mid$ *CE-val v* == *CE-val* (*V-lit L-false*) *IMP* *c-of t z* $|\!\}$) **and** *atom z* $\sharp$ (Γ,*t*)
  shows *wfT* Θ $\mathcal{B}$ Γ *t*
  using *assms* **proof**(*nominal-induct t avoiding*: Γ *z rule*: *τ.strong-induct*)
  case (*T-refined-type* z' b' c')
  show *?case* **proof**(*rule wfT-wfT-if*)
    show ‹ $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!|\ z : b' \mid [\ v\ ]^{ce} == [\ [\ L\text{-}false\ ]^{v}\ ]^{ce}\ \ IMP\ \ c'[z'::=[\ z]^{v}]_{cv}\ |\!\}$ ›
      using *T-refined-type b-of.simps c-of.simps subst-defs* **by** *metis*
    show ‹*atom z* $\sharp$ (c', Γ)› **using** *T-refined-type fresh-prodN τ-fresh-c* **by** *metis*
  qed
qed

lemma *fresh-u-replace-true*:
  fixes *bv*::*bv* **and** Γ::Γ
  assumes *atom bv* $\sharp$ Γ' @ (x, b, c) $\#_\Gamma$ Γ
  shows *atom bv* $\sharp$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ
  using *fresh-append-g fresh-GCons assms fresh-Pair c.fresh*(*1*) **by** *auto*

lemma *wf-replace-true1*:
  fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** c''::*c* **and** c'::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** b'::*b* **and** *b*::*b* **and** *s*::*s*
    and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows $\Theta; \mathcal{B}; G \vdash_{wf} v : b' \Longrightarrow G = \Gamma'\ @\ (x,\ b,\ c)\ \#_\Gamma\ \Gamma \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ @\ ((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma)$

$\vdash_{wf}\ v : b'$ **and**
$\Theta; \mathcal{B};\ G\ \vdash_{wf}\ c'' \Longrightarrow\ G =\ \Gamma' @(x,\ b,\ c)\ \#_\Gamma\ \Gamma \Longrightarrow\ \Theta\ ;\ \mathcal{B}\ ; \Gamma'\ @\ ((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma) \vdash_{wf}\ c''$ **and**
$\Theta\ ; \mathcal{B} \vdash_{wf}\ G \Longrightarrow\ G =\ \Gamma' @(x,\ b,\ c)\ \#_\Gamma\ \Gamma \Longrightarrow\ \Theta\ ; \mathcal{B} \vdash_{wf}\ \Gamma'\ @\ ((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma)$ **and**
$\Theta; \mathcal{B};\ G \vdash_{wf}\ \tau \Longrightarrow\ G =\ \Gamma' @(x,\ b,\ c)\ \#_\Gamma\ \Gamma \Longrightarrow\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ @\ ((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma) \vdash_{wf}\ \tau$ **and**
$\Theta; \mathcal{B}; \Gamma\ \vdash_{wf}\ ts \Longrightarrow\ True$ **and**
$\vdash_{wf}\ P \Longrightarrow\ True$ **and**
$\Theta\ ; \mathcal{B} \vdash_{wf}\ b \Longrightarrow\ True$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ G\ \vdash_{wf}\ ce : b' \Longrightarrow\ G =\ \Gamma' @(x,\ b,\ c)\ \#_\Gamma\ \Gamma \Longrightarrow\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ @\ ((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma) \vdash_{wf}\ ce : b'$ **and**
$\Theta\ \vdash_{wf}\ td \Longrightarrow\ True$
proof(*nominal-induct*
b' **and** c'' **and** G **and** τ **and** ts **and** P **and** b **and** b' **and** td
arbitrary: $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$ **and** $\Gamma\ \Gamma'$
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfB-intI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-boolI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-unitI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-bitvecI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-pairI* Θ $\mathcal{B}$ *b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-consI* Θ *s dclist* $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-appI* Θ *b s bv dclist* $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfV-varI* Θ $\mathcal{B}$ Γ'' b' c x')
hence *wfg*: ‹ $\Theta\ ; \mathcal{B}\ \vdash_{wf}\ \Gamma'\ @\ (x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma$ › **by** *auto*
show *?case* **proof**(*cases* $x=x'$)
case *True*
hence *Some* $(b,\ TRUE)$ = *lookup* $(\Gamma'\ @\ (x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma)\ x'$ **using** *lookup.simps lookup-inside-wf wfg* **by** *simp*
thus *?thesis* **using** *Wellformed.wfV-varI*[*OF wfg*]
by (*metis True lookup-inside-wf old.prod.inject option.inject wfV-varI.hyps*(*1*) *wfV-varI.hyps*(*3*) *wfV-varI.prems*)
next
case *False*
hence *Some* $(b',\ c)$ = *lookup* $(\Gamma'\ @\ (x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma)\ x'$ **using** *lookup-inside2 wfV-varI* **by** *metis*
then show *?thesis* **using** *Wellformed.wfV-varI*[*OF wfg*]
by (*metis wfG-elim2 wfG-suffix wfV-varI.hyps*(*1*) *wfV-varI.hyps*(*2*) *wfV-varI.hyps*(*3*)

wfV-varI.prems Wellformed.wfV-varI wf-replace-inside(*1*))
qed
next
case (*wfV-litI* Θ $\mathcal{B}$ Γ *l*)
then show *?case* **using** *wf-intros* **using** *wf-intros* **by** *metis*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfV-consI s dclist* Θ *dc x b′ c* $\mathcal{B}$ Γ *v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfV-conspI s bv dclist* Θ *dc xc bc cc* $\mathcal{B}$ *b′* Γ′′ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *metis*
show ‹(*dc*, ⦃ *xc* : *bc* | *cc* ⦄) ∈ *set dclist*› **using** *wfV-conspI* **by** *metis*
show ‹ Θ ;$\mathcal{B}$ $\vdash_{wf}$ *b′* › **using** *wfV-conspI* **by** *metis*
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *v* : *bc*[*bv*::=*b′*]$_{bb}$ › **using** *wfV-conspI* **by** *metis*
have *atom bv* ♯ Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ **using** *fresh-u-replace-true wfV-conspI* **by** *metis*
thus ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ, *b′*, *v*)› **using** *wfV-conspI fresh-prodN* **by** *metis*
qed
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-eqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ′′ *b′ c′*)
show *?case* **proof**
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ)› **using** *wfTI fresh-append-g fresh-GCons fresh-prodN*
by *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b′* › **using** *wfTI* **by** *metis*
show ‹ Θ; $\mathcal{B}$; (*z*, *b′*, *TRUE*) $\#_\Gamma$ Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c′* › **using** *wfTI append-g.simps*

by *metis*
  qed
next
  case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfC-trueI* Θ $\mathcal{B}$ Γ)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfC-falseI* Θ $\mathcal{B}$ Γ)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfC-conjI* Θ $\mathcal{B}$ Γ *c1 c2*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfC-disjI* Θ $\mathcal{B}$ Γ *c1 c2*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfC-notI* Θ $\mathcal{B}$ Γ *c1*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfC-impI* Θ $\mathcal{B}$ Γ *c1 c2*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfG-nilI* Θ $\mathcal{B}$)
  then show *?case* **using** *GNil-append* **by** *blast*
next
  case (*wfG-cons1I c* Θ $\mathcal{B}$ Γ'' *x b*)
  then show *?case* **using** *wf-intros wfG-cons-TRUE2 wfG-elims*(*2*) *wfG-replace-inside wfG-suffix*
    by (*metis* (*no-types*, *lifting*))
next
  case (*wfG-cons2I c* Θ $\mathcal{B}$ Γ'' *x' b*)
  then show *?case* **using** *wf-intros*
    by (*metis wfG-cons-TRUE2 wfG-elims*(*2*) *wfG-replace-inside wfG-suffix*)
next
  case *wfTh-emptyI*
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfTh-consI tdef* Θ)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfTD-simpleI* Θ *lst s*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfTD-poly* Θ *bv lst s*)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfTs-nil* Θ $\mathcal{B}$ Γ)
  then show *?case* **using** *wf-intros* **by** *metis*
next
  case (*wfTs-cons* Θ $\mathcal{B}$ Γ τ *dc ts*)
  then show *?case* **using** *wf-intros* **by** *metis*

qed

lemma *wf-replace-true2*:
fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** c''::c **and** c'::c **and** τ::τ **and** ts::($string*\tau$) *list* **and** Δ::Δ **and** b'::b **and** b::b **and** s::s
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows Θ ; Φ ; $\mathcal{B}$; G ; D $\vdash_{wf}$ e : $b' \Longrightarrow G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ); $D \vdash_{wf}$ e : b' **and**
Θ ; Φ ; $\mathcal{B}$; G ; $\Delta \vdash_{wf}$ s : $b' \Longrightarrow G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) ; $\Delta \vdash_{wf}$ s : b' **and**
Θ ; Φ ; $\mathcal{B}$; G ; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ *cs* : $b' \Longrightarrow$ $G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) ; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ *cs* : b' **and**
Θ ; Φ ; $\mathcal{B}$; G ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : $b' \Longrightarrow$ $G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : b' **and**

$\Theta \vdash_{wf} \Phi \Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$; G $\vdash_{wf}$ $\Delta \Longrightarrow$ $G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ Δ **and**

Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** b' **and** b' **and** b' **and** Φ **and** Δ **and** *ftq* **and** *ft*
arbitrary: Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ v b)
then show *?case* **using** *wf-intros* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-eqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)

then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-appI* Θ Φ *ℬ* Γ Δ *f x b c τ s v*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-appPI* Θ Φ *ℬ* Γ′′ Δ *b′ bv v τ f x1 b1 c1 s*)
show *?case* **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfE-appPI wf-replace-true1* **by** *metis*
show ‹ Θ; *ℬ*; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfE-appPI* **by** *metis*
show ‹ Θ ; *ℬ* $\vdash_{wf}$ *b′* › **using** *wfE-appPI* **by** *metis*
have *atom bv* ♯ Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ **using** *fresh-u-replace-true wfE-appPI fresh-prodN* **by** *metis*
thus ‹*atom bv* ♯ (Φ, Θ, *ℬ*, Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ, Δ, *b′*, *v*, (*b-of τ*)[*bv*::=*b′*]$_b$)›
using *wfE-appPI fresh-prodN* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x1 b1 c1 τ s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *metis*
show ‹ Θ; *ℬ*; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *v* : *b1*[*bv*::=*b′*]$_b$ › **using** *wfE-appPI wf-replace-true1* **by** *metis*
qed
next
case (*wfE-mvarI* Θ Φ *ℬ* Γ Δ *u τ*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next

case (*wfS-valI* Θ Φ *ℬ* Γ *v b* Δ)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-letI* Θ Φ *ℬ* Γ′′ Δ *e b′ x1 s b1*)
show *?case* **proof**
show ‹ Θ ; Φ ; *ℬ* ; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *e* : *b′* › **using** *wfS-letI wf-replace-true1* **by** *metis*
have ‹ Θ ; Φ ; *ℬ* ; ((*x1*, *b′*, *TRUE*) $\#_\Gamma$ Γ′) @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s* : *b1* › **apply**(*rule wfS-letI*(*4*))
using *wfS-letI append-g.simps* **by** *simp*
thus ‹ Θ ; Φ ; *ℬ* ; (*x1*, *b′*, *TRUE*) $\#_\Gamma$ Γ′@ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s* : *b1* › **using** *append-g.simps* **by** *auto*
show ‹ Θ; *ℬ*; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfS-letI* **by** *metis*
show *atom x1* ♯ (Φ, Θ, *ℬ*, Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ, Δ, *e*, *b1*) **using** *fresh-append-g fresh-GCons fresh-prodN wfS-letI* **by** *auto*
qed
next
case (*wfS-assertI* Θ Φ *ℬ x′ c* Γ′′ Δ *s b′*)
show *?case* **proof**
show ‹ Θ ; Φ ; *ℬ* ; (*x′*, *B-bool*, *c*) $\#_\Gamma$ Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s* : *b′* ›
using *wfS-assertI* (*2*)[*of* (*x′*, *B-bool*, *c*) $\#_\Gamma$ Γ′ Γ] *wfS-assertI* **by** *simp*
show ‹ Θ; *ℬ*; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c* › **using** *wfS-assertI wf-replace-true1* **by** *metis*
show ‹ Θ; *ℬ*; Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfS-assertI* **by** *metis*
show ‹*atom x′* ♯ (Φ, Θ, *ℬ*, Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ, Δ, *c*, *b′*, *s*)› **using** *wfS-assertI fresh-prodN* **by** *simp*
qed
next
case (*wfS-let2I* Θ Φ *ℬ* Γ′′ Δ *s1 τ x′ s2 ba′*)
show *?case* **proof**

show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s1* : *b-of* τ › **using** *wfS-let2I wf-replace-true1* **by** *metis*
show ‹ Θ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ τ › **using** *wfS-let2I wf-replace-true1* **by** *metis*
have ‹ Θ ; Φ ; $\mathcal{B}$; ((x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ') @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s2* : *ba'* ›
apply(*rule wfS-let2I*(*5*))
using *wfS-let2I append-g.simps* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s2* : *ba'* › **using** *wfS-let2I append-g.simps* **by** *auto*
show ‹*atom* x' ♯ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, *s1*, *ba'*, τ)› **using** *fresh-append-g fresh-GCons fresh-prodN wfS-let2I* **by** *auto*
qed
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ v Φ Δ *s1* b *s2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ'' τ v u Φ Δ b' s)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ τ › **using** *wfS-varI wf-replace-true1* **by** *metis*
show ‹ Θ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ v : *b-of* τ › **using** *wfS-varI wf-replace-true1* **by** *metis*
show ‹*atom* u ♯ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, τ, v, b')› **using** *wfS-varI u-fresh-g fresh-prodN* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; (u, τ) $\#_\Delta$ Δ $\vdash_{wf}$ s : b' › **using** *wfS-varI* **by** *metis*
qed

next
case (*wfS-assignI* u τ Δ Θ $\mathcal{B}$ Γ Φ v)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2* b)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2* b)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ'' v *tid dclist* Δ Φ *cs* b')
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ v : *B-id tid* › **using** *wfS-matchI wf-replace-true1* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ› **using** *wfS-matchI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfS-matchI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfS-matchI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *cs* : b' › **using** *wfS-matchI* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ x' τ Γ'' Δ s b' *tid dc*)
show *?case* **proof**
have ‹ Θ ; Φ ; $\mathcal{B}$; ((x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ') @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ s : b' › **using** *wfS-branchI append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ s : b' › **using** *wfS-branchI append-g.simps append-g.simps* **by** *metis*
show ‹*atom* x' ♯ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, τ)› **using**

wfS-branchI **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dc t cs b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dc t cs b dclist css*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ u)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfPhi-consI* f Θ Φ *ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTNone* Θ Φ *ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTSome* Θ Φ *bv ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTI* Θ B b Φ x c s τ)
then show *?case* **using** *wf-intros* **by** *metis*
qed

lemmas *wf-replace-true* = *wf-replace-true1 wf-replace-true2*

12.2 Subtyping

lemma *subtype-reflI2*:
fixes τ::τ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ τ
shows Θ; $\mathcal{B}$; $\Gamma \vdash \tau \lesssim \tau$
proof −
obtain z b c **where** $*$:τ = $\{\!|$ z : b | c $|\!\}$ $\wedge$ *atom* z $\sharp$ (Θ,$\mathcal{B}$,Γ) $\wedge$ Θ; $\mathcal{B}$; (z, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ c
using *wfT-elims*(*1*)[*OF assms*] **by** *metis*
obtain x::x **where** $**$: *atom* x $\sharp$ (Θ, $\mathcal{B}$, Γ, c, z ,c, z , c) **using** *obtain-fresh* **by** *metis*
have Θ; $\mathcal{B}$; $\Gamma \vdash$ $\{\!|$ z : b | c $|\!\}$ $\lesssim$ $\{\!|$ z : b | c $|\!\}$ **proof**
show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ z : b | c $|\!\}$ **using** $*$ *assms* **by** *auto*
show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ z : b | c $|\!\}$ **using** $*$ *assms* **by** *auto*
show *atom* x $\sharp$ (Θ, $\mathcal{B}$, Γ, z , c , z , c) **using** *fresh-prod6 fresh-prod5* $**$ **by** *metis*
thus Θ; $\mathcal{B}$; (x, b, $c[z{::=}V\text{-}var\ x]_v$) $\#_\Gamma$ Γ $\models$ $c[z{::=}V\text{-}var\ x]_v$ **using** *wfT-wfC-cons assms* $*$ $**$ *subst-v-c-def* **by** *simp*
qed
thus *?thesis* **using** $*$ **by** *auto*

qed

lemma *subtype-reflI*:
  assumes ⦃ *z1* : *b* | *c1* ⦄ = ⦃ *z2* : *b* | *c2* ⦄ **and** *wf1*: Θ; ℬ;Γ $\vdash_{wf}$ (⦃ *z1* : *b* | *c1* ⦄)
  shows Θ; ℬ; Γ ⊢ (⦃ *z1* : *b* | *c1* ⦄) ≲ (⦃ *z2* : *b* | *c2* ⦄)
  using *assms subtype-reflI2* **by** *metis*

nominal-function *base-eq* :: Γ ⇒ τ ⇒ τ ⇒ *bool* **where**
  base-eq - ⦃ *z1* : *b1*| *c1* ⦄ ⦃ *z2* : *b2* | *c2* ⦄ = (*b1* = *b2*)
  apply(*auto*,*simp add*: *eqvt-def base-eq-graph-aux-def*)
  by (*meson* τ.*exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *subtype-wfT*:
  fixes *t1*::τ **and** *t2*::τ
  assumes Θ; ℬ; Γ ⊢ *t1* ≲ *t2*
  shows Θ; ℬ; Γ $\vdash_{wf}$ *t1* ∧ Θ; ℬ; Γ $\vdash_{wf}$ *t2*
  using *assms subtype-elims* **by** *metis*

lemma *subtype-eq-base*:
  assumes Θ; ℬ; Γ ⊢ (⦃ *z1* : *b1*| *c1* ⦄) ≲ (⦃ *z2* : *b2* | *c2* ⦄)
  shows *b1*=*b2*
  using *subtype.simps assms* **by** *auto*

lemma *subtype-eq-base2*:
  assumes Θ; ℬ; Γ ⊢ *t1* ≲ *t2*
  shows *b-of t1* = *b-of t2*
  using *assms* **proof**(*rule subtype.induct*[*of* Θ ℬ Γ *t1 t2*],*goal-cases*)
  case (*1* Θ Γ *z1 b c1 z2 c2 x*)
  then show *?case* **using** *subtype-eq-base* **by** *auto*
qed

lemma *subtype-wf*:
  fixes τ*1*::τ **and** τ*2*::τ
  assumes Θ; ℬ; Γ ⊢ τ*1* ≲ τ*2*
  shows Θ; ℬ; Γ $\vdash_{wf}$ τ*1* ∧Θ; ℬ; Γ $\vdash_{wf}$ τ*2*
  using *assms*
proof(*rule subtype.induct*[*of* Θ ℬ Γ τ*1* τ*2*],*goal-cases*)
  case (*1* Θ Γ*G z1 b c1 z2 c2 x*)
  then show *?case* **by** *blast*
qed

lemma *subtype-g-wf*:
  fixes τ*1*::τ **and** τ*2*::τ **and** Γ::Γ
  assumes Θ; ℬ; Γ ⊢ τ*1* ≲ τ*2*
  shows Θ ; ℬ$\vdash_{wf}$ Γ
  using *assms*
proof(*rule subtype.induct*[*of* Θ ℬ Γ τ*1* τ*2*],*goal-cases*)
  case (*1* Θ ℬ Γ *z1 b c1 z2 c2 x*)
  then show *?case* **using** *wfX-wfY* **by** *auto*
qed

For when we have a particular y that satisfies the freshness conditions that we want the validity check to use

lemma *valid-flip-simple*:
  assumes Θ; $\mathcal{B}$; $(z, b, c) \#_\Gamma \Gamma \models c'$ **and** *atom z* $\sharp$ Γ **and** *atom x* $\sharp$ (z, c, z, c', Γ)
  shows Θ; $\mathcal{B}$; $(x, b, (z \leftrightarrow x) \cdot c) \#_\Gamma \Gamma \models (z \leftrightarrow x) \cdot c'$
proof –
  have $(z \leftrightarrow x) \cdot \Theta$; $\mathcal{B}$; $(z \leftrightarrow x) \cdot ((z, b, c) \#_\Gamma \Gamma) \models (z \leftrightarrow x) \cdot c'$
    using *True-eqvt valid.eqvt assms beta-flip-eq wfX-wfY* **by** *metis*
  moreover have $\vdash_{wf} \Theta$ **using** *valid.simps wfC-wf wfG-wf assms* **by** *metis*
  ultimately show *?thesis*
    using *theta-flip-eq G-cons-flip-fresh3*[*of x* Γ *z b c*] *assms fresh-Pair flip-commute* **by** *metis*
qed

lemma *valid-wf-all*:
  assumes Θ; $\mathcal{B}$; $(z0,b,c0)\#_\Gamma G \models c$
  shows *wfG* Θ $\mathcal{B}$ *G* **and** *wfC* Θ $\mathcal{B}$ $((z0,b,c0)\#_\Gamma G)$ *c* **and** *atom z0* $\sharp$ *G*
  using *valid.simps wfC-wf wfG-cons assms* **by** *metis+*

lemma *valid-wfT*:
  fixes *z::x*
  assumes Θ; $\mathcal{B}$; $(z0,b,c0[z::=V\text{-}var\ z0]_v)\#_\Gamma G \models c[z::=V\text{-}var\ z0]_v$ **and** *atom z0* $\sharp$ $(\Theta, \mathcal{B}, G,c,c0)$
  shows Θ; $\mathcal{B}$; $G \vdash_{wf} \{\!\{ z : b \mid c0 \}\!\}$ **and** Θ; $\mathcal{B}$; $G \vdash_{wf} \{\!\{ z : b \mid c \}\!\}$
proof –
  have *atom z0* $\sharp$ *c0* **using** *assms fresh-Pair* **by** *auto*
  moreover have $*$: Θ ; $\mathcal{B}$ $\vdash_{wf}$ $(z0,b,c0[z::=V\text{-}var\ z0]_{cv})\#_\Gamma G$ **using** *valid-wf-all wfX-wfY assms subst-v-c-def* **by** *metis*
  ultimately show *wft*: Θ; $\mathcal{B}$; $G \vdash_{wf} \{\!\{ z : b \mid c0 \}\!\}$ **using** *wfG-wfT*[*OF* $*$] **by** *auto*

  have *atom z0* $\sharp$ *c* **using** *assms fresh-Pair* **by** *auto*
  moreover have *wfc*: Θ; $\mathcal{B}$; $(z0,b,c0[z::=V\text{-}var\ z0]_v)\#_\Gamma G \vdash_{wf} c[z::=V\text{-}var\ z0]_v$ **using** *valid-wf-all assms* **by** *metis*
  have Θ; $\mathcal{B}$; $G \vdash_{wf} \{\!\{ z0 : b \mid c[z::=V\text{-}var\ z0]_v \}\!\}$ **proof**
    show ‹*atom z0* $\sharp$ $(\Theta, \mathcal{B}, G)$› **using** *assms fresh-prodN* **by** *simp*
    show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wft wfT-wfB* **by** *force*
    show ‹ Θ; $\mathcal{B}$; $(z0, b, TRUE) \#_\Gamma G \vdash_{wf} c[z::=[z0]^v]_v$ › **using** *wfc wfC-replace-inside*[*OF wfc*, *of GNil z0 b c0*$[z::=[z0]^v]_v$ *G C-true*] *wfC-trueI*
        append-g.simps
      by (*metis local.$*$ wfG-elim2 wf-trans*(*2*))
  qed
  moreover have $\{\!\{ z0 : b \mid c[z::=V\text{-}var\ z0]_v \}\!\} = \{\!\{ z : b \mid c \}\!\}$ **using** ‹*atom z0* $\sharp$ *c0*› *τ.eq-iff Abs1-eq-iff*(*3*)
    using *calculation*(*1*) *subst-v-c-def* **by** *auto*
  ultimately show Θ; $\mathcal{B}$; $G \vdash_{wf} \{\!\{ z : b \mid c \}\!\}$ **by** *auto*
qed

lemma *valid-flip*:
  fixes *c::c* **and** *z::x* **and** *z0::x* **and** *xx2::x*
  assumes Θ; $\mathcal{B}$; $(xx2, b, c0[z0::=V\text{-}var\ xx2]_v) \#_\Gamma \Gamma \models c[z::=V\text{-}var\ xx2]_v$ **and**
    atom xx2 $\sharp$ $(c0,\Gamma,c,z)$ **and** *atom z0* $\sharp$ (Γ,c,z)
  shows Θ; $\mathcal{B}$; $(z0, b, c0) \#_\Gamma \Gamma \models c[z::=V\text{-}var\ z0]_v$
proof –

  have $\vdash_{wf} \Theta$ **using** *assms valid-wf-all wfX-wfY* **by** *metis*

hence Θ ; $\mathcal{B}$; $(xx2 \leftrightarrow z0\ ) \cdot ((xx2,\ b,\ c0[z0::=V\text{-}var\ xx2]_v)\ \#_\Gamma\ \Gamma)\ \models ((xx2 \leftrightarrow z0\ ) \cdot c[z::=V\text{-}var\ xx2]_v)$
  using *valid.eqvt True-eqvt assms beta-flip-eq theta-flip-eq* **by** *metis*
hence Θ; $\mathcal{B}$; $(((xx2 \leftrightarrow z0\ ) \cdot xx2,\ b,\ (xx2 \leftrightarrow z0\ ) \cdot c0[z0::=V\text{-}var\ xx2]_v)\ \#_\Gamma\ (xx2 \leftrightarrow z0\ ) \cdot \Gamma)\ \models ((xx2 \leftrightarrow z0\ ) \cdot (c[z::=V\text{-}var\ xx2]_v))$
  using *G-cons-flip*[*of xx2 z0 xx2 b* $c0[z0::=V\text{-}var\ xx2]_v$ Γ] **by** *auto*
moreover have $(xx2 \leftrightarrow z0\ ) \cdot xx2 = z0$ **by** *simp*
moreover have $(xx2 \leftrightarrow z0\ ) \cdot c0[z0::=V\text{-}var\ xx2]_v = c0$
  using *assms subst-cv-v-flip*[*of xx2 c0 z0 V-var z0*] *assms fresh-prod4* **by** *auto*
moreover have $(xx2 \leftrightarrow z0\ ) \cdot \Gamma = \Gamma$ **proof** −
  have *atom xx2* $\sharp$ Γ **using** *assms* **by** *auto*
  moreover have *atom z0* $\sharp$ Γ **using** *assms* **by** *auto*
  ultimately show *?thesis* **using** *flip-fresh-fresh* **by** *auto*
qed
moreover have $(xx2 \leftrightarrow z0\ ) \cdot (c[z::=V\text{-}var\ xx2]_v) = c[z::=V\text{-}var\ z0]_v$
  using *subst-cv-v-flip3 assms* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *subtype-valid*:
  assumes Θ; $\mathcal{B}$; $\Gamma \vdash t1 \lesssim t2$ **and** *atom y* $\sharp$ Γ **and** $t1 = \{\!\{\ z1 : b\ \mid c1\ \}\!\}$ **and** $t2 = \{\!\{\ z2 : b\ \mid c2\ \}\!\}$
  shows Θ; $\mathcal{B}$; $((y,\ b,\ c1[z1::=V\text{-}var\ y]_v)\ \#_\Gamma\ \Gamma) \models c2[z2::=V\text{-}var\ y]_v$
  using *assms* **proof**(*nominal-induct t2 avoiding*: *y rule*: *subtype.strong-induct*)
  case (*subtype-baseI x* Θ $\mathcal{B}$ Γ *z c z' c' ba*)

hence $(x \leftrightarrow y) \cdot \Theta$; $(x \leftrightarrow y) \cdot \mathcal{B}$; $(x \leftrightarrow y) \cdot ((x,\ ba,\ c[z::=[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma)\ \models (x \leftrightarrow y) \cdot c'[z'::=[\ x\ ]^v]_v$ **using** *valid.eqvt*
  using *permute-boolI* **by** *blast*
moreover have $\vdash_{wf} \Theta$ **using** *valid.simps wfC-wf wfG-wf subtype-baseI* **by** *metis*
ultimately have Θ; $\mathcal{B}$; $((y,\ ba,\ (x \leftrightarrow y) \cdot c[z::=[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma)\ \models (x \leftrightarrow y) \cdot c'[z'::=[\ x\ ]^v]_v$
  using *subtype-baseI theta-flip-eq beta-flip-eq* τ*.eq-iff G-cons-flip-fresh3*[*of y* Γ *x ba*] **by** (*metis flip-commute*)
moreover have $(x \leftrightarrow y) \cdot c[z::=[\ x\ ]^v]_v = c1[z1::=[\ y\ ]^v]_v$
  by (*metis subtype-baseI permute-flip-cancel subst-cv-id subst-cv-v-flip3 subst-cv-var-flip type-eq-subst-eq wfT-fresh-c subst-v-c-def*)
moreover have $(x \leftrightarrow y) \cdot c'[z'::=[\ x\ ]^v]_v = c2[z2::=[\ y\ ]^v]_v$
  by (*metis subtype-baseI permute-flip-cancel subst-cv-id subst-cv-v-flip3 subst-cv-var-flip type-eq-subst-eq wfT-fresh-c subst-v-c-def*)

ultimately show *?case* **using** *subtype-baseI* τ*.eq-iff* **by** *metis*
qed

lemma *subtype-valid-simple*:
  assumes Θ; $\mathcal{B}$; $\Gamma \vdash t1 \lesssim t2$ **and** *atom z* $\sharp$ Γ **and** $t1 = \{\!\{\ z : b\ \mid c1\ \}\!\}$ **and** $t2 = \{\!\{\ z : b\ \mid c2\ \}\!\}$
  shows Θ; $\mathcal{B}$; $((z,\ b,\ c1)\ \#_\Gamma\ \Gamma) \models c2$
  using *subst-v-c-def subst-v-id assms subtype-valid*[*OF assms*] **by** *simp*

lemma *obtain-for-t-with-fresh*:
  assumes *atom x* $\sharp$ *t*
  shows $\exists\, b\ c.\ t = \{\!\{\ x : b \mid c\ \}\!\}$
proof −
  obtain *z1 b1 c1* **where** ∗: $t = \{\!\{\ z1 : b1 \mid c1\ \}\!\} \wedge$ *atom z1* $\sharp$ *t* **using** *obtain-fresh-z* **by** *metis*

then have $t = (x \leftrightarrow z1) \cdot t$ **using** *flip-fresh-fresh assms* **by** *metis*
also have ... $= \{\!\mid (x \leftrightarrow z1) \cdot z1 : (x \leftrightarrow z1) \cdot b1 \mid (x \leftrightarrow z1) \cdot c1 \mid\!\}$ **using** ∗ *assms* **by** *simp*
also have ... $= \{\!\mid x : b1 \mid (x \leftrightarrow z1) \cdot c1 \mid\!\}$ **using** ∗ *assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *subtype-trans*:
assumes Θ; 𝓑; Γ ⊢ τ1 ≲ τ2 **and** Θ; 𝓑; Γ ⊢ τ2 ≲ τ3
shows Θ; 𝓑; Γ ⊢ τ1 ≲ τ3
using *assms* **proof**(*nominal-induct avoiding*: τ3 *rule*: *subtype.strong-induct*)
case (*subtype-baseI x* Θ 𝓑 Γ *z c z′ c′ b*)
hence *b-of* τ3 = *b* **using** *subtype-eq-base2 b-of.simps* **by** *metis*
then obtain *z′′ c′′* **where** *t3*: τ3 = ⦃ *z′′* : *b* | *c′′*⦄ ∧ *atom z′′* ♯ *x*
using *obtain-fresh-z2* **by** *metis*
hence *xf*: *atom x* ♯ (*z′′*, *c′′*) **using** *fresh-prodN subtype-baseI* τ*.fresh* **by** *auto*
have Θ; 𝓑; Γ ⊢ ⦃ *z* : *b* | *c* ⦄ ≲ ⦃ *z′′* : *b* | *c′′*⦄
proof(*rule Typing.subtype-baseI*)
show ‹*atom x* ♯ (Θ, 𝓑, Γ, *z*, *c*, *z′′*, *c′′*)› **using** *t3 fresh-prodN subtype-baseI xf* **by** *simp*
show ‹ Θ; 𝓑; Γ $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄ › **using** *subtype-baseI* **by** *auto*
show ‹ Θ; 𝓑; Γ $\vdash_{wf}$ ⦃ *z′′* : *b* | *c′′* ⦄ › **using** *subtype-baseI t3 subtype-elims* **by** *metis*
have Θ; 𝓑; (*x*, *b*, $c'[z'::=[\ x\ ]^v]_v$) $\#_\Gamma$ Γ ⊨ $c''[z''::=[\ x\ ]^v]_v$
using *subtype-valid*[*OF* ‹Θ; 𝓑; Γ ⊢ ⦃ *z′* : *b* | *c′* ⦄ ≲ τ3› , *of x z′ b c′ z′′ c′′*] *subtype-baseI*
t3 **by** *simp*
thus ‹Θ; 𝓑; (*x*, *b*, $c[z::=[\ x\ ]^v]_v$) $\#_\Gamma$ Γ ⊨ $c''[z''::=[\ x\ ]^v]_v$ ›
using *valid-trans-full*[*of* Θ 𝓑 *x b c z* Γ *c′ z′ c′′ z′′*] *subtype-baseI t3* **by** *simp*
qed
thus *?case* **using** *t3* **by** *simp*
qed

lemma *subtype-eq-e*:
assumes ∀ *i s1 s2 G*. *wfG P 𝓑 G* ∧ *wfI P G i* ∧ *eval-e i e1 s1* ∧ *eval-e i e2 s2* ⟶ *s1* = *s2* **and** *atom z1* ♯ *e1* **and** *atom z2* ♯ *e2* **and** *atom z1* ♯ Γ **and** *atom z2* ♯ Γ
and *wfCE P* 𝓑 Γ *e1 b* **and** *wfCE P* 𝓑 Γ *e2 b*
shows *P*; 𝓑; Γ ⊢ ⦃ *z1* : *b* | *CE-val* (*V-var z1*) == *e1* ⦄ ≲ (⦃ *z2* : *b* | *CE-val* (*V-var z2*) == *e2* ⦄)
proof −

have *wfCE P* 𝓑 Γ *e1 b* **and** *wfCE P* 𝓑 Γ *e2 b* **using** *assms* **by** *auto*

have *wst1*: *wfT P* 𝓑 Γ (⦃ *z1* : *b* | *CE-val* (*V-var z1*) == *e1* ⦄)
using *wfC-e-eq wfTI assms wfX-wfB wfG-fresh-x*
by (*simp add*: *wfT-e-eq*)

moreover have *wst2*:*wfT P* 𝓑 Γ (⦃ *z2* : *b* | *CE-val* (*V-var z2*) == *e2* ⦄)
using *wfC-e-eq wfX-wfB wfTI assms wfG-fresh-x*
by (*simp add*: *wfT-e-eq*)

moreover obtain *x*::*x* **where** *xf*: *atom x* ♯ (*P*, 𝓑 , *z1*, *CE-val* (*V-var z1*) == *e1* , *z2*, *CE-val* (*V-var z2*) == *e2* , Γ) **using** *obtain-fresh* **by** *blast*
moreover have *vld*: *P*; 𝓑 ; (*x*, *b*, (*CE-val* (*V-var z1*) == *e1*)$[z1::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ ⊨ (*CE-val* (*V-var z2*) == *e2*)$[z2::=V\text{-}var\ x]_v$ (**is** *P*; 𝓑 ; *?G* ⊨ *?c*)
proof −

have *wbg*: P; $\mathcal{B} \vdash_{wf}$ *?G* $\wedge$ P ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ *toSet* $\Gamma \subseteq$ *toSet ?G* **proof** −
have P; $\mathcal{B} \vdash_{wf}$ *?G* **proof**(*rule wfG-consI*)
show P; $\mathcal{B} \vdash_{wf}$ Γ **using** *assms wfX-wfY* **by** *metis*
show *atom x* ♯ Γ **using** *xf* **by** *auto*
show P; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *assms*(*6*) *wfX-wfB* **by** *auto*
show P; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ (*CE-val* (*V-var z1*) == *e1*)[*z1*::=*V-var x*]$_v$
using *wfC-e-eq*[*OF assms*(*6*)] *wf-subst*(*2*)
by (*simp add*: ‹*atom x* ♯ Γ› *assms*(*2*) *subst-v-c-def*)
qed
moreover hence P; $\mathcal{B} \vdash_{wf}$ Γ **using** *wfG-elims* **by** *metis*
ultimately show *?thesis* **using** *toSet.simps* **by** *auto*
qed

have *wsc*: *wfC P* $\mathcal{B}$ *?G ?c* **proof** −
have *wfCE P* $\mathcal{B}$ *?G* (*CE-val* (*V-var x*)) *b* **proof**
show ‹ P; $\mathcal{B}$; (*x*, *b*, (*CE-val* (*V-var z1*) == *e1*)[*z1*::=*V-var x*]$_v$) $\#_\Gamma$ Γ $\vdash_{wf}$ *V-var x* : *b* ›
using *wfV-varI lookup.simps wbg* **by** *auto*
qed
moreover have *wfCE P* $\mathcal{B}$ *?G e2 b* **using** *wf-weakening assms wbg* **by** *metis*
ultimately have *wfC P* $\mathcal{B}$ *?G* (*CE-val* (*V-var x*) == *e2*) **using** *wfC-eqI* **by** *simp*
thus *?thesis* **using** *subst-cv.simps*(*6*) ‹*atom z2* ♯ *e2*› *subst-v-c-def* **by** *simp*
qed

moreover have ∀ *i*. *wfI P ?G i* $\wedge$ *is-satis-g i ?G* ⟶ *is-satis i ?c* **proof**(*rule allI* , *rule impI*)
fix *i*
assume *as*: *wfI P ?G i* $\wedge$ *is-satis-g i ?G*
hence *is-satis i* ((*CE-val* (*V-var z1*) == *e1*)[*z1*::=*V-var x*]$_v$)
by (*simp add*: *is-satis-g.simps*(*2*))
hence *is-satis i* (*CE-val* (*V-var x*) == *e1*) **using** *subst-cv.simps assms subst-v-c-def* **by** *auto*
then obtain *s1* **and** *s2* **where** ∗:*eval-e i* (*CE-val* (*V-var x*)) *s1* $\wedge$ *eval-e i e1 s2* $\wedge$ *s1*=*s2* **using** *is-satis.simps eval-c-elims* **by** *metis*
moreover hence *eval-e i e2 s1* **proof** −
have ∗∗:*wfI P ?G i* **using** *as* **by** *auto*
moreover have *wfCE P* $\mathcal{B}$ *?G e1 b* $\wedge$ *wfCE P* $\mathcal{B}$ *?G e2 b* **using** *assms xf wf-weakening wbg* **by** *metis*
moreover then obtain *s2′* **where** *eval-e i e2 s2′* **using** *assms wfI-wfCE-eval-e* ∗∗ **by** *metis*
ultimately show *?thesis* **using** ∗ *assms*(*1*) *wfX-wfY* **by** *metis*
qed
ultimately have *is-satis i* (*CE-val* (*V-var x*) == *e2*) **using** *is-satis.simps eval-c-eqI* **by** *force*
thus *is-satis i* ((*CE-val* (*V-var z2*) == *e2*)[*z2*::=*V-var x*]$_v$) **using** *is-satis.simps eval-c-eqI assms subst-cv.simps subst-v-c-def* **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed
moreover have *atom x* ♯ (P, $\mathcal{B}$, Γ, *z1* , *CE-val* (*V-var z1*) == *e1*, *z2*, *CE-val* (*V-var z2*) == *e2*)
unfolding *fresh-prodN* **using** *xf fresh-prod7* τ*.fresh* **by** *fast*
ultimately show *?thesis* **using** *subtype-baseI*[*OF* - *wst1 wst2 vld*] *xf* **by** *simp*
qed

lemma *subtype-eq-e-nil*:

assumes ∀ *i s1 s2 G*. *wfG P* $\mathcal{B}$ *G* ∧ *wfI P G i* ∧ *eval-e i e1 s1* ∧ *eval-e i e2 s2* ⟶ *s1* = *s2* **and** *supp e1* = {} **and** *supp e2* = {} **and** *wfTh P*
and *wfCE P* $\mathcal{B}$ *GNil e1 b* **and** *wfCE P* $\mathcal{B}$ *GNil e2 b* **and** *atom z1* ♯ *GNil* **and** *atom z2* ♯ *GNil*
shows *P*; $\mathcal{B}$; *GNil* ⊢ ⦃ *z1* : *b* | *CE-val* (*V-var z1*) == *e1* ⦄ ≲ (⦃ *z2* : *b* | *CE-val* (*V-var z2*) == *e2* ⦄)
apply(*rule subtype-eq-e*,*auto simp add*: *assms e.fresh*)
using *assms fresh-def e.fresh supp-GNil* **by** *metis*+

lemma *subtype-gnil-fst-aux*:
assumes *supp* v_1 = {} **and** *supp* (*V-pair* v_1 v_2) = {} **and** *wfTh P* **and** *wfCE P* $\mathcal{B}$ *GNil* (*CE-val* v_1) *b* **and** *wfCE P* $\mathcal{B}$ *GNil* (*CE-fst* [*V-pair* v_1 v_2]^{ce}) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-val* v_2) *b2* **and** *atom z1* ♯ *GNil* **and** *atom z2* ♯ *GNil*
shows *P*; $\mathcal{B}$; *GNil* ⊢ (⦃ *z1* : *b* | *CE-val* (*V-var z1*) == *CE-val* v_1 ⦄) ≲ (⦃ *z2* : *b* | *CE-val* (*V-var z2*) == *CE-fst* [*V-pair* v_1 v_2]^{ce} ⦄)
proof −
have ∀ *i s1 s2 G* . *wfG P* $\mathcal{B}$ *G* ∧ *wfI P G i* ∧ *eval-e i* (*CE-val* v_1) *s1* ∧ *eval-e i* (*CE-fst* [*V-pair* v_1 v_2]^{ce}) *s2* ⟶ *s1* = *s2* **proof**(*rule*+)
fix *i s1 s2 G*
assume *as*: *wfG P* $\mathcal{B}$ *G* ∧ *wfI P G i* ∧ *eval-e i* (*CE-val* v_1) *s1* ∧ *eval-e i* (*CE-fst* [*V-pair* v_1 v_2]^{ce}) *s2*
hence *wfCE P* $\mathcal{B}$ *G* (*CE-val* v_2) *b2* **using** *assms wf-weakening*
by (*metis empty-subsetI toSet.simps*(*1*))
then obtain *s3* **where** *eval-e i* (*CE-val* v_2) *s3* **using** *wfI-wfCE-eval-e as* **by** *metis*
hence *eval-v i* ((*V-pair* v_1 v_2)) (*SPair s1 s3*)
by (*meson as eval-e-elims*(*1*) *eval-v-pairI*)
hence *eval-e i* (*CE-fst* [*V-pair* v_1 v_2]^{ce}) *s1* **using** *eval-e-fstI eval-e-valI* **by** *metis*
show *s1* = *s2* **using** *as eval-e-uniqueness*
using ‹*eval-e i* (*CE-fst* [*V-pair* v_1 v_2]^{ce}) *s1*› **by** *auto*
qed
thus *?thesis* **using** *subtype-eq-e-nil ce.supp assms* **by** *auto*
qed

lemma *subtype-gnil-snd-aux*:
assumes *supp* v_2 = {} **and** *supp* (*V-pair* v_1 v_2) = {} **and** *wfTh P* **and** *wfCE P* $\mathcal{B}$ *GNil* (*CE-val* v_2) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-snd* [(*V-pair* v_1 v_2)]^{ce}) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-val* v_1) *b2* **and** *atom z1* ♯ *GNil* **and** *atom z2* ♯ *GNil*
shows *P*; $\mathcal{B}$; *GNil* ⊢ (⦃ *z1* : *b* | *CE-val* (*V-var z1*) == *CE-val* v_2 ⦄) ≲ (⦃ *z2* : *b* | *CE-val* (*V-var z2*) == *CE-snd* [(*V-pair* v_1 v_2)]^{ce} ⦄)
proof −
have ∀ *i s1 s2 G*. *wfG P* $\mathcal{B}$ *G* ∧ *wfI P G i* ∧ *eval-e i* (*CE-val* v_2) *s1* ∧ *eval-e i* (*CE-snd* [(*V-pair* v_1 v_2)]^{ce}) *s2* ⟶ *s1* = *s2* **proof**(*rule*+)
fix *i s1 s2 G*
assume *as*: *wfG P* $\mathcal{B}$ *G* ∧ *wfI P G i* ∧ *eval-e i* (*CE-val* v_2) *s1* ∧ *eval-e i* (*CE-snd* [(*V-pair* v_1 v_2)]^{ce}) *s2*
hence *wfCE P* $\mathcal{B}$ *G* (*CE-val* v_1) *b2* **using** *assms wf-weakening*
by (*metis empty-subsetI toSet.simps*(*1*))
then obtain *s3* **where** *eval-e i* (*CE-val* v_1) *s3* **using** *wfI-wfCE-eval-e as* **by** *metis*
hence *eval-v i* ((*V-pair* v_1 v_2)) (*SPair s3 s1*)
by (*meson as eval-e-elims eval-v-pairI*)
hence *eval-e i* (*CE-snd* [(*V-pair* v_1 v_2)]^{ce}) *s1* **using** *eval-e-sndI eval-e-valI* **by** *metis*
show *s1* = *s2* **using** *as eval-e-uniqueness*

using ‹*eval-e i* (*CE-snd* [*V-pair* v_1 v_2]^{ce}) *s1*› **by** *auto*
qed
thus *?thesis* **using** *assms subtype-eq-e-nil* **by** (*simp add*: *ce.supp ce.supp*)
qed

lemma *subtype-gnil-fst*:
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ [*#1*[[v_1,v_2]^v]^{ce}]^{ce} : *b*
shows Θ ; {||} ; *GNil* ⊢ (⦃ z_1 : *b* | [[z_1]^v]^{ce} == [v_1]^{ce} ⦄) ≲ (⦃ z_2 : *b* | [[z_2]^v]^{ce} == [*#1*[[v_1, v_2]^v]^{ce}]^{ce} ⦄)
proof −
obtain *b2* **where** ∗∗: Θ ; {||} ; *GNil* $\vdash_{wf}$ *V-pair* v_1 v_2 : *B-pair b b2* **using** *wfCE-elims*(*4*)[*OF assms*] *wfCE-elims* **by** *metis*
obtain *b1′ b2′* **where** ∗:*B-pair b b2* = *B-pair b1′ b2′* ∧ Θ ; {||} ; *GNil* $\vdash_{wf}$ v_1 : *b1′* ∧ Θ ; {||} ; *GNil* $\vdash_{wf}$ v_2 : *b2′*
using *wfV-elims*(*3*)[*OF* ∗∗] **by** *metis*

show *?thesis* **proof**(*rule subtype-gnil-fst-aux*)
show ‹*supp* v_1 = {}› **using** ∗ *wfV-supp-nil* **by** *auto*
show ‹*supp* (*V-pair* v_1 v_2) = {}› **using** ∗∗ *wfV-supp-nil e.supp* **by** *metis*
show ‹$\vdash_{wf}$ Θ› **using** *assms wfX-wfY* **by** *metis*
show ‹Θ; {||}; *GNil* $\vdash_{wf}$ *CE-val* v_1 : *b* › **using** *wfCE-valI* ∗ **by** *auto*
show ‹Θ; {||}; *GNil* $\vdash_{wf}$ *CE-fst* [*V-pair* v_1 v_2]^{ce} : *b* › **using** *assms* **by** *auto*
show ‹Θ; {||}; *GNil* $\vdash_{wf}$ *CE-val* v_2 : *b2* ›**using** *wfCE-valI* ∗ **by** *auto*
show ‹*atom* z_1 ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
show ‹*atom* z_2 ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
qed
qed

lemma *subtype-gnil-snd*:
assumes *wfCE P* {||} *GNil* (*CE-snd* ([*V-pair* v_1 v_2]^{ce})) *b*
shows *P* ; {||} ; *GNil* ⊢ (⦃ *z1* : *b* | *CE-val* (*V-var z1*) == *CE-val* v_2 ⦄) ≲ (⦃ *z2* : *b* | *CE-val* (*V-var z2*) == *CE-snd* [(*V-pair* v_1 v_2)]^{ce} ⦄)
proof −
obtain *b1* **where** ∗∗: *P* ; {||} ; *GNil* $\vdash_{wf}$ *V-pair* v_1 v_2 : *B-pair b1 b* **using** *wfCE-elims assms* **by** *metis*
obtain *b1′ b2′* **where** ∗:*B-pair b1 b* = *B-pair b1′ b2′* ∧ *P* ; {||} ; *GNil* $\vdash_{wf}$ v_1 : *b1′* ∧ *P* ; {||} ; *GNil* $\vdash_{wf}$ v_2 : *b2′* **using** *wfV-elims*(*3*)[*OF* ∗∗] **by** *metis*

show *?thesis* **proof**(*rule subtype-gnil-snd-aux*)
show ‹*supp* v_2 = {}› **using** ∗ *wfV-supp-nil* **by** *auto*
show ‹*supp* (*V-pair* v_1 v_2) = {}› **using** ∗∗ *wfV-supp-nil e.supp* **by** *metis*
show ‹$\vdash_{wf}$ *P*› **using** *assms wfX-wfY* **by** *metis*
show ‹*P*; {||}; *GNil* $\vdash_{wf}$ *CE-val* v_1 : *b1* › **using** *wfCE-valI* ∗ **by** *simp*
show ‹*P*; {||}; *GNil* $\vdash_{wf}$ *CE-snd* [(*V-pair* v_1 v_2)]^{ce} : *b* › **using** *assms* **by** *auto*
show ‹*P*; {||}; *GNil* $\vdash_{wf}$ *CE-val* v_2 : *b* ›**using** *wfCE-valI* ∗ **by** *simp*
show ‹*atom z1* ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
show ‹*atom z2* ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
qed
qed

lemma *subtype-fresh-tau*:
fixes *x*::*x*

assumes *atom x* ♯ *t1* **and** *atom x* ♯ Γ **and** *P*; *B*; Γ ⊢ *t1* ≲ *t2*
shows *atom x* ♯ *t2*
proof −
have *wfg*: *P*; *B* $\vdash_{wf}$ Γ **using** *subtype-wf wfX-wfY assms* **by** *metis*
have *wft*: *wfT P B* Γ *t2* **using** *subtype-wf wfX-wfY assms* **by** *blast*
hence *supp t2* ⊆ *atom-dom* Γ ∪ *supp B* **using** *wf-supp*
using *atom-dom.simps* **by** *auto*
moreover have *atom x* ∉ *atom-dom* Γ **using** ‹*atom x* ♯ Γ› *wfG-atoms-supp-eq wfg fresh-def* **by** *blast*

ultimately show *atom x* ♯ *t2* **using** *fresh-def*
by (*metis Un-iff contra-subsetD x-not-in-b-set*)
qed

lemma *subtype-if-simp*:
assumes *wfT P B GNil* (⦃ *z1* : *b* | *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ ⦄) **and**
wfT P B GNil (⦃ *z* : *b* | *c* ⦄) **and** *atom z1* ♯ *c*
shows *P*; *B* ; *GNil* ⊢ (⦃ *z1* : *b* | *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ ⦄) ≲ (⦃ *z* : *b* | *c* ⦄)
proof −
obtain *x*::*x* **where** *xx*: *atom x* ♯ (*P* , *B* , *z1*, *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z*, *c*, *GNil*) **using** *obtain-fresh-z* **by** *blast*
hence *xx2*: *atom x* ♯ (*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *c*, *GNil*) **using** *fresh-prod7 fresh-prod3* **by** *fast*
have ∗:*P*; *B* ; (*x*, *b*, (*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) $\#_\Gamma$ *GNil* ⊨ $c[z::=V\text{-}var\ x]_v$ (**is** *P*; *B* ; *?G* ⊨ *?c*) **proof** −
have *wfC P B ?G ?c* **using** *wfT-wfC-cons*[*OF assms*(*1*) *assms*(*2*),*of x*] *xx fresh-prod5 fresh-prod3 subst-v-c-def* **by** *metis*
moreover have (∀ *i*. *wfI P ?G i* ∧ *is-satis-g i ?G* ⟶ *is-satis i ?c*) **proof**(*rule allI*, *rule impI*)
fix *i*
assume *as1*: *wfI P ?G i* ∧ *is-satis-g i ?G*
have ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) = ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ x]_v$))
using *assms subst-v-c-def* **by** *auto*
hence *is-satis i* ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ x]_v$)) **using** *is-satis-g.simps as1* **by** *presburger*
moreover have *is-satis i* ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*))) **using** *is-satis.simps eval-c-eqI*[*of i* (*CE-val* (*V-lit l*)) *eval-l l*] *eval-e-uniqueness*
eval-e-valI eval-v-litI **by** *metis*
ultimately show *is-satis i ?c* **using** *is-satis-mp*[*of i*] **by** *metis*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed
moreover have *atom x* ♯ (*P*, *B*, *GNil*, *z1* , *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z*, *c*)
unfolding *fresh-prod5 τ.fresh* **using** *xx fresh-prodN x-fresh-b* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI assms xx xx2* **by** *metis*
qed

lemma *subtype-if*:
assumes *P*; *B*; Γ ⊢ ⦃ *z* : *b* | *c* ⦄ ≲ ⦃ *z′* : *b* | *c′* ⦄ **and**
wfT P B Γ (⦃ *z1* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ ⦄) **and**

wfT P B Γ (⦃ *z2* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$ ⦄) **and**
atom z1 ♯ *v* **and** *atom z* ♯ Γ **and** *atom z1* ♯ *c* **and** *atom z2* ♯ *c'* **and** *atom z2* ♯ *v*
shows *P*; *B* ; Γ ⊢ ⦃ *z1* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ ⦄ ≲ ⦃ *z2* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$ ⦄
proof −
obtain *x*::*x* **where** *xx*: *atom x* ♯ (*P*,*B*,*z*,*c*,*z'*, *c'*, *z1*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z2*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$, Γ)
using *obtain-fresh-z* **by** *blast*
hence *xf*: *atom x* ♯ (*z*, *c*, *z'*, *c'*, Γ) **by** *simp*
have *xf2*: *atom x* ♯ (*z1*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z2*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$, Γ)
using *xx fresh-prod4 fresh-prodN* **by** *metis*

moreover have *P*; *B* ; (*x*, *b*, (*CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ ⊨ (*CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$)$[z2::=V\text{-}var\ x]_v$
(**is** *P*; *B* ; *?G* ⊨ *?c*)
proof −
have *wbc*: *wfC P B ?G ?c* **using** *assms xx fresh-prod4 fresh-prod2 wfT-wfC-cons assms subst-v-c-def* **by** *metis*

moreover have ∀ *i*. *wfI P ?G i* ∧ *is-satis-g i ?G* ⟶ *is-satis i ?c* **proof**(*rule allI*, *rule impI*)
fix *i*
assume *a1*: *wfI P ?G i* ∧ *is-satis-g i ?G*

have ∗: *is-satis i* ((*CE-val v* == *CE-val* (*V-lit l*))) ⟶ *is-satis i* (($c'[z'::=V\text{-}var\ z2]_v$)$[z2::=V\text{-}var\ x]_v$)
proof
assume *a2*: *is-satis i* ((*CE-val v* == *CE-val* (*V-lit l*)))

have *is-satis i* ((*CE-val v* == *CE-val* (*V-lit l*) *IMP* ($c[z::=V\text{-}var\ z1]_v$))$[z1::=V\text{-}var\ x]_v$)
using *a1 is-satis-g.simps* **by** *simp*
moreover have ((*CE-val v* == *CE-val* (*V-lit l*) *IMP* ($c[z::=V\text{-}var\ z1]_v$))$[z1::=V\text{-}var\ x]_v$) = (*CE-val v* == *CE-val* (*V-lit l*) *IMP* (($c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$))
using *assms subst-v-c-def* **by** *simp*
ultimately have *is-satis i* (*CE-val v* == *CE-val* (*V-lit l*) *IMP* (($c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$)) **by** *argo*

hence *is-satis i* (($c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) **using** *a2 is-satis-mp* **by** *auto*
moreover have (($c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) = (($c[z::=V\text{-}var\ x]_v$)) **using** *assms* **by** *auto*
ultimately have *is-satis i* (($c[z::=V\text{-}var\ x]_v$)) **using** *a2 is-satis.simps* **by** *auto*

hence *is-satis-g i* ((*x*,*b*,($c[z::=V\text{-}var\ x]_v$)) $\#_\Gamma$ Γ) **using** *a1 is-satis-g.simps* **by** *meson*
moreover have *wfI P* ((*x*,*b*,($c[z::=V\text{-}var\ x]_v$)) $\#_\Gamma$ Γ) *i* **proof** −
obtain *s* **where** *Some s* = *i x* ∧ *wfRCV P s b* ∧ *wfI P* Γ *i* **using** *wfI-def a1* **by** *auto*
thus *?thesis* **using** *wfI-def* **by** *auto*
qed
ultimately have *is-satis i* (($c'[z'::=V\text{-}var\ x]_v$)) **using** *subtype-valid assms*(*1*) *xf valid.simps* **by** *simp*

moreover have ($c'[z'::=V\text{-}var\ x]_v$) = (($c'[z'::=V\text{-}var\ z2]_v$)$[z2::=V\text{-}var\ x]_v$) **using** *assms* **by** *auto*

ultimately show *is-satis i* $((c'[z'::=V\text{-}var\ z2]_v\ )[z2::=V\text{-}var\ x]_v)$ **by** *auto*
qed

moreover have *?c* = $((CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}lit\ l))\ IMP\ ((c'[z'::=V\text{-}var\ z2]_v\ )[z2::=V\text{-}var\ x]_v))$
using *assms subst-v-c-def* **by** *simp*

moreover have $\exists\, b1\ b2.\ eval\text{-}c\ i\ (CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}lit\ l)\ )\ b1\ \wedge$ $eval\text{-}c\ i\ c'[z'::=V\text{-}var\ z2]_v[z2::=V\text{-}var\ x]_v\ b2$ **proof** −

have *wfC P* $\mathcal{B}$ *?G* (*CE-val v* == *CE-val* (*V-lit l*)) **using** *wbc wfC-elims*(*7*) *assms subst-cv.simps subst-v-c-def* **by** *fastforce*

moreover have *wfC P* $\mathcal{B}$ *?G* $(c'[z'::=V\text{-}var\ z2]_v[z2::=V\text{-}var\ x]_v)$ **proof**(*rule wfT-wfC-cons*)
show ‹ $P;\ \mathcal{B};\ \Gamma \vdash_{wf} \{\!|\ z1 : b\ \mid CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}lit\ l)\ \ IMP\ \ (c[z::=V\text{-}var\ z1]_v)\ |\!\}$ › **using** *assms subst-v-c-def* **by** *auto*
have $\{\!|\ z2 : b\ \mid c'[z'::=V\text{-}var\ z2]_v\ |\!\} = \{\!|\ z' : b\ \mid c'\ |\!\}$ **using** *assms subst-v-c-def* **by** *auto*
thus ‹ $P;\ \mathcal{B};\ \Gamma \vdash_{wf} \{\!|\ z2 : b\ \mid c'[z'::=V\text{-}var\ z2]_v\ |\!\}$ › **using** *assms subtype-elims* **by** *metis*
show ‹*atom x* ♯ (*CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, $c'[z'::=V\text{-}var\ z2]_v$, Γ)› **using** *xx fresh-Pair c.fresh* **by** *metis*
qed

ultimately show *?thesis* **using** *wfI-wfC-eval-c a1 subst-v-c-def* **by** *simp*
qed

ultimately show *is-satis i ?c* **using** *is-satis-imp*[*OF* ∗] **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed
moreover have *atom x* ♯ (*P*, $\mathcal{B}$, Γ, *z1* , *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z2* , *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$)
unfolding *fresh-prod5 τ.fresh* **using** *xx xf2 fresh-prodN x-fresh-b* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI assms xf2* **by** *metis*
qed

lemma *eval-e-concat-eq*:
assumes *wfI* Θ Γ *i*
shows ∃ *s. eval-e i* (*CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*)))) *s* ∧ *eval-e i* (*CE-concat* $[(V\text{-}lit\ (L\text{-}bitvec\ v1))]^{ce}$ $[(V\text{-}lit\ (L\text{-}bitvec\ v2))]^{ce}$) *s*
using *eval-e-valI eval-e-concatI eval-v-litI eval-l.simps* **by** *metis*

lemma *is-satis-eval-e-eq-imp*:
assumes *wfI* Θ Γ *i* **and** *eval-e i e1 s* **and** *eval-e i e2 s*
and *is-satis i* (*CE-val* (*V-var x*) == *e1*) (**is** *is-satis i ?c1*)
shows *is-satis i* (*CE-val* (*V-var x*) == *e2*)
proof −
have ∗:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
hence *eval-e i* (*CE-val* (*V-var x*)) *s* **using** *assms is-satis.simps eval-c-elims*
by (*metis* (*full-types*) *eval-e-uniqueness*)
thus *?thesis* **using** *is-satis.simps eval-c.intros assms* **by** *fastforce*
qed

lemma *valid-eval-e-eq*:
fixes *e1::ce* **and** *e2::ce*
assumes ∀ Γ *i*. *wfI* Θ Γ *i* ⟶ (∃ *s*. *eval-e i e1 s* ∧ *eval-e i e2 s*) **and** Θ; *B*; *GNil* $\vdash_{wf}$ *e1* : *b* **and** Θ; *B*; *GNil* $\vdash_{wf}$ *e2* : *b*
shows Θ; *B*; (*x*, *b*, (*CE-val* (*V-var x*) == *e1*)) #$_\Gamma$ *GNil* ⊨ (*CE-val* (*V-var x*) == *e2*)
proof(*rule validI*)
show Θ; *B*; (*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) == *e2*
proof
have Θ ; *B* ; (*x*, *b*, *TRUE*)#$_\Gamma$*GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) == *e1* **using** *assms wfC-eqI wfE-valI wfV-varI wfX-wfY*
by (*simp add*: *fresh-GNil wfC-e-eq*)
hence Θ ; *B* $\vdash_{wf}$ (*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil* **using** *wfG-consI fresh-GNil wfX-wfY assms wfX-wfB* **by** *metis*
thus Θ; *B*; (*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) : *b* **using** *wfCE-valI wfV-varI wfX-wfY*
lookup.simps assms wfX-wfY **by** *simp*
show Θ; *B*; (*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil* $\vdash_{wf}$ *e2* : *b* **using** *assms wf-weakening wfX-wfY*
by (*metis* (*full-types*) ‹Θ; *B*; (*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) : *b*› *empty-iff subsetI toSet.simps*(*1*))
qed
show ∀ *i*. *wfI* Θ ((*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil*) *i* ∧ *is-satis-g i* ((*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil*) ⟶ *is-satis i* (*CE-val* (*V-var x*) == *e2*)
proof(*rule*,*rule*)
fix *i*
assume *wfI* Θ ((*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil*) *i* ∧ *is-satis-g i* ((*x*, *b*, *CE-val* (*V-var x*) == *e1*) #$_\Gamma$ *GNil*)
moreover then obtain *s* **where** *eval-e i e1 s* ∧ *eval-e i e2 s* **using** *assms* **by** *auto*
ultimately show *is-satis i* (*CE-val* (*V-var x*) == *e2*) **using** *assms is-satis-eval-e-eq-imp is-satis-g.simps* **by** *meson*
qed
qed

lemma *subtype-concat*:
assumes $\vdash_{wf}$ Θ
shows Θ; *B*; *GNil* ⊢ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) ⦄ ≲
⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-concat* [(*V-lit* (*L-bitvec v1*))]^{ce} [(*V-lit* (*L-bitvec v2*))]^{ce} ⦄ (**is** Θ; *B*; *GNil* ⊢ *?t1* ≲ *?t2*)
proof −
obtain *x::x* **where** *x*: *atom x* ♯ (Θ, *B*, *GNil*, *z* , *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))),
z , *CE-val* (*V-var z*) == *CE-concat* [*V-lit* (*L-bitvec v1*)]^{ce} [*V-lit* (*L-bitvec v2*)]^{ce})
(**is** *?xfree*)
using *obtain-fresh* **by** *auto*

have *wb1*: Θ; *B*; *GNil* $\vdash_{wf}$ *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) : *B-bitvec* **using** *wfX-wfY wfCE-valI wfV-litI assms base-for-lit.simps wfG-nilI* **by** *metis*
hence *wb2*: Θ; *B*; *GNil* $\vdash_{wf}$ *CE-concat* [(*V-lit* (*L-bitvec v1*))]^{ce} [(*V-lit* (*L-bitvec v2*))]^{ce} : *B-bitvec*
using *wfCE-concatI wfX-wfY wfV-litI base-for-lit.simps wfCE-valI* **by** *metis*

show *?thesis* **proof**

show Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t1* **using** *wfT-e-eq fresh-GNil wb1 wb2* **by** *metis*
show Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t2* **using** *wfT-e-eq fresh-GNil wb1 wb2* **by** *metis*
show *?xfree* **using** *x* **by** *auto*
show Θ; $\mathcal{B}$; (*x*, *B-bitvec*, (*CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))))[*z*::=*V-var* *x*]$_v$) $\#_\Gamma$
GNil $\models$ (*CE-val* (*V-var z*) == *CE-concat* [(*V-lit* (*L-bitvec v1*))]^{ce} [(*V-lit* (*L-bitvec v2*))]^{ce})[*z*::=*V-var x*]$_v$
using *valid-eval-e-eq eval-e-concat-eq wb1 wb2 subst-v-c-def* **by** *fastforce*
qed
qed

lemma *subtype-len*:
assumes $\vdash_{wf}$ Θ
shows Θ; $\mathcal{B}$; *GNil* $\vdash$ ⦃ *z'* : *B-int* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) ⦄ $\lesssim$
⦃ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* [(*V-lit* (*L-bitvec v*))]^{ce} ⦄ (**is** Θ; $\mathcal{B}$; *GNil* $\vdash$ *?t1* $\lesssim$ *?t2*)
proof −

have ∗: Θ $\vdash_{wf}$ [] ∧ Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ []$_\Delta$ **using** *assms wfG-nilI wfD-emptyI wfPhi-emptyI* **by** *auto*
obtain *x*::*x* **where** *x*: *atom x* ♯ (Θ, $\mathcal{B}$, *GNil*, *z'* , *CE-val* (*V-var z'*) ==
CE-val (*V-lit* (*L-num* (*int* (*length v*)))) , *z*, *CE-val* (*V-var z*) == *CE-len* [(*V-lit* (*L-bitvec* *v*))]^{ce})
(**is** *atom x* ♯ *?F*)
using *obtain-fresh* **by** *metis*
then show *?thesis* **proof**
have Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) : *B-int*
using *wfCE-valI* ∗ *wfV-litI base-for-lit.simps*
by (*metis wfE-valI wfX-wfY*)

thus Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t1* **using** *wfT-e-eq fresh-GNil* **by** *auto*

have Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-len* [(*V-lit* (*L-bitvec v*))]^{ce} : *B-int*
using *wfE-valI* ∗ *wfV-litI base-for-lit.simps wfE-valI wfX-wfY*
by (*metis wfCE-lenI wfCE-valI*)

thus Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t2* **using** *wfT-e-eq fresh-GNil* **by** *auto*

show Θ; $\mathcal{B}$; (*x*, *B-int*, (*CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))))[*z'*::=*V-var* *x*]$_v$) $\#_\Gamma$ *GNil* $\models$ (*CE-val* (*V-var z*) == *CE-len* [(*V-lit* (*L-bitvec v*))]^{ce})[*z*::=*V-var x*]$_v$
(**is** Θ; $\mathcal{B}$; *?G* $\models$ *?c*) **using** *valid-len assms subst-v-c-def* **by** *auto*
qed
qed

lemma *subtype-base-fresh*:
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄ **and** Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *z* : *b* | *c'* ⦄ **and**
atom z ♯ Γ **and** Θ; $\mathcal{B}$; (*z*, *b*, *c*) $\#_\Gamma$ Γ $\models$ *c'*
shows Θ; $\mathcal{B}$; Γ $\vdash$ ⦃ *z* : *b* | *c* ⦄ $\lesssim$ ⦃ *z* : *b* | *c'* ⦄
proof −
obtain *x*::*x* **where** ∗:*atom x* ♯ ((Θ , $\mathcal{B}$, *z*, *c*, *z*, *c'*, Γ) , (Θ, $\mathcal{B}$, Γ, ⦃ *z* : *b* | *c* ⦄, ⦃ *z* : *b* | *c'* ⦄)) **using** *obtain-fresh* **by** *metis*
moreover hence *atom x* ♯ Γ **using** *fresh-Pair* **by** *auto*

moreover hence Θ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ ⊨ *c*′[*z*::=*V-var x*]$_v$ **using** *assms valid-flip-simple ∗ subst-v-c-def* **by** *auto*
ultimately show *?thesis* **using** *subtype-baseI assms τ.fresh fresh-Pair* **by** *metis*
qed

lemma *subtype-bop-arith*:
assumes *wfG* Θ $\mathcal{B}$ Γ **and** (*opp* = *Plus* ∧ *ll* = (*L-num* (*n1*+*n2*))) ∨ (*opp* = *LEq* ∧ *ll* = (*if n1*≤*n2 then L-true else L-false*))
and (*opp* = *Plus* ⟶ *b* = *B-int*) ∧ (*opp* = *LEq* ⟶ *b* = *B-bool*)
shows Θ; $\mathcal{B}$; Γ ⊢ (⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit* (*ll*))) ⦄) ≲
⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op opp* [(*V-lit* (*L-num n1*))]^{ce} [(*V-lit* (*L-num n2*))]^{ce}) ⦄ (**is** Θ; $\mathcal{B}$; Γ ⊢ *?T1* ≲ *?T2*)
proof −
obtain *x*::*x* **where** *xf*: *atom x* ♯ (*z*, *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*ll*)) , *z*, *CE-val* (*V-var z*) == *CE-op opp* [(*V-lit* (*L-num n1*))]^{ce} [(*V-lit* (*L-num n2*))]^{ce} , Γ)
using *obtain-fresh* **by** *blast*

have Θ; $\mathcal{B}$; Γ ⊢ (⦃ *x* : *b* | *C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*ll*))) ⦄) ≲
⦃ *x* : *b* | *C-eq* (*CE-val* (*V-var x*)) (*CE-op opp* [(*V-lit* (*L-num n1*))]^{ce} [(*V-lit* (*L-num n2*))]^{ce}) ⦄ (**is** Θ; $\mathcal{B}$; Γ ⊢ *?S1* ≲ *?S2*)
proof(*rule subtype-base-fresh*)

show *atom x* ♯ Γ **using** *xf fresh-Pair* **by** *auto*

show *wfT* Θ $\mathcal{B}$ Γ (⦃ *x* : *b* | *CE-val* (*V-var x*) == *CE-val* (*V-lit ll*) ⦄) (**is** *wfT* Θ $\mathcal{B}$ *?A ?B*)
proof(*rule wfT-e-eq*)
have Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ (*V-lit ll*) : *b* **using** *wfV-litI base-for-lit.simps assms* **by** *metis*
thus Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-val* (*V-lit ll*) : *b* **using** *wfCE-valI* **by** *auto*
show *atom x* ♯ Γ **using** *xf fresh-Pair* **by** *auto*
qed

consider *opp* = *Plus* | *opp* = *LEq* **using** *opp.exhaust assms* **by** *blast*
then show *wfT* Θ $\mathcal{B}$ Γ (⦃ *x* : *b* | *CE-val* (*V-var x*) == *CE-op opp* [(*V-lit* (*L-num n1*))]^{ce} [(*V-lit* (*L-num n2*))]^{ce} ⦄) (**is** *wfT* Θ $\mathcal{B}$ *?A ?C*)
proof(*cases*)
case *1*
then show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *x* : *b* | [[*x*]^v]^{ce} == [*opp* [[*L-num n1*]^v]^{ce} [[*L-num n2*]^v]^{ce}]^{ce} ⦄
using *wfCE-valI wfCE-plusI assms wfV-litI base-for-lit.simps assms*
by (*metis* ‹*atom x* ♯ Γ› *wfT-e-eq*)
next
case *2*
then show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *x* : *b* | [[*x*]^v]^{ce} == [*opp* [[*L-num n1*]^v]^{ce} [[*L-num n2*]^v]^{ce}]^{ce} ⦄
using *wfCE-valI wfCE-plusI assms wfV-litI base-for-lit.simps assms*

by (*metis* ‹*atom x* ♯ Γ› *wfCE-leqI wfT-e-eq*)
qed

show Θ; $\mathcal{B}$; (*x*, *b*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*ll*)))) $\#_\Gamma$ Γ
⊨ (*CE-val* (*V-var x*) == *CE-op opp* [*V-lit* (*L-num n1*)]^{ce} [*V-lit* (*L-num n2*)]^{ce})
(**is** Θ; $\mathcal{B}$; *?G* ⊨ *?c*)

using *valid-arith-bop assms xf* **by** *simp*

qed
moreover have *?S1 = ?T1* **using** *type-l-eq* **by** *auto*
moreover have *?S2 = ?T2* **using** *type-e-eq ce.fresh v.fresh supp-l-empty fresh-def empty-iff fresh-e-opp*

by (*metis ms-fresh-all*(*4*))
ultimately show *?thesis* **by** *auto*

qed

lemma *subtype-bop-eq*:
assumes *wfG* Θ $\mathcal{B}$ Γ **and** *base-for-lit l1 = base-for-lit l2*
shows Θ; $\mathcal{B}$; Γ ⊢ (⦃ *z* : *B-bool* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit* (*if l1 = l2 then L-true else L-false*))) ⦄) $\lesssim$
⦃ *z* : *B-bool* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op Eq* $[(V\text{-}lit\ l1)]^{ce}$ $[(V\text{-}lit\ l2)]^{ce}$) ⦄ (**is** Θ; $\mathcal{B}$; Γ ⊢ *?T1* $\lesssim$ *?T2*)
proof −
let *?ll = if l1 = l2 then L-true else L-false*
obtain *x::x* **where** *xf*: *atom x* ♯ (*z*, *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*if l1 = l2 then L-true else L-false*)) , *z*, *CE-val* (*V-var z*) == *CE-op Eq* $[(V\text{-}lit\ l1)]^{ce}$ $[(V\text{-}lit\ l2)]^{ce}$, Γ, (Θ, $\mathcal{B}$, Γ))
using *obtain-fresh* **by** *blast*

have Θ; $\mathcal{B}$; Γ ⊢ (⦃ *x* : *B-bool* | *C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*?ll*))) ⦄) $\lesssim$
⦃ *x* : *B-bool* | *C-eq* (*CE-val* (*V-var x*)) (*CE-op Eq* $[(V\text{-}lit\ (l1))]^{ce}$ $[(V\text{-}lit\ (l2))]^{ce}$) ⦄ (**is** Θ; $\mathcal{B}$; Γ ⊢ *?S1* $\lesssim$ *?S2*)
proof(*rule subtype-base-fresh*)

show *atom x* ♯ Γ **using** *xf fresh-Pair* **by** *auto*

show *wfT* Θ $\mathcal{B}$ Γ (⦃ *x* : *B-bool* | *CE-val* (*V-var x*) == *CE-val* (*V-lit ?ll*) ⦄) (**is** *wfT* Θ $\mathcal{B}$ *?A ?B*)
proof(*rule wfT-e-eq*)
have Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ (*V-lit ?ll*) : *B-bool* **using** *wfV-litI base-for-lit.simps assms* **by** *metis*
thus Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-val* (*V-lit ?ll*) : *B-bool* **using** *wfCE-valI* **by** *auto*
show *atom x* ♯ Γ **using** *xf fresh-Pair* **by** *auto*
qed

show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *x* : *B-bool* | $[\ [\ x\ ]^v\ ]^{ce}$ == $[\ eq\ [\ [\ l1\ ]^v\ ]^{ce}\ [\ [\ l2\ ]^v\ ]^{ce}\ ]^{ce}$ ⦄
proof(*rule wfT-e-eq*)
show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $[\ eq\ [\ [\ l1\ ]^v\ ]^{ce}\ [\ [\ l2\ ]^v\ ]^{ce}\ ]^{ce}$: *B-bool*
apply(*rule wfCE-eqI*, *rule wfCE-valI*)
apply(*rule wfV-litI*, *simp add*: *assms*)
using *wfV-litI assms wfCE-valI* **by** *auto*
show *atom x* ♯ Γ **using** *xf fresh-Pair* **by** *auto*
qed

show Θ; $\mathcal{B}$; (*x*, *B-bool*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*?ll*)))) $\#_\Gamma$ Γ
⊨ (*CE-val* (*V-var x*) == *CE-op Eq* $[V\text{-}lit\ (l1)]^{ce}$ $[V\text{-}lit\ (l2)]^{ce}$) (**is** Θ; $\mathcal{B}$; *?G* ⊨ *?c*)
using *valid-eq-bop assms xf* **by** *auto*

qed

moreover have *?S1* = *?T1* **using** *type-l-eq* **by** *auto*
moreover have *?S2* = *?T2* **using** *type-e-eq ce.fresh v.fresh supp-l-empty fresh-def empty-iff fresh-e-opp*

by (*metis ms-fresh-all*(*4*))
ultimately show *?thesis* **by** *auto*

qed

lemma *subtype-top*:
assumes *wfT* Θ *B* *G* ({| *z* : *b* | *c* |})
shows Θ; *B*; *G* ⊢ ({| *z* : *b* | *c* |}) ≲ ({| *z* : *b* | *TRUE* |})
proof −
obtain *x*::*x* **where** ∗: *atom x* ♯ (Θ, *B*, *G*, *z* , *c*, *z* , *TRUE*) **using** *obtain-fresh* **by** *blast*
then show *?thesis* **proof**(*rule subtype-baseI*)
show ‹ Θ; *B*; *G* $\vdash_{wf}$ {| *z* : *b* | *c* |} › **using** *assms* **by** *auto*
show ‹ Θ; *B*;*G* $\vdash_{wf}$ {| *z* : *b* | *TRUE* |} › **using** *wfT-TRUE assms wfX-wfY b-of.simps wfT-wf*
by (*metis wfX-wfB*(*8*))
hence Θ ; *B* $\vdash_{wf}$ (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ *G* **using** *wfT-wf-cons3 assms fresh-Pair* ∗ *subst-v-c-def* **by** *auto*
thus ‹Θ; *B*; (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ *G* ⊨ (*TRUE*)[*z*::=*V-var x*]$_v$ › **using** *valid-trueI subst-cv.simps subst-v-c-def* **by** *metis*
qed
qed

lemma *if-simp*:
(*if x* = *x then e1 else e2*) = *e1*
by *auto*

lemma *subtype-split*:
assumes *split n v* (*v1*,*v2*) **and** $\vdash_{wf}$ Θ
shows Θ ; {||} ; *GNil* ⊢ {| *z* : [*B-bitvec* , *B-bitvec*]^b | [[*z*]^v]^{ce} == [[[*L-bitvec*
v1]^v , [*L-bitvec*
v2]^v]^v]^{ce} |} ≲ {| *z* : [*B-bitvec* , *B-bitvec*]^b | [[*L-bitvec*
v]^v]^{ce} == [[#*1*[[*z*]^v]^{ce}]^{ce} @@ [#*2*[[*z*]^v]^{ce}]^{ce}]^{ce} *AND* [| [#*1*[[*z*]^v]^{ce}]^{ce} |]^{ce} == [[*L-num*
n]^v]^{ce} |}
(**is** Θ ;*?B* ; *GNil* ⊢ {| *z* : [*B-bitvec* , *B-bitvec*]^b | *?c1* |} ≲ {| *z* : [*B-bitvec* , *B-bitvec*]^b | *?c2* |})
proof −
obtain *x*::*x* **where** *xf*:*atom x* ♯ (Θ, *?B*, *GNil*, *z*, *?c1*,*z*, *?c2*) **using** *obtain-fresh* **by** *auto*
then show *?thesis* **proof**(*rule subtype-baseI*)
show ∗: ‹Θ ; *?B* ; (*x*, [*B-bitvec* , *B-bitvec*]^b, (*?c1*)[*z*::=[*x*]^v]$_v$) $\#_\Gamma$
GNil ⊨ (*?c2*)[*z*::=[*x*]^v]$_v$ ›
unfolding *subst-v-c-def subst-cv.simps subst-cev.simps subst-vv.simps if-simp*
using *valid-split*[*OF assms*, *of x*] **by** *simp*
show ‹ Θ ; *?B* ; *GNil* $\vdash_{wf}$ {| *z* : [*B-bitvec* , *B-bitvec*]^b| *?c1* |} › **using** *valid-wfT*[*OF* ∗] *xf fresh-prodN* **by** *metis*
show ‹ Θ ; *?B* ; *GNil* $\vdash_{wf}$ {| *z* : [*B-bitvec* , *B-bitvec*]^b| *?c2* |} › **using** *valid-wfT*[*OF* ∗] *xf fresh-prodN* **by** *metis*
qed
qed

lemma *subtype-range*:

fixes *n::int* **and** Γ::Γ
assumes *0* ≤ *n* ∧ *n* ≤ *int* (*length v*) **and** Θ ; {||} $\vdash_{wf}$ Γ
shows Θ ; {||} ; Γ ⊢ ⦃ *z* : *B-int* | [[*z*]^{v}]^{ce} == [[*L-num n*]^{v}]^{ce} ⦄ ≲
⦃ *z* : *B-int* | ([*leq* [[*L-num 0*]^{v}]^{ce} [[*z*]^{v}]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}) *AND* (
[*leq* [[*z*]^{v}]^{ce} [| [[*L-bitvec v*]^{v}]^{ce} |]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}) ⦄
(**is** Θ ; *?B* ; Γ ⊢ ⦃ *z* : *B-int* | *?c1* ⦄ ≲ ⦃ *z* : *B-int* | *?c2 AND ?c3* ⦄)
proof −
obtain *x::x* **where** ∗:‹*atom x* ♯ (Θ, *?B*, Γ, *z*, *?c1* , *z*, *?c2 AND ?c3*)› **using** *obtain-fresh* **by** *auto*
moreover have ∗∗:‹Θ ; *?B* ; (*x*, *B-int*, (*?c1*)[*z*::=[*x*]^{v}]$_{v}$) $\#_{\Gamma}$ Γ ⊨ (*?c2 AND ?c3*)[*z*::=[*x*]^{v}]$_{v}$›
unfolding *subst-v-c-def subst-cv.simps subst-cev.simps subst-vv.simps if-simp* **using** *valid-range-length*[*OF assms*(*1*)] *assms fresh-prodN* ∗ **by** *simp*

moreover hence ‹ Θ ; *?B* ; Γ $\vdash_{wf}$ ⦃ *z* : *B-int* | [[*z*]^{v}]^{ce} == [[*L-num n*]^{v}]^{ce} ⦄ › **using** *valid-wfT* ∗ *fresh-prodN* **by** *metis*
moreover have ‹ Θ ; *?B* ; Γ $\vdash_{wf}$ ⦃ *z* : *B-int* | *?c2 AND ?c3* ⦄ ›
using *valid-wfT*[*OF* ∗∗] ∗ *fresh-prodN* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI* **by** *auto*
qed

lemma *check-num-range*:
assumes *0* ≤ *n* ∧ *n* ≤ *int* (*length v*) **and** $\vdash_{wf}$ Θ
shows Θ ; {||} ; *GNil* ⊢ ([*L-num n*]^{v}) ⇐ ⦃ *z* : *B-int* | ([*leq* [[*L-num 0*]^{v}]^{ce} [[*z*]^{v}]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}) *AND*
[*leq* [[*z*]^{v}]^{ce} [| [[*L-bitvec v*]^{v}]^{ce} |]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce} ⦄
using *assms subtype-range check-v.intros infer-v-litI wfG-nilI*
by (*meson infer-natI*)

12.3 Literals

nominal-function *type-for-lit* :: *l* ⇒ τ **where**
type-for-lit (*L-true*) = (⦃ *z* : *B-bool* | [[*z*]^{v}]^{ce} == [*V-lit L-true*]^{ce} ⦄)
| *type-for-lit* (*L-false*) = (⦃ *z* : *B-bool* | [[*z*]^{v}]^{ce} == [*V-lit L-false*]^{ce} ⦄)
| *type-for-lit* (*L-num n*) = (⦃ *z* : *B-int* | [[*z*]^{v}]^{ce} == [*V-lit* (*L-num n*)]^{ce} ⦄)
| *type-for-lit* (*L-unit*) = (⦃ *z* : *B-unit* | [[*z*]^{v}]^{ce} == [*V-lit* (*L-unit*)]^{ce} ⦄)
| *type-for-lit* (*L-bitvec v*) = (⦃ *z* : *B-bitvec* | [[*z*]^{v}]^{ce} == [*V-lit* (*L-bitvec v*)]^{ce} ⦄)
by (*auto simp*: *eqvt-def type-for-lit-graph-aux-def*, *metis l.strong-exhaust*,(*simp add*: *permute-int-def flip-bitvec0*)+)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *type-for-var* :: Γ ⇒ τ ⇒ *x* ⇒ τ **where**
type-for-var G τ *x* = (*case lookup G x of*
None ⇒ τ
| *Some* (*b*,*c*) ⇒ (⦃ *x* : *b* | *c* ⦄))
apply *auto* **unfolding** *eqvt-def* **apply**(*rule allI*) **unfolding** *type-for-var-graph-aux-def eqvt-def* **by** *simp*
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *infer-l-form*:
fixes *l::l* **and** *tm::'a::fs*
assumes ⊢ *l* ⇒ τ
shows ∃ *z b*. τ = (⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) ⦄) ∧ *atom z* ♯ *tm*

proof −
obtain z' **and** b **where** t:τ = ({ z' : b | *C-eq* (*CE-val* (*V-var* z')) (*CE-val* (*V-lit* l)) }) **using** *infer-l-elims assms* **using** *infer-l.simps type-for-lit.simps*
type-for-lit.cases **by** *blast*
obtain z::x **where** *zf*: *atom z* ♯ *tm* **using** *obtain-fresh* **by** *metis*
have τ = { z : b | *C-eq* (*CE-val* (*V-var* z)) (*CE-val* (*V-lit* l)) } **using** *type-e-eq ce.fresh v.fresh l.fresh*

by (*metis t type-l-eq*)
thus *?thesis* **using** *zf* **by** *auto*
qed

lemma *infer-l-form3*:
fixes l::l
assumes ⊢ l ⇒ τ
shows ∃ z. τ = ({ z : *base-for-lit l* | *C-eq* (*CE-val* (*V-var* z)) (*CE-val* (*V-lit* l)) })
using *infer-l-elims* **using** *assms* **using** *infer-l.simps type-for-lit.simps base-for-lit.simps* **by** *auto*

lemma *infer-l-form4*[*simp*]:
fixes Γ::Γ
assumes Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ
shows ∃ z. ⊢ l ⇒ ({ z : *base-for-lit l* | *C-eq* (*CE-val* (*V-var* z)) (*CE-val* (*V-lit* l)) })
using *assms infer-l-form2 infer-l-form3* **by** *metis*

lemma *infer-v-unit-form*:
fixes v::v
assumes P ; $\mathcal{B}$; Γ ⊢ v ⇒ ({ *z1* : *B-unit* | *c1* }) **and** *supp v* = {}
shows v = *V-lit L-unit*
using *assms* **proof**(*nominal-induct* Γ v { *z1* : *B-unit* | *c1* } *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ *c x z*)
then show *?case* **using** *supp-at-base* **by** *auto*
next
case (*infer-v-litI* Θ $\mathcal{B}$ Γ l)
from ‹⊢ l ⇒ { *z1* : *B-unit* | *c1* }› **show** *?case* **by**(*nominal-induct* { *z1* : *B-unit* | *c1* } *rule*: *infer-l.strong-induct*,*auto*)
qed

lemma *base-for-lit-wf*:
assumes $\vdash_{wf}$ Θ
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ *base-for-lit l*
using *base-for-lit.simps* **using** *wfV-elims wf-intros assms l.exhaust* **by** *metis*

lemma *infer-l-t-wf*:
fixes Γ::Γ
assumes Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ ∧ *atom z* ♯ Γ
shows Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ { z : *base-for-lit l* | *C-eq* (*CE-val* (*V-var* z)) (*CE-val* (*V-lit* l)) }
proof
show *atom z* ♯ (Θ, $\mathcal{B}$, Γ) **using** *wfG-fresh-x assms* **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *base-for-lit l* **using** *base-for-lit-wf assms wfX-wfY* **by** *metis*
thus Θ; $\mathcal{B}$; (z, *base-for-lit l*, *TRUE*) $\#_{\Gamma}$ Γ $\vdash_{wf}$ *CE-val* (*V-var* z) == *CE-val* (*V-lit* l) **using** *wfC-v-eq wfV-litI assms wfX-wfY* **by** *metis*
qed

lemma *infer-l-wf*:
fixes *l*::*l* **and** Γ::Γ **and** τ::τ **and** Θ::Θ
assumes ⊢ *l* ⇒ τ **and** Θ ; *B* ⊢$_{wf}$ Γ
shows ⊢$_{wf}$ Θ **and** Θ ; *B* ⊢$_{wf}$ Γ **and** Θ; *B*; Γ ⊢$_{wf}$ τ
proof −
show ∗:Θ ; *B* ⊢$_{wf}$ Γ **using** *assms infer-l-elims* **by** *auto*
thus ⊢$_{wf}$ Θ **using** *wfX-wfY* **by** *auto*
show ∗:Θ ; *B* ; Γ ⊢$_{wf}$ τ **using** *infer-l-t-wf assms infer-l-form3* ∗
by (*metis* ‹⊢$_{wf}$ Θ› *fresh-GNil wfG-nilI wfT-weakening-nil*)
qed

lemma *infer-l-uniqueness*:
fixes *l*::*l*
assumes ⊢ *l* ⇒ τ **and** ⊢ *l* ⇒ τ′
shows τ = τ′
using *assms*
proof −
obtain *z* **and** *b* **where** *zt*: τ = (⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) ⦄) **using** *infer-l-form assms* **by** *blast*
obtain *z′* **and** *b* **where** *z′t*: τ′ = (⦃ *z′* : *b* | *C-eq* (*CE-val* (*V-var z′*)) (*CE-val* (*V-lit l*)) ⦄) **using** *infer-l-form assms* **by** *blast*
thus *?thesis* **using** *type-l-eq zt z′t assms infer-l.simps infer-l-elims l.distinct*
by (*metis infer-l-form3*)
qed

12.4 Values

lemma *type-v-eq*:
assumes ⦃ *z1* : *b1* | *c1* ⦄ = ⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var x*)) ⦄ **and** *atom z* ♯ *x*
shows *b* = *b1* **and** *c1* = *C-eq* (*CE-val* (*V-var z1*)) (*CE-val* (*V-var x*))
using *assms* **by** (*auto*,*metis Abs1-eq-iff* τ*.eq-iff assms c.fresh ce.fresh type-e-eq v.fresh*)

lemma *infer-var2* [*elim*]:
assumes *P*; *B* ; *G* ⊢ *V-var x* ⇒ τ
shows ∃ *b c*. *Some* (*b*,*c*) = *lookup G x*
using *assms infer-v-elims lookup-iff* **by** (*metis* (*no-types*, *lifting*))

lemma *infer-var3* [*elim*]:
assumes Θ; *B*; Γ ⊢ *V-var x* ⇒ τ
shows ∃ *z b c*. *Some* (*b*,*c*) = *lookup* Γ *x* ∧ τ = (⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var x*)) ⦄) ∧ *atom z* ♯ *x* ∧ *atom z* ♯ (Θ, *B*, Γ)
using *infer-v-elims*(*1*)[*OF assms*(*1*)] **by** *metis*

lemma *infer-bool-options2*:
fixes *v*::*v*
assumes Θ; *B*; Γ ⊢ *v* ⇒ ⦃ *z* : *b* | *c* ⦄ **and** *supp v* = {} ∧ *b* = *B-bool*
shows *v* = *V-lit L-true* ∨ (*v* = (*V-lit L-false*))
using *assms*
proof(*nominal-induct* ⦃ *z* : *b* | *c* ⦄ *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ *B* Γ *c x z*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next

case (*infer-v-litI* Θ *ℬ* Γ *l*)
from ‹ ⊢ *l* ⇒ ⦃ *z* : *b* | *c* ⦄› **show** *?case* **proof**(*nominal-induct* ⦃ *z* : *b* | *c* ⦄ *rule*: *infer-l.strong-induct*)
case (*infer-trueI z*)
then show *?case* **by** *auto*
next
case (*infer-falseI z*)
then show *?case* **by** *auto*
next
case (*infer-natI n z*)
then show *?case* **using** *infer-v-litI* **by** *simp*
next
case (*infer-unitI z*)
then show *?case* **using** *infer-v-litI* **by** *simp*
next
case (*infer-bitvecI bv z*)
then show *?case* **using** *infer-v-litI* **by** *simp*
qed
qed(*auto*+)

lemma *infer-bool-options*:
fixes *v*::*v*
assumes Θ; *ℬ*; Γ ⊢ *v* ⇒ ⦃ *z* : *B-bool* | *c* ⦄ **and** *supp v* = {}
shows *v* = *V-lit L-true* ∨ (*v* = (*V-lit L-false*))
using *infer-bool-options2 assms* **by** *blast*

lemma *infer-int2*:
fixes *v*::*v*
assumes Θ; *ℬ*; Γ ⊢ *v* ⇒ ⦃ *z* : *b* | *c* ⦄
shows *supp v* = {} ∧ *b* = *B-int* ⟶ (∃ *n*. *v* = *V-lit* (*L-num n*))
using *assms*
proof(*nominal-induct* ⦃ *z* : *b* | *c* ⦄ *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ *ℬ* Γ *c x z*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next
case (*infer-v-litI* Θ *ℬ* Γ *l*)
from ‹ ⊢ *l* ⇒ ⦃ *z* : *b* | *c* ⦄› **show** *?case* **proof**(*nominal-induct* ⦃ *z* : *b* | *c* ⦄ *rule*: *infer-l.strong-induct*)
case (*infer-trueI z*)
then show *?case* **by** *auto*
next
case (*infer-falseI z*)
then show *?case* **by** *auto*
next
case (*infer-natI n z*)
then show *?case* **using** *infer-v-litI* **by** *simp*
next
case (*infer-unitI z*)
then show *?case* **using** *infer-v-litI* **by** *simp*
next
case (*infer-bitvecI bv z*)
then show *?case* **using** *infer-v-litI* **by** *simp*
qed
qed(*auto*+)

lemma *infer-bitvec*:
  fixes Θ::Θ **and** *v*::*v*
  assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ ⦃ *z'* : *B-bitvec* | *c'* ⦄ **and** *supp v* = {}
  shows ∃ *bv*. *v* = *V-lit* (*L-bitvec bv*)
  using *assms* **proof**(*nominal-induct v rule*: *v.strong-induct*)
  case (*V-lit l*)
  then show *?case* **by**(*nominal-induct l rule*: *l.strong-induct*,*force+*)
next
  case (*V-consp s dc b v*)
  then show *?case* **using** *infer-v-elims*(*7*)[*OF V-consp*(*2*)] *τ.eq-iff* **by** *auto*
next
  case (*V-var x*)
  then show *?case* **using** *supp-at-base* **by** *auto*
qed(*force+*)

lemma *infer-int*:
  assumes *infer-v* Θ $\mathcal{B}$ Γ *v* (⦃ *z* : *B-int* | *c* ⦄) **and** *supp v*= {}
  shows ∃ *n*. *V-lit* (*L-num n*) = *v*
  using *assms infer-int2* **by** (*metis* (*no-types*, *lifting*))

lemma *infer-lit*:
  assumes *infer-v* Θ $\mathcal{B}$ Γ *v* (⦃ *z* : *b* | *c* ⦄) **and** *supp v*= {} **and** *b* ∈ { *B-bool* , *B-int* , *B-unit* }
  shows ∃ *l*. *V-lit l* = *v*
  using *assms* **proof**(*nominal-induct v rule*: *v.strong-induct*)
  case (*V-lit x*)
  then show *?case* **by** (*simp add*: *supp-at-base*)
next
  case (*V-var x*)
  then show *?case*
    by (*simp add*: *supp-at-base*)
next
  case (*V-pair x1a x2a*)
  then show *?case* **using** *supp-at-base* **by** *auto*
next
  case (*V-cons x1a x2a x3*)
  then show *?case* **using** *supp-at-base* **by** *auto*
next
  case (*V-consp x1a x2a x3 x4*)
  then show *?case* **using** *supp-at-base* **by** *auto*
qed

lemma *infer-v-form*[*simp*]:
  fixes *v*::*v*
  assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ *τ*
  shows ∃ *z b*. *τ* = (⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)⦄) ∧ *atom z* ♯ *v* ∧ *atom z* ♯ (Θ, $\mathcal{B}$, Γ)
  using *assms*
proof(*nominal-induct rule*: *infer-v.strong-induct*)
  case (*infer-v-varI* Θ $\mathcal{B}$ Γ *b c x z*)
  then show *?case* **by** *force*
next
  case (*infer-v-litI* Θ $\mathcal{B}$ Γ *l τ*)

then obtain *z* **and** *b* **where** $\tau = \{\!\!\{ z : b \mid$ *CE-val* (*V-var z*) == *CE-val* (*V-lit l*) $\}\!\!\}$ ∧*atom z* ♯ (Θ, $\mathcal{B}$, Γ)
using *infer-l-form* **by** *metis*
moreover hence *atom z* ♯ (*V-lit l*) **using** *supp-l-empty v.fresh*(*1*) *fresh-prod2 fresh-def* **by** *blast*
ultimately show *?case* **by** *metis*
next
case (*infer-v-pairI z v1 v2* Θ $\mathcal{B}$ Γ *t1 t2*)
then show *?case* **by** *force*
next
case (*infer-v-consI s dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv z*)
moreover hence *atom z* ♯ (*V-cons s dc v*) **using**
Un-commute b.supp(*3*) *fresh-def sup-bot.right-neutral supp-b-empty v.supp*(*4*) *pure-supp* **by** *metis*
ultimately show *?case* **using** *fresh-prodN* **by** *metis*
next
case (*infer-v-conspI s bv dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv b z*)
moreover hence *atom z* ♯ (*V-consp s dc b v*) **unfolding** *v.fresh* **using** *pure-fresh fresh-prodN* ∗ **by** *metis*
ultimately show *?case* **using** *fresh-prodN* **by** *metis*
qed

lemma *infer-v-form2*:
fixes *v*::*v*
assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ ($\{\!\!\{ z : b \mid c \}\!\!\}$) **and** *atom z* ♯ *v*
shows *c* = *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)
using *assms*
proof −
obtain *z′* **and** *b′* **where** ($\{\!\!\{ z : b \mid c \}\!\!\}$) = ($\{\!\!\{ z' : b' \mid$ *CE-val* (*V-var z′*) == *CE-val v* $\}\!\!\}$) ∧ *atom z′* ♯ *v*
using *infer-v-form assms* **by** *meson*
thus *?thesis* **using** *Abs1-eq-iff*(*3*) *τ.eq-iff type-e-eq*
by (*metis assms*(*2*) *ce.fresh*(*1*))
qed

lemma *infer-v-form3*:
fixes *v*::*v*
assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ *τ* **and** *atom z* ♯ (*v*,Γ)
shows Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ $\{\!\!\{ z :$ *b-of τ* $\mid$ *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)$\}\!\!\}$
proof −
obtain *z′* **and** *b′* **where** *τ* = $\{\!\!\{ z' : b' \mid$ *C-eq* (*CE-val* (*V-var z′*)) (*CE-val v*)$\}\!\!\}$ ∧ *atom z′* ♯ *v* ∧ *atom z′* ♯ (Θ, $\mathcal{B}$, Γ)
using *infer-v-form assms* **by** *metis*
moreover hence $\{\!\!\{ z' : b' \mid$ *C-eq* (*CE-val* (*V-var z′*)) (*CE-val v*)$\}\!\!\}$ = $\{\!\!\{ z : b' \mid$ *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)$\}\!\!\}$
using *assms type-e-eq fresh-Pair ce.fresh* **by** *auto*
ultimately show *?thesis* **using** *b-of.simps assms* **by** *auto*
qed

lemma *infer-v-form4*:
fixes *v*::*v*
assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ *τ* **and** *atom z* ♯ (*v*,Γ) **and** *b* = *b-of τ*
shows Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ $\{\!\!\{ z : b \mid$ *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)$\}\!\!\}$
using *assms infer-v-form3* **by** *simp*

```
lemma infer-v-v-wf:
  fixes v::v
  shows Θ; ℬ ; G ⊢ v ⇒ τ ⟹  Θ; ℬ; G ⊢wf v : (b-of τ)
proof(induct rule: infer-v.induct)
  case (infer-v-varI Θ ℬ Γ b c x z)
  then show ?case using  wfC-elims  wf-intros by auto
next
  case (infer-v-pairI z v1 v2 Θ ℬ Γ t1 t2)
  then show ?case using  wfC-elims  wf-intros by auto
next
  case (infer-v-litI Θ ℬ Γ l τ)
  hence b-of τ = base-for-lit l using infer-l-form3 b-of.simps by metis
  then show ?case using wfV-litI infer-l-wf infer-v-litI wfG-b-weakening
    by (metis fempty-fsubsetI)
next
  case (infer-v-consI s dclist Θ dc tc ℬ Γ v tv z)
  then show ?case using  wfC-elims  wf-intros
    by (metis (no-types, lifting) b-of.simps has-fresh-z2 subtype-eq-base2)
next
  case (infer-v-conspI s bv dclist Θ dc tc ℬ Γ v tv b z)
  obtain z1 b1 c1 where t:tc = ⦃ z1 : b1 | c1 ⦄ using obtain-fresh-z by metis
  show ?case unfolding b-of.simps proof(rule wfV-conspI)
    show ‹AF-typedef-poly s bv dclist ∈ set Θ› using infer-v-conspI by auto
    show ‹(dc,  ⦃ z1 : b1 | c1 ⦄ ) ∈ set dclist› using infer-v-conspI t by auto
    show ‹ Θ ; ℬ  ⊢wf b › using infer-v-conspI by auto
    show ‹atom bv ♯ (Θ, ℬ, Γ, b, v)› using infer-v-conspI by auto
   have  b1[bv::=b]bb = b-of tv using subtype-eq-base2[OF infer-v-conspI(5)] b-of.simps t subst-tb.simps
by auto
    thus ‹ Θ; ℬ; Γ ⊢wf v : b1[bv::=b]bb › using infer-v-conspI by auto
  qed
qed

lemma infer-v-t-form-wf:
  assumes  wfB Θ ℬ b and wfV  Θ ℬ Γ v b and atom z ♯ Γ
  shows wfT Θ ℬ Γ ⦃ z : b | C-eq (CE-val (V-var z)) (CE-val v)⦄
  using wfT-v-eq assms by auto

lemma infer-v-t-wf:
  fixes v::v
  assumes Θ; ℬ; G ⊢ v ⇒ τ
  shows wfT Θ ℬ G τ ∧  wfB  Θ ℬ (b-of τ)
proof –
  obtain z and b where τ = ⦃ z : b  | CE-val (V-var z)  ==  CE-val v  ⦄ ∧ atom z ♯ v ∧ atom z ♯
(Θ, ℬ, G) using infer-v-form assms by metis
  moreover have  wfB  Θ ℬ b using infer-v-v-wf b-of.simps wfX-wfB(1) assms
    using calculation by fastforce
  ultimately show wfT Θ ℬ G τ ∧   wfB  Θ ℬ (b-of τ) using infer-v-v-wf  infer-v-t-form-wf assms
by fastforce
qed

lemma infer-v-wf:
```

fixes $v::v$
assumes $\Theta; \mathcal{B}; G \vdash v \Rightarrow \tau$
shows $\Theta; \mathcal{B}; G \vdash_{wf} v : (b\text{-}of\ \tau)$ **and** $wfT\ \Theta\ \mathcal{B}\ G\ \tau$ **and** $wfTh\ \Theta$ **and** $wfG\ \Theta\ \mathcal{B}\ G$
proof −
show $\Theta; \mathcal{B}; G \vdash_{wf} v : b\text{-}of\ \tau$ **using** *infer-v-v-wf assms* **by** *auto*
show $\Theta; \mathcal{B}; G \vdash_{wf} \tau$ **using** *infer-v-t-wf assms* **by** *auto*
thus $\Theta ; \mathcal{B} \vdash_{wf} G$ **using** *wfX-wfY* **by** *auto*
thus $\vdash_{wf} \Theta$ **using** *wfX-wfY* **by** *auto*
qed

lemma *check-bool-options*:
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{\!| z : B\text{-}bool \mid TRUE |\!\}$ **and** $supp\ v = \{\}$
shows $v = V\text{-}lit\ L\text{-}true \vee v = V\text{-}lit\ L\text{-}false$
proof −
obtain *t1* **where** $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim \{\!| z : B\text{-}bool \mid TRUE |\!\} \wedge \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow t1$ **using** *check-v-elims*

using *assms* **by** *blast*
thus *?thesis* **using** *infer-bool-options assms*
by (*metis* τ*.exhaust b-of.simps subtype-eq-base2*)
qed

lemma *check-v-wf*:
fixes $v::v$ **and** $\Gamma::\Gamma$ **and** $\tau::\tau$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$
shows $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$ **and** $\Theta; \mathcal{B};\Gamma \vdash_{wf} v : b\text{-}of\ \tau$ **and** $\Theta; \mathcal{B};\Gamma \vdash_{wf} \tau$
proof −
obtain τ' **where** $*$: $\Theta; \mathcal{B}; \Gamma \vdash \tau' \lesssim \tau \wedge \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau'$ **using** *check-v-elims assms* **by** *auto*
thus $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$ **and** $\Theta; \mathcal{B};\Gamma \vdash_{wf} v : b\text{-}of\ \tau$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$
using *infer-v-wf infer-v-v-wf subtype-eq-base2* $*$ *subtype-wf* **by** *metis+*
qed

lemma *infer-v-form-fresh*:
fixes $v::v$ **and** $t::'a::fs$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$
shows $\exists z\ b.\ \tau = \{\!| z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)|\!\} \wedge atom\ z \sharp (t,v)$
proof −
obtain z' **and** b' **where** $\tau = \{\!| z' : b' \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z'))\ (CE\text{-}val\ v)|\!\}$ **using** *infer-v-form assms* **by** *blast*
moreover then obtain z **and** b **and** c **where** $\tau = \{\!| z : b \mid c |\!\} \wedge atom\ z \sharp (t,v)$ **using** *obtain-fresh-z* **by** *metis*
ultimately have $\tau = \{\!| z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)|\!\} \wedge atom\ z \sharp (t,v)$
using *assms infer-v-form2* **by** *auto*
thus *?thesis* **by** *blast*
qed

More generally, if support of a term is empty then any G will do

lemma *infer-v-form-consp*:
assumes $\Theta; \mathcal{B}; \Gamma \vdash V\text{-}consp\ s\ dc\ b\ v \Rightarrow \tau$
shows $b\text{-}of\ \tau = B\text{-}app\ s\ b$
using *assms* **proof**(*nominal-induct V-consp s dc b v* τ *rule*: *infer-v.strong-induct*)
case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ Γ *tv z*)
then show *?case* **using** *b-of.simps* **by** *metis*

qed

lemma *lookup-in-rig-b*:
  assumes *Some* (*b2*, *c2*) = *lookup* (Γ[*x*⟼*c'*]) *x'* **and**
    Some (*b1*, *c1*) = *lookup* Γ *x'*
  shows *b1* = *b2*
  using *assms lookup-in-rig*[*OF assms*(*2*)]
  by (*metis option.inject prod.inject*)

lemma *infer-v-uniqueness-rig*:
  fixes *x*::*x* **and** *c*::*c*
  assumes *infer-v P B G v τ* **and** *infer-v P B* (*replace-in-g G x c'*) *v τ'*
  shows *τ* = *τ'*
  using *assms*
proof(*nominal-induct v arbitrary*: *τ' τ rule*: *v.strong-induct*)
  case (*V-lit l*)
  hence *infer-l l τ* **and** *infer-l l τ'* **using** *assms*(*1*) *infer-v-elims*(*2*) **by** *auto*
  then show *?case* **using** *infer-l-uniqueness* **by** *presburger*
next
  case (*V-var y*)

  obtain *b* **and** *c* **where** *bc*: *Some* (*b*,*c*) = *lookup G y*
    using *assms*(*1*) *infer-v-elims*(*2*) **using** *V-var.prems*(*1*) *lookup-iff* **by** *force*
  then obtain *c''* **where** *bc'*:*Some* (*b*,*c''*) = *lookup* (*replace-in-g G x c'*) *y*
    using *lookup-in-rig* **by** *blast*

  obtain *z* **where** *τ* = (⦃ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var y*)) ⦄) **using** *infer-v-elims*(*1*)[*of P B G y τ*] *V-var*
      bc option.inject prod.inject lookup-in-g **by** *metis*
  moreover obtain *z'* **where** *τ'* = (⦃ *z'* : *b* | *C-eq* (*CE-val* (*V-var z'*)) (*CE-val* (*V-var y*)) ⦄) **using** *infer-v-elims*(*1*)[*of P B - y τ'*] *V-var*
      option.inject prod.inject lookup-in-rig **by** (*metis bc'*)
  ultimately show *?case* **using** *type-e-eq*
    by (*metis V-var.prems*(*1*) *V-var.prems*(*2*) *τ.eq-iff ce.fresh*(*1*) *finite.emptyI fresh-atom-at-base*
       fresh-finite-insert infer-v-elims(*1*) *v.fresh*(*2*))
next
  case (*V-pair v1 v2*)
  obtain *z* **and** *z1* **and** *z2* **and** *t1* **and** *t2* **and** *c1* **and** *c2* **where**
    t1: *τ* = (⦃ *z* : [*b-of t1* , *b-of t2*]^b | *CE-val* (*V-var z*) == *CE-val* (*V-pair v1 v2*) ⦄) ∧
        atom z ♯ (*v1*, *v2*) ∧ *P* ; *B* ; *G* ⊢ *v1* ⇒ *t1* ∧ *P* ; *B* ; *G* ⊢ *v2* ⇒ *t2*
    using *infer-v-elims*(*3*)[*OF V-pair*(*3*)] **by** *metis*
  moreover obtain *z'* **and** *z1'* **and** *z2'* **and** *t1'* **and** *t2'* **and** *c1'* **and** *c2'* **where**
    t2: *τ'* = (⦃ *z'* : [*b-of t1'* , *b-of t2'*]^b | *CE-val* (*V-var z'*) == *CE-val* (*V-pair v1 v2*) ⦄) ∧
          atom z' ♯ (*v1*, *v2*) ∧ *P* ; *B* ; (*replace-in-g G x c'*) ⊢ *v1* ⇒ *t1'* ∧
          P ; *B* ; (*replace-in-g G x c'*) ⊢ *v2* ⇒ *t2'*
    using *infer-v-elims*(*3*)[*OF V-pair*(*4*)] **by** *metis*
  ultimately have *t1* = *t1'* ∧ *t2* = *t2'* **using** *V-pair.hyps*(*1*) *V-pair.hyps*(*2*) *τ.eq-iff* **by** *blast*
  then show *?case* **using** *t1 t2* **by** *simp*
next
  case (*V-cons s dc v*)
  obtain *x* **and** *z* **and** *tc* **and** *dclist* **where** *t1*: *τ* = (⦃ *z* : *B-id s* | *CE-val* (*V-var z*) == *CE-val* (*V-cons s dc v*) ⦄) ∧

AF-typedef s dclist ∈ *set P* ∧
(*dc*, *tc*) ∈ *set dclist* ∧ *atom z* ♯ *v*
using *infer-v-elims*(*4*)[*OF V-cons*(*2*)] **by** *metis*
moreover obtain *x′* **and** *z′* **and** *tc′* **and** *dclist′* **where** *t2*: *τ′* = (⦃ *z′* : *B-id s* | *CE-val* (*V-var z′*) == *CE-val* (*V-cons s dc v*) ⦄)
∧ *AF-typedef s dclist′* ∈ *set P* ∧ (*dc*, *tc′*) ∈ *set dclist′* ∧ *atom z′* ♯ *v*
using *infer-v-elims*(*4*)[*OF V-cons*(*3*)] **by** *metis*
moreover have *a*: *AF-typedef s dclist′* ∈ *set P* **and** *b*:(*dc*,*tc′*) ∈ *set dclist′* **and** *c*:*AF-typedef s dclist* ∈ *set P* **and**
d:(*dc*, *tc*) ∈ *set dclist* **using** *t1 t2* **by** *auto*
ultimately have *tc* = *tc′* **using** *wfTh-dc-t-unique2 infer-v-wf*(*3*)[*OF V-cons*(*2*)] **by** *metis*

moreover have *atom z* ♯ *CE-val* (*V-cons s dc v*) ∧ *atom z′* ♯ *CE-val* (*V-cons s dc v*)
using *e.fresh*(*1*) *v.fresh*(*4*) *t1 t2 pure-fresh* **by** *auto*
ultimately have (⦃ *z* : *B-id s* | *CE-val* (*V-var z*) == *CE-val* (*V-cons s dc v*) ⦄) = (⦃ *z′* : *B-id s* | *CE-val* (*V-var z′*) == *CE-val* (*V-cons s dc v*) ⦄)
using *type-e-eq* **by** *metis*
thus *?case* **using** *t1 t2* **by** *simp*
next
case (*V-consp s dc b v*)
from *V-consp*(*2*) *V-consp* **show** *?case* **proof**(*nominal-induct V-consp s dc b v τ arbitrary*: *v rule*:*infer-v.strong-induct*)

case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ Γ *v tv z*)

obtain *z3* **and** *b3* **where** ∗:*τ′* = ⦃ *z3* : *b3* | [[*z3*]^{v}]^{ce} == [*V-consp s dc b v*]^{ce} ⦄ ∧ *atom z3* ♯ *V-consp s dc b v*
using *infer-v-form*[*OF* ‹Θ; $\mathcal{B}$; Γ[*x*⟼*c′*] ⊢ *V-consp s dc b v* ⇒ *τ′*›] **by** *metis*
moreover then have *b3* = *B-app s b* **using** *infer-v-form-consp b-of.simps* ∗ *infer-v-conspI* **by** *metis*

moreover have ⦃ *z3* : *B-app s b* | [[*z3*]^{v}]^{ce} == [*V-consp s dc b v*]^{ce} ⦄ = ⦃ *z* : *B-app s b* | [[*z*]^{v}]^{ce} == [*V-consp s dc b v*]^{ce} ⦄
proof −
have *atom z3* ♯ [*V-consp s dc b v*]^{ce} **using** ∗ *ce.fresh* **by** *auto*
moreover have *atom z* ♯ [*V-consp s dc b v*]^{ce} **using** ∗ *infer-v-conspI ce.fresh v.fresh pure-fresh* **by** *metis*
ultimately show *?thesis* **using** *type-e-eq infer-v-conspI v.fresh ce.fresh* **by** *metis*
qed
ultimately show *?case* **using** ∗ **by** *auto*
qed
qed

lemma *infer-v-uniqueness*:
assumes *infer-v P B G v τ* **and** *infer-v P B G v τ′*
shows *τ* = *τ′*
proof −
obtain *x*::*x* **where** *atom x* ♯ *G* **using** *obtain-fresh* **by** *metis*
hence *G* [*x* ⟼ *C-true*] = *G* **using** *replace-in-g-forget assms infer-v-wf* **by** *fast*
thus *?thesis* **using** *infer-v-uniqueness-rig assms* **by** *metis*
qed

lemma *infer-v-tid-form*:

fixes *v*::*v*
assumes Θ ; *B* ; Γ $\vdash$ *v* $\Rightarrow$ {| *z* : *B-id tid* | *c* |} **and** *AF-typedef tid dclist* $\in$ *set* Θ **and** *supp v* = {}
shows $\exists$ *dc v' t*. *v* = *V-cons tid dc v'* $\wedge$ (*dc* , *t*) $\in$ *set dclist*
using *assms* **proof**(*nominal-induct v* {| *z* : *B-id tid* | *c* |} *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ *c x z*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next
case (*infer-v-litI* Θ $\mathcal{B}$ *l*)
then show *?case* **by** *auto*
next
case (*infer-v-consI dclist1* Θ *dc tc* $\mathcal{B}$ Γ *v tv z*)
hence *supp v* = {} **using** *v.supp* **by** *simp*
then obtain *dca* **and** *v'* **where** $*$: *V-cons tid dc v* = *V-cons tid dca v'* **using** *infer-v-consI* **by** *auto*
hence *dca* = *dc* **using** *v.eq-iff*(*4*) **by** *auto*
hence *V-cons tid dc v* = *V-cons tid dca v'* $\wedge$ (*dca*, *tc*) $\in$ *set dclist1* **using** *infer-v-consI* $*$ **by** *auto*
moreover have *dclist* = *dclist1* **using** *wfTh-dclist-unique infer-v-consI wfX-wfY* ‹*dca=dc*›
proof −
show *?thesis*
by (*meson* ‹*AF-typedef tid dclist1* $\in$ *set* Θ› ‹Θ; $\mathcal{B}$; Γ $\vdash$ *v* $\Rightarrow$ *tv*› *infer-v-consI.prems infer-v-wf*(*4*) *wfTh-dclist-unique wfX-wfY*)
qed
ultimately show *?case* **by** *auto*
qed

lemma *check-v-tid-form*:
assumes Θ ; *B* ; Γ $\vdash$ *v* $\Leftarrow$ {| *z* : *B-id tid* | *TRUE* |} **and** *AF-typedef tid dclist* $\in$ *set* Θ **and** *supp v* = {}
shows $\exists$ *dc v' t*. *v* = *V-cons tid dc v'* $\wedge$ (*dc* , *t*) $\in$ *set dclist*
using *assms* **proof**(*nominal-induct v* {| *z* : *B-id tid* | *TRUE* |} *rule*: *check-v.strong-induct*)
case (*check-v-subtypeI* Θ $\mathcal{B}$ Γ τ*1 v*)
then obtain *z* **and** *c* **where** τ*1* = {| *z* : *B-id tid* | *c* |} **using** *subtype-eq-base2 b-of.simps*
by (*metis obtain-fresh-z2*)
then show *?case* **using** *infer-v-tid-form check-v-subtypeI* **by** *simp*
qed

lemma *check-v-num-leq*:
fixes *n*::*int* **and** Γ::Γ
assumes *0* $\leq$ *n* $\wedge$ *n* $\leq$ *int* (*length v*) **and** $\vdash_{wf}$ Θ **and** Θ ; {||} $\vdash_{wf}$ Γ
shows Θ ; {||} ; Γ $\vdash$ [*L-num n*]^v $\Leftarrow$ {| *z* : *B-int* | ([*leq* [[*L-num 0*]^v]^{ce} [[*z*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce})
AND ([*leq* [[*z*]^v]^{ce} [| [[*L-bitvec v*]^v]^{ce} |]^{ce}]^{ce} == [[*L-true*]^v]^{ce}) |}
proof −
have Θ ; {||} ; Γ $\vdash$ [*L-num n*]^v $\Rightarrow$ {| *z* : *B-int* | [[*z*]^v]^{ce} == [[*L-num n*]^v]^{ce} |}
using *infer-v-litI infer-natI wfG-nilI assms* **by** *auto*
thus *?thesis* **using** *subtype-range*[*OF assms*(*1*)] *assms check-v-subtypeI* **by** *metis*
qed

lemma *check-int*:
assumes *check-v* Θ $\mathcal{B}$ Γ *v* ({| *z* : *B-int* | *c* |}) **and** *supp v* = {}
shows $\exists$ *n*. *V-lit* (*L-num n*) = *v*
using *assms infer-int check-v-elims* **by** (*metis b-of.simps infer-v-form subtype-eq-base2*)

definition *sble* :: $\Theta \Rightarrow \Gamma \Rightarrow$ *bool* **where**
sble $\Theta\ \Gamma = (\exists i.\ i \models \Gamma \wedge \Theta\ ;\ \Gamma \vdash i)$

lemma *check-v-range*:
assumes $\Theta\ ;\ B\ ;\ \Gamma \vdash v2 \Leftarrow \{\!|\ z : B\text{-}int\ \ |\ [\ leq\ [\ [\ L\text{-}num\ 0\ ]^{v}\ ]^{ce}\ [\ [\ z\ ]^{v}\ ]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^{v}\ ]^{ce}$ *AND*
$[\ leq\ [\ [\ z\ ]^{v}\ ]^{ce}\ [|\ [\ v1\ ]^{ce}\ |]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^{v}\ ]^{ce}\ \ |\!\}$
(**is** $\Theta\ ;\ ?B\ ;\ \Gamma \vdash v2 \Leftarrow \{\!|\ z : B\text{-}int\ |\ ?c1\ |\!\}$)
and $v1 = V\text{-}lit\ (L\text{-}bitvec\ bv) \wedge v2 = V\text{-}lit\ (L\text{-}num\ n)$ **and** *atom* $z \sharp \Gamma$ **and** *sble* $\Theta\ \Gamma$
shows $0 \leq n \wedge n \leq int\ (length\ bv)$
proof −
have $\Theta\ ;\ ?B\ ;\ \Gamma \vdash\ \{\!|\ \ z : B\text{-}int\ \ |\ \ [\ [\ z\ ]^{v}\ ]^{ce} == [\ [\ L\text{-}num\ n\ ]^{v}\ ]^{ce}\ |\!\} \lesssim \{\!|\ z : B\text{-}int\ |\ ?c1\ |\!\}$
using *check-v-elims assms*
by (*metis infer-l-uniqueness infer-natI infer-v-elims*(*2*))
moreover have *atom* $z \sharp \Gamma$ **using** *fresh-GNil assms* **by** *simp*
ultimately have $\Theta\ ;\ ?B\ ;\ ((z,\ B\text{-}int,\ [\ [\ z\ ]^{v}\ ]^{ce}\ ==\ [\ [\ L\text{-}num\ n\ ]^{v}\ ]^{ce}\ ) \#_{\Gamma}\ \Gamma)\ \models\ ?c1$
using *subtype-valid-simple* **by** *auto*
thus *?thesis* **using** *assms valid-range-length-inv check-v-wf wfX-wfY sble-def* **by** *metis*
qed

12.5 Expressions

lemma *infer-e-plus*[*elim*]:
fixes *v1*::*v* **and** *v2*::*v*
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}op\ Plus\ v1\ v2 \Rightarrow \tau$
shows $\exists z\ .\ (\{\!|\ z : B\text{-}int\ |\ \ C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce})\ |\!\} = \tau)$
using *infer-e-elims assms* **by** *metis*

lemma *infer-e-leq*[*elim*]:
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}op\ LEq\ v1\ v2 \Rightarrow \tau$
shows $\exists z\ .\ (\{\!|\ z : B\text{-}bool\ |\ \ C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}op\ LEq\ [v1]^{ce}\ [v2]^{ce})\ |\!\} = \tau)$
using *infer-e-elims assms* **by** *metis*

lemma *infer-e-eq*[*elim*]:
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}op\ Eq\ v1\ v2 \Rightarrow \tau$
shows $\exists z\ .\ (\{\!|\ z : B\text{-}bool\ |\ \ C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}op\ Eq\ [v1]^{ce}\ [v2]^{ce})\ |\!\} = \tau)$
using *infer-e-elims*(*25*)[*OF assms*] **by** *metis*

lemma *infer-e-e-wf*:
fixes *e*::*e*
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash e \Rightarrow \tau$
shows $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash_{wf} e : b\text{-}of\ \tau$
using *assms* **proof**(*nominal-induct* τ *avoiding*: τ *rule*: *infer-e.strong-induct*)
case (*infer-e-valI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'\ \Phi\ v\ \tau$)
then show *?case* **using** *infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-plusI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'\ \Phi$ *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-leqI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'$ *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next

case (*infer-e-eqI* Θ $\mathcal{B}$ Γ Δ' *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ *f x b c* τ' s' *v* τ'')
have Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-app f v* : *b-of* τ' **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *infer-e-appI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ' s'))) = *lookup-fun* Φ *f*› **using** *infer-e-appI* **by** *auto*
show Θ; $\mathcal{B}$;Γ $\vdash_{wf}$ *v* : *b* **using** *infer-e-appI check-v-wf b-of.simps* **by** *metis*
qed
moreover have *b-of* τ' = *b-of* ($\tau'[x::=v]_v$) **using** *subst-tbase-eq subst-v-τ-def* **by** *auto*
ultimately show *?case* **using** *infer-e-appI subst-v-c-def subst-b-τ-def* **by** *auto*
next
case (*infer-e-appPI* Θ $\mathcal{B}$ Γ Δ Φ b' *f bv x b c* τ'' s' *v* τ')

have Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-appP f* b' *v* : (*b-of* τ'')$[bv::=b']_b$ **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *infer-e-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ'' s'))) = *lookup-fun* Φ *f*› **using** * *infer-e-appPI* **by** *metis*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ b' **using** *infer-e-appPI* **by** *auto*
show Θ; $\mathcal{B}$;Γ $\vdash_{wf}$ *v* : ($b[bv::=b']_b$) **using** *infer-e-appPI check-v-wf b-of.simps subst-b-b-def* **by** *metis*
have *atom bv* ♯ (*b-of* τ'')$[bv::=b']_{bb}$ **using** *fresh-subst-if subst-b-b-def infer-e-appPI* **by** *metis*
thus *atom bv* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, b', *v*, (*b-of* τ'')$[bv::=b']_b$) **using** *infer-e-appPI fresh-prodN subst-b-b-def* **by** *metis*
qed
moreover have *b-of* τ' = (*b-of* τ'')$[bv::=b']_b$
using ‹$\tau''[bv::=b']_b[x::=v]_v$ = τ'› *b-of-subst-bb-commute subst-tbase-eq subst-b-b-def subst-v-τ-def subst-b-τ-def* **by** *auto*
ultimately show *?case* **using** *infer-e-appI* **by** *auto*
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ' Φ *v* z' *b1 b2 c z*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ' Φ *v* z' *b1 b2 c z*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ Δ' Φ *v* z' *c z*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-mvarI* Θ Γ Φ Δ *u* τ)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-concatI* Θ $\mathcal{B}$ Γ Δ' Φ *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-splitI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
have Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-split v1 v2* : *B-pair B-bitvec B-bitvec*
proof
show Θ $\vdash_{wf}$ Φ **using** *infer-e-splitI* **by** *auto*
show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ **using** *infer-e-splitI* **by** *auto*

show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v1* : *B-bitvec* **using** *infer-e-splitI b-of.simps infer-v-wf* **by** *metis*
show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v2* : *B-int* **using** *infer-e-splitI b-of.simps check-v-wf* **by** *metis*
qed
then show *?case* **using** *b-of.simps* **by** *auto*
qed

lemma *infer-e-t-wf*:
fixes *e*::*e* **and** Γ::Γ **and** τ::τ **and** Δ::Δ **and** Φ::Φ
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *e* ⇒ τ
shows Θ; $\mathcal{B}$;Γ $\vdash_{wf}$ τ ∧ Θ $\vdash_{wf}$ Φ
using *assms* **proof**(*induct rule*: *infer-e.induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ Δ′ Φ *v* τ)
then show *?case* **using** *infer-v-t-wf* **by** *auto*
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$: *B-int* **using** *wfCE-plusI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-plusI* **by** *auto*
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$: *B-bool* **using** *wfCE-leqI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-leqI* **by** *auto*
next
case (*infer-e-eqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 b c1 v2 z2 c2 z3*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$: *B-bool* **using** *wfCE-eqI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-eqI* **by** *auto*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ *f x b c* τ *s′ v* τ′)
show *?case* **proof**
show Θ $\vdash_{wf}$ Φ **using** *infer-e-appI* **by** *auto*
show Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ′ **proof** −
have ∗: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* **using** *infer-e-appI check-v-wf*(*2*) *b-of.simps* **by** *metis*
moreover have ∗:Θ; $\mathcal{B}$; (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ τ **proof**(*rule wf-weakening1*(*4*))
show ‹ Θ; $\mathcal{B}$; (*x*,*b*,*c*)$\#_\Gamma$*GNil* $\vdash_{wf}$ τ › **using** *wfPhi-f-simple-wfT wfD-wf infer-e-appI wb-b-weakening* **by** *fastforce*
have Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *x* : *b* | *c* ⦄ **using** *infer-e-appI check-v-wf*(*3*) **by** *auto*
thus ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*) $\#_\Gamma$ Γ › **using** *infer-e-appI*
wfT-wfC[*THEN wfG-consI*[*rotated 3*]] ∗ *wfT-wf-cons fresh-prodN* **by** *simp*
show ‹*toSet* ((*x*,*b*,*c*)$\#_\Gamma$*GNil*) ⊆ *toSet* ((*x*, *b*, *c*) $\#_\Gamma$ Γ)› **using** *toSet.simps* **by** *auto*
qed
moreover have ((*x*, *b*, *c*) $\#_\Gamma$ Γ)[*x*::=*v*]$_{\Gamma v}$ = Γ **using** *subst-gv.simps* **by** *auto*

ultimately show *?thesis* **using** *infer-e-appI wf-subst1*(*4*)[*OF* ∗, *of GNil x b c* Γ *v*] *subst-v-τ-def* **by** *auto*
qed
qed
next

case (*infer-e-appPI* Θ $\mathcal{B}$ Γ Δ Φ *b' f bv x b c* τ' *s' v* τ)

have Θ; $\mathcal{B}$; ((*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ)[*x*::=*v*]$_{\Gamma v}$ $\vdash_{wf}$ (τ'[*bv*::=*b'*]$_b$)[*x*::=*v*]$_{\tau v}$
proof(*rule wf-subst*(*4*))
show ‹ Θ; $\mathcal{B}$; (*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ $\vdash_{wf}$ τ'[*bv*::=*b'*]$_b$ ›
proof(*rule wf-weakening1*(*4*))
have ‹ Θ ; {|*bv*|} ; (*x*,*b*,*c*)$\#_\Gamma$*GNil* $\vdash_{wf}$ τ' › **using** *wfPhi-f-poly-wfT infer-e-appI infer-e-appPI* **by** *simp*
thus ‹ Θ; $\mathcal{B}$; (*x*,*b*[*bv*::=*b'*]$_{bb}$,*c*[*bv*::=*b'*]$_{cb}$)$\#_\Gamma$*GNil* $\vdash_{wf}$ τ'[*bv*::=*b'*]$_b$ ›
using *wfT-subst-wfT infer-e-appPI wb-b-weakening subst-b-τ-def subst-v-τ-def* **by** *presburger*
have Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *x* : *b*[*bv*::=*b'*]$_{bb}$ | *c*[*bv*::=*b'*]$_{cb}$ ⦄
using *infer-e-appPI check-v-wf*(*3*) *subst-b-b-def subst-b-c-def* **by** *metis*
thus ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ ›
using *infer-e-appPI wfT-wfC*[*THEN wfG-consI*[*rotated 3*]] ∗ *wfX-wfY wfT-wf-cons wb-b-weakening* **by** *metis*
show ‹*toSet* ((*x*,*b*[*bv*::=*b'*]$_{bb}$,*c*[*bv*::=*b'*]$_{cb}$)$\#_\Gamma$*GNil*) ⊆ *toSet* ((*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ)› **using** *toSet.simps* **by** *auto*
qed
show ‹(*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ = *GNil* @ (*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ› **using** *append-g.simps* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* :*b*[*bv*::=*b'*]$_{bb}$ › **using** *infer-e-appPI check-v-wf*(*2*) *b-of.simps subst-b-b-def* **by** *metis*
qed
moreover have ((*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) $\#_\Gamma$ Γ)[*x*::=*v*]$_{\Gamma v}$ = Γ **using** *subst-gv.simps* **by** *auto*
ultimately show *?case* **using** *infer-e-appPI subst-v-τ-def* **by** *simp*
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ Φ *v z' b1 b2 c z*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-fst* [*v*]^{ce}: *b1* **using** *wfCE-fstI wfD-emptyI wfPhi-emptyI infer-v-v-wf b-of.simps* **using** *wfCE-valI* **by** *fastforce*
then show *?case* **using** *wfT-e-eq infer-e-fstI* **by** *auto*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ Φ *v z' b1 b2 c z*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-snd* [*v*]^{ce}: *b2* **using** *wfCE-sndI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-sndI* **by** *auto*
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ Δ Φ *v z' c z*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-len* [*v*]^{ce}: *B-int* **using** *wfCE-lenI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-lenI* **by** *auto*
next
case (*infer-e-mvarI* Θ Γ Φ Δ *u* τ)
then show *?case* **using** *wfD-wfT* **by** *blast*
next
case (*infer-e-concatI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-concat* [*v1*]^{ce} [*v2*]^{ce}: *B-bitvec* **using** *wfCE-concatI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-concatI* **by** *auto*
next

case (*infer-e-splitI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 z3*)

hence *wfg*: Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*z3*, [*B-bitvec* , *B-bitvec* $]^b$, *TRUE*) $\#_\Gamma$ Γ
using *infer-v-wf wfG-cons2I wfB-pairI wfB-bitvecI* **by** *simp*
have *wfz*: Θ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec* $]^b$, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [[*z3* $]^v$ $]^{ce}$: [*B-bitvec* , *B-bitvec* $]^b$
apply(*rule wfCE-valI* , *rule wfV-varI*)
using *wfg* **apply** *simp*
using *lookup.simps*(*2*)[*of z3* [*B-bitvec* , *B-bitvec* $]^b$ *TRUE* Γ *z3*] **by** *simp*
have *1*: Θ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec* $]^b$, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [*v2* $]^{ce}$: *B-int*
using *check-v-wf*[*OF infer-e-splitI*(*4*)] *wf-weakening*(*1*)[*OF - wfg*] *b-of.simps toSet.simps wfCE-valI* **by** *fastforce*
have *2*: Θ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec* $]^b$, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [*v1* $]^{ce}$: *B-bitvec*
using *infer-v-wf*[*OF infer-e-splitI*(*3*)] *wf-weakening*(*1*)[*OF - wfg*] *b-of.simps toSet.simps wfCE-valI* **by** *fastforce*

have Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ ⦃ *z3* : [*B-bitvec* , *B-bitvec* $]^b$ | [*v1* $]^{ce}$ == [[#*1*[[*z3* $]^v$ $]^{ce}]^{ce}$ @@ [#*2*[[*z3* $]^v$ $]^{ce}]^{ce}$ $]^{ce}$ *AND* [| [#*1*[[*z3* $]^v$ $]^{ce}]^{ce}$ |$]^{ce}$ == [*v2* $]^{ce}$ ⦄
proof
show *atom z3* ♯ (Θ, $\mathcal{B}$, Γ) **using** *infer-e-splitI wfTh-x-fresh wfX-wfY fresh-prod3 wfG-fresh-x* **by** *metis*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ [*B-bitvec* , *B-bitvec* $]^b$ **using** *wfB-pairI wfB-bitvecI infer-e-splitI wfX-wfY* **by** *metis*
show Θ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec* $]^b$, *TRUE*) $\#_\Gamma$
Γ $\vdash_{wf}$ [*v1* $]^{ce}$ == [[#*1*[[*z3* $]^v$ $]^{ce}]^{ce}$ @@ [#*2*[[*z3* $]^v$ $]^{ce}]^{ce}$ $]^{ce}$ *AND* [| [#*1*[[*z3* $]^v$ $]^{ce}]^{ce}$ |$]^{ce}$ == [*v2* $]^{ce}$
using *wfg wfz 1 2 wf-intros* **by** *meson*
qed
thus *?case* **using** *infer-e-splitI* **by** *auto*
qed

lemma *infer-e-wf*:
fixes *e*::*e* **and** Γ::Γ **and** *τ*::*τ* **and** Δ::Δ **and** Φ::Φ
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *e* ⇒ *τ*
shows Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *τ* **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ **and** Θ $\vdash_{wf}$ Φ **and** Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : (*b-of τ*)
using *infer-e-t-wf infer-e-e-wf wfE-wf assms* **by** *metis+*

lemma *infer-e-fresh*:
fixes *x*::*x*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *e* ⇒ *τ* **and** *atom x* ♯ Γ
shows *atom x* ♯ (*e*,*τ*)
proof −
have *atom x* ♯ *e* **using** *infer-e-e-wf*[*THEN wfE-x-fresh*,*OF assms*(*1*)] *assms*(*2*) **by** *auto*
moreover have *atom x* ♯ *τ* **using** *assms infer-e-wf wfT-x-fresh* **by** *metis*
ultimately show *?thesis* **using** *fresh-Pair* **by** *auto*
qed

inductive *check-e* :: Θ ⇒ Φ ⇒ $\mathcal{B}$ ⇒ Γ ⇒ Δ ⇒ *e* ⇒ *τ* ⇒ *bool* (‹ - ; - ; - ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*] *50*) **where**
check-e-subtypeI: ⟦ *infer-e T P B G D e τ'* ; *subtype T B G τ' τ* ⟧ ⟹ *check-e T P B G D e τ*
equivariance *check-e*
nominal-inductive *check-e* .

inductive-cases *check-e-elims*[*elim*!]:
check-e F D B G Θ (*AE-val v*) τ
check-e F D B G Θ *e* τ

lemma *infer-e-fst-pair*:
fixes *v1*::*v*
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ [#*1*[*v1* , *v2*]^v]^e ⇒ τ
shows ∃τ′. Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ [*v1*]^e ⇒ τ′ ∧
Θ ; {||} ; *GNil* ⊢ τ′ ≲ τ
proof −
obtain *z*′ **and** *b1* **and** *b2* **and** *c* **and** *z* **where** ∗∗ : τ = ({| *z* : *b1* | *C-eq* (*CE-val* (*V-var z*)) (*CE-fst* [(*V-pair v1 v2*)]^{ce}) |}) ∧
wfD Θ {||} *GNil* Δ ∧ *wfPhi* Θ Φ ∧
(Θ ; {||} ; *GNil* ⊢ *V-pair v1 v2* ⇒ {| *z*′ : *B-pair b1 b2* | *c* |}) ∧ *atom z* ♯ *V-pair v1 v2*
using *infer-e-elims assms* **by** *metis*
hence ∗: Θ ; {||} ; *GNil* ⊢ *V-pair v1 v2* ⇒ {| *z*′ : *B-pair b1 b2* | *c* |} **by** *auto*

obtain *t1a* **and** *t2a* **where**
∗: Θ ; {||} ; *GNil* ⊢ *v1* ⇒ *t1a* ∧ Θ ; {||} ; *GNil* ⊢ *v2* ⇒ *t2a* ∧ *B-pair b1 b2* = *B-pair* (*b-of t1a*) (*b-of t2a*)
using *infer-v-elims*(*5*)[*OF* ∗] **by** *metis*

hence *suppv*: *supp v1* = {} ∧ *supp v2* = {} ∧ *supp* (*V-pair v1 v2*) = {} **using** ∗∗ *infer-v-v-wf wfV-supp atom-dom.simps toSet.simps supp-GNil*
by (*meson wfV-supp-nil*)

obtain *z1* **and** *b1*′ **where** *t1a* = {| *z1* : *b1*′ | [[*z1*]^v]^{ce} == [*v1*]^{ce} |}
using *infer-v-form*[*of* Θ {||} *GNil v1 t1a*] ∗ **by** *auto*
moreover hence *b1*′ = *b1* **using** ∗ *b-of.simps* **by** *simp*
ultimately have Θ ; {||} ; *GNil* ⊢ *v1* ⇒ {| *z1* : *b1* | *CE-val* (*V-var z1*) == *CE-val v1* |} **using** ∗ **by** *auto*
moreover have Θ ; {||} ; *GNil* $\vdash_{wf}$ *CE-fst* [*V-pair v1 v2*]^{ce} : *b1* **using** *wfCE-fstI infer-v-wf*(*1*) ∗∗ *b-of.simps wfCE-valI* **by** *metis*
moreover hence *st*: Θ ; {||} ; *GNil* ⊢ {| *z1* : *b1* | *CE-val* (*V-var z1*) == *CE-val v1* |} ≲ ({| *z* : *b1* | *CE-val* (*V-var z*) == *CE-fst* [*V-pair v1 v2*]^{ce} |})
using *subtype-gnil-fst infer-v-v-wf* **by** *auto*
moreover have *wfD* Θ {||} *GNil* Δ ∧ *wfPhi* Θ Φ **using** ∗∗ **by** *auto*
ultimately show *?thesis* **using** *wfX-wfY* ∗∗ *infer-e-valI* **by** *metis*
qed

lemma *infer-e-snd-pair*:
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-snd* (*V-pair v1 v2*) ⇒ τ
shows ∃τ′. Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val v2* ⇒ τ′ ∧ Θ ; {||} ; *GNil* ⊢ τ′ ≲ τ
proof −
obtain *z*′ **and** *b1* **and** *b2* **and** *c* **and** *z* **where** ∗∗ : (τ = ({| *z* : *b2* | *C-eq* (*CE-val* (*V-var z*)) (*CE-snd* [(*V-pair v1 v2*)]^{ce}) |})) ∧
(*wfD* Θ {||} *GNil* Δ) ∧ (*wfPhi* Θ Φ) ∧
Θ ; {||} ; *GNil* ⊢ *V-pair v1 v2* ⇒ {| *z*′ : *B-pair b1 b2* | *c* |} ∧ *atom z* ♯ *V-pair v1 v2*
using *infer-e-elims*(*9*)[*OF assms*(*1*)] **by** *metis*
hence ∗: Θ ; {||} ; *GNil* ⊢ *V-pair v1 v2* ⇒ {| *z*′ : *B-pair b1 b2* | *c* |} **by** *auto*

obtain *t1a* **and** *t2a* **where**
$*$: Θ ; {||} ; *GNil* ⊢ *v1* ⇒ *t1a* ∧ Θ ; {||} ; *GNil* ⊢ *v2* ⇒ *t2a* ∧ *B-pair b1 b2* = *B-pair* (*b-of t1a*) (*b-of t2a*)
using *infer-v-elims*(*5*)[*OF* $*$] **by** *metis*

hence *suppv*: *supp v1* = {} ∧ *supp v2* = {} ∧ *supp* (*V-pair v1 v2*) = {} **using** *infer-v-v-wf wfV.simps v.supp* **by** (*meson* $**$ *wfV-supp-nil*)

obtain *z2* **and** *b2'* **where** *t2a* = ⦃ *z2* : *b2'* | [[*z2*]^v]^{ce} == [*v2*]^{ce} ⦄
using *infer-v-form*[*of* Θ {||} *GNil v2 t2a*] $*$ **by** *auto*
moreover hence *b2'* = *b2* **using** $*$ *b-of.simps* **by** *simp*

ultimately have Θ ; {||} ; *GNil* ⊢ *v2* ⇒ ⦃ *z2* : *b2* | *CE-val* (*V-var z2*) == *CE-val v2* ⦄ **using** $*$ **by** *auto*
moreover have Θ ; {||} ; *GNil* $\vdash_{wf}$ *CE-snd* [*V-pair v1 v2*]^{ce} : *b2* **using** *wfCE-sndI infer-v-wf*(*1*) $**$ *b-of.simps wfCE-valI* **by** *metis*
moreover hence *st*: Θ ; {||} ; *GNil* ⊢ ⦃ *z2* : *b2* | *CE-val* (*V-var z2*) == *CE-val v2* ⦄ ≲ (⦃ *z* : *b2* | *CE-val* (*V-var z*) == *CE-snd* [*V-pair v1 v2*]^{ce} ⦄)
using *subtype-gnil-snd infer-v-v-wf* **by** *auto*
moreover have *wfD* Θ {||} *GNil* Δ ∧ *wfPhi* Θ Φ **using** $**$ **by** *metis*
ultimately show *?thesis* **using** *wfX-wfY* $**$ *infer-e-valI* **by** *metis*
qed

12.6 Statements

lemma *check-s-v-unit*:
assumes Θ; ℬ; Γ ⊢ (⦃ *z* : *B-unit* | *TRUE* ⦄) ≲ τ **and** *wfD* Θ ℬ Γ Δ **and** *wfPhi* Θ Φ
shows Θ ; Φ ; ℬ ; Γ ; Δ ⊢ *AS-val* (*V-lit L-unit*) ⇐ τ
proof −
have *wfG* Θ ℬ Γ **using** *assms subtype-g-wf* **by** *meson*
moreover hence *wfTh* Θ **using** *wfG-wf* **by** *simp*
moreover obtain *z'*::*x* **where** *atom z'* ♯ Γ **using** *obtain-fresh* **by** *auto*
ultimately have $*$:Θ; ℬ; Γ ⊢ *V-lit L-unit* ⇒ ⦃ *z'* : *B-unit* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit L-unit*) ⦄
using *infer-v-litI infer-unitI* **by** *simp*
moreover have *wfT* Θ ℬ Γ (⦃ *z'* : *B-unit* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit L-unit*) ⦄) **using** *infer-v-t-wf*
by (*meson calculation*)
moreover then have Θ; ℬ; Γ ⊢ (⦃ *z'* : *B-unit* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit L-unit*) ⦄) ≲ τ **using** *subtype-trans subtype-top assms*
type-for-lit.simps(*4*) *wfX-wfY* **by** *metis*
ultimately show *?thesis* **using** *check-valI assms* $*$ **by** *auto*
qed

lemma *check-s-check-branch-s-wf*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** Θ::Θ **and** Φ::Φ **and** Γ::Γ **and** Δ::Δ **and** *v*::*v* **and** τ::τ **and** *css*::*branch-list*
shows Θ ; Φ ; *B* ; Γ ; Δ ⊢ *s* ⇐ τ ⟹ Θ ; *B* $\vdash_{wf}$ Γ ∧ *wfTh* Θ ∧ *wfD* Θ *B* Γ Δ ∧ *wfT* Θ *B* Γ τ ∧ *wfPhi* Θ Φ **and**
check-branch-s Θ Φ *B* Γ Δ *tid cons const v cs* τ ⟹ Θ ; *B* $\vdash_{wf}$ Γ ∧ *wfTh* Θ ∧ *wfD* Θ *B* Γ Δ ∧ *wfT* Θ *B* Γ τ ∧ *wfPhi* Θ Φ
check-branch-list Θ Φ *B* Γ Δ *tid dclist v css* τ ⟹ Θ ; *B* $\vdash_{wf}$ Γ ∧ *wfTh* Θ ∧ *wfD* Θ *B* Γ Δ ∧ *wfT* Θ *B* Γ τ ∧ *wfPhi* Θ Φ

proof(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Θ *B* Γ Δ Φ *v* τ′ τ)
then show *?case* **using** *infer-v-wf infer-v-wf subtype-wf wfX-wfY wfS-valI*
by (*metis subtype-eq-base2*)
next
case (*check-letI x* Θ Φ ℬ Γ Δ *e* τ *z s b c*)
then have ∗:*wfT* Θ ℬ ((*x*, *b* , *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ) τ **by** *force*
moreover have *atom x* ♯ τ **using** *check-letI fresh-prodN* **by** *force*
ultimately have Θ; ℬ;Γ $\vdash_{wf}$ τ **using** *wfT-restrict2* **by** *force*
then show *?case* **using** *check-letI infer-e-wf wfS-letI wfX-wfY wfG-elims* **by** *metis*
next
case (*check-assertI x* Θ Φ ℬ Γ Δ *c* τ *s*)
then have ∗:*wfT* Θ ℬ ((*x*, *B-bool* , *c*) $\#_\Gamma$ Γ) τ **by** *force*
moreover have *atom x* ♯ τ **using** *check-assertI fresh-prodN* **by** *force*
ultimately have Θ; ℬ;Γ $\vdash_{wf}$ τ **using** *wfT-restrict2* **by** *force*
then show *?case* **using** *check-assertI wfS-assertI wfX-wfY wfG-elims* **by** *metis*
next
case (*check-branch-s-branchI* Θ ℬ Γ Δ τ *cons const x v* Φ *s tid*)
then show *?case* **using** *wfX-wfY* **by** *metis*
next
case (*check-branch-list-consI* Θ Φ ℬ Γ Δ *tid dclist′ v cs* τ *css*)
then show *?case* **using** *wfX-wfY* **by** *metis*
next
case (*check-branch-list-finalI* Θ Φ ℬ Γ Δ *tid dclist′ v cs* τ)
then show *?case* **using** *wfX-wfY* **by** *metis*
next
case (*check-ifI z* Θ Φ ℬ Γ Δ *v s1 s2* τ)
hence ∗:*wfT* Θ ℬ Γ (⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP c-of* τ *z* ⦄) (**is** *wfT* Θ ℬ Γ *?tau*) **by** *auto*
hence *wfT* Θ ℬ Γ τ **using** *wfT-wfT-if1 check-ifI fresh-prodN* **by** *metis*
hence Θ; ℬ; Γ $\vdash_{wf}$ τ **using** *check-ifI b-of-c-of-eq fresh-prodN* **by** *auto*
thus *?case* **using** *check-ifI* **by** *metis*
next
case (*check-let2I x* Θ Φ ℬ *G* Δ *t s1* τ *s2*)
then have *wfT* Θ ℬ ((*x*, *b-of t*, (*c-of t x*)) $\#_\Gamma$ *G*) τ **by** *fastforce*
moreover have *atom x* ♯ τ **using** *check-let2I* **by** *force*
ultimately have *wfT* Θ ℬ *G* τ **using** *wfT-restrict2* **by** *metis*
then show *?case* **using** *check-let2I* **by** *argo*
next
case (*check-varI u* Δ *P G v* τ′ Φ *s* τ)
then show *?case* **using** *wfG-elims wfD-elims*
list.distinct list.inject **by** *metis*
next
case (*check-assignI* Θ Φ ℬ Γ Δ *u* τ *v z* τ′)
obtain *z*′::*x* **where** ∗:*atom z*′ ♯ Γ **using** *obtain-fresh* **by** *metis*
moreover have ⦃ *z* : *B-unit* | *TRUE* ⦄ = ⦃ *z*′ : *B-unit* | *TRUE* ⦄ **by** *auto*
moreover hence *wfT* Θ ℬ Γ ⦃ *z*′ : *B-unit* |*TRUE* ⦄ **using** *wfT-TRUE check-assignI check-v-wf* ∗ *wfB-unitI wfG-wf* **by** *metis*
ultimately show *?case* **using** *check-v.cases infer-v-wf subtype-wf check-assignI wfT-wf check-v-wf wfG-wf*
by (*meson subtype-wf*)
next

case (*check-whileI* Φ Δ *G P s1 z s2* τ′)
then show *?case* **using** *subtype-wf subtype-wf* **by** *auto*
next
case (*check-seqI* Δ *G P s1 z s2* τ)
then show *?case* **by** *fast*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *dclist cs* τ *tid v z*)
then show *?case* **by** *fast*
qed

lemma *check-s-check-branch-s-wfS*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** Θ::Θ **and** Φ::Φ **and** Γ::Γ **and** Δ::Δ **and** *v*::*v* **and** τ::τ **and** *css*::*branch-list*
shows Θ ; Φ ; *B* ; Γ ; Δ ⊢ *s* ⇐ τ ⟹ Θ ; Φ ; *B* ; Γ ; Δ $\vdash_{wf}$ *s* : *b-of* τ **and**
check-branch-s Θ Φ *B* Γ Δ *tid cons const v cs* τ ⟹ *wfCS* Θ Φ *B* Γ Δ *tid cons const cs* (*b-of* τ)
check-branch-list Θ Φ *B* Γ Δ *tid dclist v css* τ ⟹ *wfCSS* Θ Φ *B* Γ Δ *tid dclist css* (*b-of* τ)
proof(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ *v* τ′ τ)
then show *?case* **using** *infer-v-wf infer-v-wf subtype-wf wfX-wfY wfS-valI*
by (*metis subtype-eq-base2*)
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b* › **using** *infer-e-wf check-letI b-of.simps* **by** *metis*
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_{\Gamma}$ Γ ; Δ $\vdash_{wf}$ *s* : *b-of* τ ›
using *check-letI b-of.simps wf-replace-true2*(*2*)[*OF check-letI*(*5*)] *append-g.simps* **by** *metis*
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *infer-e-wf check-letI b-of.simps* **by** *metis*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, *e*, *b-of* τ)›
apply(*simp add*: *fresh-prodN*, *intro conjI*)
using *check-letI*(*1*) *fresh-prod7* **by** *simp+*
qed
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
show *?case* **proof**

show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_{\Gamma}$ Γ ; Δ $\vdash_{wf}$ *s* : *b-of* τ › **using** *check-assertI* **by** *auto*
next
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* › **using** *check-assertI* **by** *auto*
next
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *check-assertI* **by** *auto*
next
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, *c*, *b-of* τ, *s*)› **using** *check-assertI* **by** *auto*
qed
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of const*, *TRUE*) $\#_{\Gamma}$ Γ ; Δ $\vdash_{wf}$ *s* : *b-of* τ ›
using *wf-replace-true append-g.simps check-branch-s-branchI* **by** *metis*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, Γ, *const*)›
using *wf-replace-true append-g.simps check-branch-s-branchI fresh-prodN* **by** *metis*
show ‹ Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *wf-replace-true append-g.simps check-branch-s-branchI* **by** *metis*
qed
next

```
  case (check-branch-list-consI Θ Φ ℬ Γ Δ tid cons const v cs τ dclist css)
  then show ?case using wf-intros by metis
next
  case (check-branch-list-finalI Θ Φ ℬ Γ Δ tid cons const v cs τ)
  then show ?case using wf-intros by metis
next
  case (check-ifI z Θ Φ ℬ Γ Δ v s1 s2 τ)
  show ?case using wfS-ifI check-v-wf check-ifI b-of.simps by metis
next
  case (check-let2I x Θ Φ ℬ G Δ t s1 τ s2)
  show ?case proof
    show ‹ Θ ; Φ ; ℬ ; G ; Δ ⊢wf s1 : b-of t › using check-let2I b-of.simps by metis
    show ‹ Θ; ℬ; G ⊢wf t › using check-let2I check-s-check-branch-s-wf by metis
    show ‹ Θ ; Φ ; ℬ ; (x, b-of t, TRUE) #Γ G ; Δ ⊢wf s2 : b-of τ ›
      using check-let2I(5) wf-replace-true2(2) append-g.simps check-let2I by metis
    show ‹atom x ♯ (Φ, Θ, ℬ, G, Δ, s1, b-of τ, t)›
      apply(simp add: fresh-prodN, intro conjI)
      using check-let2I(1) fresh-prod7 by simp+
  qed
next
  case (check-varI u Θ Φ ℬ Γ Δ τ' v τ s)
  show ?case proof
    show ‹ Θ; ℬ; Γ ⊢wf τ' › using check-v-wf check-varI by metis
    show ‹ Θ; ℬ; Γ ⊢wf v : b-of τ' › using check-v-wf check-varI by metis
    show ‹atom u ♯ (Φ, Θ, ℬ, Γ, Δ, τ', v, b-of τ)› using check-varI fresh-prodN u-fresh-b by metis
    show ‹ Θ ; Φ ; ℬ ; Γ ; (u, τ') #Δ Δ ⊢wf s : b-of τ › using check-varI by metis
  qed
next
  case (check-assignI Θ Φ ℬ Γ Δ u τ v z τ')
  then show ?case using wf-intros check-v-wf subtype-eq-base2 b-of.simps by metis
next
  case (check-whileI Θ Φ ℬ Γ Δ s1 z s2 τ')
  thus ?case using wf-intros b-of.simps check-v-wf subtype-eq-base2 b-of.simps by metis
next
  case (check-seqI Θ Φ ℬ Γ Δ s1 z s2 τ)
  thus ?case using wf-intros b-of.simps by metis
next
  case (check-caseI Θ Φ ℬ Γ Δ tid dclist v cs τ z)
  show ?case proof
    show ‹ Θ; ℬ; Γ ⊢wf v : B-id tid › using check-caseI check-v-wf b-of.simps by metis
    show ‹AF-typedef tid dclist ∈ set Θ› using check-caseI by metis
    show ‹ Θ; ℬ; Γ ⊢wf Δ › using check-caseI check-s-check-branch-s-wf by metis
    show ‹ Θ ⊢wf Φ › using check-caseI check-s-check-branch-s-wf by metis
    show ‹ Θ ; Φ ; ℬ ; Γ ; Δ ; tid ; dclist ⊢wf cs : b-of τ › using check-caseI by metis
  qed
qed

lemma check-s-wf:
  fixes s::s
  assumes Θ ; Φ ; B ; Γ ; Δ ⊢ s ⇐ τ
  shows Θ ; B ⊢wf Γ ∧ wfT Θ B Γ τ ∧ wfPhi Θ Φ ∧ wfTh Θ ∧ wfD Θ B Γ Δ ∧ wfS Θ Φ B Γ Δ s
(b-of τ)
```

using *check-s-check-branch-s-wf check-s-check-branch-s-wfS assms* **by** *meson*

lemma *check-s-flip-u1*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
assumes $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau$
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow \tau$
proof −
have $(u \leftrightarrow u') \cdot \Theta ; (u \leftrightarrow u') \cdot \Phi ; (u \leftrightarrow u') \cdot \mathcal{B} ; (u \leftrightarrow u') \cdot \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow (u \leftrightarrow u') \cdot \tau$
using *check-s.eqvt assms* **by** *blast*
thus *?thesis* **using** *check-s-wf*[*OF assms*] *flip-u-eq phi-flip-eq* **by** *metis*
qed

lemma *check-s-flip-u2*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
assumes $\Theta ; \Phi ; B ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow \tau$
shows $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau$
proof −
have $\Theta ; \Phi ; B ; \Gamma ; (u \leftrightarrow u') \cdot (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot (u \leftrightarrow u') \cdot s \Leftarrow \tau$
using *check-s-flip-u1 assms* **by** *blast*
thus *?thesis* **using** *permute-flip-cancel* **by** *force*
qed

lemma *check-s-flip-u*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
shows $\Theta ; \Phi ; B ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \vdash (u \leftrightarrow u') \cdot s \Leftarrow \tau = (\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau)$
using *check-s-flip-u1 check-s-flip-u2* **by** *metis*

lemma *check-s-abs-u*:
fixes $s::s$ **and** $s'::s$ **and** $u::u$ **and** $u'::u$ **and** $\tau'::\tau$
assumes $[[atom\ u]]lst.\ s = [[atom\ u']]lst.\ s'$ **and** $atom\ u \sharp \Delta$ **and** $atom\ u' \sharp \Delta$
and $\Theta ; B ; \Gamma \vdash_{wf} \tau'$
and $\Theta ; \Phi ; B ; \Gamma ; (u , \tau') \#_{\Delta}\Delta \vdash s \Leftarrow \tau$
shows $\Theta ; \Phi ; B ; \Gamma ; (u', \tau') \#_{\Delta}\Delta \vdash s' \Leftarrow \tau$
proof −
have $\Theta ; \Phi ; B ; \Gamma ; (u' \leftrightarrow u) \cdot ((u , \tau') \#_{\Delta}\Delta) \vdash (u' \leftrightarrow u) \cdot s \Leftarrow \tau$
using *assms check-s-flip-u* **by** *metis*
moreover have $(u' \leftrightarrow u) \cdot ((u , \tau') \#_{\Delta} \Delta) = (u' , \tau') \#_{\Delta}\Delta$ **proof** −
have $(u' \leftrightarrow u) \cdot ((u , \tau') \#_{\Delta} \Delta) = ((u' \leftrightarrow u) \cdot u, (u' \leftrightarrow u) \cdot \tau') \#_{\Delta} (u' \leftrightarrow u) \cdot \Delta$
using *DCons-eqvt Pair-eqvt* **by** *auto*
also have ... $= (u' , (u' \leftrightarrow u) \cdot \tau') \#_{\Delta} \Delta$
using *assms flip-fresh-fresh* **by** *auto*
also have ... $= (u' , \tau') \#_{\Delta} \Delta$ **using**
u-not-in-t fresh-def flip-fresh-fresh assms **by** *metis*
finally show *?thesis* **by** *auto*
qed
moreover have $(u' \leftrightarrow u) \cdot s = s'$ **using** *assms Abs1-eq-iff*(*3*)[*of u' s' u s*] **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

12.7 Additional Elimination and Intros

12.7.1 Values

nominal-function *b-for* :: *opp* ⇒ *b* **where**
b-for Plus = *B-int*
| *b-for LEq* = *B-bool* | *b-for Eq* = *B-bool*
apply(*auto*,*simp add*: *eqvt-def b-for-graph-aux-def*)
by (*meson opp.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *infer-v-pair2I*:
fixes v_1::*v* **and** v_2::*v*
assumes Θ; ℬ; Γ ⊢ v_1 ⇒ τ_1 **and** Θ; ℬ; Γ ⊢ v_2 ⇒ τ_2
shows ∃ τ. Θ; ℬ; Γ ⊢ *V-pair* v_1 v_2 ⇒ τ ∧ *b-of* τ = *B-pair* (*b-of* τ_1) (*b-of* τ_2)
proof −
obtain *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where** *zbc*: τ_1 = (⦃ *z1* : *b1* | *c1* ⦄) ∧ τ_2 = (⦃ *z2* : *b2* | *c2* ⦄)
using *τ.exhaust* **by** *meson*
obtain *z*::*x* **where** *atom z* ♯ (v_1, v_2,Θ, ℬ,Γ) **using** *obtain-fresh*
by *blast*
hence *atom z* ♯ (v_1, v_2) ∧ *atom z* ♯ (Θ, ℬ,Γ) **using** *fresh-prodN* **by** *metis*
hence Θ; ℬ; Γ ⊢ *V-pair* v_1 v_2 ⇒ ⦃ *z* : [*b-of* τ_1 , *b-of* τ_2]^b | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) ⦄
using *assms infer-v-pairI zbc* **by** *metis*
moreover obtain τ **where** τ = (⦃ *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) ⦄) **by** *blast*
moreover hence *b-of* τ = *B-pair* (*b-of* τ_1) (*b-of* τ_2) **using** *b-of.simps zbc* **by** *presburger*
ultimately show *?thesis* **using** *b-of.simps* **by** *metis*
qed

lemma *infer-v-pair2I-zbc*:
fixes v_1::*v* **and** v_2::*v*
assumes Θ; ℬ; Γ ⊢ v_1 ⇒ τ_1 **and** Θ; ℬ; Γ ⊢ v_2 ⇒ τ_2
shows ∃ *z* τ. Θ; ℬ; Γ ⊢ *V-pair* v_1 v_2 ⇒ τ ∧ τ = (⦃ *z* : *B-pair* (*b-of* τ_1) (*b-of* τ_2) | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-pair* v_1 v_2)) ⦄) ∧ *atom z* ♯ (v_1,v_2) ∧ *atom z* ♯ Γ
proof −
obtain *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where** *zbc*: τ_1 = (⦃ *z1* : *b1* | *c1* ⦄) ∧ τ_2 = (⦃ *z2* : *b2* | *c2* ⦄)
using *τ.exhaust* **by** *meson*
obtain *z*::*x* **where** ∗ : *atom z* ♯ (v_1, v_2,Γ,Θ , ℬ) **using** *obtain-fresh*
by *blast*
hence *vinf*: Θ; ℬ; Γ ⊢ *V-pair* v_1 v_2 ⇒ ⦃ *z* :[*b-of* τ_1 , *b-of* τ_2]^b | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) ⦄
using *assms infer-v-pairI* **by** *simp*
moreover obtain τ **where** τ = (⦃ *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) ⦄) **by** *blast*
moreover have *b-of* τ_1 = *b1* ∧ *b-of* τ_2 = *b2* **using** *zbc b-of.simps* **by** *auto*
ultimately have Θ; ℬ; Γ ⊢ *V-pair* v_1 v_2 ⇒ τ ∧ τ = (⦃ *z* : *B-pair* (*b-of* τ_1) (*b-of* τ_2) | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) ⦄) **by** *auto*
thus *?thesis* **using** ∗ *fresh-prod2 fresh-prod3* **by** *metis*
qed

lemma *infer-v-pair2E*:
assumes Θ; $\mathcal{B}$; $\Gamma \vdash$ *V-pair* v_1 $v_2 \Rightarrow \tau$
shows $\exists \tau_1\ \tau_2\ z$. Θ; $\mathcal{B}$; $\Gamma \vdash v_1 \Rightarrow \tau_1 \wedge \Theta$; $\mathcal{B}$; $\Gamma \vdash v_2 \Rightarrow \tau_2 \wedge$
$\tau = (\{\!\!\{\ z : B\text{-}pair\ (b\text{-}of\ \tau_1)\ (b\text{-}of\ \tau_2)\ |\ C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ (V\text{-}pair\ v_1\ v_2))\ \}\!\!\}) \wedge$
atom $z \sharp\ (v_1, v_2)$
proof –
obtain z **and** *t1* **and** *t2* **where**
$\tau = (\{\!\!\{\ z : B\text{-}pair\ (b\text{-}of\ t1)\ (b\text{-}of\ t2)\ |\ CE\text{-}val\ (V\text{-}var\ z)\ ==\ CE\text{-}val\ (V\text{-}pair\ v_1\ v_2)\ \}\!\!\}) \wedge$
atom $z \sharp (v_1, v_2) \wedge\ \Theta$; $\mathcal{B}$; $\Gamma \vdash v_1 \Rightarrow t1\ \wedge\ \Theta$; $\mathcal{B}$; $\Gamma \vdash v_2 \Rightarrow t2$ **using** *infer-v-elims*(*3*)[*OF assms*
] **by** *metis*
thus *?thesis* **using** *b-of.simps* **by** *metis*
qed

12.7.2 Expressions

lemma *infer-e-app2E*:
fixes Φ::Φ **and** Θ::Θ
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-app f v* $\Rightarrow \tau$
shows $\exists x\ b\ c\ s'\ \tau'$. *wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta \wedge$ *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ' s')))
= *lookup-fun* Φ f $\wedge$ $\Theta \vdash_{wf} \Phi \wedge$
Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!\!\{\ x : b\ |\ c\ \}\!\!\} \wedge \tau = \tau'[x::=v]_{\tau v} \wedge$ *atom* $x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, \tau)$
using *infer-e-elims*(*6*)[*OF assms*] *b-of.simps subst-defs* **by** *metis*

lemma *infer-e-appP2E*:
fixes Φ::Φ **and** Θ::Θ
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-appP f b v* $\Rightarrow \tau$
shows $\exists bv\ x\ ba\ c\ s'\ \tau'$. *wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta \wedge$ *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x ba*
c τ' s'))) = *lookup-fun* Φ f $\wedge$ $\Theta \vdash_{wf} \Phi \wedge$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b $\wedge$
(Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!\!\{\ x : ba[bv::=b]_{bb}\ |\ c[bv::=b]_{cb}\ \}\!\!\}) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge$ *atom* $x \sharp \Gamma \wedge$
atom $bv \sharp v$
proof –
obtain $bv\ x\ ba\ c\ s'\ \tau'$ **where** $*$:*wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta \wedge$ *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ*
x ba c τ' s'))) = *lookup-fun* Φ f $\wedge$ $\Theta \vdash_{wf} \Phi \wedge$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b $\wedge$
(Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!\!\{\ x : ba[bv::=b]_{bb}\ |\ c[bv::=b]_{cb}\ \}\!\!\}) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge$ *atom* $x \sharp \Gamma \wedge$
atom $bv \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, b, v, \tau)$
using *infer-e-elims*(*21*)[*OF assms*] *subst-defs* **by** *metis*
moreover then have *atom* $bv \sharp v$ **using** *fresh-prodN* **by** *metis*
ultimately show *?thesis* **by** *metis*
qed

12.8 Weakening

Lemmas showing that typing judgements hold when a context is extended

lemma *subtype-weakening*:
fixes Γ'::Γ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash \tau1 \lesssim \tau2$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}$; $\Gamma' \vdash \tau1 \lesssim \tau2$
using *assms* **proof**(*nominal-induct* $\tau2$ *avoiding*: Γ' *rule*: *subtype.strong-induct*)

case (*subtype-baseI* x Θ $\mathcal{B}$ Γ z c z' c' b)

show *?case* **proof**
show $*$:Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄ **using** *wfT-weakening subtype-baseI* **by** *metis*
show Θ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ ⦃ *z*′ : *b* | *c*′ ⦄ **using** *wfT-weakening subtype-baseI* **by** *metis*
show *atom x* ♯ (Θ, $\mathcal{B}$, Γ′, *z* , *c* , *z*′ , *c*′) **using** *subtype-baseI fresh-Pair* **by** *metis*
have *toSet* ((*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ) ⊆ *toSet* ((*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ′) **using** *subtype-baseI toSet.simps* **by** *blast*
moreover have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ′ **using** *wfT-wf-cons3*[*OF* ∗, *of x*] *subtype-baseI fresh-Pair subst-defs* **by** *metis*
ultimately show Θ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ′ ⊨ *c*′[*z*′::=*V-var x*]$_v$ **using** *valid-weakening subtype-baseI* **by** *metis*
qed
qed

method *many-rules* **uses** *add* = ((*rule*+),((*simp add*: *add*)+)*?*)

lemma *infer-v-g-weakening*:
fixes *e*::*e* **and** Γ′::Γ **and** *v*::*v*
assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ τ **and** *toSet* Γ ⊆ *toSet* Γ′ **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′
shows Θ; $\mathcal{B}$; Γ′ ⊢ *v* ⇒ τ
using *assms* **proof**(*nominal-induct avoiding*: Γ′ *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ Γ *b c x*′ *z*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ › **using** *infer-v-varI* **by** *auto*
show ‹*Some* (*b*, *c*) = *lookup* Γ′ *x*′› **using** *infer-v-varI lookup-weakening* **by** *metis*
show ‹*atom z* ♯ *x*′› **using** *infer-v-varI* **by** *auto*
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′)› **using** *infer-v-varI* **by** *auto*
qed
next
case (*infer-v-litI* Θ $\mathcal{B}$ Γ *l* τ)
then show *?case* **using** *infer-v.intros* **by** *simp*
next
case (*infer-v-pairI z v1 v2* Θ $\mathcal{B}$ Γ *t1 t2*)
then show *?case* **using** *infer-v.intros* **by** *simp*
next
case (*infer-v-consI s dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv z*)
show *?case* **proof**
show ‹*AF-typedef s dclist* ∈ *set* Θ› **using** *infer-v-consI* **by** *auto*
show ‹(*dc*, *tc*) ∈ *set dclist*› **using** *infer-v-consI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ ⊢ *v* ⇒ *tv*› **using** *infer-v-consI* **by** *auto*
show ‹Θ; $\mathcal{B}$; Γ′ ⊢ *tv* ≲ *tc*› **using** *infer-v-consI subtype-weakening* **by** *auto*
show ‹*atom z* ♯ *v*› **using** *infer-v-consI* **by** *auto*
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′)› **using** *infer-v-consI* **by** *auto*
qed
next
case (*infer-v-conspI s bv dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv b z*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *infer-v-conspI* **by** *auto*
show ‹(*dc*, *tc*) ∈ *set dclist*› **using** *infer-v-conspI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ ⊢ *v* ⇒ *tv*› **using** *infer-v-conspI* **by** *auto*
show ‹Θ; $\mathcal{B}$; Γ′ ⊢ *tv* ≲ *tc*[*bv*::=*b*]$_{\tau b}$› **using** *infer-v-conspI subtype-weakening* **by** *auto*
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′, *v*, *b*)› **using** *infer-v-conspI* **by** *auto*
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ′, *v*, *b*)› **using** *infer-v-conspI* **by** *auto*

show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b › **using** *infer-v-conspI* **by** *auto*
qed
qed

lemma *check-v-g-weakening*:
fixes e::e **and** Γ'::Γ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \tau$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}$; $\Gamma' \vdash v \Leftarrow \tau$
using *subtype-weakening infer-v-g-weakening check-v-elims check-v-subtypeI assms* **by** *metis*

lemma *infer-e-g-weakening*:
fixes e::e **and** Γ'::Γ
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow \tau$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** Θ ; $\mathcal{B}$ $\vdash_{wf} \Gamma'$
shows Θ ; Φ ; $\mathcal{B}$; Γ'; $\Delta \vdash e \Rightarrow \tau$
using *assms* **proof**(*nominal-induct* τ *avoiding*: Γ' *rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ Δ' Φ v τ)
then show *?case* **using** *infer-v-g-weakening wf-weakening infer-e.intros* **by** *metis*
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

obtain z'::x **where** z': *atom* z' ♯ *v1* ∧ *atom* z' ♯ *v2* ∧ *atom* z' ♯ Γ' **using** *obtain-fresh fresh-prod3* **by** *metis*
moreover hence $*$:⦃ *z3* : *B-int* | *CE-val* (*V-var z3*) == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ = (⦃ z' : *B-int* | *CE-val* (*V-var* z') == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ ⦄)
using *infer-e-plusI type-e-eq ce.fresh fresh-e-opp* **by** *auto*

have Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ ⊢ *AE-op Plus v1 v2* ⇒ ⦃ z' : *B-int* | *CE-val* (*V-var* z') == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-plusI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-plusI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' ⊢ *v1* ⇒ ⦃ *z1* : *B-int* | *c1* ⦄› **using** *infer-v-g-weakening infer-e-plusI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' ⊢ *v2* ⇒ ⦃ *z2* : *B-int* | *c2* ⦄› **using** *infer-v-g-weakening infer-e-plusI* **by** *auto*
show ‹*atom* z' ♯ *AE-op Plus v1 v2*› **using** z' **by** *auto*
show ‹*atom* z' ♯ Γ'› **using** z' **by** *auto*
qed
thus *?case* **using** $*$ **by** *metis*

next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
obtain z'::x **where** z': *atom* z' ♯ *v1* ∧ *atom* z' ♯ *v2* ∧ *atom* z' ♯ Γ' **using** *obtain-fresh fresh-prod3* **by** *metis*

moreover hence $*$:⦃ *z3* : *B-bool* | *CE-val* (*V-var z3*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ = (⦃ z' : *B-bool* | *CE-val* (*V-var* z') == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄)
using *infer-e-leqI type-e-eq ce.fresh fresh-e-opp* **by** *auto*

have Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ ⊢ *AE-op LEq v1 v2* ⇒ ⦃ z' : *B-bool* | *CE-val* (*V-var* z') == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-leqI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-leqI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' ⊢ *v1* ⇒ ⦃ *z1* : *B-int* | *c1* ⦄› **using** *infer-v-g-weakening infer-e-leqI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' ⊢ *v2* ⇒ ⦃ *z2* : *B-int* | *c2* ⦄› **using** *infer-v-g-weakening infer-e-leqI* **by** *auto*

show ‹*atom* z' ♯ *AE-op LEq v1 v2*› **using** z' **by** *auto*
show ‹*atom* z' ♯ Γ′› **using** z' **by** *auto*
qed
thus *?case* **using** ∗ **by** *metis*
next
case (*infer-e-eqI* Θ *B* Γ Δ Φ *v1 z1 bb c1 v2 z2 c2 z3*)
obtain z'::*x* **where** z': *atom* z' ♯ *v1* ∧ *atom* z' ♯ *v2* ∧ *atom* z' ♯ Γ′ **using** *obtain-fresh fresh-prod3* **by** *metis*

moreover hence ∗:⦃ *z3* : *B-bool* | *CE-val* (*V-var z3*) == *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ = (⦃ z' : *B-bool* | *CE-val* (*V-var* z') == *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄)
using *infer-e-eqI type-e-eq ce.fresh fresh-e-opp* **by** *auto*

have Θ ; Φ ; *B* ; Γ′ ; Δ ⊢ *AE-op Eq v1 v2* ⇒ ⦃ z' : *B-bool* | *CE-val* (*V-var* z') == *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ **proof**
show ‹ Θ ; *B* ; Γ′ $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-eqI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-eqI* **by** *auto*
show ‹ Θ ; *B* ; Γ′ ⊢ *v1* ⇒ ⦃ *z1* : *bb* | *c1* ⦄› **using** *infer-v-g-weakening infer-e-eqI* **by** *auto*
show ‹ Θ ; *B* ; Γ′ ⊢ *v2* ⇒ ⦃ *z2* : *bb* | *c2* ⦄› **using** *infer-v-g-weakening infer-e-eqI* **by** *auto*
show ‹*atom* z' ♯ *AE-op Eq v1 v2*› **using** z' **by** *auto*
show ‹*atom* z' ♯ Γ′› **using** z' **by** *auto*
show *bb* ∈ {*B-bool*, *B-int*, *B-unit*} **using** *infer-e-eqI* **by** *auto*
qed
thus *?case* **using** ∗ **by** *metis*
next
case (*infer-e-appI* Θ *B* Γ Δ Φ *f x b c* τ′ *s′ v* τ)
show *?case* **proof**
show Θ; *B*; Γ′ $\vdash_{wf}$ Δ **using** *wf-weakening infer-e-appI* **by** *auto*
show Θ $\vdash_{wf}$ Φ **using** *wf-weakening infer-e-appI* **by** *auto*
show *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ′ *s′*))) = *lookup-fun* Φ *f* **using** *wf-weakening infer-e-appI* **by** *auto*
show Θ; *B*; Γ′ ⊢ *v* ⇐ ⦃ *x* : *b* | *c* ⦄ **using** *wf-weakening infer-e-appI check-v-g-weakening* **by** *auto*
show *atom x* ♯ (Θ, Φ, *B*, Γ′, Δ, *v*, τ) **using** *wf-weakening infer-e-appI* **by** *auto*
show $\tau'[x{::=}v]_v = \tau$ **using** *wf-weakening infer-e-appI* **by** *auto*
qed

next
case (*infer-e-appPI* Θ *B* Γ Δ Φ *b′ f bv x b c* τ′ *s′ v* τ)

hence ∗:Θ ; Φ ; *B* ; Γ ; Δ ⊢ *AE-appP f b′ v* ⇒ τ **using** *Typing.infer-e-appPI* **by** *auto*

obtain x'::*x* **where** x':*atom* x' ♯ (*s′*, *c*, τ′, (Γ′,*v*,τ)) ∧ (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ′ *s′*))) = (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ* x' *b* ((x' ↔ *x*) • *c*) ((x' ↔ *x*) • τ′) ((x' ↔ *x*) • *s′*))))
using *obtain-fresh-fun-def*[*of s′ c* τ′ (Γ′,*v*,τ) *f x b*]
by (*metis fun-def.eq-iff fun-typ-q.eq-iff*(*2*))

hence ∗∗: ⦃ *x* : *b* | *c* ⦄ = ⦃ x' : *b* | (x' ↔ *x*) • *c* ⦄
using *fresh-PairD*(*1*) *fresh-PairD*(*2*) *type-eq-flip* **by** *blast*
have *atom* x' ♯ Γ **using** x' *infer-e-appPI fresh-weakening fresh-Pair* **by** *metis*

show *?case* **proof**(*rule Typing.infer-e-appPI*[**where** *x*=x' **and** *bv*=*bv* **and** *b*=*b* **and** *c*=(x' ↔ *x*) • *c*

and τ'=$(x' \leftrightarrow x) \cdot \tau'$**and** s'=$((x' \leftrightarrow x) \cdot s')$])
show ‹ Θ ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-appPI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wf-weakening infer-e-appPI* **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ b' **using** *infer-e-appPI* **by** *auto*
show *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ* x' *b* $((x' \leftrightarrow x) \cdot c)$ $((x' \leftrightarrow x) \cdot \tau')$ $((x' \leftrightarrow x) \cdot s')$))) = *lookup-fun* Φ *f* **using** x' *infer-e-appPI* **by** *argo*
show Θ; $\mathcal{B}$; Γ′ ⊢ v ⇐ ⦃ x' : $b[bv::=b']_b$ | $((x' \leftrightarrow x) \cdot c)[bv::=b']_b$ ⦄ **using** ∗∗
τ.eq-iff check-v-g-weakening infer-e-appPI.hyps infer-e-appPI.prems(*1*) *infer-e-appPI.prems subst-defs*
subst-tb.simps **by** *metis*
show *atom* x' ♯ Γ′ **using** x' *fresh-prodN* **by** *metis*

have *atom x* ♯ (v, τ) ∧ *atom* x' ♯ (v, τ) **using** x' *infer-e-fresh*[*OF* ∗] *e.fresh*(*2*) *fresh-Pair infer-e-appPI* ‹*atom* x' ♯ Γ› *e.fresh* **by** *metis*
moreover then have $((x' \leftrightarrow x) \cdot \tau')[bv::=b']_{\tau b} = (x' \leftrightarrow x) \cdot (\tau'[bv::=b']_{\tau b})$ **using** *subst-b-x-flip*
by (*metis subst-b-τ-def*)
ultimately show $((x' \leftrightarrow x) \cdot \tau')[bv::=b']_b[x'::=v]_v = \tau$
using *infer-e-appPI subst-tv-flip subst-defs* **by** *metis*

show *atom bv* ♯ (Θ, Φ, $\mathcal{B}$, Γ′, Δ, b', v, τ) **using** *infer-e-appPI fresh-prodN* **by** *metis*
qed

next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ Φ *v* z'' *b1 b2 c z*)

obtain z'::*x* **where** ∗: *atom* z' ♯ Γ′ ∧ *atom* z' ♯ *v* ∧ *atom* z' ♯ *c* **using** *obtain-fresh-z fresh-Pair* **by** *metis*
hence ∗∗:⦃ *z* : *b1* | *CE-val* (*V-var z*) == *CE-fst* $[v]^{ce}$ ⦄ = ⦃ z' : *b1* | *CE-val* (*V-var* z') == *CE-fst* $[v]^{ce}$ ⦄
using *type-e-eq infer-e-fstI v.fresh e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ⊢ *AE-fst v* ⇒ ⦃ z' : *b1* | *CE-val* (*V-var* z') == *CE-fst* $[v]^{ce}$ ⦄ **proof**
show ‹ Θ ; $\mathcal{B}$; Γ′ $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-fstI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wf-weakening infer-e-fstI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ′ ⊢ *v* ⇒ ⦃ z'' : *B-pair b1 b2* | *c* ⦄ **using** *infer-v-g-weakening infer-e-fstI* **by** *metis*
show *atom* z' ♯ *AE-fst v* **using** ∗ ∗∗ *e.supp* **by** *auto*
show *atom* z' ♯ Γ′ **using** ∗ **by** *auto*
qed
thus *?case* **using** ∗ ∗∗ **by** *metis*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ Φ *v* z'' *b1 b2 c z*)

obtain z'::*x* **where** ∗: *atom* z' ♯ Γ′ ∧ *atom* z' ♯ *v* ∧ *atom* z' ♯ *c* **using** *obtain-fresh-z fresh-Pair* **by** *metis*
hence ∗∗:⦃ *z* : *b2* | *CE-val* (*V-var z*) == *CE-snd* $[v]^{ce}$ ⦄ = ⦃ z' : *b2* | *CE-val* (*V-var* z') == *CE-snd* $[v]^{ce}$ ⦄
using *type-e-eq infer-e-sndI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ⊢ *AE-snd v* ⇒ ⦃ z' : *b2* | *CE-val* (*V-var* z') == *CE-snd* $[v]^{ce}$ ⦄ **proof**
show ‹ Θ ; $\mathcal{B}$;Γ′ $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-sndI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wf-weakening infer-e-sndI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ′ ⊢ *v* ⇒ ⦃ z'' : *B-pair b1 b2* | *c* ⦄ **using** *infer-v-g-weakening infer-e-sndI* **by**

metis
  show *atom* z' ♯ *AE-snd v* **using** ∗ *e.supp* **by** *auto*
  show *atom* z' ♯ Γ′ **using** ∗ **by** *auto*
 qed
 thus *?case* **using** ∗∗ **by** *metis*
next
 case (*infer-e-lenI* Θ ℬ Γ Δ Φ *v* z'' *c z*)

 obtain z'::*x* **where** ∗: *atom* z' ♯ Γ′ ∧ *atom* z' ♯ *v* ∧ *atom* z' ♯ *c* **using** *obtain-fresh-z fresh-Pair* **by** *metis*
 hence ∗∗:⦃ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* $[v]^{ce}$ ⦄ = ⦃ z' : *B-int* | *CE-val* (*V-var* z') == *CE-len* $[v]^{ce}$ ⦄
  using *type-e-eq infer-e-lenI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

 have Θ ; Φ ; ℬ ; Γ′ ; Δ ⊢ *AE-len v* ⇒ ⦃ z' : *B-int* | *CE-val* (*V-var* z') == *CE-len* $[v]^{ce}$ ⦄ **proof**
  show ‹ Θ; ℬ; Γ′ $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-lenI* **by** *auto*
  show ‹ Θ $\vdash_{wf}$ Φ › **using** *wf-weakening infer-e-lenI* **by** *auto*
  show Θ ; ℬ ; Γ′ ⊢ *v* ⇒ ⦃ z'' : *B-bitvec* | *c* ⦄ **using** *infer-v-g-weakening infer-e-lenI* **by** *metis*
  show *atom* z' ♯ *AE-len v* **using** ∗ *e.supp* **by** *auto*
  show *atom* z' ♯ Γ′ **using** ∗ **by** *auto*
 qed
 thus *?case* **using** ∗ ∗∗ **by** *metis*
next
 case (*infer-e-mvarI* Θ Γ Φ Δ *u* τ)
 then show *?case* **using** *wf-weakening infer-e.intros* **by** *metis*
next
 case (*infer-e-concatI* Θ ℬ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

 obtain z'::*x* **where** ∗: *atom* z' ♯ Γ′ ∧ *atom* z' ♯ *v1* ∧ *atom* z' ♯ *v2* **using** *obtain-fresh-z fresh-Pair* **by** *metis*
 hence ∗∗:⦃ *z3* : *B-bitvec* | *CE-val* (*V-var z3*) == *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ = ⦃ z' : *B-bitvec* | *CE-val* (*V-var* z') == *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ ⦄
  using *type-e-eq infer-e-concatI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

 have Θ ; Φ ; ℬ ; Γ′ ; Δ ⊢ *AE-concat v1 v2* ⇒ ⦃ z' : *B-bitvec* | *CE-val* (*V-var* z') == *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ ⦄ **proof**
  show ‹ Θ; ℬ; Γ′ $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-concatI* **by** *auto*
  show ‹ Θ $\vdash_{wf}$ Φ › **using** *wf-weakening infer-e-concatI* **by** *auto*
  show Θ ; ℬ ; Γ′ ⊢ *v1* ⇒ ⦃ *z1* : *B-bitvec* | *c1* ⦄ **using** *infer-v-g-weakening infer-e-concatI* **by** *metis*
  show Θ ; ℬ ; Γ′ ⊢ *v2* ⇒ ⦃ *z2* : *B-bitvec* | *c2* ⦄ **using** *infer-v-g-weakening infer-e-concatI* **by** *metis*
  show *atom* z' ♯ *AE-concat v1 v2* **using** ∗ *e.supp* **by** *auto*
  show *atom* z' ♯ Γ′ **using** ∗ **by** *auto*
 qed
 thus *?case* **using** ∗ ∗∗ **by** *metis*
next
 case (*infer-e-splitI* Θ ℬ Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
 show *?case* **proof**
  show Θ; ℬ; Γ′ $\vdash_{wf}$ Δ **using** *infer-e-splitI wf-weakening* **by** *auto*
  show Θ $\vdash_{wf}$ Φ **using** *infer-e-splitI wf-weakening* **by** *auto*
  show Θ; ℬ; Γ′ ⊢ *v1* ⇒ ⦃ *z1* : *B-bitvec* | *c1* ⦄ **using** *infer-v-g-weakening infer-e-splitI* **by** *metis*

show Θ; $\mathcal{B}$; Γ′ ⊢ *v2* ⇐ ⦃ *z2* : *B-int* | [*leq* [[*L-num 0*]^v]^{ce} [[*z2*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce} *AND* [*leq* [[*z2*]^v]^{ce} [| [*v1*]^{ce} |]^{ce}]^{ce} == [[*L-true*]^v]^{ce} ⦄
using *check-v-g-weakening infer-e-splitI* **by** *metis*
show *atom z1* ♯ *AE-split v1 v2* **using** *infer-e-splitI* **by** *auto*
show *atom z1* ♯ Γ′ **using** *infer-e-splitI* **by** *auto*
show *atom z2* ♯ *AE-split v1 v2* **using** *infer-e-splitI* **by** *auto*
show *atom z2* ♯ Γ′ **using** *infer-e-splitI* **by** *auto*
show *atom z3* ♯ *AE-split v1 v2* **using** *infer-e-splitI* **by** *auto*
show *atom z3* ♯ Γ′ **using** *infer-e-splitI* **by** *auto*
qed
qed

Special cases proved explicitly, other cases at the end with method +

lemma *infer-e-d-weakening*:
fixes *e*::*e*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *e* ⇒ τ **and** *setD* Δ ⊆ *setD* Δ′ **and** *wfD* Θ $\mathcal{B}$ Γ Δ′
shows Θ; Φ ; $\mathcal{B}$; Γ ; Δ′ ⊢ *e* ⇒ τ
using *assms* **by**(*nominal-induct* τ *avoiding*: Δ′ *rule*: *infer-e.strong-induct*,*auto simp add*:*infer-e.intros*)

lemma *wfG-x-fresh-in-v-simple*:
fixes *x*::*x* **and** *v* :: *v*
assumes Θ; $\mathcal{B}$; Γ ⊢ *v* ⇒ τ **and** *atom x* ♯ Γ
shows *atom x* ♯ *v*
using *wfV-x-fresh infer-v-wf assms* **by** *metis*

lemma *check-s-g-weakening*:
fixes *v*::*v* **and** *s*::*s* **and** *cs*::*branch-s* **and** *x*::*x* **and** *c*::*c* **and** *b*::*b* **and** Γ′::Γ **and** Θ::Θ **and** *css*::*branch-list*
shows *check-s* Θ Φ $\mathcal{B}$ Γ Δ *s t* ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *check-s* Θ Φ $\mathcal{B}$ Γ′ Δ *s t* **and**
check-branch-s Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs t* ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *check-branch-s* Θ Φ $\mathcal{B}$ Γ′ Δ *tid cons const v cs t* **and**
check-branch-list Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v css t* ⟹ *toSet* Γ ⊆ *toSet* Γ′ ⟹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ ⟹ *check-branch-list* Θ Φ $\mathcal{B}$ Γ′ Δ *tid dclist v css t*
proof(*nominal-induct t* **and** *t* **and** *t avoiding*: Γ′ *rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ′ Φ *v* τ′ τ)
then show *?case* **using** *Typing.check-valI infer-v-g-weakening wf-weakening subtype-weakening* **by** *metis*
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
hence *xf*:*atom x* ♯ Γ′ **by** *metis*
show *?case* **proof**
show *atom x* ♯ (Θ, Φ, $\mathcal{B}$, Γ′, Δ, *e*, τ) **using** *check-letI* **using** *fresh-prod4 xf* **by** *metis*
show Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ⊢ *e* ⇒ ⦃ *z* : *b* | *c* ⦄ **using** *infer-e-g-weakening check-letI* **by** *metis*
show *atom z* ♯ (*x*, Θ, Φ, $\mathcal{B}$, Γ′, Δ, *e*, τ, *s*)
by(*unfold fresh-prodN*,*auto simp add*: *check-letI fresh-prodN*)
have *toSet* ((*x*, *b*, *c*[*z*::=*V-var x*]$_v$) #$_\Gamma$ Γ) ⊆ *toSet* ((*x*, *b*, *c*[*z*::=*V-var x*]$_v$) #$_\Gamma$ Γ′) **using** *check-letI toSet.simps*
by (*metis Un-commute Un-empty-right Un-insert-right insert-mono*)
moreover hence Θ ; $\mathcal{B}$ $\vdash_{wf}$ ((*x*, *b*, *c*[*z*::=*V-var x*]$_v$) #$_\Gamma$ Γ′) **using** *check-letI wfG-cons-weakening check-s-wf* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) #$_\Gamma$ Γ′ ; Δ ⊢ *s* ⇐ τ **using** *check-letI* **by** *metis*
qed

next
case (*check-let2I x* Θ Φ *ℬ G* Δ *t s1* τ *s2*)
show *?case* **proof**
show *atom x* ♯ (Θ, Φ, ℬ, Γ′, Δ, *t*, *s1*, τ) **using** *check-let2I* **using** *fresh-prod4* **by** *auto*
show Θ ; Φ ; ℬ ; Γ′ ; Δ ⊢ *s1* ⇐ *t* **using** *check-let2I* **by** *metis*
have *toSet* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G*) ⊆ *toSet* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ Γ′) **using** *check-let2I* **by** *auto*
moreover hence Θ ; ℬ $\vdash_{wf}$ ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ Γ′) **using** *check-let2I wfG-cons-weakening check-s-wf* **by** *metis*
ultimately show Θ ; Φ ; ℬ ; (*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ Γ′ ; Δ ⊢ *s2* ⇐ τ **using** *check-let2I* **by** *metis*
qed
next
case (*check-branch-list-consI* Θ Φ ℬ Γ Δ *tid dclist′ v cs* τ *css dclist*)
thus *?case* **using** *Typing.check-branch-list-consI* **by** *metis*
next
case (*check-branch-list-finalI* Θ Φ ℬ Γ Δ *tid dclist′ v cs* τ *dclist*)
thus *?case* **using** *Typing.check-branch-list-finalI* **by** *metis*
next
case (*check-branch-s-branchI* Θ ℬ Γ Δ τ *const x* Φ *tid cons v s*)
show *?case* **proof**
show Θ; ℬ; Γ′ $\vdash_{wf}$ Δ **using** *wf-weakening2*(*6*) *check-branch-s-branchI* **by** *metis*
show $\vdash_{wf}$ Θ **using** *check-branch-s-branchI* **by** *auto*
show Θ; ℬ; Γ′ $\vdash_{wf}$ τ **using** *check-branch-s-branchI wfT-weakening* ‹*wfG* Θ ℬ Γ′› **by** *presburger*

show Θ ; {||} ; *GNil* $\vdash_{wf}$ *const* **using** *check-branch-s-branchI* **by** *auto*
show *atom x* ♯ (Θ, Φ, ℬ, Γ′, Δ, *tid*, *cons*, *const*, *v*, τ) **using** *check-branch-s-branchI* **by** *auto*
have *toSet* ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) ⊆ *toSet* ((*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ′)
using *check-branch-s-branchI* **by** *auto*
moreover hence Θ ; ℬ $\vdash_{wf}$ ((*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ′)
using *check-branch-s-branchI wfG-cons-weakening check-s-wf* **by** *metis*
ultimately show Θ ; Φ ; ℬ ; (*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ′ ; Δ ⊢ *s* ⇐ τ
using *check-branch-s-branchI* **using** *fresh-dom-free* **by** *auto*
qed
next
case (*check-ifI z* Θ Φ ℬ Γ Δ *v s1 s2* τ)
show *?case* **proof**
show ‹*atom z* ♯ (Θ, Φ, ℬ, Γ′, Δ, *v*, *s1*, *s2*, τ)› **using** *fresh-prodN check-ifI* **by** *auto*
show ‹Θ; ℬ; Γ′ ⊢ *v* ⇐ ⦃ *z* : *B-bool* | *TRUE* ⦄› **using** *check-v-g-weakening check-ifI* **by** *auto*
show ‹ Θ ; Φ ; ℬ ; Γ′ ; Δ ⊢ *s1* ⇐ ⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-true*) *IMP c-of* τ *z* ⦄› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; ℬ ; Γ′ ; Δ ⊢ *s2* ⇐ ⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP c-of* τ *z* ⦄› **using** *check-ifI* **by** *auto*
qed
next
case (*check-whileI* Δ *G P s1 z s2* τ′)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros check-v-g-weakening subtype-weakening wf-weakening*
by (*meson infer-v-g-weakening*)

next
case (*check-seqI* Δ *G P s1 z s2* τ)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros check-v-g-weakening subtype-weakening wf-weakening*
by (*meson infer-v-g-weakening*)
next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)
thus *?case* **using** *check-v-g-weakening check-s-check-branch-s-check-branch-list.intros* **by** *auto*
next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ *v z* τ')
show *?case* **proof**
show ‹$\Theta \vdash_{wf} \Phi$ › **using** *check-assignI* **by** *auto*
show ‹Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$› **using** *check-assignI wf-weakening* **by** *auto*
show ‹$(u, \tau) \in$ *setD* Δ› **using** *check-assignI* **by** *auto*
show ‹Θ; $\mathcal{B}$; $\Gamma' \vdash v \Leftarrow \tau$› **using** *check-assignI check-v-g-weakening* **by** *auto*
show ‹Θ; $\mathcal{B}$; $\Gamma' \vdash$ ⦃ z : *B-unit* | *TRUE* ⦄ $\lesssim \tau'$› **using** *subtype-weakening check-assignI* **by** *auto*
qed
next
case (*check-caseI* Δ Γ Θ *dclist cs* τ *tid v z*)

then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros check-v-g-weakening subtype-weakening wf-weakening*
by (*meson infer-v-g-weakening*)
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
show *?case* **proof**
show ‹*atom x* ♯ (Θ, Φ, $\mathcal{B}$, Γ', Δ, *c*, τ, *s*)› **using** *check-assertI* **by** *auto*

have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ **using** *check-assertI check-s-wf* **by** *metis*
hence $*$: Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ' **using** *wfG-cons-weakening check-assertI* **by** *metis*
moreover have *toSet* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) $\subseteq$ *toSet* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ') **using** *check-assertI* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ' ; Δ $\vdash s \Leftarrow \tau$› **using** *check-assertI*(*11*) [*OF* - $*$] **by** *auto*

show ‹Θ; $\mathcal{B}$; $\Gamma' \models c$ › **using** *check-assertI valid-weakening* **by** *metis*
show ‹ Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$ › **using** *check-assertI wf-weakening* **by** *metis*
qed
qed

lemma *wfG-xa-fresh-in-v*:
fixes *c*::*c* **and** Γ::Γ **and** *G*::Γ **and** *v*::*v* **and** *xa*::*x*
assumes Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *G*=(Γ'@ (*x*, *b*, $c[z::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) **and** *atom xa* ♯ *G* **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ *G*
shows *atom xa* ♯ *v*
proof −
have Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b-of* τ **using** *infer-v-wf assms* **by** *metis*
hence *supp v* $\subseteq$ *atom-dom* Γ $\cup$ *supp* $\mathcal{B}$ **using** *wfV-supp* **by** *simp*
moreover have *atom xa* $\notin$ *atom-dom G*
using *assms wfG-atoms-supp-eq*[*OF assms*(*4*)] *fresh-def* **by** *metis*
ultimately show *?thesis* **using** *fresh-def*
using *assms infer-v-wf wfG-atoms-supp-eq*
fresh-GCons fresh-append-g subsetCE

by (*metis wfG-x-fresh-in-v-simple*)
qed

lemma *fresh-z-subst-g*:
fixes G::Γ
assumes *atom* $z' \sharp (x,v)$ **and** ‹*atom* $z' \sharp G$›
shows *atom* $z' \sharp G[x::=v]_{\Gamma v}$
proof –
have *atom* $z' \sharp v$ **using** *assms fresh-prod2* **by** *auto*
thus *?thesis* **using** *fresh-subst-gv assms* **by** *metis*
qed

lemma *wfG-xa-fresh-in-subst-v*:
fixes c::c **and** v::v **and** x::x **and** Γ::Γ **and** G::Γ **and** xa::x
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ **and** G=(Γ'@ (x, b, $c[z::=V\text{-}var\ x]_v$) $\#_\Gamma \Gamma$) **and** *atom* $xa \sharp G$ **and** $\Theta ; \mathcal{B} \vdash_{wf} G$
shows *atom* $xa \sharp$ (*subst-gv G x v*)
proof –
have *atom* $xa \sharp v$ **using** *wfG-xa-fresh-in-v assms* **by** *metis*
thus *?thesis* **using** *fresh-subst-gv assms* **by** *metis*
qed

12.8.1 Weakening Immutable Variable Context

declare *check-s-check-branch-s-check-branch-list.intros*[*simp*]
declare *check-s-check-branch-s-check-branch-list.intros*[*intro*]

lemma *check-s-d-weakening*:
fixes s::s **and** v::v **and** cs::*branch-s* **and** css::*branch-list*
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ *wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta' \Longrightarrow \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta' \vdash s \Leftarrow \tau$ **and**
check-branch-s $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *tid cons const v cs* $\tau \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ *wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta' \Longrightarrow$ *check-branch-s* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta'$ *tid cons const v cs* τ **and**
check-branch-list $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *tid dclist v css* $\tau \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ *wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta' \Longrightarrow$ *check-branch-list* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta'$ *tid dclist v css* τ
proof(*nominal-induct* τ **and** τ **and** τ *avoiding*: Δ' *arbitrary*: v *rule*: *check-s-check-branch-s-check-branch-list.strong-indu*
case (*check-valI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v\ \tau'\ \tau$)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *blast*
next
case (*check-letI* $x\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ \tau\ z\ s\ b\ c$)
show *?case* **proof**
show *atom* $x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta', e, \tau)$ **using** *check-letI* **by** *auto*
show *atom* $z \sharp (x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta', e, \tau, s)$ **using** *check-letI* **by** *auto*
show $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta' \vdash e \Rightarrow \{\!| z : b \mid c |\!\}$ **using** *check-letI infer-e-d-weakening* **by** *auto*
have $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma$ **using** *check-letI check-s-wf* **by** *metis*
moreover have $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta'$ **using** *check-letI check-s-wf* **by** *metis*
ultimately have $\Theta; \mathcal{B}; (x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma \vdash_{wf} \Delta'$ **using** *wf-weakening2*(*6*) *toSet.simps* **by** *fast*
thus $\Theta ; \Phi ; \mathcal{B} ; (x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma ; \Delta' \vdash s \Leftarrow \tau$ **using** *check-letI* **by** *simp*
qed
next
case (*check-branch-s-branchI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \tau$ *const x* Φ *tid cons v s*)
moreover have $\Theta ;\mathcal{B} \vdash_{wf}$ (x, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND*

c-of const x) $\#_\Gamma$ Γ
  using *check-s-wf*[*OF check-branch-s-branchI*(*16*)] **by** *metis*
 moreover hence Θ ;*B* ; (*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ′
  using *wf-weakening2*(*6*) *check-branch-s-branchI* **by** *fastforce*
 ultimately show *?case*
  using *check-s-check-branch-s-check-branch-list.intros* **by** *simp*
next
 case (*check-branch-list-consI* Θ Φ *B* Γ Δ *tid dclist v cs* τ *css*)
 then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
 case (*check-branch-list-finalI* Θ Φ *B* Γ Δ *tid dclist v cs* τ)
 then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
 case (*check-ifI z* Θ Φ *B* Γ Δ *v s1 s2* τ)
 show *?case* **proof**
  show ‹*atom z* ♯ (Θ, Φ, *B*, Γ, Δ′, *v*, *s1*, *s2*, τ)› **using** *fresh-prodN check-ifI* **by** *auto*
  show ‹Θ; *B*; Γ ⊢ *v* ⇐ ⦃ *z* : *B-bool* | *TRUE* ⦄› **using** *check-ifI* **by** *auto*
  show ‹ Θ ; Φ ; *B* ; Γ ; Δ′ ⊢ *s1* ⇐ ⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-true*) *IMP c-of* τ *z* ⦄› **using** *check-ifI* **by** *auto*
  show ‹ Θ ; Φ ; *B* ; Γ ; Δ′ ⊢ *s2* ⇐ ⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP c-of* τ *z* ⦄› **using** *check-ifI* **by** *auto*
 qed
next
 case (*check-assertI x* Θ Φ *B* Γ Δ *c* τ *s*)
 show *?case* **proof**
  show *atom x* ♯ (Θ, Φ, *B*, Γ, Δ′, *c*, τ,*s*) **using** *fresh-prodN check-assertI* **by** *auto*
  show ∗: Θ; *B*; Γ $\vdash_{wf}$ Δ′ **using** *check-assertI* **by** *auto*
  hence Θ; *B*; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ′ **using** *wf-weakening2*(*6*)[*OF* ∗, *of* (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ] *check-assertI check-s-wf toSet.simps* **by** *auto*
  thus Θ ; Φ ; *B* ; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ′ ⊢ *s* ⇐ τ
   using *check-assertI*(*11*)[*OF* ‹*setD* Δ ⊆ *setD* Δ′›] **by** *simp*

  show Θ; *B*; Γ ⊨ *c* **using** *fresh-prodN check-assertI* **by** *auto*

 qed
next
 case (*check-let2I x* Θ Φ *B G* Δ *t s1* τ *s2*)
 show *?case* **proof**
  show *atom x* ♯ (Θ, Φ, *B*, *G*, Δ′, *t* , *s1*, τ) **using** *check-let2I* **by** *auto*

  show Θ ; Φ ; *B* ; *G* ; Δ′ ⊢ *s1* ⇐ *t* **using** *check-let2I infer-e-d-weakening* **by** *auto*
  have Θ; *B*; (*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G* $\vdash_{wf}$ Δ′ **using** *check-let2I wf-weakening2*(*6*) *check-s-wf* **by** *fastforce*
  thus Θ ; Φ ; *B* ; (*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G* ; Δ′ ⊢ *s2* ⇐ τ **using** *check-let2I* **by** *simp*
 qed
next
 case (*check-varI u* Θ Φ *B* Γ Δ τ′ *v* τ *s*)
 show *?case* **proof**
  show *atom u* ♯ (Θ, Φ, *B*, Γ, Δ′, τ′, *v*, τ) **using** *check-varI* **by** *auto*
  show Θ; *B*; Γ ⊢ *v* ⇐ τ′ **using** *check-varI* **by** *auto*
  have *setD* ((*u*, τ′) $\#_\Delta$Δ) ⊆ *setD* ((*u*, τ′) $\#_\Delta$Δ′) **using** *setD.simps check-varI* **by** *auto*

moreover have $u \notin$ *fst ‘ setD* Δ' **using** *check-varI(1) setD.simps fresh-DCons* **by** (*simp add*: *fresh-d-not-in*)
moreover hence Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ (u, τ') $\#_\Delta\Delta'$ **using** *wfD-cons fresh-DCons setD.simps check-varI check-v-wf* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; Γ ; (u, τ') $\#_\Delta\Delta'$ $\vdash s \Leftarrow \tau$ **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ *v z* τ')
moreover hence $(u, \tau) \in$ *setD* Δ' **by** *auto*
ultimately show *?case* **using** *Typing.check-assignI* **by** *simp*
next
case (*check-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ')
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
case (*check-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *z*)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*

qed

lemma *valid-ce-eq*:
fixes *v*::*v* **and** *ce2*::*ce*
assumes *ce1* = *ce2*$[x::=v]_{cev}$ **and** *wfV* Θ $\mathcal{B}$ *GNil v b* **and** *wfCE* Θ $\mathcal{B}$ ((*x*, *b*, *TRUE*) $\#_\Gamma$ *GNil*) *ce2 b'* **and** *wfCE* Θ $\mathcal{B}$ *GNil ce1 b'*
shows ‹Θ; $\mathcal{B}$; (*x*, *b*, ($[[x]^v]^{ce}$ == $[v]^{ce}$)) $\#_\Gamma$ *GNil* $\models$ *ce1* == *ce2* ›
unfolding *valid.simps* **proof**
have *wfg*: Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil*
using *wfG-cons1I wfG-nilI wfX-wfY assms wf-intros*
by (*meson fresh-GNil wfC-e-eq wfG-intros2*)

show *wf*: ‹Θ; $\mathcal{B}$; (*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce1* == *ce2* ›
apply(*rule wfC-eqI*[**where** *b*=*b'*])
using *wfg toSet.simps assms wfCE-weakening* **apply** *simp*

using *wfg assms wf-replace-inside1*(*8*) *assms*
using *wfC-trueI wf-trans*(*8*) **by** *auto*

show ‹$\forall$ *i*. ((Θ ; (*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil* $\vdash$ *i*) $\wedge$ (*i* $\models$ (*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil*) $\longrightarrow$
(*i* $\models$ (*ce1* == *ce2*)))› **proof**(*rule*+,*goal-cases*)
fix *i*
assume *as*:Θ ; (*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil* $\vdash$ *i* $\wedge$ *i* $\models$ (*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil*
have *1*:*wfV* Θ $\mathcal{B}$ ((*x*, *b*, $[[x]^v]^{ce}$ == $[v]^{ce}$) $\#_\Gamma$ *GNil*) *v b*
using *wf-weakening assms append-g.simps toSet.simps wf wfX-wfY*
by (*metis empty-subsetI*)
hence $\exists$ *s*. *i* $[\![v]\!]$ ~ *s* **using** *eval-v-exist*[*OF - 1*] *as* **by** *auto*
then obtain *s* **where** *iv*:*i*$[\![v]\!]$ ~ *s* **..**

hence *ix*:*i x* = *Some s* **proof** −

have $i \models [\ [\ x\ ]^v\ ]^{ce}\ ==\ [\ v\ ]^{ce}$ **using** *is-satis-g.simps as* **by** *auto*
hence $i\ [\![\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ [\ v\ ]^{ce}\ ]\!]\ \sim$ *True* **using** *is-satis.simps* **by** *auto*
hence $i\ [\![\ \ [\ [\ x\ ]^v\ ]^{ce}\ ]\!]\ \sim\ s$ **using**
iv eval-e-elims
by (*metis eval-c-elims*(*7*) *eval-e-uniqueness eval-e-valI*)
thus *?thesis* **using** *eval-v-elims*(*2*) *eval-e-elims*(*1*) **by** *metis*
qed

have *1*:*wfCE* $\Theta\ \mathcal{B}$ ((*x*, *b*, $[\ [\ x\ ]^v\ ]^{ce}\ ==\ [\ v\ ]^{ce}$) $\#_\Gamma$ *GNil*) *ce1 b′*
using *wfCE-weakening assms append-g.simps toSet.simps wf wfX-wfY*
by (*metis empty-subsetI*)
hence $\exists$ *s1*. $i\ [\![\ ce1\ ]\!]\ \sim\ s1$ **using** *eval-e-exist assms as* **by** *auto*
then obtain *s1* **where** *s1*: $i[\![ce1]\!]\ \sim\ s1$ **..**

moreover have $i[\![\ ce2\ ]\!]\ \sim\ s1$ **proof** −
have $i[\![\ ce2[x::=v]_{cev}\ ]\!]\ \sim\ s1$ **using** *assms s1* **by** *auto*
moreover have $ce1 = ce2[x::=v]_{cev}$ **using** *subst-v-ce-def assms subst-v-simple-commute* **by** *auto*
ultimately have $i(x \mapsto s)\ [\![\ ce2\ ]\!]\ \sim\ s1$
using *ix subst-e-eval-v*[*of i ce1 s1* $ce2[z::=[\ x\ ]^v]_v$ *x v s*] *iv s1* **by** *auto*
moreover have $i(x \mapsto s) = i$ **using** *ix* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
ultimately show $i\ [\![\ ce1\ ==\ ce2\ ]\!]\ \sim$ *True* **using** *eval-c-eqI* **by** *metis*
qed
qed

lemma *check-v-top*:
fixes *v*::*v*
assumes Θ; $\mathcal{B}$; *GNil* $\vdash v \Leftarrow \tau$ **and** $ce1 = ce2[z::=v]_{cev}$ **and** Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $\{\!\mid z : b\text{-}of\ \tau\ \mid ce1\ == ce2\ \mid\!\}$
and *supp ce1* $\subseteq$ *supp* $\mathcal{B}$
shows Θ; $\mathcal{B}$; *GNil* $\vdash v \Leftarrow \{\!\mid z : b\text{-}of\ \tau\ \mid ce1 == ce2\ \mid\!\}$
proof −
obtain *t* **where** *t*: Θ; $\mathcal{B}$; *GNil* $\vdash v \Rightarrow t \land \Theta$; $\mathcal{B}$; *GNil* $\vdash t \lesssim \tau$
using *assms check-v-elims* **by** *metis*

then obtain *z′* **and** *b′* **where** ∗:$t = \{\!\mid z' : b'\ \mid [\ [\ z'\ ]^v\ ]^{ce}\ ==\ [\ v\ ]^{ce}\ \mid\!\} \land$ *atom z′* $\sharp$ *v* $\land$ *atom z′* $\sharp$ (Θ, $\mathcal{B}$,*GNil*)
using *assms infer-v-form* **by** *metis*
have *beq*: *b-of t* = *b-of* τ **using** *subtype-eq-base2 b-of.simps t* **by** *auto*
obtain *x*::*x* **where** *xf*: ‹*atom x* $\sharp$ (Θ, $\mathcal{B}$, *GNil*, *z′*, $[\ [\ z'\ ]^v\ ]^{ce}\ ==\ [\ v\ ]^{ce}$, *z*, *ce1* == *ce2*)›
using *obtain-fresh* **by** *metis*

have Θ; $\mathcal{B}$; (*x*, *b-of* τ, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ ($ce1[z::=[\ x\ ]^v]_v\ ==\ ce2[z::=[\ x\ ]^v]_v$)
using *wfT-wfC2*[*OF assms*(*3*), *of x*] *subst-cv.simps*(*6*) *subst-v-c-def subst-v-ce-def fresh-GNil* **by** *simp*

then obtain *b2* **where** *b2*: Θ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ $ce1[z::=[\ x\ ]^v]_v : b2 \land$
Θ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ $ce2[z::=[\ x\ ]^v]_v : b2$ **using** *wfC-elims*(*3*)
beq **by** *metis*

from *xf* **have** Θ; *ℬ*; *GNil* ⊢ ⦃ *z'* : *b-of t* | [[*z'*]^v]^{ce} == [*v*]^{ce} ⦄ ≲ ⦃ *z* : *b-of t* | *ce1* == *ce2* ⦄
proof
show ‹ Θ; *ℬ*; *GNil* $\vdash_{wf}$ ⦃ *z'* : *b-of t* | [[*z'*]^v]^{ce} == [*v*]^{ce} ⦄ › **using** *b-of.simps assms infer-v-wf t* ∗ **by** *auto*
show ‹ Θ; *ℬ*; *GNil* $\vdash_{wf}$ ⦃ *z* : *b-of t* | *ce1* == *ce2* ⦄ › **using** *beq assms* **by** *auto*
have ‹Θ; *ℬ*; (*x*, *b-of t*, ([[*x*]^v]^{ce} == [*v*]^{ce})) $\#_\Gamma$ *GNil* ⊨ (*ce1*[*z*::=[*x*]^v]$_v$ == *ce2*[*z*::=[*x*]^v]$_v$) ›
proof(*rule valid-ce-eq*)
show ‹*ce1*[*z*::=[*x*]^v]$_v$ = *ce2*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_{cev}$› **proof** −
have *atom z* ♯ *ce1* **using** *assms fresh-def x-not-in-b-set* **by** *fast*
hence *ce1*[*z*::=[*x*]^v]$_v$ = *ce1*
using *forget-subst-v* **by** *auto*
also have ... = *ce2*[*z*::=*v*]$_{cev}$ **using** *assms* **by** *auto*
also have ... = *ce2*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_{cev}$ **proof** −
have *atom x* ♯ *ce2* **using** *xf fresh-prodN c.fresh* **by** *metis*
thus *?thesis* **using** *subst-v-simple-commute subst-v-ce-def* **by** *simp*
qed
finally show *?thesis* **by** *auto*
qed
show ‹ Θ; *ℬ*; *GNil* $\vdash_{wf}$ *v* : *b-of t* › **using** *infer-v-wf t* **by** *simp*
show ‹ Θ; *ℬ*; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce2*[*z*::=[*x*]^v]$_v$: *b2* › **using** *b2* **by** *auto*

have Θ; *ℬ*; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce1*[*z*::=[*x*]^v]$_v$: *b2* **using** *b2* **by** *auto*
moreover have *atom x* ♯ *ce1*[*z*::=[*x*]^v]$_v$
using *fresh-subst-v-if assms fresh-def*
using ‹Θ; *ℬ*; *GNil* $\vdash_{wf}$ *v* : *b-of t*› ‹*ce1*[*z*::=[*x*]^v]$_v$ = *ce2*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_{cev}$›
fresh-GNil subst-v-ce-def wfV-x-fresh **by** *auto*
ultimately show ‹ Θ; *ℬ*; *GNil* $\vdash_{wf}$ *ce1*[*z*::=[*x*]^v]$_v$: *b2* › **using**
wf-restrict(*8*) **by** *force*
qed
moreover have *v*: *v*[*z'*::=[*x*]^v]$_{vv}$ = *v*
using *forget-subst assms infer-v-wf wfV-supp x-not-in-b-set*
by (*simp add*: *local*.∗)
ultimately show Θ; *ℬ*; (*x*, *b-of t*, ([[*z'*]^v]^{ce} == [*v*]^{ce})[*z'*::=[*x*]^v]$_v$) $\#_\Gamma$ *GNil* ⊨ (*ce1* == *ce2*)[*z*::=[*x*]^v]$_v$
unfolding *subst-cv.simps subst-v-c-def subst-cev.simps subst-vv.simps*
using *subst-v-ce-def* **by** *simp*
qed
thus *?thesis* **using** *b-of.simps assms* ∗ *check-v-subtypeI t b-of.simps subtype-eq-base2* **by** *metis*
qed

end

declare *freshers*[*simp del*]

Chapter 13

Context Subtyping Lemmas

Lemmas allowing us to replace the type of a variable in the context with a subtype and have the judgement remain valid. Also known as narrowing.

13.1 Replace or exchange type of variable in a context

Because the G-context is extended by the statements like let, we will need a generalised substitution lemma for statements. For this we setup a function that replaces in G (rig) for a particular x the constraint for it. We also define a well-formedness relation for RIGs that ensures that each new constraint implies the old one

nominal-function *replace-in-g-many* :: $\Gamma \Rightarrow (x*c)$ *list* $\Rightarrow \Gamma$ **where**
  replace-in-g-many G xcs = List.foldr $(\lambda(x,c)\ G.\ G[x \longmapsto c])$ *xcs G*
  by(*auto,simp add*: *eqvt-def replace-in-g-many-graph-aux-def*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *replace-in-g-subtyped* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (x*c)$ *list* $\Rightarrow \Gamma \Rightarrow$ *bool* (‹ - ; - ⊢ - ⟨ - ⟩ ⇝ -› [*100,50,50*] *50*) **where**
  replace-in-g-subtyped-nilI: $\Theta;\ \mathcal{B} \vdash G\ \langle\ []\ \rangle \rightsquigarrow G$
| *replace-in-g-subtyped-consI*: ⟦
    Some $(b,c') =$ *lookup G x* ;
    $\Theta;\ \mathcal{B};\ G \vdash_{wf}\ c$;
    $\Theta;\ \mathcal{B};\ G[x \longmapsto c] \models c'$;
    $\Theta;\ \mathcal{B} \vdash G[x \longmapsto c]\ \langle\ xcs\ \rangle \rightsquigarrow G'$; $x \notin$ *fst ‘ set xcs* ⟧ $\Longrightarrow$
    $\Theta;\ \mathcal{B} \vdash G\ \langle\ (x,c)\#xcs\ \rangle \rightsquigarrow G'$
equivariance *replace-in-g-subtyped*
nominal-inductive *replace-in-g-subtyped* **.**

inductive-cases *replace-in-g-subtyped-elims*[*elim*!]:
  $\Theta;\ \mathcal{B} \vdash G\ \langle\ []\ \rangle \rightsquigarrow G'$
  $\Theta;\ \mathcal{B} \vdash ((x,b,c)\#_\Gamma\Gamma\ G)\ \langle\ acs\ \rangle \rightsquigarrow ((x,b,c)\#_\Gamma G')$
  $\Theta;\ \mathcal{B} \vdash G'\ \langle\ (x,c)\#\ acs\ \rangle \rightsquigarrow G$

lemma *rigs-atom-dom-eq*:
  assumes $\Theta;\ \mathcal{B} \vdash G\ \langle\ xcs\ \rangle \rightsquigarrow G'$
  shows *atom-dom G = atom-dom G'*
  using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)

case (*replace-in-g-subtyped-nilI G*)
then show *?case* by *simp*
next
case (*replace-in-g-subtyped-consI b c′ G x Θ ℬ c xcs G′*)
then show *?case* using *rig-dom-eq atom-dom.simps dom.simps* by *simp*
qed

lemma *replace-in-g-wfG*:
assumes Θ; ℬ ⊢ G ⟨ *xcs* ⟩ ⇝ G′ **and** *wfG* Θ ℬ G
shows *wfG* Θ ℬ G′
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)
case (*replace-in-g-subtyped-nilI* Θ G)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c′ G x* Θ *c xcs G′*)
then show *?case* **using** *valid-g-wf* **by** *auto*
qed

lemma *wfD-rig-single*:
fixes Δ::Δ **and** *x*::*x* **and** *c*::*c* **and** *G*::Γ
assumes Θ; ℬ; G $\vdash_{wf}$ Δ **and** *wfG* Θ ℬ (G[x⟼c])
shows Θ; ℬ; G[x⟼c] $\vdash_{wf}$ Δ
proof(*cases atom x* ∈ *atom-dom G*)
case *False*
hence (G[x⟼c]) = G **using** *assms replace-in-g-forget wfX-wfY* **by** *metis*
then show *?thesis* **using** *assms* **by** *auto*
next
case *True*
then obtain *G1 G2 b c′* **where** ∗: G=G1@(x,b,c′)$\#_\Gamma$G2 **using** *split-G* **by** *fastforce*
hence ∗∗: (G[x⟼c]) = G1@(x,b,c)$\#_\Gamma$G2 **using** *replace-in-g-inside wfD-wf assms wfD-wf* **by** *metis*

hence *wfG* Θ ℬ ((x,b,c)$\#_\Gamma$G2) **using** *wfG-suffix assms* **by** *auto*
hence Θ; ℬ; (x, b, TRUE) $\#_\Gamma$ G2 $\vdash_{wf}$ c **using** *wfG-elim2* **by** *auto*

thus *?thesis* **using** *wf-replace-inside1 assms* ∗ ∗∗
by (*simp add*: *wf-replace-inside2*(*6*))
qed

lemma *wfD-rig*:
assumes Θ; ℬ ⊢ G ⟨ *xcs* ⟩ ⇝ G′ **and** *wfD* Θ ℬ G Δ
shows *wfD* Θ ℬ G′ Δ
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)
case (*replace-in-g-subtyped-nilI* Θ G)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c′ G x* Θ *c xcs G′*)
then show *?case* **using** *wfD-rig-single valid.simps wfC-wf* **by** *auto*
qed

lemma *replace-in-g-fresh*:
fixes *x*::*x*
assumes Θ; ℬ ⊢ Γ ⟨ *xcs* ⟩ ⇝ Γ′ **and** *wfG* Θ ℬ Γ **and** *wfG* Θ ℬ Γ′ **and** *atom x* ♯ Γ

shows *atom x* ♯ Γ′
using *wfG-dom-supp assms fresh-def rigs-atom-dom-eq* **by** *metis*

lemma *replace-in-g-fresh1*:
fixes *x*::*x*
assumes Θ; ℬ ⊢ Γ ⟨ *xcs* ⟩ ⇝ Γ′ **and** *wfG* Θ ℬ Γ **and** *atom x* ♯ Γ
shows *atom x* ♯ Γ′
proof −
have *wfG* Θ ℬ Γ′ **using** *replace-in-g-wfG assms* **by** *auto*
thus *?thesis* **using** *assms replace-in-g-fresh* **by** *metis*
qed

Wellscoping for an eXchange list

inductive *wsX*:: Γ ⇒ (*x*∗*c*) *list* ⇒ *bool* **where**
wsX-NilI: *wsX G* []
| *wsX-ConsI*: ⟦ *wsX G xcs* ; *atom x* ∈ *atom-dom G* ; *x* ∉ *fst* ‘ *set xcs* ⟧ ⟹ *wsX G* ((*x*,*c*)#*xcs*)
equivariance *wsX*
nominal-inductive *wsX* **.**

lemma *wsX-if1*:
assumes *wsX G xcs*
shows ((*atom* ‘ *fst* ‘ *set xcs*) ⊆ *atom-dom G*) ∧ *List.distinct* (*List.map fst xcs*)
using *assms* **by**(*induct rule*: *wsX.induct*,*force*+)

lemma *wsX-if2*:
assumes ((*atom* ‘ *fst* ‘ *set xcs*) ⊆ *atom-dom G*) ∧ *List.distinct* (*List.map fst xcs*)
shows *wsX G xcs*
using *assms* **proof**(*induct xcs*)
case *Nil*
then show *?case* **using** *wsX-NilI* **by** *fast*
next
case (*Cons a xcs*)
then obtain *x* **and** *c* **where** *xc*: *a*=(*x*,*c*) **by** *force*
have *wsX G xcs* **proof** −
have *distinct* (*map fst xcs*) **using** *Cons* **by** *force*
moreover have *atom* ‘ *fst* ‘ *set xcs* ⊆ *atom-dom G* **using** *Cons* **by** *simp*
ultimately show *?thesis* **using** *Cons* **by** *fast*
qed
moreover have *atom x* ∈ *atom-dom G* **using** *Cons xc*
by *simp*
moreover have *x* ∉ *fst* ‘ *set xcs* **using** *Cons xc*
by *simp*
ultimately show *?case* **using** *wsX-ConsI xc* **by** *blast*
qed

lemma *wsX-iff*:
wsX G xcs = (((*atom* ‘ *fst* ‘ *set xcs*) ⊆ *atom-dom G*) ∧ *List.distinct* (*List.map fst xcs*))
using *wsX-if1 wsX-if2* **by** *meson*

inductive-cases *wsX-elims*[*elim*!]:
wsX G []
wsX G ((*x*,*c*)#*xcs*)

lemma *wsX-cons*:
assumes *wsX* Γ *xcs* **and** $x \notin$ *fst* ' *set xcs*
shows *wsX* ((*x*, *b*, *c1*) $\#_\Gamma$ Γ) ((*x*, *c2*) # *xcs*)
using *assms* **proof**(*induct* Γ)
case *GNil*
then show *?case* **using** *atom-dom.simps wsX-iff* **by** *auto*
next
case (*GCons xbc* Γ)
obtain x' **and** b' **and** c' **where** *xbc*: $xbc = (x', b', c')$ **using** *prod-cases3* **by** *blast*
then have *atom* ' *fst* ' *set xcs* ⊆ *atom-dom* (*xbc* $\#_\Gamma$ Γ) ∧ *distinct* (*map fst xcs*)
using *GCons.prems*(*1*) *wsX-iff* **by** *blast*
then have *wsX* ((*x*, *b*, *c1*) $\#_\Gamma$ *xbc* $\#_\Gamma$ Γ) *xcs*
by (*simp add*: *Un-commute subset-Un-eq wsX-if2*)
then show *?case* **by** (*simp add*: *GCons.prems*(*2*) *wsX-ConsI*)
qed

lemma *wsX-cons2*:
assumes *wsX* Γ *xcs* **and** $x \notin$ *fst* ' *set xcs*
shows *wsX* ((*x*, *b*, *c1*) $\#_\Gamma$ Γ) *xcs*
using *assms* **proof**(*induct* Γ)
case *GNil*
then show *?case* **using** *atom-dom.simps wsX-iff* **by** *auto*
next
case (*GCons xbc* Γ)
obtain x' **and** b' **and** c' **where** *xbc*: $xbc = (x', b', c')$ **using** *prod-cases3* **by** *blast*
then have *atom* ' *fst* ' *set xcs* ⊆ *atom-dom* (*xbc* $\#_\Gamma$ Γ) ∧ *distinct* (*map fst xcs*)
using *GCons.prems*(*1*) *wsX-iff* **by** *blast* **then show** *?case* **by** (*simp add*: *Un-commute subset-Un-eq wsX-if2*)
qed

lemma *wsX-cons3*:
assumes *wsX* Γ *xcs*
shows *wsX* ((*x*, *b*, *c1*) $\#_\Gamma$ Γ) *xcs*
using *assms* **proof**(*induct* Γ)
case *GNil*
then show *?case* **using** *atom-dom.simps wsX-iff* **by** *auto*
next
case (*GCons xbc* Γ)
obtain x' **and** b' **and** c' **where** *xbc*: $xbc = (x', b', c')$ **using** *prod-cases3* **by** *blast*
then have *atom* ' *fst* ' *set xcs* ⊆ *atom-dom* (*xbc* $\#_\Gamma$ Γ) ∧ *distinct* (*map fst xcs*)
using *GCons.prems*(*1*) *wsX-iff* **by** *blast* **then show** *?case* **by** (*simp add*: *Un-commute subset-Un-eq wsX-if2*)
qed

lemma *wsX-fresh*:
assumes *wsX G xcs* **and** *atom x* ♯ *G* **and** *wfG* Θ $\mathcal{B}$ *G*
shows $x \notin$ *fst* ' *set xcs*
proof −
have *atom x* ∉ *atom-dom G* **using** *assms*
using *fresh-def wfG-dom-supp* **by** *auto*
thus *?thesis* **using** *wsX-iff assms* **by** *blast*

qed

lemma *replace-in-g-dist*:
  assumes $x' \neq x$
  shows *replace-in-g* $((x,\ b,c)\ \#_\Gamma\ G)\ x'\ c'' = ((x,\ b,c)\ \#_\Gamma\ (\textit{replace-in-g}\ G\ x'\ c''))$ **using** *replace-in-g.simps assms* **by** *presburger*

lemma *wfG-replace-inside-rig*:
  fixes $c''::c$
  assumes ‹Θ; $\mathcal{B} \vdash_{wf}\ G[x' \longmapsto c'']$› ‹Θ; $\mathcal{B} \vdash_{wf}\ (x,\ b,\ c)\ \#_\Gamma\ G$ ›
  shows Θ; $\mathcal{B} \vdash_{wf}\ (x,\ b,\ c)\ \#_\Gamma\ G[x' \longmapsto c'']$
proof(*rule wfG-consI*)

  have *wfG* Θ $\mathcal{B}$ *G* **using** *wfG-cons assms* **by** *auto*

  show ∗:Θ; $\mathcal{B} \vdash_{wf}\ G[x' \longmapsto c'']$ **using** *assms* **by** *auto*
  show *atom x* ♯ $G[x' \longmapsto c'']$ **using** *replace-in-g-fresh-single*[*OF* ∗] *assms wfG-elims assms* **by** *metis*
  show ∗∗:Θ; $\mathcal{B} \vdash_{wf}\ b$ **using** *wfG-elim2 assms* **by** *auto*
  show Θ; $\mathcal{B}$; $(x,\ b,\ TRUE)\ \#_\Gamma\ G[x' \longmapsto c'']\ \vdash_{wf}\ c$
  proof(*cases atom* $x' \notin$ *atom-dom G*)
    case *True*
    hence $G = G[x' \longmapsto c'']$ **using** *replace-in-g-forget* ‹*wfG* Θ $\mathcal{B}$ *G*› **by** *auto*
    thus *?thesis* **using** *assms wfG-wfC* **by** *auto*
  next
    case *False*
    then obtain *G1 G2 b' c'* **where** ∗∗:$G = G1 @ (x',b',c') \#_\Gamma G2$
      using *split-G* **by** *fastforce*
    hence ∗∗∗: $(G[x' \longmapsto c'']) = G1 @ (x',b',c'') \#_\Gamma G2$
      using *replace-in-g-inside* ‹*wfG* Θ $\mathcal{B}$ *G* › **by** *metis*
    hence Θ; $\mathcal{B}$; $(x,\ b,\ TRUE)\ \#_\Gamma\ G1 @ (x',b',c') \#_\Gamma G2\ \vdash_{wf}\ c$ **using** ∗ ∗∗ *assms wfG-wfC* **by** *auto*
    hence Θ; $\mathcal{B}$; $(x,\ b,\ TRUE)\ \#_\Gamma\ G1 @ (x',b',c'') \#_\Gamma G2\ \vdash_{wf}\ c$ **using** ∗ ∗∗∗ *wf-replace-inside assms*
      by (*metis* ∗∗ *append-g.simps*(*2*) *wfG-elim2 wfG-suffix*)
    thus *?thesis* **using** ∗∗ ∗ ∗∗∗ **by** *auto*
  qed
qed

lemma *replace-in-g-valid-weakening*:
  assumes Θ; $\mathcal{B}$; $\Gamma[x' \longmapsto c''] \models c'$ **and** $x' \neq x$ **and** Θ; $\mathcal{B} \vdash_{wf}\ (x,\ b,\ c)\ \#_\Gamma\ \Gamma[x' \longmapsto c'']$
  shows Θ; $\mathcal{B}$; $((x,\ b,c)\ \#_\Gamma\ \Gamma)[x' \longmapsto\ c'']\ \models c'$
  apply(*subst replace-in-g-dist*,*simp add*: *assms*,*rule valid-weakening*)
  using *assms* **by** *auto*+

lemma *replace-in-g-subtyped-cons*:
  assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ *G xcs G'* **and** *wfG* Θ $\mathcal{B}$ $((x,b,c) \#_\Gamma G)$
  shows $x \notin$ *fst ‘ set xcs* ⟹ *replace-in-g-subtyped* Θ $\mathcal{B}$ $((x,b,c) \#_\Gamma G)$ *xcs* $((x,b,c) \#_\Gamma G')$
  using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)
  case (*replace-in-g-subtyped-nilI G*)
  then show *?case*
    by (*simp add*: *replace-in-g-subtyped.replace-in-g-subtyped-nilI*)
next
  case (*replace-in-g-subtyped-consI* $b'\ c'\ G\ x'$ Θ $\mathcal{B}$ $c''\ xcs'\ G'$)
  hence Θ; $\mathcal{B} \vdash_{wf}\ G[x' \longmapsto c'']$ **using** *valid.simps wfC-wf* **by** *auto*

show *?case* **proof**(*rule replace-in-g-subtyped.replace-in-g-subtyped-consI*)
show *Some* $(b', c') = lookup\ ((x, b,c) \#_\Gamma G)\ x'$ **using** *lookup.simps*
fst-conv image-iff Γ*-set-intros surj-pair replace-in-g-subtyped-consI* **by** *force*
show *wbc*: $\Theta; \mathcal{B}; (x, b, c) \#_\Gamma G \vdash_{wf} c''$ **using** *wf-weakening* ‹ $\Theta; \mathcal{B}; G \vdash_{wf} c''$› ‹$\Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_\Gamma G$ › **by** *fastforce*
have $x' \neq x$ **using** *replace-in-g-subtyped-consI* **by** *auto*
have *wbc1*: $\Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_\Gamma G[x' \longmapsto c'']$ **proof** −
have $(x, b, c) \#_\Gamma G[x' \longmapsto c''] = ((x, b, c) \#_\Gamma G)[x' \longmapsto c'']$ **using** ‹$x' \neq x$› **using** *replace-in-g.simps* **by** *auto*
thus *?thesis* **using** *wfG-replace-inside-rig* ‹$\Theta; \mathcal{B} \vdash_{wf} G[x' \longmapsto c'']$› ‹$\Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_\Gamma G$ › **by** *fastforce*
qed
show $*$: $\Theta; \mathcal{B};$ *replace-in-g* $((x, b,c) \#_\Gamma G)\ x'\ c'' \models c'$
proof −
have $\Theta; \mathcal{B}; G[x' \longmapsto c''] \models c'$ **using** *replace-in-g-subtyped-consI* **by** *auto*
thus *?thesis* **using** *replace-in-g-valid-weakening wbc1* ‹$x' \neq x$› **by** *auto*
qed

show *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ (*replace-in-g* $((x, b,c) \#_\Gamma G)\ x'\ c''$) xcs' $((x, b,c) \#_\Gamma G')$
using *replace-in-g-subtyped-consI wbc1* **by** *auto*
show $x' \notin$ *fst* ‘ *set xcs'*
using *replace-in-g-subtyped-consI* **by** *linarith*
qed
qed

lemma *replace-in-g-split*:
fixes G::Γ
assumes $\Gamma =$ *replace-in-g* $\Gamma'\ x\ c$ **and** $\Gamma' = G'@(x,b,c')\#_\Gamma G$ **and** *wfG* $\Theta\ \mathcal{B}\ \Gamma'$
shows $\Gamma = G'@(x,b,c)\#_\Gamma G$
using *assms* **proof**(*induct* G' *arbitrary*: $G\ \Gamma\ \Gamma'$ *rule*: Γ*-induct*)
case *GNil*
then show *?case* **by** *simp*
next
case (*GCons x1 b1 c1* Γ*1*)
hence $x1 \neq x$
using *wfG-cons-fresh2*[*of* $\Theta\ \mathcal{B}$ *x1 b1 c1* Γ*1 x b*]
using *GCons.prems*(*2*) *GCons.prems*(*3*) *append-g.simps*(*2*) **by** *auto*
moreover hence $*$: $\Theta; \mathcal{B} \vdash_{wf} (\Gamma 1\ @\ (x, b, c') \#_\Gamma G)$ **using** *GCons append-g.simps wfG-elims* **by** *metis*
moreover hence *replace-in-g* $(\Gamma 1\ @\ (x, b, c') \#_\Gamma G)\ x\ c = \Gamma 1\ @\ (x, b, c) \#_\Gamma G$ **using** *GCons replace-in-g-inside*[*OF* $*$, *of c*] **by** *auto*

ultimately show *?case* **using** *replace-in-g.simps*(*2*)[*of x1 b1 c1* $\Gamma 1\ @\ (x, b, c') \#_\Gamma G\ x\ c$] *GCons*
by (*simp add*: *GCons.prems*(*1*) *GCons.prems*(*2*))
qed

lemma *replace-in-g-subtyped-split0*:
fixes G::Γ
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'[(x,c)]\ \Gamma$ **and** $\Gamma' = G'@(x,b,c')\#_\Gamma G$ **and** *wfG* $\Theta\ \mathcal{B}\ \Gamma'$
shows $\Gamma = G'@(x,b,c)\#_\Gamma G$
proof −

have Γ = *replace-in-g* Γ' *x c* **using** *assms replace-in-g-subtyped.simps*
by (*metis Pair-inject list.distinct*(*1*) *list.inject*)
thus *?thesis* **using** *assms replace-in-g-split* **by** *blast*
qed

lemma *replace-in-g-subtyped-split*:
assumes *Some* (*b*, *c'*) = *lookup G x* **and** Θ; $\mathcal{B}$; *replace-in-g G x c* $\models$ *c'* **and** *wfG* Θ $\mathcal{B}$ *G*
shows $\exists$ Γ Γ'. $G = \Gamma'@(x,b,c')\#_\Gamma\Gamma \wedge \Theta$; $\mathcal{B}$; $\Gamma'@(x,b,c)\#_\Gamma\Gamma \models c'$
proof −
obtain Γ **and** Γ' **where** $G = \Gamma'@(x,b,c')\#_\Gamma\Gamma$ **using** *assms lookup-split* **by** *blast*
moreover hence *replace-in-g G x c* = $\Gamma'@(x,b,c)\#_\Gamma\Gamma$ **using** *replace-in-g-split assms* **by** *blast*
ultimately show *?thesis* **by** (*metis assms*(*2*))
qed

13.2 Validity and Subtyping

lemma *wfC-replace-in-g*:
fixes *c*::*c* **and** *c0*::*c*
assumes Θ; $\mathcal{B}$; $\Gamma'@(x,b,c0')\#_\Gamma\Gamma \vdash_{wf} c$ **and** Θ; $\mathcal{B}$; $(x,b,TRUE)\#_\Gamma\Gamma \vdash_{wf} c0$
shows Θ; $\mathcal{B}$; Γ' @ (*x*, *b*, *c0*) $\#_\Gamma$ $\Gamma \vdash_{wf} c$
using *wf-replace-inside1*(*2*) *assms* **by** *auto*

lemma *ctx-subtype-valid*:
assumes Θ; $\mathcal{B}$; $\Gamma'@(x,b,c0')\#_\Gamma\Gamma \models c$ **and**
Θ; $\mathcal{B}$; $\Gamma'@(x,b,c0)\#_\Gamma\Gamma \models c0'$
shows Θ; $\mathcal{B}$; $\Gamma'@(x,b,c0)\#_\Gamma\Gamma \models c$
proof(*rule validI*)
show Θ; $\mathcal{B}$; Γ' @ (*x*, *b*, *c0*) $\#_\Gamma$ $\Gamma \vdash_{wf} c$ **proof** −
have Θ; $\mathcal{B}$; $\Gamma'@(x,b,c0')\#_\Gamma\Gamma \vdash_{wf} c$ **using** *valid.simps assms* **by** *auto*
moreover have Θ; $\mathcal{B}$; $(x,b,TRUE)\#_\Gamma\Gamma \vdash_{wf} c0$ **proof** −
have *wfG* Θ $\mathcal{B}$ ($\Gamma'@(x,b,c0)\#_\Gamma\Gamma$) **using** *assms valid.simps wfC-wf* **by** *auto*
hence *wfG* Θ $\mathcal{B}$ ($(x,b,c0)\#_\Gamma\Gamma$) **using** *wfG-suffix* **by** *auto*
thus *?thesis* **using** *wfG-wfC* **by** *auto*
qed
ultimately show *?thesis* **using** *assms wfC-replace-in-g* **by** *auto*
qed

show $\forall$ *i*. *wfI* Θ (Γ' @ (*x*, *b*, *c0*) $\#_\Gamma$ Γ) *i* $\wedge$ *is-satis-g i* (Γ' @ (*x*, *b*, *c0*) $\#_\Gamma$ Γ) $\longrightarrow$ *is-satis i c*
proof(*rule*,*rule*)
fix *i*
assume $*$: *wfI* Θ (Γ' @ (*x*, *b*, *c0*) $\#_\Gamma$ Γ) *i* $\wedge$ *is-satis-g i* (Γ' @ (*x*, *b*, *c0*) $\#_\Gamma$ Γ)

hence *is-satis-g i* (Γ'@(*x*, *b*, *c0*) $\#_\Gamma$ Γ) $\wedge$ *wfI* Θ (Γ'@(*x*, *b*, *c0*) $\#_\Gamma$ Γ) *i* **using** *is-satis-g-append wfI-suffix* **by** *metis*
moreover hence *is-satis i c0'* **using** *valid.simps assms* **by** *presburger*

moreover have *is-satis-g i* Γ' **using** *is-satis-g-append* $*$ **by** *simp*
ultimately have *is-satis-g i* (Γ' @ (*x*, *b*, *c0'*) $\#_\Gamma$ Γ) **using** *is-satis-g-append* **by** *simp*
moreover have *wfI* Θ (Γ' @ (*x*, *b*, *c0'*) $\#_\Gamma$ Γ) *i* **using** *wfI-def wfI-suffix* $*$ *wfI-def wfI-replace-inside* **by** *metis*
ultimately show *is-satis i c* **using** *assms valid.simps* **by** *metis*

qed
qed

lemma *ctx-subtype-subtype*:
fixes Γ::Γ
shows Θ; $\mathcal{B}$; *G* ⊢ *t1* $\lesssim$ *t2* ⟹ *G* = Γ′@(*x*,*b0*,*c0*′)#$_\Gamma$Γ ⟹ Θ; $\mathcal{B}$; Γ′@(*x*,*b0*,*c0*)#$_\Gamma$Γ ⊨ *c0*′ ⟹ Θ; $\mathcal{B}$; Γ′@(*x*,*b0*,*c0*)#$_\Gamma$Γ ⊢ *t1* $\lesssim$ *t2*
proof(*nominal-induct avoiding*: *c0 rule*: *subtype.strong-induct*)

case (*subtype-baseI x*′ Θ $\mathcal{B}$ Γ′′ *z c z*′ *c*′ *b*)
let *?*Γ*c0* = Γ′@(*x*,*b0*,*c0*)#$_\Gamma$Γ
have *wb1*: *wfG* Θ $\mathcal{B}$ *?*Γ*c0* **using** *valid.simps wfC-wf subtype-baseI* **by** *metis*
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ ⊢$_{wf}$ ⦃ *z* : *b* | *c* ⦄ › **using** *wfT-replace-inside2*[*OF* - *wb1*] *subtype-baseI* **by** *metis*
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ ⊢$_{wf}$ ⦃ *z*′ : *b* | *c*′ ⦄ › **using** *wfT-replace-inside2*[*OF* - *wb1*] *subtype-baseI* **by** *metis*
have *atom x*′ ♯ Γ′ @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ **using** *fresh-prodN subtype-baseI fresh-replace-inside wb1 subtype-wf wfX-wfY* **by** *metis*
thus ‹*atom x*′ ♯ (Θ, $\mathcal{B}$, Γ′ @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ, *z*, *c* , *z*′ , *c*′)› **using** *subtype-baseI fresh-prodN* **by** *metis*
have Θ; $\mathcal{B}$; ((*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′) @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ ⊨ *c*′[*z*′::=*V-var x*′]$_v$ **proof**(*rule ctx-subtype-valid*)
show *1*: ‹Θ; $\mathcal{B}$; ((*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′) @ (*x*, *b0*, *c0*′) #$_\Gamma$ Γ ⊨ *c*′[*z*′::=*V-var x*′]$_v$ ›
using *subtype-baseI append-g.simps subst-defs* **by** *metis*
have ∗:Θ; $\mathcal{B}$ ⊢$_{wf}$ ((*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′) @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ **proof**(*rule wfG-replace-inside2*)
show Θ; $\mathcal{B}$ ⊢$_{wf}$ ((*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′) @ (*x*, *b0*, *c0*′) #$_\Gamma$ Γ
using ∗ *valid-wf-all wfC-wf 1 append-g.simps* **by** *metis*
show Θ; $\mathcal{B}$ ⊢$_{wf}$ (*x*, *b0*, *c0*) #$_\Gamma$ Γ **using** *wfG-suffix wb1* **by** *auto*
qed
moreover have *toSet* (Γ′ @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ) ⊆ *toSet* (((*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′) @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ) **using** *toSet.simps append-g.simps* **by** *auto*
ultimately show ‹Θ; $\mathcal{B}$; ((*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′) @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ ⊨ *c0*′ › **using** *valid-weakening subtype-baseI* ∗ **by** *blast*
qed
thus ‹Θ; $\mathcal{B}$; (*x*′, *b*, *c*[*z*::=*V-var x*′]$_v$) #$_\Gamma$ Γ′ @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ ⊨ *c*′[*z*′::=*V-var x*′]$_v$ › **using** *append-g.simps subst-defs* **by** *simp*
qed
qed

lemma *ctx-subtype-subtype-rig*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ′ [(*x*,*c0*)] Γ **and** Θ; $\mathcal{B}$; Γ′ ⊢ *t1* $\lesssim$ *t2*
shows Θ; $\mathcal{B}$; Γ ⊢ *t1* $\lesssim$ *t2*
proof −
have *wf*: *wfG* Θ $\mathcal{B}$ Γ′ **using** *subtype-g-wf assms* **by** *auto*
obtain *b* **and** *c0*′ **where** *Some* (*b*, *c0*′) = *lookup* Γ′ *x* ∧ (Θ; $\mathcal{B}$; *replace-in-g* Γ′ *x c0* ⊨ *c0*′) **using** *replace-in-g-subtyped.simps*[*of* Θ $\mathcal{B}$ Γ′ [(*x*, *c0*)] Γ] *assms*(*1*)

by (*metis fst-conv list.inject list.set-intros*(*1*) *list.simps*(*15*) *not-Cons-self2 old.prod.exhaust prod.inject set-ConsD surj-pair*)
moreover then obtain *G* **and** *G*′ **where** ∗: Γ′ = *G*′@(*x*,*b*,*c0*′)#$_\Gamma$*G* ∧ Θ; $\mathcal{B}$; *G*′@(*x*,*b*,*c0*)#$_\Gamma$*G* ⊨ *c0*′

using *replace-in-g-subtyped-split*[*of b c0′* Γ′ *x* Θ *ℬ c0*] *wf* **by** *metis*

ultimately show *?thesis* **using** *ctx-subtype-subtype*
assms(*1*) *assms*(*2*) *replace-in-g-subtyped-split0 subtype-g-wf*
by (*metis* (*no-types*, *lifting*) *local.wf replace-in-g-split*)
qed

We now prove versions of the *ctx-subtype* lemmas above using *replace-in-g*. First we do case where the replace is just for a single variable (indicated by suffix rig) and then the general case for multiple replacements (indicated by suffix rigs)

lemma *ctx-subtype-subtype-rigs*:
assumes *replace-in-g-subtyped* Θ *ℬ* Γ′ *xcs* Γ **and** Θ; *ℬ*; Γ′ ⊢ *t1* ≲ *t2*
shows Θ; *ℬ*; Γ ⊢ *t1* ≲ *t2*
using *assms* **proof**(*induct xcs arbitrary*: Γ Γ′)
case *Nil*
moreover have Γ′ = Γ **using** *replace-in-g-subtyped-nilI*
using *calculation*(*1*) **by** *blast*
ultimately show *?case* **by** *auto*
next
case (*Cons a xcs*)

then obtain *x* **and** *c* **where** *a*=(*x*,*c*) **by** *fastforce*
then obtain *b* **and** *c′* **where** *bc*: *Some* (*b*, *c′*) = *lookup* Γ′ *x* ∧
replace-in-g-subtyped Θ *ℬ* (*replace-in-g* Γ′ *x c*) *xcs* Γ ∧ Θ; *ℬ*; Γ′ $\vdash_{wf}$ *c* ∧
x ∉ *fst* ‘ *set xcs* ∧ Θ; *ℬ*; (*replace-in-g* Γ′ *x c*) ⊨ *c′* **using** *replace-in-g-subtyped-elims*(*3*)[*of* Θ *ℬ* Γ′ *x c xcs* Γ] *Cons*
by (*metis valid.simps*)

hence ∗: *replace-in-g-subtyped* Θ *ℬ* Γ′ [(*x*,*c*)] (*replace-in-g* Γ′ *x c*) **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct*(*1*) *list.set-cases replace-in-g-subtyped-nilI*)

hence Θ; *ℬ*; (*replace-in-g* Γ′ *x c*) ⊢ *t1* ≲ *t2*
using *ctx-subtype-subtype-rig* ∗ *assms Cons.prems*(*2*) **by** *auto*

moreover have *replace-in-g-subtyped* Θ *ℬ* (*replace-in-g* Γ′ *x c*) *xcs* Γ **using** *Cons*
using *bc* **by** *blast*

ultimately show *?case* **using** *Cons* **by** *blast*
qed

lemma *replace-in-g-inside-valid*:
assumes *replace-in-g-subtyped* Θ *ℬ* Γ′ [(*x*,*c0*)] Γ **and** *wfG* Θ *ℬ* Γ′
shows ∃ *b c0′ G G′*. Γ′ = *G′* @ (*x*,*b*,*c0′*)$\#_\Gamma$*G* ∧ Γ = *G′* @ (*x*,*b*,*c0*)$\#_\Gamma$*G* ∧ Θ; *ℬ*; *G′*@ (*x*,*b*,*c0*)$\#_\Gamma$*G* ⊨ *c0′*
proof −
obtain *b* **and** *c0′* **where** *bc*: *Some* (*b*, *c0′*) = *lookup* Γ′ *x* ∧ Θ; *ℬ*; *replace-in-g* Γ′ *x c0* ⊨ *c0′* **using**
replace-in-g-subtyped.simps[*of* Θ *ℬ* Γ′ [(*x*, *c0*)] Γ] *assms*(*1*)
by (*metis fst-conv list.inject list.set-intros*(*1*) *list.simps*(*15*) *not-Cons-self2 old.prod.exhaust prod.inject set-ConsD surj-pair*)
then obtain *G* **and** *G′* **where** ∗: Γ′ = *G′*@(*x*,*b*,*c0′*)$\#_\Gamma$*G* ∧ Θ; *ℬ*; *G′*@(*x*,*b*,*c0*)$\#_\Gamma$*G* ⊨ *c0′* **using** *replace-in-g-subtyped-split*[*of b c0′* Γ′ *x* Θ *ℬ c0*] *assms*
by *metis*

thus *?thesis* **using** *replace-in-g-inside bc*
using *assms(1) assms(2)* **by** *blast*
qed

lemma *replace-in-g-valid*:
assumes Θ; $\mathcal{B}$ ⊢ G ⟨ *xcs* ⟩ ⇝ G' **and** Θ; $\mathcal{B}$; G ⊨ c
shows ‹Θ; $\mathcal{B}$; G' ⊨ c ›
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.inducts*)
case (*replace-in-g-subtyped-nilI* Θ $\mathcal{B}$ *G*)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c1 G x* Θ $\mathcal{B}$ *c2 xcs G'*)
hence Θ; $\mathcal{B}$; *G*[*x*⟼*c2*] ⊨ *c*
by (*metis ctx-subtype-valid replace-in-g-split replace-in-g-subtyped-split valid-g-wf*)
then show *?case* **using** *replace-in-g-subtyped-consI* **by** *auto*
qed

13.3 Literals

13.4 Values

lemma *lookup-inside-unique-b*[*simp*]:
assumes Θ ; *B* $\vdash_{wf}$ (Γ'@(*x*,*b0*,*c0*)$\#_\Gamma$Γ) **and** Θ ; *B* $\vdash_{wf}$ (Γ'@(*x*,*b0*,*c0'*)$\#_\Gamma$Γ)
and *Some* (*b*, *c*) = *lookup* (Γ' @ (*x*, *b0*, *c0'*) $\#_\Gamma$ Γ) *y* **and** *Some* (*b0*,*c0*) = *lookup* (Γ'@((*x*,*b0*,*c0*))$\#_\Gamma$Γ) *x* **and** *x*=*y*
shows *b* = *b0*
by (*metis assms(2) assms(3) assms(5) lookup-inside-wf old.prod.exhaust option.inject prod.inject*)

lemma *ctx-subtype-v-aux*:
fixes *v*::*v*
assumes Θ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0'*)$\#_\Gamma$Γ) ⊢ *v* ⇒ *t1* **and** Θ; $\mathcal{B}$; Γ'@(*x*,*b0*,*c0*)$\#_\Gamma$Γ ⊨ *c0'*
shows Θ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0*)$\#_\Gamma$Γ) ⊢ *v* ⇒ *t1*
using *assms* **proof**(*nominal-induct* Γ'@((*x*,*b0*,*c0'*)$\#_\Gamma$Γ) *v t1 avoiding*: *c0* *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ *b c xa z*)
have *wf*:‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ › **using** *wfG-inside-valid2 infer-v-varI* **by** *metis*
have *xf1*:‹*atom z* ♯ *xa*› **using** *infer-v-varI* **by** *metis*
have *xf2*: ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ)› **apply**(*fresh-mth add*: *infer-v-varI*)
using *fresh-def infer-v-varI wfG-supp fresh-append-g fresh-GCons fresh-prodN* **by** *metis+*
show *?case* **proof** (*cases x*=*xa*)
case *True*
moreover have *b* = *b0* **using** *infer-v-varI True* **by** *simp*
moreover hence ‹*Some* (*b*, *c0*) = *lookup* (Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) *xa*› **using** *lookup-inside-wf*[*OF wf*] *infer-v-varI True* **by** *auto*
ultimately show *?thesis* **using** *wf xf1 xf2 Typing.infer-v-varI* **by** *metis*
next
case *False*
moreover hence ‹*Some* (*b*, *c*) = *lookup* (Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) *xa*› **using** *lookup-inside2 infer-v-varI* **by** *metis*
ultimately show *?thesis* **using** *wf xf1 xf2 Typing.infer-v-varI* **by** *simp*
qed
next
case (*infer-v-litI* Θ $\mathcal{B}$ *l* τ)

thus *?case* **using** *Typing.infer-v-litI wfG-inside-valid2* **by** *simp*
next
  case (*infer-v-pairI z v1 v2* Θ *B t1′ t2′ c0*)
  show *?case* **proof**
    show *atom z* ♯ (*v1*, *v2*) **using** *infer-v-pairI fresh-Pair* **by** *simp*
    show *atom z* ♯ (Θ, *B*, Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) **apply**(*fresh-mth add*: *infer-v-pairI*)
      using *fresh-def infer-v-pairI wfG-supp fresh-append-g fresh-GCons fresh-prodN* **by** *metis+*
    show Θ; *B*; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v1* ⇒ *t1′* **using** *infer-v-pairI* **by** *simp*
    show Θ; *B*; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v2* ⇒ *t2′* **using** *infer-v-pairI* **by** *simp*
  qed
next
  case (*infer-v-consI s dclist* Θ *dc tc B v tv z*)
  show *?case* **proof**
    show ‹*AF-typedef s dclist* ∈ *set* Θ› **using** *infer-v-consI* **by** *auto*
    show ‹(*dc*, *tc*) ∈ *set dclist*› **using** *infer-v-consI* **by** *auto*
    show ‹ Θ; *B*; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇒ *tv*› **using** *infer-v-consI* **by** *auto*
    show ‹Θ; *B*; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *tv* ≲ *tc*› **using** *infer-v-consI ctx-subtype-subtype* **by** *auto*
    show ‹*atom z* ♯ *v*› **using** *infer-v-consI* **by** *auto*
    show ‹*atom z* ♯ (Θ, *B*, Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ)› **apply**(*fresh-mth add*: *infer-v-consI*)
      using *fresh-def infer-v-consI wfG-supp fresh-append-g fresh-GCons fresh-prodN* **by** *metis+*
  qed
next
  case (*infer-v-conspI s bv dclist* Θ *dc tc B v tv b z*)
  show *?case* **proof**
    show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *infer-v-conspI* **by** *auto*
    show ‹(*dc*, *tc*) ∈ *set dclist*› **using** *infer-v-conspI* **by** *auto*
    show ‹ Θ; *B*; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇒ *tv*› **using** *infer-v-conspI* **by** *auto*
    show ‹Θ; *B*; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *tv* ≲ *tc*[*bv*::=*b*]$_{\tau b}$› **using** *infer-v-conspI ctx-subtype-subtype*
by *auto*
    show ‹*atom z* ♯ (Θ, *B*, Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ, *v*, *b*)› **apply**(*fresh-mth add*: *infer-v-conspI*)
      using *fresh-def infer-v-conspI wfG-supp fresh-append-g fresh-GCons fresh-prodN* **by** *metis+*
    show ‹*atom bv* ♯ (Θ, *B*, Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ, *v*, *b*)› **apply**(*fresh-mth add*: *infer-v-conspI*)
      using *fresh-def infer-v-conspI wfG-supp fresh-append-g fresh-GCons fresh-prodN* **by** *metis+*
    show ‹ Θ; *B* $\vdash_{wf}$ *b* › **using** *infer-v-conspI* **by** *auto*
  qed
qed

lemma *ctx-subtype-v*:
  fixes *v*::*v*
  assumes Θ; *B*; Γ′@((*x*,*b0*,*c0′*)$\#_\Gamma$Γ) ⊢ *v* ⇒ *t1* **and** Θ; *B*; Γ′@(*x*,*b0*,*c0*)$\#_\Gamma$Γ ⊨ *c0′*
  shows ∃ *t2*. Θ; *B*; Γ′@((*x*,*b0*,*c0*)$\#_\Gamma$Γ) ⊢ *v* ⇒ *t2* ∧ Θ; *B*; Γ′@((*x*,*b0*,*c0*)$\#_\Gamma$Γ) ⊢ *t2* ≲ *t1*
proof −

  have Θ; *B*; Γ′@((*x*,*b0*,*c0*)$\#_\Gamma$Γ) ⊢ *v* ⇒ *t1* **using** *ctx-subtype-v-aux assms* **by** *auto*
  moreover hence Θ; *B*; Γ′@((*x*,*b0*,*c0*)$\#_\Gamma$Γ) ⊢ *t1* ≲ *t1* **using** *subtype-reflI2 infer-v-wf* **by** *simp*
  ultimately show *?thesis* **by** *auto*
qed

lemma *ctx-subtype-v-eq*:
  fixes *v*::*v*
  assumes
    Θ; *B*; Γ′@((*x*,*b0*,*c0′*)$\#_\Gamma$Γ) ⊢ *v* ⇒ *t1* **and**

Θ; $\mathcal{B}$; Γ'@$(x,b0,c0)\#_\Gamma\Gamma \models c0'$
shows Θ; $\mathcal{B}$; Γ'@$((x,b0,c0)\#_\Gamma\Gamma) \vdash v \Rightarrow t1$
proof −
obtain *t1'* **where** Θ; $\mathcal{B}$; Γ'@$((x,b0,c0)\#_\Gamma\Gamma) \vdash v \Rightarrow t1'$ **using** *ctx-subtype-v assms* **by** *metis*
moreover have *replace-in-g* $(\Gamma'$@$((x,b0,c0')\#_\Gamma\Gamma))$ *x c0* = Γ'@$((x,b0,c0)\#_\Gamma\Gamma)$ **using** *replace-in-g-inside infer-v-wf assms* **by** *metis*
ultimately show *?thesis* **using** *infer-v-uniqueness-rig assms* **by** *metis*
qed

lemma *ctx-subtype-check-v-eq*:
assumes Θ; $\mathcal{B}$; Γ'@$((x,b0,c0')\#_\Gamma\Gamma) \vdash v \Leftarrow t1$ **and** Θ; $\mathcal{B}$; Γ'@$(x,b0,c0)\#_\Gamma\Gamma \models c0'$
shows Θ; $\mathcal{B}$; Γ'@$((x,b0,c0)\#_\Gamma\Gamma) \vdash v \Leftarrow t1$
proof −
obtain *t2* **where** *t2*: Θ; $\mathcal{B}$; Γ'@$((x,b0,c0')\#_\Gamma\Gamma) \vdash v \Rightarrow t2 \wedge$ Θ; $\mathcal{B}$; Γ'@$((x,b0,c0')\#_\Gamma\Gamma) \vdash t2 \lesssim t1$
using *check-v-elims assms* **by** *blast*
hence *t3*: Θ; $\mathcal{B}$; Γ'@$((x,b0,c0)\#_\Gamma\Gamma) \vdash v \Rightarrow t2$
using *assms ctx-subtype-v-eq* **by** *blast*

have Θ; $\mathcal{B}$; Γ'@$((x,b0,c0)\#_\Gamma\Gamma) \vdash v \Rightarrow t2$ **using** *t3* **by** *auto*
moreover have Θ; $\mathcal{B}$; Γ'@$((x,b0,c0)\#_\Gamma\Gamma) \vdash t2 \lesssim t1$ **proof** −

have Θ; $\mathcal{B}$; Γ'@$((x,b0,c0')\#_\Gamma\Gamma) \vdash t2 \lesssim t1$ **using** *t2* **by** *auto*
thus *?thesis* **using** *subtype-trans*
using *assms(2) ctx-subtype-subtype* **by** *blast*
qed
ultimately show *?thesis* **using** *check-v.intros* **by** *presburger*
qed

Basically the same as *ctx-subtype-v-eq* but in a different form

lemma *ctx-subtype-v-rig-eq*:
fixes *v*::*v*
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' $[(x,c0)]$ Γ **and**
Θ; $\mathcal{B}$; $\Gamma' \vdash v \Rightarrow t1$
shows Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow t1$
proof −
obtain *b* **and** *c0'* **and** *G* **and** *G'* **where** $\Gamma' = G'$ @ $(x,b,c0')\#_\Gamma G \wedge \Gamma = G'$ @ $(x,b,c0)\#_\Gamma G \wedge$ Θ; $\mathcal{B}$; G'@ $(x,b,c0)\#_\Gamma G \models c0'$
using *assms replace-in-g-inside-valid infer-v-wf* **by** *metis*
thus *?thesis* **using** *ctx-subtype-v-eq*[*of* Θ $\mathcal{B}$ *G' x b c0' G v t1 c0*] *assms* **by** *simp*
qed

lemma *ctx-subtype-v-rigs-eq*:
fixes *v*::*v*
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' *xcs* Γ **and**
Θ; $\mathcal{B}$; $\Gamma' \vdash v \Rightarrow t1$
shows Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow t1$
using *assms* **proof**(*induct xcs arbitrary*: Γ Γ' *t1*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a xcs*)
then obtain *x* **and** *c* **where** *a*=(*x*,*c*) **by** *fastforce*

then obtain *b* **and** *c′* **where** *bc*: *Some* (*b*, *c′*) = *lookup* Γ′ *x* ∧
replace-in-g-subtyped Θ ℬ (*replace-in-g* Γ′ *x c*) *xcs* Γ ∧ Θ; ℬ; Γ′ $\vdash_{wf}$ *c* ∧
x ∉ *fst* ‘ *set xcs* ∧ Θ; ℬ; (*replace-in-g* Γ′ *x c*) ⊨ *c′*
using *replace-in-g-subtyped-elims*(*3*)[*of* Θ ℬ Γ′ *x c xcs* Γ] *Cons* **by** (*metis valid.simps*)

hence ∗: *replace-in-g-subtyped* Θ ℬ Γ′ [(*x*,*c*)] (*replace-in-g* Γ′ *x c*) **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct*(*1*) *list.set-cases replace-in-g-subtyped-nilI*)
hence *t2*: Θ; ℬ; (*replace-in-g* Γ′ *x c*) ⊢ *v* ⇒ *t1* **using** *ctx-subtype-v-rig-eq*[*OF* ∗ *Cons*(*3*)] **by** *blast*
moreover have ∗∗: *replace-in-g-subtyped* Θ ℬ (*replace-in-g* Γ′ *x c*) *xcs* Γ **using** *bc* **by** *auto*
ultimately have *t2′*: Θ; ℬ; Γ ⊢ *v* ⇒ *t1* **using** *Cons* **by** *blast*
thus *?case* **by** *blast*
qed

lemma *ctx-subtype-check-v-rigs-eq*:
assumes *replace-in-g-subtyped* Θ ℬ Γ′ *xcs* Γ **and**
Θ; ℬ; Γ′ ⊢ *v* ⇐ *t1*
shows Θ; ℬ; Γ ⊢ *v* ⇐ *t1*
proof −
obtain *t2* **where** Θ; ℬ; Γ′ ⊢ *v* ⇒ *t2* ∧ Θ; ℬ; Γ′ ⊢ *t2* ≲ *t1* **using** *check-v-elims assms* **by** *fast*
hence Θ; ℬ; Γ ⊢ *v* ⇒ *t2* ∧ Θ; ℬ; Γ ⊢ *t2* ≲ *t1* **using** *ctx-subtype-v-rigs-eq ctx-subtype-subtype-rigs*
using *assms*(*1*) **by** *blast*
thus *?thesis*
using *check-v-subtypeI* **by** *blast*
qed

13.5 Expressions

lemma *valid-wfC*:
fixes *c0*::*c*
assumes Θ; ℬ; Γ′@(*x*,*b0*,*c0*)$\#_\Gamma$Γ ⊨ *c0′*
shows Θ; ℬ; (*x*, *b0*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c0*
using *wfG-elim2 valid.simps wfG-suffix*
using *assms valid-g-wf* **by** *metis*

lemma *ctx-subtype-e-eq*:
fixes *G*::Γ
assumes
Θ ; Φ ; ℬ ; *G* ; Δ ⊢ *e* ⇒ *t1* **and** *G* = Γ′@((*x*,*b0*,*c0′*)$\#_\Gamma$Γ)
Θ; ℬ; Γ′@(*x*,*b0*,*c0*)$\#_\Gamma$Γ ⊨ *c0′*
shows Θ ; Φ ; ℬ ; Γ′@((*x*,*b0*,*c0*)$\#_\Gamma$Γ) ; Δ ⊢ *e* ⇒ *t1*
using *assms* **proof**(*nominal-induct t1 avoiding*: *c0 rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ ℬ Γ″ Δ Φ *v* τ)
show *?case* **proof**
show ‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-valI*
by *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-valI* **by** *auto*
show ‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇒ τ› **using** *infer-e-valI ctx-subtype-v-eq* **by** *auto*
qed
next
case (*infer-e-plusI* Θ ℬ Γ″ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **proof**

show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-plusI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-plusI* **by** *auto*
show ∗:‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v1* ⇒ ⦃ *z1* : *B-int* | *c1* ⦄› **using** *infer-e-plusI ctx-subtype-v-eq* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v2* ⇒ ⦃ *z2* : *B-int* | *c2* ⦄› **using** *infer-e-plusI ctx-subtype-v-eq* **by** *auto*
show ‹*atom z3* ♯ *AE-op Plus v1 v2*› **using** *infer-e-plusI* **by** *auto*
show ‹*atom z3* ♯ Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ› **using** ∗ *infer-e-plusI fresh-replace-inside infer-v-wf* **by** *metis*
qed
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-leqI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-leqI* **by** *auto*
show ∗:‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v1* ⇒ ⦃ *z1* : *B-int* | *c1* ⦄› **using** *infer-e-leqI ctx-subtype-v-eq* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v2* ⇒ ⦃ *z2* : *B-int* | *c2* ⦄› **using** *infer-e-leqI ctx-subtype-v-eq* **by** *auto*
show ‹*atom z3* ♯ *AE-op LEq v1 v2*› **using** *infer-e-leqI* **by** *auto*
show ‹*atom z3* ♯ Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ› **using** ∗ *infer-e-leqI fresh-replace-inside infer-v-wf* **by** *metis*
qed
next
case (*infer-e-eqI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 bb c1 v2 z2 c2 z3*)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-eqI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-eqI* **by** *auto*
show ∗:‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v1* ⇒ ⦃ *z1* : *bb* | *c1* ⦄› **using** *infer-e-eqI ctx-subtype-v-eq* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v2* ⇒ ⦃ *z2* : *bb* | *c2* ⦄› **using** *infer-e-eqI ctx-subtype-v-eq* **by** *auto*
show ‹*atom z3* ♯ *AE-op Eq v1 v2*› **using** *infer-e-eqI* **by** *auto*
show ‹*atom z3* ♯ Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ› **using** ∗ *infer-e-eqI fresh-replace-inside infer-v-wf* **by** *metis*
show *bb* ∈ {*B-bool*, *B-int*, *B-unit*} **using** *infer-e-eqI* **by** *auto*
qed
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ'' Δ Φ *f* x' *b c* τ' s' *v* τ)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-appI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x' *b c* τ' s'))) = *lookup-fun* Φ *f*› **using** *infer-e-appI* **by** *auto*
show ‹Θ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ ⊢ *v* ⇐ ⦃ x' : *b* | *c* ⦄› **using** *infer-e-appI ctx-subtype-check-v-eq* **by** *auto*
thus ‹*atom* x' ♯ (Θ, Φ, $\mathcal{B}$, Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ, Δ, *v*, τ)›
using *infer-e-appI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0*$'$ Γ *c0 x*$'$] *infer-v-wf* **by** *auto*

show ‹$\tau'[x'::=v]_v = \tau$› **using** *infer-e-appI* **by** *auto*
qed
next
case (*infer-e-appPI* Θ $\mathcal{B}$ Γ*1* Δ Φ *b′ f bv x1 b c* τ′ *s′ v* τ)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-appPI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appPI* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ *b′* › **using** *infer-e-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x1 b c* τ′ *s′*))) = *lookup-fun* Φ *f*› **using** *infer-e-appPI* **by** *auto*
show ‹Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇐ ⦃ *x1* : *b*[*bv*::=*b′*]$_b$ | *c*[*bv*::=*b′*]$_b$ ⦄› **using** *infer-e-appPI ctx-subtype-check-v-eq subst-defs* **by** *auto*
thus ‹*atom x1* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ′ *x b0 c0′* Γ *c0 x1*] *infer-v-wf infer-e-appPI* **by** *auto*
show ‹$\tau'[bv::=b']_b[x1::=v]_v = \tau$› **using** *infer-e-appPI* **by** *auto*
have *atom bv* ♯ Γ′ @ (*x*, *b0*, *c0′*) $\#_\Gamma$ Γ **using** *infer-e-appPI* **by** *metis*
hence *atom bv* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ
unfolding *fresh-append-g fresh-GCons fresh-prod3* **using** ‹*atom bv* ♯ *c0* › *fresh-append-g* **by** *metis*
thus ‹*atom bv* ♯ (Θ, Φ, $\mathcal{B}$, Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ, Δ, *b′*, *v*, τ)› **using** *infer-e-appPI* **by** *auto*
qed
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ′′ Δ Φ *v z′ b1 b2 c z*)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-fstI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-fstI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇒ ⦃ *z′* : *B-pair b1 b2* | *c* ⦄› **using** *infer-e-fstI ctx-subtype-v-eq* **by** *auto*
thus ‹*atom z* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-fstI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ′ *x b0 c0′* Γ *c0 z*] *infer-v-wf* **by** *auto*
show ‹*atom z* ♯ *AE-fst v*› **using** *infer-e-fstI* **by** *auto*
qed
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ′′ Δ Φ *v z′ b1 b2 c z*)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-sndI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-sndI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇒ ⦃ *z′* : *B-pair b1 b2* | *c* ⦄› **using** *infer-e-sndI ctx-subtype-v-eq* **by** *auto*
thus ‹*atom z* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-sndI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ′ *x b0 c0′* Γ *c0 z*] *infer-v-wf* **by** *auto*
show ‹*atom z* ♯ *AE-snd v*› **using** *infer-e-sndI* **by** *auto*
qed
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ′′ Δ Φ *v z′ c z*)
show *?case* **proof**
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-lenI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-lenI* **by** *auto*
show ‹ Θ; $\mathcal{B}$; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v* ⇒ ⦃ *z′* : *B-bitvec* | *c* ⦄› **using** *infer-e-lenI ctx-subtype-v-eq*

by *auto*
  thus ‹*atom z* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-lenI fresh-replace-inside*[*of* Θ ℬ Γ′ *x b0 c0*′ Γ *c0 z*] *infer-v-wf* **by** *auto*
  show ‹*atom z* ♯ *AE-len v*› **using** *infer-e-lenI* **by** *auto*
 qed
next
 case (*infer-e-mvarI* Θ ℬ Γ′′ Φ Δ *u* τ)
 show *?case* **proof**
  show Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-mvarI* **by** *auto*
  thus Θ; ℬ $\vdash_{wf}$ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ **using** *infer-e-mvarI fresh-replace-inside wfD-wf* **by** *blast*
  show Θ $\vdash_{wf}$ Φ **using** *infer-e-mvarI* **by** *auto*
  show (*u*, τ) ∈ *setD* Δ **using** *infer-e-mvarI* **by** *auto*
 qed
next
 case (*infer-e-concatI* Θ ℬ Γ′′ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
 show *?case* **proof**
  show ‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-concatI* **by** *auto*
  thus ‹*atom z3* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-concatI fresh-replace-inside*[*of* Θ ℬ Γ′ *x b0 c0*′ Γ *c0 z3*] *infer-v-wf wfX-wfY* **by** *metis*
  show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-concatI* **by** *auto*
  show ‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v1* ⇒ ⦃ *z1* : *B-bitvec* | *c1* ⦄› **using** *infer-e-concatI ctx-subtype-v-eq* **by** *auto*
  show ‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v2* ⇒ ⦃ *z2* : *B-bitvec* | *c2* ⦄› **using** *infer-e-concatI ctx-subtype-v-eq* **by** *auto*
  show ‹*atom z3* ♯ *AE-concat v1 v2*› **using** *infer-e-concatI* **by** *auto*
 qed
next
 case (*infer-e-splitI* Θ ℬ Γ′′ Δ Φ *v1 z1 c1 v2 z2 z3*)
 show *?case* **proof**
  show ∗:‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-splitI* **by** *auto*
  show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-splitI* **by** *auto*
  show ‹ Θ; ℬ; Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v1* ⇒ ⦃ *z1* : *B-bitvec* | *c1* ⦄› **using** *infer-e-splitI ctx-subtype-v-eq* **by** *auto*
  show ‹Θ; ℬ; Γ′ @
    (*x*, *b0*, *c0*) $\#_\Gamma$
   Γ ⊢ *v2* ⇐ ⦃ *z2* : *B-int* | [*leq* [[*L-num 0*]^v]^{ce} [[*z2*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce} *AND*
      [*leq* [[*z2*]^v]^{ce} [| [*v1*]^{ce} |]^{ce}]^{ce} == [[*L-true*]^v]^{ce} ⦄›
   using *infer-e-splitI ctx-subtype-check-v-eq* **by** *auto*

  show ‹*atom z1* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ ℬ Γ′ *x b0 c0*′ Γ *c0 z1*] *infer-e-splitI infer-v-wf wfX-wfY* ∗ **by** *metis*
  show ‹*atom z2* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ ℬ Γ′ *x b0 c0*′ Γ *c0*] *infer-e-splitI infer-v-wf wfX-wfY* ∗ **by** *metis*
  show ‹*atom z3* ♯ Γ′ @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ ℬ Γ′ *x b0 c0*′ Γ *c0*] *infer-e-splitI infer-v-wf wfX-wfY* ∗ **by** *metis*
  show ‹*atom z1* ♯ *AE-split v1 v2*› **using** *infer-e-splitI* **by** *auto*
  show ‹*atom z2* ♯ *AE-split v1 v2*› **using** *infer-e-splitI* **by** *auto*
  show ‹*atom z3* ♯ *AE-split v1 v2*› **using** *infer-e-splitI* **by** *auto*
 qed

qed

lemma *ctx-subtype-e-rig-eq*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' $[(x,c0)]$ Γ **and**
Θ ; Φ ; $\mathcal{B}$; Γ' ; $\Delta \vdash e \Rightarrow t1$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow t1$
proof –
obtain b **and** $c0'$ **and** G **and** G' **where** $\Gamma' = G' @ (x,b,c0')\#_\Gamma G \wedge \Gamma = G' @ (x,b,c0)\#_\Gamma G \wedge \Theta; \mathcal{B}; G'@ (x,b,c0)\#_\Gamma G \models c0'$
using *assms replace-in-g-inside-valid infer-e-wf* **by** *meson*
thus *?thesis*
using *assms ctx-subtype-e-eq* **by** *presburger*
qed

lemma *ctx-subtype-e-rigs-eq*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' *xcs* Γ **and**
Θ ; Φ ; $\mathcal{B}$; Γ'; $\Delta \vdash e \Rightarrow t1$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow t1$
using *assms* **proof**(*induct xcs arbitrary*: Γ Γ' *t1*)
case *Nil*
moreover have $\Gamma' = \Gamma$ **using** *replace-in-g-subtyped-nilI*
using *calculation*(*1*) **by** *blast*
moreover have $\Theta;\mathcal{B};\Gamma \vdash t1 \lesssim t1$ **using** *subtype-reflI2 Nil infer-e-t-wf* **by** *blast*
ultimately show *?case* **by** *blast*
next
case (*Cons a xcs*)

then obtain x **and** c **where** $a{=}(x,c)$ **by** *fastforce*
then obtain b **and** c' **where** *bc*: *Some* $(b,\ c')$ = *lookup* Γ' x $\wedge$
replace-in-g-subtyped Θ $\mathcal{B}$ (*replace-in-g* Γ' x c) *xcs* Γ $\wedge$ Θ; $\mathcal{B}$; Γ' $\vdash_{wf}$ c $\wedge$
$x \notin$ *fst* ‘ *set xcs* $\wedge$ Θ; $\mathcal{B}$; (*replace-in-g* Γ' x c) $\models c'$ **using** *replace-in-g-subtyped-elims*(*3*)[*of* Θ $\mathcal{B}$ Γ' x c *xcs* Γ] *Cons*
by (*metis valid.simps*)

hence $*$: *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' $[(x,c)]$ (*replace-in-g* Γ' x c) **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct*(*1*) *list.set-cases replace-in-g-subtyped-nilI*)
hence *t2*: Θ ; Φ ; $\mathcal{B}$; (*replace-in-g* Γ' x c) ; $\Delta \vdash e \Rightarrow t1$ **using** *ctx-subtype-e-rig-eq*[*OF* $*$ *Cons*(*3*)] **by** *blast*
moreover have $**$: *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* Γ' x c) *xcs* Γ **using** *bc* **by** *auto*
ultimately have *t2'*: Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow t1$ **using** *Cons* **by** *blast*
thus *?case* **by** *blast*
qed

13.6 Statements

lemma *ctx-subtype-s-rigs*:
fixes *c0*::*c* **and** *s*::*s* **and** G'::Γ **and** *xcs* :: $(x*c)$ *list* **and** *css*::*branch-list*
shows
check-s Θ Φ $\mathcal{B}$ G Δ *s* *t1* $\Longrightarrow$ *wsX G xcs* $\Longrightarrow$ *replace-in-g-subtyped* Θ $\mathcal{B}$ G *xcs* G' $\Longrightarrow$ *check-s* Θ Φ $\mathcal{B}$ G' Δ *s* *t1* **and**
check-branch-s Θ Φ $\mathcal{B}$ G Δ *tid cons const v cs* *t1* $\Longrightarrow$ *wsX G xcs* $\Longrightarrow$ *replace-in-g-subtyped* Θ $\mathcal{B}$ G *xcs* G' $\Longrightarrow$ *check-branch-s* Θ Φ $\mathcal{B}$ G' Δ *tid cons const v cs t1*

check-branch-list Θ Φ $\mathcal{B}$ *G* Δ *tid dclist v css t1* $\Longrightarrow$ *wsX G xcs* $\Longrightarrow$ *replace-in-g-subtyped* Θ $\mathcal{B}$ *G xcs G'* $\Longrightarrow$ *check-branch-list* Θ Φ $\mathcal{B}$ *G'* Δ *tid dclist v css t1*
proof(*induction arbitrary*: *xcs G'* **and** *xcs G'* **and** *xcs G'* *rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ *v* τ' τ)
hence $*$:Θ; $\mathcal{B}$; G' $\vdash$ $v \Rightarrow \tau' \wedge$ Θ; $\mathcal{B}$; G' $\vdash \tau' \lesssim \tau$ **using** *ctx-subtype-v-rigs-eq ctx-subtype-subtype-rigs*

by (*meson check-v.simps*)
show *?case* **proof**
show ‹Θ; $\mathcal{B}$; $G' \vdash_{wf}$ Δ› **using** *check-valI wfD-rig* **by** *auto*
show ‹$\Theta \vdash_{wf}$ Φ › **using** *check-valI* **by** *auto*
show ‹Θ; $\mathcal{B}$; G' $\vdash$ $v \Rightarrow \tau'$› **using** $*$ **by** *auto*
show ‹Θ; $\mathcal{B}$; G' $\vdash \tau' \lesssim \tau$› **using** $*$ **by** *auto*
qed
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z' s b' c'*)

show *?case* **proof**
have *wfG*: Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ Θ; $\mathcal{B}$ $\vdash_{wf}$ G' **using** *infer-e-wf check-letI replace-in-g-wfG* **using** *infer-e-wf*(*2*) **by** (*auto simp add*: *freshers*)
hence *atom x* $\sharp$ *G'* **using** *check-letI replace-in-g-fresh replace-in-g-wfG* **by** *auto*
thus *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, *G'*, Δ, *e*, τ) **using** *check-letI* **by** *auto*
have *atom z'* $\sharp$ *G'* **apply**(*rule replace-in-g-fresh*[*OF check-letI*(*7*)])
using *replace-in-g-wfG check-letI fresh-prodN infer-e-wf* **by** *metis*+
thus *atom z'* $\sharp$ (*x*, Θ, Φ, $\mathcal{B}$, *G'*, Δ, *e*, τ, *s*) **using** *check-letI fresh-prodN* **by** *metis*

show Θ ; Φ ; $\mathcal{B}$; G' ; Δ $\vdash$ $e \Rightarrow \{\!\!\{ z' : b' \mid c' \}\!\!\}$
using *check-letI ctx-subtype-e-rigs-eq* **by** *blast*
show Θ ; Φ ; $\mathcal{B}$; (*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ G' ; Δ $\vdash$ $s \Leftarrow \tau$
proof(*rule check-letI*(*5*))
have *vld*: Θ;$\mathcal{B}$;((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) $\models$ $c'[z'::=V\text{-}var\ x]_{cv}$ **proof** $-$
have *wfG* Θ $\mathcal{B}$ ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) **using** *check-letI check-s-wf* **by** *metis*
hence *wfC* Θ $\mathcal{B}$ ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) ($c'[z'::=V\text{-}var\ x]_{cv}$) **using** *wfC-refl subst-defs* **by** *auto*
thus *?thesis* **using** *valid-reflI*[*of* Θ $\mathcal{B}$ *x b'* $c'[z'::=V\text{-}var\ x]_v$ Γ $c'[z'::=V\text{-}var\ x]_v$] *subst-defs* **by** *auto*
qed
have *xf*: $x \notin$ *fst* ‘ *set xcs* **proof** $-$
have *atom* ‘ *fst* ‘ *set xcs* $\subseteq$ *atom-dom* Γ **using** *check-letI wsX-iff* **by** *meson*
moreover have *wfG* Θ $\mathcal{B}$ Γ **using** *infer-e-wf check-letI* **by** *metis*
ultimately show *?thesis* **using** *fresh-def check-letI wfG-dom-supp*
using *wsX-fresh* **by** *auto*
qed
show *replace-in-g-subtyped* Θ $\mathcal{B}$ ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) ((*x*, $c'[z'::=V\text{-}var\ x]_v$) # *xcs*) ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ *G'*) **proof** $-$

have *Some* (*b'*, $c'[z'::=V\text{-}var\ x]_v$) = *lookup* ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) *x* **by** *auto*

moreover have Θ; $\mathcal{B}$; *replace-in-g* ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) *x* ($c'[z'::=V\text{-}var\ x]_v$) $\models$ $c'[z'::=V\text{-}var\ x]_v$ **proof** $-$
have *replace-in-g* ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) *x* ($c'[z'::=V\text{-}var\ x]_v$) = ((*x*, *b'*, $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ)
using *replace-in-g.simps* **by** *presburger*

thus *?thesis* **using** *vld subst-defs* **by** *auto*
qed

moreover have *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* ((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ) *x* (*c*′[*z*′::=*V-var x*]$_v$)) *xcs* (((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ *G*′)) **proof** −
have *wfG* Θ $\mathcal{B}$ (((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ)) **using** *check-letI check-s-wf* **by** *metis*
hence *replace-in-g-subtyped* Θ $\mathcal{B}$ (((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ)) *xcs* (((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ *G*′))
using *check-letI replace-in-g-subtyped-cons xf* **by** *meson*
moreover have *replace-in-g* ((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ) *x* (*c*′[*z*′::=*V-var x*]$_v$) = (((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ))
using *replace-in-g.simps* **by** *presburger*
ultimately show *?thesis* **by** *argo*
qed
moreover have Θ; $\mathcal{B}$; (*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ $\vdash_{wf}$ *c*′[*z*′::=*V-var x*]$_v$ **using** *vld subst-defs* **by** *auto*
ultimately show *?thesis* **using** *replace-in-g-subtyped-consI xf replace-in-g.simps*(*2*) **by** *metis*
qed

show *wsX* ((*x*, *b*′, *c*′[*z*′::=*V-var x*]$_v$) $\#_\Gamma$ Γ) ((*x*, *c*′[*z*′::=*V-var x*]$_v$) # *xcs*)
using *check-letI xf subst-defs* **by** (*simp add*: *wsX-cons*)
qed
qed

next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **using** *Typing.check-branch-list-consI* **by** *auto*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
then show *?case* **using** *Typing.check-branch-list-finalI* **by** *auto*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)

have *wfcons*: *wfG* Θ $\mathcal{B}$ ((*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) **using** *check-s-wf check-branch-s-branchI*
by *meson*
hence *wf*: *wfG* Θ $\mathcal{B}$ Γ **using** *wfG-cons* **by** *metis*

moreover have *atom x* ♯ (*const*, *G*′,*v*) **proof** −
have *atom x* ♯ *G*′ **using** *check-branch-s-branchI wf replace-in-g-fresh*
wfG-dom-supp replace-in-g-wfG **by** *simp*
thus *?thesis* **using** *check-branch-s-branchI fresh-prodN* **by** *simp*
qed

moreover have *st*: Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ *G*′ ; Δ ⊢ *s* ⇐ τ **proof** −
have *wsX* ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) *xcs* **using** *check-branch-s-branchI wsX-cons2 wsX-fresh wf* **by** *force*
moreover have *replace-in-g-subtyped* Θ $\mathcal{B}$ ((*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) *xcs* ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ *G*′)
using *replace-in-g-subtyped-cons wsX-fresh wf check-branch-s-branchI wfcons* **by** *auto*

thus *?thesis* **using** *check-branch-s-branchI calculation* **by** *meson*
qed
moreover have *wft*: *wfT* Θ *ℬ* *G′* τ **using**
check-branch-s-branchI ctx-subtype-subtype-rigs subtype-reflI2 subtype-wf **by** *metis*
moreover have *wfD* Θ *ℬ* *G′* Δ **using** *check-branch-s-branchI wfD-rig* **by** *presburger*
ultimately show *?case* **using**
Typing.check-branch-s-branchI
using *check-branch-s-branchI.hyps* **by** *simp*

next
case (*check-ifI z* Θ Φ *ℬ* Γ Δ *v s1 s2* τ)
hence *wf*:*wfG* Θ *ℬ* Γ **using** *check-s-wf* **by** *presburger*
show *?case* **proof**(*rule check-s-check-branch-s-check-branch-list.check-ifI*)
show ‹*atom z* ♯ (Θ, Φ, *ℬ*, *G′*, Δ, *v*, *s1*, *s2*, τ)› **using** *fresh-prodN replace-in-g-fresh1 wf check-ifI* **by** *auto*
show ‹Θ; *ℬ*; *G′* ⊢ *v* ⇐ ⦃ *z* : *B-bool* | *TRUE* ⦄› **using** *ctx-subtype-check-v-rigs-eq check-ifI* **by** *presburger*
show ‹ Θ ; Φ ; *ℬ* ; *G′* ; Δ ⊢ *s1* ⇐ ⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-true*) *IMP c-of* τ *z* ⦄› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; *ℬ* ; *G′* ; Δ ⊢ *s2* ⇐ ⦃ *z* : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP c-of* τ *z* ⦄› **using** *check-ifI* **by** *auto*
qed
next

case (*check-let2I x P* Φ *ℬ* *G* Δ *t s1* τ *s2*)
show *?case* **proof**
have *wfG P ℬ G* **using** *check-let2I check-s-wf* **by** *metis*
show ∗: *P* ; Φ ; *ℬ* ; *G′* ; Δ ⊢ *s1* ⇐*t* **using** *check-let2I* **by** *blast*
show *atom x* ♯ (*P*, Φ, *ℬ*, *G′*, Δ, *t*, *s1*, τ) **proof** −
have *wfG P ℬ G′* **using** *check-s-wf* ∗ **by** *blast*
hence *atom-dom G* = *atom-dom G′* **using** *check-let2I rigs-atom-dom-eq* **by** *presburger*
moreover have *atom x* ♯ *G* **using** *check-let2I* **by** *auto*
moreover have *wfG P ℬ G* **using** *check-s-wf* ∗ *replace-in-g-wfG check-let2I* **by** *simp*
ultimately have *atom x* ♯ *G′* **using** *wfG-dom-supp fresh-def* ‹*wfG P ℬ G′*› **by** *metis*
thus *?thesis* **using** *check-let2I* **by** *auto*
qed
show *P* ; Φ ; *ℬ* ;(*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G′* ; Δ ⊢ *s2* ⇐ τ **proof** −
have *wsX* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G*) *xcs* **using** *check-let2I wsX-cons2 wsX-fresh* ‹*wfG P ℬ G*› **by** *simp*
moreover have *replace-in-g-subtyped P ℬ* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G*) *xcs* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G′*) **proof**(*rule replace-in-g-subtyped-cons*)
show *replace-in-g-subtyped P ℬ G xcs G′* **using** *check-let2I* **by** *auto*
have *atom x* ♯ *G* **using** *check-let2I* **by** *auto*
moreover have *wfT P ℬ G t* **using** *check-let2I check-s-wf* **by** *metis*

moreover have *atom x* ♯ *t* **using** *check-let2I check-s-wf wfT-supp* **by** *auto*
ultimately show *wfG P ℬ* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G*) **using** *wfT-wf-cons b-of-c-of-eq*[*of x t*] **by** *auto*
show *x* ∉ *fst* ‘ *set xcs* **using** *check-let2I wsX-fresh* ‹*wfG P ℬ G*› **by** *simp*
qed
ultimately show *?thesis* **using** *check-let2I* **by** *presburger*
qed

qed
next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ′ *v* τ *s*)
show *?case* **proof**
have *atom u* ♯ *G*′ **unfolding** *fresh-def*
apply(*rule u-not-in-g* , *rule replace-in-g-wfG*)
using *check-v-wf check-varI* **by** *simp*+
thus ‹*atom u* ♯ (Θ, Φ, $\mathcal{B}$, *G*′, Δ, τ′, *v*, τ)› **unfolding** *fresh-prodN* **using** *check-varI* **by** *simp*
show ‹Θ; $\mathcal{B}$; *G*′ ⊢ *v* ⇐ τ′› **using** *ctx-subtype-check-v-rigs-eq check-varI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; *G*′ ; (*u*, τ′) $\#_\Delta$ Δ ⊢ *s* ⇐ τ› **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI P* Φ $\mathcal{B}$ *G* Δ *u* τ *v z* τ′)
show *?case* **proof**
show ‹*P* $\vdash_{wf}$ Φ › **using** *check-assignI* **by** *auto*
show ‹*P* ; $\mathcal{B}$; *G*′ $\vdash_{wf}$ Δ › **using** *check-assignI wfD-rig* **by** *auto*
show ‹(*u*, τ) ∈ *setD* Δ› **using** *check-assignI* **by** *auto*
show ‹*P* ; $\mathcal{B}$; *G*′ ⊢ *v* ⇐ τ› **using** *ctx-subtype-check-v-rigs-eq check-assignI* **by** *auto*
show ‹*P* ; $\mathcal{B}$; *G*′ ⊢ ⦃ *z* : *B-unit* | *TRUE* ⦄ ≲ τ′› **using** *ctx-subtype-subtype-rigs check-assignI* **by** *auto*
qed
next
case (*check-whileI* Δ *G P s1 z s2* τ′)
then show *?case* **using** *Typing.check-whileI*
by (*meson ctx-subtype-subtype-rigs*)
next
case (*check-seqI* Δ *G P s1 z s2* τ)
then show *?case*
using *check-s-check-branch-s-check-branch-list.check-seqI* **by** *blast*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *z*)
show *?case* **proof**
show Θ ; Φ ; $\mathcal{B}$; *G*′ ; Δ ; *tid* ; *dclist* ; *v* ⊢ *cs* ⇐ τ **using** *check-caseI ctx-subtype-check-v-rigs-eq* **by** *auto*
show *AF-typedef tid dclist* ∈ *set* Θ **using** *check-caseI* **by** *auto*
show Θ; $\mathcal{B}$; *G*′ ⊢ *v* ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄ **using** *check-caseI ctx-subtype-check-v-rigs-eq* **by** *auto*
show $\vdash_{wf}$ Θ **using** *check-caseI* **by** *auto*
qed
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
show *?case* **proof**
have *wfG*: Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ∧ Θ; $\mathcal{B}$ $\vdash_{wf}$ *G*′ **using** *check-s-wf check-assertI replace-in-g-wfG wfX-wfY* **by** *metis*
hence *atom x* ♯ *G*′ **using** *check-assertI replace-in-g-fresh replace-in-g-wfG* **by** *auto*
thus ‹*atom x* ♯ (Θ, Φ, $\mathcal{B}$, *G*′, Δ, *c*, τ, *s*)› **using** *check-assertI fresh-prodN* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ *G*′ ; Δ ⊢ *s* ⇐ τ› **proof**(*rule check-assertI*(*5*))
show *wsX* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) *xcs* **using** *check-assertI wsX-cons3* **by** *simp*
show Θ; $\mathcal{B}$ ⊢ (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ⟨ *xcs* ⟩ ⇝ (*x*, *B-bool*, *c*) $\#_\Gamma$ *G*′ **proof**(*rule replace-in-g-subtyped-cons*)
show ‹ Θ; $\mathcal{B}$ ⊢ Γ ⟨ *xcs* ⟩ ⇝ *G*′› **using** *check-assertI* **by** *auto*
show ‹ Θ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ › **using** *check-assertI check-s-wf* **by** *metis*
thus ‹*x* ∉ *fst* ‘ *set xcs*› **using** *check-assertI wsX-fresh wfG-elims wfX-wfY* **by** *metis*

qed
qed
show ‹Θ; $\mathcal{B}$; G' $\models$ c › **using** *check-assertI replace-in-g-valid* **by** *auto*
show ‹ Θ; $\mathcal{B}$; $G' \vdash_{wf}$ Δ › **using** *check-assertI wfD-rig* **by** *auto*
qed
qed

lemma *replace-in-g-subtyped-empty*:
assumes *wfG* Θ $\mathcal{B}$ (Γ′ @ (x, b, $c[z{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)
shows *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* (Γ′ @ (x, b, $c[z{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) x ($c'[z'{::=}V\text{-}var\ x]_{cv}$)) [] (Γ′ @ ($x$, b, $c'[z'{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)
proof −
have *replace-in-g* (Γ′ @ (x, b, $c[z{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) x ($c'[z'{::=}V\text{-}var\ x]_{cv}$) = (Γ′ @ ($x$, b, $c'[z'{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *replace-in-g.simps* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
have $x \notin$ *fst* ‘ *toSet* ((*x1*,*b1*,*c1*)$\#_\Gamma$Γ*1*) **using** *GCons wfG-inside-fresh atom-dom.simps dom.simps toSet.simps append-g.simps* **by** *fast*
hence *x1* $\neq$ x **using** *assms wfG-inside-fresh GCons* **by** *force*
hence ((*x1*,*b1*,*c1*) $\#_\Gamma$ (Γ*1* @ (x, b, $c[z{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ))[$x \longmapsto c'[z'{::=}V\text{-}var\ x]_{cv}$] = (*x1*,*b1*,*c1*) $\#_\Gamma$ (Γ*1* @ (x, b, $c'[z'{::=}V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)
using *replace-in-g.simps GCons wfG-elims append-g.simps* **by** *metis*
thus *?case* **using** *append-g.simps* **by** *simp*
qed
thus *?thesis* **using** *replace-in-g-subtyped-nilI* **by** *presburger*
qed

lemma *ctx-subtype-s*:
fixes *s*::*s*
assumes Θ ; Φ ; $\mathcal{B}$; Γ′@((x,b,$c[z{::=}V\text{-}var\ x]_{cv}$)$\#_\Gamma$Γ) ; Δ ⊢ $s \Leftarrow \tau$ **and**
Θ; $\mathcal{B}$; Γ ⊢ ⦃ z' : b | c' ⦄ ≲ ⦃ z : b | c ⦄ **and**
atom x ♯ (z,z',c,c')
shows Θ ; Φ ; $\mathcal{B}$; Γ′@(x,b,$c'[z'{::=}V\text{-}var\ x]_{cv}$)$\#_\Gamma$Γ ; Δ ⊢ $s \Leftarrow \tau$
proof −

have *wf*: *wfG* Θ $\mathcal{B}$ (Γ′@((x,b,$c[z{::=}V\text{-}var\ x]_{cv}$)$\#_\Gamma$Γ)) **using** *check-s-wf assms* **by** *meson*
hence ∗:$x \notin$ *fst* ‘ *toSet* Γ′ **using** *wfG-inside-fresh* **by** *force*
have *wfG* Θ $\mathcal{B}$ ((x,b,$c[z{::=}V\text{-}var\ x]_{cv}$)$\#_\Gamma$Γ) **using** *wf wfG-suffix* **by** *metis*
hence *xfg*: *atom* x ♯ Γ **using** *wfG-elims* **by** *metis*
have $x \neq z'$ **using** *assms fresh-at-base fresh-prod4* **by** *metis*
hence *a2*: *atom* x ♯ c' **using** *assms fresh-prod4* **by** *metis*

have *atom* x ♯ (z', c', z, c, Γ) **proof** −
have $x \neq z$ **using** *assms* **using** *assms fresh-at-base fresh-prod4* **by** *metis*
hence *a1* : *atom* x ♯ c **using** *assms subtype-wf subtype-wf assms wfT-fresh-c xfg* **by** *meson*
thus *?thesis* **using** *a1 a2* ‹*atom* x ♯ (z,z',c,c')› *fresh-prod4 fresh-Pair xfg* **by** *simp*
qed
hence *wc1*: Θ; $\mathcal{B}$; (x, b, $c'[z'{::=}V\text{-}var\ x]_v$) $\#_\Gamma$ Γ $\models$ $c[z{::=}V\text{-}var\ x]_v$
using *subtype-valid assms fresh-prodN* **by** *metis*

have *vld*: Θ;$\mathcal{B}$; (Γ'@(x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) $\models$ $c[z::= V\text{-}var\ x]_{cv}$ **proof** −

have *toSet* ((x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) $\subseteq$ *toSet* (Γ'@(x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) **by** *auto*
moreover have *wfG* Θ $\mathcal{B}$ (Γ'@(x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) **proof** −
have $*$:*wfT* Θ $\mathcal{B}$ Γ ($\{\!|$ z' : b $|$ c' $|\!\}$) **using** *subtype-wf assms* **by** *meson*
moreover have *atom* x $\sharp$ (c',Γ) **using** *xfg a2* **by** *simp*
ultimately have *wfG* Θ $\mathcal{B}$ ((x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) **using** *wfT-wf-cons-flip freshers* **by** *blast*
thus *?thesis* **using** *wfG-replace-inside2 check-s-wf assms* **by** *metis*
qed
ultimately show *?thesis* **using** *wc1 valid-weakening subst-defs* **by** *metis*
qed
hence *wbc*: Θ; $\mathcal{B}$; Γ' @ (x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ $\vdash_{wf}$ $c[z::= V\text{-}var\ x]_{cv}$ **using** *valid.simps* **by** *auto*
have *wbc1*: Θ; $\mathcal{B}$; (x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ $\vdash_{wf}$ $c[z::= V\text{-}var\ x]_{cv}$ **using** *wc1 valid.simps subst-defs* **by** *auto*
have *wsX* (Γ'@((x,b,$c[z::= V\text{-}var\ x]_{cv}$)$\#_\Gamma\Gamma$)) [($x$, $c'[z'::= V\text{-}var\ x]_{cv}$)] **proof**
show *wsX* (Γ' @ (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) [] **using** *wsX-NilI* **by** *auto*
show *atom* x $\in$ *atom-dom* (Γ' @ (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) **by** *simp*
show $x \notin$ *fst* ‘ *set* [] **by** *auto*
qed
moreover have *replace-in-g-subtyped* Θ $\mathcal{B}$ (Γ'@((x,b,$c[z::= V\text{-}var\ x]_{cv}$)$\#_\Gamma\Gamma$)) [($x$, $c'[z'::= V\text{-}var\ x]_{cv}$)] ($\Gamma'$@($x$,$b$,$c'[z'::= V\text{-}var\ x]_{cv}$)$\#_\Gamma\Gamma$) **proof**
show *Some* (b, $c[z::= V\text{-}var\ x]_{cv}$) = *lookup* ($\Gamma'$ @ (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) x **using** *lookup-inside$*$* **by** *auto*
show Θ; $\mathcal{B}$; *replace-in-g* (Γ' @ (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) x ($c'[z'::= V\text{-}var\ x]_{cv}$) $\models$ $c[z::= V\text{-}var\ x]_{cv}$ **using** *vld replace-in-g-split wf* **by** *metis*
show *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* (Γ' @ (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) x ($c'[z'::= V\text{-}var\ x]_{cv}$)) [] ($\Gamma'$ @ (x, b, $c'[z'::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ)
using *replace-in-g-subtyped-empty wf* **by** *presburger*
show $x \notin$ *fst* ‘ *set* [] **by** *auto*
show Θ; $\mathcal{B}$; Γ' @ (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ $\vdash_{wf}$ $c'[z'::= V\text{-}var\ x]_{cv}$
proof(*rule wf-weakening*)
show ‹Θ; $\mathcal{B}$; (x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ $\vdash_{wf}$ $c'[z'::=[\ x\ ]^v]_{cv}$ › **using** *wfC-cons-switch*[*OF wbc1*] *wf-weakening*(*6*) *check-s-wf assms toSet.simps* **by** *metis*
show ‹Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (x, b, $c[z::=[\ x\ ]^v]_{cv}$) $\#_\Gamma$ Γ › **using** *wfC-cons-switch*[*OF wbc1*] *wf-weakening*(*6*) *check-s-wf assms toSet.simps* **by** *metis*
show ‹*toSet* ((x, b, $c[z::= V\text{-}var\ x]_{cv}$) $\#_\Gamma$ Γ) $\subseteq$ *toSet* (Γ' @ (x, b, $c[z::=[\ x\ ]^v]_{cv}$) $\#_\Gamma$ Γ)› **using** *append-g.simps toSet.simps* **by** *auto*
qed
qed
ultimately show *?thesis* **using** *ctx-subtype-s-rigs*(*1*)[*OF assms*(*1*)] **by** *presburger*
qed

end

Chapter 14

Immutable Variable Substitution Lemmas

Lemmas that show that types are preserved, in some way, under immutable variable substitution

14.1 Proof Methods

method *subst-mth* = (*metis subst-g-inside infer-e-wf infer-v-wf infer-v-wf*)

method *subst-tuple-mth* **uses** *add* = (
(*unfold fresh-prodN*), (*simp add*: *add*)+,
(*rule*,*metis fresh-z-subst-g add fresh-Pair*),
(*metis fresh-subst-dv add fresh-Pair*))

14.2 Prelude

lemma *subst-top-eq*:
$\{\!\mid z : b \mid TRUE \mid\!\} = \{\!\mid z : b \mid TRUE \mid\!\}[x::=v]_{\tau v}$
proof −
obtain z'::x **and** c' **where** *zeq*: $\{\!\mid z : b \mid TRUE \mid\!\} = \{\!\mid z' : b \mid c' \mid\!\} \wedge atom\ z' \sharp (x,v)$ **using** *obtain-fresh-z2 b-of.simps* **by** *metis*
hence $\{\!\mid z' : b \mid TRUE \mid\!\}[x::=v]_{\tau v} = \{\!\mid z' : b \mid TRUE \mid\!\}$ **using** *subst-tv.simps subst-cv.simps* **by** *metis*
moreover have c' = *C-true* **using** τ*.eq-iff Abs1-eq-iff*(*3*) *c.fresh flip-fresh-fresh* **by** (*metis zeq*)
ultimately show *?thesis* **using** *zeq* **by** *metis*
qed

lemma *wfD-subst*:
fixes τ_1::τ **and** v::v **and** Δ::Δ **and** Θ::Θ **and** Γ::Γ
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **and** *wfD* $\Theta\ \mathcal{B}\ (\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma))\ \Delta$ **and** *b-of* $\tau_1 = b_1$
shows $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$
proof −
have $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b_1$ **using** *infer-v-v-wf assms* **by** *auto*
moreover have $(\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma))[x::=v]_{\Gamma v} = \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma$ **using** *subst-g-inside wfD-wf assms* **by** *metis*
ultimately show *?thesis* **using** *wf-subst assms* **by** *metis*
qed

lemma *subst-v-c-of*:
assumes *atom xa* ♯ (*v*,*x*)
shows *c-of* $t[x::=v]_{\tau v}$ *xa* = (*c-of t xa*)$[x::=v]_{cv}$
using *assms* **proof**(*nominal-induct t avoiding*: *x v xa rule*:*τ.strong-induct*)
case (*T-refined-type z′ b′ c′*)
then have *c-of* ⦃ *z′* : *b′* | *c′* ⦄$[x::=v]_{\tau v}$ *xa* = *c-of* ⦃ *z′* : *b′* | $c'[x::=v]_{cv}$ ⦄ *xa*
using *subst-tv.simps fresh-Pair* **by** *metis*
also have ... = $c'[x::=v]_{cv}$ $[z'::= V\text{-}var\ xa]_{cv}$ **using** *c-of.simps T-refined-type* **by** *metis*
also have ... = *c′* $[z'::= V\text{-}var\ xa]_{cv}[x::=v]_{cv}$
using *subst-cv-commute-full*[*of z′ v x V-var xa c′*] *subst-v-c-def T-refined-type fresh-Pair fresh-at-base v.fresh fresh-x-neq* **by** *metis*
finally show *?case* **using** *c-of.simps T-refined-type* **by** *metis*
qed

14.3 Context

lemma *subst-lookup*:
assumes *Some* (*b*,*c*) = *lookup* $(\Gamma'@((x,b_1,c_1)\#_\Gamma\Gamma))$ *y* **and** *x* ≠ *y* **and** *wfG* Θ ℬ $(\Gamma'@((x,b_1,c_1)\#_\Gamma\Gamma))$
shows ∃ *d*. *Some* (*b*,*d*) = *lookup* $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ *y*
using *assms* **proof**(*induct* Γ′ *rule*: Γ*-induct*)
case *GNil*
hence *Some* (*b*,*c*) = *lookup* Γ *y* **by** (*simp add*: *assms*(*1*))
then show *?case* **using** *subst-gv.simps* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
show *?case* **proof**(*cases x1* = *x*)
case *True*
hence *atom x* ♯ (Γ*1* @ (*x*, b_1, c_1) $\#_\Gamma$ Γ) **using** *GCons wfG-elims*(*2*)
append-g.simps **by** *metis*
moreover have *atom x* ∈ *atom-dom* (Γ*1* @ (*x*, b_1, c_1) $\#_\Gamma$ Γ) **by** *simp*
ultimately show *?thesis*
using *forget-subst-gv not-GCons-self2 subst-gv.simps append-g.simps*
by (*metis GCons.prems*(*3*) *True wfG-cons-fresh2*)
next
case *False*
hence ((*x1*,*b1*,*c1*) $\#_\Gamma$ Γ*1*)$[x::=v]_{\Gamma v}$ = (*x1*,*b1*,*c1*$[x::=v]_{cv}$)$\#_\Gamma$Γ*1*$[x::=v]_{\Gamma v}$ **using** *subst-gv.simps* **by** *auto*
then show *?thesis* **proof**(*cases x1*=*y*)
case *True*
then show *?thesis* **using** *GCons* **using** *lookup.simps*
by (*metis* ‹((*x1*, *b1*, *c1*) $\#_\Gamma$ Γ*1*)$[x::=v]_{\Gamma v}$ = (*x1*, *b1*, *c1*$[x::=v]_{cv}$) $\#_\Gamma$ Γ*1*$[x::=v]_{\Gamma v}$› *append-g.simps fst-conv option.inject*)
next
case *False*
then show *?thesis* **using** *GCons* **using** *lookup.simps*
using ‹((*x1*, *b1*, *c1*) $\#_\Gamma$ Γ*1*)$[x::=v]_{\Gamma v}$ = (*x1*, *b1*, *c1*$[x::=v]_{cv}$) $\#_\Gamma$ Γ*1*$[x::=v]_{\Gamma v}$› *append-g.simps* Γ*.distinct* Γ*.inject wfG.simps wfG-elims* **by** *metis*
qed
qed
qed

14.4 Validity

lemma *subst-self-valid*:
fixes $v::v$
assumes $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{\!| z : b \mid c |\!\}$ **and** $atom\ z \sharp v$
shows $\Theta ; \mathcal{B} ; G \models c[z::=v]_{cv}$
proof –
have $c= (CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v\ )$ **using** *infer-v-form2 assms* **by** *presburger*
hence $c[z::=v]_{cv} = (CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v\ )[z::=v]_{cv}$ **by** *auto*
also have ... $= (((CE\text{-}val\ (V\text{-}var\ z))[z::=v]_{cev}) == ((CE\text{-}val\ v)[z::=v]_{cev}))$ **by** *fastforce*
also have ... $= ((CE\text{-}val\ v) == ((CE\text{-}val\ v)[z::=v]_{cev}))$ **using** *subst-cev.simps subst-vv.simps* **by** *presburger*
also have ... $= (CE\text{-}val\ v == CE\text{-}val\ v\ )$ **using** *infer-v-form subst-cev.simps assms forget-subst-vv* **by** *presburger*
finally have $*{:}c[z::=v]_{cv} = (CE\text{-}val\ v == CE\text{-}val\ v\ )$ **by** *auto*

have $**{:}\Theta ; \mathcal{B} ; G\vdash_{wf} CE\text{-}val\ v : b$ **using** *wfCE-valI assms infer-v-v-wf b-of.simps* **by** *metis*

show *?thesis* **proof**(*rule validI*)
show $\Theta ; \mathcal{B} ; G\vdash_{wf} c[z::=v]_{cv}$ **proof** –
have $\Theta ; \mathcal{B} ; G\vdash_{wf} v : b$ **using** *infer-v-v-wf assms b-of.simps* **by** *metis*
moreover have $\Theta \vdash_{wf} ([]::\Phi) \wedge \Theta ; \mathcal{B} ; G\vdash_{wf} []_\Delta$ **using** *wfD-emptyI wfPhi-emptyI infer-v-wf assms* **by** *auto*
ultimately show *?thesis* **using** $*$ *wfCE-valI wfC-eqI* **by** *metis*
qed
show $\forall i.\ wfI\ \Theta\ G\ i \wedge is\text{-}satis\text{-}g\ i\ G \longrightarrow is\text{-}satis\ i\ c[z::=v]_{cv}$ **proof**(*rule,rule*)
fix *i*
assume ‹$wfI\ \Theta\ G\ i \wedge is\text{-}satis\text{-}g\ i\ G$›
thus ‹$is\text{-}satis\ i\ c[z::=v]_{cv}$› **using** $*$ $**$ *is-satis-eq* **by** *auto*
qed
qed
qed

lemma *subst-valid-simple*:
fixes $v::v$
assumes $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{\!| z0 : b \mid c0 |\!\}$ **and**
$atom\ z0 \sharp c$ **and** $atom\ z0 \sharp v$
$\Theta; \mathcal{B} ; (z0,b,c0)\#_\Gamma G \models c[z::=V\text{-}var\ z0]_{cv}$
shows $\Theta ; \mathcal{B} ; G \models c[z::=v]_{cv}$
proof –
have $\Theta ; \mathcal{B} ; G \models c0[z0::=v]_{cv}$ **using** *subst-self-valid assms* **by** *metis*
moreover have $atom\ z0 \sharp G$ **using** *assms valid-wf-all* **by** *meson*
moreover have $wfV\ \Theta\ \mathcal{B}\ G\ v\ b$ **using** *infer-v-v-wf assms b-of.simps* **by** *metis*
moreover have $(c[z::=V\text{-}var\ z0]_{cv})[z0::=v]_{cv} = c[z::=v]_{cv}$ **using** *subst-v-simple-commute assms subst-v-c-def* **by** *metis*
ultimately show *?thesis* **using** *valid-trans assms subst-defs* **by** *metis*
qed

lemma *wfI-subst1*:
assumes $wfI\ \Theta\ (G'[x::=v]_{\Gamma v}\ @\ G)\ i$ **and** $wfG\ \Theta\ \mathcal{B}\ (G'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ G)$ **and** $eval\text{-}v\ i\ v\ sv$ **and** $wfRCV\ \Theta\ sv\ b$
shows $wfI\ \Theta\ (G'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ G)\ (\ i(\ x \mapsto sv))$
proof –

{
fix *xa*::*x* **and** *ba*::*b* **and** *ca*::*c*
assume *as*: $(xa,ba,ca) \in toSet\ ((G' @ ((x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ G)))$
then have $\exists s.\ Some\ s = (i(x \mapsto sv))\ xa \wedge wfRCV\ \Theta\ s\ ba$
proof(*cases x=xa*)
case *True*
have $Some\ sv = (i(x \mapsto sv))\ x \wedge wfRCV\ \Theta\ sv\ b$ **using** *as assms wfI-def* **by** *auto*
moreover have *b=ba* **using** *assms as True wfG-member-unique* **by** *metis*
ultimately show *?thesis* **using** *True* **by** *auto*
next
case *False*

then obtain ca' **where** $(xa,\ ba,\ ca') \in toSet\ (G'[x::=v]_{\Gamma v}\ @\ G)$ **using** *wfG-member-subst2 assms as* **by** *metis*
then obtain *s* **where** $Some\ s = i\ xa \wedge wfRCV\ \Theta\ s\ ba$ **using** *wfI-def assms False* **by** *blast*
thus *?thesis* **using** *False* **by** *auto*
qed
}
from *this* **show** *?thesis* **using** *wfI-def allI* **by** *blast*
qed

lemma *subst-valid*:
fixes *v*::*v* **and** c'::*c* **and** Γ ::Γ
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \models c[z::=v]_{cv}$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\vdash_{wf}\ v : b$ **and**
$\Theta\ ;\ \mathcal{B}\vdash_{wf}\ \Gamma$ **and** $atom\ x \sharp\ c$ **and** $atom\ x \sharp\ \Gamma$ **and**
$\Theta\ ;\ \mathcal{B}\vdash_{wf}\ (\Gamma'@(x,b,c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma)$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma \models c'$ (**is** $\Theta\ ;\ \mathcal{B};\ ?G \models c'$)
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}@\Gamma \models c'[x::=v]_{cv}$
proof −
have $*$:$wfC\ \Theta\ \mathcal{B}\ (\Gamma'@(x,b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma)\ c'$ **using** *valid.simps assms* **by** *metis*
hence $wfC\ \Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma)\ (c'[x::=v]_{cv})$ **using** *wf-subst*(*2*)[*OF* ∗] *b-of.simps assms subst-g-inside wfC-wf* **by** *metis*
moreover have $\forall i.\ wfI\ \Theta\ (\Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma)\ i \wedge is\text{-}satis\text{-}g\ i\ (\Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma) \longrightarrow is\text{-}satis\ i\ (c'[x::=v]_{cv})$

proof(*rule*,*rule*)
fix *i*
assume *as*: $wfI\ \Theta\ (\Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma)\ i \wedge is\text{-}satis\text{-}g\ i\ (\Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma)$

hence *wfig*: $wfI\ \Theta\ \Gamma\ i$ **using** *wfI-suffix infer-v-wf assms* **by** *metis*
then obtain *s* **where** *s*:*eval-v i v s* **and** *b*:$wfRCV\ \Theta\ s\ b$ **using** *eval-v-exist infer-v-v-wf b-of.simps assms* **by** *metis*

have *is1*: $is\text{-}satis\text{-}g\ (\ i(\ x \mapsto s))\ (\Gamma'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma)$ **proof**(*rule is-satis-g-i-upd2*)
show $is\text{-}satis\ (i(x \mapsto s))\ (c[z::=[x]^v]_{cv})$ **proof** −
have $is\text{-}satis\ i\ \ (c[z::=v]_{cv})$
using *subst-valid-simple assms as valid.simps infer-v-wf assms*
is-satis-g-suffix wfI-suffix **by** *metis*
hence $is\text{-}satis\ i\ ((c[z::=[x]^v]_{cv})[x::=v]_{cv})$ **using** *assms subst-v-simple-commute*[*of x c z v*] *subst-v-c-def* **by** *metis*
moreover have $\Theta\ ;\ \mathcal{B}\ ;\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma \vdash_{wf}\ c[z::=[x]^v]_{cv}$ **using** *wfC-refl wfG-suffix assms* **by** *metis*

moreover have Θ ; $\mathcal{B}$; $\Gamma\vdash_{wf}$ v : b **using** *assms infer-v-v-wf b-of.simps* **by** *metis*
ultimately show *?thesis* **using** *subst-c-satis*[*OF s* , *of* Θ $\mathcal{B}$ x b $c[z::=[x]^v]_{cv}$ Γ $c[z::=[x]^v]_{cv}$] *wfig* **by** *auto*
qed
show *atom x* $\sharp$ Γ **using** *assms* **by** *metis*
show *wfG* Θ $\mathcal{B}$ (Γ' @ (x, b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ) **using** *valid-wf-all assms* **by** *metis*
show Θ ; $\mathcal{B}$; $\Gamma\vdash_{wf}$ v : b **using** *assms infer-v-v-wf* **by** *force*
show i $[\![$ v $]\!]$ $\sim$ s **using** *s* **by** *auto*
show Θ ; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ i **using** *as* **by** *auto*
show $i \models \Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *as* **by** *auto*
qed
hence *is-satis* (i($x \mapsto s$)) c' **proof** −
have *wfI* Θ (Γ' @ (x, b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ) (i($x \mapsto s$))
using *wfI-subst1*[*of* Θ Γ' x v Γ i $\mathcal{B}$ b c z s] *as b s assms* **by** *metis*
thus *?thesis* **using** *is1 valid.simps assms* **by** *presburger*
qed

thus *is-satis i* ($c'[x::=v]_{cv}$) **using** *subst-c-satis-full*[*OF s*] *valid.simps as infer-v-v-wf b-of.simps assms* **by** *metis*

qed
ultimately show *?thesis* **using** *valid.simps* **by** *auto*
qed

lemma *subst-valid-infer-v*:
fixes *v*::*v* **and** *c'*::*c*
assumes Θ ; $\mathcal{B}$; $G \vdash v \Rightarrow \{\!|$ $z0$: b $|$ $c0$ $|\!\}$ **and** *atom x* $\sharp$ c **and** *atom x* $\sharp$ G **and** *wfG* Θ $\mathcal{B}$ (G'@(x,b,$c[z::=[x]^v]_{cv}$) $\#_\Gamma$ G) **and** *atom z0* $\sharp$ v
Θ;$\mathcal{B}$;($z0$,b,$c0$)$\#_\Gamma G \models c[z::=V\text{-}var\ z0]_{cv}$ **and** *atom z0* $\sharp$ c **and**
Θ;$\mathcal{B}$;G'@(x,b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ $G \models c'$ (**is** Θ ; $\mathcal{B}$; *?G* $\models c'$)
shows Θ;$\mathcal{B}$;$G'[x::=v]_{\Gamma v}$@$G \models c'[x::=v]_{cv}$
proof −
have Θ ; $\mathcal{B}$; $G \models c[z::=v]_{cv}$
using *infer-v-wf subst-valid-simple valid.simps assms* **using** *subst-valid-simple assms valid.simps infer-v-wf assms*
is-satis-g-suffix wfI-suffix **by** *metis*
moreover have *wfV* Θ $\mathcal{B}$ G v b **and** *wfG* Θ $\mathcal{B}$ G
using *assms infer-v-wf b-of.simps* **apply** *metis* **using** *assms infer-v-wf* **by** *metis*
ultimately show *?thesis* **using** *assms subst-valid* **by** *metis*
qed

14.5 Subtyping

lemma *subst-subtype*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow$ ($\{\!|z0{:}b|c0|\!\}$) **and**
Θ;$\mathcal{B}$;$\Gamma \vdash$ ($\{\!|z0{:}b|c0|\!\}$) $\lesssim$ ($\{\!|$ z : b $|$ c $|\!\}$) **and**
Θ;$\mathcal{B}$;Γ'@((x,b,$c[z::=[x]^v]_{cv}$)$\#_\Gamma\Gamma$) $\vdash$ ($\{\!|$ $z1$: $b1$ $|$ $c1$ $|\!\}$) $\lesssim$ ($\{\!|$ $z2$: $b1$ $|$ $c2$ $|\!\}$) (**is** Θ ; $\mathcal{B}$; *?G1* $\vdash$ *?t1* $\lesssim$ *?t2*) **and**
atom z $\sharp$ (x,v) $\wedge$ *atom z0* $\sharp$ (c,x,v,z,Γ) $\wedge$ *atom z1* $\sharp$ (x,v) $\wedge$ *atom z2* $\sharp$ (x,v) **and** *wsV* Θ $\mathcal{B}$ Γ v
shows Θ;$\mathcal{B}$;$\Gamma'[x::=v]_{\Gamma v}$@$\Gamma \vdash$ $\{\!|$ $z1$: $b1$ $|$ $c1$ $|\!\}[x::=v]_{\tau v}$ $\lesssim$ $\{\!|$ $z2$: $b1$ $|$ $c2$ $|\!\}[x::=v]_{\tau v}$
proof −

have *z2*: *atom z2* ♯ (*x*,*v*) **using** *assms* **by** *auto*
hence *x* ≠ *z2* **by** *auto*

obtain *xx*::*x* **where** *xxf*: *atom xx* ♯ (*x*,*z1*, *c1*, *z2*, *c2*, Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ, *c1*[*x*::=*v*]$_{cv}$, *c2*[*x*::=*v*]$_{cv}$, Γ′[*x*::=*v*]$_{Γv}$ @ Γ,
(Θ , *ℬ* , Γ′[*x*::=*v*]$_{Γv}$@Γ, *z1* , *c1*[*x*::=*v*]$_{cv}$, *z2* , *c2*[*x*::=*v*]$_{cv}$)) (**is** *atom xx* ♯ *?tup*)
using *obtain-fresh* **by** *blast*
hence *xxf2*: *atom xx* ♯ (*z1*, *c1*, *z2*, *c2*, Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ) **using** *fresh-prod9 fresh-prod5* **by** *fast*

have *vd1*: Θ;*ℬ*;((*xx*, *b1*, *c1*[*z1*::=*V-var xx*]$_{cv}$) #$_Γ$ Γ′)[*x*::=*v*]$_{Γv}$ @ Γ ⊨ (*c2*[*z2*::=*V-var xx*]$_{cv}$)[*x*::=*v*]$_{cv}$
proof(*rule subst-valid-infer-v*[*of* Θ - - - *z0 b c0* - *c*, **where** *z*=*z*])
show Θ ; *ℬ* ; Γ ⊢ *v* ⇒ ⦃ *z0* : *b* | *c0* ⦄ **using** *assms* **by** *auto*

show *xf*: *atom x* ♯ Γ **using** *subtype-g-wf wfG-inside-fresh-suffix assms* **by** *metis*

show *atom x* ♯ *c* **proof** −
have *wfT* Θ *ℬ* Γ (⦃ *z* : *b* | *c* ⦄) **using** *subtype-wf*[*OF assms*(*2*)] **by** *auto*
moreover have *x* ≠ *z* **using** *assms*(*4*)
using *fresh-Pair not-self-fresh* **by** *blast*
ultimately show *?thesis* **using** *xf wfT-fresh-c assms* **by** *presburger*
qed

show Θ ; *ℬ*⊢$_{wf}$ ((*xx*, *b1*, *c1*[*z1*::=*V-var xx*]$_{cv}$) #$_Γ$ Γ′) @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ
proof(*subst append-g.simps*,*rule wfG-consI*)
show ∗: ‹ Θ ; *ℬ*⊢$_{wf}$ Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ › **using** *subtype-g-wf assms* **by** *metis*
show ‹*atom xx* ♯ Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ› **using** *xxf fresh-prod9* **by** *metis*
show ‹ Θ ; *ℬ*⊢$_{wf}$ *b1* › **using** *subtype-elims*[*OF assms*(*3*)] *wfT-wfC wfC-wf wfG-cons* **by** *metis*
show Θ ; *ℬ* ; (*xx*, *b1*, *TRUE*) #$_Γ$ Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ ⊢$_{wf}$ *c1*[*z1*::=*V-var xx*]$_{cv}$ **proof**(*rule wfT-wfC*)
have ⦃ *z1* : *b1* | *c1* ⦄ = ⦃ *xx* : *b1* | *c1*[*z1*::=*V-var xx*]$_{cv}$ ⦄ **using** *xxf fresh-prod9 type-eq-subst xxf2 fresh-prodN* **by** *metis*
thus Θ ; *ℬ* ; Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ ⊢$_{wf}$ ⦃ *xx* : *b1* | *c1*[*z1*::=*V-var xx*]$_{cv}$ ⦄ **using** *subtype-wfT*[*OF assms*(*3*)] **by** *metis*
show *atom xx* ♯ Γ′ @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ **using** *xxf fresh-prod9* **by** *metis*
qed
qed

show *atom z0* ♯ *v* **using** *assms fresh-prod5* **by** *auto*
have Θ ; *ℬ* ; (*z0*, *b*, *c0*) #$_Γ$ Γ ⊨ *c*[*z*::=*V-var z0*]$_v$
apply(*rule obtain-fresh*[*of* (*z0*,*c0*, Γ, *c*, *z*)],*rule subtype-valid*[*OF assms*(*2*), *THEN valid-flip*],
(*fastforce simp add*: *assms fresh-prodN*)+) **done**
thus Θ ; *ℬ* ; (*z0*, *b*, *c0*) #$_Γ$ Γ ⊨ *c*[*z*::=*V-var z0*]$_{cv}$ **using** *subst-defs* **by** *auto*

show *atom z0* ♯ *c* **using** *assms fresh-prod5* **by** *auto*
show Θ ; *ℬ* ; ((*xx*, *b1*, *c1*[*z1*::=*V-var xx*]$_{cv}$) #$_Γ$ Γ′) @ (*x*, *b*, *c*[*z*::=[*x*]^*v*]$_{cv}$) #$_Γ$ Γ ⊨ *c2*[*z2*::=*V-var xx*]$_{cv}$
using *subtype-valid assms*(*3*) *xxf xxf2 fresh-prodN append-g.simps subst-defs* **by** *metis*
qed

have *xfw1*: *atom z1* ♯ *v* ∧ *atom x* ♯ [*xx*]^*v* ∧ *x* ≠ *z1*
apply(*intro conjI*)

apply(*simp add*: *assms xxf fresh-at-base fresh-prodN freshers fresh-x-neq*)+
using *fresh-x-neq fresh-prodN xxf* **apply** *blast*
using *fresh-x-neq fresh-prodN assms* **by** *blast*

have *xfw2*: $atom\ z2 \sharp v \wedge atom\ x \sharp [\ xx\ ]^v \wedge x \neq z2$
apply(*auto simp add*: *assms xxf fresh-at-base fresh-prodN freshers*)
by(*insert xxf fresh-at-base fresh-prodN assms*, *fast*+)

have *wf1*: $wfT\ \Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)\ (\{\!\!\{\ z1 : b1\ \mid c1[x::=v]_{cv}\ \}\!\!\})$ **proof** −
have $wfT\ \Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)\ (\{\!\!\{\ z1 : b1\ \mid c1\ \}\!\!\})[x::=v]_{\tau v}$
using *wf-subst*(*4*) *assms b-of.simps infer-v-v-wf subtype-wf subst-tv.simps subst-g-inside wfT-wf* **by** *metis*
moreover have $atom\ z1 \sharp (x,v)$ **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *subst-tv.simps* **by** *auto*
qed
moreover have *wf2*: $wfT\ \Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)\ (\{\!\!\{\ z2 : b1\ \mid c2[x::=v]_{cv}\ \}\!\!\})$ **proof** −
have $wfT\ \Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)\ (\{\!\!\{\ z2 : b1\ \mid c2\ \}\!\!\})[x::=v]_{\tau v}$ **using** *wf-subst*(*4*) *assms b-of.simps infer-v-v-wf subtype-wf subst-tv.simps subst-g-inside wfT-wf* **by** *metis*
moreover have $atom\ z2 \sharp (x,v)$ **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *subst-tv.simps* **by** *auto*
qed
moreover have $\Theta ; \mathcal{B} ; (xx, b1, c1[x::=v]_{cv}[z1::=V\text{-}var\ xx]_{cv}) \#_\Gamma (\Gamma'[x::=v]_{\Gamma v} @ \Gamma) \models (c2[x::=v]_{cv})[z2::=V\text{-}var\ xx]_{cv}$ **proof** −
have $xx \neq x$ **using** *xxf fresh-Pair fresh-at-base* **by** *fast*
hence $((xx, b1, subst\text{-}cv\ c1\ z1\ (V\text{-}var\ xx)) \#_\Gamma \Gamma')[x::=v]_{\Gamma v} = (xx, b1, (subst\text{-}cv\ c1\ z1\ (V\text{-}var\ xx))[x::=v]_{cv}) \#_\Gamma (\Gamma'[x::=v]_{\Gamma v})$
using *subst-gv.simps* **by** *auto*
moreover have $(c1[z1::=V\text{-}var\ xx]_{cv})[x::=v]_{cv} = (c1[x::=v]_{cv})[z1::=V\text{-}var\ xx]_{cv}$ **using** *subst-cv-commute-full xfw1* **by** *metis*
moreover have $c2[z2::=[\ xx\ ]^v]_{cv}[x::=v]_{cv} = (c2[x::=v]_{cv})[z2::=V\text{-}var\ xx]_{cv}$ **using** *subst-cv-commute-full xfw2* **by** *metis*
ultimately show *?thesis* **using** *vd1 append-g.simps* **by** *metis*
qed
moreover have $atom\ xx \sharp (\Theta , \mathcal{B} , \Gamma'[x::=v]_{\Gamma v}@\Gamma, z1 , c1[x::=v]_{cv} , z2 ,c2[x::=v]_{cv})$
using *xxf fresh-prodN* **by** *metis*
ultimately have $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v}@\Gamma \vdash \{\!\!\{\ z1 : b1 \mid c1[x::=v]_{cv}\ \}\!\!\} \lesssim \{\!\!\{\ z2 : b1 \mid c2[x::=v]_{cv}\ \}\!\!\}$
using *subtype-baseI subst-defs* **by** *metis*
thus *?thesis* **using** *subst-tv.simps assms* **by** *presburger*
qed

lemma *subst-subtype-tau*:
fixes $v::v$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$ **and**
$\Theta ; \mathcal{B} ; \Gamma \vdash \tau \lesssim (\{\!\!\{\ z : b \mid c\ \}\!\!\})$
$\Theta ; \mathcal{B} ; \Gamma'@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash \tau 1 \lesssim \tau 2$ **and**
$atom\ z \sharp (x,v)$
shows $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v}@\Gamma \vdash \tau 1[x::=v]_{\tau v} \lesssim \tau 2[x::=v]_{\tau v}$
proof −
obtain *z0* **and** *b0* **and** *c0* **where** *zbc0*: $\tau=(\{\!\!\{\ z0 : b0 \mid c0\ \}\!\!\}) \wedge atom\ z0 \sharp (c,x,v,z,\Gamma)$
using *obtain-fresh-z* **by** *metis*
obtain *z1* **and** *b1* **and** *c1* **where** *zbc1*: $\tau 1=(\{\!\!\{\ z1 : b1 \mid c1\ \}\!\!\}) \wedge atom\ z1 \sharp (x,v)$
using *obtain-fresh-z* **by** *metis*

obtain *z2* **and** *b2* **and** *c2* **where** *zbc2*: $\tau 2$=(⦃ *z2* : *b2* | *c2* ⦄) ∧ *atom z2* ♯ (*x*,*v*)
  using *obtain-fresh-z* **by** *metis*

have *b0*=*b* **using** *subtype-eq-base zbc0 assms* **by** *blast*

hence *vinf*: Θ ; ℬ ; Γ ⊢ *v* ⇒ ⦃ *z0* : *b* | *c0* ⦄ **using** *assms zbc0* **by** *blast*
have *vsub*: Θ ; ℬ ; Γ ⊢⦃ *z0* : *b* | *c0* ⦄ ≲ ⦃ *z* : *b* | *c* ⦄ **using** *assms zbc0* ‹*b0*=*b*› **by** *blast*
have *beq*:*b1*=*b2* **using** *subtype-eq-base*
  using *zbc1 zbc2 assms* **by** *blast*
have Θ ; ℬ ; Γ′$[x::=v]_{\Gamma v}$ @ Γ ⊢ ⦃ *z1* : *b1* | *c1* ⦄$[x::=v]_{\tau v}$ ≲ ⦃ *z2* : *b1* | *c2* ⦄$[x::=v]_{\tau v}$
proof(*rule subst-subtype*[*OF vinf vsub*])
  show Θ ; ℬ ; Γ′@((*x*,*b*,*c*$[z::=[x]^v]_{cv}$)$\#_\Gamma$Γ) ⊢ ⦃ *z1* : *b1* | *c1* ⦄ ≲ ⦃ *z2* : *b1* | *c2* ⦄
    using *beq assms zbc1 zbc2* **by** *auto*
  show *atom z* ♯ (*x*, *v*) ∧ *atom z0* ♯ (*c*, *x*, *v*, *z*, Γ) ∧ *atom z1* ♯ (*x*, *v*) ∧ *atom z2* ♯ (*x*, *v*)
    using *zbc0 zbc1 zbc2 assms* **by** *blast*
  show *wfV* Θ ℬ Γ *v* (*b-of* τ) **using** *infer-v-wf assms* **by** *simp*
qed

thus *?thesis* **using** *zbc1 zbc2* ‹*b1*=*b2*› *assms* **by** *blast*
qed

lemma *subtype-if1*:
  fixes *v*::*v*
  assumes *P* ; ℬ ; Γ ⊢ *t1* ≲ *t2* **and** *wfV P* ℬ Γ *v* (*base-for-lit l*) **and**
    atom z1 ♯ *v* **and** *atom z2* ♯ *v* **and** *atom z1* ♯ *t1* **and** *atom z2* ♯ *t2* **and** *atom z1* ♯ Γ **and** *atom z2* ♯ Γ
  shows *P* ; ℬ ; Γ ⊢ ⦃ *z1* : *b-of t1* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t1 z1*) ⦄ ≲ ⦃ *z2* : *b-of t2* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t2 z2*) ⦄
proof −
  obtain *z1′* **where** *t1*: *t1* = ⦃ *z1′* : *b-of t1* | *c-of t1 z1′*⦄ ∧ *atom z1′* ♯ (*z1*,Γ,*t1*) **using** *obtain-fresh-z-c-of* **by** *metis*
  obtain *z2′* **where** *t2*: *t2* = ⦃ *z2′* : *b-of t2* | *c-of t2 z2′*⦄ ∧ *atom z2′* ♯ (*z2*,*t2*) **using** *obtain-fresh-z-c-of* **by** *metis*
  have *beq*:*b-of t1* = *b-of t2* **using** *subtype-eq-base2 assms* **by** *auto*

  have *c1*: (*c-of t1 z1′*)$[z1'::=[\ z1\ ]^v]_{cv}$ = *c-of t1 z1* **using** *c-of-switch t1 assms* **by** *simp*
  have *c2*: (*c-of t2 z2′*)$[z2'::=[\ z2\ ]^v]_{cv}$ = *c-of t2 z2* **using** *c-of-switch t2 assms* **by** *simp*

  have *P* ; ℬ ; Γ ⊢ ⦃ *z1* : *b-of t1* | $[\ v\ ]^{ce}$ == $[\ [\ l\ ]^v\ ]^{ce}$ *IMP* (*c-of t1 z1′*)$[z1'::=[z1]^v]_v$ ⦄ ≲ ⦃ *z2* : *b-of t1* | $[\ v\ ]^{ce}$ == $[\ [\ l\ ]^v\ ]^{ce}$ *IMP* (*c-of t2 z2′*)$[z2'::=[z2]^v]_v$ ⦄
  proof(*rule subtype-if*)
    show ‹*P* ; ℬ ; Γ ⊢ ⦃ *z1′* : *b-of t1* | *c-of t1 z1′* ⦄ ≲ ⦃ *z2′* : *b-of t1* | *c-of t2 z2′* ⦄› **using** *t1 t2 assms beq* **by** *auto*
    show ‹ *P* ; ℬ ; Γ $\vdash_{wf}$ ⦃ *z1* : *b-of t1* | $[\ v\ ]^{ce}$ == $[\ [\ l\ ]^v\ ]^{ce}$ *IMP* (*c-of t1 z1′*)$[z1'::=[\ z1\ ]^v]_v$ ⦄ › **using** *wfT-wfT-if-rev assms subtype-wfT c1 subst-defs* **by** *metis*
    show ‹ *P* ; ℬ ; Γ $\vdash_{wf}$ ⦃ *z2* : *b-of t1* | $[\ v\ ]^{ce}$ == $[\ [\ l\ ]^v\ ]^{ce}$ *IMP* (*c-of t2 z2′*)$[z2'::=[\ z2\ ]^v]_v$ ⦄ › **using** *wfT-wfT-if-rev assms subtype-wfT c2 subst-defs beq* **by** *metis*
    show ‹*atom z1* ♯ *v*› **using** *assms* **by** *auto*
    show ‹*atom z1′* ♯ Γ› **using** *t1* **by** *auto*
    show ‹*atom z1* ♯ *c-of t1 z1′*› **using** *t1 assms c-of-fresh* **by** *force*
    show ‹*atom z2* ♯ *c-of t2 z2′*› **using** *t2 assms c-of-fresh* **by** *force*
    show ‹*atom z2* ♯ *v*› **using** *assms* **by** *auto*

qed
then show *?thesis* **using** *t1 t2 assms c1 c2 beq subst-defs* **by** *metis*
qed

14.6 Values

lemma *subst-infer-aux*:
fixes τ_1::τ **and** v'::v
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau_1$ **and** Θ ; $\mathcal{B}$; $\Gamma' \vdash v' \Rightarrow \tau_2$ **and** *b-of* τ_1 = *b-of* τ_2
shows $\tau_1 = (\tau_2[x::=v]_{\tau v})$
proof −
obtain *z1* **and** *b1* **where** *zb1*: τ_1 = (⦃ *z1* : *b1* | *C-eq* (*CE-val* (*V-var z1*)) (*CE-val* ($v'[x::=v]_{vv}$)) ⦄) ∧ *atom z1* ♯ ((*CE-val* ($v'[x::=v]_{vv}$), *CE-val v*),$v'[x::=v]_{vv}$)
using *infer-v-form-fresh*[*OF assms*(*1*)] **by** *fastforce*
obtain *z2* **and** *b2* **where** *zb2*: τ_2 = (⦃ *z2* : *b2* | *C-eq* (*CE-val* (*V-var z2*)) (*CE-val* v') ⦄) ∧ *atom z2* ♯ ((*CE-val* ($v'[x::=v]_{vv}$), *CE-val v,x,v*),v')
using *infer-v-form-fresh* [*OF assms*(*2*)] **by** *fastforce*
have *beq*: *b1* = *b2* **using** *assms zb1 zb2* **by** *simp*

hence (⦃ *z2* : *b2* | *C-eq* (*CE-val* (*V-var z2*)) (*CE-val* v') ⦄)$[x::=v]_{\tau v}$ = (⦃ *z2* : *b2* | *C-eq* (*CE-val* (*V-var z2*)) (*CE-val* ($v'[x::=v]_{vv}$)) ⦄)
using *subst-tv.simps subst-cv.simps subst-ev.simps forget-subst-vv*[*of x V-var z2*] *zb2* **by** *force*
also have ... = (⦃ *z1* : *b1* | *C-eq* (*CE-val* (*V-var z1*)) (*CE-val* ($v'[x::=v]_{vv}$)) ⦄)
using *type-e-eq*[*of z2 CE-val* ($v'[x::=v]_{vv}$)*z1 b1*] *zb1 zb2 fresh-PairD*(*1*) *assms beq* **by** *metis*
finally show *?thesis* **using** *zb1 zb2* **by** *argo*
qed

lemma *subst-t-b-eq*:
fixes *x*::*x* **and** *v*::*v*
shows *b-of* ($\tau[x::=v]_{\tau v}$) = *b-of* τ
proof −
obtain *z* **and** *b* **and** *c* **where** τ = ⦃ *z* : *b* | *c* ⦄ ∧ *atom z* ♯ (*x*,*v*)
using *has-fresh-z* **by** *blast*
thus *?thesis* **using** *subst-tv.simps* **by** *simp*
qed

lemma *fresh-g-fresh-v*:
fixes *x*::*x*
assumes *atom x* ♯ Γ **and** *wfV* Θ $\mathcal{B}$ Γ *v b*
shows *atom x* ♯ *v*
using *assms wfV-supp wfX-wfY wfG-atoms-supp-eq fresh-def*
by (*metis wfV-x-fresh*)

lemma *infer-v-fresh-g-fresh-v*:
fixes *x*::*x* **and** Γ::Γ **and** *v*::*v*
assumes *atom x* ♯ Γ'@Γ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$
shows *atom x* ♯ *v*
proof −
have *atom x* ♯ Γ **using** *fresh-suffix assms* **by** *auto*
moreover have *wfV* Θ $\mathcal{B}$ Γ *v* (*b-of* τ) **using** *infer-v-wf assms* **by** *auto*
ultimately show *?thesis* **using** *fresh-g-fresh-v* **by** *metis*
qed

lemma *infer-v-fresh-g-fresh-xv*:
fixes *xa*::*x* **and** *v*::*v* **and** Γ::Γ
assumes *atom xa* ♯ Γ'@((*x*,*b*,*c*[*z*::=[*x*]^v]$_{cv}$)#$_Γ$Γ) **and** Θ ; ℬ ; Γ ⊢ *v* ⇒ τ
shows *atom xa* ♯ (*x*,*v*)
proof −
have *atom xa* ♯ *x* **using** *assms fresh-in-g fresh-def* **by** *blast*
moreover have Γ'@((*x*,*b*,*c*[*z*::=[*x*]^v]$_{cv}$)#$_Γ$Γ) = ((Γ'@(*x*,*b*,*c*[*z*::=[*x*]^v]$_{cv}$)#$_Γ$ *GNil*)@Γ) **using** *append-g.simps append-g-assoc* **by** *simp*
moreover hence *atom xa* ♯ *v* **using** *infer-v-fresh-g-fresh-v assms* **by** *metis*
ultimately show *?thesis* **by** *auto*
qed

lemma *wfG-subst-infer-v*:
fixes *v*::*v*
assumes Θ ; ℬ ⊢$_{wf}$ Γ' @ (*x*, *b*, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_Γ$ Γ **and** Θ ; ℬ ; Γ ⊢ *v* ⇒ τ **and** *b-of* τ = *b*
shows Θ ; ℬ⊢$_{wf}$ Γ'[*x*::=*v*]$_{Γv}$ @ Γ
using *wfG-subst-wfV infer-v-v-wf assms* **by** *auto*

lemma *fresh-subst-gv-inside*:
fixes Γ::Γ
assumes *atom z* ♯ Γ' @ (*x*, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_Γ$ Γ **and** *atom z* ♯ *v*
shows *atom z* ♯ Γ'[*x*::=*v*]$_{Γv}$@Γ
unfolding *fresh-append-g* **using** *fresh-append-g assms fresh-subst-gv fresh-GCons* **by** *metis*

lemma *subst-t*:
fixes *x*::*x* **and** *b*::*b* **and** *xa*::*x*
assumes *atom z* ♯ *x* **and** *atom z* ♯ *v*
shows (⦃ *z* : *b* | [[*z*]^v]^{ce} == [*v*'[*x*::=*v*]$_{vv}$]^{ce} ⦄) = (⦃ *z* : *b* | [[*z*]^v]^{ce} == [*v*']^{ce} ⦄[*x*::=*v*]$_{τv}$)
using *assms subst-vv.simps subst-tv.simps subst-cv.simps subst-cev.simps* **by** *auto*

lemma *infer-l-fresh*:
assumes ⊢ *l* ⇒ τ
shows *atom x* ♯ τ
proof −
have [] ; {||} ⊢$_{wf}$ *GNil* **using** *wfG-nilI wfTh-emptyI* **by** *auto*
hence [] ; {||} ; *GNil* ⊢$_{wf}$ τ **using** *assms infer-l-wf* **by** *auto*
thus *?thesis* **using** *fresh-def wfT-supp* **by** *force*
qed

lemma *subst-infer-v*:
fixes *v*::*v* **and** *v*'::*v*
assumes Θ ; ℬ ; Γ'@((*x*,b_1,*c0*[*z0*::=[*x*]^v]$_{cv}$)#$_Γ$Γ) ⊢ *v*' ⇒ $τ_2$ **and**
Θ ; ℬ ; Γ ⊢ *v* ⇒ $τ_1$ **and**
Θ ; ℬ ; Γ ⊢ $τ_1$ ≲ (⦃ *z0* : b_1 | *c0* ⦄) **and** *atom z0* ♯ (*x*,*v*)
shows Θ ; ℬ ; (Γ'[*x*::=*v*]$_{Γv}$)@Γ ⊢ *v*'[*x*::=*v*]$_{vv}$ ⇒ $τ_2$[*x*::=*v*]$_{τv}$
using *assms* **proof**(*nominal-induct* Γ'@((*x*,b_1,*c0*[*z0*::=[*x*]^v]$_{cv}$)#$_Γ$Γ) *v*' $τ_2$ *avoiding*: *x v rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ ℬ *b c xa z*)
have Θ ; ℬ ; Γ'[*x*::=*v*]$_{Γv}$ @ Γ ⊢ [*xa*]^v[*x*::=*v*]$_{vv}$ ⇒ ⦃ *z* : *b* | [[*z*]^v]^{ce} == [[*xa*]^v[*x*::=*v*]$_{vv}$]^{ce} ⦄
proof(*cases x*=*xa*)
case *True*

have Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v* ⇒ ⦃ *z* : *b* | [[*z*]^{v}]^{ce} == [*v*]^{ce} ⦄
proof(*rule infer-v-g-weakening*)
show ∗:‹ Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ ⦃ *z* : *b* | [[*z*]^{v}]^{ce} == [*v*]^{ce} ⦄›
using *infer-v-form infer-v-varI*
by (*metis True lookup-inside-unique-b lookup-inside-wf ms-fresh-all*(*32*) *subtype-eq-base type-e-eq*)
show ‹*toSet* Γ ⊆ *toSet* (Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ)› **by** *simp*
have Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *v* : b_1 **using** *infer-v-wf subtype-eq-base2 b-of.simps infer-v-varI* **by** *metis*
thus ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ›
using *wfG-subst*[*OF infer-v-varI*(*3*), *of* Γ′ *x* b_1 *c0*[*z0*::=[*x*]^{v}]$_{cv}$ Γ *v*] *subst-g-inside infer-v-varI* **by** *metis*
qed

thus *?thesis* **using** *subst-vv.simps True* **by** *simp*
next
case *False*
then obtain *c′* **where** *c*: *Some* (*b*, *c′*) = *lookup* (Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ) *xa* **using** *lookup-subst2 infer-v-varI* **by** *metis*
have Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ [*xa*]^{v} ⇒ ⦃ *z* : *b* | [[*z*]^{v}]^{ce} == [[*xa*]^{v}]^{ce} ⦄
proof
have Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *v* : b_1 **using** *infer-v-wf subtype-eq-base2 b-of.simps infer-v-varI* **by** *metis*
thus Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ **using** *infer-v-varI*
using *wfG-subst*[*OF infer-v-varI*(*3*), *of* Γ′ *x* b_1 *c0*[*z0*::=[*x*]^{v}]$_{cv}$ Γ *v*] *subst-g-inside infer-v-varI* **by** *metis*
show *atom z* ♯ *xa* **using** *infer-v-varI* **by** *auto*
show *Some* (*b*, *c′*) = *lookup* (Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ) *xa* **using** *c* **by** *auto*
show *atom z* ♯ (Θ, $\mathcal{B}$, Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ) **by** (*fresh-mth add*: *infer-v-varI fresh-subst-gv-inside*)
qed
then show *?thesis* **using** *subst-vv.simps False* **by** *simp*
qed
thus *?case* **using** *subst-t fresh-prodN infer-v-varI* **by** *metis*
next
case (*infer-v-litI* Θ $\mathcal{B}$ *l* τ)
show *?case* **unfolding** *subst-vv.simps* **proof**
show Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ **using** *wfG-subst-infer-v infer-v-litI subtype-eq-base2 b-of.simps* **by** *metis*
have *atom x* ♯ τ **using** *infer-v-litI infer-l-fresh* **by** *metis*
thus ⊢ *l* ⇒ τ[*x*::=*v*]$_{\tau v}$ **using** *infer-v-litI type-v-subst-fresh* **by** *simp*
qed
next
case (*infer-v-pairI z v1 v2* Θ $\mathcal{B}$ *t1 t2*)
have Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @
Γ ⊢ [*v1*[*x*::=*v*]$_{vv}$, *v2*[*x*::=*v*]$_{vv}$]^{v} ⇒ ⦃ *z* : [*b-of t1*[*x*::=*v*]$_{\tau v}$, *b-of t2*[*x*::=*v*]$_{\tau v}$]^{b} | [[*z*]^{v}]^{ce} == [[*v1*[*x*::=*v*]$_{vv}$, *v2*[*x*::=*v*]$_{vv}$]^{v}]^{ce} ⦄
proof
show ‹*atom z* ♯ (*v1*[*x*::=*v*]$_{vv}$, *v2*[*x*::=*v*]$_{vv}$)› **by** (*fresh-mth add*: *infer-v-pairI*)
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ)› **by** (*fresh-mth add*: *infer-v-pairI fresh-subst-gv-inside*)
show ‹ Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v1*[*x*::=*v*]$_{vv}$ ⇒ *t1*[*x*::=*v*]$_{\tau v}$› **using** *infer-v-pairI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v2*[*x*::=*v*]$_{vv}$ ⇒ *t2*[*x*::=*v*]$_{\tau v}$› **using** *infer-v-pairI* **by** *metis*
qed
then show *?case* **using** *subst-vv.simps subst-tv.simps infer-v-pairI b-of-subst* **by** *simp*
next
case (*infer-v-consI s dclist* Θ *dc tc* $\mathcal{B}$ *va tv z*)

have Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash$ (*V-cons s dc va*$[x::=v]_{vv}$) $\Rightarrow$ ⦃ *z* : *B-id s* | [[*z* $]^v$ $]^{ce}$ == [*V-cons s dc va*$[x::=v]_{vv}$ $]^{ce}$ ⦄
proof
show *td*:‹*AF-typedef s dclist* $\in$ *set* Θ› **using** *infer-v-consI* **by** *auto*
show *dc*:‹(*dc*, *tc*) $\in$ *set dclist*› **using** *infer-v-consI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash$ *va*$[x::=v]_{vv} \Rightarrow$ *tv*$[x::=v]_{\tau v}$› **using** *infer-v-consI* **by** *auto*
have ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ *tv*$[x::=v]_{\tau v} \lesssim$ *tc*$[x::=v]_{\tau v}$›
using *subst-subtype-tau infer-v-consI* **by** *metis*
moreover have *atom x* ♯ *tc* **using** *wfTh-lookup-supp-empty*[*OF td dc*] *infer-v-wf infer-v-consI fresh-def* **by** *fast*
ultimately show ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ *tv*$[x::=v]_{\tau v} \lesssim$ *tc*› **by** *simp*
show ‹*atom z* ♯ *va*$[x::=v]_{vv}$› **using** *infer-v-consI* **by** *auto*
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ)› **by** (*fresh-mth add*: *infer-v-consI fresh-subst-gv-inside*)
qed
thus *?case* **using** *subst-vv.simps subst-t*[*of z x v*] *infer-v-consI* **by** *metis*

next
case (*infer-v-conspI s bv dclist* Θ *dc tc* $\mathcal{B}$ *va tv b z*)
have Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash$ (*V-consp s dc b va*$[x::=v]_{vv}$) $\Rightarrow$ ⦃ *z* : *B-app s b* | [[*z* $]^v$ $]^{ce}$ == [*V-consp s dc b va*$[x::=v]_{vv}$ $]^{ce}$ ⦄
proof
show *td*:‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ› **using** *infer-v-conspI* **by** *auto*
show *dc*:‹(*dc*, *tc*) $\in$ *set dclist*› **using** *infer-v-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash$ *va*$[x::=v]_{vv} \Rightarrow$ *tv*$[x::=v]_{\tau v}$› **using** *infer-v-conspI* **by** *metis*
have ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ *tv*$[x::=v]_{\tau v} \lesssim$ *tc*$[bv::=b]_{\tau b}[x::=v]_{\tau v}$›
using *subst-subtype-tau infer-v-conspI* **by** *metis*
moreover have *atom x* ♯ *tc*$[bv::=b]_{\tau b}$ **proof** −
have *supp tc* $\subseteq$ { *atom bv* } **using** *wfTh-poly-lookup-supp infer-v-conspI wfX-wfY* **by** *metis*
hence *atom x* ♯ *tc* **using** *x-not-in-b-set*
using *fresh-def* **by** *fastforce*
moreover have *atom x* ♯ *b* **using** *x-fresh-b* **by** *auto*
ultimately show *?thesis* **using** *fresh-subst-if subst-b-*τ*-def* **by** *metis*
qed
ultimately show ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ *tv*$[x::=v]_{\tau v} \lesssim$ *tc*$[bv::=b]_{\tau b}$› **by** *simp*
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ, *va*$[x::=v]_{vv}$, *b*)› **proof** −
have *atom z* ♯ *va*$[x::=v]_{vv}$ **using** *fresh-subst-v-if infer-v-conspI subst-v-v-def* **by** *metis*
moreover have *atom z* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *fresh-subst-gv-inside infer-v-conspI* **by** *metis*
ultimately show *?thesis* **using** *fresh-prodN infer-v-conspI* **by** *metis*
qed
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ, *va*$[x::=v]_{vv}$, *b*)› **proof** −
have *atom bv* ♯ *va*$[x::=v]_{vv}$ **using** *fresh-subst-v-if infer-v-conspI subst-v-v-def* **by** *metis*
moreover have *atom bv* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *fresh-subst-gv-inside infer-v-conspI* **by** *metis*
ultimately show *?thesis* **using** *fresh-prodN infer-v-conspI* **by** *metis*
qed
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *infer-v-conspI* **by** *auto*
qed
thus *?case* **using** *subst-vv.simps subst-t*[*of z x v*] *infer-v-conspI* **by** *metis*

qed

lemma *subst-infer-check-v*:

fixes *v*::*v* **and** *v'*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
check-v Θ $\mathcal{B}$ $(\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma))$ v' τ_2 **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim$ {| *z0* : b_1 | *c0* |} **and** *atom z0* $\sharp$ (x,v)
shows *check-v* Θ $\mathcal{B}$ $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ $(v'[x::=v]_{vv})$ $(\tau_2[x::=v]_{\tau v})$
proof –
obtain τ_2' **where** *t2*: *infer-v* Θ $\mathcal{B}$ $(\Gamma'$ @ $(x,$ $b_1,$ $c0[z0::=[x]^v]_{cv})$ $\#_\Gamma$ $\Gamma)$ v' $\tau_2' \wedge$ Θ ; $\mathcal{B}$; $(\Gamma'$ @ $(x,$ $b_1,$ $c0[z0::=[x]^v]_{cv})$ $\#_\Gamma$ $\Gamma)$ $\vdash \tau_2' \lesssim \tau_2$
using *check-v-elims assms* **by** *blast*
hence *infer-v* Θ $\mathcal{B}$ $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ $(v'[x::=v]_{vv})$ $(\tau_2'[x::=v]_{\tau v})$
using *subst-infer-v*[*OF - assms*(*1*) *assms*(*3*) *assms*(*4*)] **by** *blast*
moreover hence Θ; $\mathcal{B}$; $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ $\vdash \tau_2'[x::=v]_{\tau v} \lesssim \tau_2[x::=v]_{\tau v}$
using *subst-subtype assms t2* **by** (*meson subst-subtype-tau subtype-trans*)
ultimately show *?thesis* **using** *check-v.intros* **by** *blast*
qed

lemma *type-veq-subst*[*simp*]:
assumes *atom z* $\sharp$ (x,v)
shows {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val v'* |}$[x::=v]_{\tau v}$ = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val* $v'[x::=v]_{vv}$ |}
using *assms* **by** *auto*

lemma *subst-infer-v-form*:
fixes *v*::*v* **and** *v'*::*v* **and** Γ::Γ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
Θ ; $\mathcal{B}$; $\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash v' \Rightarrow \tau_2$ **and** *b*= *b-of* τ_2
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim$ ({| *z0* : b_1 | *c0* |}) **and** *atom z0* $\sharp$ (x,v) **and** *atom z3'* $\sharp$ $(x,v,v',\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma)$)
shows ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash v'[x::=v]_{vv} \Rightarrow$ {| *z3'* : *b* | *CE-val* (*V-var z3'*) == *CE-val* $v'[x::=v]_{vv}$ |}›
proof –
have Θ ; $\mathcal{B}$; $\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash v' \Rightarrow$ {| *z3'* : *b-of* τ_2 | *C-eq* (*CE-val* (*V-var z3'*)) (*CE-val v'*) |}
proof(*rule infer-v-form4*)
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,$ $b_1,$ $c0[z0::=[\ x\ ]^v]_{cv})$ $\#_\Gamma$ $\Gamma \vdash v' \Rightarrow \tau_2$› **using** *assms* **by** *metis*
show ‹*atom z3'* $\sharp$ $(v',$ Γ' @ $(x,$ $b_1,$ $c0[z0::=[\ x\ ]^v]_{cv})$ $\#_\Gamma$ $\Gamma)$› **using** *assms fresh-prodN* **by** *metis*
show ‹*b-of* τ_2 = *b-of* τ_2› **by** *auto*
qed
hence ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash v'[x::=v]_{vv} \Rightarrow$ {| *z3'* : *b-of* τ_2 | *CE-val* (*V-var z3'*) == *CE-val* v' |}$[x::=v]_{\tau v}$›
using *subst-infer-v assms* **by** *metis*
thus *?thesis* **using** *type-veq-subst fresh-prodN assms* **by** *metis*
qed

14.7 Expressions

For operator, fst and snd cases, we use elimination to get one or more values, apply the substitution lemma for values. The types always have the same form and are equal under substitution. For function application, the subst value is a subtype of the value which is a subtype of the argument. The return of the function is the same under substitution.

Observe a similar pattern for each case

lemma *subst-infer-e*:
 fixes $v::v$ **and** $e::e$ **and** $\Gamma'::\Gamma$
 assumes
  $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash e \Rightarrow \tau_2$ **and** $G = (\Gamma'@((x,b_1,subst\text{-}cv\ c0\ z0\ (V\text{-}var\ x))\#_\Gamma\Gamma))$
  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **and**
  $\Theta; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \{\!\!\{ z0 : b_1 \mid c0 \}\!\!\}$ **and** $atom\ z0 \sharp (x,v)$
 shows $\Theta ; \Phi ; \mathcal{B} ; ((\Gamma'[x::=v]_{\Gamma v})@\Gamma) ; (\Delta[x::=v]_{\Delta v}) \vdash (subst\text{-}ev\ e\ x\ v\ ) \Rightarrow \tau_2[x::=v]_{\tau v}$
 using *assms* **proof**(*nominal-induct avoiding*: *x v rule*: *infer-e.strong-induct*)
 case (*infer-e-valI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi\ v'\ \tau$)

 have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma ; \Delta[x::=v]_{\Delta v} \vdash (AE\text{-}val\ (v'[x::=v]_{vv})) \Rightarrow \tau[x::=v]_{\tau v}$
 proof
  show $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ **using** *wfD-subst infer-e-valI subtype-eq-base2*
   by (*metis b-of.simps infer-v-v-wf subst-g-inside-simple wfD-wf wf-subst*(*11*))
  show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-valI* **by** *auto*
  show $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau[x::=v]_{\tau v}$ **using** *subst-infer-v infer-e-valI* **using** *wfD-subst infer-e-valI subtype-eq-base2*
   by *metis*
 qed
 thus *?case* **using** *subst-ev.simps* **by** *simp*
next
 case (*infer-e-plusI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi$ *v1 z1 c1 v2 z2 c2 z3*)

 hence *z3f*: *atom z3* $\sharp$ *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

 obtain *z3'*::*x* **where** $*$:*atom z3'* $\sharp$ (*x*,*v*,*AE-op Plus v1 v2*, *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$, *AE-op Plus* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$, *CE-op Plus* $[v1[x::=v]_{vv}]^{ce}$ $[v2[x::=v]_{vv}]^{ce}$,$\Gamma'[x::=v]_{\Gamma v} @ \Gamma$)
  using *obtain-fresh* **by** *metis*
 hence $**$:($\{\!\!\{$ *z3* : *B-int* $\mid$ *CE-val* (*V-var z3*) == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ $\}\!\!\}$) = $\{\!\!\{$ *z3'* : *B-int* $\mid$ *CE-val* (*V-var z3'*) == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ $\}\!\!\}$
  using *type-e-eq infer-e-plusI fresh-Pair z3f* **by** *metis*

 obtain *z1' b1' c1'* **where** *z1*:*atom z1'* $\sharp$ (*x*,*v*) $\wedge$ $\{\!\!\{$ *z1* : *B-int* $\mid$ *c1* $\}\!\!\}$ = $\{\!\!\{$ *z1'* : *b1'* $\mid$ *c1'* $\}\!\!\}$ **using** *obtain-fresh-z* **by** *metis*
 obtain *z2' b2' c2'* **where** *z2*:*atom z2'* $\sharp$ (*x*,*v*) $\wedge$ $\{\!\!\{$ *z2* : *B-int* $\mid$ *c2* $\}\!\!\}$ = $\{\!\!\{$ *z2'* : *b2'* $\mid$ *c2'* $\}\!\!\}$ **using** *obtain-fresh-z* **by** *metis*

 have *bb*:*b1'* = *B-int* $\wedge$ *b2'* = *B-int* **using** *z1 z2* τ*.eq-iff* **by** *metis*

 have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma ; \Delta[x::=v]_{\Delta v} \vdash$ (*AE-op Plus* ($v1[x::=v]_{vv}$) ($v2[x::=v]_{vv}$)) $\Rightarrow$ $\{\!\!\{$ *z3'* : *B-int* $\mid$ *CE-val* (*V-var z3'*) == *CE-op Plus* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) $\}\!\!\}$
 proof
  show ‹ $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ ›
   using *infer-e-plusI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*
  show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-plusI* **by** *blast*
  show ‹ $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow$ $\{\!\!\{$ *z1'* : *B-int* $\mid$ $c1'[x::=v]_{cv}$ $\}\!\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-plusI z1 bb* **by** *metis*
  show ‹ $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v2[x::=v]_{vv} \Rightarrow$ $\{\!\!\{$ *z2'* : *B-int* $\mid$ $c2'[x::=v]_{cv}$ $\}\!\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-plusI z2 bb* **by** *metis*
  show ‹*atom z3'* $\sharp$ *AE-op Plus* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$› **using** *fresh-prod6* $*$ **by** *metis*
  show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v} @ \Gamma$› **using** $*$ **by** *auto*

qed
moreover have ⦃ *z3′* : *B-int* | *CE-val* (*V-var z3′*) == *CE-op Plus* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) ⦄ = ⦃ *z3′* : *B-int* | *CE-val* (*V-var z3′*) == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ ⦄$[x::=v]_{\tau v}$
by(*subst subst-tv.simps*,*auto simp add*: ∗)
ultimately show *?case* **using** *subst-ev.simps* ∗ ∗∗ **by** *metis*
next
case (*infer-e-leqI* Θ ℬ Γ″ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

hence *z3f*: *atom z3* ♯ *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

obtain *z3′*::*x* **where** ∗:*atom z3′* ♯ (*x*,*v*,*AE-op LEq v1 v2*, *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$, *CE-op LEq* $[v1[x::=v]_{vv}]^{ce}$ $[v2[x::=v]_{vv}]^{ce}$, *AE-op LEq* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ)
using *obtain-fresh* **by** *metis*
hence ∗∗:(⦃ *z3* : *B-bool* | *CE-val* (*V-var z3*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄) = ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄
using *type-e-eq infer-e-leqI fresh-Pair z3f* **by** *metis*

obtain *z1′ b1′ c1′* **where** *z1*:*atom z1′* ♯ (*x*,*v*) ∧ ⦃ *z1* : *B-int* | *c1* ⦄ = ⦃ *z1′* : *b1′* | *c1′* ⦄ **using** *obtain-fresh-z* **by** *metis*
obtain *z2′ b2′ c2′* **where** *z2*:*atom z2′* ♯ (*x*,*v*) ∧ ⦃ *z2* : *B-int* | *c2* ⦄ = ⦃ *z2′* : *b2′* | *c2′* ⦄ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b1′* = *B-int* ∧ *b2′* = *B-int* **using** *z1 z2 τ.eq-iff* **by** *metis*

have Θ ; Φ ; ℬ ; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ ⊢ (*AE-op LEq* ($v1[x::=v]_{vv}$) ($v2[x::=v]_{vv}$)) ⇒ ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op LEq* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) ⦄
proof
show ‹ Θ ; ℬ ; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-leqI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ$\vdash_{wf}$ Φ › **using** *infer-e-leqI*(*2*) **by** *auto*
show ‹ Θ ; ℬ ; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ⊢ $v1[x::=v]_{vv}$ ⇒ ⦃ *z1′* : *B-int* | $c1'[x::=v]_{cv}$ ⦄› **using** *subst-tv.simps subst-infer-v infer-e-leqI z1 bb* **by** *metis*
show ‹ Θ ; ℬ ; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ⊢ $v2[x::=v]_{vv}$ ⇒ ⦃ *z2′* : *B-int* | $c2'[x::=v]_{cv}$ ⦄› **using** *subst-tv.simps subst-infer-v infer-e-leqI z2 bb* **by** *metis*
show ‹*atom z3′* ♯ *AE-op LEq* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$› **using** *fresh-Pair* ∗ **by** *metis*
show ‹*atom z3′* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** ∗ **by** *auto*
qed
moreover have ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op LEq* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) ⦄ = ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄$[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* ∗ **by** *auto*
ultimately show *?case* **using** *subst-ev.simps* ∗ ∗∗ **by** *metis*
next
case (*infer-e-eqI* Θ ℬ Γ″ Δ Φ *v1 z1 bb c1 v2 z2 c2 z3*)

hence *z3f*: *atom z3* ♯ *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

obtain *z3′*::*x* **where** ∗:*atom z3′* ♯ (*x*,*v*,*AE-op Eq v1 v2*, *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$, *CE-op Eq* $[v1[x::=v]_{vv}]^{ce}$ $[v2[x::=v]_{vv}]^{ce}$, *AE-op Eq* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ)
using *obtain-fresh* **by** *metis*
hence ∗∗:(⦃ *z3* : *B-bool* | *CE-val* (*V-var z3*) == *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄) = ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$ ⦄
using *type-e-eq infer-e-eqI fresh-Pair z3f* **by** *metis*

obtain *z1′ b1′ c1′* **where** *z1*:*atom z1′* ♯ (*x*,*v*) ∧ ⦃ *z1* : *bb* | *c1* ⦄ = ⦃ *z1′* : *b1′* | *c1′* ⦄ **using** *obtain-fresh-z* **by** *metis*
obtain *z2′ b2′ c2′* **where** *z2*:*atom z2′* ♯ (*x*,*v*) ∧ ⦃ *z2* : *bb* | *c2* ⦄ = ⦃ *z2′* : *b2′* | *c2′* ⦄ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b1′* = *bb* ∧ *b2′* = *bb* **using** *z1 z2* τ*.eq-iff* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ; Δ[*x*::=*v*]$_{\Delta v}$ ⊢ (*AE-op Eq* (*v1*[*x*::=*v*]$_{vv}$) (*v2*[*x*::=*v*]$_{vv}$)) ⇒ ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op Eq* ([*v1*[*x*::=*v*]$_{vv}$]^{ce}) ([*v2*[*x*::=*v*]$_{vv}$]^{ce}) ⦄
proof
show ‹ Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢$_{wf}$ Δ[*x*::=*v*]$_{\Delta v}$ › **using** *wfD-subst infer-e-eqI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ⊢$_{wf}$ Φ › **using** *infer-e-eqI*(*2*) **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v1*[*x*::=*v*]$_{vv}$ ⇒ ⦃ *z1′* : *bb* | *c1′*[*x*::=*v*]$_{cv}$ ⦄› **using** *subst-tv.simps subst-infer-v infer-e-eqI z1 bb* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v2*[*x*::=*v*]$_{vv}$ ⇒ ⦃ *z2′* : *bb* | *c2′*[*x*::=*v*]$_{cv}$ ⦄› **using** *subst-tv.simps subst-infer-v infer-e-eqI z2 bb* **by** *metis*
show ‹*atom z3′* ♯ *AE-op Eq v1*[*x*::=*v*]$_{vv}$ *v2*[*x*::=*v*]$_{vv}$› **using** *fresh-Pair* ∗ **by** *metis*
show ‹*atom z3′* ♯ Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ› **using** ∗ **by** *auto*
show *bb* ∈ {*B-bool*, *B-int*, *B-unit*} **using** *infer-e-eqI* **by** *auto*
qed
moreover have ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op Eq* ([*v1*[*x*::=*v*]$_{vv}$]^{ce}) ([*v2*[*x*::=*v*]$_{vv}$]^{ce}) ⦄ = ⦃ *z3′* : *B-bool* | *CE-val* (*V-var z3′*) == *CE-op Eq* [*v1*]^{ce} [*v2*]^{ce} ⦄[*x*::=*v*]$_{\tau v}$
using *subst-tv.simps subst-ev.simps* ∗ **by** *auto*
ultimately show *?case* **using** *subst-ev.simps* ∗ ∗∗ **by** *metis*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ′′ Δ Φ *f x′ b c* τ′ *s′ v′* τ)

hence *x* ≠ *x′* **using** ‹*atom x′* ♯ Γ′′› **using** *wfG-inside-x-neq wfX-wfY* **by** *metis*

show *?case* **proof**(*subst subst-ev.simps*,*rule*)
show ‹ Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢$_{wf}$ Δ[*x*::=*v*]$_{\Delta v}$ › **using** *infer-e-appI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ⊢$_{wf}$ Φ › **using** *infer-e-appI* **by** *metis*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x′ b c* τ′ *s′*))) = *lookup-fun* Φ *f*› **using** *infer-e-appI* **by** *metis*

have ‹Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v′*[*x*::=*v*]$_{vv}$ ⇐ ⦃ *x′* : *b* | *c* ⦄[*x*::=*v*]$_{\tau v}$› **proof**(*rule subst-infer-check-v*)
show Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ τ_1 **using** *infer-e-appI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ′ @ (*x*, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ ⊢ *v′* ⇐ ⦃ *x′* : *b* | *c* ⦄ **using** *infer-e-appI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ ⊢ τ_1 ≲ ⦃ *z0* : b_1 | *c0* ⦄ **using** *infer-e-appI* **by** *metis*
show *atom z0* ♯ (*x*, *v*) **using** *infer-e-appI* **by** *metis*
qed
moreover have *atom x* ♯ *c* **using** *wfPhi-f-simple-supp-c infer-e-appI fresh-def* ‹*x*≠*x′*›
atom-eq-iff empty-iff infer-e-appI.hyps insert-iff subset-singletonD **by** *metis*

moreover hence *atom x* ♯ ⦃ *x′* : *b* | *c* ⦄ **using** τ*.fresh supp-b-empty fresh-def* **by** *blast*
ultimately show ‹Θ ; $\mathcal{B}$; Γ′[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *v′*[*x*::=*v*]$_{vv}$ ⇐ ⦃ *x′* : *b* | *c* ⦄› **using** *forget-subst-tv* **by** *metis*

have ∗: *atom* x' ♯ (x,v) **using** *infer-v-fresh-g-fresh-xv infer-e-appI check-v-wf* **by** *blast*

show ‹*atom* x' ♯ $(\Theta, \Phi, \mathcal{B}, \Gamma'[x::=v]_{\Gamma v} @ \Gamma, \Delta[x::=v]_{\Delta v}, v'[x::=v]_{vv}, \tau[x::=v]_{\tau v})$›
apply(*unfold fresh-prodN*, *intro conjI*)
apply (*fresh-subst-mth-aux add*: *infer-e-appI fresh-subst-gv wfD-wf subst-g-inside*)
using *infer-e-appI fresh-subst-gv wfD-wf subst-g-inside* **apply** *metis*
using *infer-e-appI fresh-subst-dv-if* **apply** *metis*
done

have *supp* $\tau' \subseteq \{$ *atom* x' $\} \cup$ *supp* $\mathcal{B}$ **using** *infer-e-appI wfT-supp wfPhi-f-simple-wfT*
by (*meson infer-e-appI.hyps*(*2*) *le-supI1 wfPhi-f-simple-supp-t*)
hence *atom* x ♯ τ' **using** ‹$x \neq x'$› *fresh-def supp-at-base x-not-in-b-set* **by** *fastforce*
thus ‹$\tau'[x'::=v'[x::=v]_{vv}]_v = \tau[x::=v]_{\tau v}$› **using** *subst-tv-commute infer-e-appI subst-defs* **by** *metis*
qed
next
case (*infer-e-appPI* Θ $\mathcal{B}$ Γ'' Δ Φ b' *f bv* x' *b c* τ' s' v' τ)

hence $x \neq x'$ **using** ‹*atom* x' ♯ Γ''› **using** *wfG-inside-x-neq wfX-wfY* **by** *metis*

show *?case* **proof**(*subst subst-ev.simps*,*rule*)
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ › **using** *infer-e-appPI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-appPI*(*4*) **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} b'$ **using** *infer-e-appPI*(*5*) **by** *auto*
show *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ* x' *b c* τ' s'))) = *lookup-fun* Φ *f* **using** *infer-e-appPI*(*6*) **by** *auto*

show Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash v'[x::=v]_{vv} \Leftarrow$ ⦃ x' : $b[bv::=b']_b$ | $c[bv::=b']_b$ ⦄ **proof** −
have ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash v'[x::=v]_{vv} \Leftarrow$ ⦃ x' : $b[bv::=b']_{bb}$ | $c[bv::=b']_{cb}$ ⦄$[x::=v]_{\tau v}$› **proof**(*rule subst-infer-check-v*)
show Θ ; $\mathcal{B}$; Γ $\vdash v \Rightarrow \tau_1$ **using** *infer-e-appPI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ' @ $(x, b_1, c0[z0::=[x]^v]_{cv})$ $\#_\Gamma$ Γ $\vdash v' \Leftarrow$ ⦃ x' : $b[bv::=b']_{bb}$ | $c[bv::=b']_{cb}$ ⦄ **using** *infer-e-appPI subst-defs* **by** *metis*
show Θ ; $\mathcal{B}$; Γ $\vdash \tau_1 \lesssim$ ⦃ *z0* : b_1 | *c0* ⦄ **using** *infer-e-appPI* **by** *metis*
show *atom z0* ♯ (x, v) **using** *infer-e-appPI* **by** *metis*
qed
moreover have *atom* x ♯ c **proof** −
have *supp c* $\subseteq$ {*atom* x', *atom bv*} **using** *wfPhi-f-poly-supp-c*[*OF infer-e-appPI*(*6*)] *infer-e-appPI* **by** *metis*
thus *?thesis* **unfolding** *fresh-def* **using** ‹$x \neq x'$› *atom-eq-iff* **by** *auto*
qed
moreover hence *atom* x ♯ ⦃ x' : $b[bv::=b']_{bb}$ | $c[bv::=b']_{cb}$ ⦄ **using** *τ.fresh supp-b-empty fresh-def subst-b-fresh-x*
by (*metis subst-b-c-def*)
ultimately show *?thesis* **using** *forget-subst-tv subst-defs* **by** *metis*
qed
have *supp* $\tau' \subseteq \{$ *atom* x', *atom bv* $\}$ **using** *wfPhi-f-poly-supp-t infer-e-appPI* **by** *metis*
hence *atom* x ♯ τ' **using** *fresh-def* ‹$x \neq x'$› **by** *force*
hence ∗:*atom* x ♯ $\tau'[bv::=b']_{\tau b}$ **using** *subst-b-fresh-x subst-b-τ-def* **by** *metis*
have *atom* x' ♯ (x,v) **using** *infer-v-fresh-g-fresh-xv infer-e-appPI check-v-wf* **by** *blast*
thus *atom* x' ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *infer-e-appPI fresh-subst-gv wfD-wf subst-g-inside fresh-Pair*

by *metis*
show $\tau'[bv::=b']_b[x'::=v'[x::=v]_{vv}]_v = \tau[x::=v]_{\tau v}$ **using** *infer-e-appPI subst-tv-commute*[*OF* ∗] *subst-defs* **by** *metis*
show *atom bv* ♯ (Θ, Φ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ, $\Delta[x::=v]_{\Delta v}$, b', $v'[x::=v]_{vv}$, $\tau[x::=v]_{\tau v}$)
by (*fresh-mth add*: *infer-e-appPI fresh-subst-gv-inside*)
qed
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ'' Δ Φ *v' z' b1 b2 c z*)

hence *zf*: *atom z* ♯ *CE-fst* $[v']^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain *z3'*::*x* **where** ∗:*atom z3'* ♯ (*x*,*v*,*AE-fst v'*, *CE-fst* $[v']^{ce}$, *AE-fst* $v'[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ) **using** *obtain-fresh* **by** *auto*
hence ∗∗:(⦃ *z* : *b1* | *CE-val* (*V-var z*) == *CE-fst* $[v']^{ce}$ ⦄) = ⦃ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v']^{ce}$ ⦄
using *type-e-eq infer-e-fstI*(*4*) *fresh-Pair zf* **by** *metis*

obtain *z1' b1' c1'* **where** *z1*:*atom z1'* ♯ (*x*,*v*) ∧ ⦃ *z'* : *B-pair b1 b2* | *c* ⦄ = ⦃ *z1'* : *b1'* | *c1'* ⦄ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b1'* = *B-pair b1 b2* **using** *z1 τ.eq-iff* **by** *metis*
have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ ⊢ (*AE-fst* $v'[x::=v]_{vv}$) ⇒ ⦃ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v'[x::=v]_{vv}]^{ce}$ ⦄
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-fstI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ$\vdash_{wf}$ Φ › **using** *infer-e-fstI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ⊢ $v'[x::=v]_{vv}$ ⇒ ⦃ *z1'* : *B-pair b1 b2* | $c1'[x::=v]_{cv}$ ⦄› **using** *subst-tv.simps subst-infer-v infer-e-fstI z1 bb* **by** *metis*

show ‹*atom z3'* ♯ *AE-fst* $v'[x::=v]_{vv}$ › **using** *fresh-Pair* ∗ **by** *metis*
show ‹*atom z3'* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** ∗ **by** *auto*
qed
moreover have ⦃ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v'[x::=v]_{vv}]^{ce}$ ⦄ = ⦃ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v']^{ce}$ ⦄$[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* ∗ **by** *auto*
ultimately show *?case* **using** *subst-ev.simps* ∗ ∗∗ **by** *metis*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ'' Δ Φ *v' z' b1 b2 c z*)
hence *zf*: *atom z* ♯ *CE-snd* $[v']^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain *z3'*::*x* **where** ∗:*atom z3'* ♯ (*x*,*v*,*AE-snd v'*, *CE-snd* $[v']^{ce}$, *AE-snd* $v'[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ ,*v'*, Γ'') **using** *obtain-fresh* **by** *auto*
hence ∗∗:(⦃ *z* : *b2* | *CE-val* (*V-var z*) == *CE-snd* $[v']^{ce}$ ⦄) = ⦃ *z3'* : *b2* | *CE-val* (*V-var z3'*) == *CE-snd* $[v']^{ce}$ ⦄
using *type-e-eq infer-e-sndI*(*4*) *fresh-Pair zf* **by** *metis*

obtain *z1' b2' c1'* **where** *z1*:*atom z1'* ♯ (*x*,*v*) ∧ ⦃ *z'* : *B-pair b1 b2* | *c* ⦄ = ⦃ *z1'* : *b2'* | *c1'* ⦄ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b2'* = *B-pair b1 b2* **using** *z1 τ.eq-iff* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ; Δ[x::=v]$_{Δv}$ ⊢ (*AE-snd* (v′[x::=v]$_{vv}$)) ⇒ ⦃ *z3′* : *b2* | *CE-val* (*V-var z3′*) == *CE-snd* ([v′[x::=v]$_{vv}$]^{ce}) ⦄
proof
show ‹ Θ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ⊢$_{wf}$ Δ[x::=v]$_{Δv}$ › **using** *wfD-subst infer-e-sndI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ⊢$_{wf}$ Φ › **using** *infer-e-sndI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ⊢ v′[x::=v]$_{vv}$ ⇒ ⦃ *z1′* : *B-pair b1 b2* | *c1′*[x::=v]$_{cv}$ ⦄› **using** *subst-tv.simps subst-infer-v infer-e-sndI z1 bb* **by** *metis*

show ‹*atom z3′* ♯ *AE-snd* v′[x::=v]$_{vv}$ › **using** *fresh-Pair* ∗ **by** *metis*
show ‹*atom z3′* ♯ Γ′[x::=v]$_{Γv}$ @ Γ› **using** ∗ **by** *auto*
qed
moreover have ⦃ *z3′* : *b2* | *CE-val* (*V-var z3′*) == *CE-snd* ([v′[x::=v]$_{vv}$]^{ce}) ⦄ = ⦃ *z3′* : *b2* | *CE-val* (*V-var z3′*) == *CE-snd* [v′]^{ce} ⦄[x::=v]$_{τv}$
by(*subst subst-tv.simps*, *auto simp add*: *fresh-prodN* ∗)
ultimately show *?case* **using** *subst-ev.simps* ∗ ∗∗ **by** *metis*
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ′′ Δ Φ v′ z′ c z)
hence *zf*: *atom z* ♯ *CE-len* [v′]^{ce} **using** *ce.fresh e.fresh opp.fresh* **by** *metis*
obtain *z3′*::x **where** ∗:*atom z3′* ♯ (x,v,*AE-len* v′, *CE-len* [v′]^{ce} , *AE-len* v′[x::=v]$_{vv}$,Γ′[x::=v]$_{Γv}$ @ Γ , Γ′′,v′) **using** *obtain-fresh* **by** *auto*
hence ∗∗:(⦃ z : *B-int* | *CE-val* (*V-var* z) == *CE-len* [v′]^{ce} ⦄) = ⦃ *z3′* : *B-int* | *CE-val* (*V-var z3′*) == *CE-len* [v′]^{ce} ⦄
using *type-e-eq infer-e-lenI fresh-Pair zf* **by** *metis*

have ∗∗∗: Θ ; $\mathcal{B}$; Γ′′ ⊢ v′ ⇒ ⦃ *z3′* : *B-bitvec* | *CE-val* (*V-var z3′*) == *CE-val* v′ ⦄
using *infer-e-lenI infer-v-form3*[*OF infer-e-lenI*(*3*), *of z3′*] *b-of.simps* ∗ *fresh-Pair* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ; Δ[x::=v]$_{Δv}$ ⊢ (*AE-len* (v′[x::=v]$_{vv}$)) ⇒ ⦃ *z3′* : *B-int* | *CE-val* (*V-var z3′*) == *CE-len* ([v′[x::=v]$_{vv}$]^{ce}) ⦄
proof
show ‹ Θ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ⊢$_{wf}$ Δ[x::=v]$_{Δv}$ › **using** *wfD-subst infer-e-lenI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ⊢$_{wf}$ Φ › **using** *infer-e-lenI* **by** *metis*

have ‹ Θ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ⊢ v′[x::=v]$_{vv}$ ⇒ ⦃ *z3′* : *B-bitvec* | *CE-val* (*V-var z3′*) == *CE-val* v′ ⦄[x::=v]$_{τv}$›
proof(*rule subst-infer-v*)
show ‹ Θ ; $\mathcal{B}$; Γ ⊢ v ⇒ $τ_1$› **using** *infer-e-lenI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ′ @ (x, b_1, *c0*[*z0*::=[x]^v]$_{cv}$) #$_Γ$ Γ ⊢ v′ ⇒ ⦃ *z3′* : *B-bitvec* | [[*z3′*]^v]^{ce} == [v′]^{ce} ⦄› **using** ∗∗∗ *infer-e-lenI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ ⊢ $τ_1$ ≲ ⦃ *z0* : b_1 | *c0* ⦄ **using** *infer-e-lenI* **by** *metis*
show *atom z0* ♯ (x, v) **using** *infer-e-lenI* **by** *metis*
qed

thus ‹ Θ ; $\mathcal{B}$; Γ′[x::=v]$_{Γv}$ @ Γ ⊢ v′[x::=v]$_{vv}$ ⇒ ⦃ *z3′* : *B-bitvec* | *CE-val* (*V-var z3′*) == *CE-val* v′[x::=v]$_{vv}$ ⦄›
using *subst-tv.simps subst-ev.simps fresh-Pair* ∗ *fresh-prodN subst-vv.simps* **by** *auto*

show ‹*atom z3′* ♯ *AE-len* v′[x::=v]$_{vv}$› **using** *fresh-Pair* ∗ **by** *metis*
show ‹*atom z3′* ♯ Γ′[x::=v]$_{Γv}$ @ Γ› **using** *fresh-Pair* ∗ **by** *metis*
qed

moreover have $\{\!\!\{ z3' : B\text{-}int \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}len\ ([v'[x::=v]_{vv}]^{ce}) \}\!\!\} = \{\!\!\{ z3' : B\text{-}int \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}len\ [v']^{ce} \}\!\!\}[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* ∗ **by** *auto*

ultimately show *?case* **using** *subst-ev.simps* ∗ ∗∗ **by** *metis*
next
case $(infer\text{-}e\text{-}mvarI\ \Theta\ \mathcal{B}\ \Gamma''\ \Phi\ \Delta\ u\ \tau)$

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma ; \Delta[x::=v]_{\Delta v} \vdash (AE\text{-}mvar\ u) \Rightarrow \tau[x::=v]_{\tau v}$
proof
show ‹ $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v} @ \Gamma$› **proof** −
have *wfV* $\Theta\ \mathcal{B}\ \Gamma\ v\ (b\text{-}of\ \tau_1)$ **using** *infer-v-wf infer-e-mvarI* **by** *auto*
moreover have $b\text{-}of\ \tau_1 = b_1$ **using** *subtype-eq-base2 infer-e-mvarI b-of.simps* **by** *simp*
ultimately show *?thesis* **using** *wf-subst(3)[OF infer-e-mvarI(1), of* $\Gamma'\ x\ b_1\ c0[z0::=[x]^v]_{cv}\ \Gamma\ v$*] infer-e-mvarI subst-g-inside* **by** *metis*
qed
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-mvarI* **by** *auto*
show ‹ $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-mvarI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹$(u,\ \tau[x::=v]_{\tau v}) \in setD\ \Delta[x::=v]_{\Delta v}$› **using** *infer-e-mvarI subst-dv-member* **by** *metis*
qed
moreover have $(AE\text{-}mvar\ u) = (AE\text{-}mvar\ u)[x::=v]_{ev}$ **using** *subst-ev.simps* **by** *auto*
ultimately show *?case* **by** *auto*

next
case $(infer\text{-}e\text{-}concatI\ \Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ c2\ z3)$

hence *zf*: *atom z3* ♯ $CE\text{-}concat\ [v1]^{ce}\ [v2]^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain *z3'::x* **where** ∗:*atom z3'* ♯ $(x,v,v1,v2,AE\text{-}concat\ v1\ v2,\ CE\text{-}concat\ [v1]^{ce}\ [v2]^{ce}\ ,\ AE\text{-}concat\ (v1[x::=v]_{vv})\ (v2[x::=v]_{vv})\ ,\Gamma'[x::=v]_{\Gamma v} @ \Gamma\ ,\ \Gamma'',v1\ ,\ v2)$ **using** *obtain-fresh* **by** *auto*

hence ∗∗:$(\{\!\!\{ z3 : B\text{-}bitvec \mid CE\text{-}val\ (V\text{-}var\ z3) == CE\text{-}concat\ [v1]^{ce}\ [v2]^{ce} \}\!\!\}) = \{\!\!\{ z3' : B\text{-}bitvec \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}concat\ [v1]^{ce}\ [v2]^{ce} \}\!\!\}$
using *type-e-eq infer-e-concatI fresh-Pair zf* **by** *metis*
have *zfx*: *atom x* ♯ *z3'* **using** *fresh-at-base fresh-prodN* ∗ **by** *auto*

have *v1*: $\Theta ; \mathcal{B} ; \Gamma'' \vdash v1 \Rightarrow \{\!\!\{ z3' : B\text{-}bitvec \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}val\ v1 \}\!\!\}$
using *infer-e-concatI infer-v-form3 b-of.simps* ∗ *fresh-Pair* **by** *metis*
have *v2*: $\Theta ; \mathcal{B} ; \Gamma'' \vdash v2 \Rightarrow \{\!\!\{ z3' : B\text{-}bitvec \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}val\ v2 \}\!\!\}$
using *infer-e-concatI infer-v-form3 b-of.simps* ∗ *fresh-Pair* **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma ; \Delta[x::=v]_{\Delta v} \vdash (AE\text{-}concat\ (v1[x::=v]_{vv})\ (v2[x::=v]_{vv})) \Rightarrow \{\!\!\{ z3' : B\text{-}bitvec \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}concat\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce}) \}\!\!\}$
proof
show ‹ $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-concatI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta \vdash_{wf} \Phi$ › **by**(*simp add*: *infer-e-concatI*)
show ‹ $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow \{\!\!\{ z3' : B\text{-}bitvec \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}val\ (v1[x::=v]_{vv}) \}\!\!\}$›
using *subst-infer-v-form infer-e-concatI fresh-prodN* ∗ *b-of.simps* **by** *metis*

show ‹ Θ ; ℬ ; Γ′[x::=v]$_{Γv}$ @ Γ ⊢ *v2*[x::=v]$_{vv}$ ⇒ ⦃ *z3′* : *B-bitvec* | *CE-val* (*V-var z3′*) == *CE-val* (*v2*[x::=v]$_{vv}$) ⦄›
using *subst-infer-v-form infer-e-concatI fresh-prodN* ∗ *b-of.simps* **by** *metis*
show ‹*atom z3′* ♯ *AE-concat v1*[x::=v]$_{vv}$ *v2*[x::=v]$_{vv}$› **using** *fresh-Pair* ∗ **by** *metis*
show ‹*atom z3′* ♯ Γ′[x::=v]$_{Γv}$ @ Γ› **using** *fresh-Pair* ∗ **by** *metis*
qed

moreover have ⦃ *z3′* : *B-bitvec* | *CE-val* (*V-var z3′*) == *CE-concat* ([*v1*[x::=v]$_{vv}$]^{ce}) ([*v2*[x::=v]$_{vv}$]^{ce}) ⦄ = ⦃ *z3′* : *B-bitvec* | *CE-val* (*V-var z3′*) == *CE-concat* [*v1*]^{ce} [*v2*]^{ce} ⦄[x::=v]$_{τv}$
using *subst-tv.simps subst-ev.simps* ∗ **by** *auto*

ultimately show *?case* **using** *subst-ev.simps* ∗∗ ∗ **by** *metis*
next
case (*infer-e-splitI* Θ ℬ Γ′′ Δ Φ *v1 z1 c1 v2 z2 z3*)
hence ∗:*atom z3* ♯ (*x*,*v*) **using** *fresh-Pair* **by** *auto*
have ‹*x* ≠ *z3* › **using** *infer-e-splitI* **by** *force*
have Θ ; Φ ; ℬ ; (Γ′[x::=v]$_{Γv}$ @ Γ) ; Δ[x::=v]$_{Δv}$ ⊢ (*AE-split v1*[x::=v]$_{vv}$ *v2*[x::=v]$_{vv}$) ⇒
⦃ *z3* : [*B-bitvec* , *B-bitvec*]^{b} | [*v1*[x::=v]$_{vv}$]^{ce} == [[#*1*[[*z3*]^{v}]^{ce}]^{ce} @@ [#*2*[[*z3*]^{v}]^{ce}]^{ce}]^{ce} *AND*
[| [#*1*[[*z3*]^{v}]^{ce}]^{ce} |]^{ce} == [*v2*[x::=v]$_{vv}$]^{ce} ⦄
proof
show ‹ Θ ; ℬ ; Γ′[x::=v]$_{Γv}$ @ Γ ⊢$_{wf}$ Δ[x::=v]$_{Δv}$ › **using** *wfD-subst infer-e-splitI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ Θ ⊢$_{wf}$ Φ › **using** *infer-e-splitI* **by** *auto*
have ‹ Θ ; ℬ ; Γ′[x::=v]$_{Γv}$ @ Γ ⊢ *v1*[x::=v]$_{vv}$ ⇒ ⦃ *z1* : *B-bitvec* | *c1* ⦄[x::=v]$_{τv}$›
using *subst-infer-v infer-e-splitI* **by** *metis*
thus ‹ Θ ; ℬ ; Γ′[x::=v]$_{Γv}$ @ Γ ⊢ *v1*[x::=v]$_{vv}$ ⇒ ⦃ *z1* : *B-bitvec* | *c1*[x::=v]$_{cv}$ ⦄›
using *infer-e-splitI subst-tv.simps fresh-Pair* **by** *metis*
have ‹*x* ≠ *z2* › **using** *infer-e-splitI* **by** *force*
have (⦃ *z2* : *B-int* | ([*leq* [[*L-num 0*]^{v}]^{ce} [[*z2*]^{v}]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce})
AND ([*leq* [[*z2*]^{v}]^{ce} [| [*v1*[x::=v]$_{vv}$]^{ce} |]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}) ⦄) =
(⦃ *z2* : *B-int* | ([*leq* [[*L-num 0*]^{v}]^{ce} [[*z2*]^{v}]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce})
AND ([*leq* [[*z2*]^{v}]^{ce} [| [*v1*]^{ce} |]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}) ⦄[x::=v]$_{τv}$)
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps* **using** ‹*x* ≠ *z2*› *infer-e-splitI fresh-Pair* **by** *simp*
thus ‹Θ ; ℬ ; Γ′[x::=v]$_{Γv}$ @
Γ ⊢ *v2*[x::=v]$_{vv}$ ⇐ ⦃ *z2* : *B-int* | [*leq* [[*L-num 0*]^{v}]^{ce} [[*z2*]^{v}]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}
AND [*leq* [[*z2*]^{v}]^{ce} [| [*v1*[x::=v]$_{vv}$]^{ce} |]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce} ⦄›
using *infer-e-splitI subst-infer-check-v fresh-Pair* **by** *metis*

show ‹*atom z1* ♯ *AE-split v1*[x::=v]$_{vv}$ *v2*[x::=v]$_{vv}$› **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*
show ‹*atom z2* ♯ *AE-split v1*[x::=v]$_{vv}$ *v2*[x::=v]$_{vv}$› **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*
show ‹*atom z3* ♯ *AE-split v1*[x::=v]$_{vv}$ *v2*[x::=v]$_{vv}$› **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*

show ‹*atom z3* ♯ Γ′[x::=v]$_{Γv}$ @ Γ› **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
show ‹*atom z2* ♯ Γ′[x::=v]$_{Γv}$ @ Γ› **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
show ‹*atom z1* ♯ Γ′[x::=v]$_{Γv}$ @ Γ› **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
qed
thus *?case* **apply** (*subst subst-tv.simps*)
using *infer-e-splitI fresh-Pair* **apply** *metis*
unfolding *subst-tv.simps subst-ev.simps subst-cv.simps subst-cev.simps subst-vv.simps* ∗

using ‹$x \neq z3$› **by** *simp*
qed

lemma *infer-e-uniqueness*:
assumes Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_1 \Rightarrow \tau_1$ **and** Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_1 \Rightarrow \tau_2$
shows $\tau_1 = \tau_2$
using *assms* **proof**(*nominal-induct rule*:*e.strong-induct*)
case (*AE-val x*)
then show *?case* **using** *infer-e-elims*(*7*)[*OF AE-val*(*1*)] *infer-e-elims*(*7*)[*OF AE-val*(*2*)] *infer-v-uniqueness* **by** *metis*
next
case (*AE-app f v*)

obtain *x1 b1 c1 s1′ τ1′* **where** *t1*: *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1 τ1′ s1′*))) = *lookup-fun* Φ *f* $\wedge$ $\tau_1 = \tau 1'[x1::=v]_{\tau v}$ **using** *infer-e-app2E*[*OF AE-app*(*1*)] **by** *metis*
moreover obtain *x2 b2 c2 s2′ τ2′* **where** *t2*: *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x2 b2 c2 τ2′ s2′*))) = *lookup-fun* Φ *f* $\wedge$ $\tau_2 = \tau 2'[x2::=v]_{\tau v}$ **using** *infer-e-app2E*[*OF AE-app*(*2*)] **by** *metis*

have $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$ **using** *t1* **and** *t2* *fun-ret-unique* **by** *metis*
thus *?thesis* **using** *t1 t2* **by** *auto*
next
case (*AE-appP f b v*)
obtain *bv1 x1 b1 c1 s1′ τ1′* **where** *t1*: *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1′ s1′*))) = *lookup-fun* Φ *f* $\wedge$ $\tau_1 = \tau 1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v}$ **using** *infer-e-appP2E*[*OF AE-appP*(*1*)] **by** *metis*
moreover obtain *bv2 x2 b2 c2 s2′ τ2′* **where** *t2*: *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2′ s2′*))) = *lookup-fun* Φ *f* $\wedge$ $\tau_2 = \tau 2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$ **using** *infer-e-appP2E*[*OF AE-appP*(*2*)] **by** *metis*

have $\tau 1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau 2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$ **using** *t1* **and** *t2* *fun-poly-ret-unique* **by** *metis*
thus *?thesis* **using** *t1 t2* **by** *auto*
next
case (*AE-op opp v1 v2*)
show *?case* **proof**(*rule opp.exhaust*)
assume *opp* = *plus*
hence Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op Plus v1 v2* $\Rightarrow \tau_1$ **and** Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op Plus v1 v2* $\Rightarrow \tau_2$ **using** *AE-op* **by** *auto*
thus *?thesis* **using** *infer-e-elims*(*11*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op Plus v1 v2* $\Rightarrow \tau_1$›] *infer-e-elims*(*11*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op Plus v1 v2* $\Rightarrow \tau_2$›]
by *force*
next
assume *opp* = *leq*
hence *opp* = *LEq* **using** *opp.strong-exhaust* **by** *auto*
hence Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op LEq v1 v2* $\Rightarrow \tau_1$ **and** Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op LEq v1 v2* $\Rightarrow \tau_2$ **using** *AE-op* **by** *auto*
thus *?thesis* **using** *infer-e-elims*(*12*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op LEq v1 v2* $\Rightarrow \tau_1$›] *infer-e-elims*(*12*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AE-op LEq v1 v2* $\Rightarrow \tau_2$›]
by *force*
next
assume *opp* = *eq*
hence *opp* = *Eq* **using** *opp.strong-exhaust* **by** *auto*

hence Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op Eq v1 v2* ⇒ τ_1 **and** Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op Eq v1 v2* ⇒ τ_2 **using** *AE-op* **by** *auto*
thus *?thesis* **using** *infer-e-elims(25)[OF ‹Θ ; Φ ; $\mathcal{B}$; GNil ; Δ ⊢ AE-op Eq v1 v2 ⇒ τ_1›]* *infer-e-elims(25)[OF ‹Θ ; Φ ; $\mathcal{B}$; GNil ; Δ ⊢ AE-op Eq v1 v2 ⇒ τ_2›]*
by *force*
qed
next
case (*AE-concat v1 v2*)

obtain *z3::x* **where** *t1:*τ_1 = ⦃ *z3* : *B-bitvec* | [[*z3*]^v]^{ce} == *CE-concat* [*v1*]^{ce} [*v2*]^{ce} ⦄ ∧ *atom z3* ♯ *v1* ∧ *atom z3* ♯ *v2* **using** *infer-e-elims(18)[OF AE-concat(1)]* **by** *metis*
obtain *z3'::x* **where** *t2:*τ_2 = ⦃ *z3'* : *B-bitvec* | [[*z3'*]^v]^{ce} == *CE-concat* [*v1*]^{ce} [*v2*]^{ce} ⦄ ∧ *atom z3'* ♯ *v1* ∧ *atom z3'* ♯ *v2* **using** *infer-e-elims(18)[OF AE-concat(2)]* **by** *metis*

thus *?case* **using** *t1 t2 type-e-eq ce.fresh* **by** *metis*

next
case (*AE-fst v*)

obtain *z1* **and** *b1* **where** τ_1 = ⦃ *z1* : *b1* | *CE-val* (*V-var z1*) == (*CE-fst* [*v*]^{ce}) ⦄ **using** *infer-v-form AE-fst* **by** *auto*

obtain *xx* :: *x* **and** *bb* :: *b* **and** *xxa* :: *x* **and** *bba* :: *b* **and** *cc* :: *c* **where**
f1: τ_2 = ⦃ *xx* : *bb* | *CE-val* (*V-var xx*) == *CE-fst* [*v*]^{ce} ⦄ ∧ Θ ; $\mathcal{B}$; *GNil*$\vdash_{wf}$ Δ ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇒ ⦃ *xxa* : *B-pair bb bba* | *cc* ⦄ ∧ *atom xx* ♯ *v*
using *infer-e-elims(8)[OF AE-fst(2)]* **by** *metis*
obtain *xxb* :: *x* **and** *bbb* :: *b* **and** *xxc* :: *x* **and** *bbc* :: *b* **and** *cca* :: *c* **where**
f2: τ_1 = ⦃ *xxb* : *bbb* | *CE-val* (*V-var xxb*) == *CE-fst* [*v*]^{ce} ⦄ ∧ Θ ; $\mathcal{B}$; *GNil*$\vdash_{wf}$ Δ ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇒ ⦃ *xxc* : *B-pair bbb bbc* | *cca* ⦄ ∧ *atom xxb* ♯ *v*
using *infer-e-elims(8)[OF AE-fst(1)]* **by** *metis*
then have *B-pair bb bba* = *B-pair bbb bbc*
using *f1* **by** (*metis* (*no-types*) *b-of.simps infer-v-uniqueness*)
then have ⦃ *xx* : *bbb* | *CE-val* (*V-var xx*) == *CE-fst* [*v*]^{ce} ⦄ = τ_2
using *f1* **by** *auto*
then show *?thesis*
using *f2* **by** (*meson ce.fresh fresh-GNil type-e-eq wfG-x-fresh-in-v-simple*)
next
case (*AE-snd v*)
obtain *xx* :: *x* **and** *bb* :: *b* **and** *xxa* :: *x* **and** *bba* :: *b* **and** *cc* :: *c* **where**
f1: τ_2 = ⦃ *xx* : *bba* | *CE-val* (*V-var xx*) == *CE-snd* [*v*]^{ce} ⦄ ∧ Θ ; $\mathcal{B}$; *GNil*$\vdash_{wf}$ Δ ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇒ ⦃ *xxa* : *B-pair bb bba* | *cc* ⦄ ∧ *atom xx* ♯ *v*
using *infer-e-elims(9)[OF AE-snd(2)]* **by** *metis*
obtain *xxb* :: *x* **and** *bbb* :: *b* **and** *xxc* :: *x* **and** *bbc* :: *b* **and** *cca* :: *c* **where**
f2: τ_1 = ⦃ *xxb* : *bbc* | *CE-val* (*V-var xxb*) == *CE-snd* [*v*]^{ce} ⦄ ∧ Θ ; $\mathcal{B}$; *GNil*$\vdash_{wf}$ Δ ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇒ ⦃ *xxc* : *B-pair bbb bbc* | *cca* ⦄ ∧ *atom xxb* ♯ *v*
using *infer-e-elims(9)[OF AE-snd(1)]* **by** *metis*
then have *B-pair bb bba* = *B-pair bbb bbc*
using *f1* **by** (*metis* (*no-types*) *b-of.simps infer-v-uniqueness*)
then have ⦃ *xx* : *bbc* | *CE-val* (*V-var xx*) == *CE-snd* [*v*]^{ce} ⦄ = τ_2
using *f1* **by** *auto*
then show *?thesis*
using *f2* **by** (*meson ce.fresh fresh-GNil type-e-eq wfG-x-fresh-in-v-simple*)

next
case (*AE-mvar x*)
then show *?case* **using** *infer-e-elims*(*10*)[*OF AE-mvar*(*1*)] *infer-e-elims*(*10*)[*OF AE-mvar*(*2*)] *wfD-unique*
by *metis*
next
case (*AE-len x*)
then show *?case* **using** *infer-e-elims*(*16*)[*OF AE-len*(*1*)] *infer-e-elims*(*16*)[*OF AE-len*(*2*)] **by** *force*
next
case (*AE-split x1a x2*)
then show *?case* **using** *infer-e-elims*(*22*)[*OF AE-split*(*1*)] *infer-e-elims*(*22*)[*OF AE-split*(*2*)] **by** *force*
qed

14.8 Statements

lemma *subst-infer-check-v1*:
fixes v::v **and** v'::v **and** Γ::Γ
assumes $\Gamma = \Gamma_1 @((x,b_1,c0[z0::=[x]^v]_{cv}) \#_\Gamma \Gamma_2)$ **and**
$\Theta ; \mathcal{B} ; \Gamma_2 \vdash v \Rightarrow \tau_1$ **and**
$\Theta ; \mathcal{B} ; \Gamma \vdash v' \Leftarrow \tau_2$ **and**
$\Theta ; \mathcal{B} ; \Gamma_2 \vdash \tau_1 \lesssim \{\!| z0 : b_1 \mid c0 |\!\}$ **and** *atom z0* $\sharp$ (x,v)
shows $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash v'[x::=v]_{vv} \Leftarrow \tau_2[x::=v]_{\tau v}$
using *subst-g-inside check-v-wf assms subst-infer-check-v* **by** *metis*

lemma *infer-v-c-valid*:
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash \tau \lesssim \{\!| z : b \mid c |\!\}$
shows ‹$\Theta ; \mathcal{B} ; \Gamma \models c[z::=v]_{cv}$ ›
proof −
obtain *z1* **and** *b1* **and** *c1* **where** $*$:$\tau = \{\!| z1 : b1 \mid c1 |\!\} \wedge$ *atom z1* $\sharp$ (c,v,Γ) **using** *obtain-fresh-z*
by *metis*
then have $b1 = b$ **using** *assms subtype-eq-base* **by** *metis*
moreover then have $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \{\!| z1 : b \mid c1 |\!\}$ **using** *assms* $*$ **by** *auto*

moreover have $\Theta ; \mathcal{B} ; (z1, b, c1) \#_\Gamma \Gamma \models c[z::=[z1]^v]_{cv}$ **proof** −
have $\Theta ; \mathcal{B} ; (z1, b, c1[z1::=[z1]^v]_v) \#_\Gamma \Gamma \models c[z::=[z1]^v]_v$
using *subtype-valid*[*OF assms*(*2*), *of z1 z1 b c1 z c*] $*$ *fresh-prodN* ‹$b1 = b$› **by** *metis*
moreover have $c1[z1::=[z1]^v]_v = c1$ **using** *subst-v-v-def* **by** *simp*
ultimately show *?thesis* **using** *subst-v-c-def* **by** *metis*
qed
ultimately show *?thesis* **using** $*$ *fresh-prodN subst-valid-simple* **by** *metis*
qed

Substitution Lemma for Statements

lemma *subst-infer-check-s*:
fixes v::v **and** s::s **and** cs::*branch-s* **and** x::x **and** c::c **and** b::b **and**
Γ_1::Γ **and** Γ_2::Γ **and** *css*::*branch-list*
assumes $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **and** $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \{\!| z : b \mid c |\!\}$ **and**
atom z $\sharp$ (x, v)
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma; \Delta \vdash s \Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2 @((x,b,c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1)) \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash s[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$
and
$\Theta ; \Phi ; \mathcal{B} ; \Gamma; \Delta;$ *tid* ; *cons* ; *const* ; $v' \vdash cs \Leftarrow \tau' \Longrightarrow$

$\Gamma = (\Gamma_2 @((x,b,c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1)) \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v};$
$tid ; cons ; const ; v'[x::=v]_{vv} \vdash cs[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$
and
$\Theta ; \Phi ; \mathcal{B} ; \Gamma; \Delta; tid ; dclist ; v' \vdash css \Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2 @((x,b,c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1)) \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v}; tid ; dclist ; v'[x::=v]_{vv} \vdash$
subst-branchlv css x v $\Leftarrow \tau'[x::=v]_{\tau v}$

using *assms* **proof**(*nominal-induct* τ' **and** τ' **and** τ' *avoiding*: *x v arbitrary*: Γ_2 **and** Γ_2 **and** Γ_2
rule: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ v' τ' τ'')

have *sg*: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v} @\Gamma_1$ **using** *check-valI* **by** *subst-mth*
have $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash (AS\text{-}val\ (v'[x::=v]_{vv})) \Leftarrow \tau''[x::=v]_{\tau v}$ **proof**
have $\Theta ; \mathcal{B} ; \Gamma_1 \vdash_{wf} v : b$ **using** *infer-v-v-wf subtype-eq-base2 b-of.simps check-valI* **by** *metis*
thus ‹$\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$› **using** *wf-subst*(*15*) *check-valI* **by** *auto*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *check-valI* **by** *auto*
show ‹ $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash v'[x::=v]_{vv} \Rightarrow \tau'[x::=v]_{\tau v}$› **proof**(*subst sg*, *rule subst-infer-v*)
show $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **using** *check-valI* **by** *auto*
show $\Theta ; \mathcal{B} ; \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1 \vdash v' \Rightarrow \tau'$ **using** *check-valI* **by** *metis*
show $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \{\!\mid z: b \mid c \mid\!\}$ **using** *check-valI* **by** *auto*
show *atom z* $\sharp$ (x, v) **using** *check-valI* **by** *auto*
qed
show ‹$\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash \tau'[x::=v]_{\tau v} \lesssim \tau''[x::=v]_{\tau v}$› **using** *subst-subtype-tau check-valI sg* **by** *metis*
qed

thus *?case* **using** *Typing.check-valI subst-sv.simps sg* **by** *auto*
next
case (*check-letI xa* Θ Φ $\mathcal{B}$ Γ Δ *ea* τa *za sa ba ca*)
have $*$:$(AS\text{-}let\ xa\ ea\ sa)[x::=v]_{sv} = (AS\text{-}let\ xa\ (ea[x::=v]_{ev})\ sa[x::=v]_{sv})$
using *subst-sv.simps* ‹ *atom xa* $\sharp$ *x*› ‹ *atom xa* $\sharp$ *v*› **by** *auto*
show *?case* **unfolding** $*$ **proof**

show *atom xa* $\sharp$ $(\Theta,\Phi,\mathcal{B},\Gamma[x::=v]_{\Gamma v},\Delta[x::=v]_{\Delta v},ea[x::=v]_{ev},\tau a[x::=v]_{\tau v})$
by(*subst-tuple-mth add*: *check-letI*)

show *atom za* $\sharp$ $(xa,\Theta,\Phi,\mathcal{B},\Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v},ea[x::=v]_{ev},$
$\tau a[x::=v]_{\tau v},sa[x::=v]_{sv})$
by(*subst-tuple-mth add*: *check-letI*)

show $\Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash$
$ea[x::=v]_{ev} \Rightarrow \{\!\mid za : ba \mid ca[x::=v]_{cv} \mid\!\}$
proof −
have $\Theta; \Phi; \mathcal{B}; \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1; \Delta[x::=v]_{\Delta v} \vdash$
$ea[x::=v]_{ev} \Rightarrow \{\!\mid za : ba \mid ca \mid\!\}[x::=v]_{\tau v}$
using *check-letI subst-infer-e* **by** *metis*
thus *?thesis* **using** *check-letI subst-tv.simps*
by (*metis fresh-prod2I infer-e-wf subst-g-inside-simple*)
qed

show $\Theta; \Phi; \mathcal{B}; (xa,\ ba,\ ca[x::=v]_{cv}[za::=V\text{-}var\ xa]_v)\ \#_\Gamma\ \Gamma[x::=v]_{\Gamma v};$
$\Delta[x::=v]_{\Delta v}\ \vdash\ sa[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$

proof –

have $\Theta; \Phi; \mathcal{B}; ((xa,\ ba,\ ca[za::=V\text{-}var\ xa]_v)\ \#_\Gamma\ \Gamma)[x::=v]_{\Gamma v}\ ;$
$\Delta[x::=v]_{\Delta v}\ \vdash\ sa[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$

apply(*rule check-letI*(*23*)[*of* $(xa,\ ba,\ ca[za::=V\text{-}var\ xa]_{cv})\ \#_\Gamma\ \Gamma_2$])

by(*metis check-letI append-g.simps subst-defs*)+

moreover have $(xa,\ ba,\ ca[x::=v]_{cv}[za::=V\text{-}var\ xa]_{cv})\ \#_\Gamma\ \Gamma[x::=v]_{\Gamma v} =$
$((xa,\ ba,\ ca[za::=V\text{-}var\ xa]_{cv})\ \#_\Gamma\ \Gamma)[x::=v]_{\Gamma v}$

using *subst-cv-commute subst-gv.simps check-letI*

by (*metis ms-fresh-all*(*39*) *ms-fresh-all*(*49*) *subst-cv-commute-full*)

ultimately show *?thesis*

using *subst-defs* **by** *auto*

qed

qed

next

case (*check-assertI xa* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *ca* τ *s*)

show *?case* **unfolding** *subst-sv.simps* **proof**

show ‹*atom xa* $\sharp$ $(\Theta,\ \Phi,\ \mathcal{B},\ \Gamma[x::=v]_{\Gamma v},\ \Delta[x::=v]_{\Delta v},\ ca[x::=v]_{cv},\ \tau[x::=v]_{\tau v},\ s[x::=v]_{sv})$›

by(*subst-tuple-mth add*: *check-assertI*)

have $xa \neq x$ **using** *check-assertI* **by** *fastforce*

thus ‹ $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ (xa,\ B\text{-}bool,\ ca[x::=v]_{cv})\ \#_\Gamma\ \Gamma[x::=v]_{\Gamma v}\ ;\ \Delta[x::=v]_{\Delta v}\ \vdash\ s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$›

using *check-assertI*(*12*)[*of* $(xa,\ B\text{-}bool,\ c)\ \#_\Gamma\ \Gamma_2\ x\ v$] *check-assertI subst-gv.simps append-g.simps*
by *metis*

have ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2[x::=v]_{\Gamma v}\ @\ \Gamma_1\ \models\ ca[x::=v]_{cv}$ › **proof**(*rule subst-valid*)

show ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1\ \models\ c[z::=v]_{cv}$ › **using** *infer-v-c-valid check-assertI* **by** *metis*

show ‹ $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1 \vdash_{wf}\ v\ :\ b$ › **using** *check-assertI infer-v-wf b-of.simps subtype-eq-base*

by (*metis subtype-eq-base2*)

show ‹ $\Theta\ ;\ \mathcal{B}\ \vdash_{wf}\ \Gamma_1$ › **using** *check-assertI infer-v-wf* **by** *metis*

have $\Theta\ ;\ \mathcal{B}\ \vdash_{wf}\ \Gamma_2\ @\ (x,\ b,\ c[z::=[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma_1$ **using** *check-assertI wfX-wfY* **by** *metis*

thus ‹*atom x* $\sharp$ Γ_1› **using** *check-assertI wfG-suffix wfG-elims* **by** *metis*

moreover have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1 \vdash_{wf}\ \{\!|\ z\ :\ b\ \ |\ c\ |\!\}$ **using** *subtype-wfT check-assertI* **by** *metis*

moreover have $x \neq z$ **using** *fresh-Pair check-assertI fresh-x-neq* **by** *metis*

ultimately show ‹*atom x* $\sharp$ *c*› **using** *check-assertI wfT-fresh-c* **by** *metis*

show ‹ $\Theta\ ;\ \mathcal{B}\ \vdash_{wf}\ \Gamma_2\ @\ (x,\ b,\ c[z::=[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma_1$ › **using** *check-assertI wfX-wfY* **by** *metis*

show ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2\ @\ (x,\ b,\ c[z::=[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma_1\ \models\ ca$ › **using** *check-assertI* **by** *auto*

qed

thus ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ \models\ ca[x::=v]_{cv}$ › **using** *check-assertI*

proof –

show *?thesis*

by (*metis* (*no-types*) ‹$\Gamma = \Gamma_2\ @\ (x,\ b,\ c[z::=[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma_1$› ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \models ca$› ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2[x::=v]_{\Gamma v}$
$@\ \Gamma_1 \models ca[x::=v]_{cv}$› *subst-g-inside valid-g-wf*)

qed

have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1 \vdash_{wf}\ v\ :\ b$ **using** *infer-v-wf b-of.simps check-assertI*

by (*metis subtype-eq-base2*)

thus ‹ $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v} \vdash_{wf}\ \Delta[x::=v]_{\Delta v}$ › **using** *wf-subst2*(*6*) *check-assertI* **by** *metis*

qed

next

case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist vv cs* τ *css*)
show *?case* **unfolding** ∗ **using** *subst-sv.simps check-branch-list-consI* **by** *simp*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
show *?case* **unfolding** ∗ **using** *subst-sv.simps check-branch-list-finalI* **by** *simp*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const xa* Φ *tid cons va sa*)
hence ∗:(*AS-branch cons xa sa*)$[x::=v]_{sv}$ = (*AS-branch cons xa sa*$[x::=v]_{sv}$) **using** *subst-branchv.simps fresh-Pair* **by** *metis*
show *?case* **unfolding** ∗ **proof**

show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$
using *wf-subst check-branch-s-branchI subtype-eq-base2 b-of.simps infer-v-wf* **by** *metis*

show $\vdash_{wf}$ Θ **using** *check-branch-s-branchI* **by** *metis*

show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v} \vdash_{wf} \tau[x::=v]_{\tau v}$
using *wf-subst check-branch-s-branchI subtype-eq-base2 b-of.simps infer-v-wf* **by** *metis*

show *wft*:Θ ; {||} ; *GNil*$\vdash_{wf}$ *const* **using** *check-branch-s-branchI* **by** *metis*

show *atom xa* ♯ (Θ, Φ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, *tid*, *cons*, *const*,*va*$[x::=v]_{vv}$, $\tau[x::=v]_{\tau v}$)
apply(*unfold fresh-prodN*, (*simp add*: *check-branch-s-branchI*)+)
apply(*rule*,*metis fresh-z-subst-g check-branch-s-branchI fresh-Pair*)
by(*metis fresh-subst-dv check-branch-s-branchI fresh-Pair*)

have Θ ; Φ ; $\mathcal{B}$; ((*xa*, *b-of const*, *CE-val va* == *CE-val*(*V-cons tid cons* (*V-var xa*)) *AND c-of const xa*) $\#_{\Gamma}$ Γ)$[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *sa*$[x::=v]_{sv}$ ⇐ $\tau[x::=v]_{\tau v}$
using *check-branch-s-branchI* **by** (*metis append-g.simps*(*2*))

moreover have (*xa*, *b-of const*, *CE-val va*$[x::=v]_{vv}$ == *CE-val* (*V-cons tid cons* (*V-var xa*)) *AND c-of* (*const*) *xa*) $\#_{\Gamma}$ $\Gamma[x::=v]_{\Gamma v}$ =
((*xa*, *b-of const* , *CE-val va* == *CE-val* (*V-cons tid cons* (*V-var xa*)) *AND c-of const xa*) $\#_{\Gamma}$ Γ)$[x::=v]_{\Gamma v}$
proof −
have ∗:*xa* ≠ *x* **using** *check-branch-s-branchI fresh-at-base* **by** *metis*
have *atom x* ♯ *const* **using** *wfT-nil-supp*[*OF wft*] *fresh-def* **by** *auto*
hence *atom x* ♯ (*const*,*xa*) **using** *fresh-at-base wfT-nil-supp*[*OF wft*] *fresh-Pair fresh-def* ∗ **by** *auto*
moreover hence (*c-of* (*const*) *xa*)$[x::=v]_{cv}$ = *c-of* (*const*) *xa*
using *c-of-fresh*[*of x const xa*] *forget-subst-cv wfT-nil-supp wft* **by** *metis*
moreover hence (*V-cons tid cons* (*V-var xa*))$[x::=v]_{vv}$ = (*V-cons tid cons* (*V-var xa*)) **using** *check-branch-s-branchI subst-vv.simps* ∗ **by** *metis*
ultimately show *?thesis* **using** *subst-gv.simps check-branch-s-branchI subst-cv.simps subst-cev.simps* ∗ **by** *presburger*
qed

ultimately show Θ ; Φ ; $\mathcal{B}$; (*xa*, *b-of const*, *CE-val va*$[x::=v]_{vv}$ == *CE-val* (*V-cons tid cons* (*V-var xa*)) *AND c-of const xa*) $\#_{\Gamma}$ $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *sa*$[x::=v]_{sv}$ ⇐ $\tau[x::=v]_{\tau v}$
by *metis*
qed

next

case (*check-let2I xa* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τ*a s2*)
hence ∗:(*AS-let2 xa t s1 s2*)$[x::=v]_{sv}$ = (*AS-let2 xa t*$[x::=v]_{\tau v}$ (*s1*$[x::=v]_{sv}$) *s2*$[x::=v]_{sv}$) **using** *subst-sv.simps fresh-Pair* **by** *metis*
have $xa \neq x$ **using** *check-let2I fresh-at-base* **by** *metis*
show *?case* **unfolding** ∗ **proof**
show *atom xa* ♯ (Θ, Φ, $\mathcal{B}$, $G[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $t[x::=v]_{\tau v}$, *s1*$[x::=v]_{sv}$, τ*a*$[x::=v]_{\tau v}$)
by(*subst-tuple-mth add*: *check-let2I*)
show Θ ; Φ ; $\mathcal{B}$; $G[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *s1*$[x::=v]_{sv}$ ⇐ $t[x::=v]_{\tau v}$ **using** *check-let2I* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; ((*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *G*)$[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *s2*$[x::=v]_{sv}$ ⇐ τ*a*$[x::=v]_{\tau v}$
proof(*rule check-let2I*(*14*))
show ‹(*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *G* = (((*xa*, *b-of t*, *c-of t xa*)$\#_\Gamma$ Γ_2)) @ (*x*, *b*, *c*$[z::=[\ x\ ]^v]_{cv}$) $\#_\Gamma$ Γ_1›
using *check-let2I append-g.simps* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ_1 ⊢ *v* ⇒ τ› **using** *check-let2I* **by** *metis*
show ‹Θ ; $\mathcal{B}$; Γ_1 ⊢ τ ≲ ⦃ *z* : *b* | *c* ⦄› **using** *check-let2I* **by** *metis*
show ‹*atom z* ♯ (*x*, *v*)› **using** *check-let2I* **by** *metis*
qed
moreover have *c-of t*$[x::=v]_{\tau v}$ *xa* = (*c-of t xa*)$[x::=v]_{cv}$ **using** *subst-v-c-of fresh-Pair check-let2I* **by** *metis*
moreover have *b-of t*$[x::=v]_{\tau v}$ = *b-of t* **using** *b-of.simps subst-tv.simps b-of-subst* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; (*xa*, *b-of t*$[x::=v]_{\tau v}$, *c-of t*$[x::=v]_{\tau v}$ *xa*) $\#_\Gamma$ $G[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *s2*$[x::=v]_{sv}$ ⇐ τ*a*$[x::=v]_{\tau v}$
using *check-let2I*(*14*) *subst-gv.simps* ‹$xa \neq x$› *b-of.simps* **by** *metis*
qed

next

case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ′ *va* τ′′ *s*)
have ∗∗: $\Gamma[x::=v]_{\Gamma v}$ = $\Gamma_2[x::=v]_{\Gamma v}$@$\Gamma_1$ **using** *subst-g-inside check-s-wf check-varI* **by** *meson*

have Θ ; Φ ;$\mathcal{B}$; *subst-gv* Γ *x v* ; $\Delta[x::=v]_{\Delta v}$ ⊢ *AS-var u* τ′$[x::=v]_{\tau v}$ (*va*$[x::=v]_{vv}$) (*subst-sv s x v*) ⇐ τ′′$[x::=v]_{\tau v}$
proof(*rule Typing.check-varI*)
show *atom u* ♯ (Θ, Φ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, τ′$[x::=v]_{\tau v}$, *va*$[x::=v]_{vv}$, τ′′$[x::=v]_{\tau v}$)
by(*subst-tuple-mth add*: *check-varI*)
show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ ⊢ *va*$[x::=v]_{vv}$ ⇐ τ′$[x::=v]_{\tau v}$ **using** *check-varI subst-infer-check-v* ∗∗ **by** *metis*
show Θ ; Φ ; $\mathcal{B}$; *subst-gv* Γ *x v* ; (*u*, τ′$[x::=v]_{\tau v}$) $\#_\Delta$ $\Delta[x::=v]_{\Delta v}$ ⊢ *s*$[x::=v]_{sv}$ ⇐ τ′′$[x::=v]_{\tau v}$ **proof** −
have *wfD* Θ $\mathcal{B}$ (Γ_2 @ (*x*, *b*, *c*$[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ_1) ((*u*,τ′)$\#_\Delta$ Δ) **using** *check-varI check-s-wf* **by** *meson*
moreover have *wfV* Θ $\mathcal{B}$ Γ_1 *v* (*b-of* τ) **using** *infer-v-wf check-varI*(*6*) *check-varI* **by** *metis*
have *wfD* Θ $\mathcal{B}$ ($\Gamma[x::=v]_{\Gamma v}$) ((*u*, τ′$[x::=v]_{\tau v}$) $\#_\Delta$ $\Delta[x::=v]_{\Delta v}$) **proof**(*subst subst-dv.simps*(*2*)[*symmetric*], *subst* ∗∗, *rule wfD-subst*)
show Θ ; $\mathcal{B}$; Γ_1 ⊢ *v* ⇒ τ **using** *check-varI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ_2 @ (*x*, *b*, *c*$[z::=[x]^v]_{cv}$) $\#_\Gamma$ $\Gamma_1$$\vdash_{wf}$ (*u*, τ′) $\#_\Delta$ Δ **using** *check-varI check-s-wf* **by** *simp*
show *b-of* τ = *b* **using** *check-varI subtype-eq-base2 b-of.simps* **by** *auto*
qed
thus *?thesis* **using** *check-varI* **by** *auto*
qed

qed
moreover have *atom u* $\sharp$ *(x,v)* **using** *u-fresh-xv* **by** *auto*
ultimately show *?case* **using** *subst-sv.simps(7)* **by** *auto*

next
case (*check-assignI P* Φ $\mathcal{B}$ Γ Δ *u* $\tau 1$ v' *z1* τ')

have *wfG P* $\mathcal{B}$ Γ **using** *check-v-wf check-assignI* **by** *simp*
hence *gs*: $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 = $\Gamma[x::=v]_{\Gamma v}$ **using** *subst-g-inside check-assignI* **by** *simp*

have *P* ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *AS-assign u* ($v'[x::=v]_{vv}$) $\Leftarrow$ $\tau'[x::=v]_{\tau v}$
proof(*rule Typing.check-assignI*)
show *P* $\vdash_{wf}$ Φ **using** *check-assignI* **by** *auto*
show *wfD P* $\mathcal{B}$ ($\Gamma[x::=v]_{\Gamma v}$) $\Delta[x::=v]_{\Delta v}$ **using** *wf-subst(15)[OF check-assignI(2)] gs infer-v-v-wf check-assignI b-of.simps subtype-eq-base2* **by** *metis*
thus (*u*, $\tau 1[x::=v]_{\tau v}$) $\in$ *setD* $\Delta[x::=v]_{\Delta v}$ **using** *check-assignI subst-dv-member* **by** *metis*
thus *P* ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash$ $v'[x::=v]_{vv}$ $\Leftarrow$ $\tau 1[x::=v]_{\tau v}$ **using** *subst-infer-check-v check-assignI gs* **by** *metis*

have *P* ; $\mathcal{B}$; $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 $\vdash$ $\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}[x::=v]_{\tau v}$ $\lesssim$ $\tau'[x::=v]_{\tau v}$ **proof**(*rule subst-subtype-tau*)
show *P* ; $\mathcal{B}$; Γ_1 $\vdash$ *v* $\Rightarrow$ τ **using** *check-assignI* **by** *auto*
show *P* ; $\mathcal{B}$; Γ_1 $\vdash$ τ $\lesssim$ $\{\!\mid$ *z* : *b* $\mid$ *c* $\mid\!\}$ **using** *check-assignI* **by** *meson*
show *P* ; $\mathcal{B}$; Γ_2 @ (*x*, *b*, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ_1 $\vdash$ $\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}$ $\lesssim$ τ' **using** *check-assignI*
by (*metis Abs1-eq-iff(3)* τ*.eq-iff c.fresh(1) c.perm-simps(1)*)
show *atom z* $\sharp$ (*x*, *v*) **using** *check-assignI* **by** *auto*
qed
moreover have $\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}[x::=v]_{\tau v}$ = $\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}$ **using** *subst-tv.simps subst-cv.simps check-assignI* **by** *presburger*
ultimately show *P* ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash$ $\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}$ $\lesssim$ $\tau'[x::=v]_{\tau v}$ **using** *gs* **by** *auto*
qed
thus *?case* **using** *subst-sv.simps(5)* **by** *auto*

next
case (*check-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1* z' *s2* τ')
have *wfG* Θ $\mathcal{B}$ (Γ_2 @ (*x*, *b*, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ_1) **using** *check-whileI check-s-wf* **by** *meson*
hence $**$: $\Gamma[x::=v]_{\Gamma v}$ = $\Gamma_2[x::=v]_{\Gamma v}$@$\Gamma_1$ **using** *subst-g-inside wf check-whileI* **by** *auto*
have *teq*: ($\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}$)$[x::=v]_{\tau v}$ = ($\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}$) **by**(*auto simp add*: *subst-sv.simps check-whileI*)
moreover have ($\{\!\mid$ *z* : *B-unit* $\mid$ *TRUE* $\mid\!\}$) = ($\{\!\mid$ z' : *B-unit* $\mid$ *TRUE* $\mid\!\}$) **using** *type-eq-flip c.fresh flip-fresh-fresh* **by** *metis*
ultimately have *teq2*:($\{\!\mid$ z' : *B-unit* $\mid$ *TRUE* $\mid\!\}$)$[x::=v]_{\tau v}$ = ($\{\!\mid$ z' : *B-unit* $\mid$ *TRUE* $\mid\!\}$) **by** *metis*

hence Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *s1*$[x::=v]_{sv}$ $\Leftarrow$ $\{\!\mid$ z' : *B-bool* $\mid$ *TRUE* $\mid\!\}$ **using** *check-whileI subst-sv.simps subst-top-eq* **by** *metis*
moreover have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *s2*$[x::=v]_{sv}$ $\Leftarrow$ $\{\!\mid$ z' : *B-unit* $\mid$ *TRUE* $\mid\!\}$ **using** *check-whileI subst-top-eq* **by** *metis*
moreover have Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash$ $\{\!\mid$ z' : *B-unit* $\mid$ *TRUE* $\mid\!\}$ $\lesssim$ $\tau'[x::=v]_{\tau v}$ **proof** –
have Θ ; $\mathcal{B}$; $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 $\vdash$ $\{\!\mid$ z' : *B-unit* $\mid$ *TRUE* $\mid\!\}[x::=v]_{\tau v}$ $\lesssim$ $\tau'[x::=v]_{\tau v}$ **proof**(*rule subst-subtype-tau*)
show Θ ; $\mathcal{B}$; Γ_1 $\vdash$ *v* $\Rightarrow$ τ **by**(*auto simp add*: *check-whileI*)
show Θ ; $\mathcal{B}$; Γ_1 $\vdash$ τ $\lesssim$ $\{\!\mid$ *z* : *b* $\mid$ *c* $\mid\!\}$ **by**(*auto simp add*: *check-whileI*)

show Θ ; $\mathcal{B}$; Γ_2 @ $(x,\ b,\ c[z::=[x]^v]_{cv})$ $\#_\Gamma$ Γ_1 $\vdash$ $\{\!\!\{\ z' : B\text{-}unit\ \mid TRUE\ \}\!\!\} \lesssim \tau'$ **using** *check-whileI* **by** *metis*
show *atom z* $\sharp$ $(x,\ v)$ **by**(*auto simp add*: *check-whileI*)
qed
thus *?thesis* **using** *teq2* $**$ **by** *auto*
qed

ultimately have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *AS-while s1*$[x::=v]_{sv}$ *s2*$[x::=v]_{sv}$ $\Leftarrow$ $\tau'[x::=v]_{\tau v}$
using *Typing.check-whileI* **by** *metis*
then show *?case* **using** *subst-sv.simps* **by** *metis*
next
case (*check-seqI P* Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ)
hence *P* ; Φ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *AS-seq* (*s1*$[x::=v]_{sv}$) (*s2*$[x::=v]_{sv}$) $\Leftarrow$ $\tau[x::=v]_{\tau v}$ **using** *Typing.check-seqI subst-top-eq check-seqI* **by** *metis*
then show *?case* **using** *subst-sv.simps* **by** *metis*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v' cs* τ *za*)

have *wf*: *wfG* Θ $\mathcal{B}$ Γ **using** *check-caseI check-v-wf* **by** *simp*
have $**$: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1$ **using** *subst-g-inside wf check-caseI* **by** *auto*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *AS-match* ($v'[x::=v]_{vv}$) (*subst-branchlv cs x v*) $\Leftarrow$ $\tau[x::=v]_{\tau v}$ **proof**(*rule Typing.check-caseI*)
show *check-branch-list* Θ Φ $\mathcal{B}$ ($\Gamma[x::=v]_{\Gamma v}$) $\Delta[x::=v]_{\Delta v}$ *tid dclist* $v'[x::=v]_{vv}$ (*subst-branchlv cs x v*) ($\tau[x::=v]_{\tau v}$) **using** *check-caseI* **by** *auto*
show *AF-typedef tid dclist* $\in$ *set* Θ **using** *check-caseI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash$ $v'[x::=v]_{vv}$ $\Leftarrow$ $\{\!\!\{\ za : B\text{-}id\ tid\ \mid TRUE\ \}\!\!\}$ **proof** $-$
have Θ ; $\mathcal{B}$; Γ_2 @ $(x,\ b,\ c[z::=[x]^v]_{cv})$ $\#_\Gamma$ Γ_1 $\vdash$ v' $\Leftarrow$ $\{\!\!\{\ za : B\text{-}id\ tid\ \mid TRUE\ \}\!\!\}$
using *check-caseI* **by** *argo*
hence Θ ; $\mathcal{B}$; $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 $\vdash$ $v'[x::=v]_{vv}$ $\Leftarrow$ $(\{\!\!\{\ za : B\text{-}id\ tid\ \mid TRUE\ \}\!\!\})[x::=v]_{\tau v}$
using *check-caseI subst-infer-check-v*[*OF check-caseI*(*7*) - *check-caseI*(*8*) *check-caseI*(*9*)] **by** *meson*
moreover have $(\{\!\!\{\ za : B\text{-}id\ tid\ \mid TRUE\ \}\!\!\}) = ((\{\!\!\{\ za : B\text{-}id\ tid\ \mid TRUE\ \}\!\!\})[x::=v]_{\tau v})$
using *subst-cv.simps subst-tv.simps subst-cv-true* **by** *fast*
ultimately show *?thesis* **using** *check-caseI* $**$ **by** *argo*
qed
show *wfTh* Θ **using** *check-caseI* **by** *auto*
qed
thus *?case* **using** *subst-branchlv.simps subst-sv.simps*(*4*) **by** *metis*
next
case (*check-ifI* z' Θ Φ $\mathcal{B}$ Γ Δ *va s1 s2* τ')
show *?case* **unfolding** *subst-sv.simps* **proof**
show ‹*atom* z' $\sharp$ (Θ, Φ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $va[x::=v]_{vv}$, *s1*$[x::=v]_{sv}$, *s2*$[x::=v]_{sv}$, $\tau'[x::=v]_{\tau v}$)›

by(*subst-tuple-mth add*: *check-ifI*)
have $*$:$\{\!\!\{\ z' : B\text{-}bool\ \mid TRUE\ \}\!\!\}[x::=v]_{\tau v} = \{\!\!\{\ z' : B\text{-}bool\ \mid TRUE\ \}\!\!\}$ **using** *subst-tv.simps check-ifI* **by** (*metis freshers*(*19*) *subst-cv.simps*(*1*) *type-eq-subst*)
have $**$: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1$ **using** *subst-g-inside wf check-ifI check-v-wf* **by** *metis*
show ‹Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ $\vdash$ $va[x::=v]_{vv}$ $\Leftarrow$ $\{\!\!\{\ z' : B\text{-}bool\ \mid TRUE\ \}\!\!\}$›
proof(*subst* $*$[*symmetric*], *rule subst-infer-check-v1*[**where** Γ_1=Γ_2 **and** Γ_2=Γ_1])
show $\Gamma = \Gamma_2$ @ $((x,\ b\ ,c[z::=[\ x\ ]^v]_{cv})$ $\#_\Gamma$ $\Gamma_1)$ **using** *check-ifI* **by** *metis*

show ‹ Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Rightarrow \tau$› **using** *check-ifI* **by** *metis*
show ‹Θ ; $\mathcal{B}$; Γ $\vdash$ *va* $\Leftarrow$ ⦃ z' : *B-bool* | *TRUE* ⦄› **using** *check-ifI* **by** *metis*
show ‹Θ ; $\mathcal{B}$; Γ_1 $\vdash \tau \lesssim$ ⦃ z : b | c ⦄› **using** *check-ifI* **by** *metis*
show ‹*atom z* ♯ (x, v)› **using** *check-ifI* **by** *metis*
qed

have ⦃ z' : *b-of* $\tau'[x::=v]_{\tau v}$ | [$va[x::=v]_{vv}$]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* $\tau'[x::=v]_{\tau v}$ z' ⦄ = ⦃ z' : *b-of* τ' | [*va*]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* τ' z' ⦄$[x::=v]_{\tau v}$
by(*simp add*: *subst-tv.simps fresh-Pair check-ifI b-of-subst subst-v-c-of*)

thus ‹ Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s1[x::=v]_{sv}$ $\Leftarrow$ ⦃ z' : *b-of* $\tau'[x::=v]_{\tau v}$ | [$va[x::=v]_{vv}$]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* $\tau'[x::=v]_{\tau v}$ z' ⦄›
using *check-ifI* **by** *metis*
have ⦃ z' : *b-of* $\tau'[x::=v]_{\tau v}$ | [$va[x::=v]_{vv}$]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* $\tau'[x::=v]_{\tau v}$ z' ⦄ = ⦃ z' : *b-of* τ' | [*va*]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* τ' z' ⦄$[x::=v]_{\tau v}$
by(*simp add*: *subst-tv.simps fresh-Pair check-ifI b-of-subst subst-v-c-of*)
thus ‹ Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s2[x::=v]_{sv}$ $\Leftarrow$ ⦃ z' : *b-of* $\tau'[x::=v]_{\tau v}$ | [$va[x::=v]_{vv}$]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* $\tau'[x::=v]_{\tau v}$ z' ⦄›
using *check-ifI* **by** *metis*
qed
qed

lemma *subst-check-check-s*:
fixes *v*::*v* **and** *s*::*s* **and** *cs*::*branch-s* **and** *x*::*x* **and** *c*::*c* **and** *b*::*b* **and** Γ_1::Γ **and** Γ_2::Γ
assumes Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Leftarrow$ ⦃ z : b | c ⦄ **and** *atom z* ♯ (x, v)
and *check-s* Θ Φ $\mathcal{B}$ Γ Δ s τ' **and** Γ = (Γ_2@((x,b,$c[z::=[x]^v]_{cv}$)$\#_\Gamma\Gamma_1$))
shows *check-s* Θ Φ $\mathcal{B}$ (*subst-gv* Γ *x v*) $\Delta[x::=v]_{\Delta v}$ ($s[x::=v]_{sv}$) (*subst-tv* τ' *x v*)
proof –
obtain τ **where** Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Rightarrow \tau \wedge \Theta$; $\mathcal{B}$; $\Gamma_1 \vdash \tau \lesssim$ ⦃ z : b | c ⦄ **using** *check-v-elims assms* **by** *auto*
thus *?thesis* **using** *subst-infer-check-s assms* **by** *metis*
qed

If a statement checks against a type τ then it checks against a supertype of τ

lemma *check-s-supertype*:
fixes *v*::*v* **and** *s*::*s* **and** *cs*::*branch-s* **and** *x*::*x* **and** *c*::*c* **and** *b*::*b* **and** Γ::Γ **and** Γ'::Γ **and** *css*::*branch-list*
shows *check-s* Θ Φ $\mathcal{B}$ G Δ s *t1* $\Longrightarrow$ Θ ; $\mathcal{B}$; $G \vdash$ *t1* $\lesssim$ *t2* $\Longrightarrow$ *check-s* Θ Φ $\mathcal{B}$ G Δ s *t2* **and**
check-branch-s Θ Φ $\mathcal{B}$ G Δ *tid cons const v cs t1* $\Longrightarrow$ Θ ; $\mathcal{B}$; $G \vdash$ *t1* $\lesssim$ *t2* $\Longrightarrow$ *check-branch-s* Θ Φ $\mathcal{B}$ G Δ *tid cons const v cs t2* **and**
check-branch-list Θ Φ $\mathcal{B}$ G Δ *tid dclist v css t1* $\Longrightarrow$ Θ ; $\mathcal{B}$; $G \vdash$ *t1* $\lesssim$ *t2* $\Longrightarrow$ *check-branch-list* Θ Φ $\mathcal{B}$ G Δ *tid dclist v css t2*
proof(*induct arbitrary*: *t2* **and** *t2* **and** *t2 rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ v τ' τ)
hence Θ ; $\mathcal{B}$; $\Gamma \vdash \tau' \lesssim$ *t2* **using** *subtype-trans* **by** *meson*
then show *?case* **using** *subtype-trans Typing.check-valI check-valI* **by** *metis*

next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
show *?case* **proof**(*rule Typing.check-letI*)
show *atom x* ♯(Θ, Φ, $\mathcal{B}$, Γ, Δ, *e*, *t2*) **using** *check-letI subtype-fresh-tau fresh-prodN* **by** *metis*
show *atom z* ♯ (x, Θ, Φ, $\mathcal{B}$, Γ, Δ, *e*, *t2*, *s*) **using** *check-letI*(*2*) *subtype-fresh-tau*[*of z* τ Γ - - *t2*] *fresh-prodN check-letI*(*6*) **by** *auto*

show Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *e* ⇒ ⦃ *z* : *b* | *c* ⦄ **using** *check-letI* **by** *meson*

have *wfG* Θ $\mathcal{B}$ ((*x*, *b*, *c*[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ Γ) **using** *check-letI check-s-wf subst-defs* **by** *metis*
moreover have *toSet* Γ ⊆ *toSet* ((*x*, *b*, *c*[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ Γ) **by** *auto*
ultimately have Θ ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ Γ ⊢ τ ≲ *t2* **using** *subtype-weakening*[*OF check-letI*(*6*)] **by** *auto*
thus Θ ; Φ ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ Γ ; Δ ⊢ *s* ⇐ *t2* **using** *check-letI subst-defs* **by** *metis*
qed
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **using** *Typing.check-branch-list-consI* **by** *auto*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
then show *?case* **using** *Typing.check-branch-list-finalI* **by** *auto*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
show *?case* **proof**
have *atom x* ♯ *t2* **using** *subtype-fresh-tau*[*of x* τ] *check-branch-s-branchI*(*5*,*8*) *fresh-prodN* **by** *metis*
thus *atom x* ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *tid*, *cons*, *const*, *v*, *t2*) **using** *check-branch-s-branchI fresh-prodN* **by** *metis*
show *wfT* Θ $\mathcal{B}$ Γ *t2* **using** *subtype-wf check-branch-s-branchI* **by** *meson*
show Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ ; Δ ⊢ *s* ⇐ *t2* **proof** −
have *wfG* Θ $\mathcal{B}$ ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) **using** *check-s-wf check-branch-s-branchI* **by** *metis*
moreover have *toSet* Γ ⊆ *toSet* ((*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) **by** *auto*
hence Θ ; $\mathcal{B}$; ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) ⊢ τ ≲ *t2*
using *check-branch-s-branchI subtype-weakening*
using *calculation* **by** *presburger*
thus *?thesis* **using** *check-branch-s-branchI* **by** *presburger*
qed
qed(*auto simp add*: *check-branch-s-branchI*)
next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
show *?case* **proof**(*rule Typing.check-ifI*)
have ∗:*atom z* ♯ *t2* **using** *subtype-fresh-tau*[*of z* τ Γ] *check-ifI fresh-prodN* **by** *auto*
thus ‹*atom z* ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *v*, *s1*, *s2*, *t2*)› **using** *check-ifI fresh-prodN* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇐ ⦃ *z* : *B-bool* | *TRUE* ⦄› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *s1* ⇐ ⦃ *z* : *b-of t2* | [*v*]^{ce} == [[*L-true*]^v]^{ce} *IMP c-of t2 z* ⦄›
using *check-ifI subtype-if1 fresh-prodN base-for-lit.simps b-of.simps* ∗ *check-v-wf* **by** *metis*

show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *s2* ⇐ ⦃ *z* : *b-of t2* | [*v*]^{ce} == [[*L-false*]^v]^{ce} *IMP c-of t2 z* ⦄›
using *check-ifI subtype-if1 fresh-prodN base-for-lit.simps b-of.simps* ∗ *check-v-wf* **by** *metis*
qed
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)

show *?case* **proof**
have *atom x* ♯ *t2* **using** *subtype-fresh-tau*[*OF* - - ‹Θ ; $\mathcal{B}$; Γ ⊢ τ ≲ *t2*›] *check-assertI fresh-prodN*

by *simp*
thus *atom x* ♯ (Θ, Φ, *ℬ*, Γ, Δ, *c*, *t2*, *s*) **using** *subtype-fresh-tau check-assertI fresh-prodN* **by** *simp*
have Θ ; *ℬ* ; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ⊢ τ ≲ *t2* **apply**(*rule subtype-weakening*)
using *check-assertI* **apply** *simp*
using *toSet.simps* **apply** *blast*
using *check-assertI check-s-wf* **by** *simp*
thus Θ ; Φ ; *ℬ* ; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ ⊢ *s* ⇐ *t2* **using** *check-assertI* **by** *simp*
show Θ ; *ℬ* ; Γ ⊨ *c* **using** *check-assertI* **by** *auto*
show Θ ; *ℬ* ; Γ $\vdash_{wf}$ Δ **using** *check-assertI* **by** *auto*
qed
next
case (*check-let2I x P* Φ *ℬ G* Δ *t s1* τ *s2*)
have *wfG P ℬ* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G*)
using *check-let2I check-s-wf* **by** *metis*
moreover have *toSet G* ⊆ *toSet* ((*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G*) **by** *auto*
ultimately have ∗:*P* ; *ℬ* ; (*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G* ⊢ τ ≲ *t2* **using** *check-let2I subtype-weakening* **by** *metis*
show *?case* **proof**(*rule Typing.check-let2I*)
have *atom x* ♯ *t2* **using** *subtype-fresh-tau*[*of x* τ] *check-let2I fresh-prodN* **by** *metis*
thus *atom x* ♯ (*P*, Φ, *ℬ*, *G*, Δ, *t*, *s1*, *t2*) **using** *check-let2I fresh-prodN* **by** *metis*
show *P* ; Φ ; *ℬ* ; *G* ; Δ ⊢ *s1* ⇐ *t* **using** *check-let2I* **by** *blast*
show *P* ; Φ ; *ℬ* ;(*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G* ; Δ ⊢ *s2* ⇐ *t2* **using** *check-let2I* ∗ **by** *blast*
qed
next
case (*check-varI u* Θ Φ *ℬ* Γ Δ τ′ *v* τ *s*)
show *?case* **proof**(*rule Typing.check-varI*)
have *atom u* ♯ *t2* **using** *u-fresh-t* **by** *auto*
thus ‹*atom u* ♯ (Θ, Φ, *ℬ*, Γ, Δ, τ′, *v*, *t2*)› **using** *check-varI fresh-prodN* **by** *auto*
show ‹Θ ; *ℬ* ; Γ ⊢ *v* ⇐ τ′› **using** *check-varI* **by** *auto*
show ‹ Θ ; Φ ; *ℬ* ; Γ ; (*u*, τ′) $\#_\Delta$ Δ ⊢ *s* ⇐ *t2*› **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI* Δ *u* τ *P G v z* τ′)
then show *?case* **using** *Typing.check-assignI* **by** (*meson subtype-trans*)
next
case (*check-whileI* Δ *G P s1 z s2* τ′)
then show *?case* **using** *Typing.check-whileI* **by** (*meson subtype-trans*)
next
case (*check-seqI* Δ *G P s1 z s2* τ)
then show *?case* **using** *Typing.check-seqI* **by** *blast*
next
case (*check-caseI* Δ Γ Θ *tid cs* τ *v z*)
then show *?case* **using** *Typing.check-caseI subtype-trans* **by** *meson*

qed

lemma *subtype-let*:
fixes *s*′::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows Θ ; Φ ; *ℬ* ; *GNil* ; Δ ⊢ *AS-let x* e_1 *s* ⇐ τ ⟹ Θ ; Φ ; *ℬ* ; *GNil* ; Δ ⊢ e_1 ⇒ τ_1 ⟹
Θ ; Φ ; *ℬ* ; *GNil* ; Δ ⊢ e_2 ⇒ τ_2 ⟹ Θ ; *ℬ* ; *GNil* ⊢ τ_2 ≲ τ_1 ⟹ Θ ; Φ ; *ℬ* ; *GNil* ; Δ ⊢ *AS-let x* e_2 *s* ⇐ τ **and**

check-branch-s Θ Φ {||} *GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ {||} Γ Δ *tid dclist v css* τ ⟹ *True*
proof(*nominal-induct GNil* Δ *AS-let x* e_1 *s* τ **and** τ **and** τ *avoiding*: e_2 τ_1 τ_2
rule: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-letI x1* Θ Φ *ℬ* Δ *τ1 z1 s1 b1 c1*)
obtain *z2* **and** *b2* **and** *c2* **where** *t2*:τ_2 = ⦃ *z2* : *b2* | *c2* ⦄ ∧ *atom z2* ♯ (*x1*, Θ, Φ, *ℬ*, *GNil*, Δ, e_2, *τ1*, *s1*)
using *obtain-fresh-z* **by** *metis*

obtain *z1a* **and** *b1a* **and** *c1a* **where** *t1*:τ_1 = ⦃ *z1a* : *b1a* | *c1a* ⦄ ∧ *atom z1a* ♯ *x1* **using** *infer-e-uniqueness check-letI* **by** *metis*
hence *t3*: ⦃ *z1a* : *b1a* | *c1a* ⦄ = ⦃ *z1* : *b1* | *c1* ⦄ **using** *infer-e-uniqueness check-letI* **by** *metis*

have *beq*: *b1a* = *b2* ∧ *b2* = *b1* **using** *check-letI subtype-eq-base t1 t2 t3* **by** *metis*

have Θ ; Φ ; *ℬ* ; *GNil* ; Δ ⊢ *AS-let x1* e_2 *s1* ⇐ *τ1* **proof**
show ‹*atom x1* ♯ (Θ, Φ, *ℬ*, *GNil*, Δ, e_2, *τ1*)› **using** *check-letI t2 fresh-prodN* **by** *metis*
show ‹*atom z2* ♯ (*x1*, Θ, Φ, *ℬ*, *GNil*, Δ, e_2, *τ1*, *s1*)› **using** *check-letI t2* **using** *check-letI t2 fresh-prodN* **by** *metis*
show ‹Θ ; Φ ; *ℬ* ; *GNil* ; Δ ⊢ e_2 ⇒ ⦃ *z2* : *b2* | *c2* ⦄› **using** *check-letI t2* **by** *metis*

have ‹ Θ ; Φ ; *ℬ* ; *GNil*@(*x1*, *b2*, *c2*[*z2*::=[*x1*]^v]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ ⊢ *s1* ⇐ *τ1*›
proof(*rule ctx-subtype-s*)
have *c1a*[*z1a*::=[*x1*]^v]$_{cv}$ = *c1*[*z1*::=[*x1*]^v]$_{cv}$ **using** *subst-v-flip-eq-two subst-v-c-def t3 τ.eq-iff* **by** *metis*
thus ‹ Θ ; Φ ; *ℬ* ; *GNil* @ (*x1*, *b2*, *c1a*[*z1a*::=[*x1*]^v]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ ⊢ *s1* ⇐ *τ1*› **using** *check-letI beq append-g.simps subst-defs* **by** *metis*
show ‹Θ ; *ℬ* ; *GNil* ⊢ ⦃ *z2* : *b2* | *c2* ⦄ ≲ ⦃ *z1a* : *b2* | *c1a* ⦄› **using** *check-letI beq t1 t2* **by** *metis*
have *atom x1* ♯ *c2* **using** *t2 check-letI τ-fresh-c fresh-prodN* **by** *blast*
moreover have *atom x1* ♯ *c1a* **using** *t1 check-letI τ-fresh-c fresh-prodN* **by** *blast*
ultimately show ‹*atom x1* ♯ (*z1a*, *z2*, *c1a*, *c2*)› **using** *t1 t2 fresh-prodN fresh-x-neq* **by** *metis*
qed

thus ‹ Θ ; Φ ; *ℬ* ; (*x1*, *b2*, *c2*[*z2*::=[*x1*]^v]$_v$) $\#_\Gamma$ *GNil* ; Δ ⊢ *s1* ⇐ *τ1*› **using** *append-g.simps subst-defs* **by** *metis*
qed

moreover have *AS-let x1* e_2 *s1* = *AS-let x* e_2 *s* **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *metis*

ultimately show *?case* **by** *metis*

qed(*auto*+)

end

Chapter 15

Basic Type Variable Substitution Lemmas

Lemmas that show that types are preserved, in some way, under basic type variable substitution

lemma *subst-vv-subst-bb-commute*:
  fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v::v$
  assumes *atom bv* ♯ *v*
  shows $(v'[x::=v]_{vv})[bv::=b]_{vb} = (v'[bv::=b]_{vb})[x::=v]_{vv}$
  using *assms* **proof**(*nominal-induct v′ rule*: *v.strong-induct*)
  case (*V-lit x*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
next
  case (*V-var y*)
  hence $v[bv::=b]_{vb}=v$ **using** *forget-subst subst-b-v-def* **by** *metis*
  then show *?case* **unfolding** *subst-vb.simps*(*2*) *subst-vv.simps*(*2*) **using** *V-var* **by** *auto*
next
  case (*V-pair x1a x2a*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
next
  case (*V-cons x1a x2a x3*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
next
  case (*V-consp x1a x2a x3 x4*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
qed

lemma *subst-cev-subst-bb-commute*:
  fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v::v$
  assumes *atom bv* ♯ *v*
  shows $(ce[x::=v]_v)[bv::=b]_{ceb} = (ce[bv::=b]_{ceb})[x::=v]_v$
  using *assms* **apply** (*nominal-induct ce rule*: *ce.strong-induct*, (*simp add*: *subst-vv-subst-bb-commute subst-ceb.simps subst-cv.simps*))
  using *assms subst-vv-subst-bb-commute subst-ceb.simps subst-cv.simps*
  by (*simp add*: *subst-v-ce-def*)+

lemma *subst-cv-subst-bb-commute*:
  fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v::v$

assumes *atom bv* ♯ *v*
shows $c[x::=v]_{cv}[bv::=b]_{cb} = (c[bv::=b]_{cb})[x::=v]_{cv}$
using *assms* **apply** (*nominal-induct c rule*: *c.strong-induct*)
using *assms subst-vv-subst-bb-commute subst-ceb.simps subst-cv.simps subst-v-c-def subst-b-c-def* **apply** *auto*
using *subst-cev-subst-bb-commute subst-v-ce-def* **by** *auto+*

lemma *subst-b-c-of*:
$(c\text{-}of\ \tau\ z)[bv::=b]_{cb} = c\text{-}of\ (\tau[bv::=b]_{\tau b})\ z$
proof(*nominal-induct* τ *avoiding*: *z rule*:τ*.strong-induct*)
case (*T-refined-type z′ b′ c′*)
moreover have *atom bv* ♯ $[\ z\ ]^v$ **using** *fresh-at-base v.fresh* **by** *auto*
ultimately show *?case* **using** *subst-cv-subst-bb-commute*[*of bv V-var z c′ z′ b*] *c-of.simps subst-tb.simps* **by** *metis*
qed

lemma *subst-b-b-of*:
$(b\text{-}of\ \tau)[bv::=b]_{bb} = b\text{-}of\ (\tau[bv::=b]_{\tau b})$
by(*nominal-induct* τ *rule*:τ*.strong-induct*, *simp add*: *b-of.simps subst-tb.simps*)

lemma *subst-b-if*:
$\{\!\!\{\ z : b\text{-}of\ \tau[bv::=b]_{\tau b}\ \mid CE\text{-}val\ (v[bv::=b]_{vb})\ ==\ CE\text{-}val\ (V\text{-}lit\ ll)\ \ IMP\ \ c\text{-}of\ \tau[bv::=b]_{\tau b}\ z\ \}\!\!\} = \{\!\!\{\ z : b\text{-}of\ \tau \mid CE\text{-}val\ (v)\ ==\ CE\text{-}val\ (V\text{-}lit\ ll)\ \ IMP\ \ c\text{-}of\ \tau\ z\ \}\!\!\}[bv::=b]_{\tau b}$
unfolding *subst-tb.simps subst-cb.simps subst-ceb.simps subst-vb.simps* **using** *subst-b-b-of subst-b-c-of* **by** *auto*

lemma *subst-b-top-eq*:
$\{\!\!\{\ z : B\text{-}unit\ \mid TRUE\ \}\!\!\}[bv::=b]_{\tau b} = \{\!\!\{\ z : B\text{-}unit\ \mid TRUE\ \}\!\!\}$ **and** $\{\!\!\{\ z : B\text{-}bool\ \mid TRUE\ \}\!\!\}[bv::=b]_{\tau b} = \{\!\!\{\ z : B\text{-}bool\ \mid TRUE\ \}\!\!\}$ **and** $\{\!\!\{\ z : B\text{-}id\ tid \mid TRUE\ \}\!\!\} = \{\!\!\{\ z : B\text{-}id\ tid \mid TRUE\ \}\!\!\}[bv::=b]_{\tau b}$
unfolding *subst-tb.simps subst-bb.simps subst-cb.simps* **by** *auto*

lemmas *subst-b-eq* = *subst-b-c-of subst-b-b-of subst-b-if subst-b-top-eq*

lemma *subst-cx-subst-bb-commute*[*simp*]:
fixes *bv*::*bv* **and** *b*::*b* **and** *x*::*x* **and** *v′*::*c*
shows $(v'[x::=V\text{-}var\ y]_{cv})[bv::=b]_{cb} = (v'[bv::=b]_{cb})[x::=V\text{-}var\ y]_{cv}$
using *subst-cv-subst-bb-commute fresh-at-base v.fresh* **by** *auto*

lemma *subst-b-infer-b*:
fixes *l*::*l* **and** *b*::*b*
assumes $\vdash l \Rightarrow \tau$ **and** Θ ; {||} $\vdash_{wf}$ *b* **and** *B* = {|*bv*|}
shows $\vdash l \Rightarrow (\tau[bv::=b]_{\tau b})$
using *assms infer-l-form3 infer-l-form4 wf-b-subst infer-l-wf subst-tb.simps base-for-lit.simps subst-tb.simps subst-b-base-for-lit subst-cb.simps*(*6*) *subst-ceb.simps*(*1*) *subst-vb.simps*(*1*) *subst-vb.simps*(*2*) *type-l-eq*
by *metis*

lemma *subst-b-subtype*:
fixes *s*::*s* **and** *b′*::*b*
assumes Θ ; {|*bv*|} ; Γ $\vdash \tau 1 \lesssim \tau 2$ **and** Θ ; {||} $\vdash_{wf}$ *b′* **and** *B* = {|*bv*|}
shows Θ ; {||} ; $\Gamma[bv::=b']_{\Gamma b} \vdash \tau 1[bv::=b']_{\tau b} \lesssim \tau 2[bv::=b']_{\tau b}$
using *assms* **proof**(*nominal-induct* {|*bv*|} Γ *τ1 τ2 rule*:*subtype.strong-induct*)
case (*subtype-baseI x* Θ Γ *z c z′ c′ b*)

hence ∗∗: Θ ; $\{|bv|\}$; $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})$ $\#_{\Gamma}$ Γ $\models$ $c'[z'::=V\text{-}var\ x]_{cv}$ **using** *validI subst-defs* **by** *metis*

have Θ ; $\{||\}$; $\Gamma[bv::=b']_{\Gamma b}$ $\vdash$ $\{\!\mid z : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} \mid\!\}$ $\lesssim$ $\{\!\mid z' : b[bv::=b']_{bb} \mid c'[bv::=b']_{cb} \mid\!\}$
proof(*rule Typing.subtype-baseI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b']_{\Gamma b}$ $\vdash_{wf}$ $\{\!\mid z : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} \mid\!\}$
using *subtype-baseI assms wf-b-subst(4) subst-tb.simps subst-defs* **by** *metis*
show Θ ; $\{||\}$; $\Gamma[bv::=b']_{\Gamma b}$ $\vdash_{wf}$ $\{\!\mid z' : b[bv::=b']_{bb} \mid c'[bv::=b']_{cb} \mid\!\}$
using *subtype-baseI assms wf-b-subst(4) subst-tb.simps* **by** *metis*
show *atom x* $\sharp(\Theta,\ \{||\}::bv\ fset,\ \Gamma[bv::=b']_{\Gamma b},\ z\ ,\ c[bv::=b']_{cb}\ ,\ z'\ ,\ c'[bv::=b']_{cb}\ )$
apply(*unfold fresh-prodN,auto simp add*: ∗ *fresh-prodN fresh-empty-fset*)
using *subst-b-fresh-x* ∗ *fresh-prodN* ‹*atom x* ♯ *c*› ‹*atom x* ♯ *c*′› *subst-defs subtype-baseI* **by** *metis+*
have Θ ; $\{||\}$; $(x,\ b[bv::=b']_{bb},\ c[z::=V\text{-}var\ x]_{v}[bv::=b']_{cb})$ $\#_{\Gamma}$ $\Gamma[bv::=b']_{\Gamma b}$ $\models$ $c'[z'::=V\text{-}var\ x]_{v}[bv::=b']_{cb}$
using ∗∗ *subst-b-valid subst-gb.simps assms subtype-baseI* **by** *metis*
thus Θ ; $\{||\}$; $(x,\ b[bv::=b']_{bb},\ (c[bv::=b']_{cb})[z::=V\text{-}var\ x]_{v})$ $\#_{\Gamma}$ $\Gamma[bv::=b']_{\Gamma b}$ $\models$ $(c'[bv::=b']_{cb})[z'::=V\text{-}var\ x]_{v}$
using *subst-defs subst-cv-subst-bb-commute* **by** (*metis subst-cx-subst-bb-commute*)
qed
thus *?case* **using** *subtype-baseI subst-tb.simps subst-defs* **by** *metis*
qed

lemma *b-of-subst-bv*:
$(b\text{-}of\ \tau)[x::=v]_{bb} = b\text{-}of\ (\tau[x::=v]_{\tau b})$
proof −
obtain *z b c* **where** ∗:$\tau = \{\!\mid z : b \mid c \mid\!\} \wedge atom\ z\ \sharp\ (x,v)$ **using** *obtain-fresh-z* **by** *metis*
thus *?thesis* **using** *subst-tv.simps* ∗ **by** *auto*
qed

lemma *subst-b-infer-v*:
fixes *v*::*v* **and** *b*::*b*
assumes Θ ; B ; $G \vdash v \Rightarrow \tau$ **and** Θ ; $\{||\}$ $\vdash_{wf}$ b **and** $B = \{|bv|\}$
shows Θ ; $\{||\}$; $G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow (\tau[bv::=b]_{\tau b})$
using *assms* **proof**(*nominal-induct avoiding*: *b bv rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ Γ b' *c x z*)
show *?case* **unfolding** *subst-b-simps* **proof**
show Θ ; $\{||\}$ $\vdash_{wf}$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *infer-v-varI wf-b-subst* **by** *metis*
show *Some* $(b'[bv::=b]_{bb},\ c[bv::=b]_{cb}) = lookup\ \Gamma[bv::=b]_{\Gamma b}\ x$ **using** *subst-b-lookup infer-v-varI* **by** *metis*
show *atom z* ♯ *x* **using** *infer-v-varI* **by** *auto*
show *atom z* ♯ $(\Theta,\ \{||\},\ \Gamma[bv::=b]_{\Gamma b})$ **by**(*fresh-mth add*: *infer-v-varI subst-b-fresh-x subst-b-*Γ*-def fresh-prodN fresh-empty-fset*)
qed
next
case (*infer-v-litI* Θ $\mathcal{B}$ Γ *l* τ)
then show *?case* **using** *Typing.infer-v-litI subst-b-infer-b*
using *wf-b-subst1(3)* **by** *auto*
next
case (*infer-v-pairI z v1 v2* Θ $\mathcal{B}$ Γ *t1 t2*)
show *?case* **unfolding** *subst-b-simps b-of-subst-bv* **proof**
show *atom z* ♯ $(v1[bv::=b]_{vb},\ v2[bv::=b]_{vb})$ **by**(*fresh-mth add*: *infer-v-pairI subst-b-fresh-x*)

show *atom z* ♯ (Θ, {||}, $\Gamma[bv::=b]_{\Gamma b}$) **by**(*fresh-mth add*: *infer-v-pairI subst-b-fresh-x subst-b-Γ-def fresh-empty-fset*)
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow t1[bv::=b]_{\tau b}$ **using** *infer-v-pairI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow t2[bv::=b]_{\tau b}$ **using** *infer-v-pairI* **by** *auto*
qed
next
case (*infer-v-consI s dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv z*)
show *?case* **unfolding** *subst-b-simps b-of-subst-bv* **proof**
show *AF-typedef s dclist* ∈ *set* Θ **using** *infer-v-consI* **by** *auto*
show (*dc*, *tc*) ∈ *set dclist* **using** *infer-v-consI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow tv[bv::=b]_{\tau b}$ **using** *infer-v-consI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc$ **proof** −
have *atom bv* ♯ *tc* **using** *wfTh-lookup-supp-empty fresh-def infer-v-consI infer-v-wf* **by** *fast*
moreover have Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv::=b]_{\tau b}$
using *subst-b-subtype infer-v-consI* **by** *simp*
ultimately show *?thesis* **using** *forget-subst subst-b-τ-def* **by** *metis*
qed
show *atom z* ♯ $v[bv::=b]_{vb}$ **using** *infer-v-consI* **using** *subst-b-fresh-x subst-b-v-def* **by** *metis*
show *atom z* ♯ (Θ, {||}, $\Gamma[bv::=b]_{\Gamma b}$) **by**(*fresh-mth add*: *infer-v-consI subst-b-fresh-x subst-b-Γ-def fresh-empty-fset*)
qed
next
case (*infer-v-conspI s bv2 dclist2* Θ *dc tc* $\mathcal{B}$ Γ *v tv ba z*)

have Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash$ *V-consp s dc* ($ba[bv::=b]_{bb}$) ($v[bv::=b]_{vb}$) ⇒ ⦃ *z* : *B-app s* ($ba[bv::=b]_{bb}$) | [[*z*]^{v}]^{ce} == [*V-consp s dc* ($ba[bv::=b]_{bb}$) ($v[bv::=b]_{vb}$)]^{ce} ⦄
proof(*rule Typing.infer-v-conspI*)
show *AF-typedef-poly s bv2 dclist2* ∈ *set* Θ **using** *infer-v-conspI* **by** *auto*
show (*dc*, *tc*) ∈ *set dclist2* **using** *infer-v-conspI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow tv[bv::=b]_{\tau b}$
using *infer-v-conspI subst-tb.simps* **by** *metis*

show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv2::=ba[bv::=b]_{bb}]_{\tau b}$ **proof** −
have *supp tc* ⊆ { *atom bv2* } **using** *infer-v-conspI wfTh-poly-lookup-supp wfX-wfY* **by** *metis*
moreover have *bv2* ≠ *bv* **using** ‹*atom bv2* ♯ $\mathcal{B}$› ‹$\mathcal{B}$ = {|*bv*|} › *fresh-at-base fresh-def*
using *fresh-finsert* **by** *fastforce*
ultimately have *atom bv* ♯ *tc* **unfolding** *fresh-def* **by** *auto*
hence $tc[bv2::=ba[bv::=b]_{bb}]_{\tau b} = tc[bv2::=ba]_{\tau b}[bv::=b]_{\tau b}$
using *subst-tb-commute* **by** *metis*
moreover have Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv2::=ba]_{\tau b}[bv::=b]_{\tau b}$
using *infer-v-conspI*(*7*) *subst-b-subtype infer-v-conspI* **by** *metis*
ultimately show *?thesis* **by** *auto*
qed
show *atom z* ♯ (Θ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $v[bv::=b]_{vb}$, $ba[bv::=b]_{bb}$)
apply(*unfold fresh-prodN*, *intro conjI*, *auto simp add*: *infer-v-conspI fresh-empty-fset*)
using ‹*atom z* ♯ Γ› *fresh-subst-if subst-b-Γ-def x-fresh-b* **apply** *metis*
using ‹*atom z* ♯ *v*› *fresh-subst-if subst-b-v-def x-fresh-b* **by** *metis*
show *atom bv2* ♯ (Θ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $v[bv::=b]_{vb}$, $ba[bv::=b]_{bb}$)
apply(*unfold fresh-prodN*, *intro conjI*, *auto simp add*: *infer-v-conspI fresh-empty-fset*)
using ‹*atom bv2* ♯ *b*› ‹*atom bv2* ♯ Γ› *fresh-subst-if subst-b-Γ-def* **apply** *metis*
using ‹*atom bv2* ♯ *b*› ‹*atom bv2* ♯ *v*› *fresh-subst-if subst-b-v-def* **apply** *metis*
using ‹*atom bv2* ♯ *b*› ‹*atom bv2* ♯ *ba*› *fresh-subst-if subst-b-b-def* **by** *metis*

show Θ ; {||} $\vdash_{wf}$ *ba*[*bv*::=*b*]$_{bb}$
using *infer-v-conspI wf-b-subst* **by** *metis*
qed
thus *?case* **using** *subst-vb.simps subst-tb.simps subst-bb.simps* **by** *simp*

qed

lemma *subst-b-check-v*:
fixes *v*::*v* **and** *b*::*b*
assumes Θ ; *B* ; *G* ⊢ *v* ⇐ τ **and** Θ ; {||} $\vdash_{wf}$ *b* **and** *B* = {|*bv*|}
shows Θ ; {||} ; *G*[*bv*::=*b*]$_{\Gamma b}$ ⊢ *v*[*bv*::=*b*]$_{vb}$ ⇐ (τ[*bv*::=*b*]$_{\tau b}$)
proof −
obtain τ′ **where** Θ ; *B* ; *G* ⊢ *v* ⇒ τ′ ∧ Θ ; *B* ; *G* ⊢ τ′ ≲ τ **using** *check-v-elims*[*OF assms*(*1*)] **by** *metis*
thus *?thesis* **using** *subst-b-subtype subst-b-infer-v assms*
by (*metis* (*no-types*) *check-v-subtypeI subst-b-infer-v subst-b-subtype*)
qed

lemma *subst-vv-subst-vb-switch*:
shows (*v*′[*bv*::=*b*′]$_{vb}$)[*x*::=*v*[*bv*::=*b*′]$_{vb}$]$_{vv}$ = *v*′[*x*::=*v*]$_{vv}$[*bv*::=*b*′]$_{vb}$
proof(*nominal-induct v*′ *rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *subst-vv.simps subst-vb.simps* **by** *auto*
next
case (*V-var x*)
then show *?case* **using** *subst-vv.simps subst-vb.simps* **by** *auto*
next
case (*V-pair x1a x2a*)
then show *?case* **using** *subst-vv.simps subst-vb.simps v.fresh* **by** *auto*
next
case (*V-cons x1a x2a x3*)
then show *?case* **using** *subst-vv.simps subst-vb.simps v.fresh* **by** *auto*
next
case (*V-consp x1a x2a x3 x4*)
then show *?case* **using** *subst-vv.simps subst-vb.simps v.fresh pure-fresh*
by (*metis forget-subst subst-b-b-def*)
qed

lemma *subst-cev-subst-vb-switch*:
shows (*ce*[*bv*::=*b*′]$_{ceb}$)[*x*::=*v*[*bv*::=*b*′]$_{vb}$]$_{cev}$ = (*ce*[*x*::=*v*]$_{cev}$)[*bv*::=*b*′]$_{ceb}$
by(*nominal-induct ce rule*:*ce.strong-induct*, *auto simp add*: *subst-vv-subst-vb-switch ce.fresh*)

lemma *subst-cv-subst-vb-switch*:
shows (*c*[*bv*::=*b*′]$_{cb}$)[*x*::=*v*[*bv*::=*b*′]$_{vb}$]$_{cv}$ = *c*[*x*::=*v*]$_{cv}$[*bv*::=*b*′]$_{cb}$
by(*nominal-induct c rule*:*c.strong-induct*, *auto simp add*: *subst-cev-subst-vb-switch c.fresh*)

lemma *subst-tv-subst-vb-switch*:
shows (τ[*bv*::=*b*′]$_{\tau b}$)[*x*::=*v*[*bv*::=*b*′]$_{vb}$]$_{\tau v}$ = τ[*x*::=*v*]$_{\tau v}$[*bv*::=*b*′]$_{\tau b}$
proof(*nominal-induct* τ *avoiding*: *x v rule*:τ*.strong-induct*)
case (*T-refined-type z b c*)
hence *ceq*: (*c*[*bv*::=*b*′]$_{cb}$)[*x*::=*v*[*bv*::=*b*′]$_{vb}$]$_{cv}$ = *c*[*x*::=*v*]$_{cv}$[*bv*::=*b*′]$_{cb}$ **using** *subst-cv-subst-vb-switch* **by** *auto*

moreover have *atom z* ♯ $v[bv::=b']_{vb}$ **using** *x-fresh-b fresh-subst-if subst-b-v-def T-refined-type* **by** *metis*

hence $\{\!|\ z : b\ \mid c\ |\!\}[bv::=b']_{\tau b}[x::=v[bv::=b']_{vb}]_{\tau v} = \{\!|\ z : b[bv::=b']_{bb}\ \mid (c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv}\ |\!\}$
using *subst-tv.simps subst-tb.simps T-refined-type fresh-Pair* **by** *metis*

moreover have $\{\!|\ z : b[bv::=b']_{bb}\ \mid (c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv}\ |\!\} = \{\!|\ z : b\ \mid c[x::=v]_{cv}\ |\!\}[bv::=b']_{\tau b}$
using *subst-tv.simps subst-tb.simps ceq τ.fresh forget-subst[of bv b b'] subst-b-b-def T-refined-type* **by** *metis*

ultimately show *?case* **using** *subst-tv.simps subst-tb.simps ceq T-refined-type* **by** *auto*
qed

lemma *subst-tb-triple*:
assumes *atom bv* ♯ τ'
shows $\tau'[bv'::=b'[bv::=b]_{bb}]_{\tau b}[x'::=v'[bv::=b]_{vb}]_{\tau v} = \tau'[bv'::=b']_{\tau b}[x'::=v']_{\tau v}[bv::=b]_{\tau b}$
proof −
have $\tau'[bv'::=b'[bv::=b]_{bb}]_{\tau b}[x'::=v'[bv::=b]_{vb}]_{\tau v} = \tau'[bv'::=b']_{\tau b}[bv::=b]_{\tau b}\ [x'::=v'[bv::=b]_{vb}]_{\tau v}$
using *subst-tb-commute* ‹*atom bv* ♯ τ'› **by** *auto*
also have ... $= \tau'[bv'::=b']_{\tau b}\ [x'::=v']_{\tau v}[bv::=b]_{\tau b}$
using *subst-tv-subst-vb-switch* **by** *auto*
finally show *?thesis* **using** *fresh-subst-if forget-subst* **by** *auto*
qed

lemma *subst-b-infer-e*:
fixes *s::s* **and** *b::b*
assumes Θ ; Φ ; B ; G; $D \vdash e \Rightarrow \tau$ **and** Θ ; $\{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
shows Θ ; Φ ; $\{||\}$; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b} \vdash (e[bv::=b]_{eb}) \Rightarrow (\tau[bv::=b]_{\tau b})$
using *assms* **proof**(*nominal-induct avoiding*: *b rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ *ℬ* Γ Δ Φ *v* τ)
thus *?case* **using** *subst-eb.simps infer-e.intros wf-b-subst subst-db.simps wf-b-subst infer-v-wf subst-b-infer-v*

by (*metis forget-subst ms-fresh-all*(*1*) *wfV-b-fresh*)
next
case (*infer-e-plusI* Θ *ℬ* Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

show *?case* **unfolding** *subst-b-simps subst-eb.simps* **proof**(*rule Typing.infer-e-plusI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-plusI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-plusI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow \{\!|\ z1 : B\text{-}int\ \mid c1[bv::=b]_{cb}\ |\!\}$ **using** *subst-b-infer-v infer-e-plusI subst-b-simps* **by** *force*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow \{\!|\ z2 : B\text{-}int\ \mid c2[bv::=b]_{cb}\ |\!\}$ **using** *subst-b-infer-v infer-e-plusI subst-b-simps* **by** *force*
show *atom z3* ♯ *AE-op Plus* ($v1[bv::=b]_{vb}$) ($v2[bv::=b]_{vb}$) **using** *subst-b-simps infer-e-plusI subst-b-fresh-x subst-b-e-def* **by** *metis*
show *atom z3* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-plusI* **by** *auto*
qed

next
  case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
  show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-leqI*)
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-leqI wfX-wfY*
      by (*metis wf-b-subst*(*15*))
    show Θ $\vdash_{wf}$ Φ **using** *infer-e-leqI* **by** *auto*
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow$ ⦃ *z1* : *B-int* | $c1[bv::=b]_{cb}$ ⦄ **using** *subst-b-infer-v infer-e-leqI subst-b-simps* **by** *force*
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow$ ⦃ *z2* : *B-int* | $c2[bv::=b]_{cb}$ ⦄ **using** *subst-b-infer-v infer-e-leqI subst-b-simps* **by** *force*
    show *atom z3* ♯ *AE-op LEq* ($v1[bv::=b]_{vb}$) ($v2[bv::=b]_{vb}$) **using** *subst-b-simps infer-e-leqI subst-b-fresh-x subst-b-e-def* **by** *metis*
    show *atom z3* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-leqI* **by** *auto*
  qed
next
  case (*infer-e-eqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 bb c1 v2 z2 c2 z3*)
  show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-eqI*)
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-eqI wfX-wfY*
      by (*metis wf-b-subst*(*15*))
    show Θ $\vdash_{wf}$ Φ **using** *infer-e-eqI* **by** *auto*
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow$ ⦃ *z1* : $bb[bv::=b]_{bb}$ | $c1[bv::=b]_{cb}$ ⦄ **using** *subst-b-infer-v infer-e-eqI subst-b-simps* **by** *force*
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow$ ⦃ *z2* : $bb[bv::=b]_{bb}$ | $c2[bv::=b]_{cb}$ ⦄ **using** *subst-b-infer-v infer-e-eqI subst-b-simps* **by** *force*
    show *atom z3* ♯ *AE-op Eq* ($v1[bv::=b]_{vb}$) ($v2[bv::=b]_{vb}$) **using** *subst-b-simps infer-e-eqI subst-b-fresh-x subst-b-e-def* **by** *metis*
    show *atom z3* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-eqI* **by** *auto*
    show $bb[bv::=b]_{bb}$ ∈ {*B-bool*, *B-int*, *B-unit*} **using** *infer-e-eqI* **by** *auto*
  qed
next
  case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ *f x b′ c τ′ s′ v τ*)
  show *?case* **proof**(*subst subst-eb.simps*, *rule Typing.infer-e-appI*)
    show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-appI wfX-wfY* **by** (*metis wf-b-subst*(*15*))
    show Θ $\vdash_{wf}$ Φ **using** *infer-e-appI* **by** *auto*
    show *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b′ c τ′ s′*))) = *lookup-fun* Φ *f* **using** *infer-e-appI* **by** *auto*

    have *atom bv* ♯ *b′* **using** ‹Θ $\vdash_{wf}$ Φ › *infer-e-appI wfPhi-f-supp fresh-def*[*of atom bv b′*] **by** *simp*
    hence $b' = b'[bv::=b]_{bb}$ **using** *subst-b-simps*
      using *has-subst-b-class.forget-subst subst-b-b-def* **by** *force*
    moreover have *ceq*:$c = c[bv::=b]_{cb}$ **using** *subst-b-simps* **proof** −
      have *supp c* ⊆ {*atom x*} **using** *infer-e-appI wfPhi-f-simple-supp-c*[*OF* - ‹Θ $\vdash_{wf}$ Φ ›] **by** *simp*
      hence *atom bv* ♯ *c* **using**
          fresh-def[*of atom bv c*]
        using *fresh-def fresh-finsert insert-absorb*
          insert-subset ms-fresh-all supp-at-base x-not-in-b-set fresh-prodN **by** *metis*
      thus *?thesis*
        using *forget-subst subst-b-c-def fresh-def*[*of atom bv c*] **by** *metis*
    qed

show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $v[bv::=b]_{vb}$ $\Leftarrow$ ⦃ x : b' | c ⦄
using *subst-b-check-v subst-tb.simps subst-vb.simps infer-e-appI*
proof −
have Θ ; {|bv|} ; $\Gamma \vdash v \Leftarrow$ ⦃ x : b' | c ⦄
by (*metis* ‹$\mathcal{B}$ = {|bv|}› ‹Θ ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow$ ⦃ x : b' | c ⦄›)
then show *?thesis*
by (*metis* (*no-types*) ‹Θ ; {||} $\vdash_{wf}$ b› ‹$b' = b'[bv::=b]_{bb}$› *subst-b-check-v subst-tb.simps ceq*)
qed
show *atom x* ♯ (Θ, Φ, {||}::*bv fset*, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $v[bv::=b]_{vb}$, $\tau[bv::=b]_{\tau b}$)
apply (*fresh-mth add*: *fresh-prodN subst-g-b-x-fresh infer-e-appI*)
using *subst-b-fresh-x infer-e-appI* **apply** *metis+*
done
have *supp* $\tau' \subseteq$ { *atom x* } **using** *wfPhi-f-simple-supp-t infer-e-appI* **by** *auto*
hence *atom bv* ♯ τ' **using** *fresh-def fresh-at-base* **by** *force*
then show $\tau'[x::=v[bv::=b]_{vb}]_v = \tau[bv::=b]_{\tau b}$ **using** *infer-e-appI*
forget-subst subst-b-τ-def subst-tv-subst-vb-switch subst-defs **by** *metis*
qed
next
case (*infer-e-appPI* Θ' $\mathcal{B}$ Γ' Δ Φ' b' f' bv' x' *b1* c τ' s' v' $\tau 1$)

have *beq*: $b1[bv'::=b']_{bb}[bv::=b]_{bb} = b1[bv'::=b'[bv::=b]_{bb}]_{bb}$
proof −
have *supp b1* $\subseteq$ { *atom bv'* } **using** *wfPhi-f-poly-supp-b infer-e-appPI*
using *supp-at-base* **by** *blast*
moreover have $bv \neq bv'$ **using** *infer-e-appPI fresh-def supp-at-base*
by (*simp add*: *fresh-def supp-at-base*)
ultimately have *atom bv* ♯ *b1* **using** *fresh-def fresh-at-base* **by** *force*
thus *?thesis* **by** *simp*
qed

have *ceq*: $(c[bv'::=b']_{cb})[bv::=b]_{cb} = c[bv'::=b'[bv::=b]_{bb}]_{cb}$ **proof** −
have *supp c* $\subseteq$ { *atom bv'*, *atom x'* } **using** *wfPhi-f-poly-supp-c infer-e-appPI*
using *supp-at-base* **by** *blast*
moreover have $bv \neq bv'$ **using** *infer-e-appPI fresh-def supp-at-base*
by (*simp add*: *fresh-def supp-at-base*)
moreover have *atom x'* $\neq$ *atom bv* **by** *auto*
ultimately have *atom bv* ♯ c **using** *fresh-def*[*of atom bv c*] *fresh-at-base* **by** *auto*
thus *?thesis* **by** *simp*
qed

show *?case* **proof**(*subst subst-eb.simps*, *rule Typing.infer-e-appPI*)
show Θ' ; {||} ; $\Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst subst-db.simps infer-e-appPI wfX-wfY*
by *metis*
show $\Theta' \vdash_{wf} \Phi'$ **using** *infer-e-appPI* **by** *auto*
show *Some* (*AF-fundef* f' (*AF-fun-typ-some* bv' (*AF-fun-typ* x' *b1* c τ' s'))) = *lookup-fun* Φ' f'
using *infer-e-appPI* **by** *auto*
thus Θ' ; {||} ; $\Gamma'[bv::=b]_{\Gamma b} \vdash v'[bv::=b]_{vb} \Leftarrow$ ⦃ x' : $b1[bv'::=b'[bv::=b]_{bb}]_b$ | $c[bv'::=b'[bv::=b]_{bb}]_b$
⦄
using *subst-b-check-v subst-tb.simps subst-b-simps infer-e-appPI*
proof −
have Θ' ; {||} ; $\Gamma'[bv::=b]_{\Gamma b} \vdash v'[bv::=b]_{vb} \Leftarrow$ ⦃ x' : $b1[bv'::=b']_b[bv::=b]_{bb}$ | $(c[bv'::=b']_b)[bv::=b]_{cb}$
⦄

using *infer-e-appPI subst-b-check-v subst-tb.simps* **by** *metis*
thus *?thesis* **using** *beq ceq subst-defs* **by** *metis*
qed
show *atom* $x' \sharp \Gamma'[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-appPI* **by** *auto*
show $\tau'[bv'::=b'[bv::=b]_{bb}]_b[x'::=v'[bv::=b]_{vb}]_v = \tau 1[bv::=b]_{\tau b}$ **proof** −

have *supp* $\tau' \subseteq \{$ *atom* x', *atom* bv' $\}$ **using** *wfPhi-f-poly-supp-t infer-e-appPI* **by** *auto*
moreover hence $bv \neq bv'$ **using** *infer-e-appPI fresh-def supp-at-base*
by (*simp add*: *fresh-def supp-at-base*)
ultimately have *atom* $bv \sharp \tau'$ **using** *fresh-def* **by** *force*
hence $\tau'[bv'::=b'[bv::=b]_{bb}]_b[x'::=v'[bv::=b]_{vb}]_v = \tau'[bv'::=b']_b[x'::=v']_v[bv::=b]_{\tau b}$ **using** *subst-tb-triple subst-defs* **by** *auto*
thus *?thesis* **using** *infer-e-appPI* **by** *metis*
qed
show *atom* $bv' \sharp (\Theta', \Phi', \{||\}, \Gamma'[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, b'[bv::=b]_{bb}, v'[bv::=b]_{vb}, \tau 1[bv::=b]_{\tau b})$
unfolding *fresh-prodN* **apply**(*auto simp add*: *infer-e-appPI fresh-empty-fset*)
using *fresh-subst-if subst-b-Γ-def subst-b-Δ-def subst-b-b-def subst-b-v-def subst-b-τ-def infer-e-appPI* **by** *metis+*
show Θ' ; $\{||\}$ $\vdash_{wf}$ $b'[bv::=b]_{bb}$ **using** *infer-e-appPI wf-b-subst* **by** *simp*
qed
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ Φ v z' *b1 b2 c z*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-fstI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-fstI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-fstI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : B\text{-}pair\ b1[bv::=b]_{bb}\ b2[bv::=b]_{bb}\ \mid c[bv::=b]_{cb}\ |\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-fstI* **by** *force*
show *atom* $z \sharp$ *AE-fst* $(v[bv::=b]_{vb})$ **using** *infer-e-fstI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom* $z \sharp \Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-fstI* **by** *auto*
qed
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ Φ v z' *b1 b2 c z*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-sndI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-sndI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-sndI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : B\text{-}pair\ b1[bv::=b]_{bb}\ b2[bv::=b]_{bb}\ \mid c[bv::=b]_{cb}\ |\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-sndI* **by** *force*
show *atom* $z \sharp$ *AE-snd* $(v[bv::=b]_{vb})$ **using** *infer-e-sndI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom* $z \sharp \Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-sndI* **by** *auto*
qed
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ Δ Φ v z' *c z*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-lenI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-lenI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-lenI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : B\text{-}bitvec\ \mid c[bv::=b]_{cb}\ |\!\}$

using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-lenI* **by** *force*
show *atom z* ♯ *AE-len* $(v[bv::=b]_{vb})$ **using** *infer-e-lenI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom z* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-lenI* **by** *auto*
qed
next
case (*infer-e-mvarI* Θ $\mathcal{B}$ Γ Φ Δ *u* τ)
show *?case* **proof**(*subst subst subst-eb.simps*, *rule Typing.infer-e-mvarI*)
show Θ ; $\{||\}$ $\vdash_{wf}$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *infer-e-mvarI wf-b-subst* **by** *auto*
show Θ $\vdash_{wf}$ Φ **using** *infer-e-mvarI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *infer-e-mvarI* **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-sndI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show (*u*, $\tau[bv::=b]_{\tau b}$) $\in$ *setD* $\Delta[bv::=b]_{\Delta b}$ **using** *infer-e-mvarI subst-db.simps set-insert subst-d-b-member* **by** *simp*
qed
next
case (*infer-e-concatI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-concatI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-concatI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show Θ $\vdash_{wf}$ Φ **using** *infer-e-concatI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v1*$[bv::=b]_{vb}$ $\Rightarrow$ ⦃ *z1* : *B-bitvec* | *c1*$[bv::=b]_{cb}$ ⦄
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-concatI* **by** *force*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v2*$[bv::=b]_{vb}$ $\Rightarrow$ ⦃ *z2* : *B-bitvec* | *c2*$[bv::=b]_{cb}$ ⦄
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-concatI* **by** *force*
show *atom z3* ♯ *AE-concat* (*v1*$[bv::=b]_{vb}$) (*v2*$[bv::=b]_{vb}$) **using** *infer-e-concatI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom z3* ♯ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-concatI* **by** *auto*
qed
next
case (*infer-e-splitI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-splitI*)
show ‹ Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ › **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-splitI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-splitI* **by** *auto*
show ‹ Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v1*$[bv::=b]_{vb}$ $\Rightarrow$ ⦃ *z1* : *B-bitvec* | *c1*$[bv::=b]_{cb}$ ⦄›
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-splitI* **by** *force*
show ‹Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v2*$[bv::=b]_{vb}$ $\Leftarrow$ ⦃ *z2* : *B-int* | [*leq* [[*L-num 0* $]^{v}$ $]^{ce}$ [[*z2* $]^{v}$ $]^{ce}$ $]^{ce}$ == [[*L-true* $]^{v}$ $]^{ce}$ *AND*
[*leq* [[*z2* $]^{v}$ $]^{ce}$ [| [*v1*$[bv::=b]_{vb}$ $]^{ce}$ |$]^{ce}$ $]^{ce}$ == [[*L-true* $]^{v}$ $]^{ce}$ ⦄›
using *subst-b-check-v subst-tb.simps subst-b-simps infer-e-splitI*
proof −
have Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v2*$[bv::=b]_{vb}$ $\Leftarrow$ ⦃ *z2* : *B-int* | [*leq* [[*L-num 0* $]^{v}$ $]^{ce}$ [[*z2* $]^{v}$ $]^{ce}$ $]^{ce}$ == [[*L-true* $]^{v}$ $]^{ce}$ *AND* [*leq* [[*z2* $]^{v}$ $]^{ce}$ [| [*v1* $]^{ce}$ |$]^{ce}$ $]^{ce}$ == [[*L-true* $]^{v}$ $]^{ce}$ ⦄$[bv::=b]_{\tau b}$
using *infer-e-splitI.hyps*(*7*) *infer-e-splitI.prems*(*1*) *infer-e-splitI.prems*(*2*) *subst-b-check-v* **by** *presburger*
then show *?thesis*
by *simp*
qed
show ‹*atom z1* ♯ *AE-split* (*v1*$[bv::=b]_{vb}$) (*v2*$[bv::=b]_{vb}$)› **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def*

e.fresh **by** *metis*
  show ‹*atom z1* ♯ $\Gamma[bv::=b]_{\Gamma b}$› **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*

 show ‹*atom z2* ♯ *AE-split* $(v1[bv::=b]_{vb})$ $(v2[bv::=b]_{vb})$› **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*

  show ‹*atom z2* ♯ $\Gamma[bv::=b]_{\Gamma b}$› **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*
 show ‹*atom z3* ♯ *AE-split* $(v1[bv::=b]_{vb})$ $(v2[bv::=b]_{vb})$› **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
  show ‹*atom z3* ♯ $\Gamma[bv::=b]_{\Gamma b}$› **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*
 qed
qed

This is needed for preservation. When we apply a function "f [b] v" we need to substitute into the body of the function f replacing type-variable with b

lemma *subst-b-c-of-forget*:
 assumes *atom bv* ♯ *const*
 shows $(c\text{-}of\ const\ x)[bv::=b]_{cb} = c\text{-}of\ const\ x$
 using *assms* **proof**(*nominal-induct const avoiding*: *x rule*:*τ.strong-induct*)
 case (*T-refined-type x′ b′ c′*)
 hence *c-of* ⦃ *x′* : *b′* | *c′* ⦄ *x* = $c'[x'::=V\text{-}var\ x]_{cv}$ **using** *c-of.simps* **by** *metis*
 moreover have *atom bv* ♯ $c'[x'::=V\text{-}var\ x]_{cv}$ **proof** −
  have *atom bv* ♯ *c′* **using** *T-refined-type τ.fresh* **by** *simp*
  moreover have *atom bv* ♯ *V-var x* **using** *v.fresh* **by** *simp*
  ultimately show *?thesis*
   using *T-refined-type τ.fresh subst-b-c-def fresh-subst-if*
    τ-fresh-c fresh-subst-cv-if has-subst-b-class.subst-b-fresh-x ms-fresh-all(*37*) *ms-fresh-all assms* **by** *metis*
 qed
 ultimately show *?case* **using** *forget-subst subst-b-c-def* **by** *metis*
qed

lemma *subst-b-check-s*:
 fixes *s*::*s* **and** *b*::*b* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v* **and** *τ*::*τ*
 assumes Θ ; {||} $\vdash_{wf}$ *b* **and** *B* = {|*bv*|}
 shows Θ ; Φ ; *B* ; *G*; *D* ⊢ *s* ⇐ *τ* ⟹ Θ ; Φ ; {||} ; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b}$ ⊢ $(s[bv::=b]_{sb})$ ⇐ $(\tau[bv::=b]_{\tau b})$ **and**
  Θ ; Φ ; *B* ; *G*; *D* ; *tid* ; *cons* ; *const* ; *v* ⊢ *cs* ⇐ *τ* ⟹ Θ ; Φ ; {||} ; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b}$; *tid* ; *cons* ; *const* ; $v[bv::=b]_{vb}$ ⊢ (*subst-branchb cs bv b*) ⇐ $(\tau[bv::=b]_{\tau b})$ **and**
  Θ ; Φ ; *B* ; *G*; *D* ; *tid* ; *dclist* ; *v* ⊢ *css* ⇐ *τ* ⟹ Θ ; Φ ; {||} ; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b}$; *tid* ; *dclist* ; $v[bv::=b]_{vb}$ ⊢ (*subst-branchlb css bv b*) ⇐ $(\tau[bv::=b]_{\tau b})$
 using *assms* **proof**(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
 note *facts* = *wfD-emptyI wfX-wfY wf-b-subst subst-b-subtype subst-b-infer-v*
 case (*check-valI* Θ 𝓑 Γ Δ Φ *v τ′ τ*)
 show *?case*
  apply(*subst subst-sb.simps*, *rule Typing.check-valI*)
  using *facts check-valI* **apply** *metis*
  using *check-valI subst-b-infer-v wf-b-subst subst-b-subtype* **apply** *blast*
  using *check-valI subst-b-infer-v wf-b-subst subst-b-subtype* **apply** *blast*
  using *check-valI subst-b-infer-v wf-b-subst subst-b-subtype* **by** *metis*
next
 case (*check-letI x* Θ Φ 𝓑 Γ Δ *e τ z s b′ c*)

show *?case* **proof**(*subst subst-sb.simps*, *rule Typing.check-letI*)

show *atom x* $\sharp$(Θ, Φ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $e[bv::=b]_{eb}$, $\tau[bv::=b]_{\tau b}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-letI fresh-empty-fset*)+
apply(*metis* ∗ *subst-b-fresh-x check-letI fresh-prodN*)+ **done**
show *atom z* $\sharp$ (*x*, Θ, Φ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $e[bv::=b]_{eb}$, $\tau[bv::=b]_{\tau b}$, $s[bv::=b]_{sb}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-letI fresh-empty-fset*)+
apply(*metis* ∗ *subst-b-fresh-x check-letI fresh-prodN*)+ **done**
show Θ ; Φ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ ⊢ $e[bv::=b]_{eb}$ ⇒ ⦃ *z* : $b'[bv::=b]_{bb}$ | $c[bv::=b]_{cb}$ ⦄
using *check-letI subst-b-infer-e subst-tb.simps* **by** *metis*
have $c[z::=[\ x\ ]^v]_{cv}[bv::=b]_{cb}$ = $(c[bv::=b]_{cb})[z::=V\text{-}var\ x]_{cv}$
using *subst-cv-subst-bb-commute*[*of bv V-var x c z b*] *fresh-at-base* **by** *simp*
thus Θ ; Φ ; {||} ; ((*x*, $b'[bv::=b]_{bb}$, $(c[bv::=b]_{cb})[z::=V\text{-}var\ x]_v$) $\#_\Gamma$ $\Gamma[bv::=b]_{\Gamma b}$) ; $\Delta[bv::=b]_{\Delta b}$ ⊢ $s[bv::=b]_{sb}$ ⇐ $\tau[bv::=b]_{\tau b}$
using *check-letI subst-gb.simps subst-defs* **by** *metis*
qed
next
case (*check-assertI x* Θ Φ *ℬ* Γ Δ *c* τ *s*)
show *?case* **proof**(*subst subst-sb.simps*, *rule Typing.check-assertI*)
show *atom x* $\sharp$ (Θ, Φ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $c[bv::=b]_{cb}$, $\tau[bv::=b]_{\tau b}$, $s[bv::=b]_{sb}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-assertI fresh-empty-fset*)+
apply(*metis* ∗ *subst-b-fresh-x check-assertI fresh-prodN*)+ **done**

have Θ ; Φ ; {||} ; ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ)$[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ ⊢ $s[bv::=b]_{sb}$ ⇐ $\tau[bv::=b]_{\tau b}$ **using** *check-assertI*
by *metis*
thus Θ ; Φ ; {||} ; (*x*, *B-bool*, $c[bv::=b]_{cb}$) $\#_\Gamma$ $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ ⊢ $s[bv::=b]_{sb}$ ⇐ $\tau[bv::=b]_{\tau b}$ **using** *subst-gb.simps* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ ⊨ $c[bv::=b]_{cb}$ **using** *subst-b-valid check-assertI* **by** *simp*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst2*(*6*) *check-assertI* **by** *simp*
qed
next
case (*check-branch-list-consI* Θ Φ *ℬ* Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **unfolding** *subst-branchlb.simps* **using** *Typing.check-branch-list-consI* **by** *simp*
next
case (*check-branch-list-finalI* Θ Φ *ℬ* Γ Δ *tid dclist v cs* τ)
then show *?case* **unfolding** *subst-branchlb.simps* **using** *Typing.check-branch-list-finalI* **by** *simp*
next
case (*check-branch-s-branchI* Θ *ℬ* Γ Δ τ *const x* Φ *tid cons v s*)

show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-branch-s-branchI*)
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *check-branch-s-branchI wf-b-subst subst-db.simps* **by** *metis*
show $\vdash_{wf}$ Θ **using** *check-branch-s-branchI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\tau[bv::=b]_{\tau b}$ **using** *check-branch-s-branchI wf-b-subst* **by** *metis*

show *atom x* $\sharp$(Θ, Φ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, *tid*, *cons* , *const*, $v[bv::=b]_{vb}$, $\tau[bv::=b]_{\tau b}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-branch-s-branchI fresh-empty-fset*)+

apply(*metis* ∗ *subst-b-fresh-x check-branch-s-branchI fresh-prodN*)+
done
show *wft*:Θ ; {||} ; *GNil* $\vdash_{wf}$ *const* **using** *check-branch-s-branchI* **by** *auto*
hence (*b-of const*) = (*b-of const*)$[bv{::=}b]_{bb}$
using *wfT-nil-supp fresh-def*[*of atom bv*] *forget-subst subst-b-b-def τ.supp*
bot.extremum-uniqueI ex-in-conv fresh-def supp-empty-fset
by (*metis b-of-supp*)
moreover have (*c-of const x*)$[bv{::=}b]_{cb}$ = *c-of const x*
using *wft wfT-nil-supp fresh-def*[*of atom bv*] *forget-subst subst-b-c-def τ.supp*
bot.extremum-uniqueI ex-in-conv fresh-def supp-empty-fset subst-b-c-of-forget **by** *metis*
ultimately show Θ ; Φ ; {||} ; (*x*, *b-of const*, *CE-val* ($v[bv{::=}b]_{vb}$) == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ $\Gamma[bv{::=}b]_{\Gamma b}$; $\Delta[bv{::=}b]_{\Delta b}$ ⊢ $s[bv{::=}b]_{sb} \Leftarrow \tau[bv{::=}b]_{\tau b}$
using *check-branch-s-branchI subst-gb.simps* **by** *auto*
qed
next
case (*check-ifI z* Θ Φ *ℬ* Γ Δ *v s1 s2 τ*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-ifI*)
show ‹*atom z* ♯ (Θ, Φ, {||}, $\Gamma[bv{::=}b]_{\Gamma b}$, $\Delta[bv{::=}b]_{\Delta b}$, $v[bv{::=}b]_{vb}$, *s1*$[bv{::=}b]_{sb}$, *s2*$[bv{::=}b]_{sb}$, $\tau[bv{::=}b]_{\tau b}$)›
by(*unfold fresh-prodN*,*auto*, *auto simp add*: *check-ifI fresh-empty-fset subst-b-fresh-x*)
have ⦃ *z* : *B-bool* | *TRUE* ⦄$[bv{::=}b]_{\tau b}$ = ⦃ *z* : *B-bool* | *TRUE* ⦄ **by** *auto*
thus ‹Θ ; {||} ; $\Gamma[bv{::=}b]_{\Gamma b}$ ⊢ $v[bv{::=}b]_{vb}$ ⇐ ⦃ *z* : *B-bool* | *TRUE* ⦄› **using** *check-ifI subst-b-check-v* **by** *metis*
show ‹ Θ ; Φ ; {||} ; $\Gamma[bv{::=}b]_{\Gamma b}$; $\Delta[bv{::=}b]_{\Delta b}$ ⊢ *s1*$[bv{::=}b]_{sb}$ ⇐ ⦃ *z* : *b-of* $\tau[bv{::=}b]_{\tau b}$ | *CE-val* ($v[bv{::=}b]_{vb}$) == *CE-val* (*V-lit L-true*) *IMP c-of* $\tau[bv{::=}b]_{\tau b}$ *z* ⦄›
using *subst-b-if check-ifI* **by** *metis*
show ‹ Θ ; Φ ; {||} ; $\Gamma[bv{::=}b]_{\Gamma b}$; $\Delta[bv{::=}b]_{\Delta b}$ ⊢ *s2*$[bv{::=}b]_{sb}$ ⇐ ⦃ *z* : *b-of* $\tau[bv{::=}b]_{\tau b}$ | *CE-val* ($v[bv{::=}b]_{vb}$) == *CE-val* (*V-lit L-false*) *IMP c-of* $\tau[bv{::=}b]_{\tau b}$ *z* ⦄›
using *subst-b-if check-ifI* **by** *metis*
qed
next
case (*check-let2I x* Θ Φ *ℬ G* Δ *t s1 τ s2*)
show *?case* **unfolding** *subst-b-simps* **proof** (*rule Typing.check-let2I*)
have *atom x* ♯ *b* **using** *x-fresh-b* **by** *auto*
show ‹*atom x* ♯ (Θ, Φ, {||}, $G[bv{::=}b]_{\Gamma b}$, $\Delta[bv{::=}b]_{\Delta b}$, $t[bv{::=}b]_{\tau b}$, *s1*$[bv{::=}b]_{sb}$, $\tau[bv{::=}b]_{\tau b}$)›
apply(*unfold fresh-prodN*, *auto*, *auto simp add*: *check-let2I fresh-prodN fresh-empty-fset*)
apply(*metis subst-b-fresh-x check-let2I fresh-prodN*)+
done

show ‹ Θ ; Φ ; {||} ; $G[bv{::=}b]_{\Gamma b}$; $\Delta[bv{::=}b]_{\Delta b}$ ⊢ *s1*$[bv{::=}b]_{sb}$ ⇐ $t[bv{::=}b]_{\tau b}$ › **using** *check-let2I subst-tb.simps* **by** *auto*
show ‹ Θ ; Φ ; {||} ; (*x*, *b-of* $t[bv{::=}b]_{\tau b}$, *c-of* $t[bv{::=}b]_{\tau b}$ *x*) $\#_\Gamma$ $G[bv{::=}b]_{\Gamma b}$; $\Delta[bv{::=}b]_{\Delta b}$ ⊢ *s2*$[bv{::=}b]_{sb}$ ⇐ $\tau[bv{::=}b]_{\tau b}$›
using *check-let2I subst-tb.simps subst-gb.simps b-of.simps subst-b-c-of subst-b-b-of* **by** *auto*
qed
next
case (*check-varI u* Θ Φ *ℬ* Γ Δ *τ′ v τ s*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-varI*)
show *atom u* ♯ (Θ, Φ, {||}, $\Gamma[bv{::=}b]_{\Gamma b}$, $\Delta[bv{::=}b]_{\Delta b}$, $\tau'[bv{::=}b]_{\tau b}$, $v[bv{::=}b]_{vb}$, $\tau[bv{::=}b]_{\tau b}$)
by(*unfold fresh-prodN*,*auto simp add*: *check-varI fresh-empty-fset subst-b-fresh-u*)
show Θ ; {||} ; $\Gamma[bv{::=}b]_{\Gamma b}$ ⊢ $v[bv{::=}b]_{vb}$ ⇐ $\tau'[bv{::=}b]_{\tau b}$ **using** *check-varI subst-b-check-v* **by** *auto*
show Θ ; Φ ; {||} ; (*subst-gb* Γ *bv b*) ; (*u*, ($\tau'[bv{::=}b]_{\tau b}$)) $\#_\Delta$ (*subst-db* Δ *bv b*) ⊢ ($s[bv{::=}b]_{sb}$) ⇐

($\tau[bv::=b]_{\tau b}$) **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ *v z* τ')
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-assignI*)
show Θ $\vdash_{wf}$ Φ **using** *check-assignI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst check-assignI* **by** *auto*
show (u, $\tau[bv::=b]_{\tau b}$) ∈ *setD* $\Delta[bv::=b]_{\Delta b}$ **using** *check-assignI subst-d-b-member* **by** *simp*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ ⊢ $v[bv::=b]_{vb} \Leftarrow \tau[bv::=b]_{\tau b}$ **using** *check-assignI subst-b-check-v* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ ⊢ ⦃ *z* : *B-unit* | *TRUE* ⦄ ≲ $\tau'[bv::=b]_{\tau b}$ **using** *check-assignI subst-b-subtype subst-b-simps subst-tb.simps* **by** *fastforce*
qed
next
case (*check-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ')
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-whileI*)
show Θ ; Φ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ ⊢ $s1[bv::=b]_{sb}$ ⇐ ⦃ *z* : *B-bool* | *TRUE* ⦄ **using** *check-whileI* **by** *auto*
show Θ ; Φ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ ⊢ $s2[bv::=b]_{sb}$ ⇐ ⦃ *z* : *B-unit* | *TRUE* ⦄ **using** *check-whileI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ ⊢ ⦃ *z* : *B-unit* | *TRUE* ⦄ ≲ $\tau'[bv::=b]_{\tau b}$ **using** *subst-b-subtype check-whileI* **by** *fastforce*
qed
next
case (*check-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ)
then show *?case* **unfolding** *subst-sb.simps* **using** *check-seqI Typing.check-seqI subst-b-eq* **by** *metis*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *z*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-caseI*)
show ‹ Θ ; Φ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$; *tid* ; *dclist* ; $v[bv::=b]_{vb}$ ⊢ *subst-branchlb cs bv b* ⇐ $\tau[bv::=b]_{\tau b}$› **using** *check-caseI* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ› **using** *check-caseI* **by** *auto*
show ‹Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ ⊢ $v[bv::=b]_{vb}$ ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄› **using** *check-caseI subst-b-check-v subst-b-simps subst-tb.simps subst-b-simps*
proof −
have ⦃ *z* : *B-id tid* | *TRUE* ⦄ = ⦃ *z* : *B-id tid* | *TRUE* ⦄$[bv::=b]_{\tau b}$ **using** *subst-b-eq* **by** *auto*
then show *?thesis*
by (*metis* (*no-types*) *check-caseI.hyps*(*4*) *check-caseI.prems*(*1*) *check-caseI.prems*(*2*) *subst-b-check-v*)

qed
show ‹ $\vdash_{wf}$ Θ › **using** *check-caseI* **by** *auto*
qed
qed

end

method *supp-calc* = (*metis* (*mono-tags*, *opaque-lifting*) *pure-supp c.supp e.supp v.supp supp-l-empty opp.supp sup-bot.right-neutral supp-at-base*)
declare *infer-e.intros*[*simp*]
declare *infer-e.intros*[*intro*]

Chapter 16

Safety

Lemmas about the operational semantics leading up to progress and preservation and then safety.

16.1 Store Lemmas

abbreviation *delta-ext* (‹ - $\sqsubseteq$ - ›) **where**
delta-ext $\Delta\ \Delta' \equiv$ (*setD* $\Delta \subseteq$ *setD* Δ')

nominal-function *dc-of* :: *branch-s* ⇒ *string* **where**
dc-of (*AS-branch dc* - -) = *dc*
apply(*auto*,*simp add*: *eqvt-def dc-of-graph-aux-def*)
using *s-branch-s-branch-list.exhaust* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *delta-sim-fresh*:
assumes $\Theta \vdash \delta \sim \Delta$ **and** *atom u* $\sharp$ δ
shows *atom u* $\sharp$ Δ
using *assms* **proof**(*induct rule* : *delta-sim.inducts*)
case (*delta-sim-nilI* Θ)
then show *?case* **using** *fresh-def supp-DNil* **by** *blast*
next
case (*delta-sim-consI* Θ δ Δ *v* τ *u'*)
hence Θ ; {||} ; *GNil* $\vdash_{wf}$ τ **using** *check-v-wf* **by** *meson*
hence *supp* τ = {} **using** *wfT-supp* **by** *fastforce*
moreover have *atom u* $\sharp$ *u'* **using** *delta-sim-consI fresh-Cons fresh-Pair* **by** *blast*
moreover have *atom u* $\sharp$ Δ **using** *delta-sim-consI fresh-Cons* **by** *blast*
ultimately show *?case* **using** *fresh-Pair fresh-DCons fresh-def* **by** *blast*
qed

lemma *delta-sim-v*:
fixes Δ::Δ
assumes $\Theta \vdash \delta \sim \Delta$ **and** (*u*,*v*) ∈ *set* δ **and** (*u*,τ) ∈ *setD* Δ **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ
shows Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ
using *assms* **proof**(*induct* δ *arbitrary*: Δ)
case *Nil*
then show *?case* **by** *auto*

next
  case (*Cons uv* δ)
  obtain u' **and** v' **where** *uv* : *uv*=(u',v') **by** *fastforce*
  show *?case* **proof**(*cases* u'=*u*)
    case *True*
    hence $*$:$\Theta \vdash ((u,v')\#\delta) \sim \Delta$ **using** *uv Cons* **by** *blast*
    then obtain τ' **and** Δ' **where** *tt*: Θ ; {||} ; *GNil* $\vdash v' \Leftarrow \tau' \wedge u \notin$ *fst* ' *set* $\delta \wedge \Delta = (u,\tau')\#_\Delta\Delta'$
using *delta-sim-elims*(*3*)[*OF* $*$] **by** *metis*
    moreover hence v'=*v* **using** *Cons True*
      by (*metis Pair-inject fst-conv image-eqI set-ConsD uv*)
    moreover have τ=τ' **using** *wfD-unique tt Cons*
        setD.simps list.set-intros **by** *blast*
    ultimately show *?thesis* **by** *metis*
  next
    case *False*
    hence $*$:$\Theta \vdash ((u',v')\#\delta) \sim \Delta$ **using** *uv Cons* **by** *blast*
    then obtain τ' **and** Δ' **where** *tt*: $\Theta \vdash \delta \sim \Delta' \wedge \Theta$; {||} ; *GNil* $\vdash v' \Leftarrow \tau' \wedge u' \notin$ *fst* ' *set* $\delta \wedge \Delta$ $= (u',\tau')\#_\Delta\Delta'$ **using** *delta-sim-elims*(*3*)[*OF* $*$] **by** *metis*

    moreover hence Θ ; {||} ; *GNil* $\vdash_{wf} \Delta'$ **using** *wfD-elims Cons delta-sim-elims* **by** *metis*
    ultimately show *?thesis* **using** *Cons*
      using *False* **by** *auto*
  qed
qed

lemma *delta-sim-delta-lookup*:
  assumes $\Theta \vdash \delta \sim \Delta$ **and** (*u*, ⦃ *z* : *b* | *c* ⦄) $\in$ *setD* Δ
  shows $\exists$ *v*. (*u*,*v*) $\in$ *set* δ
  using *assms* **by**(*induct rule*: *delta-sim.inducts*,*auto*+)

lemma *update-d-stable*:
  fst ' *set* δ = *fst* ' *set* (*update-d* δ *u v*)
proof(*induct* δ)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons a* δ)
  then show *?case* **using** *update-d.simps*
    by (*metis* (*no-types*, *lifting*) *eq-fst-iff image-cong image-insert list.simps*(*15*) *prod.exhaust-sel*)
qed

lemma *update-d-sim*:
  fixes Δ::Δ
  assumes $\Theta \vdash \delta \sim \Delta$ **and** Θ ; {||} ; *GNil* $\vdash v \Leftarrow \tau$ **and** $(u,\tau) \in$ *setD* Δ **and** Θ ; {||} ; *GNil* $\vdash_{wf} \Delta$
  shows $\Theta \vdash$ (*update-d* δ *u v*) $\sim \Delta$
  using *assms* **proof**(*induct* δ *arbitrary*: Δ)
  case *Nil*
  then show *?case* **using** *delta-sim-consI* **by** *simp*
next
  case (*Cons uv* δ)
  obtain u' **and** v' **where** *uv* : *uv*=(u',v') **by** *fastforce*

hence $*:\Theta \vdash ((u',v')\#\delta) \sim \Delta$ **using** *uv Cons* **by** *blast*
then obtain τ' **and** Δ' **where** *tt*: $\Theta \vdash \delta \sim \Delta' \wedge \Theta$; $\{||\}$; *GNil* $\vdash v' \Leftarrow \tau' \wedge u' \notin$ *fst* ' *set* $\delta \wedge \Delta = (u',\tau')\#_\Delta\Delta'$ **using** *delta-sim-elims* $*$ **by** *metis*

show *?case* **proof**(*cases* $u=u'$)
case *True*
then have $(u,\tau') \in$ *setD* Δ **using** *tt* **by** *auto*
then have $\tau = \tau'$ **using** *Cons wfD-unique* **by** *metis*
moreover have *update-d* $((u',v')\#\delta)$ u v = $((u',v)\#\delta)$ **using** *update-d.simps True* **by** *presburger*
ultimately show *?thesis* **using** *delta-sim-consI tt Cons True*
by (*simp add*: *tt uv*)
next
case *False*
have $\Theta \vdash (u',v')$ # (*update-d* δ u v) $\sim (u',\tau')\#_\Delta\Delta'$
proof(*rule delta-sim-consI*)
show $\Theta \vdash$ *update-d* δ u $v \sim \Delta'$ **using** *Cons* **using** *delta-sim-consI delta-sim.simps update-d.simps Cons delta-sim-elims uv tt False fst-conv set-ConsD wfG-elims wfD-elims* **by** (*metis setD-ConsD*)
show Θ ; $\{||\}$; *GNil* $\vdash v' \Leftarrow \tau'$ **using** *tt* **by** *auto*
show $u' \notin$ *fst* ' *set* (*update-d* δ u v) **using** *update-d.simps Cons update-d-stable tt* **by** *auto*
qed
thus *?thesis* **using** *False update-d.simps uv*
by (*simp add*: *tt*)
qed
qed

16.2 Preservation

Types are preserved under reduction step. Broken down into lemmas about different operations

16.2.1 Function Application

lemma *check-s-x-fresh*:
fixes *x*::*x* **and** *s*::*s*
assumes Θ ; Φ ; B ; *GNil* ; $D \vdash s \Leftarrow \tau$
shows *atom* $x \sharp s \wedge$ *atom* $x \sharp \tau \wedge$ *atom* $x \sharp D$
proof −
have Θ ; Φ ; B ; *GNil* ; $D \vdash_{wf} s$: *b-of* τ **using** *check-s-wf*[*OF assms*] **by** *auto*
moreover have Θ ; B ; *GNil* $\vdash_{wf} \tau$ **using** *check-s-wf assms* **by** *auto*
moreover have Θ ; B ; *GNil* $\vdash_{wf} D$ **using** *check-s-wf assms* **by** *auto*
ultimately show *?thesis* **using** *wf-supp x-fresh-u*
by (*meson fresh-GNil wfS-x-fresh wfT-x-fresh wfD-x-fresh*)
qed

lemma *check-funtyp-subst-b*:
fixes b'::*b*
assumes *check-funtyp* Θ Φ $\{|bv|\}$ (*AF-fun-typ* x b c τ s) **and** ‹ Θ ; $\{||\}$ $\vdash_{wf} b'$ ›
shows *check-funtyp* Θ Φ $\{||\}$ (*AF-fun-typ* x $b[bv::=b']_{bb}$ $(c[bv::=b']_{cb})$ $\tau[bv::=b']_{\tau b}$ $s[bv::=b']_{sb})$
using *assms* **proof** (*nominal-induct* $\{|bv|\}$ *AF-fun-typ* x b c τ s *rule*: *check-funtyp.strong-induct*)
case (*check-funtypI* x' Θ Φ c' s' τ')
have *check-funtyp* Θ Φ $\{||\}$ (*AF-fun-typ* x' $b[bv::=b']_{bb}$ $(c'[bv::=b']_{cb})$ $\tau'[bv::=b']_{\tau b}$ $s'[bv::=b']_{sb})$ **proof**

show ‹*atom* x' ♯ (Θ, Φ, {||}::*bv fset*, $b[bv::=b']_{bb}$)› **using** *check-funtypI fresh-prodN x-fresh-b fresh-empty-fset* **by** *metis*

have ‹ Θ ; Φ ; {||} ; $((x', b, c') \#_\Gamma$ *GNil*$)[bv::=b']_{\Gamma b}$; $[]_\Delta[bv::=b']_{\Delta b}$ ⊢ $s'[bv::=b']_{sb} \Leftarrow \tau'[bv::=b']_{\tau b}$›
proof(*rule subst-b-check-s*)
show ‹ Θ ; {||} $\vdash_{wf}$ b' › **using** *check-funtypI* **by** *metis*
show ‹{|*bv*|} = {|*bv*|}› **by** *auto*
show ‹ Θ ; Φ ; {|*bv*|} ; $(x', b, c') \#_\Gamma$ *GNil* ; $[]_\Delta$ ⊢ $s' \Leftarrow \tau'$› **using** *check-funtypI* **by** *metis*
qed

thus ‹ Θ ; Φ ; {||} ; $(x', b[bv::=b']_{bb}, c'[bv::=b']_{cb}) \#_\Gamma$ *GNil* ; $[]_\Delta$ ⊢ $s'[bv::=b']_{sb} \Leftarrow \tau'[bv::=b']_{\tau b}$›
using *subst-gb.simps subst-db.simps* **by** *simp*
qed

moreover have (*AF-fun-typ x b c τ s*) = (*AF-fun-typ* x' *b* c' τ' s') **using** *fun-typ.eq-iff check-funtypI* **by** *metis*
moreover hence (*AF-fun-typ x* $b[bv::=b']_{bb}$ $(c[bv::=b']_{cb})$ $\tau[bv::=b']_{\tau b}$ $s[bv::=b']_{sb}$) = (*AF-fun-typ* x' $b[bv::=b']_{bb}$ $(c'[bv::=b']_{cb})$ $\tau'[bv::=b']_{\tau b}$ $s'[bv::=b']_{sb}$)
using *subst-ft-b.simps* **by** *metis*
ultimately show *?case* **by** *metis*
qed

lemma *funtyp-simple-check*:
fixes *s*::*s* **and** Δ::Δ **and** τ::τ **and** *v*::*v*
assumes *check-funtyp* Θ Φ ({||}::*bv fset*) (*AF-fun-typ x b c τ s*) **and**
Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *x* : *b* | *c* ⦄
shows Θ ; Φ ; {||} ; *GNil* ; *DNil* ⊢ $s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* ({||}::*bv fset*) *AF-fun-typ x b c τ s avoiding*: *v x* *rule*: *check-funtyp.strong-induct*)
case (*check-funtypI* x' Θ Φ c' s' τ')

hence *eq1*: ⦃ x' : *b* | c' ⦄ = ⦃ *x* : *b* | *c* ⦄ **using** *funtyp-eq-iff-equalities* **by** *metis*

obtain x'' **and** c'' **where** *xf*:⦃ *x* : *b* | *c* ⦄ = ⦃ x'' : *b* | c'' ⦄ ∧ *atom* x'' ♯ (x',*v*) ∧ *atom* x'' ♯ (*x*,*c*) **using** *obtain-fresh-z3* **by** *metis*
moreover have *atom* x' ♯ c'' **proof** −
have *supp* ⦃ x'' : *b* | c'' ⦄ = {} **using** *eq1 check-funtypI xf check-v-wf wfT-nil-supp* **by** *metis*
hence *supp* c'' ⊆ { *atom* x'' } **using** *τ.supp eq1 xf* **by** (*auto simp add*: *freshers*)
moreover have *atom* x' ≠ *atom* x'' **using** *xf fresh-Pair fresh-x-neq* **by** *metis*
ultimately show *?thesis* **using** *xf fresh-Pair fresh-x-neq fresh-def fresh-at-base* **by** *blast*
qed
ultimately have *eq2*: $c''[x''::=[\ x'\ ]^v]_{cv} = c'$ **using** *eq1* *type-eq-subst-eq3*(*1*)[*of* x' *b* c' x'' *b* c''] **by** *metis*

have *atom* x' ♯ *c* **proof** −
have *supp* ⦃ *x* : *b* | *c* ⦄ = {} **using** *eq1 check-funtypI xf check-v-wf wfT-nil-supp* **by** *metis*
hence *supp c* ⊆ { *atom x* } **using** *τ.supp* **by** *auto*
moreover have *atom x* ≠ *atom* x' **using** *check-funtypI fresh-Pair fresh-x-neq* **by** *metis*
ultimately show *?thesis* **using** *fresh-def* **by** *force*
qed
hence *eq*: $c[x::=[\ x'\ ]^v]_{cv} = c' \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge \tau'[x'::=v]_{\tau v} = \tau[x::=v]_{\tau v}$
using *funtyp-eq-iff-equalities type-eq-subst-eq3 check-funtypI* **by** *metis*

have Θ ; Φ ; $\{||\}$; $((x', b, c''[x''::=[x']^v]_{cv})$ $\#_\Gamma$ $GNil)[x'::=v]_{\Gamma v}$; $[]_\Delta[x'::=v]_{\Delta v}$ $\vdash s'[x'::=v]_{sv} \Leftarrow \tau'[x'::=v]_{\tau v}$
proof(*rule subst-check-check-s*)
show ‹Θ ; $\{||\}$; *GNil* $\vdash v \Leftarrow \{\!| x'' : b \mid c'' |\!\}$› **using** *check-funtypI eq1 xf* **by** *metis*
show ‹*atom* x'' ♯ (x', v)› **using** *check-funtypI fresh-x-neq fresh-Pair xf* **by** *metis*
show ‹ Θ ; Φ ; $\{||\}$; $(x', b, c''[x''::=[x']^v]_{cv})$ $\#_\Gamma$ *GNil* ; $[]_\Delta$ $\vdash s' \Leftarrow \tau'$› **using** *check-funtypI eq2* **by** *metis*
show ‹ $(x', b, c''[x''::=[x']^v]_{cv})$ $\#_\Gamma$ *GNil* = *GNil* @ $(x', b, c''[x''::=[x']^v]_{cv})$ $\#_\Gamma$ *GNil*› **using** *append-g.simps* **by** *auto*
qed
hence Θ; Φ; $\{||\}$; *GNil*; $[]_\Delta$ $\vdash s'[x'::=v]_{sv} \Leftarrow \tau'[x'::=v]_{\tau v}$ **using** *subst-gv.simps subst-dv.simps* **by** *auto*
thus *?case* **using** *eq* **by** *auto*
qed

lemma *funtypq-simple-check*:
fixes *s::s* **and** Δ::Δ **and** τ::τ **and** *v::v*
assumes *check-funtypq* Θ Φ (*AF-fun-typ-none* (*AF-fun-typ x b c t s*)) **and**
Θ ; $\{||\}$; *GNil* $\vdash v \Leftarrow \{\!| x : b \mid c |\!\}$
shows Θ; Φ; $\{||\}$; *GNil*; *DNil* $\vdash s[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fun-typ-none* (*AF-fun-typ x b c t s*)) *avoiding*: *v rule*: *check-funtypq.strong-induct*)
case (*check-fundefq-simpleI* Θ Φ x' c' t' s')
hence *eq*: $\{\!| x : b \mid c |\!\} = \{\!| x' : b \mid c' |\!\} \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge t[x::=v]_{\tau v} = t'[x'::=v]_{\tau v}$
using *funtyp-eq-iff-equalities* **by** *metis*
hence Θ; Φ; $\{||\}$; *GNil*; $[]_\Delta$ $\vdash s'[x'::=v]_{sv} \Leftarrow t'[x'::=v]_{\tau v}$
using *funtyp-simple-check*[*OF check-fundefq-simpleI*(*1*)] *check-fundefq-simpleI* **by** *metis*
thus *?case* **using** *eq* **by** *metis*
qed

lemma *funtyp-poly-eq-iff-equalities*:
assumes [[*atom* bv']]*lst*. *AF-fun-typ* x' b'' c' t' s' = [[*atom bv*]]*lst*. *AF-fun-typ x b c t s*
shows $\{\!| x' : b''[bv'::=b']_{bb} \mid c'[bv'::=b']_{cb} |\!\} = \{\!| x : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} |\!\} \wedge$
$s'[bv'::=b']_{sb}[x'::=v]_{sv} = s[bv::=b']_{sb}[x::=v]_{sv} \wedge t'[bv'::=b']_{\tau b}[x'::=v]_{\tau v} = t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
proof −
have *subst-ft-b* (*AF-fun-typ* x' b'' c' t' s') bv' b' = *subst-ft-b* (*AF-fun-typ x b c t s*) *bv* b'
using *subst-b-flip-eq-two subst-b-fun-typ-def assms* **by** *metis*
thus *?thesis* **using** *fun-typ.eq-iff subst-ft-b.simps funtyp-eq-iff-equalities subst-tb.simps*
by (*metis* (*full-types*) *assms fun-poly-arg-unique*)

qed

lemma *funtypq-poly-check*:
fixes *s::s* **and** Δ::Δ **and** τ::τ **and** *v::v* **and** b'::*b*
assumes *check-funtypq* Θ Φ (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*)) **and**
Θ ; $\{||\}$; *GNil* $\vdash v \Leftarrow \{\!| x : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} |\!\}$ **and**
Θ ; $\{||\}$ $\vdash_{wf}$ b'
shows Θ; Φ; $\{||\}$; *GNil*; *DNil* $\vdash s[bv::=b']_{sb}[x::=v]_{sv} \Leftarrow t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*)) *avoiding*: *v rule*: *check-funtypq.strong-induct*)
case (*check-funtypq-polyI* bv' Θ Φ x' b'' c' t' s')

hence $**$:⦃ x' : $b''[bv'::=b']_{bb}$ | $c'[bv'::=b']_{cb}$ ⦄ = ⦃ x : $b[bv::=b']_{bb}$ | $c[bv::=b']_{cb}$ ⦄ ∧
$s'[bv'::=b']_{sb}[x'::=v]_{sv} = s[bv::=b']_{sb}[x::=v]_{sv} \wedge t'[bv'::=b']_{\tau b}[x'::=v]_{\tau v} = t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
using *funtyp-poly-eq-iff-equalities* **by** *metis*

have $*$:*check-funtyp* Θ Φ {||} (*AF-fun-typ* x' $b''[bv'::=b']_{bb}$ ($c'[bv'::=b']_{cb}$) ($t'[bv'::=b']_{\tau b}$) $s'[bv'::=b']_{sb}$)
using *check-funtyp-subst-b*[*OF check-funtypq-polyI*(*5*) *check-funtypq-polyI*(*8*)] **by** *metis*
moreover have Θ ; {||} ; *GNil* ⊢ v ⇐ ⦃ x' : $b''[bv'::=b']_{bb}$ | $c'[bv'::=b']_{cb}$ ⦄ **using** $**$ *check-funtypq-polyI*
by *metis*
ultimately have Θ; Φ; {||}; *GNil*; $[]_\Delta$ ⊢ $s'[bv'::=b']_{sb}[x'::=v]_{sv}$ ⇐ $t'[bv'::=b']_{\tau b}[x'::=v]_{\tau v}$
using *funtyp-simple-check*[*OF* $*$] *check-funtypq-polyI* **by** *metis*
thus *?case* **using** $**$ **by** *metis*

qed

lemma *fundef-simple-check*:
fixes *s*::*s* **and** Δ::Δ **and** τ::τ **and** *v*::*v*
assumes *check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c t s*))) **and**
Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *x* : *b* | *c* ⦄ **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ
shows Θ; Φ; {||}; *GNil*; Δ ⊢ $s[x::=v]_{sv}$ ⇐ $t[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c t s*))) *avoiding*:
v rule: *check-fundef.strong-induct*)
case (*check-fundefI* Θ Φ)
then show *?case* **using** *funtypq-simple-check*[*THEN check-s-d-weakening*(*1*)] *setD.simps* **by** *auto*
qed

lemma *fundef-poly-check*:
fixes *s*::*s* **and** Δ::Δ **and** τ::τ **and** *v*::*v* **and** b'::*b*
assumes *check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*))) **and**
Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *x* : $b[bv::=b']_{bb}$ | $c[bv::=b']_{cb}$ ⦄ **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ **and** Θ ; {||}
$\vdash_{wf}$ b'
shows Θ; Φ; {||}; *GNil*; Δ ⊢ $s[bv::=b']_{sb}[x::=v]_{sv}$ ⇐ $t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*))) *avoiding*: *v rule*: *check-fundef.strong-induct*)
case (*check-fundefI* Θ Φ)
then show *?case* **using** *funtypq-poly-check*[*THEN check-s-d-weakening*(*1*)] *setD.simps* **by** *auto*
qed

lemma *preservation-app*:
assumes
Some (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = *lookup-fun* Φ *f* **and** (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
shows Θ ; Φ ; *B* ; *G* ; Δ ⊢ *ss* ⇐ τ ⟹ *B* = {||} ⟹ *G* = *GNil* ⟹ *ss* = *LET x* = (*AE-app f v*) *IN s* ⟹
Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* : ($τ1'[x1::=v]_{\tau v}$) = ($s1'[x1::=v]_{sv}$) *IN s* ⇐ τ **and**
check-branch-s Θ Φ ℬ *GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ ℬ Γ Δ *tid dclist v css* τ ⟹ *True*
using *assms* **proof**(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v rule*: *check-s-check-branch-s-check-branch-list.strong-ind*
case (*check-letI x2* Θ Φ ℬ Γ Δ *e* τ *z s2 b c*)

hence *eq*: *e* = (*AE-app f v*) **by** *simp*
hence $*$:Θ ; Φ ; {||} ;*GNil* ; Δ ⊢ (*AE-app f v*) ⇒ ⦃ *z* : *b* | *c* ⦄ **using** *check-letI* **by** *auto*

then obtain *x3 b3 c3* τ*3 s3* **where**
$**$:Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ $\wedge$ Θ $\vdash_{wf}$ Φ $\wedge$ *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x3 b3 c3* τ*3 s3*))) = *lookup-fun* Φ *f* $\wedge$
Θ ; {||} ; *GNil* $\vdash$ *v* $\Leftarrow$ ⦃ *x3* : *b3* | *c3* ⦄ $\wedge$ *atom x3* ♯ (Θ, Φ, ({||}::*bv fset*), *GNil*, Δ, *v*, ⦃ *z* : *b* | *c* ⦄) $\wedge$ $\tau 3[x3::=v]_{\tau v}$ = ⦃ *z* : *b* | *c* ⦄
using *infer-e-elims*(*6*)[*OF* $*$] *subst-defs* **by** *metis*

obtain *z3* **where** *z3*:⦃ *x3* : *b3* | *c3* ⦄ = ⦃ *z3* : *b3* | $c3[x3::=V\text{-}var\ z3]_{cv}$ ⦄ $\wedge$ *atom z3* ♯ (*x3*, *v*,*c3*,*x1*,*c1*) **using** *obtain-fresh-z3* **by** *metis*

have *seq*:[[*atom x3*]]*lst. s3* = [[*atom x1*]]*lst. s1'* **using** *fun-def-eq check-letI* $**$ *option.inject* **by** *metis*

let *?ft* = *AF-fun-typ x3 b3 c3* τ*3 s3*

have *sup*: *supp* τ*3* $\subseteq$ { *atom x3*} $\wedge$ *supp s3* $\subseteq$ { *atom x3*} **using** *wfPhi-f-supp* $**$ **by** *metis*

have Θ; Φ; {||}; *GNil*; Δ $\vdash$ *AS-let2 x2* $\tau 3[x3::=v]_{\tau v}$ ($s3[x3::=v]_{sv}$) *s2* $\Leftarrow$ τ **proof**
show ‹*atom x2* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, $\tau 3[x3::=v]_{\tau v}$, $s3[x3::=v]_{sv}$, τ)›
unfolding *fresh-prodN* **using** *check-letI fresh-subst-v-if subst-v-*τ*-def sup*
by (*metis all-not-in-conv fresh-def fresh-empty-fset fresh-subst-sv-if fresh-subst-tv-if singleton-iff subset-singleton-iff*)

show ‹ Θ; Φ; {||}; *GNil*; Δ $\vdash$ $s3[x3::=v]_{sv}$ $\Leftarrow$ $\tau 3[x3::=v]_{\tau v}$› **proof**(*rule fundef-simple-check*)
show ‹*check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x3 b3 c3* τ*3 s3*)))› **using** $**$ *check-letI lookup-fun-member* **by** *metis*
show ‹Θ ; {||} ; *GNil* $\vdash$ *v* $\Leftarrow$ ⦃ *x3* : *b3* | *c3* ⦄› **using** $**$ **by** *auto*
show ‹ Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ › **using** $**$ **by** *auto*
qed
show ‹ Θ ; Φ ; {||} ; (*x2*, *b-of* $\tau 3[x3::=v]_{\tau v}$, *c-of* $\tau 3[x3::=v]_{\tau v}$ *x2*) $\#_{\Gamma}$ *GNil* ; Δ $\vdash$ *s2* $\Leftarrow$ τ›
using *check-letI* $**$ *b-of.simps c-of.simps subst-defs* **by** *metis*
qed

moreover have *AS-let2 x2* $\tau 3[x3::=v]_{\tau v}$ ($s3[x3::=v]_{sv}$) *s2* = *AS-let2 x* ($\tau 1'[x1::=v]_{\tau v}$) ($s1'[x1::=v]_{sv}$) *s* **proof** −
have $*$: [[*atom x2*]]*lst. s2* = [[*atom x*]]*lst. s* **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *auto*
moreover have $\tau 3[x3::=v]_{\tau v}$ = $\tau 1'[x1::=v]_{\tau v}$ **using** *fun-ret-unique* $**$ *check-letI* **by** *metis*
moreover have $s3[x3::=v]_{sv}$ = ($s1'[x1::=v]_{sv}$) **using** *subst-v-flip-eq-two subst-v-s-def seq* **by** *metis*
ultimately show *?thesis* **using** *s-branch-s-branch-list.eq-iff* **by** *metis*
qed

ultimately show *?case* **using** *check-letI* **by** *auto*
qed(*auto+*)

lemma *fresh-subst-v-subst-b*:
fixes *x2*::*x* **and** *tm*::*'a*::{*has-subst-v*,*has-subst-b*} **and** *x*::*x*
assumes *supp tm* $\subseteq$ { *atom bv*, *atom x* } **and** *atom x2* ♯ *v*
shows *atom x2* ♯ $tm[bv::=b]_b[x::=v]_v$
using *assms* **proof**(*cases x2=x*)
case *True*
then show *?thesis* **using** *fresh-subst-v-if assms* **by** *blast*
next

case *False*
hence *atom x2* ♯ *tm* **using** *assms fresh-def fresh-at-base* **by** *force*
hence *atom x2* ♯ $tm[bv::=b]_b$ **using** *assms fresh-subst-if x-fresh-b False* **by** *force*
then show *?thesis* **using** *fresh-subst-v-if assms* **by** *auto*
qed

lemma *preservation-poly-app*:
assumes
Some (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*))) = *lookup-fun* Φ *f* **and**
(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
shows Θ ; Φ ; *B* ; *G* ; Δ ⊢ *ss* ⇐ τ ⟹ *B* = {||} ⟹ *G* = *GNil* ⟹ *ss* = *LET x* = (*AE-appP f b′ v*) *IN s* ⟹ Θ ; {||} $\vdash_{wf}$ *b′* ⟹
Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* : ($\tau 1'[bv1::=b']_{\tau b}[x1::=v]_{\tau v}$) = ($s1'[bv1::=b']_{sb}[x1::=v]_{sv}$) *IN s* ⇐ τ **and**
check-branch-s Θ Φ *ℬ GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ *ℬ* Γ Δ *tid dclist v css* τ ⟹ *True*
using *assms* **proof**(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v x1 rule*: *check-s-check-branch-s-check-branch-list.strong-*
case (*check-letI x2* Θ Φ *ℬ* Γ Δ *e* τ *z s2 b c*)

hence *eq*: *e* = (*AE-appP f b′ v*) **by** *simp*
hence ∗:Θ ; Φ ; {||} ;*GNil* ; Δ ⊢ (*AE-appP f b′ v*) ⇒ ⦃ *z* : *b* | *c* ⦄ **using** *check-letI* **by** *auto*

then obtain *x3 b3 c3* τ*3 s3 bv3* **where**
∗∗:Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ ∧ Θ $\vdash_{wf}$ Φ ∧ *Some* (*AF-fundef f* (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*))) = *lookup-fun* Φ *f* ∧
Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *x3* : $b3[bv3::=b']_{bb}$ | $c3[bv3::=b']_{cb}$ ⦄ ∧ *atom x3* ♯ *GNil* ∧ $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ = ⦃ *z* : *b* | *c* ⦄
∧ Θ ; {||} $\vdash_{wf}$ *b′*
using *infer-e-elims*(*21*)[*OF* ∗] *subst-defs* **by** *metis*

obtain *z3* **where** *z3*:⦃ *x3* : *b3* | *c3* ⦄ = ⦃ *z3* : *b3* | $c3[x3::=V\text{-}var\ z3]_{cv}$ ⦄ ∧ *atom z3* ♯ (*x3*, *v*,*c3*,*x1*,*c1*) **using** *obtain-fresh-z3* **by** *metis*

let *?ft* = (*AF-fun-typ x3* ($b3[bv3::=b']_{bb}$) ($c3[bv3::=b']_{cb}$) ($\tau 3[bv3::=b']_{\tau b}$) ($s3[bv3::=b']_{sb}$))

have ∗:*check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*))) **using** ∗∗ *check-letI lookup-fun-member* **by** *metis*

hence *ftq*:*check-funtypq* Θ Φ (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*)) **using** *check-fundef-elims* **by** *auto*

let *?ft* = *AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*)

have *sup*: *supp* τ*3* ⊆ { *atom x3*, *atom bv3*} ∧ *supp s3* ⊆ { *atom x3*, *atom bv3* }
using *wfPhi-f-poly-supp-t wfPhi-f-poly-supp-s* ∗∗ **by** *metis*

have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let2 x2* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ ($s3[bv3::=b']_{sb}[x3::=v]_{sv}$) *s2* ⇐ τ
proof
show ‹*atom x2* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$, $s3[bv3::=b']_{sb}[x3::=v]_{sv}$, τ)›
proof −

have *atom x2* ♯ $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$
using *fresh-subst-v-subst-b subst-v-τ-def subst-b-τ-def* ‹ *atom x2* ♯ *v*› *sup* **by** *fastforce*
moreover have *atom x2* ♯ $s3[bv3::=b']_{sb}[x3::=v]_{sv}$
using *fresh-subst-v-subst-b subst-v-s-def subst-b-s-def* ‹ *atom x2* ♯ *v*› *sup*
proof −
have ∀ *b. atom x2* = *atom x3* ∨ *atom x2* ♯ $s3[bv3::=b]_b$
by (*metis* (*no-types*) *check-letI.hyps*(*1*) *fresh-subst-sv-if*(*1*) *fresh-subst-v-subst-b insert-commute subst-v-s-def sup*)
then show *?thesis*
by (*metis check-letI.hyps*(*1*) *fresh-subst-sb-if fresh-subst-sv-if*(*1*) *has-subst-b-class.subst-b-fresh-x x-fresh-b*)
qed
ultimately show *?thesis* **using** *fresh-prodN check-letI* **by** *metis*
qed

show ‹ Θ; Φ; {||}; *GNil*; Δ ⊢ $s3[bv3::=b']_{sb}[x3::=v]_{sv}$ ⇐ $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$› **proof**(*rule fundef-poly-check*)
show ‹*check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3 τ3 s3*)))›
using ∗∗ *lookup-fun-member check-letI* **by** *metis*
show ‹Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *x3* : $b3[bv3::=b']_{bb}$ | $c3[bv3::=b']_{cb}$ ⦄› **using** ∗∗ **by** *metis*
show ‹ Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ › **using** ∗∗ **by** *metis*
show ‹ Θ ; {||} $\vdash_{wf}$ *b'* › **using** ∗∗ **by** *metis*
qed
show ‹ Θ ; Φ ; {||} ; (*x2*, *b-of* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$, *c-of* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ *x2*) $\#_\Gamma$ *GNil* ; Δ ⊢ *s2* ⇐ τ›
using *check-letI* ∗∗ *b-of.simps c-of.simps subst-defs* **by** *metis*
qed

moreover have *AS-let2 x2* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ ($s3[bv3::=b']_{sb}[x3::=v]_{sv}$) *s2* = *AS-let2 x* ($\tau 1'[bv1::=b']_{\tau b}[x1::=v]_{\tau v}$) ($s1'[bv1::=b']_{sb}[x1::=v]_{sv}$) *s* **proof** −
have ∗: [[*atom x2*]]*lst. s2* = [[*atom x*]]*lst. s* **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *auto*
moreover have $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ = $\tau 1'[bv1::=b']_{\tau b}[x1::=v]_{\tau v}$ **using** *fun-poly-ret-unique* ∗∗ *check-letI* **by** *metis*
moreover have $s3[bv3::=b']_{sb}[x3::=v]_{sv}$ = ($s1'[bv1::=b']_{sb}[x1::=v]_{sv}$) **using** *subst-v-flip-eq-two subst-v-s-def fun-poly-body-unique* ∗∗ *check-letI* **by** *metis*
ultimately show *?thesis* **using** *s-branch-s-branch-list.eq-iff* **by** *metis*
qed

ultimately show *?case* **using** *check-letI* **by** *auto*
qed(*auto*+)

lemma *check-s-plus*:
assumes Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* = (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN s'* ⇐ τ
shows Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* = (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *IN s'* ⇐ τ
proof −
obtain *t1* **where** *1*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)) ⇒ *t1*
using *assms check-s-elims* **by** *metis*
then obtain *z1* **where** *2*: *t1* = ⦃ *z1* : *B-int* | *CE-val* (*V-var z1*) == *CE-op Plus* ($[V\text{-}lit\ (L\text{-}num\ n1)]^{ce}$) ($[V\text{-}lit\ (L\text{-}num\ n2)]^{ce}$) ⦄
using *infer-e-plus* **by** *metis*

obtain *z2* **where** *3*: ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-lit* (*L-num* (*n1*+*n2*))) ⇒ ⦃ *z2* : *B-int* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* (*L-num* (*n1*+*n2*))) ⦄›
using *infer-v-form infer-e-valI infer-v-litI infer-l.intros infer-e-wf 1*
by (*simp add*: *fresh-GNil*)

let *?e* = (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)))

show *?thesis* **proof**(*rule subtype-let*)
show Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = *?e IN s*′ ⇐ τ **using** *assms* **by** *auto*
show Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *?e* ⇒ *t1* **using** *1* **by** *auto*
show Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ $[$ $[$ *L-num* (*n1* + *n2*) $]^v$ $]^e$ ⇒ ⦃ *z2* : *B-int* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* (*L-num* (*n1*+*n2*))) ⦄ **using** *3* **by** *auto*
show Θ ; {||} ; *GNil* ⊢ ⦃ *z2* : *B-int* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* (*L-num* (*n1*+*n2*))) ⦄ ≲ *t1* **using** *subtype-bop-arith*
by (*metis 1* ‹⋀*thesis*. (⋀*z1*. *t1* = ⦃ *z1* : *B-int* | $[$ $[$ *z1* $]^v$ $]^{ce}$ == $[$ *plus* $[$ $[$ *L-num n1* $]^v$ $]^{ce}$ $[$ $[$ *L-num n2* $]^v$ $]^{ce}$ $]^{ce}$ ⦄ ⟹ *thesis*) ⟹ *thesis*› *infer-e-wf*(*2*) *opp.distinct*(*1*) *type-for-lit.simps*(*3*))
qed

qed

lemma *check-s-leq*:
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN s*′ ⇐ τ
shows Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* = (*AE-val* (*V-lit* (*if* (*n1* ≤ *n2*) *then L-true else L-false*))) *IN s*′ ⇐ τ
proof −
obtain *t1* **where** *1*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)) ⇒ *t1*
using *assms check-s-elims* **by** *metis*
then obtain *z1* **where** *2*: *t1* = ⦃ *z1* : *B-bool* | *CE-val* (*V-var z1*) == *CE-op LEq* ($[$*V-lit* (*L-num n1*)$]^{ce}$) ($[$*V-lit* (*L-num n2*)$]^{ce}$) ⦄
using *infer-e-leq* **by** *auto*

obtain *z2* **where** *3*: ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) ⇒ ⦃ *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) ⦄›
using *infer-v-form infer-e-valI infer-v-litI infer-l.intros infer-e-wf 1*
fresh-GNil
by *simp*

show *?thesis* **proof**(*rule subtype-let*)
show ‹ Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-op LEq* $[$ *L-num n1* $]^v$ $[$ *L-num n2* $]^v$) *s*′ ⇐ τ› **using** *assms* **by** *auto*
show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-op LEq* $[$ *L-num n1* $]^v$ $[$ *L-num n2* $]^v$ ⇒ *t1*› **using** *1* **by** *auto*
show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ $[$ $[$ *if n1* ≤ *n2 then L-true else L-false* $]^v$ $]^e$ ⇒ ⦃ *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) ⦄› **using** *3* **by** *auto*
show ‹Θ ; {||} ; *GNil* ⊢ ⦃ *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) ⦄ ≲ *t1*›
using *subtype-bop-arith*[**where** *opp*=*LEq*] *check-s-wf assms 2*
by (*metis opp.distinct*(*1*) *subtype-bop-arith type-l-eq*)
qed
qed

lemma *check-s-eq*:
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = (*AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*))) *IN s'* ⇐ τ
shows Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* = (*AE-val* (*V-lit* (*if* (*n1* = *n2*) *then L-true else L-false*))) *IN s'* ⇐ τ
proof −
obtain *t1* **where** *1*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*)) ⇒ *t1*
using *assms check-s-elims* **by** *metis*
then obtain *z1* **where** *2*: *t1* = ⦃ *z1* : *B-bool* | *CE-val* (*V-var z1*) == *CE-op Eq* ($[$*V-lit* (*n1*)$]^{ce}$) ($[$*V-lit* (*n2*)$]^{ce}$) ⦄
using *infer-e-leq* **by** *auto*

obtain *z2* **where** *3*: ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-lit* ((*if* (*n1* = *n2*) *then L-true else L-false*))) ⇒ ⦃ *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* = *n2*) *then L-true else L-false*))) ⦄›
using *infer-v-form infer-e-valI infer-v-litI infer-l.intros infer-e-wf 1 fresh-GNil*
by *simp*

show *?thesis* **proof**(*rule subtype-let*)
show ‹ Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-op Eq* $[$ *n1* $]^v$ $[$ *n2* $]^v$) *s'* ⇐ τ› **using** *assms* **by** *auto*
show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-op Eq* $[$ *n1* $]^v$ $[$ *n2* $]^v$ ⇒ *t1*› **using** *1* **by** *auto*
show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ $[$ $[$ *if n1* = *n2 then L-true else L-false* $]^v$ $]^e$ ⇒ ⦃ *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* = *n2*) *then L-true else L-false*))) ⦄› **using** *3* **by** *auto*
show ‹Θ ; {||} ; *GNil* ⊢ ⦃ *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* = *n2*) *then L-true else L-false*))) ⦄ ≲ *t1*›
proof −
have ⦃ *z2* : *B-bool* | $[$ $[$ *z2* $]^v$ $]^{ce}$ == $[$ *eq* $[$ $[$ *n1* $]^v$ $]^{ce}$ $[$ $[$ *n2* $]^v$ $]^{ce}$ $]^{ce}$ ⦄ = *t1* **using** *2*
by (*metis* τ*-fresh-c fresh-opp-all infer-l-form2 infer-l-fresh ms-fresh-all*(*31*) *ms-fresh-all*(*33*) *obtain-fresh-z type-e-eq type-l-eq*)
moreover have Θ ; {||} $\vdash_{wf}$ *GNil* **using** *assms wfX-wfY* **by** *fastforce*
moreover have *base-for-lit n1* = *base-for-lit n2* **using** *1 infer-e-wf wfE-elims*(*12*) *wfV-elims*
by *metis*
ultimately show *?thesis* **using** *subtype-bop-eq*[*OF* ‹Θ ; {||} $\vdash_{wf}$ *GNil*›, *of n1 n2 z2*] **by** *auto*
qed
qed
qed

16.2.2 Operators

lemma *preservation-plus*:
assumes Θ; Φ; Δ ⊢ ⟨ δ , *LET x* = (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN s'* ⟩ ⇐ τ

shows Θ; Φ; Δ ⊢ ⟨ δ , *LET x* = (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *IN s'* ⟩ ⇐ τ
proof −

have *tt*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *s'* ⇐ τ **and** *dsim*: Θ ⊢ δ ∼ Δ **and** *fd*:(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *assms config-type-elims* **by** *blast+*

hence Θ; Φ; {||}; *GNil*; Δ ⊢*AS-let x* (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *s'* ⇐ τ **using** *check-s-plus assms* **by** *auto*

hence Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* ((*L-num* (*n1*+*n2*))))) *s'* ⟩ ⇐ τ **using** *dsim config-typeI*

fd **by** *presburger*
  then show *?thesis* **using** *dsim config-typeI*
    by (*meson order-refl*)
qed

lemma *preservation-leq*:
  assumes Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *s′* ⟩ ⇐ τ
  shows Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* (((*if* (*n1* ≤ *n2*) *then L-true else L-false*))))) *s′* ⟩ ⇐ τ
proof −

  have *tt*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *s′* ⇐ τ
and *dsim*: Θ ⊢ δ ∼ Δ **and** *fd*:(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
    using *assms config-type-elims* **by** *blast*+

  hence Θ; Φ; {||}; *GNil*; Δ ⊢*AS-let x* (*AE-val* (*V-lit* (((*if* (*n1* ≤ *n2*) *then L-true else L-false*))))) *s′*
⇐ τ **using** *check-s-leq assms* **by** *auto*

  hence Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* ((((*if* (*n1* ≤ *n2*) *then L-true else L-false*)))))) *s′* ⟩ ⇐
τ **using** *dsim config-typeI fd* **by** *presburger*
  then show *?thesis* **using** *dsim config-typeI*
    by (*meson order-refl*)
qed

lemma *preservation-eq*:
  assumes Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*))) *s′* ⟩ ⇐ τ
  shows Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* (((*if* (*n1* = *n2*) *then L-true else L-false*))))) *s′* ⟩ ⇐ τ
proof −

  have *tt*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*))) *s′* ⇐ τ **and** *dsim*: Θ ⊢
δ ∼ Δ **and** *fd*:(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
    using *assms config-type-elims* **by** *blast*+

  hence Θ; Φ; {||}; *GNil*; Δ ⊢*AS-let x* (*AE-val* (*V-lit* (((*if* (*n1* = *n2*) *then L-true else L-false*))))) *s′*
⇐ τ **using** *check-s-eq assms* **by** *auto*

  hence Θ; Φ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* ((((*if* (*n1* = *n2*) *then L-true else L-false*)))))) *s′* ⟩ ⇐
τ **using** *dsim config-typeI fd* **by** *presburger*
  then show *?thesis* **using** *dsim config-typeI*
    by (*meson order-refl*)
qed

16.2.3 Let Statements

lemma *subst-s-abs-lst*:
  fixes *s*::*s* **and** *sa*::*s* **and** *v′*::*v*
  assumes [[*atom x*]]*lst*. *s* = [[*atom xa*]]*lst*. *sa* **and** *atom xa* ♯ *v* ∧ *atom x* ♯ *v*
  shows $s[x::=v]_{sv}$ = $sa[xa::=v]_{sv}$
proof −
  obtain *c′*::*x* **where** *cdash*: *atom c′* ♯ (*v*, *x*, *xa*, *s*, *sa*) **using** *obtain-fresh* **by** *blast*
  moreover have (*x* ↔ *c′*) · *s* = (*xa* ↔ *c′*) · *sa* **proof** −
    have *atom c′* ♯ (*s*, *sa*) ∧ *atom c′* ♯ (*x*, *xa*, *s*, *sa*) **using** *cdash* **by** *auto*
    thus *?thesis* **using** *assms* **by** *auto*
  qed

ultimately show *?thesis* **using** *assms*
  using *subst-sv-flip* **by** *auto*
qed

lemma *check-let-val*:
 fixes *v*::*v* **and** *s*::*s*
 shows Θ ; Φ ; *B* ; *G* ; Δ ⊢ *ss* ⇐ τ ⟹ *B* = {||} ⟹ *G* = *GNil* ⟹
  ss = *AS-let x* (*AE-val v*) *s* ∨ *ss* = *AS-let2 x t* (*AS-val v*) *s* ⟹ Θ; Φ; {||}; *GNil*; Δ ⊢ ($s[x::=v]_{sv}$) ⇐ τ **and**
  check-branch-s Θ Φ ℬ *GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
  check-branch-list Θ Φ ℬ Γ Δ *tid dclist v css* τ ⟹ *True*
proof(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
 case (*check-letI x1* Θ Φ ℬ Γ Δ *e* τ *z s1 b c*)
 hence ∗:*e* = *AE-val v* **by** *auto*
 let *?G* = (*x1*, *b*, $c[z::=V\text{-}var\ x1]_{cv}$) $\#_\Gamma$ Γ
 have Θ ; Φ ; ℬ ; $?G[x1::=v]_{\Gamma v}$; $\Delta[x1::=v]_{\Delta v}$ ⊢ $s1[x1::=v]_{sv}$ ⇐ $\tau[x1::=v]_{\tau v}$
 proof(*rule subst-infer-check-s*(*1*))
  show ∗∗:‹ Θ ; ℬ ; Γ ⊢ *v* ⇒ ⦃ *z* : *b* | *c* ⦄› **using** *infer-e-elims check-letI* ∗ **by** *fast*
  thus ‹Θ ; ℬ ; Γ ⊢ ⦃ *z* : *b* | *c* ⦄ ≲ ⦃ *z* : *b* | *c* ⦄› **using** *subtype-reflI infer-v-wf* **by** *metis*
  show ‹*atom z* ♯ (*x1*, *v*)› **using** *check-letI fresh-Pair* **by** *auto*
  show ‹Θ ; Φ ; ℬ ; (*x1*, *b*, $c[z::=V\text{-}var\ x1]_{cv}$) $\#_\Gamma$ Γ ; Δ ⊢ *s1* ⇐ τ› **using** *check-letI subst-defs* **by** *auto*
  show (*x1*, *b*, $c[z::=V\text{-}var\ x1]_{cv}$) $\#_\Gamma$ Γ = *GNil* @ (*x1*, *b*, $c[z::=V\text{-}var\ x1]_{cv}$) $\#_\Gamma$ Γ **by** *auto*
 qed

 hence Θ ; Φ ; ℬ ; Γ ; Δ ⊢ $s1[x1::=v]_{sv}$ ⇐ τ **using** *check-letI* **by** *auto*
 moreover have $s1[x1::=v]_{sv}$ = $s[x::=v]_{sv}$
  by (*metis* (*full-types*) *check-letI fresh-GNil infer-e-elims*(*7*) *s-branch-s-branch-list.distinct s-branch-s-branch-list.eq-iff*(

   subst-s-abs-lst wfG-x-fresh-in-v-simple)

 ultimately show *?case* **using** *check-letI* **by** *simp*
next
 case (*check-let2I x1* Θ Φ ℬ Γ Δ *t s1* τ *s2*)

 hence *s1eq*:*s1* = *AS-val v* **by** *auto*
 let *?G* = (*x1*, *b-of t*, *c-of t x1*) $\#_\Gamma$ Γ
 obtain *z*::*x* **where** ∗:*atom z* ♯ (*x1* , *v*,*t*) **using** *obtain-fresh-z* **by** *metis*
 hence *teq*:*t* = ⦃ *z* : *b-of t* | *c-of t z* ⦄ **using** *b-of-c-of-eq* **by** *auto*
 have Θ ; Φ ; ℬ ; $?G[x1::=v]_{\Gamma v}$; $\Delta[x1::=v]_{\Delta v}$ ⊢ $s2[x1::=v]_{sv}$ ⇐ $\tau[x1::=v]_{\tau v}$
 proof(*rule subst-check-check-s*(*1*))
  obtain *t′* **where** Θ ; ℬ ; Γ ⊢ *v* ⇒ *t′* ∧ Θ ; ℬ ; Γ ⊢ *t′* ≲ *t* **using** *check-s-elims*(*1*) *check-let2I*(*10*) *s1eq* **by** *auto*
  thus ∗∗:‹ Θ ; ℬ ; Γ ⊢ *v* ⇐ ⦃ *z* : *b-of t* | *c-of t z* ⦄› **using** *check-v.intros teq* **by** *auto*
  show *atom z* ♯ (*x1*, *v*) **using** ∗ **by** *auto*
  show ‹ Θ ; Φ ; ℬ ; (*x1*, *b-of t*, *c-of t x1*) $\#_\Gamma$ Γ ; Δ ⊢ *s2* ⇐ τ› **using** *check-let2I* **by** *auto*
  show (*x1*, *b-of t* , *c-of t x1*) $\#_\Gamma$ Γ = *GNil* @ (*x1*, *b-of t*, $(c\text{-}of\ t\ z)[z::=V\text{-}var\ x1]_{cv}$) $\#_\Gamma$ Γ **using** *append-g.simps c-of-switch* ∗ *fresh-prodN* **by** *metis*
 qed

 hence Θ ; Φ ; ℬ ; Γ ; Δ ⊢ $s2[x1::=v]_{sv}$ ⇐ τ **using** *check-let2I* **by** *auto*
 moreover have $s2[x1::=v]_{sv}$ = $s[x::=v]_{sv}$ **using**

check-let2I fresh-GNil check-s-elims s-branch-s-branch-list.distinct s-branch-s-branch-list.eq-iff subst-s-abs-lst wfG-x-fresh-in-v-simple
proof −
have *AS-let2 x t* (*AS-val v*) *s* = *AS-let2 x1 t s1 s2*
by (*metis check-let2I.prems*(*3*) *s-branch-s-branch-list.distinct s-branch-s-branch-list.eq-iff*(*3*))
then show *?thesis*
by (*metis* (*no-types*) *check-let2I check-let2I.prems*(*2*) *check-s-elims*(*1*) *fresh-GNil s-branch-s-branch-list.eq-iff*(*3*) *subst-s-abs-lst wfG-x-fresh-in-v-simple*)
qed

ultimately show *?case* **using** *check-let2I* **by** *simp*
qed(*auto*+)

lemma *preservation-let-val*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta\ ,\ $*AS-let x* (*AE-val v*) *s* $\rangle \Leftarrow \tau \lor \Theta$; Φ; $\Delta \vdash \langle\ \delta\ ,\ $*AS-let2 x t* (*AS-val v*) *s* $\rangle \Leftarrow \tau$ (**is** *?A* $\lor$ *?B*)
shows $\exists \Delta'. \Theta; \Phi; \Delta' \vdash \langle\ \delta\ ,\ s[x::=v]_{sv}\ \rangle \Leftarrow \tau\ \land \Delta \sqsubseteq \Delta'$
proof −
have *tt*: $\Theta \vdash \delta \sim \Delta$ **and** *fd*: ($\forall$*fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*)
using *assms* **by** *blast*+

have *?A* $\lor$ *?B* **using** *assms* **by** *auto*
then have Θ; Φ; {||}; *GNil*; $\Delta \vdash s[x::=v]_{sv} \Leftarrow \tau$
proof
assume Θ; Φ; $\Delta \vdash \langle\ \delta\ ,\ $*AS-let x* (*AE-val v*) *s* $\rangle \Leftarrow \tau$
hence ∗ : Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *AS-let x* (*AE-val v*) *s* $\Leftarrow \tau$ **by** *blast*
thus *?thesis* **using** *check-let-val* **by** *simp*
next
assume Θ; Φ; $\Delta \vdash \langle\ \delta\ ,\ $*AS-let2 x t* (*AS-val v*) *s* $\rangle \Leftarrow \tau$
hence ∗ : Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *AS-let2 x t* (*AS-val v*) *s* $\Leftarrow \tau$ **by** *blast*
thus *?thesis* **using** *check-let-val* **by** *simp*
qed

thus *?thesis* **using** *tt config-typeI fd*
order-refl **by** *metis*
qed

lemma *check-s-fst-snd*:
assumes *fst-snd* = *AE-fst* $\land$ *v*=*v1* $\lor$ *fst-snd* = *AE-snd* $\land$ *v*=*v2*
and Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) $s' \Leftarrow \tau$
shows Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *AS-let x* (*AE-val v*) $s' \Leftarrow \tau$
proof −
have *1*: ‹ Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) $s' \Leftarrow \tau$› **using** *assms* **by** *auto*

then obtain *t1* **where** *2*:Θ; Φ; {||}; *GNil*; $\Delta \vdash$ (*fst-snd* (*V-pair v1 v2*)) $\Rightarrow$ *t1* **using** *check-s-elims* **by** *auto*

show *?thesis* **using** *subtype-let 1 2 assms*
by (*meson infer-e-fst-pair infer-e-snd-pair*)
qed

lemma *preservation-fst-snd*:

assumes Θ; Φ; $\Delta \vdash \langle\ \delta\ ,\ LET\ x = (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ IN\ s'\ \rangle \Leftarrow \tau$ **and**
fst-snd = *AE-fst* ∧ *v*=*v1* ∨ *fst-snd* = *AE-snd* ∧ *v*=*v2*
shows $\exists\,\Delta'.\ \Theta;\ \Phi;\ \Delta \vdash \langle\ \delta\ ,\ LET\ x = (AE\text{-}val\ v)\ IN\ s'\ \rangle \Leftarrow \tau\ \ \wedge \Delta \sqsubseteq \Delta'$
proof −
have *tt*: Θ; Φ; {||}; *GNil*; $\Delta\ \vdash$ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) $s' \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta$ **using** *assms* **by** *blast*
hence *t2*: Θ; Φ; {||}; *GNil*; $\Delta\ \vdash$ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) $s' \Leftarrow \tau$ **by** *auto*

moreover have ∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd* **using** *assms config-type-elims* **by** *auto*
ultimately show *?thesis* **using** *config-typeI order-refl tt assms check-s-fst-snd* **by** *metis*
qed

inductive-cases *check-branch-s-elims2*[*elim*!]:
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *cons* ; *const* ; *v* ⊢ *cs* ⇐ τ

lemmas *freshers* = *freshers atom-dom.simps toSet.simps fresh-def x-not-in-b-set*
declare *freshers* [*simp*]

lemma *subtype-eq-if*:
fixes *t*::τ **and** *va*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ⦃ *z* : *b-of t* | *c-of t z* ⦄ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ⦃ *z* : *b-of t* | *c IMP c-of t z* ⦄
shows Θ ; $\mathcal{B}$; Γ ⊢ ⦃ *z* : *b-of t* | *c-of t z* ⦄ ≲ ⦃ *z* : *b-of t* | *c IMP c-of t z* ⦄
proof −
obtain *x*::*x* **where** *xf*:*atom x* ♯ ((Θ, $\mathcal{B}$, Γ, *z*, *c-of t z*, *z*, *c IMP c-of t z*),*c*) **using** *obtain-fresh* **by** *metis*

moreover have Θ ; $\mathcal{B}$; $(x,\ b\text{-}of\ t,\ (c\text{-}of\ t\ z)[z{::=}[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma\ \models (c\ \ IMP\ \ c\text{-}of\ t\ z\ )[z{::=}[\ x\ ]^v]_{cv}$
unfolding *subst-cv.simps*
proof(*rule valid-eq-imp*)

have Θ ; $\mathcal{B}$; $(x,\ b\text{-}of\ t,\ (c\text{-}of\ t\ z)[z{::=}[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma\ \ \vdash_{wf} (c\ \ IMP\ \ (c\text{-}of\ t\ z))[z{::=}[\ x\ ]^v]_v$
apply(*rule wfT-wfC-cons*)
apply(*simp add*: *assms*, *simp add*: *assms*, *unfold fresh-prodN*)
using *xf fresh-prodN* **by** *metis*+
thus Θ ; $\mathcal{B}$; $(x,\ b\text{-}of\ t,\ (c\text{-}of\ t\ z)[z{::=}[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma\ \ \vdash_{wf}\ c[z{::=}[\ x\ ]^v]_{cv}\ \ IMP\ \ (c\text{-}of\ t\ z)[z{::=}[\ x\ ]^v]_{cv}$
using *subst-cv.simps subst-defs* **by** *auto*
qed

ultimately show *?thesis* **using** *subtype-baseI assms fresh-Pair subst-defs* **by** *metis*
qed

lemma *subtype-eq-if-τ*:
fixes *t*::τ **and** *va*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *t* **and** Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ⦃ *z* : *b-of t* | *c IMP c-of t z* ⦄ **and** *atom z* ♯ *t*
shows Θ ; $\mathcal{B}$; Γ ⊢ *t* ≲ ⦃ *z* : *b-of t* | *c IMP c-of t z* ⦄
proof −
have *t* = ⦃ *z* : *b-of t* | *c-of t z* ⦄ **using** *b-of-c-of-eq assms* **by** *auto*
thus *?thesis* **using** *subtype-eq-if assms c-of.simps b-of.simps* **by** *metis*
qed

lemma *valid-conj*:

assumes Θ ; $\mathcal{B}$; $\Gamma \models$ *c1* **and** Θ ; $\mathcal{B}$; $\Gamma \models$ *c2*
shows Θ ; $\mathcal{B}$; $\Gamma \models$ *c1 AND c2*
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *c1 AND c2* › **using** *valid.simps wfC-conjI assms* **by** *auto*
show ‹$\forall i.\ \Theta ; \Gamma \vdash i \wedge i \models \Gamma \longrightarrow i \models$ *c1 AND c2* ›
proof(*rule+*)
fix *i*
assume $*$:$\Theta ; \Gamma \vdash i \wedge i \models \Gamma$
thus i ⟦ *c1* ⟧ ~ *True* **using** *assms valid.simps*
using *is-satis.cases* **by** *blast*
show i ⟦ *c2* ⟧ ~ *True* **using** *assms valid.simps*
using *is-satis.cases* $*$ **by** *blast*
qed
qed

16.2.4 Other Statements

lemma *check-if*:
fixes *s'*::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows Θ; Φ; *B*; *G*; $\Delta \vdash$ *s'* $\Leftarrow \tau \Longrightarrow$ *s'* = *IF* (*V-lit ll*) *THEN s1 ELSE s2* $\Longrightarrow$
Θ ; {||} ; *GNil* $\vdash_{wf} \tau \Longrightarrow$ *G* = *GNil* $\Longrightarrow$ *B* = {||} $\Longrightarrow$ *ll* = *L-true* $\wedge$ *s* = *s1* $\vee$ *ll* = *L-false* $\wedge$ *s* = *s2* $\Longrightarrow$
Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *s* $\Leftarrow \tau$ **and**
check-branch-s Θ Φ {||} *GNil* Δ *tid dc const v cs* $\tau \Longrightarrow$ *True* **and**
check-branch-list Θ Φ {||} Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *True*
proof(*nominal-induct* τ **and** τ **and** τ *rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
obtain *z'* **where** *teq*: τ = ⦃ *z'* : *b-of* τ | *c-of* τ *z'* ⦄ $\wedge$ *atom z'* ♯ (*z*,τ) **using** *obtain-fresh-z-c-of* **by** *metis*
hence *ceq*: (*c-of* τ *z'*)[*z'*::=[*z*]^{v}]$_{cv}$ = (*c-of* τ *z*) **using** *c-of-switch fresh-Pair* **by** *metis*
have *zf*: *atom z* ♯ *c-of* τ *z'*
using *c-of-fresh check-ifI teq fresh-Pair fresh-at-base* **by** *metis*
hence *1*:Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *s* $\Leftarrow$ ⦃ *z* : *b-of* τ | *CE-val* (*V-lit ll*) == *CE-val* (*V-lit ll*) *IMP c-of* τ *z* ⦄ **using** *check-ifI* **by** *auto*
moreover have *2*:Θ ; {||} ; *GNil* $\vdash$ (⦃ *z* : *b-of* τ | *CE-val* (*V-lit ll*) == *CE-val* (*V-lit ll*) *IMP c-of* τ *z* ⦄) $\lesssim$ τ
proof −
have Θ ; {||} ; *GNil* $\vdash_{wf}$ (⦃ *z* : *b-of* τ | *CE-val* (*V-lit ll*) == *CE-val* (*V-lit ll*) *IMP c-of* τ *z* ⦄) **using** *check-ifI check-s-wf* **by** *auto*
moreover have Θ ; {||} ; *GNil* $\vdash_{wf}$ τ **using** *check-s-wf check-ifI* **by** *auto*
ultimately show *?thesis* **using** *subtype-if-simp*[*of* Θ {||} *z b-of* τ *ll c-of* τ *z' z'*] **using** *teq ceq zf subst-defs* **by** *metis*
qed
ultimately show *?case* **using** *check-s-supertype*(*1*) *check-ifI* **by** *metis*
qed(*auto+*)

lemma *preservation-if*:
assumes Θ; Φ; $\Delta \vdash$ ⟨ δ , *IF* (*V-lit ll*) *THEN s1 ELSE s2* ⟩ $\Leftarrow \tau$ **and**
ll = *L-true* $\wedge$ *s* = *s1* $\vee$ *ll* = *L-false* $\wedge$ *s* = *s2*
shows Θ; Φ; $\Delta \vdash$ ⟨δ, *s*⟩ $\Leftarrow \tau$ $\wedge$ *setD* $\Delta \subseteq$ *setD* Δ
proof −
have $*$: Θ; Φ; {||}; *GNil*; $\Delta \vdash$ *AS-if* (*V-lit ll*) *s1 s2* $\Leftarrow \tau \wedge$ ($\forall$*fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*)
using *assms config-type-elims* **by** *metis*

hence Θ; Φ; $\{||\}$; *GNil*; Δ $\vdash$ $s \Leftarrow \tau$ **using** *check-s-wf check-if assms* **by** *metis*
hence Θ; Φ; $\Delta \vdash \langle\delta, s\rangle \Leftarrow \tau$ $\wedge$ *setD* $\Delta \subseteq$ *setD* Δ **using** *config-typeI* $*$
using *assms*(*1*) **by** *blast*
thus *?thesis* **by** *blast*
qed

lemma *wfT-conj*:
assumes Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $\{\!|$ $z : b$ | *c1* $|\!\}$ **and** Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $\{\!|$ $z : b$ | *c2* $|\!\}$
shows Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $\{\!|$ $z : b$ | *c1 AND c2* $|\!\}$
proof
show ‹*atom z* $\sharp$ (Θ, $\mathcal{B}$, *GNil*)›
apply(*unfold fresh-prodN*, *intro conjI*)
using *wfTh-fresh wfT-wf assms* **apply** *metis*
using *fresh-GNil x-not-in-b-set fresh-def* **by** *metis+*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfT-elims assms* **by** *metis*
have Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *c1* **using** *wfT-wfC fresh-GNil assms* **by** *auto*
moreover have Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *c2* **using** *wfT-wfC fresh-GNil assms* **by** *auto*
ultimately show ‹ Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *c1 AND c2* › **using** *wfC-conjI* **by** *auto*
qed

lemma *subtype-conj*:
assumes Θ ; $\mathcal{B}$; *GNil* $\vdash$ *t* $\lesssim$ $\{\!|$ $z : b$ | *c1* $|\!\}$ **and** Θ ; $\mathcal{B}$; *GNil* $\vdash$ *t* $\lesssim$ $\{\!|$ $z : b$ | *c2* $|\!\}$
shows Θ ; $\mathcal{B}$; *GNil* $\vdash$ $\{\!|$ $z : b$ | *c-of t z* $|\!\}$ $\lesssim$ $\{\!|$ $z : b$ | *c1 AND c2* $|\!\}$
proof −
have *beq*: *b-of t* = *b* **using** *subtype-eq-base2 b-of.simps assms* **by** *metis*
obtain *x*::*x* **where** *x*:‹*atom x* $\sharp$ (Θ, $\mathcal{B}$, *GNil*, *z*, *c-of t z*, *z*, *c1 AND c2*)› **using** *obtain-fresh* **by** *metis*
thus *?thesis* **proof**
have *atom z* $\sharp$ *t* **using** *subtype-wf wfT-supp fresh-def x-not-in-b-set atom-dom.simps toSet.simps assms dom.simps* **by** *fastforce*
hence *t*:*t* = $\{\!|$ z : *b-of t* | *c-of t z* $|\!\}$ **using** *b-of-c-of-eq* **by** *auto*
thus ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $\{\!|$ $z : b$ | *c-of t z* $|\!\}$ › **using** *subtype-wf beq assms* **by** *auto*

show ‹Θ ; $\mathcal{B}$; (*x*, *b*, (*c-of t z*)[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ *GNil* $\models$ (*c1 AND c2*)[*z*::=[*x*]^v]$_v$ ›
proof −
have ‹Θ ; $\mathcal{B}$; (*x*, *b*, (*c-of t z*)[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ *GNil* $\models$ *c1*[*z*::=[*x*]^v]$_v$ ›
proof(*rule subtype-valid*)
show ‹Θ ; $\mathcal{B}$; *GNil* $\vdash$ *t* $\lesssim$ $\{\!|$ $z : b$ | *c1* $|\!\}$› **using** *assms* **by** *auto*
show ‹*atom x* $\sharp$ *GNil*› **using** *fresh-GNil* **by** *auto*
show ‹*t* = $\{\!|$ $z : b$ | *c-of t z* $|\!\}$› **using** *t beq* **by** *auto*
show ‹$\{\!|$ $z : b$ | *c1* $|\!\}$ = $\{\!|$ $z : b$ | *c1* $|\!\}$› **by** *auto*
qed
moreover have ‹Θ ; $\mathcal{B}$; (*x*, *b*, (*c-of t z*)[*z*::=[*x*]^v]$_v$) $\#_\Gamma$ *GNil* $\models$ *c2*[*z*::=[*x*]^v]$_v$ ›
proof(*rule subtype-valid*)
show ‹Θ ; $\mathcal{B}$; *GNil* $\vdash$ *t* $\lesssim$ $\{\!|$ $z : b$ | *c2* $|\!\}$› **using** *assms* **by** *auto*
show ‹*atom x* $\sharp$ *GNil*› **using** *fresh-GNil* **by** *auto*
show ‹*t* = $\{\!|$ $z : b$ | *c-of t z* $|\!\}$› **using** *t beq* **by** *auto*
show ‹$\{\!|$ $z : b$ | *c2* $|\!\}$ = $\{\!|$ $z : b$ | *c2* $|\!\}$› **by** *auto*
qed
ultimately show *?thesis* **unfolding** *subst-cv.simps subst-v-c-def* **using** *valid-conj* **by** *metis*
qed
thus ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $\{\!|$ $z : b$ | *c1 AND c2* $|\!\}$ › **using** *subtype-wf wfT-conj assms* **by** *auto*

qed
qed

lemma *infer-v-conj*:
assumes Θ ; $\mathcal{B}$; *GNil* $\vdash v \Leftarrow \{\!\mid z : b \mid c1 \mid\!\}$ **and** Θ ; $\mathcal{B}$; *GNil* $\vdash v \Leftarrow \{\!\mid z : b \mid c2 \mid\!\}$
shows Θ ; $\mathcal{B}$; *GNil* $\vdash v \Leftarrow \{\!\mid z : b \mid c1\ AND\ c2 \mid\!\}$
proof −
obtain *t1* **where** *t1*: Θ ; $\mathcal{B}$; *GNil* $\vdash v \Rightarrow t1 \wedge \Theta$; $\mathcal{B}$; *GNil* $\vdash t1 \lesssim \{\!\mid z : b \mid c1 \mid\!\}$
using *assms check-v-elims* **by** *metis*
obtain *t2* **where** *t2*: Θ ; $\mathcal{B}$; *GNil* $\vdash v \Rightarrow t2 \wedge \Theta$; $\mathcal{B}$; *GNil* $\vdash t2 \lesssim \{\!\mid z : b \mid c2 \mid\!\}$
using *assms check-v-elims* **by** *metis*
have *teq*: $t1 = \{\!\mid z : b \mid c\text{-}of\ t1\ z \mid\!\}$ **using** *subtype-eq-base2 b-of.simps*
by (*metis* (*full-types*) *b-of-c-of-eq fresh-GNil infer-v-t-wf t1 wfT-x-fresh*)
have $t1 = t2$ **using** *infer-v-uniqueness t1 t2* **by** *auto*
hence Θ ; $\mathcal{B}$; *GNil* $\vdash \{\!\mid z : b \mid c\text{-}of\ t1\ z \mid\!\} \lesssim \{\!\mid z : b \mid c1\ AND\ c2 \mid\!\}$ **using** *subtype-conj t1 t2* **by** *simp*
hence Θ ; $\mathcal{B}$; *GNil* $\vdash t1 \lesssim \{\!\mid z : b \mid c1\ AND\ c2 \mid\!\}$ **using** *teq* **by** *auto*
thus *?thesis* **using** *t1* **using** *check-v.intros* **by** *auto*
qed

lemma *wfG-conj*:
fixes *c1*::*c*
assumes Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c1 AND c2*) $\#_\Gamma$ Γ
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c1*) $\#_\Gamma$ Γ
proof(*cases c1* $\in$ {*TRUE*, *FALSE*})
case *True*
then show *?thesis* **using** *assms wfG-cons2I wfG-elims wfX-wfY* **by** *metis*
next
case *False*
then show *?thesis* **using** *assms wfG-cons1I wfG-elims wfX-wfY wfC-elims*
by (*metis wfG-elim2*)
qed

lemma *check-match*:
fixes *s′*::*s* **and** *s*::*s* **and** *css*::*branch-list* **and** *cs*::*branch-s*
shows Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ $s \Leftarrow \tau \Longrightarrow$ *True* **and**
Θ ; Φ ; *B* ; *G* ; Δ ; *tid* ; *dc* ; *const* ; *vcons* $\vdash$ *cs* $\Leftarrow \tau \Longrightarrow$
vcons = *V-cons tid dc v* $\Longrightarrow$ *B* = {||} $\Longrightarrow$ *G* = *GNil* $\Longrightarrow$ *cs* = (*dc x′* $\Rightarrow$ *s′*) $\Longrightarrow$
Θ ; {||} ; *GNil* $\vdash$ *v* $\Leftarrow$ *const* $\Longrightarrow$
Θ; Φ; {||}; *GNil*; Δ $\vdash$ $s'[x'::=v]_{sv} \Leftarrow \tau$ **and**
Θ ; Φ ; *B* ; *G* ; Δ ; *tid* ; *dclist* ; *vcons* $\vdash$ *css* $\Leftarrow \tau \Longrightarrow$ *distinct* (*map fst dclist*) $\Longrightarrow$
vcons = *V-cons tid dc v* $\Longrightarrow$ *B* = {||} $\Longrightarrow$ (*dc*, *const*) $\in$ *set dclist* $\Longrightarrow$ *G* = *GNil* $\Longrightarrow$
Some (*AS-branch dc x′ s′*) = *lookup-branch dc css* $\Longrightarrow$ Θ ; {||} ; *GNil* $\vdash$ *v* $\Leftarrow$ *const* $\Longrightarrow$
Θ; Φ; {||}; *GNil*; Δ $\vdash$ $s'[x'::=v]_{sv} \Leftarrow \tau$
proof(*nominal-induct* τ **and** τ **and** τ *avoiding*: *x′ v* *rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid consa consta va cs* τ *dclist cssa*)

then obtain *xa* **and** *sa* **where** *cseq*:*cs* = *AS-branch consa xa sa* **using** *check-branch-s-elims2*[*OF check-branch-list-consI*(*1*)] **by** *metis*

show *?case* **proof**(*cases dc* = *consa*)
case *True*
hence *cs* = *AS-branch consa x′ s′* **using** *check-branch-list-consI cseq*

by (*metis lookup-branch.simps*(*2*) *option.inject*)
moreover have *const* = *consta* **using** *check-branch-list-consI distinct.simps*
by (*metis True dclist-distinct-unique list.set-intros*(*1*))
moreover have *va* = *V-cons tid consa v* **using** *check-branch-list-consI True* **by** *auto*
ultimately show *?thesis* **using** *check-branch-list-consI* **by** *auto*
next
case *False*
hence *Some* (*AS-branch dc x′ s′*) = *lookup-branch dc cssa* **using** *lookup-branch.simps*(*2*) *check-branch-list-consI*(*10*) *cseq* **by** *auto*
moreover have (*dc*, *const*) ∈ *set dclist* **using** *check-branch-list-consI False* **by** *simp*
ultimately show *?thesis* **using** *check-branch-list-consI* **by** *auto*
qed

next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid cons const va cs* τ)
hence *cs* = *AS-branch cons x′ s′* **using** *lookup.simps check-branch-list-finalI lookup-branch.simps option.inject*
by (*metis map-of.simps*(*1*) *map-of-Cons-code*(*2*) *option.distinct*(*1*) *s-branch-s-branch-list.exhaust*(*2*) *weak-map-of-SomeI*)
then show *?case* **using** *check-branch-list-finalI* **by** *auto*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons va s*)

Supporting facts here to make the main body of the proof concise

have *xf*:*atom x* ♯ τ **proof** −
have *supp* τ ⊆ *supp* $\mathcal{B}$ **using** *wf-supp*(*4*) *check-branch-s-branchI atom-dom.simps toSet.simps dom.simps* **by** *fastforce*
thus *?thesis* **using** *fresh-def x-not-in-b-set* **by** *blast*
qed

hence $\tau[x::=v]_{\tau v} = \tau$ **using** *forget-subst-v subst-v-τ-def* **by** *auto*
have $\Delta[x::=v]_{\Delta v} = \Delta$ **using** *forget-subst-dv wfD-x-fresh fresh-GNil check-branch-s-branchI* **by** *metis*

have *supp v* = {} **using** *check-branch-s-branchI check-v-wf wfV-supp-nil* **by** *metis*
hence *supp va* = {} **using** ‹ *va* = *V-cons tid cons v*› *v.supp pure-supp* **by** *auto*

let *?G* = (*x*, *b-of const*, [*va*]^{ce} == [*V-cons tid cons* [*x*]^{v}]^{ce} *AND c-of const x*) $\#_{\Gamma}$ Γ
obtain *z*::*x* **where** *z*: *const* = ⦃ *z* : *b-of const* | *c-of const z* ⦄ ∧ *atom z* ♯ (*x′*, *v*,*x*,*const*,*va*)
using *obtain-fresh-z-c-of* **by** *metis*

have *vt*: ‹Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇐ ⦃ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce} *AND c-of const z* ⦄›
proof(*rule infer-v-conj*)
obtain *t′* **where** *t*: Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇒ *t′* ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ *t′* ≲ *const*
using *check-v-elims check-branch-s-branchI* **by** *metis*
show Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇐ ⦃ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce} ⦄
proof(*rule check-v-top*)

show Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ ⦃ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce} ⦄

proof(*rule wfG-wfT*)
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b-of const*, ([*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce})$[z::=[\ x\ ]^{v}]_{cv}$) $\#_{\Gamma}$

GNil ›
proof –
have *1*: $va[z::=[\ x\ ]^v]_{vv}$ = *va* **using** *forget-subst-v subst-v-v-def z fresh-prodN* **by** *metis*
moreover have *2*: Θ ; *ℬ* $\vdash_{wf}$ (*x*, *b-of const*, [*va*]^{ce} == [*V-cons tid cons* [*x*]^v]^{ce} *AND c-of const x*) $\#_\Gamma$ *GNil*
using *check-branch-s-branchI*(*17*)[*THEN check-s-wf*] ‹Γ = *GNil*› **by** *auto*
moreover hence Θ ; *ℬ* $\vdash_{wf}$ (*x*, *b-of const*, [*va*]^{ce} == [*V-cons tid cons* [*x*]^v]^{ce}) $\#_\Gamma$ *GNil*
using *wfG-conj* **by** *metis*
ultimately show *?thesis*
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps* **by** *auto*
qed
show ‹*atom x* ♯ ([*va*]^{ce} == [*V-cons tid cons* [*z*]^v]^{ce})› **unfolding** *c.fresh ce.fresh v.fresh*
apply(*intro conjI*)
using *check-branch-s-branchI fresh-at-base z freshers* **apply** *simp*
using *check-branch-s-branchI fresh-at-base z freshers* **apply** *simp*
using *pure-supp* **apply** *force*
using *z fresh-x-neq fresh-prod5* **by** *metis*
qed
show ‹[*va*]^{ce} = [*V-cons tid cons* [*z*]^v]$^{ce}[z::=v]_{cev}$›
using ‹ *va* = *V-cons tid cons v*› *subst-cev.simps subst-vv.simps* **by** *auto*
show ‹ Θ ; *ℬ* ; *GNil* ⊢ *v* ⇐ *const* › **using** *check-branch-s-branchI* **by** *auto*
show *supp* [*va*]^{ce} ⊆ *supp ℬ* **using** ‹*supp va* = {}› *ce.supp* **by** *simp*
qed
show Θ ; *ℬ* ; *GNil* ⊢ *v* ⇐ ⦃ *z* : *b-of const* | *c-of const z* ⦄
using *check-branch-s-branchI z* **by** *auto*
qed

Main body of proof for this case

have Θ ; Φ ; *ℬ* ; (*?G*)$[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ $s[x::=v]_{sv}$ ⇐ $\tau[x::=v]_{\tau v}$
proof(*rule subst-check-check-s*)
show ‹Θ ; *ℬ* ; *GNil* ⊢ *v* ⇐ ⦃ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^v]^{ce} *AND c-of const z* ⦄› **using** *vt* **by** *auto*
show ‹*atom z* ♯ (*x*, *v*)› **using** *z fresh-prodN* **by** *auto*
show ‹ Θ ; Φ ; *ℬ* ; *?G* ; Δ ⊢ *s* ⇐ τ›
using *check-branch-s-branchI* **by** *auto*

show ‹ *?G* = *GNil* @ (*x*, *b-of const*, ([*va*]^{ce} == [*V-cons tid cons* [*z*]^v]^{ce} *AND c-of const z*)$[z::=[\ x\ ]^v]_{cv}$) $\#_\Gamma$ *GNil*›
proof –
have $va[z::=[\ x\ ]^v]_{vv}$ = *va* **using** *forget-subst-v subst-v-v-def z fresh-prodN* **by** *metis*
moreover have (*c-of const z*)$[z::=[\ x\ ]^v]_{cv}$ = *c-of const x*
using *c-of-switch*[*of z const x*] *z fresh-prodN* **by** *metis*
ultimately show *?thesis*
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps append-g.simps*
using *c-of-switch*[*of z const x*] *z fresh-prodN z fresh-prodN check-branch-s-branchI* **by** *argo*
qed
qed
moreover have $s[x::=v]_{sv}$ = $s'[x'::=v]_{sv}$
using *check-branch-s-branchI subst-v-flip-eq-two subst-v-s-def s-branch-s-branch-list.eq-iff* **by** *metis*
ultimately show *?case* **using** *check-branch-s-branchI* ‹$\tau[x::=v]_{\tau v}$ = τ› ‹$\Delta[x::=v]_{\Delta v}$ = Δ› **by** *auto*
qed(*auto+*)

Lemmas for while reduction. Making these separate lemmas allows flexibility in wiring them into the main proof and robustness if we change it

lemma *check-unit*:
fixes τ::τ **and** Φ::Φ **and** Δ::Δ **and** G::Γ
assumes Θ ; {||} ; *GNil* ⊢ ⦃ *z* : *B-unit* | *TRUE* ⦄ ≲ τ' **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ **and** Θ $\vdash_{wf}$ Φ **and** Θ ; {||} $\vdash_{wf}$ G
shows ‹ Θ ; Φ ; {||} ; G ; Δ ⊢ [[*L-unit*]^v]^s ⇐ τ'›
proof −
have ∗:Θ ; {||} ; *GNil* ⊢ [*L-unit*]^v ⇒ ⦃ *z* : *B-unit* | [[*z*]^v]^{ce} == [[*L-unit*]^v]^{ce} ⦄
using *infer-l.intros(4) infer-v-litI fresh-GNil assms wfX-wfY* **by** (*meson subtype-g-wf*)
moreover have Θ ; {||} ; *GNil* ⊢ ⦃ *z* : *B-unit* | [[*z*]^v]^{ce} == [[*L-unit*]^v]^{ce} ⦄ ≲ τ'
using *subtype-top subtype-trans* ∗ *infer-v-wf*
by (*meson assms(1)*)
ultimately show *?thesis*
using *subtype-top subtype-trans fresh-GNil assms check-valI assms check-s-g-weakening assms toSet.simps*

by (*metis bot.extremum infer-v-g-weakening subtype-weakening wfD-wf*)
qed

lemma *preservation-var*:
shows Θ; Φ; {||}; *GNil*; Δ ⊢ *VAR u* : τ' = *v IN s* ⇐ τ ⟹ Θ ⊢ δ ∼ Δ ⟹ *atom u* ♯ δ ⟹ *atom u* ♯ Δ ⟹
Θ; Φ; {||}; *GNil*; (*u*,τ')#$_\Delta\Delta$ ⊢ *s* ⇐ τ ∧ Θ ⊢ (*u*,*v*)#δ ∼ (*u*,τ')#$_\Delta\Delta$
and
check-branch-s Θ Φ {||} *GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ {||} Γ Δ *tid dclist v css* τ ⟹ *True*
proof(*nominal-induct* {||}::*bv fset GNil* Δ *VAR u* : τ' = *v IN s* τ **and** τ **and** τ *rule*: *check-s-check-branch-s-check-branch-*
case (*check-varI u'* Θ Φ Δ τ *s'*)
hence Θ; Φ; {||}; *GNil*; (*u*, τ') #$_\Delta$ Δ ⊢ *s* ⇐ τ **using** *check-s-abs-u check-v-wf* **by** *metis*

moreover have Θ ⊢ ((*u*,*v*)#δ) ∼ ((*u*,τ')#$_\Delta\Delta$) **proof**
show ‹Θ ⊢ δ ∼ Δ › **using** *check-varI* **by** *auto*
show ‹Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ'› **using** *check-varI* **by** *auto*
show ‹*u* ∉ *fst* ‘ *set* δ› **using** *check-varI fresh-d-fst-d* **by** *auto*
qed

ultimately show *?case* **by** *simp*
qed(*auto*+)

lemma *check-while*:
shows Θ; Φ; {||}; *GNil*; Δ ⊢ *WHILE s1 DO* { *s2* } ⇐ τ ⟹ *atom x* ♯ (*s1*, *s2*) ⟹ *atom z'* ♯ *x* ⟹
Θ; Φ; {||}; *GNil*; Δ ⊢ *LET x* : (⦃ *z'* : *B-bool* | *TRUE* ⦄) = *s1 IN* (*IF* (*V-var x*) *THEN* (*s2* ;; (*WHILE s1 DO* {*s2*}))
ELSE ([*V-lit L-unit*]^s)) ⇐ τ **and**
check-branch-s Θ Φ {||} *GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ {||} Γ Δ *tid dclist v css* τ ⟹ *True*
proof(*nominal-induct* {||}::*bv fset GNil* Δ *AS-while s1 s2* τ **and** τ **and** τ *avoiding*: *s1 s2 x z' rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-whileI* Θ Φ Δ *s1 z s2* τ')
have *teq*:⦃ *z'* : *B-bool* | *TRUE* ⦄ = ⦃ *z* : *B-bool* | *TRUE* ⦄ **using** τ*.eq-iff* **by** *auto*

show *?case* **proof**
have *atom x* ♯ τ' **using** *wfT-nil-supp fresh-def subtype-wfT check-whileI(5)* **by** *fast*

moreover have *atom x* ♯ ⦃ *z′* : *B-bool* | *TRUE* ⦄ **using** *τ.fresh c.fresh b.fresh* **by** *metis*
ultimately show ‹*atom x* ♯ (Θ, Φ, {||}, *GNil*, Δ, ⦃ *z′* : *B-bool* | *TRUE* ⦄, *s1*, *τ′*)›
apply(*unfold fresh-prodN*)
using *check-whileI wb-x-fresh check-s-wf wfD-x-fresh fresh-empty-fset fresh-GNil fresh-Pair* ‹*atom x* ♯ *τ′*› **by** *metis*

show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s1* ⇐ ⦃ *z′* : *B-bool* | *TRUE* ⦄› **using** *check-whileI teq* **by** *metis*

let *?G* = (*x*, *b-of* ⦃ *z′* : *B-bool* | *TRUE* ⦄, *c-of* ⦃ *z′* : *B-bool* | *TRUE* ⦄ *x*) $\#_\Gamma$ *GNil*

have *cof*:(*c-of* ⦃ *z′* : *B-bool* | *TRUE* ⦄ *x*) = *C-true* **using** *c-of.simps check-whileI subst-cv.simps* **by** *metis*
have *wfg*: Θ ; {||} $\vdash_{wf}$ *?G* **proof**
show *c-of* ⦃ *z′* : *B-bool* | *TRUE* ⦄ *x* ∈ {*TRUE*, *FALSE*} **using** *subst-cv.simps cof* **by** *auto*
show Θ ; {||} $\vdash_{wf}$ *GNil* **using** *wfG-nilI check-whileI wfX-wfY check-s-wf* **by** *metis*
show *atom x* ♯ *GNil* **using** *fresh-GNil* **by** *auto*
show Θ ; {||} $\vdash_{wf}$ *b-of* ⦃ *z′* : *B-bool* | *TRUE* ⦄ **using** *wfB-boolI wfX-wfY check-s-wf b-of.simps* **by** (*metis* ‹Θ ; {||} $\vdash_{wf}$ *GNil*›)
qed

obtain *zz*::*x* **where** *zf*:‹*atom zz* ♯ ((Θ, Φ, {||}::*bv fset*, *?G* , Δ, [*x*]^v,
AS-seq s2 (*AS-while s1 s2*), *AS-val* [*L-unit*]^v, *τ′*),*x*,*?G*)›
using *obtain-fresh* **by** *blast*
show ‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢
AS-if [*x*]^v (*AS-seq s2* (*AS-while s1 s2*)) (*AS-val* [*L-unit*]^v) ⇐ *τ′*›
proof
show *atom zz* ♯ (Θ, Φ, {||}::*bv fset*, *?G* , Δ, [*x*]^v, *AS-seq s2* (*AS-while s1 s2*), *AS-val* [*L-unit*]^v, *τ′*) **using** *zf* **by** *auto*
show ‹Θ ; {||} ; *?G* ⊢ [*x*]^v ⇐ ⦃ *zz* : *B-bool* | *TRUE* ⦄› **proof**
have *atom zz* ♯ *x* ∧ *atom zz* ♯ (Θ, {||}::*bv fset*, *?G*) **using** *zf fresh-prodN* **by** *metis*
thus ‹ Θ ; {||} ; *?G* ⊢ [*x*]^v ⇒⦃ *zz* : *B-bool* | [[*zz*]^v]^{ce} == [[*x*]^v]^{ce} ⦄›
using *infer-v-varI lookup.simps wfg b-of.simps* **by** *metis*
thus ‹Θ ; {||} ; *?G* ⊢ ⦃ *zz* : *B-bool* | [[*zz*]^v]^{ce} == [[*x*]^v]^{ce} ⦄ ≲ ⦃ *zz* : *B-bool* | *TRUE* ⦄›
using *subtype-top infer-v-wf* **by** *metis*
qed
show ‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢ *AS-seq s2* (*AS-while s1 s2*) ⇐ ⦃ *zz* : *b-of τ′* | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of τ′ zz* ⦄›
proof
have ⦃ *zz* : *B-unit* | *TRUE* ⦄ = ⦃ *z* : *B-unit* | *TRUE* ⦄ **using** *τ.eq-iff* **by** *auto*
thus ‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢ *s2* ⇐ ⦃ *zz* : *B-unit* | *TRUE* ⦄› **using** *check-s-g-weakening*(*1*) [*OF check-whileI*(*3*) - *wfg*] *toSet.simps*
by (*simp add*: ‹⦃ *zz* : *B-unit* | *TRUE* ⦄ = ⦃ *z* : *B-unit* | *TRUE* ⦄›)

show ‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢ *AS-while s1 s2* ⇐ ⦃ *zz* : *b-of τ′* | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of τ′ zz* ⦄›
proof(*rule check-s-supertype*(*1*))
have ‹ Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-while s1 s2* ⇐ *τ′*› **using** *check-whileI* **by** *auto*
thus ∗:‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢ *AS-while s1 s2* ⇐ *τ′* › **using** *check-s-g-weakening*(*1*)[*OF* - - *wfg*] *toSet.simps* **by** *auto*

show ‹Θ ; {||} ; *?G* ⊢ *τ′* ≲ ⦃ *zz* : *b-of τ′* | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of τ′ zz* ⦄›

proof(*rule subtype-eq-if-τ*)
show ‹ Θ ; {||} ; *?G* $\vdash_{wf}$ τ′ › **using** ∗ *check-s-wf* **by** *auto*
show ‹ Θ ; {||} ; *?G* $\vdash_{wf}$ ⦃ *zz* : *b-of* τ′ | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* τ′ *zz* ⦄ ›
apply(*rule wfT-eq-imp*, *simp add*: *base-for-lit.simps*)
using *subtype-wf check-whileI wfg zf fresh-prodN* **by** *metis*+
show ‹*atom zz* ♯ τ′› **using** *zf fresh-prodN* **by** *metis*
qed
qed

qed
show ‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢ *AS-val* [*L-unit*]^v ⇐ ⦃ *zz* : *b-of* τ′ | [[*x*]^v]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* τ′ *zz* ⦄›
proof(*rule check-s-supertype*(*1*))

show ∗:‹ Θ ; Φ ; {||} ; *?G* ; Δ ⊢ *AS-val* [*L-unit*]^v ⇐ τ′›
using *check-unit*[*OF check-whileI*(*5*) - - *wfg*] **using** *check-whileI wfg wfX-wfY check-s-wf* **by** *metis*
show ‹Θ ; {||} ; *?G* ⊢ τ′ ≲ ⦃ *zz* : *b-of* τ′ | [[*x*]^v]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* τ′ *zz* ⦄›
proof(*rule subtype-eq-if-τ*)
show ‹ Θ ; {||} ; *?G* $\vdash_{wf}$ τ′ › **using** ∗ *check-s-wf* **by** *metis*
show ‹ Θ ; {||} ; *?G* $\vdash_{wf}$ ⦃ *zz* : *b-of* τ′ | [[*x*]^v]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* τ′ *zz* ⦄ ›
apply(*rule wfT-eq-imp*, *simp add*: *base-for-lit.simps*)
using *subtype-wf check-whileI wfg zf fresh-prodN* **by** *metis*+
show ‹*atom zz* ♯ τ′› **using** *zf fresh-prodN* **by** *metis*
qed
qed
qed
qed
qed(*auto*+)

lemma *check-s-narrow*:
fixes *s*::*s* **and** *x*::*x*
assumes *atom x* ♯ (Θ, Φ, ℬ, Γ, Δ, *c*, τ, *s*) **and** Θ ; Φ ; ℬ ; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ ⊢ *s* ⇐ τ **and**
Θ ; ℬ ; Γ ⊨ *c*
shows Θ ; Φ ; ℬ ; Γ ; Δ ⊢ *s* ⇐ τ
proof −
let *?B* = ({||}::*bv fset*)
let *?v* = *V-lit L-true*

obtain *z*::*x* **where** *z*: *atom z* ♯ (*x*, [*L-true*]^v,*c*) **using** *obtain-fresh* **by** *metis*

have *atom z* ♯ *c* **using** *z fresh-prodN* **by** *auto*
hence *c*: *c*[*x*::=[*z*]^v]$_v$[*z*::=[*x*]^v]$_{cv}$ = *c* **using** *subst-v-c-def* **by** *simp*

have Θ ; Φ ; ℬ ; ((*x*,*B-bool*, *c*) $\#_\Gamma$ Γ)[*x*::=*?v*]$_{\Gamma v}$; Δ[*x*::=*?v*]$_{\Delta v}$ ⊢ *s*[*x*::=*?v*]$_{sv}$ ⇐ τ[*x*::=*?v*]$_{\tau v}$
proof(*rule subst-infer-check-s*(*1*))
show *vt*: ‹ Θ ; ℬ ; Γ ⊢ [*L-true*]^v ⇒ ⦃ *z* : *B-bool* | (*CE-val* (*V-var z*)) == (*CE-val* (*V-lit L-true*)) ⦄ ›
using *infer-v-litI check-s-wf wfG-elims*(*2*) *infer-trueI assms* **by** *metis*
show ‹Θ ; ℬ ; Γ ⊢ ⦃ *z* : *B-bool* | (*CE-val* (*V-var z*)) == (*CE-val* (*V-lit L-true*)) ⦄ ≲ ⦃ *z* : *B-bool*

| $c[x::=[\ z\ ]^v]_v$ |} › **proof**
  show ‹*atom x* ♯ (Θ, $\mathcal{B}$, Γ, *z*, [[*z*]^v]^{ce} == [[*L-true*]^v]^{ce} , *z*, $c[x::=[\ z\ ]^v]_v$)›
    apply(*unfold fresh-prodN*, *intro conjI*)
    prefer *5*
    using *c.fresh ce.fresh v.fresh z fresh-prodN* **apply** *auto*[*1*]
    prefer *6*
    using *fresh-subst-v-if*[*of atom x c x*] *assms fresh-prodN* **apply** *simp*
    using *z assms fresh-prodN* **apply** *metis*
    using *z assms fresh-prodN* **apply** *metis*
    using *z assms fresh-prodN* **apply** *metis*
    using *z fresh-prodN assms fresh-at-base* **by** *metis+*
  show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ {| *z* : *B-bool* | [[*z*]^v]^{ce} == [[*L-true*]^v]^{ce} |} › **using** *vt infer-v-wf* **by** *simp*
  show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ {| *z* : *B-bool* | $c[x::=[\ z\ ]^v]_v$ |} › **proof**(*rule wfG-wfT*)
    show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *B-bool*, $c[x::=[\ z\ ]^v]_v[z::=[\ x\ ]^v]_{cv}$) $\#_\Gamma$ Γ › **using** *c check-s-wf assms* **by** *metis*
    have *atom x* ♯ [*z*]^v **using** *v.fresh z fresh-at-base* **by** *auto*
    thus ‹*atom x* ♯ $c[x::=[\ z\ ]^v]_v$› **using** *fresh-subst-v-if*[*of atom x c*] **by** *auto*
  qed
  have *wfg*: Θ ; $\mathcal{B}$ $\vdash_{wf}$(*x*, *B-bool*, ([[*z*]^v]^{ce} == [[*L-true*]^v]^{ce})$[z::=[\ x\ ]^v]_v$) $\#_\Gamma$ Γ
    using *wfT-wfG vt infer-v-wf fresh-prodN assms* **by** *simp*
  show ‹Θ ; $\mathcal{B}$; (*x*, *B-bool*, ([[*z*]^v]^{ce} == [[*L-true*]^v]^{ce})$[z::=[\ x\ ]^v]_v$) $\#_\Gamma$ Γ ⊨ $c[x::=[\ z\ ]^v]_v[z::=[\ x\ ]^v]_v$ ›
    using *c valid-weakening*[*OF assms*(*3*) - *wfg*] *toSet.simps*
    using *subst-v-c-def* **by** *auto*
  qed
  show ‹*atom z* ♯ (*x*, [*L-true*]^v)› **using** *z fresh-prodN* **by** *metis*
  show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ ⊢ *s* ⇐ τ› **using** *assms* **by** *auto*

  thus ‹(*x*, *B-bool*, *c*) $\#_\Gamma$ Γ = *GNil* @ (*x*, *B-bool*, $c[x::=[\ z\ ]^v]_v[z::=[\ x\ ]^v]_{cv}$) $\#_\Gamma$ Γ› **using** *append-g.simps c* **by** *auto*
 qed

 moreover have ((*x*,*B-bool*, *c*) $\#_\Gamma$ Γ)$[x::=?v]_{\Gamma v}$ = Γ **using** *subst-gv.simps* **by** *auto*
 ultimately show *?thesis* **using** *assms forget-subst-dv forget-subst-sv forget-subst-tv fresh-prodN* **by** *metis*
qed

lemma *check-assert-s*:
 fixes *s*::*s* **and** *x*::*x*
 assumes Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-assert c s* ⇐ τ
 shows Θ; Φ; {||}; *GNil*; Δ ⊢ *s* ⇐ τ ∧ Θ ; {||} ; *GNil* ⊨ *c*
proof −
 let *?B* = ({||}::*bv fset*)
 let *?v* = *V-lit L-true*

 obtain *x* **where** *x*: Θ ; Φ ; *?B* ; (*x*,*B-bool*, *c*) $\#_\Gamma$ *GNil* ; Δ ⊢ *s* ⇐ τ ∧ *atom x* ♯ (Θ, Φ, *?B*, *GNil*, Δ, *c*, τ, *s*) ∧ Θ ; *?B* ; *GNil* ⊨ *c*
  using *check-s-elims*(*10*)[*OF* ‹Θ ; Φ ; *?B* ; *GNil* ; Δ ⊢ *AS-assert c s* ⇐ τ›] *valid.simps* **by** *metis*

 show *?thesis* **using** *assms check-s-narrow x* **by** *metis*
qed

lemma *infer-v-pair2I*:
atom z ♯ (*v1*, *v2*) ⟹
atom z ♯ (Θ, $\mathcal{B}$, Γ) ⟹
Θ ; $\mathcal{B}$; Γ ⊢ *v1* ⇒ *t1* ⟹
Θ ; $\mathcal{B}$; Γ ⊢ *v2* ⇒ *t2* ⟹
b1 = *b-of t1* ⟹ *b2* = *b-of t2* ⟹
Θ ; $\mathcal{B}$; Γ ⊢ [*v1* , *v2*]^v ⇒ ⦃ *z* : [*b1* , *b2*]^b | [[*z*]^v]^{ce} == [[*v1* , *v2*]^v]^{ce} ⦄
using *infer-v-pairI* **by** *simp*

16.2.5 Main Lemma

lemma *preservation*:
assumes Φ ⊢ ⟨δ, *s*⟩ ⟶ ⟨δ′, *s*′⟩ **and** Θ; Φ; Δ ⊢ ⟨δ, *s*⟩ ⇐ τ
shows ∃Δ′. Θ; Φ; Δ′ ⊢ ⟨δ′, *s*′⟩ ⇐ τ ∧ Δ ⊑ Δ′
using *assms*
proof(*induct arbitrary*: τ *rule*: *reduce-stmt.induct*)
case (*reduce-let-plusI δ x n1 n2 s*′)
then show *?case* **using** *preservation-plus*
by (*metis order-refl*)
next
case (*reduce-let-leqI b n1 n2 δ x s*)
then show *?case* **using** *preservation-leq* **by** (*metis order-refl*)
next
case (*reduce-let-eqI b n1 n2* Φ *δ x s*)
then show *?case* **using** *preservation-eq*[*OF reduce-let-eqI*(*2*)] *order-refl* **by** *metis*
next
case (*reduce-let-appI f z b c* τ′ *s*′ Φ *δ x v s*)
hence *tt*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-app f v*) *s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*) **using** *config-type-elims*[*OF reduce-let-appI*(*2*)] **by** *metis*
hence ∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-app f v*) *s* ⇐ τ **by** *auto*

hence Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let2 x* (τ′[*z*::=*v*]$_{\tau v}$) (*s*′[*z*::=*v*]$_{sv}$) *s* ⇐ τ
using *preservation-app reduce-let-appI tt* **by** *auto*

hence Θ; Φ; Δ ⊢ ⟨ δ , *AS-let2 x* (τ′[*z*::=*v*]$_{\tau v}$) *s*′[*z*::=*v*]$_{sv}$ *s* ⟩ ⇐ τ **using** *config-typeI tt* **by** *auto*
then show *?case* **using** *tt order-refl reduce-let-appI* **by** *metis*

next
case (*reduce-let-appPI f bv z b c* τ′ *s*′ Φ *δ x b*′ *v s*)

hence *tt*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-appP f b*′ *v*) *s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*) **using** *config-type-elims*[*OF reduce-let-appPI*(*2*)] **by** *metis*
hence ∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-appP f b*′ *v*) *s* ⇐ τ **by** *auto*

have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let2 x* (τ′[*bv*::=*b*′]$_{\tau b}$[*z*::=*v*]$_{\tau v}$) (*s*′[*bv*::=*b*′]$_{sb}$[*z*::=*v*]$_{sv}$) *s* ⇐ τ
proof(*rule preservation-poly-app*)
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ z b c* τ′ *s*′))) = *lookup-fun* Φ *f*› **using** *reduce-let-appPI* **by** *metis*
show ‹∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*› **using** *tt lookup-fun-member* **by** *metis*
show ‹ Θ ; Φ ;{||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-appP f b*′ *v*) *s* ⇐ τ› **using** ∗ **by** *auto*
show ‹ Θ ; {||} ⊢$_{wf}$ *b*′ › **using** *check-s-elims infer-e-wf wfE-elims* ∗ **by** *metis*
qed(*auto*+)

hence Θ; Φ; Δ ⊢ ⟨ δ , *AS-let2 x* (τ′[*bv*::=*b*′]$_{\tau b}$[*z*::=*v*]$_{\tau v}$) *s*′[*bv*::=*b*′]$_{sb}$[*z*::=*v*]$_{sv}$ *s* ⟩ ⇐ τ **using** *config-typeI tt* **by** *auto*
then show *?case* **using** *tt order-refl reduce-let-appI* **by** *metis*

next
case (*reduce-if-trueI* δ *s1 s2*)
then show *?case* **using** *preservation-if* **by** *metis*
next
case (*reduce-if-falseI uw* δ *s1 s2*)
then show *?case* **using** *preservation-if* **by** *metis*
next
case (*reduce-let-valI* δ *x v s*)
then show *?case* **using** *preservation-let-val* **by** *presburger*
next
case (*reduce-let-mvar u v* δ Φ *x s*)
hence ∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-mvar u*) *s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *blast*

hence ∗∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-mvar u*) *s* ⇐ τ **by** *auto*
obtain *xa*::*x* **and** *za*::*x* **and** *ca*::*c* **and** *ba*::*b* **and** *sa*::*s* **where**
sa1: *atom xa* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-mvar u*, τ) ∧ *atom za* ♯ (*xa*, Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-mvar u*, τ, *sa*) ∧
Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-mvar u* ⇒ ⦃ *za* : *ba* | *ca* ⦄ ∧
Θ ; Φ ; {||} ; (*xa*, *ba*, *ca*[*za*::=*V-var xa*]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ ⊢ *sa* ⇐ τ ∧
(∀ *c*. *atom c* ♯ (*s*, *sa*) ⟶ *atom c* ♯ (*x*, *xa*, *s*, *sa*) ⟶ (*x* ↔ *c*) · *s* = (*xa* ↔ *c*) · *sa*)
using *check-s-elims*(*2*)[*OF* ∗∗] *subst-defs* **by** *metis*

have Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *za* : *ba* | *ca* ⦄ **proof** −
have (*u* , ⦃ *za* : *ba* | *ca* ⦄) ∈ *setD* Δ **using** *infer-e-elims*(*11*) *sa1* **by** *fast*
thus *?thesis* **using** *delta-sim-v reduce-let-mvar config-type-elims check-s-wf* **by** *metis*
qed

then obtain τ′ **where** *vst*: Θ ; {||} ; *GNil* ⊢ *v* ⇒ τ′ ∧
Θ ; {||} ; *GNil* ⊢ τ′ ≲ ⦃ *za* : *ba* | *ca* ⦄ **using** *check-v-elims* **by** *blast*

obtain *za2* **and** *ba2* **and** *ca2* **where** *zbc*: τ′ = (⦃ *za2* : *ba2* | *ca2* ⦄) ∧ *atom za2* ♯ (*xa*, (*xa*, Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-val v*, τ, *sa*))
using *obtain-fresh-z* **by** *blast*
have *beq*: *ba*=*ba2* **using** *subtype-eq-base vst zbc* **by** *blast*

moreover have *xaf*: *atom xa* ♯ (*za*, *za2*)
apply(*unfold fresh-prodN*, *intro conjI*)
using *sa1 zbc fresh-prodN fresh-x-neq* **by** *metis*+

have *sat2*: Θ ; Φ ; {||} ; *GNil*@(*xa*, *ba*, *ca2*[*za2*::=*V-var xa*]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ ⊢ *sa* ⇐ τ **proof**(*rule ctx-subtype-s*)
show Θ ; Φ ; {||} ; *GNil* @ (*xa*, *ba*, *ca*[*za*::=*V-var xa*]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ ⊢ *sa* ⇐ τ **using** *sa1* **by** *auto*
show Θ ; {||} ; *GNil* ⊢ ⦃ *za2* : *ba* | *ca2* ⦄ ≲ ⦃ *za* : *ba* | *ca* ⦄ **using** *beq zbc vst* **by** *fast*
show *atom xa* ♯ (*za*, *za2*, *ca*, *ca2*) **proof** −
have ∗:Θ ; {||} ; *GNil* $\vdash_{wf}$ (⦃ *za2* : *ba2* | *ca2* ⦄) **using** *zbc vst subtype-wf* **by** *auto*

hence *supp ca2* ⊆ { *atom za2* } **using** *wfT-supp-c*[*OF* ∗] *supp-GNil* **by** *simp*
moreover have *atom za2* ♯ *xa* **using** *zbc fresh-Pair fresh-x-neq* **by** *metis*
ultimately have *atom xa* ♯ *ca2* **using** *zbc supp-at-base fresh-def*
by (*metis empty-iff singleton-iff subset-singletonD*)
moreover have *atom xa* ♯ *ca* **proof** −
have ∗:Θ ; {||} ; *GNil* $\vdash_{wf}$ (⦃ *za* : *ba* | *ca* ⦄) **using** *zbc vst subtype-wf* **by** *auto*
hence *supp ca* ⊆ { *atom za* } **using** *wfT-supp τ.supp* **by** *force*
moreover have *xa* ≠ *za* **using** *fresh-def fresh-x-neq xaf fresh-Pair* **by** *metis*
ultimately show *?thesis* **using** *fresh-def* **by** *auto*
qed
ultimately show *?thesis* **using** *xaf sa1 fresh-prod4 fresh-Pair* **by** *metis*
qed
qed
hence *dwf*: Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ **using** *sa1 infer-e-wf* **by** *meson*

have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let xa* (*AE-val v*) *sa* ⇐ τ **proof**
have *atom xa* ♯ (*AE-val v*) **using** *infer-v-wf*(*1*) *wfV-supp fresh-def e.fresh x-not-in-b-set vst* **by** *fastforce*
thus *atom xa* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-val v*, τ) **using** *sa1 freshers* **by** *simp*
have *atom za2* ♯ (*AE-val v*) **using** *infer-v-wf*(*1*) *wfV-supp fresh-def e.fresh x-not-in-b-set vst* **by** *fastforce*
thus *atom za2* ♯ (*xa*, Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-val v*, τ, *sa*) **using** *zbc freshers fresh-prodN* **by** *auto*
have Θ $\vdash_{wf}$ Φ **using** *sa1 infer-e-wf* **by** *auto*
thus Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-val v* ⇒ ⦃ *za2* : *ba* | *ca2* ⦄
using *zbc vst beq dwf infer-e-valI* **by** *blast*
show Θ ; Φ ; {||} ; (*xa*, *ba*, *ca2*[*za2*::=*V-var xa*]$_v$) $\#_\Gamma$ *GNil* ; Δ ⊢ *sa* ⇐ τ **using** *sat2 append-g.simps subst-defs* **by** *metis*
qed
moreover have *AS-let xa* (*AE-val v*) *sa* = *AS-let x* (*AE-val v*) *s* **proof** −
have [[*atom x*]]*lst*. *s* = [[*atom xa*]]*lst*. *sa*
using *sa1 Abs1-eq-iff-all*(*3*)[**where** *z*= (*s*, *sa*)] **by** *metis*
thus *?thesis* **using** *s-branch-s-branch-list.eq-iff*(*2*) **by** *metis*
qed
ultimately have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-val v*) *s* ⇐ τ **by** *auto*

then show *?case* **using** *reduce-let-mvar* ∗ *config-typeI*
by (*meson order-refl*)
next
case (*reduce-let2I* Φ δ *s1* δ′ *s1*′ *x t s2*)
hence ∗∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let2 x t s1 s2* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*) **using** *config-type-elims*[*OF reduce-let2I*(*3*)] **by** *blast*
hence ∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let2 x t s1 s2* ⇐ τ **by** *auto*

obtain *xa*::*x* **and** *z*::*x* **and** *c* **and** *b* **and** *s2a*::*s* **where** *st*: *atom xa* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *t*, *s1*, τ) ∧
Θ; Φ; {||}; *GNil*; Δ ⊢ *s1* ⇐ *t* ∧
Θ ; Φ ; {||} ; (*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *GNil* ; Δ ⊢ *s2a* ⇐ τ ∧ ([[*atom x*]]*lst*. *s2* = [[*atom xa*]]*lst*. *s2a*)
using *check-s-elims*(*4*)[*OF* ∗] *Abs1-eq-iff-all*(*3*) **by** *metis*

hence Θ; Φ; Δ ⊢ ⟨ δ , *s1* ⟩ ⇐ *t* **using** *config-typeI* ∗∗ **by** *auto*

then obtain Δ′ **where** *s1r*: Θ; Φ; Δ′ ⊢ ⟨ δ′ , *s1*′ ⟩ ⇐ *t* ∧ Δ ⊑ Δ′ **using** *reduce-let2I* **by** *presburger*

have Θ; Φ; {||}; *GNil*; Δ′ ⊢ *AS-let2 xa t s1*′ *s2a* ⇐ τ
proof(*rule check-let2I*)
  show ∗:Θ; Φ; {||}; *GNil*; Δ′ ⊢ *s1*′ ⇐ *t* **using** *config-type-elims st s1r* **by** *metis*
  show *atom xa* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ′,*t*, *s1*′, τ) **proof** −
    have *atom xa* ♯ *s1*′ **using** *check-s-x-fresh* ∗ **by** *auto*
    moreover have *atom xa* ♯ Δ′ **using** *check-s-x-fresh* ∗ **by** *auto*
    ultimately show *?thesis* **using** *st fresh-prodN* **by** *metis*
  qed

  show Θ ; Φ ; {||} ; (*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *GNil* ; Δ′ ⊢ *s2a* ⇐ τ **proof** −
    have Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ′ **using** ∗ *check-s-wf* **by** *auto*
    moreover have Θ ; {||} $\vdash_{wf}$ ((*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *GNil*) **using** *st check-s-wf* **by** *auto*
    ultimately have Θ ; {||} ; ((*xa*, *b-of t* , *c-of t xa*) $\#_\Gamma$ *GNil*) $\vdash_{wf}$ Δ′ **using** *wf-weakening* **by** *auto*
    thus *?thesis* **using** *check-s-d-weakening check-s-wf st s1r* **by** *metis*
  qed
qed
moreover have *AS-let2 xa t s1*′ *s2a* = *AS-let2 x t s1*′ *s2* **using** *st s-branch-s-branch-list.eq-iff* **by** *metis*
ultimately have Θ; Φ; {||}; *GNil*; Δ′ ⊢ *AS-let2 x t s1*′ *s2* ⇐ τ **using** *st* **by** *argo*
moreover have Θ ⊢ δ′ ∼ Δ′ **using** *config-type-elims s1r* **by** *fast*
ultimately show *?case* **using** *config-typeI* ∗∗
  by (*meson s1r*)
next
  case (*reduce-let2-valI vb* δ *x t v s*)
  then show *?case* **using** *preservation-let-val* **by** *meson*
next
  case (*reduce-varI u* δ Φ τ′ *v s*)

  hence ∗∗ : Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-var u* τ′ *v s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
    using *config-type-elims* **by** *meson*
  have *uf*: *atom u* ♯ Δ **using** *reduce-varI delta-sim-fresh* **by** *force*
  hence ∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-var u* τ′ *v s* ⇐ τ **and** Θ ⊢ δ ∼ Δ **using** ∗∗ **by** *auto*

  thus *?case* **using** *preservation-var reduce-varI config-typeI* ∗∗ *set-subset-Cons*
      setD-ConsD subsetI **by** (*metis delta-sim-fresh*)

next
  case (*reduce-assignI* Φ δ *u v*)
  hence ∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-assign u v* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
    using *config-type-elims* **by** *meson*
  then obtain *z* **and** τ′ **where** *zt*: Θ ; {||} ; *GNil* ⊢ (⦃ *z* : *B-unit* | *TRUE* ⦄) ≲ τ ∧ (*u*,τ′) ∈ *setD* Δ ∧ Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ′ ∧ Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ
    using *check-s-elims*(*8*) **by** *metis*
  hence Θ ⊢ *update-d* δ *u v* ∼ Δ **using** *update-d-sim* ∗ **by** *metis*
  moreover have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-val* (*V-lit L-unit*) ⇐ τ **using** *zt* ∗ *check-s-v-unit check-s-wf*
    by *auto*
  ultimately show *?case* **using** *config-typeI* ∗ **by** (*meson order-refl*)
next

case (*reduce-seq1I* Φ δ *s*)
hence Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *check-s-elims config-type-elims* **by** *force*
then show *?case* **using** *config-typeI* **by** *blast*
next
case (*reduce-seq2I s1 v* Φ δ δ′ *s1*′ *s2*)
hence *tt*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-seq s1 s2* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *blast*
then obtain *z* **where** *zz*: Θ ; Φ ; {||} ; *GNil*; Δ ⊢ *s1* ⇐ (⦃ *z* : *B-unit* | *TRUE* ⦄) ∧ Θ; Φ; {||}; *GNil*; Δ ⊢ *s2* ⇐ τ
using *check-s-elims* **by** *blast*
hence Θ; Φ; Δ ⊢ ⟨ δ , *s1* ⟩ ⇐ (⦃ *z* : *B-unit* | *TRUE* ⦄)
using *tt config-typeI tt* **by** *simp*
then obtain Δ′ **where** ∗: Θ; Φ; Δ′ ⊢ ⟨ δ′ , *s1*′ ⟩ ⇐ (⦃ *z* : *B-unit* | *TRUE* ⦄) ∧ Δ ⊑ Δ′
using *reduce-seq2I* **by** *meson*
moreover hence *s*′*t*: Θ; Φ; {||}; *GNil*; Δ′ ⊢ *s1*′ ⇐ (⦃ *z* : *B-unit* | *TRUE* ⦄) ∧ Θ ⊢ δ′ ∼ Δ′
using *config-type-elims* **by** *force*
moreover hence Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ′ **using** *check-s-wf* **by** *meson*
moreover hence Θ; Φ; {||}; *GNil*; Δ′ ⊢ *s2* ⇐ τ
using *calculation*(*1*) *zz check-s-d-weakening* ∗ **by** *metis*
moreover hence Θ; Φ; {||}; *GNil*; Δ′ ⊢ (*AS-seq s1*′ *s2*) ⇐ τ
using *check-seqI zz s*′*t* **by** *meson*
ultimately have Θ; Φ; Δ′ ⊢ ⟨ δ′ , *AS-seq s1*′ *s2* ⟩ ⇐ τ ∧ Δ ⊑ Δ′
using *zz config-typeI tt* **by** *meson*
then show *?case* **by** *meson*
next
case (*reduce-whileI x s1 s2 z*′ Φ δ)

hence ∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-while s1 s2* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *meson*

hence ∗∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-while s1 s2* ⇐ τ **by** *auto*
hence Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let2 x* (⦃ *z*′ : *B-bool* | *TRUE* ⦄) *s1* (*AS-if* (*V-var x*) (*AS-seq s2* (*AS-while s1 s2*)) (*AS-val* (*V-lit L-unit*))) ⇐ τ
using *check-while reduce-whileI* **by** *auto*
thus *?case* **using** *config-typeI* ∗ **by** (*meson subset-refl*)

next
case (*reduce-caseI dc x*′ *s*′ *css* Φ δ *tyid v*)

hence ∗∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-match* (*V-cons tyid dc v*) *css* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims*[*OF reduce-caseI*(*2*)] **by** *metis*
hence ∗∗∗: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-match* (*V-cons tyid dc v*) *css* ⇐ τ **by** *auto*

let *?vcons* = *V-cons tyid dc v*

obtain *dclist tid* **and** *z*::*x* **where** *cv*: Θ; {||} ; *GNil* ⊢ (*V-cons tyid dc v*) ⇐ (⦃ *z* : *B-id tid* | *TRUE* ⦄) ∧
Θ; Φ; {||}; *GNil*; Δ ; *tid* ; *dclist* ; (*V-cons tyid dc v*) ⊢ *css* ⇐ τ ∧ *AF-typedef tid dclist* ∈ *set* Θ ∧

Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄
  using *check-s-elims*(*9*)[*OF* ∗∗∗] **by** *metis*

 hence *vi*: Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄ **by** *auto*
 obtain *tcons* **where** *vi2*: Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇒ *tcons* ∧ Θ ; {||} ; *GNil* ⊢ *tcons* ≲ ⦃ *z* : *B-id tid* | *TRUE* ⦄
  using *check-v-elims*(*1*)[*OF vi*] **by** *metis*
 hence *vi1*: Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇒ *tcons* **by** *auto*

 show *?case* **proof**(*rule infer-v-elims*(*4*)[*OF vi1*],*goal-cases*)
  case (*1 dclist2 tc tv z2*)
  have *tyid* = *tid* **using** *τ.eq-iff* **using** *subtype-eq-base vi2 1* **by** *fastforce*
  hence *deq*:*dclist* = *dclist2* **using** *check-v-wf wfX-wfY cv 1 wfTh-dclist-unique* **by** *metis*
  have Θ; Φ; {||}; *GNil*; Δ ⊢ $s'[x'::=v]_{sv}$ ⇐ τ **proof**(*rule check-match*(*3*))
   show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ; *tyid* ; *dclist* ; *?vcons* ⊢ *css* ⇐ τ› **using** ‹*tyid* = *tid*› *cv* **by** *auto*
   show *distinct* (*map fst dclist*) **using** *wfTh-dclist-distinct check-v-wf wfX-wfY cv* **by** *metis*
   show ‹*?vcons* = *V-cons tyid dc v*› **by** *auto*
   show ‹{||} = {||}› **by** *auto*
   show ‹(*dc*, *tc*) ∈ *set dclist*› **using** *1 deq* **by** *auto*
   show ‹*GNil* = *GNil*› **by** *auto*
   show ‹*Some* (*AS-branch dc x′ s′*) = *lookup-branch dc css*› **using** *reduce-caseI* **by** *auto*
   show ‹Θ ; {||} ; *GNil* ⊢ *v* ⇐ *tc*› **using** *1 check-v.intros* **by** *auto*
  qed
  thus *?case* **using** *config-typeI* ∗∗ **by** *blast*
 qed

next
 case (*reduce-let-fstI* Φ δ *x v1 v2 s*)
 thus *?case* **using** *preservation-fst-snd order-refl* **by** *metis*
next
 case (*reduce-let-sndI* Φ δ *x v1 v2 s*)
 thus *?case* **using** *preservation-fst-snd order-refl* **by** *metis*
next
 case (*reduce-let-concatI* Φ δ *x v1 v2 s*)
 hence *elim*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) *s* ⇐ τ ∧
   Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
  using *config-type-elims* **by** *metis*

 obtain *z*::*x* **where** *z*: *atom z* ♯ (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)), *GNil*, *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))))
  using *obtain-fresh* **by** *metis*

 have Θ ; {||} $\vdash_{wf}$ *GNil* **using** *check-s-wf elim* **by** *auto*

 have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*)))) *s* ⇐ τ **proof**(*rule subtype-let*)
  show ‹ Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) *s* ⇐ τ› **using** *elim* **by** *auto*
  show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) ⇒ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == (*CE-concat* ($[V\text{-}lit\ (L\text{-}bitvec\ v1)]^{ce}$) ($[V\text{-}lit\ (L\text{-}bitvec\ v2)]^{ce}$))⦄ ›
   (**is** Θ; Φ; {||}; *GNil*; Δ ⊢ *?e1* ⇒ *?t1*)
  proof

show ‹ Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ ; {||} ; *GNil* ⊢ *V-lit* (*L-bitvec v1*) ⇒ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec v1*)) ⦄›
using *infer-v-litI infer-l.intros* ‹Θ ; {||} $\vdash_{wf}$ *GNil*› *fresh-GNil* **by** *auto*
show ‹ Θ ; {||} ; *GNil* ⊢ *V-lit* (*L-bitvec v2*) ⇒ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec v2*)) ⦄›
using *infer-v-litI infer-l.intros* ‹Θ ; {||} $\vdash_{wf}$ *GNil*› *fresh-GNil* **by** *auto*
show ‹*atom z* ♯ *AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))› **using** *z fresh-Pair* **by** *metis*
show ‹*atom z* ♯ *GNil*› **using** *z fresh-Pair* **by** *auto*
qed
show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) ⇒ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) ⦄ ›
(**is** Θ; Φ; {||}; *GNil*; Δ ⊢ *?e2* ⇒ *?t2*)
using *infer-e-valI infer-v-litI infer-l.intros* ‹Θ ; {||} $\vdash_{wf}$ *GNil*› *fresh-GNil check-s-wf elim* **by** *metis*
show ‹Θ ; {||} ; *GNil* ⊢ *?t2* ≲ *?t1*› **using** *subtype-concat check-s-wf elim* **by** *auto*
qed

thus *?case* **using** *config-typeI elim* **by** (*meson order-refl*)
next
case (*reduce-let-lenI* Φ δ *x v s*)
hence *elim*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-len* (*V-lit* (*L-bitvec v*))) *s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *check-s-elims config-type-elims* **by** *metis*

then obtain *t* **where** *t*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-len* (*V-lit* (*L-bitvec v*)) ⇒ *t* **using** *check-s-elims* **by** *meson*

moreover then obtain *z*::*x* **where** *t* = ⦃ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* ([*V-lit* (*L-bitvec v*)]^{ce}) ⦄ **using** *infer-e-elims* **by** *meson*

moreover obtain *z'*::*x* **where** *atom z'* ♯ *v* **using** *obtain-fresh* **by** *metis*
moreover have Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-val* (*V-lit* (*L-num* (*int* (*length v*)))) ⇒ ⦃ *z'* : *B-int* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) ⦄
using *infer-e-valI infer-v-litI infer-l.intros*(*3*) *t check-s-wf elim*
by (*metis infer-l-form2 type-for-lit.simps*(*3*))

moreover have Θ ; {||} ; *GNil* ⊢ ⦃ *z'* : *B-int* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) ⦄ ≲
⦃ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* [(*V-lit* (*L-bitvec v*))]^{ce} ⦄ **using** *subtype-len check-s-wf elim* **by** *auto*

ultimately have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-val* (*V-lit* (*L-num* (*int* (*length v*))))) *s* ⇐ τ **using** *subtype-let* **by** (*meson elim*)
thus *?case* **using** *config-typeI elim* **by** (*meson order-refl*)
next
case (*reduce-let-splitI n v v1 v2* Φ δ *x s*)
hence *elim*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) *s* ⇐ τ ∧
Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *metis*

obtain *z::x* **where** *z*: *atom z* ♯ (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*)), *GNil*, *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))),
([*L-bitvec v1*]^v, [*L-bitvec v2*]^v), Θ, {||}::*bv fset*)
using *obtain-fresh* **by** *metis*

have ∗:Θ ; {||} $\vdash_{wf}$ *GNil* **using** *check-s-wf elim* **by** *auto*

have Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-val* (*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)))) *s* ⇐ τ **proof**(*rule subtype-let*)

show ‹ Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-let x* (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) *s* ⇐ τ› **using** *elim* **by** *auto*
show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) ⇒ ⦃ *z* : *B-pair B-bitvec B-bitvec*
| ((*CE-val* (*V-lit* (*L-bitvec v*))) == (*CE-concat* (*CE-fst* (*CE-val* (*V-var z*))) (*CE-snd* (*CE-val* (*V-var z*)))))
AND (((*CE-len* (*CE-fst* (*CE-val* (*V-var z*))))) == (*CE-val* (*V-lit* (*L-num n*)))) ⦄ ›
(**is** Θ; Φ; {||}; *GNil*; Δ ⊢ *?e1* ⇒ *?t1*)
proof
show ‹ Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ ; {||} ; *GNil* ⊢ *V-lit* (*L-bitvec v*) ⇒ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec v*)) ⦄›
using *infer-v-litI infer-l.intros* ‹Θ ; {||} $\vdash_{wf}$ *GNil*› *fresh-GNil* **by** *auto*
show Θ ; {||} ; *GNil* ⊢ ([*L-num*
n]^v) ⇐ ⦃ *z* : *B-int* | (([*leq* [[*L-num 0*]^v]^{ce} [[*z*]^v]^{ce}]^{ce}) == ([[*L-true*]^v]^{ce})) *AND* [*leq* [[*z*]^v]^{ce} [| [[*L-bitvec*
v]^v]^{ce} |]^{ce}]^{ce} == [[*L-true*]^v]^{ce} ⦄ **using** *split-n reduce-let-splitI check-v-num-leq* ∗ *wfX-wfY* **by** *metis*
show ‹*atom z* ♯ *AE-split* [*L-bitvec v*]^v [*L-num n*]^v› **using** *z fresh-Pair* **by** *auto*
show ‹*atom z* ♯ *GNil*› **using** *z fresh-Pair* **by** *auto*
show ‹*atom z* ♯ *AE-split* [*L-bitvec v*]^v [*L-num n*]^v› **using** *z fresh-Pair* **by** *auto*
show ‹*atom z* ♯ *GNil*› **using** *z fresh-Pair* **by** *auto*
show ‹*atom z* ♯ *AE-split* [*L-bitvec v*]^v [*L-num n*]^v› **using** *z fresh-Pair* **by** *auto*
show ‹*atom z* ♯ *GNil*› **using** *z fresh-Pair* **by** *auto*
qed

show ‹Θ; Φ; {||}; *GNil*; Δ ⊢ *AE-val* (*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) ⇒ ⦃ *z* : *B-pair B-bitvec B-bitvec* | *CE-val* (*V-var z*) == *CE-val* ((*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)))) ⦄ ›
(**is** Θ; Φ; {||}; *GNil*; Δ ⊢ *?e2* ⇒ *?t2*)
apply(*rule infer-e-valI*)
using *check-s-wf elim* **apply** *metis*
using *check-s-wf elim* **apply** *metis*
apply(*rule infer-v-pair2I*)
using *z fresh-prodN* **apply** *metis*
using *z fresh-GNil fresh-prodN* **apply** *metis*
using *infer-v-litI infer-l.intros* ‹Θ ; {||} $\vdash_{wf}$ *GNil*› *b-of.simps* **apply** *blast+*
using *b-of.simps* **apply** *simp+*
done
show ‹Θ ; {||} ; *GNil* ⊢ *?t2* ≲ *?t1*› **using** *subtype-split check-s-wf elim reduce-let-splitI* **by** *auto*

qed

thus *?case* **using** *config-typeI elim* **by** (*meson order-refl*)
next
case (*reduce-assert1I* Φ δ *c v*)

hence *elim*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-assert c* $[v]^s$ ⇐ τ ∧
Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims reduce-assert1I* **by** *metis*
hence ∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-assert c* $[v]^s$ ⇐ τ **by** *auto*

have Θ; Φ; {||}; *GNil*; Δ ⊢ $[v]^s$ ⇐ τ **using** *check-assert-s* ∗ **by** *metis*
thus *?case* **using** *elim config-typeI* **by** *blast*
next
case (*reduce-assert2I* Φ δ *s* δ′ *s*′ *c*)

hence *elim*: Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-assert c s* ⇐ τ ∧
Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *metis*
hence ∗:Θ; Φ; {||}; *GNil*; Δ ⊢ *AS-assert c s* ⇐ τ **by** *auto*

have *cv*: Θ; Φ; {||}; *GNil*; Δ ⊢ *s* ⇐ τ ∧ Θ ; {||} ; *GNil* ⊨ *c* **using** *check-assert-s* ∗ **by** *metis*

hence Θ; Φ; Δ ⊢ ⟨δ, *s*⟩ ⇐ τ **using** *elim config-typeI* **by** *simp*
then obtain Δ′ **where** *D*: Θ; Φ; Δ′ ⊢ ⟨ δ′ , *s*′ ⟩ ⇐ τ ∧ Δ ⊑ Δ′ **using** *reduce-assert2I* **by** *metis*
hence ∗∗:Θ; Φ; {||}; *GNil*; Δ′ ⊢ *s*′ ⇐ τ ∧ Θ ⊢ δ′ ∼ Δ′ **using** *config-type-elims* **by** *metis*

obtain *x*::*x* **where** *x*:*atom x* ♯ (Θ, Φ, ({||}::*bv fset*), *GNil*, Δ′, *c*, τ, *s*′) **using** *obtain-fresh* **by** *metis*

have ∗:Θ; Φ; {||}; *GNil*; Δ′ ⊢ *AS-assert c s*′ ⇐ τ **proof**
show *atom x* ♯ (Θ, Φ, {||}, *GNil*, Δ′, *c*, τ, *s*′) **using** *x* **by** *auto*
have Θ ; {||} ; *GNil* $\vdash_{wf}$ *c* **using** ∗ *check-s-wf* **by** *auto*
hence *wfg*:Θ ; {||} $\vdash_{wf}$ (*x*, *B-bool*, *c*) $\#_\Gamma$ *GNil* **using** *wfC-wfG wfB-boolI check-s-wf* ∗ *fresh-GNil* **by** *auto*
moreover have *cs*: Θ; Φ; {||}; *GNil*; Δ′ ⊢ *s*′ ⇐ τ **using** ∗∗ **by** *auto*
ultimately show Θ ; Φ ; {||} ; (*x*, *B-bool*, *c*) $\#_\Gamma$ *GNil* ; Δ′ ⊢ *s*′ ⇐ τ **using** *check-s-g-weakening*(*1*)[*OF cs - wfg*] *toSet.simps* **by** *simp*
show Θ ; {||} ; *GNil* ⊨ *c* **using** *cv* **by** *auto*
show Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ′ **using** *check-s-wf* ∗∗ **by** *auto*
qed

thus *?case* **using** *elim config-typeI D* ∗∗ **by** *metis*
qed

lemma *preservation-many*:
assumes Φ ⊢ ⟨δ, *s*⟩ ⟶* ⟨ δ′ , *s*′ ⟩
shows Θ; Φ; Δ ⊢ ⟨δ, *s*⟩ ⇐ τ ⟹ ∃Δ′. Θ; Φ; Δ′ ⊢ ⟨ δ′ , *s*′ ⟩ ⇐ τ ∧ Δ ⊑ Δ′
using *assms* **proof**(*induct arbitrary*: Δ *rule*: *reduce-stmt-many.induct*)
case (*reduce-stmt-many-oneI* Φ δ *s* δ′ *s*′)
then show *?case* **using** *preservation* **by** *simp*
next
case (*reduce-stmt-many-manyI* Φ δ *s* δ′ *s*′ δ′′ *s*′′)

then show *?case* **using** *preservation subset-trans* **by** *metis*
qed

16.3 Progress

We prove that a well typed program is either a value or we can make a step

lemma *check-let-op-infer*:
assumes Θ; Φ; {||}; Γ; Δ ⊢ *LET x* = (*AE-op opp v1 v2*) *IN s* ⇐ τ **and** *supp* (*LET x* = (*AE-op opp v1 v2*) *IN s*) ⊆ *atom'fst'setD* Δ
shows ∃ *z b c*. Θ; Φ; {||}; Γ; Δ ⊢ (*AE-op opp v1 v2*) ⇒ ⦃*z*:*b*|*c*⦄
proof −
have *xx*: Θ; Φ; {||}; Γ; Δ ⊢ *LET x* = (*AE-op opp v1 v2*) *IN s* ⇐ τ **using** *assms* **by** *simp*
then show *?thesis* **using** *check-s-elims*(*2*)[*OF xx*] **by** *meson*
qed

lemma *infer-pair*:
assumes Θ ; *B*; Γ ⊢ *v* ⇒ ⦃ *z* : *B-pair b1 b2* | *c* ⦄ **and** *supp v* = {}
obtains *v1* **and** *v2* **where** *v* = *V-pair v1 v2*
using *assms* **proof**(*nominal-induct v rule*: *v.strong-induct*)
case (*V-lit x*)
then show *?case* **by** *auto*
next
case (*V-var x*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next
case (*V-pair x1a x2a*)
then show *?case* **by** *auto*
next
case (*V-cons x1a x2a x3*)
then show *?case* **by** *auto*
next
case (*V-consp x1a x2a x3 x4*)
then show *?case* **by** *auto*
qed

lemma *progress-fst*:
assumes Θ; Φ; {||}; Γ; Δ ⊢ *LET x* = (*AE-fst v*) *IN s* ⇐ τ **and** Θ ⊢ δ ∼ Δ **and**
supp (*LET x* = (*AE-fst v*) *IN s*) ⊆ *atom'fst'setD* Δ
shows ∃δ′ *s*′. Φ ⊢ ⟨ δ , *LET x* = (*AE-fst v*) *IN s* ⟩ ⟶ ⟨δ′, *s*′⟩
proof −
have ∗:*supp v* = {} **using** *assms s-branch-s-branch-list.supp* **by** *auto*
obtain *z* **and** *b* **and** *c* **where** Θ; Φ; {||}; Γ; Δ ⊢ (*AE-fst v*) ⇒ ⦃ *z* : *b* | *c* ⦄
using *check-s-elims*(*2*) **using** *assms* **by** *meson*
moreover obtain *z*′ **and** *b*′ **and** *c*′ **where** Θ ; {||} ; Γ ⊢ *v* ⇒ ⦃ *z*′ : *B-pair b b*′ | *c*′ ⦄
using *infer-e-elims*(*8*) **using** *calculation* **by** *auto*
moreover then obtain *v1* **and** *v2* **where** *V-pair v1 v2* = *v*
using ∗ *infer-pair* **by** *metis*
ultimately show *?thesis* **using** *reduce-let-fstI assms* **by** *metis*
qed

lemma *progress-let*:

assumes Θ; Φ; {||}; Γ; Δ ⊢ *LET* *x* = *e* *IN* *s* ⇐ τ **and** Θ ⊢ δ ∼ Δ **and**
supp (*LET* *x* = *e* *IN* *s*) ⊆ *atom* ' *fst* ' *setD* Δ **and** *sble* Θ Γ
shows ∃δ′ *s*′. Φ ⊢ ⟨ δ , *LET* *x* = *e* *IN* *s*⟩ ⟶ ⟨ δ′ , *s*′⟩
proof −
obtain *z* *b* *c* **where** ∗: Θ; Φ; {||}; Γ; Δ ⊢ *e* ⇒ ⦃ *z* : *b* | *c* ⦄ **using** *check-s-elims*(*2*)[*OF* *assms*(*1*)] **by** *metis*
have ∗∗: *supp* *e* ⊆ *atom* ' *fst* ' *setD* Δ **using** *assms* *s-branch-s-branch-list.supp* **by** *auto*
from ∗ ∗∗ *assms* **show** *?thesis* **proof**(*nominal-induct* ⦃ *z* : *b* | *c* ⦄ *rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ ℬ Γ Δ Φ *v*)
then show *?case* **using** *reduce-stmt-elims* *reduce-let-valI* **by** *metis*
next
case (*infer-e-plusI* Θ ℬ Γ Δ Φ *v1* *z1* *c1* *v2* *z2* *c2* *z3*)
hence *vf*: *supp* *v1* = {} ∧ *supp* *v2* = {} **by** *force*
then obtain *n1* **and** *n2* **where** ∗: *v1* = *V-lit* (*L-num* *n1*) ∧ *v2* = (*V-lit* (*L-num* *n2*)) **using** *infer-int* *infer-e-plusI* **by** *metis*
then show *?case* **using** *reduce-let-plusI* ∗ **by** *metis*
next
case (*infer-e-leqI* Θ ℬ Γ Δ Φ *v1* *z1* *c1* *v2* *z2* *c2* *z3*)
hence *vf*: *supp* *v1* = {} ∧ *supp* *v2* = {} **by** *force*
then obtain *n1* **and** *n2* **where** ∗: *v1* = *V-lit* (*L-num* *n1*) ∧ *v2* = (*V-lit* (*L-num* *n2*)) **using** *infer-int* *infer-e-leqI* **by** *metis*
then show *?case* **using** *reduce-let-leqI* ∗ **by** *metis*
next
case (*infer-e-eqI* Θ ℬ Γ Δ Φ *v1* *z1* *bb* *c1* *v2* *z2* *c2* *z3*)
hence *vf*: *supp* *v1* = {} ∧ *supp* *v2* = {} **by** *force*
then obtain *n1* **and** *n2* **where** ∗: *v1* = *V-lit* *n1* ∧ *v2* = (*V-lit* *n2*) **using** *infer-lit* *infer-e-eqI* **by** *metis*
then show *?case* **using** *reduce-let-eqI* **by** *blast*
next
case (*infer-e-appI* Θ ℬ Γ Δ Φ *f* *x* *b* *c* τ′ *s*′ *v*)
then show *?case* **using** *reduce-let-appI* **by** *metis*
next
case (*infer-e-appPI* Θ ℬ Γ Δ Φ *b*′ *f* *bv* *x* *b* *c* τ′ *s*′ *v*)
then show *?case* **using** *reduce-let-appPI* **by** *metis*
next
case (*infer-e-fstI* Θ ℬ Γ Δ Φ *v* *z*′ *b2* *c* *z*)
hence *supp* *v* = {} **by** *force*
then obtain *v1* **and** *v2* **where** *v* = *V-pair* *v1* *v2* **using** *infer-e-fstI* *infer-pair* **by** *metis*
then show *?case* **using** *reduce-let-fstI* ∗ **by** *metis*
next
case (*infer-e-sndI* Θ ℬ Γ Δ Φ *v* *z*′ *b1* *c* *z*)
hence *supp* *v* = {} **by** *force*
then obtain *v1* **and** *v2* **where** *v* = *V-pair* *v1* *v2* **using** *infer-e-sndI* *infer-pair* **by** *metis*
then show *?case* **using** *reduce-let-sndI* ∗ **by** *metis*
next
case (*infer-e-lenI* Θ ℬ Γ Δ Φ *v* *z*′ *c* *za*)
hence *supp* *v* = {} **by** *force*
then obtain *bvec* **where** *v* = *V-lit* (*L-bitvec* *bvec*) **using** *infer-e-lenI* *infer-bitvec* **by** *metis*
then show *?case* **using** *reduce-let-lenI* ∗ **by** *metis*
next
case (*infer-e-mvarI* Θ ℬ Γ Φ Δ *u*)
hence (*u*, ⦃ *z* : *b* | *c* ⦄) ∈ *setD* Δ **using** *infer-e-elims*(*10*) **by** *meson*

then obtain *v* **where** (*u*,*v*) ∈ *set* δ **using** *infer-e-mvarI delta-sim-delta-lookup* **by** *meson*
then show *?case* **using** *reduce-let-mvar* **by** *metis*
next
case (*infer-e-concatI* Θ *B* Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence *vf*: *supp v1* = {} ∧ *supp v2* = {} **by** *force*
then obtain *n1* **and** *n2* **where** ∗: *v1* = *V-lit* (*L-bitvec n1*) ∧ *v2* = (*V-lit* (*L-bitvec n2*)) **using** *infer-bitvec infer-e-concatI* **by** *metis*
then show *?case* **using** *reduce-let-concatI* ∗ **by** *metis*
next
case (*infer-e-splitI* Θ *B* Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
hence *vf*: *supp v1* = {} ∧ *supp v2* = {} **by** *force*
then obtain *n1* **and** *n2* **where** ∗: *v1* = *V-lit* (*L-bitvec n1*) ∧ *v2* = (*V-lit* (*L-num n2*)) **using** *infer-bitvec infer-e-splitI check-int* **by** *metis*

have *0* ≤ *n2* ∧ *n2* ≤ *int* (*length n1*) **using** *check-v-range*[*OF* - ∗] *infer-e-splitI* **by** *simp*
then obtain *bv1* **and** *bv2* **where** *split n2 n1* (*bv1* , *bv2*) **using** *obtain-split* **by** *metis*
then show *?case* **using** *reduce-let-splitI* ∗ **by** *metis*
qed
qed

lemma *check-css-lookup-branch-exist*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows
Θ; Φ; *B*; *G*; Δ ⊢ *s* ⇐ τ ⟹ *True* **and**
check-branch-s Θ Φ {||} *GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
Θ; Φ; *B*; Γ; Δ; *tid*; *dclist*; *v* ⊢ *css* ⇐ τ ⟹ (*dc*, *t*) ∈ *set dclist* ⟹
∃ *x*′ *s*′. *Some* (*AS-branch dc x*′ *s*′) = *lookup-branch dc css*
proof(*nominal-induct* τ **and** τ **and** τ *rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-branch-list-consI* Θ Φ *B* Γ Δ *tid cons const v cs* τ *dclist css*)
then show *?case* **using** *lookup-branch.simps check-branch-list-finalI* **by** *force*
next
case (*check-branch-list-finalI* Θ Φ *B* Γ Δ *tid cons const v cs* τ)
then show *?case* **using** *lookup-branch.simps check-branch-list-finalI* **by** *force*
qed(*auto*+)

lemma *progress-aux*:
shows Θ; Φ; *B*; Γ; Δ ⊢ *s* ⇐ τ ⟹ *B* = {||} ⟹ *sble* Θ Γ ⟹ *supp s* ⊆ *atom* ‘ *fst* ‘ *setD* Δ ⟹ Θ ⊢ δ ∼ Δ ⟹
(∃ *v*. *s* = [*v*]^s) ∨ (∃ δ′ *s*′. Φ ⊢ ⟨δ, *s*⟩ ⟶ ⟨δ′, *s*′⟩) **and**
Θ; Φ; {||}; Γ; Δ; *tid*; *dc*; *const*; *v2* ⊢ *cs* ⇐ τ ⟹ *supp cs* = {} ⟹ *True*
Θ; Φ; {||}; Γ; Δ; *tid*; *dclist*; *v2* ⊢ *css* ⇐ τ ⟹ *supp css* = {} ⟹ *True*
proof(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Δ Θ Γ *v* τ′ τ)
then show *?case* **by** *auto*
next
case (*check-letI x* Θ Φ *B* Γ Δ *e* τ *z s b c*)
hence Θ; Φ; {||}; Γ; Δ ⊢ *AS-let x e s* ⇐ τ **using** *Typing.check-letI* **by** *meson*
then show *?case* **using** *progress-let check-letI* **by** *metis*
next
case (*check-branch-s-branchI* Θ *B* Γ Δ τ *const x* Φ *tid cons v s*)
then show *?case* **by** *auto*
next

case (*check-branch-list-consI* Θ Φ 𝓑 Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **by** *auto*
next
case (*check-branch-list-finalI* Θ Φ 𝓑 Γ Δ *tid dclist v cs* τ)
then show *?case* **by** *auto*
next
case (*check-ifI z* Θ Φ 𝓑 Γ Δ *v s1 s2* τ)
have *supp v* = {} **using** *check-ifI s-branch-s-branch-list.supp* **by** *auto*
hence *v* = *V-lit L-true* ∨ *v* = *V-lit L-false* **using** *check-bool-options check-ifI* **by** *auto*
then show *?case* **using** *reduce-if-falseI reduce-if-trueI check-ifI* **by** *meson*
next
case (*check-let2I x* Θ Φ 𝓑 *G* Δ *t s1* τ *s2*)
then consider (∃ *v*. *s1* = *AS-val v*) | (∃ δ′ *a*. Φ ⊢ ⟨δ, *s1*⟩ ⟶ ⟨δ′, *a*⟩) **by** *auto*
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **using** *reduce-let2-valI* **by** *fast*
next
case *2*
then show *?thesis* **using** *reduce-let2I check-let2I* **by** *meson*
qed
next
case (*check-varI u* Θ Φ 𝓑 Γ Δ τ′ *v* τ *s*)

obtain *uu*::*u* **where** *uf*: *atom uu* ♯ (*u*,δ,*s*) **using** *obtain-fresh* **by** *blast*
obtain *sa* **where** (*uu* ↔ *u*) · *s* = *sa* **by** *presburger*
moreover have *atom uu* ♯ *s* **using** *uf fresh-prod3* **by** *auto*
ultimately have *AS-var uu* τ′ *v sa* = *AS-var u* τ′ *v s* **using** *s-branch-s-branch-list.eq-iff*(*7*) *Abs1-eq-iff*(*3*)[*of uu sa u s*] **by** *auto*

moreover have *atom uu* ♯ δ **using** *uf fresh-prod3* **by** *auto*
ultimately have Φ ⊢ ⟨δ, *AS-var u* τ′ *v s*⟩ ⟶ ⟨(*uu*, *v*) # δ, *sa*⟩
using *reduce-varI uf* **by** *metis*
then show *?case* **by** *auto*
next
case (*check-assignI* Δ *u* τ *P G v z* τ′)
then show *?case* **using** *reduce-assignI* **by** *blast*
next
case (*check-whileI* Θ Φ 𝓑 Γ Δ *s1 z s2* τ′)
obtain *x*::*x* **where** *atom x* ♯ (*s1*,*s2*) **using** *obtain-fresh* **by** *metis*
moreover obtain *z*::*x* **where** *atom z* ♯ *x* **using** *obtain-fresh* **by** *metis*
ultimately show *?case* **using** *reduce-whileI* **by** *fast*
next
case (*check-seqI P* Φ 𝓑 *G* Δ *s1 z s2* τ)
thus *?case* **proof**(*cases* ∃ *v*. *s1* = *AS-val v*)
case *True*
then obtain *v* **where** *v*: *s1* = *AS-val v* **by** *blast*
hence *supp v* = {} **using** *check-seqI* **by** *auto*
have ∃ *z1 c1*. *P*; 𝓑; *G* ⊢ *v* ⇒ (⦃ *z1* : *B-unit* | *c1* ⦄) **proof** −
obtain *t* **where** *t*:*P*; 𝓑; *G* ⊢ *v* ⇒ *t* ∧ *P*; 𝓑 ; *G* ⊢ *t* ≲ (⦃ *z* : *B-unit* | *TRUE* ⦄)
using *v check-seqI*(*1*) *check-s-elims*(*1*) **by** *blast*
obtain *z1* **and** *b1* **and** *c1* **where** *teq*: *t* = (⦃ *z1* : *b1* | *c1* ⦄) **using** *obtain-fresh-z* **by** *meson*
hence *b1* = *B-unit* **using** *subtype-eq-base t* **by** *meson*

thus *?thesis* **using** *t teq* **by** *fast*
qed
then obtain *z1* **and** *c1* **where** *P* ; *ℬ* ; *G* ⊢ *v* ⇒ (⦃ *z1* : *B-unit* | *c1* ⦄) **by** *auto*
hence *v* = *V-lit L-unit* **using** *infer-v-unit-form* ‹*supp v* = {}› **by** *simp*
hence *s1* = *AS-val* (*V-lit L-unit*) **using** *v* **by** *auto*
then show *?thesis* **using** *check-seqI reduce-seq1I* **by** *meson*
next
case *False*
then show *?thesis* **using** *check-seqI reduce-seq2I*
by (*metis Un-subset-iff s-branch-s-branch-list.supp*(*9*))
qed

next
case (*check-caseI* Θ Φ *ℬ* Γ Δ *tid dclist v cs* τ *z*)
hence *supp v* = {} **by** *auto*

then obtain *v′* **and** *dc* **and** *t*::τ **where** *v*: *v* = *V-cons tid dc v′* ∧ (*dc*, *t*) ∈ *set dclist*
using *check-v-tid-form check-caseI* **by** *metis*
obtain *z* **and** *b* **and** *c* **where** *teq*: *t* = (⦃ *z* : *b* | *c* ⦄) **using** *obtain-fresh-z* **by** *meson*

moreover then obtain *x′ s′* **where** *Some* (*AS-branch dc x′ s′*) = *lookup-branch dc cs* **using** *v teq check-caseI check-css-lookup-branch-exist* **by** *metis*
ultimately show *?case* **using** *reduce-caseI v check-caseI dc-of.cases* **by** *metis*
next
case (*check-assertI x* Θ Φ *ℬ* Γ Δ *c* τ *s*)
hence *sps*: *supp s* ⊆ *atom* ‘ *fst* ‘ *setD* Δ **by** *auto*
have *atom x* ♯ *c* **using** *check-assertI* **by** *auto*
have *atom x* ♯ Γ **using** *check-assertI check-s-wf wfG-elims* **by** *metis*
have *sble* Θ ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) **proof** −
obtain *i′* **where** *i′*: *i′* ⊨ Γ ∧ Θ; Γ ⊢ *i′* **using** *check-assertI sble-def* **by** *metis*
obtain *i*::*valuation* **where** *i*:*i* = *i′* (*x* ↦ *SBool True*) **by** *auto*

have *i* ⊨ (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ **proof** −
have *i′* ⊨ *c* **using** *valid.simps i′ check-assertI* **by** *metis*
hence *i* ⊨ *c* **using** *is-satis-weakening-x i* ‹*atom x* ♯ *c*› **by** *auto*
moreover have *i* ⊨ Γ **using** *is-satis-g-weakening-x i′ i check-assertI* ‹*atom x* ♯ Γ› **by** *metis*
ultimately show *?thesis* **using** *is-satis-g.simps i* **by** *auto*
qed
moreover have Θ ; ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) ⊢ *i* **proof**(*rule wfI-cons*)
show ‹ *i′* ⊨ Γ › **using** *i′* **by** *auto*
show ‹ Θ ; Γ ⊢ *i′*› **using** *i′* **by** *auto*
show ‹*i* = *i′*(*x* ↦ *SBool True*)› **using** *i* **by** *auto*
show ‹ Θ ⊢ *SBool True*: *B-bool*› **using** *wfRCV-BBoolI* **by** *auto*
show ‹*atom x* ♯ Γ› **using** *check-assertI check-s-wf wfG-elims* **by** *auto*
qed
ultimately show *?thesis* **using** *sble-def* **by** *auto*
qed
then consider (∃ *v*. *s* = $[v]^s$) | (∃ δ′ *a*. Φ ⊢ ⟨δ, *s*⟩ ⟶ ⟨ δ′, *a*⟩) **using** *check-assertI sps* **by** *metis*
hence (∃ δ′ *a*. Φ ⊢ ⟨δ, *ASSERT c IN s*⟩ ⟶ ⟨δ′, *a*⟩) **proof**(*cases*)
case *1*
then show *?thesis* **using** *reduce-assert1I* **by** *metis*
next

case *2*
then show *?thesis* **using** *reduce-assert2I* **by** *metis*
qed
thus *?case* **by** *auto*
qed

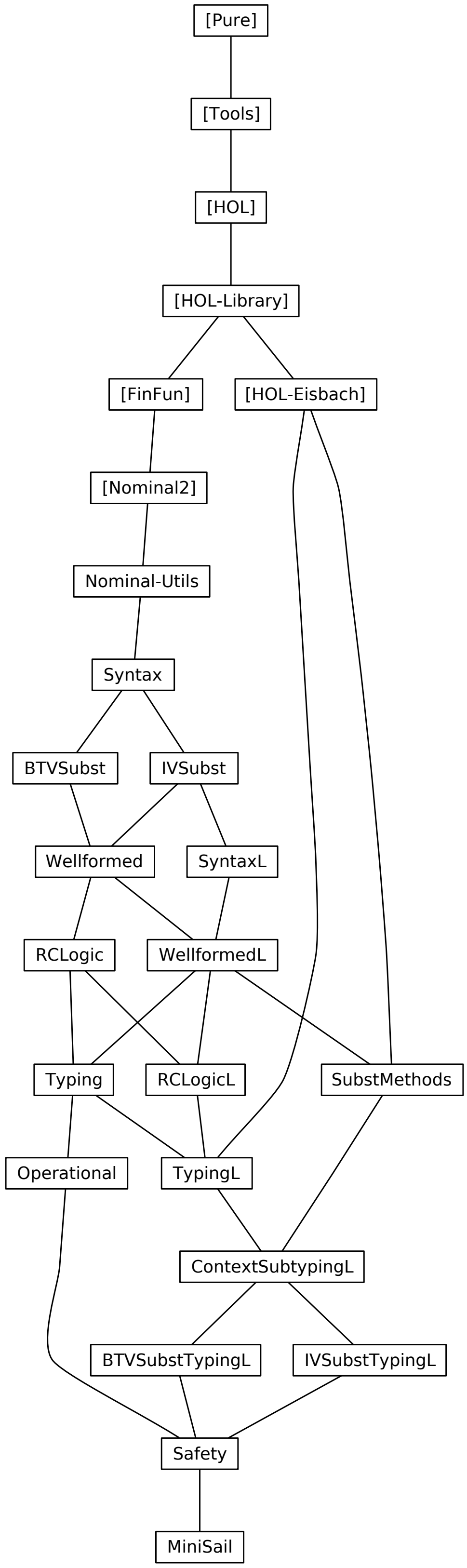
lemma *progress*:
assumes Θ; Φ; $\Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$
shows $(\exists\, v.\ s = [v]^s) \vee (\exists\, \delta'\ s'.\ \Phi\ \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$
proof −
have Θ; Φ; $\{||\}$; *GNil*; $\Delta \vdash s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$
using *config-type-elims*[*OF assms*(*1*)] **by** *auto+*
moreover hence *supp s* $\subseteq$ *atom ‘ fst ‘ setD* Δ **using** *check-s-wf wfS-supp* **by** *fastforce*
moreover have *sble* Θ *GNil* **using** *sble-def wfI-def is-satis-g.simps* **by** *simp*
ultimately show *?thesis* **using** *progress-aux* **by** *blast*
qed

16.4 Safety

lemma *safety-stmt*:
assumes $\Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$ **and** Θ; Φ; $\Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$
shows $(\exists\, v.\ s' = [v]^s) \vee (\exists\, \delta''\ s''.\ \Phi\ \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$
using *preservation-many progress assms* **by** *meson*

lemma *safety*:
assumes $\vdash \langle PROG\ \Theta\ \Phi\ \mathcal{G}\ s \rangle \Leftarrow \tau$ **and** $\Phi\ \vdash \langle \delta\text{-}of\ \mathcal{G}, s \rangle \longrightarrow^* \langle \delta', s' \rangle$
shows $(\exists\, v.\ s' = [v]^s) \vee (\exists\, \delta''\ s''.\ \Phi\ \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$
using *assms config-type-prog-elims safety-stmt* **by** *metis*

end


[Pure]
[Tools]
[HOL]
[HOL-Library]
[FinFun]
[HOL-Eisbach]
[Nominal2]
Nominal-Utils
Syntax
BTVSubst
IVSubst
Wellformed
SyntaxL
RCLogic
WellformedL
Typing
RCLogicL
SubstMethods
Operational
TypingL
ContextSubtypingL
BTVSubstTypingL
IVSubstTypingL
Safety
MiniSail

Bibliography

[1] A. Armstrong, C. Pulte, S. Flur, I. Stark, N. Krishnaswami, P. Sewell, T. Bauereiss, B. Campbell, A. Reid, K. E. Gray, R. M. Norton, P. Mundkur, M. Wassell, and J. French. ISA semantics for ARMv8-A, RISC-V, and CHERI-MIPS. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–31, 2019.

[2] N. Vazou, E. L. Seidel, and S. Peyton-jones. Refinement Types For Haskell. *ICFP '14 Proceedings of the 19th ACM SIGPLAN international conference on Functional programming*, 2014.