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** $?l \longleftrightarrow ?r$)
⟨*proof*⟩

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** $?l \longleftrightarrow ?r$)
⟨*proof*⟩

lemma *fun-eqvtI*:
assumes *f-eqvt*[*eqvt*]: $(\bigwedge\ p\ x.\ p \cdot (f\ x) = f\ (p \cdot x))$
shows $p \cdot f = f$ ⟨*proof*⟩

lemma *eqvt-at-apply*:
assumes *eqvt-at* $f\ x$
shows $(p \cdot f)\ x = f\ x$
⟨*proof*⟩

lemma *eqvt-at-apply'*:
assumes *eqvt-at* $f\ x$
shows $p \cdot f\ x = f\ (p \cdot x)$
⟨*proof*⟩

lemma *eqvt-at-apply''*:
assumes *eqvt-at* $f\ x$
shows $(p \cdot f)\ (p \cdot x) = f\ (p \cdot x)$
⟨*proof*⟩

lemma *size-list-eqvt*[*eqvt*]: $p \cdot$ *size-list* $f\ x =$ *size-list* $(p \cdot f)\ (p \cdot x)$
⟨*proof*⟩

1.2 Freshness via equivariance

lemma *eqvt-fresh-cong1*: $(\bigwedge p\ x.\ p \cdot (f\ x) = f\ (p \cdot x)) \Longrightarrow a \sharp x \Longrightarrow a \sharp f\ x$
⟨*proof*⟩

lemma *eqvt-fresh-cong2*:
assumes *eqvt*: $(\bigwedge p\ x\ y.\ p \cdot (f\ x\ y) = f\ (p \cdot x)\ (p \cdot y))$
and *fresh1*: $a \sharp x$ **and** *fresh2*: $a \sharp y$
shows $a \sharp f\ x\ y$
⟨*proof*⟩

lemma *eqvt-fresh-star-cong1*:
assumes *eqvt*: $(\bigwedge p\ x.\ p \cdot (f\ x) = f\ (p \cdot x))$
and *fresh1*: $a \sharp* x$
shows $a \sharp* f\ x$
⟨*proof*⟩

lemma *eqvt-fresh-star-cong2*:
assumes *eqvt*: $(\bigwedge p\ x\ y.\ p \cdot (f\ x\ y) = f\ (p \cdot x)\ (p \cdot y))$
and *fresh1*: $a \sharp* x$ **and** *fresh2*: $a \sharp* y$
shows $a \sharp* f\ x\ y$
⟨*proof*⟩

lemma *eqvt-fresh-cong3*:
assumes *eqvt*: $(\bigwedge p\ x\ y\ z.\ p \cdot (f\ x\ y\ z) = f\ (p \cdot x)\ (p \cdot y)\ (p \cdot z))$
and *fresh1*: $a \sharp x$ **and** *fresh2*: $a \sharp y$ **and** *fresh3*: $a \sharp z$
shows $a \sharp f\ x\ y\ z$
⟨*proof*⟩

lemma *eqvt-fresh-star-cong3*:
assumes *eqvt*: $(\bigwedge p\ x\ y\ z.\ p \cdot (f\ x\ y\ z) = f\ (p \cdot x)\ (p \cdot y)\ (p \cdot z))$
and *fresh1*: $a \sharp* x$ **and** *fresh2*: $a \sharp* y$ **and** *fresh3*: $a \sharp* z$
shows $a \sharp* f\ x\ y\ z$
⟨*proof*⟩

1.3 Additional simplification rules

lemma *not-self-fresh*[*simp*]: $atom\ x \sharp x \longleftrightarrow False$
⟨*proof*⟩

lemma *fresh-star-singleton*: $\{\ x\ \} \sharp* e \longleftrightarrow x \sharp e$
⟨*proof*⟩

1.4 Additional equivariance lemmas

lemma *eqvt-cases*:
fixes $f\ x\ \pi$
assumes *eqvt*: $\bigwedge x.\ \pi \cdot f\ x = f\ (\pi \cdot x)$
obtains $f\ x\ f\ (\pi \cdot x) \mid \neg\ f\ x \quad \neg\ f\ (\pi \cdot x)$
⟨*proof*⟩

lemma *range-eqvt*: $\pi \cdot range\ Y = range\ (\pi \cdot Y)$

⟨*proof*⟩

lemma *case-option-eqvt*[*eqvt*]:
$\pi \cdot$ *case-option d f x* = *case-option* $(\pi \cdot d)$ $(\pi \cdot f)$ $(\pi \cdot x)$
⟨*proof*⟩

lemma *supp-option-eqvt*:
supp (*case-option d f x*) ⊆ *supp d* ∪ *supp f* ∪ *supp x*
⟨*proof*⟩

lemma *funpow-eqvt*[*simp*,*eqvt*]:
$\pi \cdot ((f :: 'a \Rightarrow 'a{::}pt)$ ^^ $n) = (\pi \cdot f)$ ^^ $(\pi \cdot n)$
⟨*proof*⟩

lemma *delete-eqvt*[*eqvt*]:
$\pi \cdot$ *AList.delete x* Γ = *AList.delete* $(\pi \cdot x)$ $(\pi \cdot \Gamma)$
⟨*proof*⟩

lemma *restrict-eqvt*[*eqvt*]:
$\pi \cdot$ *AList.restrict S* Γ = *AList.restrict* $(\pi \cdot S)$ $(\pi \cdot \Gamma)$
⟨*proof*⟩

lemma *supp-restrict*:
supp (*AList.restrict S* Γ) ⊆ *supp* Γ
⟨*proof*⟩

lemma *clearjunk-eqvt*[*eqvt*]:
$\pi \cdot$ *AList.clearjunk* Γ = *AList.clearjunk* $(\pi \cdot \Gamma)$
⟨*proof*⟩

lemma *map-ran-eqvt*[*eqvt*]:
$\pi \cdot$ *map-ran f* Γ = *map-ran* $(\pi \cdot f)$ $(\pi \cdot \Gamma)$
⟨*proof*⟩

lemma *dom-perm*:
dom $(\pi \cdot f) = \pi \cdot$ (*dom f*)
⟨*proof*⟩

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

lemma *ran-perm*[*simp*]:
$\pi \cdot$ (*ran f*) = *ran* $(\pi \cdot f)$
⟨*proof*⟩

lemma *map-add-eqvt*[*eqvt*]:
$\pi \cdot (m1 ++ m2) = (\pi \cdot m1) ++ (\pi \cdot m2)$
⟨*proof*⟩

lemma *map-of-eqvt*[*eqvt*]:
$\pi \cdot$ *map-of l* = *map-of* $(\pi \cdot l)$
⟨*proof*⟩

lemma *concat-eqvt*[*eqvt*]: $\pi \cdot concat\ l = concat\ (\pi \cdot l)$
⟨*proof*⟩

lemma *tranclp-eqvt*[*eqvt*]: $\pi \cdot tranclp\ P\ v_1\ v_2 = tranclp\ (\pi \cdot P)\ (\pi \cdot v_1)\ (\pi \cdot v_2)$
⟨*proof*⟩

lemma *rtranclp-eqvt*[*eqvt*]: $\pi \cdot rtranclp\ P\ v_1\ v_2 = rtranclp\ (\pi \cdot P)\ (\pi \cdot v_1)\ (\pi \cdot v_2)$
⟨*proof*⟩

lemma *Set-filter-eqvt*[*eqvt*]: $\pi \cdot Set.filter\ P\ S = Set.filter\ (\pi \cdot P)\ (\pi \cdot S)$
⟨*proof*⟩

lemma *Sigma-eqvt'*[*eqvt*]: $\pi \cdot Sigma = Sigma$
⟨*proof*⟩

lemma *override-on-eqvt*[*eqvt*]:
$\pi \cdot (override\text{-}on\ m1\ m2\ S) = override\text{-}on\ (\pi \cdot m1)\ (\pi \cdot m2)\ (\pi \cdot S)$
⟨*proof*⟩

lemma *card-eqvt*[*eqvt*]:
$\pi \cdot (card\ S) = card\ (\pi \cdot S)$
⟨*proof*⟩

lemma *Projl-permute*:
assumes *a*: $\exists y.\ f = Inl\ y$
shows $(p \cdot (Sum\text{-}Type.projl\ f)) = Sum\text{-}Type.projl\ (p \cdot f)$
⟨*proof*⟩

lemma *Projr-permute*:
assumes *a*: $\exists y.\ f = Inr\ y$
shows $(p \cdot (Sum\text{-}Type.projr\ f)) = Sum\text{-}Type.projr\ (p \cdot f)$
⟨*proof*⟩

1.5 Freshness lemmas

lemma *fresh-list-elem*:
assumes $a \sharp \Gamma$
and $e \in set\ \Gamma$
shows $a \sharp e$
⟨*proof*⟩

lemma *set-not-fresh*:
$x \in set\ L \Longrightarrow \neg(atom\ x \sharp L)$
⟨*proof*⟩

lemma *pure-fresh-star*[*simp*]: $a \sharp* (x :: 'a :: pure)$
⟨*proof*⟩

lemma *supp-set-mem*: $x \in set\ L \Longrightarrow supp\ x \subseteq supp\ L$
⟨*proof*⟩

lemma *set-supp-mono*: *set* $L \subseteq$ *set* $L2 \Longrightarrow$ *supp* $L \subseteq$ *supp* $L2$
⟨*proof*⟩

lemma *fresh-star-at-base*:
fixes x :: $'a$ :: *at-base*
shows S ♯* $x \longleftrightarrow$ *atom* $x \notin S$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *fresh-subset*:
finite $B \Longrightarrow x$ ♯ (B :: $'a$::*at-base set*) $\Longrightarrow A \subseteq B \Longrightarrow x$ ♯ A
⟨*proof*⟩

lemma *fresh-star-subset*:
finite $B \Longrightarrow x$ ♯* (B :: $'a$::*at-base set*) $\Longrightarrow A \subseteq B \Longrightarrow x$ ♯* A
⟨*proof*⟩

lemma *fresh-star-set-subset*:
x ♯* (B :: $'a$::*at-base list*) $\Longrightarrow$ *set* $A \subseteq$ *set* $B \Longrightarrow x$ ♯* A
⟨*proof*⟩

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* ($\pi \cdot e$)
⟨*proof*⟩

lemma *fv-Nil*[*simp*]: *fv* [] = {}
⟨*proof*⟩
lemma *fv-Cons*[*simp*]: *fv* (x # xs) = *fv* x ∪ *fv* xs
⟨*proof*⟩
lemma *fv-Pair*[*simp*]: *fv* (x, y) = *fv* x ∪ *fv* y
⟨*proof*⟩
lemma *fv-append*[*simp*]: *fv* (x @ y) = *fv* x ∪ *fv* y
⟨*proof*⟩
lemma *fv-at-base*[*simp*]: *fv* a = {a::$'a$::*at-base*}
⟨*proof*⟩
lemma *fv-pure*[*simp*]: *fv* (a::$'a$::*pure*) = {}
⟨*proof*⟩

lemma *fv-set-at-base*[*simp*]: *fv* (l :: ($'a$ :: *at-base*) *list*) = *set* l
⟨*proof*⟩

lemma *flip-not-fv*: $a \notin$ *fv* $x \Longrightarrow b \notin$ *fv* $x \Longrightarrow (a \leftrightarrow b) \cdot x = x$

⟨*proof*⟩

lemma *fv-not-fresh*: *atom x* ♯ *e* ⟷ *x* ∉ *fv e*
⟨*proof*⟩

lemma *fresh-fv*: *finite* (*fv e* :: ′*a set*) ⟹ *atom* (*x* :: (′*a*::*at-base*)) ♯ (*fv e* :: ′*a set*) ⟷ *atom x* ♯ *e*
⟨*proof*⟩

lemma *finite-fv*[*simp*]: *finite* (*fv* (*e*::′*a*::*fs*) :: (′*b*::*at-base*) *set*)
⟨*proof*⟩

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*⟩

lemma *fresh-fv-list*[*simp*]:
a ♯ (*fv-list e* :: ′*b*::*at-base list*) ⟷ *a* ♯ (*fv e* :: ′*b*::*at-base set*)
⟨*proof*⟩

1.8 Other useful lemmas

lemma *pure-permute-id*: *permute p* = (λ *x*. (*x*::′*a*::*pure*))
⟨*proof*⟩

lemma *supp-set-elem-finite*:
assumes *finite S*
and (*m*::′*a*::*fs*) ∈ *S*
and *y* ∈ *supp m*
shows *y* ∈ *supp S*
⟨*proof*⟩

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

lemma *mem-permute-set*:
shows *x* ∈ *p* · *S* ⟷ (− *p* · *x*) ∈ *S*
⟨*proof*⟩

lemma *flip-set-both-not-in*:
assumes *x* ∉ *S* **and** *x*′ ∉ *S*
shows ((*x*′ ↔ *x*) · *S*) = *S*
⟨*proof*⟩

lemma *inj-atom*: *inj atom* ⟨*proof*⟩

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

lemma *eqvt-uncurry*: *eqvt f* ⟹ *eqvt* (*case-prod f*)
⟨*proof*⟩

lemma *supp-fun-app-eqvt2*:

assumes *a*: *eqvt f*
shows *supp* (*f x y*) ⊆ *supp x* ∪ *supp y*
⟨*proof*⟩

lemma *supp-fun-app-eqvt3*:
assumes *a*: *eqvt f*
shows *supp* (*f x y z*) ⊆ *supp x* ∪ *supp y* ∪ *supp z*
⟨*proof*⟩

lemma *permute-0*[*simp*]: *permute 0* = (λ *x*. *x*)
⟨*proof*⟩
lemma *permute-comp*[*simp*]: *permute x* ∘ *permute y* = *permute* (*x* + *y*) ⟨*proof*⟩

lemma *map-permute*: *map* (*permute p*) = *permute p*
⟨*proof*⟩

lemma *fresh-star-restrictA*[*intro*]: *a* ♯* Γ ⟹ *a* ♯* *AList.restrict V* Γ
⟨*proof*⟩

lemma *Abs-lst-Nil-eq*[*simp*]: [[]]*lst*. (*x*::′*a*::*fs*) = [*xs*]*lst*. *x*′ ⟷ (([],*x*) = (*xs*, *x*′))
⟨*proof*⟩

lemma *Abs-lst-Nil-eq2*[*simp*]: [*xs*]*lst*. (*x*::′*a*::*fs*) = [[]]*lst*. *x*′ ⟷ ((*xs*,*x*) = ([], *x*′))
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *fresh-prod3*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*,*c*) = (*x* ♯ *a* ∧ *x* ♯ *b* ∧ *x* ♯ *c*)
⟨*proof*⟩

lemma *fresh-prod4* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d)$
⟨*proof*⟩

lemma *fresh-prod5* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e)$
⟨*proof*⟩

lemma *fresh-prod6* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f)$
⟨*proof*⟩

lemma *fresh-prod7* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g)$
⟨*proof*⟩

lemma *fresh-prod8* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h)$
⟨*proof*⟩

lemma *fresh-prod9* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i)$
⟨*proof*⟩

lemma *fresh-prod10* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i,j) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i \wedge x \sharp j)$
⟨*proof*⟩

lemma *fresh-prod12* [*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i,j,k,l) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i \wedge x \sharp j \wedge x \sharp k \wedge x \sharp l)$
⟨*proof*⟩

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 $x \sharp (x1,x2)$ ⟨*proof*⟩

lemma *fresh-prod3I*:
fixes x **and** $x1$ **and** $x2$ **and** $x3$
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$
shows $x \sharp (x1,x2,x3)$ ⟨*proof*⟩

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 $x \sharp (x1,x2,x3,x4)$ ⟨*proof*⟩

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 $x \sharp (x1,x2,x3,x4,x5)$ ⟨*proof*⟩

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*⟩

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

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*⟩

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*⟩

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*⟩

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 $xs2 \neq []$
⟨*proof*⟩

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** $[[atom\ y1]]lst.\ xs1 = [[atom\ y2]]lst.\ xs2$
⟨*proof*⟩

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 $t1=t2$
⟨*proof*⟩

lemma *lst-supp*:
assumes $[[atom\ x1]]lst.\ t1 = [[atom\ x2]]lst.\ t2$
shows $supp\ t1 - \{atom\ x1\} = supp\ t2 - \{atom\ x2\}$
⟨*proof*⟩

lemma *lst-supp-subset*:
assumes $[[atom\ x1]]lst.\ t1 = [[atom\ x2]]lst.\ t2$ **and** $supp\ t1 \subseteq \{atom\ x1\} \cup B$
shows $supp\ t2 \subseteq \{atom\ x2\} \cup B$
⟨*proof*⟩

lemma *projl-inl-eqvt*:
fixes $\pi :: perm$
shows $\pi \cdot (projl\ (Inl\ x)) = projl\ (Inl\ (\pi \cdot x))$
⟨*proof*⟩

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* ⦄)]
⟨*proof*⟩

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*
⟨*proof*⟩

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

shows *atom x* ♯ *u*
⟨*proof*⟩

lemma *x-not-in-b-set*[*simp*]:
fixes *x*::*x* **and** *bs*::*bv fset*
shows *atom x* ∉ *supp bs*
⟨*proof*⟩

lemma *x-fresh-b*[*simp*]:
fixes *x*::*x* **and** *b*::*b*
shows *atom x* ♯ *b*
⟨*proof*⟩

lemma *x-fresh-bv*[*simp*]:
fixes *x*::*x* **and** *bv*::*bv*
shows *atom x* ♯ *bv*
⟨*proof*⟩

lemma *u-not-in-x-atoms*[*simp*]:
fixes *u*::*u* **and** *x*::*x* **and** *xs*::*x set*
shows *atom u* ∉ *atom‘xs*
⟨*proof*⟩

lemma *bv-not-in-x-atoms*[*simp*]:
fixes *bv*::*bv* **and** *x*::*x* **and** *xs*::*x set*
shows *atom bv* ∉ *atom‘xs*
⟨*proof*⟩

lemma *u-not-in-b-atoms*[*simp*]:
fixes *b* :: *b* **and** *u*::*u*
shows *atom u* ∉ *supp b*
⟨*proof*⟩

lemma *u-not-in-b-set*[*simp*]:
fixes *u*::*u* **and** *bs*::*bv fset*
shows *atom u* ∉ *supp bs*
⟨*proof*⟩

lemma *u-fresh-b*[*simp*]:
fixes *x*::*u* **and** *b*::*b*
shows *atom x* ♯ *b*
⟨*proof*⟩

lemma *supp-b-v-disjoint*:
fixes *x*::*x* **and** *bv*::*bv*
shows *supp* (*V-var x*) ∩ *supp* (*B-var bv*) = {}
⟨*proof*⟩

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

lemma *u-fresh-bv*[*simp*]:
fixes *u*::*u* **and** *b*::*bv*
shows *atom u* ♯ *b*
⟨*proof*⟩

Basic Types

nominal-function *b-of* :: τ ⇒ *b* **where**
b-of ⦃ *z* : *b* | *c* ⦄ = *b*
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *supp-b-empty*[*simp*]:
fixes *b* :: *b* **and** *x*::*x*
shows *atom x* ∉ *supp b*
⟨*proof*⟩

lemma *flip-b-id*[*simp*]:
fixes *x*::*x* **and** *b*::*b*
shows $(x \leftrightarrow x') \cdot b = b$
⟨*proof*⟩

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** $(x \leftrightarrow y) \cdot bv = bv$
⟨*proof*⟩

lemma *flip-bv-x-cancel*[*simp*]:
fixes *bv*::*bv* **and** *z*::*bv* **and** *x*::*x*
shows $(bv \leftrightarrow z) \cdot x = x$ ⟨*proof*⟩

lemma *flip-bv-u-cancel*[*simp*]:
fixes *bv*::*bv* **and** *z*::*bv* **and** *x*::*u*
shows $(bv \leftrightarrow z) \cdot x = x$ ⟨*proof*⟩

Literals

lemma *supp-bitvec-empty*:
fixes *bv*::*bit list*
shows *supp bv* = {}
⟨*proof*⟩

lemma *bitvec-pure*[*simp*]:
fixes *bv*::*bit list* **and** *x*::*x*
shows *atom x* ♯ *bv* ⟨*proof*⟩

lemma *supp-l-empty*[*simp*]:
fixes *l*:: *l*
shows *supp* (*V-lit l*) = {}
⟨*proof*⟩

lemma *type-l-nosupp*[*simp*]:

fixes x::x **and** l::l
shows *atom* $x \notin$ *supp* ($\{\!\!\{$ z : b | $[[z]^v]^{ce}$ == $[[l]^v]^{ce}$ $\}\!\!\}$)
⟨*proof*⟩

lemma *flip-bitvec0*:
fixes x::*bit list*
assumes *atom* c ♯ (z, x, z')
shows $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$
⟨*proof*⟩

lemma *flip-bitvec*:
assumes *atom* c ♯ (z, *L-bitvec* x, z')
shows $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$
⟨*proof*⟩

lemma *type-l-eq*:
shows $\{\!\!\{$ z : b | $[[z]^v]^{ce}$ == $[V\text{-}lit\ l]^{ce}$ $\}\!\!\}$ = ($\{\!\!\{$ z' : b | $[[z']^v]^{ce}$ == $[V\text{-}lit\ l]^{ce}$ $\}\!\!\}$)
⟨*proof*⟩

lemma *flip-l-eq*:
fixes x::l
shows $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$
⟨*proof*⟩

lemma *flip-l-eq1*:
fixes x::l
assumes $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x'$
shows $x' = x$
⟨*proof*⟩

Types

lemma *flip-base-eq*:
fixes b::b **and** x::x **and** y::x
shows $(x \leftrightarrow y) \cdot b = b$
⟨*proof*⟩

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 | c' $\}\!\!\}$) = ($\{\!\!\{$ z : b | $(z \leftrightarrow z') \cdot c'$ $\}\!\!\}$)
⟨*proof*⟩

lemma *has-fresh-z*:
fixes t::$'b$::*fs*
shows $\exists z\ b\ c.$ *atom* z ♯ t $\wedge$ τ = $\{\!\!\{$ z : b | c $\}\!\!\}$
⟨*proof*⟩

lemma *obtain-fresh-z*:
fixes t::$'b$::*fs*
obtains z **and** b **and** c **where** *atom* z ♯ t $\wedge$ τ = $\{\!\!\{$ z : b | c $\}\!\!\}$
⟨*proof*⟩

lemma *has-fresh-z2*:
fixes $t::'b::fs$
shows $\exists z\ c.\ atom\ z\ \sharp\ t \wedge \tau = \{\!\!\{\ z : b\text{-}of\ \tau\ |\ c\ \}\!\!\}$
⟨*proof*⟩

lemma *obtain-fresh-z2*:
fixes $t::'b::fs$
obtains z **and** c **where** $atom\ z\ \sharp\ t \wedge \tau = \{\!\!\{\ z : b\text{-}of\ \tau\ |\ c\ \}\!\!\}$
⟨*proof*⟩

Values

lemma *u-notin-supp-v*[*simp*]:
fixes $u::u$ **and** $v::v$
shows $atom\ u \notin supp\ v$
⟨*proof*⟩

lemma *u-fresh-xv*[*simp*]:
fixes $u::u$ **and** $x::x$ **and** $v::v$
shows $atom\ u\ \sharp\ (x,v)$
⟨*proof*⟩

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 $atom\ c\ \sharp\ (v, va)$ **and** $atom\ c\ \sharp\ (x, xa, v, va)$ **and** $(x \leftrightarrow c) \bullet v = (xa \leftrightarrow c) \bullet va$
shows $((v = V\text{-}lit\ l \longrightarrow (\exists l'.\ va = V\text{-}lit\ l' \wedge (x \leftrightarrow c) \bullet l = (xa \leftrightarrow c) \bullet l'))) \wedge$
$((v = V\text{-}var\ y \longrightarrow (\exists y'.\ va = V\text{-}var\ y' \wedge (x \leftrightarrow c) \bullet y = (xa \leftrightarrow c) \bullet y'))) \wedge$
$((v = V\text{-}pair\ vone\ vtwo \longrightarrow (\exists v1'\ v2'.\ va = V\text{-}pair\ v1'\ v2' \wedge (x \leftrightarrow c) \bullet vone = (xa \leftrightarrow c) \bullet v1' \wedge (x \leftrightarrow c) \bullet vtwo = (xa \leftrightarrow c) \bullet v2'))) \wedge$
$((v = V\text{-}cons\ tyid\ dc\ vone \longrightarrow (\exists v1'.\ va = V\text{-}cons\ tyid\ dc\ v1' \wedge (x \leftrightarrow c) \bullet vone = (xa \leftrightarrow c) \bullet v1'))) \wedge$
$((v = V\text{-}consp\ tyid\ dc\ b\ vone \longrightarrow (\exists v1'.\ va = V\text{-}consp\ tyid\ dc\ b\ v1' \wedge (x \leftrightarrow c) \bullet vone = (xa \leftrightarrow c) \bullet v1')))$
⟨*proof*⟩

lemma *flip-eq*:
fixes $x::x$ **and** $xa::x$ **and** $s::'a::fs$ **and** $sa::'a::fs$
assumes $(\forall c.\ atom\ c\ \sharp\ (s, sa) \longrightarrow atom\ c\ \sharp\ (x, xa, s, sa) \longrightarrow (x \leftrightarrow c) \bullet s = (xa \leftrightarrow c) \bullet sa)$ **and** $x \neq xa$
shows $(x \leftrightarrow xa) \bullet s = sa$
⟨*proof*⟩

lemma *swap-v-supp*:
fixes $v::v$ **and** $d::x$ **and** $z::x$
assumes $atom\ d\ \sharp\ v$
shows $supp\ ((z \leftrightarrow d) \bullet v) \subseteq supp\ v - \{\ atom\ z\ \} \cup \{\ atom\ d\}$
⟨*proof*⟩

Expressions

lemma *swap-e-supp*:

fixes $e::e$ **and** $d::x$ **and** $z::x$
assumes *atom d* ♯ *e*
shows *supp* $((z \leftrightarrow d) \cdot e) \subseteq$ *supp e* $- \{$ *atom z* $\} \cup \{$ *atom d* $\}$
⟨*proof*⟩

lemma *swap-ce-supp*:
fixes $e::ce$ **and** $d::x$ **and** $z::x$
assumes *atom d* ♯ *e*
shows *supp* $((z \leftrightarrow d) \cdot e) \subseteq$ *supp e* $- \{$ *atom z* $\} \cup \{$ *atom d* $\}$
⟨*proof*⟩

lemma *swap-c-supp*:
fixes $c::c$ **and** $d::x$ **and** $z::x$
assumes *atom d* ♯ *c*
shows *supp* $((z \leftrightarrow d) \cdot c) \subseteq$ *supp c* $- \{$ *atom z* $\} \cup \{$ *atom d* $\}$
⟨*proof*⟩

lemma *type-e-eq*:
assumes *atom z* ♯ *e* **and** *atom z′* ♯ *e*
shows $\{\!|\ z : b \mid [[z]^v]^{ce} == e\ |\!\} = (\{\!|\ z' : b \mid [[z']^v]^{ce} == e\ |\!\})$
⟨*proof*⟩

lemma *type-e-eq2*:
assumes *atom z* ♯ *e* **and** *atom z′* ♯ *e* **and** $b=b'$
shows $\{\!|\ z : b \mid [[z]^v]^{ce} == e\ |\!\} = (\{\!|\ z' : b' \mid [[z']^v]^{ce} == e\ |\!\})$
⟨*proof*⟩

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

lemma *fresh-opp-all*:
fixes $opp::opp$
shows *z* ♯ *opp*
⟨*proof*⟩

lemma *fresh-e-opp-all*:
shows (*z* ♯ *v1* ∧ *z* ♯ *v2*) = *z* ♯ *AE-op opp v1 v2*
⟨*proof*⟩

lemma *fresh-e-opp*:
fixes $z::x$

assumes $atom\ z \sharp v1 \wedge atom\ z \sharp v2$
shows $atom\ z \sharp AE\text{-}op\ opp\ v1\ v2$
⟨*proof*⟩

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 $(s = AS\text{-}val\ w \longrightarrow (\exists w'.\ sa = AS\text{-}val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$
$(s = AS\text{-}seq\ s1\ s2 \longrightarrow (\exists s1'\ s2'.\ sa = AS\text{-}seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2') \vee$
$(s = AS\text{-}if\ v\ s1\ s2 \longrightarrow (\exists v'\ s1'\ s2'.\ sa = AS\text{-}if\ seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2' \wedge (x \leftrightarrow c) \cdot c = (xa \leftrightarrow c) \cdot v')$
⟨*proof*⟩

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* $x*b*c\ \Gamma$ (**infixr** ‹$\#_\Gamma$› *65*)

datatype Δ =
DNil (‹$[]_\Delta$›)
| *DCons* $u*\tau\ \Delta$ (**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*⟩

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 ⟨*proof*⟩
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*⟩

lemma Φ-*induct* [*case-names PNil PConsNone PConsSome*] : P [] $\Longrightarrow$ ($\bigwedge f\ x\ b\ c\ \tau\ s'\ \Phi'$. $P\ \Phi' \Longrightarrow P$ ((*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ s'))) # Φ')) $\Longrightarrow$
($\bigwedge f\ bv\ x\ b\ c\ \tau\ s'\ \Phi'$. $P\ \Phi' \Longrightarrow P$ ((*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ s'))) # Φ')) $\Longrightarrow P\ \Phi$
⟨*proof*⟩

lemma Θ-*induct* [*case-names TNil AF-typedef AF-typedef-poly*] : P [] $\Longrightarrow$ ($\bigwedge$*tid dclist* Θ'. $P\ \Theta' \Longrightarrow P$ ((*AF-typedef tid dclist*) # Θ')) $\Longrightarrow$
($\bigwedge$*tid bv dclist* Θ'. $P\ \Theta' \Longrightarrow P$ ((*AF-typedef-poly tid bv dclist*) # Θ')) $\Longrightarrow P\ \Theta$
⟨*proof*⟩

instantiation Γ :: *pt*
begin

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

instance ⟨*proof*⟩
end

lemmas [*eqvt*] = *permute*-Γ.*simps*

lemma *G-cons-eqvt*[*simp*]:
fixes $\Gamma{::}\Gamma$
shows $p \cdot ((x,b,c)\ \#_\Gamma\ \Gamma) = ((p \cdot x,\ p \cdot b\ ,\ p \cdot c)\ \#_\Gamma\ (p \cdot \Gamma\ ))$ (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *G-cons-flip*[*simp*]:
fixes $x{::}x$ **and** $\Gamma{::}\Gamma$
shows $(x \leftrightarrow x') \cdot ((x'',b,c)\ \#_\Gamma\ \Gamma) = (((x \leftrightarrow x') \cdot x'',\ b\ ,\ (x \leftrightarrow x') \cdot c)\ \#_\Gamma\ ((x \leftrightarrow x') \cdot \Gamma\ ))$
⟨*proof*⟩

lemma *G-cons-flip-fresh*[*simp*]:
fixes $x{::}x$ **and** $\Gamma{::}\Gamma$
assumes *atom* $x \sharp (c,\Gamma)$ **and** *atom* $x' \sharp (c,\Gamma)$
shows $(x \leftrightarrow x') \cdot ((x',b,c)\ \#_\Gamma\ \Gamma) = ((x,\ b\ ,\ c)\ \#_\Gamma\ \Gamma\ )$
⟨*proof*⟩

lemma *G-cons-flip-fresh2*[*simp*]:
fixes $x{::}x$ **and** $\Gamma{::}\Gamma$
assumes *atom* $x \sharp (c,\Gamma)$ **and** *atom* $x' \sharp (c,\Gamma)$
shows $(x \leftrightarrow x') \cdot ((x,b,c)\ \#_\Gamma\ \Gamma) = ((x',\ b\ ,\ c)\ \#_\Gamma\ \Gamma\ )$
⟨*proof*⟩

lemma *G-cons-flip-fresh3*[*simp*]:

fixes $x::x$ **and** $\Gamma::\Gamma$
assumes $atom\ x \sharp \Gamma$ **and** $atom\ x' \sharp \Gamma$
shows $(x \leftrightarrow x') \cdot ((x',b,c)\ \#_\Gamma\ \Gamma) = ((x,\ \ b\ ,\ (x \leftrightarrow x') \cdot c)\ \#_\Gamma\ \Gamma\ )$
⟨*proof*⟩

lemma *neq-GNil-conv*: $(xs \neq GNil) = (\exists\, y\ ys.\ xs = y\ \#_\Gamma\ ys)$
⟨*proof*⟩

nominal-function *toList* :: $\Gamma \Rightarrow (x*b*c)$ *list* **where**
toList GNil = []
| *toList* (*GCons xbc G*) = *xbc*#(*toList G*)
⟨*proof*⟩
nominal-termination (*eqvt*)
⟨*proof*⟩

nominal-function *toSet* :: $\Gamma \Rightarrow (x*b*c)$ *set* **where**
toSet GNil = {}
| *toSet* (*GCons xbc G*) = {*xbc*} ∪ (*toSet G*)
⟨*proof*⟩
nominal-termination (*eqvt*)
⟨*proof*⟩

nominal-function *append-g* :: $\Gamma \Rightarrow \Gamma \Rightarrow \Gamma$ (**infixr** ‹@› *65*) **where**
append-g GNil g = *g*
| *append-g* $(xbc\ \#_\Gamma\ g1)\ g2 = (xbc\ \#_\Gamma\ (g1@g2))$
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

nominal-function *dom* :: $\Gamma \Rightarrow x$ *set* **where**
dom Γ = (*fst*‘ (*toSet* Γ))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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* :: $\Gamma \Rightarrow$ *atom set* **where**
atom-dom Γ = *atom*‘(*dom* Γ)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

2.2.3 Immutable Variable Context Lemmas

lemma *append-GNil*[*simp*]:
GNil @ *G* = *G*
⟨*proof*⟩

lemma *append-g-toSetU* [*simp*]: *toSet* (*G1*@*G2*) = *toSet G1* ∪ *toSet G2*
⟨*proof*⟩

lemma *supp-GNil*:
shows *supp GNil* = {}
⟨*proof*⟩

lemma *supp-GCons*:
  fixes *xs*::Γ
  shows $supp\ (x\ \#_\Gamma\ xs) = supp\ x \cup supp\ xs$
  ⟨*proof*⟩

lemma *atom-dom-eq*[*simp*]:
  fixes *G*::Γ
  shows $atom\text{-}dom\ ((x,\ b,\ c)\ \#_\Gamma\ G) = atom\text{-}dom\ ((x,\ b,\ c')\ \#_\Gamma\ G)$
  ⟨*proof*⟩

lemma *dom-append*[*simp*]:
  $atom\text{-}dom\ (\Gamma @ \Gamma') = atom\text{-}dom\ \Gamma \cup atom\text{-}dom\ \Gamma'$
  ⟨*proof*⟩

lemma *dom-cons*[*simp*]:
  $atom\text{-}dom\ ((x,b,c)\ \#_\Gamma\ G) = \{\ atom\ x\ \} \cup atom\text{-}dom\ G$
  ⟨*proof*⟩

lemma *fresh-GNil*[*ms-fresh*]:
  shows $a \sharp\ GNil$
  ⟨*proof*⟩

lemma *fresh-GCons*[*ms-fresh*]:
  fixes *xs*::Γ
  shows $a \sharp\ (x\ \#_\Gamma\ xs) \longleftrightarrow a \sharp\ x \wedge a \sharp\ xs$
  ⟨*proof*⟩

lemma *dom-supp-g*[*simp*]:
  $atom\text{-}dom\ G \subseteq supp\ G$
  ⟨*proof*⟩

lemma *fresh-append-g*[*ms-fresh*]:
  fixes *xs*::Γ
  shows $a \sharp\ (xs\ @\ ys) \longleftrightarrow a \sharp\ xs \wedge a \sharp\ ys$
  ⟨*proof*⟩

lemma *append-g-assoc*:
  fixes *xs*::Γ
  shows $(xs\ @\ ys)\ @\ zs = xs\ @\ (ys\ @\ zs)$
  ⟨*proof*⟩

lemma *append-g-inside*:
  fixes *xs*::Γ
  shows $xs\ @\ (x\ \#_\Gamma\ ys) = (xs\ @\ (x\ \#_\Gamma\ GNil))\ @\ ys$
  ⟨*proof*⟩

lemma *finite-Γ*:
  $finite\ (toSet\ \Gamma)$
  ⟨*proof*⟩

lemma *supp-Γ*:

$supp\ \Gamma = supp\ (toSet\ \Gamma)$
⟨proof⟩

lemma *supp-of-subset*:
fixes G::($'a$::*fs set*)
assumes *finite* G **and** *finite* G' **and** $G \subseteq G'$
shows $supp\ G \subseteq supp\ G'$
⟨proof⟩

lemma *supp-weakening*:
assumes $toSet\ G \subseteq toSet\ G'$
shows $supp\ G \subseteq supp\ G'$
⟨proof⟩

lemma *fresh-weakening*[*ms-fresh*]:
assumes $toSet\ G \subseteq toSet\ G'$ **and** $x \sharp G'$
shows $x \sharp G$
⟨proof⟩

instance Γ :: *fs*
⟨proof⟩

lemma *fresh-gamma-elem*:
fixes Γ::Γ
assumes $a \sharp \Gamma$
and $e \in toSet\ \Gamma$
shows $a \sharp e$
⟨proof⟩

lemma *fresh-gamma-append*:
fixes *xs*::Γ
shows $a \sharp (xs\ @\ ys) \longleftrightarrow a \sharp xs \wedge a \sharp ys$
⟨proof⟩

lemma *supp-triple*[*simp*]:
shows $supp\ (x,\ y,\ z) = supp\ x \cup supp\ y \cup supp\ z$
⟨proof⟩

lemma *supp-append-g*:
fixes *xs*::Γ
shows $supp\ (xs\ @\ ys) = supp\ xs \cup supp\ ys$
⟨proof⟩

lemma *fresh-in-g*[*simp*]:
fixes Γ::Γ **and** x'::x
shows $atom\ x' \sharp \Gamma'\ @\ (x,\ b0,\ c0)\ \#_\Gamma\ \Gamma = (atom\ x' \notin supp\ \Gamma' \cup supp\ x \cup supp\ b0 \cup supp\ c0 \cup supp\ \Gamma)$
⟨proof⟩

lemma *fresh-suffix*[*ms-fresh*]:
fixes Γ::Γ
assumes $atom\ x \sharp \Gamma'@\Gamma$

shows *atom x* ♯ Γ
⟨*proof*⟩

lemma *not-GCons-self* [*simp*]:
fixes *xs*::Γ
shows $xs \neq x \#_\Gamma xs$
⟨*proof*⟩

lemma *not-GCons-self2* [*simp*]:
fixes *xs*::Γ
shows $x \#_\Gamma xs \neq xs$
⟨*proof*⟩

lemma *fresh-restrict*:
fixes *y*::*x* **and** Γ::Γ
assumes *atom y* ♯ $(\Gamma' @ (x, b, c) \#_\Gamma \Gamma)$
shows *atom y* ♯ $(\Gamma'@\Gamma)$
⟨*proof*⟩

lemma *fresh-dom-free*:
assumes *atom x* ♯ Γ
shows $(x,b,c) \notin$ *toSet* Γ
⟨*proof*⟩

lemma Γ-*set-intros*: $x \in$ *toSet* $(x \#_\Gamma xs)$ **and** $y \in$ *toSet xs* $\Longrightarrow y \in$ *toSet* $(x \#_\Gamma xs)$
⟨*proof*⟩

lemma *fresh-dom-free2*:
assumes *atom x* $\notin$ *atom-dom* Γ
shows $(x,b,c) \notin$ *toSet* Γ
⟨*proof*⟩

2.2.4 Mutable Variable Context Lemmas

lemma *supp-DNil*:
shows *supp DNil* = {}
⟨*proof*⟩

lemma *supp-DCons*:
fixes *xs*::Δ
shows *supp* $(x \#_\Delta xs)$ = *supp x* ∪ *supp xs*
⟨*proof*⟩

lemma *fresh-DNil*[*ms-fresh*]:
shows *a* ♯ *DNil*
⟨*proof*⟩

lemma *fresh-DCons*[*ms-fresh*]:
fixes *xs*::Δ
shows *a* ♯ $(x \#_\Delta xs) \longleftrightarrow$ *a* ♯ *x* ∧ *a* ♯ *xs*
⟨*proof*⟩

instance Δ :: *fs*

⟨*proof*⟩

2.2.5 Lookup Functions

nominal-function *lookup* :: Γ ⇒ *x* ⇒ (*b*∗*c*) *option* **where**
lookup GNil x = *None*
| *lookup* ((*x*,*b*,*c*)#$_Γ$*G*) *y* = (*if x*=*y then Some* (*b*,*c*) *else lookup G y*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*)#$_Γ$*G*) *x*′ *c*′ = (*if x*=*x*′ *then* ((*x*,*b*,*c*′)#$_Γ$*G*) *else* (*x*,*b*,*c*)#$_Γ$(*replace-in-g G x*′ *c*′))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

nominal-function *name-of-type* ::*type-def* ⇒ *f* **where**
name-of-type (*AF-typedef f -*) = *f*
| *name-of-type* (*AF-typedef-poly f - -*) = *f*
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

nominal-function *name-of-fun* ::*fun-def* ⇒ *f* **where**
name-of-fun (*AF-fundef f ft*) = *f*
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *neq-DNil-conv*: $(xs \neq DNil) = (\exists\, y\ ys.\ xs = y\ \#_\Delta\ ys)$
⟨*proof*⟩

nominal-function *setD* :: $\Delta \Rightarrow (u{*}\tau)$ *set* **where**
setD DNil = {}
| *setD* (*DCons xbc G*) = {*xbc*} ∪ (*setD G*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩

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* ⟨*proof*⟩

lemma *split-concat*:
assumes *split n v* (*v1*,*v2*)
shows *v* = *append v1 v2*
⟨*proof*⟩

lemma *split-n*:
assumes *split n v* (*v1*,*v2*)
shows *0* ≤ *n* ∧ *n* ≤ *int* (*length v*)
⟨*proof*⟩

lemma *split-length*:
assumes *split n v* (*v1*,*v2*)
shows *n* = *int* (*length v1*)
⟨*proof*⟩

lemma *obtain-split*:
assumes *0* ≤ *n* **and** *n* ≤ *int* (*length bv*)
shows ∃ *bv1 bv2*. *split n bv* (*bv1* , *bv2*)
⟨*proof*⟩

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* :: $'a::fs \Rightarrow x \Rightarrow v \Rightarrow 'a::fs$ (‹-[-::=-]$_v$› [*1000*,*50*,*50*] *1000*)
  assumes *fresh-subst-v-if*: $y \sharp (subst\text{-}v\ a\ x\ v) \longleftrightarrow (atom\ x \sharp a \wedge y \sharp a) \vee (y \sharp v \wedge (y \sharp a \vee y = atom\ x))$
    and *forget-subst-v*[*simp*]: $atom\ x \sharp a \Longrightarrow subst\text{-}v\ a\ \ x\ v = a$
    and *subst-v-id*[*simp*]: $subst\text{-}v\ a\ x\ (V\text{-}var\ x) = a$
    and *eqvt*[*simp*,*eqvt*]: $(p::perm) \cdot (subst\text{-}v\ a\ x\ v) = (subst\text{-}v\ \ (p \cdot a)\ (p \cdot x)\ (p \cdot v))$
    and *flip-subst-v*[*simp*]: $atom\ x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$
    and *subst-v-simple-commute*[*simp*]: $atom\ x \sharp c \Longrightarrow (c[z::=[x]^v]_v)[x::=b]_v = c[z::=b]_v$
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 $atom\ x1 \sharp (z1,z2,c1,c2)$
  shows $(c1[z1::=[x1]^v]_v) = (c2[z2::=[x1]^v]_v)$
⟨*proof*⟩

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*⟩

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*⟩

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*)
  ⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*]:
  $j \sharp t[i{::=}x]_{vv} = ((atom\ i \sharp t \wedge j \sharp t) \vee (j \sharp x \wedge (j \sharp t \vee j = atom\ i)))$
  ⟨*proof*⟩

lemma *forget-subst-vv* [*simp*]: $atom\ a \sharp tm \Longrightarrow tm[a{::=}x]_{vv} = tm$
  ⟨*proof*⟩

lemma *subst-vv-id* [*simp*]: $tm[a{::=}V\text{-}var\ a]_{vv} = tm$
  ⟨*proof*⟩

lemma *subst-vv-commute* [*simp*]:
  $atom\ j \sharp tm \Longrightarrow tm[i{::=}t]_{vv}[j{::=}u]_{vv} = tm[i{::=}t[j{::=}u]_{vv}]_{vv}$
  ⟨*proof*⟩

lemma *subst-vv-commute-full* [*simp*]:
  $atom\ j \sharp t \Longrightarrow atom\ i \sharp u \Longrightarrow i \neq j \Longrightarrow tm[i{::=}t]_{vv}[j{::=}u]_{vv} = tm[j{::=}u]_{vv}[i{::=}t]_{vv}$
  ⟨*proof*⟩

lemma *subst-vv-var-flip*[*simp*]:
  fixes *v*::*v*
  assumes $atom\ y \sharp v$
  shows $(y \leftrightarrow x) \cdot v = v\ [x{::=}V\text{-}var\ y]_{vv}$
  ⟨*proof*⟩

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

definition
  subst-v = *subst-vv*

instance ⟨*proof*⟩

end

3.3 Expressions

nominal-function *subst-ev* :: $e \Rightarrow x \Rightarrow v \ \Rightarrow e$ **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* = $[\#1\ (\textit{subst-vv}\ v'\ x\ v\ )]^e$
| *subst-ev* $[\#2\ v']^e$ *x v* = $[\#2\ (\textit{subst-vv}\ v'\ x\ v\ )]^e$
| *subst-ev* ((*AE-mvar u*)) *x v* = *AE-mvar u*
| *subst-ev* $[|\ v'\ |]^e$ *x v* = $[|\ (\textit{subst-vv}\ \ v'\ x\ v\ )\ |]^e$
| *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*)
⟨*proof*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

abbreviation
subst-ev-abbrev :: $e \Rightarrow x \Rightarrow v \Rightarrow e$ (‹-[-::=-]$_{ev}$› [*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*
⟨*proof*⟩

lemma *forget-subst-ev* [*simp*]: *atom a* ♯ *A* ⟹ *subst-ev A a x* = *A*
⟨*proof*⟩

lemma *subst-ev-id* [*simp*]: *subst-ev A a* (*V-var a*) = *A*
⟨*proof*⟩

lemma *fresh-subst-ev-if* [*simp*]:
j ♯ (*subst-ev A i x*) = ((*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*)))
⟨*proof*⟩

lemma *subst-ev-commute* [*simp*]:
atom j ♯ *A* ⟹ $(A[i::=t]_{ev})[j::=u]_{ev} = A[i::=t[j::=u]_{vv}]_{ev}$
⟨*proof*⟩

lemma *subst-ev-var-flip*[*simp*]:
fixes *e*::*e* **and** *y*::*x* **and** *x*::*x*
assumes *atom y* ♯ *e*
shows (*y* ↔ *x*) • *e* = *e* $[x::=\textit{V-var}\ y]_{ev}$
⟨*proof*⟩

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* ↔ *c*) • *e* = (*xa* ↔ *c*) • *ea*
shows $e[x::=v']_{ev} = ea[xa::=v']_{ev}$
⟨*proof*⟩

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

⟨*proof*⟩

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

definition
subst-v = *subst-ev*

instance ⟨*proof*⟩
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-ev* $(e[z::=w]_{ev})$ *x v* = *subst-ev* $(e[x::=v]_{ev})$ *z w*
⟨*proof*⟩

lemma *subst-ev-v-flip1*[*simp*]:
fixes *e*::*e*
assumes *atom z1* ♯ (*z*,*e*) **and** *atom z1'* ♯ (*z*,*e*)
shows$(z1 \leftrightarrow z1') \cdot e[z::=v]_{ev} = e[z::= ((z1 \leftrightarrow z1') \cdot v)]_{ev}$
⟨*proof*⟩

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*)
⟨*proof*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

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*
⟨*proof*⟩

lemma *forget-subst-cev* [*simp*]: *atom a* ♯ *A* ⟹ *subst-cev A a x* = *A*
⟨*proof*⟩

lemma *subst-cev-id* [*simp*]: *subst-cev A a* (*V-var a*) = *A*
⟨*proof*⟩

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*⟩

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*)
  ⟨*proof*⟩

lemma *subst-cev-var-flip*[*simp*]:
  fixes *e*::*ce* **and** *y*::*x* **and** *x*::*x*
  assumes *atom y* ♯ *e*
  shows (*y* ↔ *x*) • *e* = *e* $[x::=V\text{-}var\ y]_{cev}$
  ⟨*proof*⟩

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* ↔ *c*) • *e* = (*xa* ↔ *c*) • *ea*
  shows $e[x::=v']_{cev}$ = $ea[xa::=v']_{cev}$
⟨*proof*⟩

lemma *subst-cev-var*[*simp*]:
  fixes *z*::*x* **and** *x*::*x*
  shows $[[x]^v]^{ce}$ $[x::=[z]^v]_{cev}$ = $[[z]^v]^{ce}$
  ⟨*proof*⟩

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

definition
  subst-v = *subst-cev*

instance ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *subst-cev-v-flip1*[*simp*]:
  fixes *e*::*ce*
  assumes *atom z1* ♯ (*z*,*e*) **and** *atom z1'* ♯ (*z*,*e*)
  shows(*z1* ↔ *z1'*) • $e[z::=v]_{cev}$ = $e[z::= ((z1 \leftrightarrow z1') \cdot v)]_{cev}$
  ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

abbreviation
subst-cv-abbrev :: *c* ⇒ *x* ⇒ *v* ⇒ *c* (‹-[-::=-]$_{cv}$› [*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*
⟨*proof*⟩

lemma *forget-subst-cv* [*simp*]: *atom a* ♯ *A* ⟹ *subst-cv A a x* = *A*
⟨*proof*⟩

lemma *subst-cv-id* [*simp*]: *subst-cv A a* (*V-var a*) = *A*
⟨*proof*⟩

lemma *fresh-subst-cv-if* [*simp*]:
j ♯ (*subst-cv A i x*) ⟷ (*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*))
⟨*proof*⟩

lemma *subst-cv-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-cv* (*subst-cv A i t*) *j u*) = *subst-cv A i* (*subst-vv t j u*)
⟨*proof*⟩

lemma *let-s-size* [*simp*]: *size s* ≤ *size* (*AS-let x e s*)
⟨*proof*⟩

lemma *subst-cv-var-flip*[*simp*]:
fixes *c*::*c*
assumes *atom y* ♯ *c*
shows (*y* ↔ *x*) · *c* = *c*[*x*::=*V-var y*]$_{cv}$
⟨*proof*⟩

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

definition
subst-v = *subst-cv*

instance ⟨*proof*⟩

end

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

⟨*proof*⟩

lemma *subst-cv-v-flip3*[*simp*]:
fixes $c::c$
assumes *atom z1* ♯ *c* **and** *atom z1'* ♯ *c*
shows$(z1 \leftrightarrow z1') \cdot c[z::=[z1]^v]_{cv} = c[z::=[z1']^v]_{cv}$
⟨*proof*⟩

lemma *subst-cv-v-flip*[*simp*]:
fixes $c::c$
assumes *atom x* ♯ *c*
shows $((x \leftrightarrow z) \cdot c)[x::=v]_{cv} = c\ [z::=v]_{cv}$
⟨*proof*⟩

lemma *subst-cv-commute-full*:
fixes $c::c$
assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** $x \neq z$
shows $(c[z::=w]_{cv})[x::=v]_{cv} = (c[x::=v]_{cv})[z::=w]_{cv}$
⟨*proof*⟩

lemma *subst-cv-eq*[*simp*]:
assumes *atom z1* ♯ *e1*
shows $(\textit{CE-val}\ (\textit{V-var}\ z1) == e1)[z1::=[x]^v]_{cv} = (\textit{CE-val}\ (\textit{V-var}\ x) == e1)$ (**is** *?A* = *?B*)
⟨*proof*⟩

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 $\textit{subst-gv} :: \Gamma \Rightarrow x \Rightarrow v \Rightarrow \Gamma$ **where**
subst-gv GNil x v = *GNil*
| $\textit{subst-gv}\ ((y,b,c)\ \#_\Gamma\ \Gamma)\ x\ v = (\textit{if}\ x = y\ \textit{then}\ \Gamma\ \textit{else}\ ((y,b,c[x::=v]_{cv})\#_\Gamma\ (\textit{subst-gv}\ \Gamma\ x\ v)))$
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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

lemma *size-subst-gv* [*simp*]: *size* (*subst-gv G i x*) ≤ *size G*
⟨*proof*⟩

lemma *forget-subst-gv* [*simp*]: *atom a* ♯ *G* ⟹ *subst-gv G a x* = *G*
⟨*proof*⟩

lemma *fresh-subst-gv*: *atom a* ♯ *G* ⟹ *atom a* ♯ *v* ⟹ *atom a* ♯ *subst-gv G x v*
⟨*proof*⟩

lemma *subst-gv-flip*:
fixes $x::x$ **and** $xa::x$ **and** $z::x$ **and** $c::c$ **and** $b::b$ **and** $\Gamma::\Gamma$

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) = $(xa, b, c[z::=V\text{-}var\ xa]_{cv})$ $\#_\Gamma$ Γ
⟨*proof*⟩

3.7 Types

nominal-function *subst-tv* :: τ ⇒ *x* ⇒ *v* ⇒ τ **where**
atom z ♯ (*x*,*v*) ⟹ *subst-tv* ⦃ *z* : *b* | *c* ⦄ *x v* = ⦃ *z* : *b* | $c[x::=v]_{cv}$ ⦄
⟨*proof*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

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

lemma *size-subst-tv* [*simp*]: *size* (*subst-tv A i x*) = *size A*
⟨*proof*⟩

lemma *forget-subst-tv* [*simp*]: *atom a* ♯ *A* ⟹ *subst-tv A a x* = *A*
⟨*proof*⟩

lemma *subst-tv-id* [*simp*]: *subst-tv A a* (*V-var a*) = *A*
⟨*proof*⟩

lemma *fresh-subst-tv-if* [*simp*]:
j ♯ (*subst-tv A i x*) ⟷ (*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*))
⟨*proof*⟩

lemma *subst-tv-commute* [*simp*]:
atom y ♯ τ ⟹ $(\tau[x::= t]_{\tau v})[y::=v]_{\tau v} = \tau[x::= t[y::=v]_{vv}]_{\tau v}$
⟨*proof*⟩

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*⟩

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

definition
subst-v = *subst-tv*

instance ⟨*proof*⟩

end

lemma *subst-tv-commute-full*:

fixes $c::\tau$
assumes *atom z* $\sharp$ *v* **and** *atom x* $\sharp$ *w* **and** $x \neq z$
shows $(c[z::=w]_{\tau v})[x::=v]_{\tau v} = (c[x::=v]_{\tau v})[z::=w]_{\tau v}$
⟨*proof*⟩

lemma *type-eq-subst-eq*:
fixes $v::v$ **and** $c1::c$
assumes $\{\!| \ z1 : b1 \ | \ c1 \ |\!\} = \{\!| \ z2 : b2 \ | \ c2 \ |\!\}$
shows $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$
⟨*proof*⟩

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 $\sharp$ *x* $\Longrightarrow$ *c-of* (*T-refined-type z b c*) $x = c[z::=[x]^v]_{cv}$
⟨*proof*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *c-of-eq*:
shows *c-of* $\{\!| \ x : b \ | \ c \ |\!\}$ $x = c$
⟨*proof*⟩

lemma *obtain-fresh-z-c-of*:
fixes $t::'b::fs$
obtains *z* **where** *atom z* $\sharp$ $t \wedge \tau = \{\!| \ z : b\text{-}of\ \tau \ | \ c\text{-}of\ \tau\ z \ |\!\}$
⟨*proof*⟩

lemma *c-of-fresh*:
fixes $x::x$
assumes *atom x* $\sharp$ (t,z)
shows *atom x* $\sharp$ *c-of t z*
⟨*proof*⟩

lemma *c-of-switch*:
fixes $z::x$
assumes *atom z* $\sharp$ *t*
shows $(c\text{-}of\ t\ z)[z::=V\text{-}var\ x]_{cv} = c\text{-}of\ t\ x$
⟨*proof*⟩

lemma *type-eq-subst-eq1*:
fixes $v::v$ **and** $c1::c$
assumes $\{\!| \ z1 : b1 \ | \ c1 \ |\!\} = (\{\!| \ z2 : b2 \ | \ c2 \ |\!\})$ **and** *atom z1* $\sharp$ *c2*
shows $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **and** $b1=b2$ **and** $c1 = (z1 \leftrightarrow z2) \cdot c2$
⟨*proof*⟩

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*⟩

lemma *type-eq-subst-eq3*:
fixes *v::v* **and** *c1::c*
assumes $\{\!\!\{ z1 : b1 \mid c1 \}\!\!\} = (\{\!\!\{ z2 : b2 \mid c2 \}\!\!\})$ **and** *atom z1* ♯ *c2*
shows $c1 = c2[z2::=V\text{-}var\ z1]_{cv}$ **and** *b1=b2*
⟨*proof*⟩

lemma *type-eq-flip*:
assumes *atom x* ♯ *c*
shows $\{\!\!\{ z : b \mid c \}\!\!\} = \{\!\!\{ x : b \mid (x \leftrightarrow z) \cdot c \}\!\!\}$
⟨*proof*⟩

lemma *c-of-true*:
$c\text{-}of\ \{\!\!\{ z' : B\text{-}bool \mid TRUE \}\!\!\}\ x = C\text{-}true$
⟨*proof*⟩

lemma *type-eq-subst*:
assumes *atom x* ♯ *c*
shows $\{\!\!\{ z : b \mid c \}\!\!\} = \{\!\!\{ x : b \mid c[z::=[x]^v]_{cv} \}\!\!\}$
⟨*proof*⟩

lemma *type-e-subst-fresh*:
fixes *x::x* **and** *z::x*
assumes *atom z* ♯ *(x,v)* **and** *atom x* ♯ *e*
shows $\{\!\!\{ z : b \mid CE\text{-}val\ (V\text{-}var\ z) == e \}\!\!\}[x::=v]_{\tau v} = \{\!\!\{ z : b \mid CE\text{-}val\ (V\text{-}var\ z) == e \}\!\!\}$
⟨*proof*⟩

lemma *type-v-subst-fresh*:
fixes *x::x* **and** *z::x*
assumes *atom z* ♯ *(x,v)* **and** *atom x* ♯ *v'*
shows $\{\!\!\{ z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v' \}\!\!\}[x::=v]_{\tau v} = \{\!\!\{ z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v' \}\!\!\}$
⟨*proof*⟩

lemma *subst-tbase-eq*:
$b\text{-}of\ \tau = b\text{-}of\ \tau[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *subst-tv-if*:
assumes *atom z1* ♯ *(x,v)* **and** *atom z'* ♯ *(x,v)*
shows $\{\!\!\{ z1 : b \mid CE\text{-}val\ (v'[x::=v]_{vv}) == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ (c'[x::=v]_{cv})[z'::=[z1]^v]_{cv} \}\!\!\} = \{\!\!\{ z1 : b \mid CE\text{-}val\ v' == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ c'[z'::=[z1]^v]_{cv} \}\!\!\}[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *subst-tv-tid*:
assumes *atom za* ♯ *(x,v)*
shows $\{\!\!\{ za : B\text{-}id\ tid \mid TRUE \}\!\!\} = \{\!\!\{ za : B\text{-}id\ tid \mid TRUE \}\!\!\}[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *b-of-subst*:
$b\text{-}of\ (\tau[x::=v]_{\tau v}) = b\text{-}of\ \tau$
⟨*proof*⟩

lemma *subst-tv-flip*:
assumes $\tau'[x::=v]_{\tau v} = \tau$ **and** *atom* $x \sharp (v,\tau)$ **and** *atom* $x' \sharp (v,\tau)$
shows $((x' \leftrightarrow x) \cdot \tau')[x'::=v]_{\tau v} = \tau$
⟨*proof*⟩

lemma *subst-cv-true*:
$\{\!| z : B\text{-}id\ tid \mid TRUE |\!\} = \{\!| z : B\text{-}id\ tid \mid TRUE |\!\}[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *t-eq-supp*:
assumes $(\{\!| z : b \mid c |\!\}) = (\{\!| z1 : b1 \mid c1 |\!\})$
shows $supp\ c - \{ atom\ z \} = supp\ c1 - \{ atom\ z1 \}$
⟨*proof*⟩

lemma *fresh-t-eq*:
fixes *x::x*
assumes $(\{\!| z : b \mid c |\!\}) = (\{\!| zz : b \mid cc |\!\})$ **and** *atom* $x \sharp c$ **and** $x \neq zz$
shows *atom* $x \sharp cc$
⟨*proof*⟩

3.8 Mutable Variable Context

nominal-function *subst-dv* :: $\Delta \Rightarrow x \Rightarrow v \Rightarrow \Delta$ **where**
subst-dv DNil x v = DNil
| *subst-dv* $((u,t) \#_\Delta \Delta)\ x\ v = ((u,t[x::=v]_{\tau v}) \#_\Delta (subst\text{-}dv\ \Delta\ x\ v\ ))$
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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

nominal-function *dmap* :: $(u*\tau \Rightarrow u*\tau) \Rightarrow \Delta \Rightarrow \Delta$ **where**
dmap f DNil = DNil
| *dmap f* $((u,t)\#_\Delta\Delta) = (f\ (u,t)\ \#_\Delta\ (dmap\ f\ \Delta\ ))$
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *subst-dv-iff*:
$\Delta[x::=v]_{\Delta v} = dmap\ (\lambda(u,t).\ (u,\ t[x::=v]_{\tau v}))\ \Delta$
⟨*proof*⟩

lemma *size-subst-dv* [*simp*]: *size* (*subst-dv G i x*) $\leq$ *size G*
⟨*proof*⟩

lemma *forget-subst-dv* [*simp*]: *atom* $a \sharp G \Longrightarrow$ *subst-dv G a x = G*
⟨*proof*⟩

lemma *subst-dv-member*:
assumes $(u,\tau) \in setD\ \Delta$

shows $(u,\ \tau[x::=v]_{\tau v}) \in$ *setD* $(\Delta[x::=v]_{\Delta v})$
⟨*proof*⟩

lemma *fresh-subst-dv*:
fixes *x*::*x*
assumes *atom xa* ♯ Δ **and** *atom xa* ♯ *v*
shows *atom xa* ♯$\Delta[x::=v]_{\Delta v}$
⟨*proof*⟩

lemma *fresh-subst-dv-if*:
fixes *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::Δ
assumes *j* ♯ *t* ∧ *j* ♯ *x*
shows (*j* ♯ *subst-dv t i x*)
⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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

$s[x::=v]_{sv}$ $\equiv$ *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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *c-eq-perm*:
assumes ((*atom z*) ⇌ (*atom z′*)) · *c* = *c′* **and** *atom z′* ♯ *c*
shows ⦃ *z* : *b* | *c* ⦄ = ⦃ *z′* : *b* | *c′* ⦄
⟨*proof*⟩

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*⟩

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*⟩

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
⟨*proof*⟩

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

definition
subst-v = *subst-sv*

instance ⟨*proof*⟩
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*]$_{sv}$
⟨*proof*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

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*))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *size-subst-ft*[*simp*]: *size* (*subst-ft-v A x v*) = *size A*
⟨*proof*⟩

lemma *forget-subst-ft* [*simp*]: **shows** *atom x* ♯ *A* ⟹ *subst-ft-v A x a* = *A*
⟨*proof*⟩

lemma *subst-ft-id* [*simp*]: *subst-ft-v A a* (*V-var a*) = *A*
⟨*proof*⟩

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

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

instance ⟨*proof*⟩
end

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

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

instance ⟨*proof*⟩

end

3.11 Variable Context

lemma *subst-dv-fst-eq*:
fst ‘ $setD$ $(\Delta[x::=v]_{\Delta v})$ = fst ‘ $setD$ Δ
⟨*proof*⟩

lemma *subst-gv-member-iff*:
fixes $x'::x$ **and** $x::x$ **and** $v::v$ **and** $c'::c$
assumes $(x',b',c') \in toSet\ \Gamma$ **and** $atom\ x \notin atom\text{-}dom\ \Gamma$
shows $(x',b',c'[x::=v]_{cv}) \in toSet\ \Gamma[x::=v]_{\Gamma v}$
⟨*proof*⟩

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 $(j \sharp subst\text{-}gv\ t\ i\ x)$
⟨*proof*⟩

3.12 Lookup

lemma *set-GConsD*: $y \in toSet\ (x\ \#_\Gamma\ xs) \Longrightarrow y{=}x \vee y \in toSet\ xs$
⟨*proof*⟩

lemma *subst-g-assoc-cons*:
assumes $x \neq x'$
shows $(((x', b', c')\ \#_\Gamma\ \Gamma')[x::=v]_{\Gamma v}\ @\ G) = ((x', b', c'[x::=v]_{cv})\ \#_\Gamma\ ((\Gamma'[x::=v]_{\Gamma v})\ @\ G))$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *subst-b-fresh-x*:
  fixes *tm*::*'a*::*fs* **and** *x*::*x*
  shows *atom x* ♯ *tm* = *atom x* ♯ *tm*[*bv*::=*b'*]$_b$

⟨*proof*⟩

lemma *subst-b-x-flip*[*simp*]:
fixes x'::x **and** x::x **and** bv::bv
shows $((x' \leftrightarrow x) \cdot tm)[bv{::=}b']_b = (x' \leftrightarrow x) \cdot (tm[bv{::=}b']_b)$
⟨*proof*⟩

end

4.2 Base Type

nominal-function *subst-bb* :: $b \Rightarrow bv \Rightarrow b \Rightarrow b$ **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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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

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

instance ⟨*proof*⟩
end

lemma *subst-bb-inject*:
assumes $b1 = b2[bv{::=}b]_{bb}$ **and** $b2 \neq$ *B-var bv*
shows
b1 = *B-int* $\Longrightarrow$ *b2* = *B-int* **and**
b1 = *B-bool* $\Longrightarrow$ *b2* = *B-bool* **and**
b1 = *B-unit* $\Longrightarrow$ *b2* = *B-unit* **and**
b1 = *B-bitvec* $\Longrightarrow$ *b2* = *B-bitvec* **and**
b1 = *B-pair b11 b12* $\Longrightarrow$ ($\exists$ *b11' b12'* . $b11 = b11'[bv{::=}b]_{bb} \wedge b12 = b12'[bv{::=}b]_{bb} \wedge$ *b2* = *B-pair b11' b12'*) **and**
b1 = *B-var bv'* $\Longrightarrow$ *b2* = *B-var bv'* **and**
b1 = *B-id tyid* $\Longrightarrow$ *b2* = *B-id tyid* **and**
b1 = *B-app tyid b11* $\Longrightarrow$ ($\exists$ *b11'*. $b11 = b11'[bv{::=}b]_{bb} \wedge$ *b2* = *B-app tyid b11'*)
⟨*proof*⟩

lemma *flip-b-subst4*:
fixes *b1*::*b* **and** *bv1*::*bv* **and** *c*::*bv* **and** *b*::*b*
assumes *atom c* ♯ (*b1*,*bv1*)

shows $b1[bv1{::=}b]_{bb} = ((bv1 \leftrightarrow c) \cdot b1)[c ::= b]_{bb}$
⟨*proof*⟩

lemma *subst-bb-flip-sym*:
fixes *b1*::*b* **and** *b2*::*b*
assumes *atom c* ♯ *b* **and** *atom c* ♯ (*bv1*,*bv2*, *b1*, *b2*) **and** $(bv1 \leftrightarrow c) \cdot b1 = (bv2 \leftrightarrow c) \cdot b2$
shows $b1[bv1{::=}b]_{bb} = b2[bv2{::=}b]_{bb}$
⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩
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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩
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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩
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*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

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

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

instance ⟨*proof*⟩
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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩
end

4.8 Statements

nominal-function (*default case-sum* (λ*x*. *Inl undefined*) (*case-sum* (λ*x*. *Inl undefined*) (λ*x*. *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*)

⟨*proof*⟩

nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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

$\langle proof \rangle$

lemma *flip-subst-subst-s*:
fixes $bv::bv$ **and** $s::s$ **and** $cs::branch\text{-}s$ **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**
$atom\ bv \sharp css \Longrightarrow subst\text{-}branchlb\ ((bv \leftrightarrow z) \cdot css)\ bv\ v = subst\text{-}branchlb\ css\ z\ v$
$\langle proof \rangle$

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

instance $\langle proof \rangle$
end

4.9 Function Type

nominal-function $subst\text{-}ft\text{-}b :: fun\text{-}typ \Rightarrow bv \Rightarrow b \Rightarrow fun\text{-}typ$ **where**
$subst\text{-}ft\text{-}b\ (\ AF\text{-}fun\text{-}typ\ z\ b\ c\ t\ (s::s))\ x\ v = AF\text{-}fun\text{-}typ\ z\ (subst\text{-}bb\ b\ x\ v)\ (subst\text{-}cb\ c\ x\ v)\ t[x::=v]_{\tau b}$ $s[x::=v]_{sb}$
$\langle proof \rangle$

nominal-termination $(eqvt)$ $\langle proof \rangle$

nominal-function $subst\text{-}ftq\text{-}b :: fun\text{-}typ\text{-}q \Rightarrow bv \Rightarrow b \Rightarrow fun\text{-}typ\text{-}q$ **where**
$atom\ bv \sharp (x,v) \Longrightarrow subst\text{-}ftq\text{-}b\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ ft)\ x\ v = (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (subst\text{-}ft\text{-}b\ ft\ x\ v))$

| $subst\text{-}ftq\text{-}b\ (AF\text{-}fun\text{-}typ\text{-}none\ ft)\ x\ v = (AF\text{-}fun\text{-}typ\text{-}none\ (subst\text{-}ft\text{-}b\ ft\ x\ v))$
$\langle proof \rangle$
nominal-termination $(eqvt)$ $\langle proof \rangle$

instantiation $fun\text{-}typ :: has\text{-}subst\text{-}b$
begin
definition $subst\text{-}b = subst\text{-}ft\text{-}b$

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 $\langle proof \rangle$
end

instantiation $fun\text{-}typ\text{-}q :: has\text{-}subst\text{-}b$
begin
definition $subst\text{-}b = subst\text{-}ftq\text{-}b$

instance $\langle proof \rangle$
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*))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩
end

lemma *subst-b-base-for-lit*:
(*base-for-lit l*)$[bv::=b]_{bb}$ = *base-for-lit l*
⟨*proof*⟩

lemma *subst-b-lookup*:
assumes *Some* (*b*, *c*) = *lookup* Γ *x*
shows *Some* $(b[bv::=b']_{bb},\ c[bv::=b']_{cb})$ = *lookup* $\Gamma[bv::=b']_{\Gamma b}$ *x*
⟨*proof*⟩

lemma *subst-g-b-x-fresh*:
fixes *x*::*x* **and** *b*::*b* **and** Γ::Γ **and** *bv*::*bv*
assumes *atom x* ♯ Γ
shows *atom x* ♯ $\Gamma[bv::=b]_{\Gamma b}$
⟨*proof*⟩

4.10.2 Mutable Variables

nominal-function *subst-db* :: Δ ⇒ *bv* ⇒ *b* ⇒ Δ **where**
subst-db $[]_{\Delta}$ *- -* = $[]_{\Delta}$
| *subst-db* $((u,t)\ \#_{\Delta}\ \Delta)$ *bv b* = $((u,t[bv::=b]_{\tau b})\ \#_{\Delta}$ (*subst-db* Δ *bv b*))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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

instantiation Δ :: *has-subst-b*
begin
definition *subst-b* = *subst-db*

instance ⟨*proof*⟩
end

lemma *subst-d-b-member*:
  assumes $(u, \tau) \in setD\ \Delta$
  shows $(u, \tau[bv::=b]_{\tau b}) \in setD\ \Delta[bv::=b]_{\Delta b}$
  ⟨*proof*⟩

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 $atom\ x \sharp \Delta[bv::=b]_{\Delta b}$
  ⟨*proof*⟩

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**
    $atom\ x \sharp s \Longrightarrow atom\ x \sharp s[bv::=b']_{sb}$
  ⟨*proof*⟩

lemma *subst-b-fresh-u-cls*:
  fixes $tm::'a::has\text{-}subst\text{-}b$ **and** $x::u$
  shows $atom\ x \sharp tm = atom\ x \sharp tm[bv::=b']_b$
  ⟨*proof*⟩

lemma *subst-g-b-u-fresh*:
  fixes $x::u$ **and** $b::b$ **and** $\Gamma::\Gamma$ **and** $bv::bv$
  assumes $atom\ x \sharp \Gamma$
  shows $atom\ x \sharp \Gamma[bv::=b]_{\Gamma b}$
  ⟨*proof*⟩

lemma *subst-d-b-u-fresh*:
  fixes $x::u$ **and** $b::b$ **and** $\Gamma::\Delta$ **and** $bv::bv$
  assumes $atom\ x \sharp \Gamma$
  shows $atom\ x \sharp \Gamma[bv::=b]_{\Delta b}$
  ⟨*proof*⟩

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**
    $atom\ x \sharp s \Longrightarrow atom\ x \sharp s[bv::=b']_{sb}$
  ⟨*proof*⟩

lemma *subst-db-u-fresh*:
  fixes $u::u$ **and** $b::b$ **and** $D::\Delta$
  assumes $atom\ u \sharp D$
  shows $atom\ u \sharp D[bv::=b]_{\Delta b}$
  ⟨*proof*⟩

lemma *flip-bt-subst4*:
  fixes $t::\tau$ **and** $bv::bv$
  assumes $atom\ bv \sharp t$
  shows $t[bv'::=b]_{\tau b} = ((bv' \leftrightarrow bv) \cdot t)[bv::=b]_{\tau b}$
  ⟨*proof*⟩

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 $t1[bv1::=b]_{\tau b} = t2[bv2::=b]_{\tau b}$
  ⟨*proof*⟩

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
$]\!] \Longrightarrow$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$|$ *wfC-eqI*: $\llbracket$
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{e1} : b$;
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{e2} : b \rrbracket \Longrightarrow$
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-eq e1 e2}$
$|$ *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*: $\llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1} ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c2} \rrbracket \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-conj c1 c2}$
$|$ *wfC-disjI*: $\llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1} ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c2} \rrbracket \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-disj c1 c2}$
$|$ *wfC-notI*: $\llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1} \rrbracket \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-not c1}$
$|$ *wfC-impI*: $\llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c1}$;
$\qquad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{c2} \rrbracket \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \textit{C-imp c1 c2}$

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

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

| *wfTh-consI*: $\llbracket$
$(\textit{name-of-type tdef}) \notin \textit{name-of-type} \text{ ` } \textit{set}\ \Theta$;
$\vdash_{wf} \Theta$;
$\Theta \vdash_{wf}\ \textit{tdef}\ \rrbracket \implies \vdash_{wf} \textit{tdef}\#\Theta$

| *wfTD-simpleI*: $\llbracket$
Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ *lst*
$\rrbracket \implies$
$\Theta \vdash_{wf}$ (*AF-typedef s lst*)

| *wfTD-poly*: $\llbracket$
Θ ; $\{|bv|\}$; *GNil* $\vdash_{wf}$ *lst*
$\rrbracket \implies$
$\Theta \vdash_{wf}$ (*AF-typedef-poly s bv lst*)

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

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

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

inductive-cases *wfCE-elims*:
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-val v* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-op Plus v1 v2* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-op LEq v1 v2* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-fst v1* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-snd v1* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-concat v1 v2* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-len v1* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-op opp v1 v2* : *b*
$\Theta;\ \mathcal{B};\ \Gamma \vdash_{wf}$ *CE-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*:
$\Theta; \mathcal{B} \vdash_{wf}$ *GNil*
$\Theta; \mathcal{B} \vdash_{wf} (x,b,c)\#_\Gamma\Gamma$
$\Theta; \mathcal{B} \vdash_{wf} (x,b,TRUE)\#_\Gamma\Gamma$
$\Theta; \mathcal{B} \vdash_{wf} (x,b,FALSE)\#_\Gamma\Gamma$

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

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

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

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

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*: ⟦
$\Theta \vdash_{wf} \Phi$;

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}int};$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v2} : \mathit{B\text{-}int}$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}op\ Plus\ v1\ v2} : \mathit{B\text{-}int}$

| *wfE-leqI*: $[\![$
$\Theta \vdash_{wf} \Phi;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}int};$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v2} : \mathit{B\text{-}int}$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}op\ LEq\ v1\ v2} : \mathit{B\text{-}bool}$

| *wfE-eqI*: $[\![$
$\Theta \vdash_{wf} \Phi;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : b;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v2} : b;$
$b \in \{ \mathit{B\text{-}bool}, \mathit{B\text{-}int}, \mathit{B\text{-}unit} \}$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}op\ Eq\ v1\ v2} : \mathit{B\text{-}bool}$

| *wfE-fstI*: $[\![$
$\Theta \vdash_{wf} \Phi;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}pair\ b1\ b2}$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}fst\ v1} : \mathit{b1}$

| *wfE-sndI*: $[\![$
$\Theta \vdash_{wf} \Phi;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}pair\ b1\ b2}$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} \mathit{AE\text{-}snd\ v1} : \mathit{b2}$

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

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

| *wfE-lenI*: ⟦
  $\Theta \vdash_{wf} \Phi$;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-bitvec*
⟧ $\Longrightarrow$
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-len v1* : *B-int*

| *wfE-appI*: ⟦
  $\Theta \vdash_{wf} \Phi$;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
  Some (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* ;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *b*
⟧ $\Longrightarrow$
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-app f v* : *b-of* τ

| *wfE-appPI*: ⟦
  $\Theta \vdash_{wf} \Phi$;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
  Θ; $\mathcal{B} \vdash_{wf} b'$;
  atom bv ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, b', *v*, (*b-of* τ)[*bv*::=b']$_b$);
  Some (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : (*b*[*bv*::=b']$_b$)
⟧ $\Longrightarrow$
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ (*AE-appP f* b' *v*) : ((*b-of* τ)[*bv*::=b']$_b$)

| *wfE-mvarI*: ⟦
  $\Theta \vdash_{wf} \Phi$;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
  (*u*,τ) $\in$ *setD* Δ
⟧ $\Longrightarrow$
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AE-mvar u* : *b-of* τ

| *wfS-valI*: ⟦
  $\Theta \vdash_{wf} \Phi$;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *b* ;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$
⟧ $\Longrightarrow$
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ (*AS-val v*) : *b*

| *wfS-letI*: ⟦
  wfE Θ Φ $\mathcal{B}$ Γ Δ *e* b' ;
  Θ ; Φ ; $\mathcal{B}$; (*x*,b',*C-true*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf}$ *s* : *b*;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
  atom x ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, *e*, *b*)
⟧ $\Longrightarrow$
  Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *LET x* = *e IN s* : *b*

| *wfS-assertI*: ⟦
  Θ ; Φ ; $\mathcal{B}$; (*x*,*B-bool*,*c*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf}$ *s* : *b*;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *c* ;
  Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;

$atom\ x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s)$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} ASSERT\ c\ IN\ \ s : b$

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

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

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

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

$|$ *wfS-whileI*: $[\![$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : B\text{-}bool$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s2 : b$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} WHILE\ s1\ DO\ \{\ s2\ \} : b$

$|$ *wfS-seqI*: $[\![$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : B\text{-}unit$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s2 : b$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1\ ;;\ s2 : b$

$|$ *wfS-matchI*: $[\![$
$wfV\ \Theta\ \ \mathcal{B}\ \Gamma\ \ v\ \ (B\text{-}id\ tid)$;
$(AF\text{-}typedef\ tid\ dclist\ ) \in set\ \Theta$;
$wfD\ \Theta\ \ \mathcal{B}\ \Gamma\ \ \Delta$;
$\Theta \vdash_{wf} \Phi$;

Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *cs* : *b*
$\rrbracket \Longrightarrow$
Θ; Φ; $\mathcal{B}$; Γ; $\Delta \vdash_{wf}$ *AS-match v cs* : *b*

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

| *wfS-finalI*: $\llbracket$
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b*
$\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; [(*dc*,*t*)] $\vdash_{wf}$ *AS-final cs* : *b*

| *wfS-cons*: $\llbracket$
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b*;
Θ; Φ; $\mathcal{B}$; Γ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b*
$\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; (*dc*,*t*)#*dclist* $\vdash_{wf}$ *AS-cons cs css* : *b*

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

| *wfD-cons*: $\llbracket$
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ::Δ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$;
u $\notin$ *fst* ‘ *setD* Δ
$\rrbracket \Longrightarrow$
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ((*u*,τ) $\#_\Delta$ Δ)

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

| *wfPhi-consI*: $\llbracket$
f $\notin$ *name-of-fun* ‘ *set* Φ;
Θ ; Φ $\vdash_{wf}$ *ft*;
$\Theta \vdash_{wf} \Phi$
$\rrbracket \Longrightarrow$
Θ $\vdash_{wf}$ ((*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* $\vdash_{wf} \tau$;
$\Theta \vdash_{wf} \Phi$
$\rrbracket \Longrightarrow$
Θ ; Φ ; *B* $\vdash_{wf}$ (*AF-fun-typ x b c* τ *s*)

inductive-cases *wfE-elims*:
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-val v* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-op Plus v1 v2* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-op LEq v1 v2* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-fst v1* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-snd v1* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-concat v1 v2* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-len v1* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-op opp v1 v2* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-app f v*: *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-appP f b′ v*: *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-mvar u* : *b*
Θ; Φ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *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*:
Θ $\vdash_{wf}$ []
Θ $\vdash_{wf}$ ((*AF-fundef f ft*)#Π)
Θ $\vdash_{wf}$ (*fd*#Φ::Φ)

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

inductive-cases *wfFTQ-elims*:
Θ ; Φ $\vdash_{wf}$ *AF-fun-typ-none ft*
Θ ; Φ $\vdash_{wf}$ *AF-fun-typ-some bv ft*
Θ ; Φ $\vdash_{wf}$ *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 ; $\mathcal{B}$; (Γ::Γ) $\vdash_{wf}$ (*u*,τ) $\#_\Delta$ Δ::Δ

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*⟩

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* ⟨*proof*⟩

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

(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* ;
atom bv ♯ (Θ, *SConsp s dc b′ s1*, *B-app s b′*);
Θ ⊢ *s1* : *b*[*bv*::=*b′*]$_{bb}$ ⟧ ⟹ Θ ⊢(*SConsp s dc b′ s1*) : (*B-app s b′*)
| *wfRCV-BUnitI*: *P* ⊢ *SUnit* : *B-unit*
| *wfRCV-BVarI*: *P* ⊢ (*SUt n*) : (*B-var bv*)
equivariance *wfRCV*
nominal-inductive *wfRCV*
avoids *wfRCV-BConsPI*: *bv*
⟨*proof*⟩

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* ⊢ *s* ~ *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*:: Θ ⇒ *rcl-val* ⇒ *b* ⇒ (*bv*∗*b*) *option* ⇒ *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* ⟧ ⟹ *wfRCV-subst P* (*SPair s1 s2*) (*B-pair b1 b2*) *sub*
| *wfRCV-subst-BConsI*: ⟦ *AF-typedef s dclist* ∈ *set* Θ;
(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* ;
wfRCV-subst Θ *s1 b None* ⟧ ⟹ *wfRCV-subst* Θ (*SCons s dc s1*) (*B-id s*) *sub*
| *wfRCV-subst-BConspI*: ⟦ *AF-typedef-poly s bv dclist* ∈ *set* Θ;
(*dc*, ⦃ *x* : *b* | *c* ⦄) ∈ *set dclist* ;
wfRCV-subst Θ *s1* (*b*[*bv*::=*b′*]$_{bb}$) *sub* ⟧ ⟹ *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* ≠ *bv* ⟹ *wfRCV-subst P* (*SUt n*) (*B-var bv*) (*Some* (*bvar*,*bin*))
| *wfRCV-subst-BVar2I*: ⟦ *bvar* = *bv*; *wfRCV-subst P s bin None* ⟧ ⟹ *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* ⟨*proof*⟩

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* ⟨*proof*⟩

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*
  ⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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* ⟨*proof*⟩

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* ⟨*proof*⟩

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* ⟨*proof*⟩

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* ⟨*proof*⟩

nominal-function *is-satis-g* :: *valuation* ⇒ Γ ⇒ *bool* (‹ - ⊨ - ›) **where**
i ⊨ *GNil* = *True*
| i ⊨ ((x,b,c) $\#_\Gamma$ G) = (i ⊨ c ∧ i ⊨ G)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

6.2 Validity

nominal-function *valid* :: Θ ⇒ $\mathcal{B}$ ⇒ Γ ⇒ *c* ⇒ *bool* (‹- ; - ; - ⊨ - › [*50*, *50*] *50*) **where**
P ; B ; G ⊨ c = ((P ; B ; G $\vdash_{wf}$ c) ∧ (∀ i. (P ; G ⊢ i) ∧ i ⊨ G ⟶ i ⊨ c))

⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩

end

Chapter 7

Syntax Lemmas

7.1 Support, lookup and contexts

lemma *supp-v-tau* [*simp*]:
assumes $atom\ z \sharp v$
shows $supp\ (\{\!|\ z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ v\ |\!\})= supp\ v \cup supp\ b$
⟨*proof*⟩

lemma *supp-v-var-tau* [*simp*]:
assumes $z \neq x$
shows $supp\ (\{\!|\ z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}var\ x)\ |\!\}) = \{\ atom\ x\ \} \cup supp\ b$
⟨*proof*⟩

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 $\exists\ y::x.\ atom\ y \sharp (s,c,\tau,t) \wedge (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)) = AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ y\ b\ ((y \leftrightarrow x) \cdot c\ )\ ((y \leftrightarrow x) \cdot \tau)\ ((y \leftrightarrow x) \cdot s))))$
⟨*proof*⟩

lemma *lookup-fun-member*:
assumes $Some\ (AF\text{-}fundef\ f\ ft) = lookup\text{-}fun\ \Phi\ f$
shows $AF\text{-}fundef\ f\ ft \in set\ \Phi$
⟨*proof*⟩

lemma *rig-dom-eq*:
$dom\ (G[x \longmapsto c]) = dom\ G$
⟨*proof*⟩

lemma *lookup-in-rig-eq*:
assumes $Some\ (b,c) = lookup\ \Gamma\ x$
shows $Some\ (b,c') = lookup\ (\Gamma[x \longmapsto c'])\ x$
⟨*proof*⟩

lemma *lookup-in-rig-neq*:
assumes $Some\ (b,c) = lookup\ \Gamma\ y$ **and** $x \neq y$
shows $Some\ (b,c) = lookup\ (\Gamma[x \longmapsto c'])\ y$

⟨*proof*⟩

lemma *lookup-in-rig*:
assumes *Some* (*b*,*c*) = *lookup* Γ *y*
shows ∃ *c''*. *Some* (*b*,*c''*) = *lookup* (Γ[*x*⟼*c'*]) *y*
⟨*proof*⟩

lemma *lookup-inside*[*simp*]:
assumes *x* ∉ *fst* ' *toSet* Γ'
shows *Some* (*b1*,*c1*) = *lookup* (Γ'@(*x*,*b1*,*c1*)#$_\Gamma$Γ) *x*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *lookup-x*:
assumes *Some* (*b*,*c*) = *lookup* *G* *x*
shows *x* ∈ *fst* ' *toSet* *G*
⟨*proof*⟩

lemma *GCons-eq-appendI*:
fixes *xs1*::Γ
shows [| *x* #$_\Gamma$ *xs1* = *ys*; *xs* = *xs1* @ *zs* |] ==> *x* #$_\Gamma$ *xs* = *ys* @ *zs*
⟨*proof*⟩

lemma *split-G*: *x* : *toSet* *xs* ⟹ ∃ *ys* *zs*. *xs* = *ys* @ *x* #$_\Gamma$ *zs*
⟨*proof*⟩

lemma *lookup-not-empty*:
assumes *Some* τ = *lookup* *G* *x*
shows *G* ≠ *GNil*
⟨*proof*⟩

lemma *lookup-in-g*:
assumes *Some* (*b*,*c*) = *lookup* Γ *x*
shows (*x*,*b*,*c*) ∈ *toSet* Γ
⟨*proof*⟩

lemma *lookup-split*:
fixes Γ::Γ
assumes *Some* (*b*,*c*) = *lookup* Γ *x*

shows $\exists$ *G G′* . Γ = *G′*@(*x*,*b*,*c*)$\#_\Gamma$*G*
⟨*proof*⟩

lemma *toSet-splitU*[*simp*]:
(*x′*,*b′*,*c′*) ∈ *toSet* (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) ⟷ (*x′*,*b′*,*c′*) ∈ (*toSet* Γ′ ∪ {(*x*, *b*, *c*)} ∪ *toSet* Γ)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *supp-list-member*:
fixes *x*::*′a*::*fs* **and** *l*::*′a list*
assumes *x* ∈ *set l*
shows *supp x* ⊆ *supp l*
⟨*proof*⟩

lemma *GNil-append*:
assumes *GNil* = *G1*@*G2*
shows *G1* = *GNil* ∧ *G2* = *GNil*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*⟩

lemma *fresh-d-fst-d*:
assumes *atom u* ♯ *δ*
shows *u* ∉ *fst* ‘ *set δ*
⟨*proof*⟩

lemma *bv-not-in-bset-supp*:
fixes *bv*::*bv*
assumes *bv* |∉| *B*
shows *atom bv* ∉ *supp B*
⟨*proof*⟩

lemma *u-fresh-d*:
assumes *atom u* ♯ *D*

shows *u* ∉ *fst* ' *setD D*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *fun-arg-unique-aux*:
assumes *AF-fun-typ x1 b1 c1 τ1' s1'* = *AF-fun-typ x2 b2 c2 τ2' s2'*
shows ⦃ *x1* : *b1* | *c1* ⦄ = ⦃ *x2* : *b2* | *c2*⦄
⟨*proof*⟩

lemma *fresh-x-neq*:
fixes *x*::*x* **and** *y*::*x*
shows *atom x* ♯ *y* = (*x* ≠ *y*)
⟨*proof*⟩

lemma *obtain-fresh-z3*:
fixes *tm*::'*b*::*fs*
obtains *z*::*x* **where** ⦃ *x* : *b* | *c* ⦄ = ⦃ *z* : *b* | $c[x{::=}V\text{-}var\ z]_{cv}$ ⦄ ∧ *atom z* ♯ *tm* ∧ *atom z* ♯ (*x*,*c*)
⟨*proof*⟩

lemma *u-fresh-v*:
fixes *u*::*u* **and** *t*::*v*

shows *atom u* ♯ *t*
⟨*proof*⟩

lemma *u-fresh-ce*:
fixes *u*::*u* **and** *t*::*ce*
shows *atom u* ♯ *t*
⟨*proof*⟩

lemma *u-fresh-c*:
fixes *u*::*u* **and** *t*::*c*
shows *atom u* ♯ *t*
⟨*proof*⟩

lemma *u-fresh-g*:
fixes *u*::*u* **and** *t*::Γ
shows *atom u* ♯ *t*
⟨*proof*⟩

lemma *u-fresh-t*:
fixes *u*::*u* **and** *t*::*τ*
shows *atom u* ♯ *t*
⟨*proof*⟩

lemma *b-of-c-of-eq*:
assumes *atom z* ♯ *τ*
shows ⦃ *z* : *b-of τ* | *c-of τ z* ⦄ = *τ*
⟨*proof*⟩

lemma *fresh-d-not-in*:
assumes *atom u2* ♯ Δ′
shows *u2* ∉ *fst* ‘ *setD* Δ′
⟨*proof*⟩

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*⟩

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 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (V\text{-}consp\ tyid\ dc\ b\ v) : b'' \Longrightarrow (\exists\ bv\ dclist\ x\ b'\ c.\ b'' = B\text{-}app\ tyid\ b\ \wedge$
  $AF\text{-}typedef\text{-}poly\ tyid\ bv\ dclist \in set\ \Theta\ \wedge$
  $(dc, \{\!\!\{\ x : b' \mid c\ \}\!\!\}) \in set\ dclist\ \wedge$
  $\Theta; \mathcal{B} \vdash_{wf} b\ \wedge\ atom\ bv\ \sharp\ (\Theta, \mathcal{B}, \Gamma, b, v)\ \wedge\ \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv{::=}b]_{bb}\ \wedge\ atom\ bv\ \sharp\ tm)$
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*⟩

8.3 Context Extension

definition *wfExt* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Gamma \Rightarrow bool$ (‹ - ; - $\vdash_{wf}$ - < - › *[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*; $\mathcal{B}$ $\vdash_{wf}$ (*z*,*b*,*c*) $\#_\Gamma$Γ
  shows *P*; $\mathcal{B}$ $\vdash_{wf}$ Γ ∧ *atom z* ♯ Γ ∧ *wfB P* $\mathcal{B}$ *b*
  ⟨*proof*⟩

lemma *wfG-cons2*[*ms-wb*]:
  fixes Γ::Γ
  assumes *P*; $\mathcal{B}$ $\vdash_{wf}$ *zbc* $\#_\Gamma$Γ
  shows *P*; $\mathcal{B}$ $\vdash_{wf}$ Γ
⟨*proof*⟩

lemma *wf-g-unique*:
  fixes Γ::Γ
  assumes Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** (*x*,*b*,*c*) ∈ *toSet* Γ **and** (*x*,*b'*,*c'*) ∈ *toSet* Γ
  shows *b*=*b'* ∧ *c*=*c'*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *lookup-iff*:
  fixes Θ::Θ **and** Γ::Γ
  assumes Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ
  shows *Some* (*b*,*c*) = *lookup* Γ *x* ⟷ (*x*,*b*,*c*) ∈ *toSet* Γ ∧ (∀ *b' c'*. (*x*,*b'*,*c'*) ∈ *toSet* Γ ⟶ *b'*=*b* ∧ *c'*=*c*)
  ⟨*proof*⟩

lemma *wfG-lookup-wf*:
  fixes Θ::Θ **and** Γ::Γ **and** *b*::*b* **and** $\mathcal{B}$::$\mathcal{B}$
  assumes Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** *Some* (*b*,*c*) = *lookup* Γ *x*
  shows Θ; $\mathcal{B}$ $\vdash_{wf}$ *b*
⟨*proof*⟩

lemma *wfG-unique*:
  fixes Γ::Γ
  assumes *wfG B* Θ ((*x*, *b*, *c*) $\#_\Gamma$ Γ) **and** (*x1*,*b1*,*c1*) ∈ *toSet* ((*x*, *b*, *c*) $\#_\Gamma$ Γ) **and** *x1*=*x*
  shows *b1* = *b* ∧ *c1* = *c*
⟨*proof*⟩

lemma *wfG-unique-full*:

fixes Γ::Γ
assumes *wfG* Θ $\mathcal{B}$ (Γ'@(x, b, c) $\#_\Gamma$ Γ) **and** ($x1$,$b1$,$c1$) $\in$ *toSet* (Γ'@(x, b, c) $\#_\Gamma$ Γ) **and** $x1=x$
shows $b1 = b \wedge c1 = c$
⟨*proof*⟩

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*$*\tau$) *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 *wfV-wf*: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
wfC-wf: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
wfG-wf :$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \vdash_{wf} \Theta$ **and**
wfT-wf: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta \wedge \Theta; \mathcal{B} \vdash_{wf}$ *b-of* τ **and**
wfTs-wf:$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
$\vdash_{wf} \Theta \Longrightarrow$ *True* **and**
wfB-wf: $\Theta; \mathcal{B} \vdash_{wf} b \Longrightarrow \vdash_{wf} \Theta$ **and**
wfCE-wf: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
wfTD-wf: $\Theta \vdash_{wf} td \Longrightarrow \vdash_{wf} \Theta$
⟨*proof*⟩

lemma *wfX-wfY2*:
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
wfE-wf: $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
wfS-wf: $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta ; tid ; dclist \vdash_{wf} css : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
wfPhi-wf: $\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow \vdash_{wf} \Theta$ **and**
wfD-wf: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
wfFTQ-wf: $\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$ **and**
wfFT-wf: $\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$
⟨*proof*⟩

lemmas *wfX-wfY*=*wfX-wfY1* *wfX-wfY2*

lemma *setD-ConsD*:
$ut \in setD\ (ut' \#_\Delta D) = (ut = ut' \vee ut \in setD\ D)$
⟨*proof*⟩

lemma *wfD-wfT*:
fixes Δ::Δ **and** τ::τ
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$

shows $\forall (u,\tau) \in$ *setD* Δ. Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$
⟨*proof*⟩

lemma *subst-b-lookup-d*:
assumes $u \notin$ *fst* ' *setD* Δ
shows $u \notin$ *fst* ' *setD* $\Delta[bv::=b]_{\Delta b}$
⟨*proof*⟩

lemma *wfG-cons-splitI*:
fixes $\Phi::\Phi$ **and** $\Gamma::\Gamma$
assumes Θ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom* $x \sharp \Gamma$ **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-true*$)$ $\#_\Gamma\Gamma \vdash_{wf} c$
shows Θ; $\mathcal{B} \vdash_{wf} ((x,b,c)$ $\#_\Gamma\Gamma)$
⟨*proof*⟩

lemma *wfG-consI*:
fixes $\Phi::\Phi$ **and** $\Gamma::\Gamma$ **and** $c::c$
assumes Θ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom* $x \sharp \Gamma$ **and** *wfB* Θ $\mathcal{B}$ b **and**
Θ ; $\mathcal{B}$; $(x,b,$*C-true*$)$ $\#_\Gamma\Gamma \vdash_{wf} c$
shows Θ ; $\mathcal{B} \vdash_{wf} ((x,b,c)$ $\#_\Gamma\Gamma)$
⟨*proof*⟩

lemma *wfG-elim2*:
fixes $c::c$
assumes *wfG* P $\mathcal{B}$ $((x,b,c)$ $\#_\Gamma\Gamma)$
shows P; $\mathcal{B}$; $(x, b,$ *TRUE*$)$ $\#_\Gamma$ Γ $\vdash_{wf} c \wedge$ *wfB* P $\mathcal{B}$ b
⟨*proof*⟩

lemma *wfG-cons-wfC*:
fixes $\Gamma::\Gamma$ **and** $c::c$
assumes Θ ; B $\vdash_{wf} (x, b, c)$ $\#_\Gamma$ Γ
shows Θ ; B ; $((x, b,$ *TRUE*$)$ $\#_\Gamma$ $\Gamma) \vdash_{wf} c$
⟨*proof*⟩

lemma *wfG-wfB*:
assumes *wfG* P $\mathcal{B}$ Γ **and** $b \in$ *fst*'*snd*'*toSet* Γ
shows *wfB* P $\mathcal{B}$ b
⟨*proof*⟩

lemma *wfG-cons-TRUE*:
fixes $\Gamma::\Gamma$ **and** $b::b$
assumes P; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom* $z \sharp \Gamma$ **and** P; $\mathcal{B} \vdash_{wf} b$
shows P ; $\mathcal{B} \vdash_{wf} (z, b,$ *TRUE*$)$ $\#_\Gamma$ Γ
⟨*proof*⟩

lemma *wfG-cons-TRUE2*:
assumes P; $\mathcal{B} \vdash_{wf} (z,b,c)$ $\#_\Gamma\Gamma$ **and** *atom* $z \sharp \Gamma$
shows P; $\mathcal{B} \vdash_{wf} (z, b,$ *TRUE*$)$ $\#_\Gamma$ Γ
⟨*proof*⟩

lemma *wfG-suffix*:

fixes Γ::Γ
assumes *wfG P* $\mathcal{B}$ (Γ′@Γ)
shows *wfG P* $\mathcal{B}$ Γ
⟨*proof*⟩

lemma *wfV-wfCE*:
fixes *v*::*v*
assumes Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b*
shows Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-val v* : *b*
⟨*proof*⟩

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; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* ⟹ *supp v* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **and**
wfC-supp: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* ⟹ *supp c* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **and**
wfG-supp: Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ⟹ *atom-dom* Γ ⊆ *supp* Γ **and**
wfT-supp: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ ⟹ *supp* τ ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **and**
wfTs-supp: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* ⟹ *supp ts* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **and**
wfTh-supp: $\vdash_{wf}$ Θ ⟹ *supp* Θ = {} **and**
wfB-supp: Θ; $\mathcal{B}$ $\vdash_{wf}$ *b* ⟹ *supp b* ⊆ *supp* $\mathcal{B}$ **and**
wfCE-supp: Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* ⟹ *supp ce* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **and**
wfTD-supp: Θ $\vdash_{wf}$ *td* ⟹ *supp td* ⊆ {}
⟨*proof*⟩

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*⟩

lemmas *wf-supp* = *wf-supp1 wf-supp2*

lemma *wfV-supp-nil*:
fixes *v*::*v*

assumes P ; $\{||\}$; *GNil* $\vdash_{wf}$ v : b
shows *supp* $v = \{\}$
⟨*proof*⟩

lemma *wfT-TRUE-aux*:
assumes *wfG* P $\mathcal{B}$ Γ **and** *atom* z ♯ $(P, \mathcal{B}, \Gamma)$ **and** *wfB* P $\mathcal{B}$ b
shows *wfT* P $\mathcal{B}$ Γ ({| z : b | *TRUE* |})
⟨*proof*⟩

lemma *wfT-TRUE*:
assumes *wfG* P $\mathcal{B}$ Γ **and** *wfB* P $\mathcal{B}$ b
shows *wfT* P $\mathcal{B}$ Γ ({| z : b | *TRUE* |})
⟨*proof*⟩

lemma *phi-flip-eq*:
assumes *wfPhi* T P
shows $(x \leftrightarrow xa) \cdot P = P$
⟨*proof*⟩

lemma *wfC-supp-cons*:
fixes c'::c **and** G::Γ
assumes P; $\mathcal{B}$; $(x', b', $ *TRUE*$)$ $\#_\Gamma G$ $\vdash_{wf}$ c'
shows *supp* $c' \subseteq$ *atom-dom* $G \cup$ *supp* $x' \cup$ *supp* $\mathcal{B}$ **and** *supp* $c' \subseteq$ *supp* $G \cup$ *supp* $x' \cup$ *supp* $\mathcal{B}$
⟨*proof*⟩

lemma *wfG-dom-supp*:
fixes x::x
assumes *wfG* P $\mathcal{B}$ G
shows *atom* $x \in$ *atom-dom* $G \longleftrightarrow$ *atom* $x \in$ *supp* G
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

lemma *theta-flip-eq2*:
assumes $\vdash_{wf}$ Θ
shows $(z \leftrightarrow za\ ) \cdot \Theta = \Theta$
⟨*proof*⟩

lemma *theta-flip-eq*:
assumes *wfTh* Θ
shows $(x \leftrightarrow xa) \cdot \Theta = \Theta$
⟨*proof*⟩

lemma *wfT-wfC*:
fixes *c*::*c*
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄ **and** *atom z* ♯ Γ
shows Θ; $\mathcal{B}$; (*z*,*b*,*TRUE*) $\#_\Gamma\Gamma \vdash_{wf}$ *c*
⟨*proof*⟩

lemma *u-not-in-dom-g*:
fixes *u*::*u*
shows *atom u* ∉ *atom-dom G*
⟨*proof*⟩

lemma *bv-not-in-dom-g*:
fixes *bv*::*bv*
shows *atom bv* ∉ *atom-dom G*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *wfT-supp-c*:
fixes $\mathcal{B}$::$\mathcal{B}$ **and** *z*::*x*
assumes *wfT P* $\mathcal{B}$ Γ (⦃ *z* : *b* | *c* ⦄)
shows *supp c* − { *atom z* } ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$
⟨*proof*⟩

lemma *wfG-wfC*[*ms-wb*]:
assumes *wfG P* $\mathcal{B}$ ((*x*,*b*,*c*) $\#_\Gamma\Gamma$)
shows *wfC P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) $\#_\Gamma\Gamma$) *c*
⟨*proof*⟩

lemma *wfT-wf-cons*:
assumes *wfT P* $\mathcal{B}$ Γ ⦃ *z* : *b* | *c* ⦄ **and** *atom z* ♯ Γ
shows *wfG P* $\mathcal{B}$ ((*z*,*b*,*c*) $\#_\Gamma\Gamma$)
⟨*proof*⟩

lemma *wfV-b-fresh*:
fixes *b*::*b* **and** *v*::*v* **and** *bv*::*bv*
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v*: *b* **and** *bv* |∉| $\mathcal{B}$
shows *atom bv* ♯ *v*
⟨*proof*⟩

lemma *wfCE-b-fresh*:
  fixes *b*::*b* **and** *ce*::*ce* **and** *bv*::*bv*
  assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *ce*: *b* **and** *bv* $|\notin|$ $\mathcal{B}$
  shows *atom bv* $\sharp$ *ce*
⟨*proof*⟩

8.7 Freshness

lemma *wfG-fresh-x*:
  fixes Γ::Γ **and** *z*::*x*
  assumes Θ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom z* $\sharp$ Γ
  shows *atom z* $\sharp$ (Θ,$\mathcal{B}$, Γ)
⟨*proof*⟩

lemma *wfG-wfT*:
  assumes *wfG P* $\mathcal{B}$ ((*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) $\#_\Gamma$ *G*) **and** *atom x* $\sharp$ *c*
  shows *P*; $\mathcal{B}$; *G* $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄
⟨*proof*⟩

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* $\sharp$ (*c*,Γ)
  shows *wfT* Θ $\mathcal{B}$ Γ ⦃ *z* : *b* | *c* ⦄
⟨*proof*⟩

lemma *wfT-fresh-c*:
  fixes *x*::*x*
  assumes *wfT P* $\mathcal{B}$ Γ ⦃ *z* : *b* | *c* ⦄ **and** *atom x* $\sharp$ Γ **and** $x \neq z$
  shows *atom x* $\sharp$ *c*
⟨*proof*⟩

lemma *wfG-x-fresh* [*simp*]:
  fixes *x*::*x*
  assumes *wfG P* $\mathcal{B}$ *G*
  shows *atom x* $\notin$ *atom-dom G* $\longleftrightarrow$ *atom x* $\sharp$ *G*
  ⟨*proof*⟩

lemma *wfD-x-fresh*:
  fixes *x*::*x*
  assumes *atom x* $\sharp$ Γ **and** *wfD P B* Γ Δ
  shows *atom x* $\sharp$ Δ
⟨*proof*⟩

lemma *wfG-fresh-x2*:
  fixes Γ::Γ **and** *z*::*x* **and** Δ::Δ **and** Φ::Φ
  assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ **and** $\Theta \vdash_{wf} \Phi$ **and** *atom z* $\sharp$ Γ
  shows *atom z* $\sharp$ (Θ,Φ,$\mathcal{B}$, Γ,Δ)
  ⟨*proof*⟩

lemma *wfV-x-fresh*:
  fixes *v*::*v* **and** *b*::*b* **and** Γ::Γ **and** *x*::*x*
  assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b$ **and** *atom x* $\sharp$ Γ

shows *atom x* ♯ *v*
⟨*proof*⟩

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; ℬ; Γ ; Δ $\vdash_{wf}$ *e* : *b* **and** *atom x* ♯ Γ
shows *atom x* ♯ *e*
⟨*proof*⟩

lemma *wfT-x-fresh*:
fixes τ::τ **and** Γ::Γ **and** *x*::*x*
assumes Θ; ℬ; Γ $\vdash_{wf}$ τ **and** *atom x* ♯ Γ
shows *atom x* ♯ τ
⟨*proof*⟩

lemma *wfS-x-fresh*:
fixes *s*::*s* **and** Δ::Δ **and** *x*::*x*
assumes Θ; Φ; ℬ; Γ; Δ $\vdash_{wf}$ *s* : *b* **and** *atom x* ♯ Γ
shows *atom x* ♯ *s*
⟨*proof*⟩

lemma *wfTh-fresh*:
fixes *x*
assumes *wfTh T*
shows *atom x* ♯ *T*
⟨*proof*⟩

lemmas *wfTh-x-fresh* = *wfTh-fresh*

lemma *wfPhi-fresh*:
fixes *x*
assumes *wfPhi T P*
shows *atom x* ♯ *P*
⟨*proof*⟩

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*) $\#_\Gamma$Γ))
shows *atom x* ∉ *atom-dom* Γ'
⟨*proof*⟩

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*]$_{cv}$) $\#_\Gamma$ Γ)
⟨*proof*⟩

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*]$_{cv}$) $\#_\Gamma$ Γ) **and** *atom xa* ♯ *G* **and** Θ; ℬ $\vdash_{wf}$ *G*

shows $xa \neq x$
⟨*proof*⟩

lemma *wfG-inside-x-fresh*:
fixes $c::c$ **and** $x::x$ **and** $\Gamma::\Gamma$ **and** $G::\Gamma$ **and** $xa::x$
assumes $G=(\Gamma'@ (x, b, c[z::=V\text{-}var\ x]_{cv}) \ \ \#_\Gamma \Gamma)$ **and** $atom\ xa \sharp G$ **and** $\Theta; \mathcal{B} \vdash_{wf} G$
shows $atom\ xa \sharp x$
⟨*proof*⟩

lemma *wfT-nil-supp*:
fixes $t::\tau$
assumes $\Theta ; \{||\} ; GNil \vdash_{wf} t$
shows $supp\ t = \{\}$
⟨*proof*⟩

8.8 Misc

lemma *wfG-cons-append*:
fixes $b'::b$
assumes $\Theta; \mathcal{B} \vdash_{wf} ((x', b', c') \ \ \#_\Gamma \Gamma') @ (x, b, c) \ \ \#_\Gamma \Gamma$
shows $\Theta; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c) \ \ \#_\Gamma \Gamma) \wedge atom\ x' \sharp (\Gamma' @ (x, b, c) \ \ \#_\Gamma \Gamma) \wedge \Theta; \mathcal{B} \vdash_{wf} b' \wedge x' \neq x$
⟨*proof*⟩

lemma *flip-u-eq*:
fixes $u::u$ **and** $u'::u$ **and** $\Theta::\Theta$ **and** $\tau::\tau$
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$
shows $(u \leftrightarrow u') \cdot \tau = \tau$ **and** $(u \leftrightarrow u') \cdot \Gamma = \Gamma$ **and** $(u \leftrightarrow u') \cdot \Theta = \Theta$ **and** $(u \leftrightarrow u') \cdot \mathcal{B} = \mathcal{B}$
⟨*proof*⟩

lemma *wfT-wf-cons-flip*:
fixes $c::c$ **and** $x::x$
assumes $wfT\ P\ \mathcal{B}\ \Gamma\ \{\!|\ z : b \ | \ c\ |\!\}$ **and** $atom\ x \sharp (c,\Gamma)$
shows $wfG\ P\ \mathcal{B}\ ((x,b,c[z::=V\text{-}var\ x]_{cv}) \ \ \#_\Gamma\Gamma)$
⟨*proof*⟩

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 $y::'a::at\text{-}base$ **and** $\Gamma::\Gamma$
assumes $atom\ y \sharp \ (\Gamma' @ (x, b, c) \ \ \#_\Gamma \Gamma)$
shows $atom\ y \sharp (\Gamma'@\Gamma)$
⟨*proof*⟩

lemma *wf-restrict1*:
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** $b::b$ **and** $ftq::fun\text{-}typ\text{-}q$ **and** $ft::fun\text{-}typ$ **and** $ce::ce$ **and** $td::type\text{-}def$
and $cs::branch\text{-}s$ **and** $css::branch\text{-}list$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Gamma=\Gamma_1@((x,b',c') \ \ \#_\Gamma\Gamma_2) \implies atom\ x \sharp v \implies atom\ x \sharp \Gamma_1 \implies \Theta; \mathcal{B}; \ \Gamma_1@\Gamma_2 \vdash_{wf} v : b$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow$ *atom* $x \sharp c \Longrightarrow$ *atom* $x \sharp \Gamma_1 \Longrightarrow \Theta ; \mathcal{B} ; \Gamma_1@\Gamma_2 \vdash_{wf} c$ **and**
$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow$ *atom* $x \sharp \Gamma_1 \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma_1@\Gamma_2$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow$ *atom* $x \sharp \tau \Longrightarrow$ *atom* $x \sharp \Gamma_1 \Longrightarrow \Theta; \mathcal{B}; \Gamma_1@\Gamma_2 \vdash_{wf} \tau$ **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**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow$ *atom* $x \sharp ce \Longrightarrow$ *atom* $x \sharp \Gamma_1 \Longrightarrow \Theta; \mathcal{B}; \Gamma_1@\Gamma_2 \vdash_{wf} ce : b$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
⟨*proof*⟩

lemma *wf-restrict2*:
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** $b{::}b$ **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 $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} e : b \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow$ *atom* $x \sharp e \Longrightarrow$ *atom* $x \sharp \Gamma_1 \Longrightarrow$ *atom* $x \sharp \Delta \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma_1@\Gamma_2 ; \Delta \vdash_{wf} e : b$ **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**
$\Theta \vdash_{wf} (\Phi{::}\Phi) \Longrightarrow$ *True* **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow$ *atom* $x \sharp \Gamma_1 \Longrightarrow$ *atom* $x \sharp \Delta \Longrightarrow \Theta; \mathcal{B}; \Gamma_1@\Gamma_2 \vdash_{wf} \Delta$ **and**
$\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow$ *True* **and**
$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow$ *True*

⟨*proof*⟩

lemmas *wf-restrict=wf-restrict1 wf-restrict2*

lemma *wfT-restrict2*:
fixes $\tau{::}\tau$
assumes *wfT* $\Theta\ \mathcal{B}\ ((x, b, c) \#_\Gamma \Gamma)\ \tau$ **and** *atom* $x \sharp \tau$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$
⟨*proof*⟩

lemma *wfG-intros2*:
assumes *wfC* $P\ \mathcal{B}\ ((x,b,TRUE) \#_\Gamma \Gamma)\ c$
shows *wfG* $P\ \mathcal{B}\ ((x,b,c) \#_\Gamma \Gamma)$
⟨*proof*⟩

8.10 Type Definitions

lemma *wf-theta-weakening1*:
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** $b{::}b$ **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** $t{::}\tau$

shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} ; \Gamma \vdash_{wf} c$ **and**
$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} \vdash_{wf} \Gamma$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} ; \Gamma \vdash_{wf} \tau$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} ; \Gamma \vdash_{wf} ts$ **and**
$\vdash_{wf} P \Longrightarrow True$ **and**
$\Theta; \mathcal{B} \vdash_{wf} b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} \vdash_{wf} b$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} ; \Gamma \vdash_{wf} ce : b$ **and**
$\Theta \vdash_{wf} td \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' \vdash_{wf} td$
⟨*proof*⟩

lemma *wf-theta-weakening2*:
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** $b::b$ **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** $t::\tau$
shows
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} e : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} e : b$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} s : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s : b$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' \vdash_{wf} (\Phi::\Phi)$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$ **and**
$\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \Phi \vdash_{wf} ftq$ **and**
$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' ; \Phi ; \mathcal{B} \vdash_{wf} ft$

⟨*proof*⟩

lemmas *wf-theta-weakening* = *wf-theta-weakening1 wf-theta-weakening2*

lemma *lookup-wfTD*:
fixes *td*::*type-def*
assumes $td \in set\ \Theta$ **and** $\vdash_{wf} \Theta$
shows $\Theta \vdash_{wf} td$
⟨*proof*⟩

8.10.1 Simple

lemma *wfTh-dclist-unique*:
assumes *wfTh* Θ **and** *AF-typedef tid dclist1* $\in set\ \Theta$ **and** *AF-typedef tid dclist2* $\in set\ \Theta$
shows *dclist1* = *dclist2*
⟨*proof*⟩

lemma *wfTs-ctor-unique*:
fixes *dclist*::$(string*\tau)$ *list*
assumes $\Theta ; \{||\} ; GNil \vdash_{wf} dclist$ **and** $(c, t1) \in set\ dclist$ **and** $(c,t2) \in set\ dclist$
shows $t1 = t2$
⟨*proof*⟩

lemma *wfTD-ctor-unique*:
assumes $\Theta \vdash_{wf} (AF\text{-}typedef\ tid\ dclist)$ **and** $(c, t1) \in set\ dclist$ **and** $(c,t2) \in set\ dclist$

shows $t1 = t2$
⟨*proof*⟩

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 $t1 = t2$
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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* = {}
⟨*proof*⟩

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* ⦄)
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*⟩

lemma *wfTh-poly-supp-b*:
assumes *AF-typedef-poly tid bv dclist* $\in$ *set* Θ **and** $(dc, \{\!\mid z : b \mid c \mid\!\})$ $\in$ *set dclist* **and** $\vdash_{wf} \Theta$
shows *supp b* $\subseteq$ $\{$*atom bv*$\}$
⟨*proof*⟩

lemma *subst-g-inside*:
fixes x::x **and** c::c **and** Γ::Γ **and** Γ'::Γ
assumes *wfG P* $\mathcal{B}$ $(\Gamma' @ (x, b, c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$
shows $(\Gamma' @ (x, b, c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)[x::=v]_{\Gamma v} = (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$
⟨*proof*⟩

lemma *wfTh-td-eq*:
assumes *td1* $\in$ *set* (*td2* # *P*) **and** *wfTh* (*td2* # *P*) **and** *name-of-type td1* = *name-of-type td2*
shows *td1* = *td2*
⟨*proof*⟩

lemma *wfTh-td-unique*:
assumes *td1* $\in$ *set P* **and** *td2* $\in$ *set P* **and** *wfTh P* **and** *name-of-type td1* = *name-of-type td2*
shows *td1* = *td2*
⟨*proof*⟩

lemma *wfTs-distinct*:
fixes *dclist*::(*string* $*$ τ) *list*
assumes Θ ; *B* ; *GNil* $\vdash_{wf}$ *dclist*
shows *distinct* (*map fst dclist*)
⟨*proof*⟩

lemma *wfTh-dclist-distinct*:
assumes *AF-typedef s dclist* $\in$ *set P* **and** *wfTh P*
shows *distinct* (*map fst dclist*)
⟨*proof*⟩

lemma *wfTh-dc-t-unique2*:
assumes *AF-typedef s dclist'* $\in$ *set P* **and** (*dc*,*tc'*) $\in$ *set dclist'* **and** *AF-typedef s dclist* $\in$ *set P* **and** *wfTh P* **and**
(*dc*, *tc*) $\in$ *set dclist*
shows *tc*= *tc'*
⟨*proof*⟩

lemma *wfTh-dc-t-unique*:
assumes *AF-typedef s dclist'* $\in$ *set P* **and** $(dc, \{\!\mid x' : b' \mid c' \mid\!\})$ $\in$ *set dclist'* **and** *AF-typedef s dclist* $\in$ *set P* **and** *wfTh P* **and**
$(dc, \{\!\mid x : b \mid c \mid\!\})$ $\in$ *set dclist*
shows $\{\!\mid x' : b' \mid c' \mid\!\} = \{\!\mid x : b \mid c \mid\!\}$
⟨*proof*⟩

lemma *wfTs-wfT*:
fixes *dclist*::(*string* $*\tau$) *list* **and** *t*::τ
assumes Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *dclist* **and** (*dc*,*t*) $\in$ *set dclist*
shows Θ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *t*
⟨*proof*⟩

lemma *wfTh-wfT*:
fixes $t::\tau$
assumes *wfTh P* **and** *AF-typedef tid dclist* $\in$ *set P* **and** $(dc,t) \in$ *set dclist*
shows P ; $\{||\}$; *GNil* $\vdash_{wf}$ t
⟨*proof*⟩

lemma *td-lookup-eq-iff*:
fixes *dc* :: *string* **and** *bva1*::*bv* **and** *bva2*::*bv*
assumes [[*atom bva1*]]*lst. dclist1* = [[*atom bva2*]]*lst. dclist2* **and** $(dc, \{\!|\ x : b \ \mid c\ |\!\}) \in$ *set dclist1*
shows $\exists$ *x2 b2 c2*. $(dc, \{\!|\ x2 : b2 \ \mid c2\ |\!\}) \in$ *set dclist2*
⟨*proof*⟩

lemma *lst-t-b-eq-iff*:
fixes *bva1*::*bv* **and** *bva2*::*bv*
assumes [[*atom bva1*]]*lst.* $\{\!|\ x1 : b1 \ \mid c1\ |\!\}$ = [[*atom bva2*]]*lst.* $\{\!|\ x2 : b2 \ \mid c2\ |\!\}$
shows [[*atom bva1*]]*lst. b1* = [[*atom bva2*]]*lst.b2*
⟨*proof*⟩

lemma *wfTh-typedef-poly-b-eq-iff*:
assumes *AF-typedef-poly tyid bva1 dclist1* $\in$ *set P* **and** $(dc, \{\!|\ x1 : b1 \ \mid c1\ |\!\}) \in$ *set dclist1*
and *AF-typedef-poly tyid bva2 dclist2* $\in$ *set P* **and** $(dc, \{\!|\ x2 : b2 \ \mid c2\ |\!\}) \in$ *set dclist2* **and** $\vdash_{wf}$ P
shows $b1[bva1::=b]_{bb} = b2[bva2::=b]_{bb}$
⟨*proof*⟩

8.11 Equivariance Lemmas

lemma *x-not-in-u-set*[*simp*]:
fixes *x*::*x* **and** *us*::*u fset*
shows *atom x* $\notin$ *supp us*
⟨*proof*⟩

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* $\sharp$ Γ **and**
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ **and**
Θ ; Φ ; $\mathcal{B}$; $(x1, b, c1)$ $\#_{\Gamma}$ Γ ; $\Delta \vdash_{wf}$ *s1* : *b-of t1*
shows Θ ; Φ ; $\mathcal{B}$; $(x2, b, c2)$ $\#_{\Gamma}$ Γ ; Δ $\vdash_{wf}$ *s2* : *b-of t2*
⟨*proof*⟩

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'
⟨*proof*⟩

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

⟨*proof*⟩

lemma *lookup-weakening*:
fixes Θ::Θ **and** Γ::Γ **and** Γ′::Γ
assumes *Some* (*b*,*c*) = *lookup* Γ *x* **and** *toSet* Γ ⊆ *toSet* Γ′ **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ
shows *Some* (*b*,*c*) = *lookup* Γ′ *x*
⟨*proof*⟩

lemma *wfPhi-lookup-fun-unique*:
fixes Φ::Φ
assumes Θ $\vdash_{wf}$ Φ **and** *AF-fundef f fd* ∈ *set* Φ
shows *Some* (*AF-fundef f fd*) = *lookup-fun* Φ *f*
⟨*proof*⟩

lemma *lookup-fun-weakening*:
fixes Φ′::Φ
assumes *Some fd* = *lookup-fun* Φ *f* **and** *set* Φ ⊆ *set* Φ′ **and** Θ $\vdash_{wf}$ Φ′
shows *Some fd* = *lookup-fun* Φ′ *f*
⟨*proof*⟩

lemma *fundef-poly-fresh-bv*:
assumes *atom bv2* ♯ (*bv1*,*b1*,*c1*,*τ1*,*s1*)
shows ∗ : (*AF-fun-typ-some bv2* (*AF-fun-typ x1* ((*bv1*↔*bv2*) · *b1*) ((*bv1*↔*bv2*) ·*c1*) ((*bv1*↔*bv2*) · *τ1*) ((*bv1*↔*bv2*) · *s1*)) = (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1 s1*)))
(**is** (*AF-fun-typ-some ?bv ?fun-typ* = *AF-fun-typ-some ?bva ?fun-typa*))
⟨*proof*⟩

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 Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* ⟹ $\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *v* : *b* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* ⟹$\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *c* **and**
Θ; $\mathcal{B}$ $\vdash_{wf}$ Γ ⟹$\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ Θ; $\mathcal{B}'$ $\vdash_{wf}$ Γ **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *τ* ⟹ $\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *τ* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* ⟹ $\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *ts* **and**
$\vdash_{wf}$ *P* ⟹ *True* **and**
wfB Θ $\mathcal{B}$ *b* ⟹ $\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ *wfB* Θ $\mathcal{B}'$ *b* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* ⟹ $\mathcal{B}$ |⊆| $\mathcal{B}'$ ⟹ Θ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *ce* : *b* **and**
Θ $\vdash_{wf}$ *td* ⟹ *True*
⟨*proof*⟩

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*∗*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
$\Theta; \Phi; \mathcal{B}; \Gamma \ ; \Delta \vdash_{wf} e : b \Longrightarrow \mathcal{B} \mid\subseteq\mid \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \ \Gamma ; \Delta \vdash_{wf} e : b$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} s : b \Longrightarrow \mathcal{B} \mid\subseteq\mid \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \ \Gamma ; \Delta \vdash_{wf} s : b$ **and**
$\Theta ; \Phi ; \mathcal{B} \ ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \Longrightarrow \mathcal{B} \mid\subseteq\mid \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b$ **and**
$\Theta ; \Phi ; \mathcal{B} \ ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \Longrightarrow \mathcal{B} \mid\subseteq\mid \mathcal{B}' \Longrightarrow \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow True$ **and**
$\Theta; \mathcal{B}; \Gamma \ \vdash_{wf} \Delta \Longrightarrow \mathcal{B} \mid\subseteq\mid \mathcal{B}' \Longrightarrow \Theta; \mathcal{B}' ; \Gamma \vdash_{wf} \ \Delta$ **and**
$\Theta ; \Phi \ \ \vdash_{wf} ftq \Longrightarrow True$ **and**
$\Theta ; \Phi \ \ ; \mathcal{B} \vdash_{wf} ft \Longrightarrow \mathcal{B} \mid\subseteq\mid \mathcal{B}' \Longrightarrow \Theta ; \Phi \ \ ; \mathcal{B}' \vdash_{wf} ft$
⟨*proof*⟩

lemmas *wb-b-weakening* = *wb-b-weakening1 wb-b-weakening2*

lemma *wfG-b-weakening*:
fixes $\Gamma::\Gamma$
assumes $\mathcal{B} \mid\subseteq\mid \mathcal{B}'$ **and** $\Theta; \mathcal{B} \vdash_{wf} \Gamma$
shows $\Theta; \mathcal{B}' \ \vdash_{wf} \Gamma$
⟨*proof*⟩

lemma *wfT-b-weakening*:
fixes $\Gamma::\Gamma$ **and** $\Theta::\Theta$ **and** $\tau::\tau$
assumes $\mathcal{B} \mid\subseteq\mid \mathcal{B}'$ **and** $\Theta; \mathcal{B}; \Gamma \ \vdash_{wf} \tau$
shows $\Theta; \mathcal{B}' ; \Gamma \ \vdash_{wf} \tau$
⟨*proof*⟩

lemma *wfB-subst-wfB*:
fixes $\tau::\tau$ **and** $b'::b$ **and** $b::b$
assumes $\Theta ; \{|bv|\} \ \vdash_{wf} b$ **and** $\Theta; \mathcal{B} \ \vdash_{wf} b'$
shows $\Theta; \mathcal{B} \ \vdash_{wf} b[bv::=b']_{bb}$
⟨*proof*⟩

lemma *wfT-subst-wfB*:
fixes $\tau::\tau$ **and** $b'::b$
assumes $\Theta ; \{|bv|\} ; (x,\ b,\ c) \ \ \#_{\Gamma} \ GNil \ \ \vdash_{wf} \tau$ **and** $\Theta; \mathcal{B} \ \vdash_{wf} b'$
shows $\Theta; \mathcal{B} \ \vdash_{wf} (b\text{-}of\ \tau)[bv::=b']_{bb}$
⟨*proof*⟩

lemma *wfG-cons-unique*:
assumes $(x1,b1,c1) \in toSet\ (((x,b,c)\ \#_{\Gamma}\Gamma))$ **and** $wfG\ \Theta\ \mathcal{B}\ (((x,b,c)\ \#_{\Gamma}\Gamma))$ **and** $x = x1$
shows $b1 = b \wedge c1 = c$
⟨*proof*⟩

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 $b1 = b \wedge c1 = c$
⟨*proof*⟩

8.13 Function Definitions

lemma *wb-phi-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* **and** Φ::Φ
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*⟩

lemma *wfT-fun-return-t*:
fixes $\tau a'$::τ **and** τ'::τ
assumes $\Theta; \mathcal{B}; (xa, b, ca) \#_\Gamma GNil \vdash_{wf} \tau a'$ **and** $(AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s') = (AF\text{-}fun\text{-}typ\ xa\ b\ ca\ \tau a'\ sa')$
shows $\Theta; \mathcal{B}; (x, b, c) \#_\Gamma GNil \vdash_{wf} \tau'$
⟨*proof*⟩

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 $\Theta ; \Phi ; B \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$
shows $\Theta ; B ; (x,b,c) \#_\Gamma GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge supp\ s \subseteq \{ atom\ x \} \cup supp\ B$
⟨*proof*⟩

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 $\Theta ; \Phi \vdash_{wf} (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s))$
shows $\Theta ; \{||\} ; (x,b,c) \#_\Gamma GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge supp\ s \subseteq \{ atom\ x \}$
⟨*proof*⟩

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 $\Theta ; \Phi \vdash_{wf} (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s))$
shows $\Theta ; \{|bv|\} ; (x,b,c) \#_\Gamma GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge \Theta ; \Phi ; \{|bv|\} \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$
⟨*proof*⟩

lemma *wfFT-poly-wfT*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes $\Theta ; \Phi \vdash_{wf} (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s))$
shows $\Theta ; \{| bv |\} ; (x,b,c) \#_\Gamma GNil \vdash_{wf} \tau$
⟨*proof*⟩

lemma *b-of-supp*:
$supp\ (b\text{-}of\ t) \subseteq supp\ t$

⟨*proof*⟩

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*)) $\in$ *set* Φ **and** $\Theta \vdash_{wf} \Phi$ **and** *set* $\Phi \subseteq$ *set* Φ' **and** $\Theta \vdash_{wf} \Phi'$
shows Θ ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge$ *supp s* $\subseteq$ { *atom x* }
⟨*proof*⟩

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** $\Theta \vdash_{wf}$ Φ
shows Θ ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau$
⟨*proof*⟩

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** $\Theta \vdash_{wf}$ Φ
shows *supp b* = {}
⟨*proof*⟩

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** $\Theta \vdash_{wf}$ Φ
shows *supp* $\tau \subseteq$ { *atom x* }
⟨*proof*⟩

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** $\Theta \vdash_{wf}$ Φ
shows *supp c* $\subseteq$ { *atom x* }
⟨*proof*⟩

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** $\Theta \vdash_{wf}$ Φ
shows *supp s* $\subseteq$ {*atom x*}
⟨*proof*⟩

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*)
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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* }
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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 $x1$ $b1$ $c1$ $\tau 1'$ $s1'$) = (AF-fun-typ $x2$ $b2$ $c2$ $\tau 2'$ $s2'$)
shows $s1'[x1::=v]_{sv} = s2'[x2::=v]_{sv}$
⟨*proof*⟩

lemma *fun-ret-unique*:
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1* $\tau 1'$ $s1'$))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x2 b2 c2* $\tau 2'$ $s2'$))) = *lookup-fun* Φ *f*
shows $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$
⟨*proof*⟩

lemma *fun-poly-arg-unique*:
fixes $bv1$::bv **and** $bv2$::bv **and** b::b **and** $\tau 1$::τ **and** $\tau 2$::τ
assumes [[*atom bv1*]]*lst*. (*AF-fun-typ x1 b1 c1* $\tau 1$ *s1*) = [[*atom bv2*]]*lst*. (*AF-fun-typ x2 b2 c2* $\tau 2$ *s2*) (**is** [[*atom ?x*]]*lst*. *?a* = [[*atom ?y*]]*lst*. *?b*)
shows ⦃ $x1$: $b1[bv1::=b]_{bb}$ | $c1[bv1::=b]_{cb}$ ⦄ = ⦃ $x2$: $b2[bv2::=b]_{bb}$ | $c2[bv2::=b]_{cb}$ ⦄
⟨*proof*⟩

lemma *fun-poly-ret-unique*:
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ $s1'$))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ $s2'$))) = *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*⟩

lemma *fun-poly-body-unique*:
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ $s1'$))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ $s2'$))) = *lookup-fun* Φ *f*
shows $s1'[bv1::=b]_{sb}[x1::=v]_{sv} = s2'[bv2::=b]_{sb}[x2::=v]_{sv}$
⟨*proof*⟩

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 | c' ⦄ = ⦃ x : b | c ⦄ ∧ $s'[x'::=v]_{sv} = s[x::=v]_{sv}$ ∧ $\tau'[x'::=v]_{\tau v} = \tau[x::=v]_{\tau v}$
⟨*proof*⟩

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** $\mathcal{B}$::$\mathcal{B}$ **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 *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*⟩

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*⟩

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*$*\tau$) *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 *wfV-weakening*: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \mathcal{B}; \Gamma' \vdash_{wf} v : b$ **and**
wfC-weakening: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c$ **and**
$\Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow True$ **and**
wfT-weakening: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \tau$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow True$ **and**
$\vdash_{wf} P \Longrightarrow True$ **and**
wfB-weakening: $wfB\ \Theta\ \mathcal{B}\ b \Longrightarrow \mathcal{B} |\subseteq| \mathcal{B}' \Longrightarrow wfB\ \Theta\ \mathcal{B}\ b$ **and**
wfCE-weakening: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \mathcal{B}; \Gamma' \vdash_{wf} ce : b$ **and**
$\Theta \vdash_{wf} td \Longrightarrow True$
⟨*proof*⟩

lemma *wf-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
wfE-weakening: $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} e : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma' ; \Delta \vdash_{wf} e : b$ **and**
wfS-weakening: $\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta \vdash_{wf} s : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma' ; \Delta \vdash_{wf} s : b$ **and**
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma' ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b$ **and**
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow toSet\ \Gamma \subseteq toSet\ \Gamma' \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma' ; \Delta ; tid ; dclist \vdash_{wf} css : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow True$ **and**

wfD-weakning: Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta \Longrightarrow \Theta$; $\mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow$ *toSet* $\Gamma \subseteq$ *toSet* $\Gamma' \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$
and
Θ ; $\Phi \vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\Longrightarrow$ *True*
⟨*proof*⟩

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; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b$ **and** *atom y* $\sharp$ Γ **and** Θ; $\mathcal{B}$; $((y,b',TRUE)$ $\#_\Gamma$ $\Gamma) \vdash_{wf} c$
shows Θ; $\mathcal{B}$; (y,b',c) $\#_\Gamma\Gamma \vdash_{wf} v : b$
⟨*proof*⟩

lemma *wfG-cons-weakening*:
fixes Γ'::Γ
assumes Θ; $\mathcal{B} \vdash_{wf} ((x, b, c)$ $\#_\Gamma$ $\Gamma)$ **and** Θ; $\mathcal{B} \vdash_{wf} \Gamma'$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** *atom x* $\sharp$ Γ'
shows Θ; $\mathcal{B} \vdash_{wf} ((x, b, c)$ $\#_\Gamma$ $\Gamma')$
⟨*proof*⟩

lemma *wfT-weakening-aux*:
fixes Γ::Γ **and** Γ'::Γ **and** *c*::*c*
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!\!\{ z : b \mid c \}\!\!\}$ **and** Θ; $\mathcal{B} \vdash_{wf} \Gamma'$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** *atom z* $\sharp$ Γ'
shows Θ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \{\!\!\{ z : b \mid c \}\!\!\}$
⟨*proof*⟩

lemma *wfT-weakening-all*:
fixes Γ::Γ **and** Γ'::Γ **and** τ::τ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$ **and** Θ; $\mathcal{B}' \vdash_{wf} \Gamma'$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$
shows Θ; $\mathcal{B}'$; $\Gamma' \vdash_{wf} \tau$
⟨*proof*⟩

lemma *wfT-weakening-nil*:
fixes Γ::Γ **and** Γ'::Γ **and** τ::τ
assumes Θ ; $\{||\}$; *GNil* $\vdash_{wf} \tau$ **and** Θ; $\mathcal{B}' \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}'$; $\Gamma' \vdash_{wf} \tau$
⟨*proof*⟩

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**
$(dc, \tau) \in$ *set dclist* **and** Θ ; $B \vdash_{wf} G$
shows *supp* $\tau = \{\}$ **and** $\tau[x::=v]_{\tau v} = \tau$ **and** *wfT* Θ *B G* τ
⟨*proof*⟩

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
Θ; Φ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta$; Φ; $\mathcal{B}$; Γ ; $\Delta' \vdash_{wf}$
$e : b$ **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*⟩

8.15 Useful well-formedness instances

Well-formedness for particular constructs that we will need later

lemma *wfC-e-eq*:
fixes $ce::ce$ **and** $\Gamma::\Gamma$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ce : b$ **and** $atom\ x \sharp \Gamma$
shows $\Theta ; \mathcal{B} ; ((x, b, TRUE) \ \#_\Gamma \ \Gamma) \vdash_{wf} (CE\text{-}val\ (V\text{-}var\ x) \ == \ ce\ )$
⟨*proof*⟩

lemma *wfC-e-eq2*:
fixes $e1::ce$ **and** $e2::ce$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} e1 : b$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} e2 : b$ **and** $\vdash_{wf} \Theta$ **and** $atom\ x \sharp \Gamma$
shows $\Theta; \mathcal{B}; (x, b, (CE\text{-}val\ (V\text{-}var\ x)) == e1) \ \#_\Gamma \ \Gamma \vdash_{wf} (CE\text{-}val\ (V\text{-}var\ x)) == e2$
⟨*proof*⟩

lemma *wfT-wfT-if-rev*:
assumes $wfV\ P\ \mathcal{B}\ \Gamma\ v\ (base\text{-}for\text{-}lit\ l)$ **and** $wfT\ P\ \mathcal{B}\ \Gamma\ t$ **and** ‹$atom\ z1 \sharp \Gamma$›
shows $wfT\ P\ \mathcal{B}\ \Gamma\ (\{\!| z1 : b\text{-}of\ t \mid CE\text{-}val\ v \ == \ CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ (c\text{-}of\ t\ z1) |\!\})$
⟨*proof*⟩

lemma *wfT-eq-imp*:
fixes $zz::x$ **and** $ll::l$ **and** $\tau'::\tau$
assumes $base\text{-}for\text{-}lit\ ll = B\text{-}bool$ **and** $\Theta ; \{||\} ; GNil \vdash_{wf} \tau'$ **and**
$\Theta ; \{||\} \vdash_{wf} (x, b\text{-}of \{\!| z' : B\text{-}bool \mid TRUE |\!\}, c\text{-}of \{\!| z' : B\text{-}bool \mid TRUE |\!\}\ x) \ \#_\Gamma \ GNil$ **and** $atom\ zz \sharp x$
shows $\Theta ; \{||\} ; (x, b\text{-}of \{\!| z' : B\text{-}bool \mid TRUE |\!\}, c\text{-}of \{\!| z' : B\text{-}bool \mid TRUE |\!\}\ x) \ \#_\Gamma$
$GNil \vdash_{wf} \{\!| zz : b\text{-}of\ \tau' \mid [\ [\ x\ ]^v\ ]^{ce} \ == \ [\ [\ ll\ ]^v\ ]^{ce}\ \ IMP\ \ c\text{-}of\ \tau'\ zz\ |\!\}$
⟨*proof*⟩

lemma *wfC-v-eq*:
fixes $ce::ce$ **and** $\Gamma::\Gamma$ **and** $v::v$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$ **and** $atom\ x \sharp \Gamma$
shows $\Theta ; \mathcal{B} ; ((x, b, TRUE) \ \#_\Gamma \ \Gamma) \vdash_{wf} (CE\text{-}val\ (V\text{-}var\ x) \ == \ CE\text{-}val\ v\ )$
⟨*proof*⟩

lemma *wfT-e-eq*:
fixes $ce::ce$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ce : b$ **and** $atom\ z \sharp \Gamma$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!| z : b \mid CE\text{-}val\ (V\text{-}var\ z) == ce |\!\}$

⟨*proof*⟩

lemma *wfT-v-eq*:
  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*)⦄
  ⟨*proof*⟩

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*⟩

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*⟩

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))
⟨*proof*⟩

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*
⟨*proof*⟩

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)
  ⟨*proof*⟩

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**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Longrightarrow$ *True* **and**
$\vdash_{wf} P \Longrightarrow$ *True* **and**
$\Theta; \mathcal{B} \vdash_{wf} b \Longrightarrow$ *True* **and**
wfCE-replace-inside: $\Theta ; \mathcal{B} ; G \vdash_{wf} ce : b' \Longrightarrow G = (\Gamma' @ (x, b, c') \#_\Gamma \Gamma) \Longrightarrow \Theta; \mathcal{B}; ((x,b,TRUE) \#_\Gamma\Gamma) \vdash_{wf} c \Longrightarrow \Theta ; \mathcal{B} ; (\Gamma' @ (x, b, c) \#_\Gamma \Gamma) \vdash_{wf} ce : b'$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
⟨*proof*⟩

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*⟩

lemmas *wf-replace-inside* = *wf-replace-inside1 wf-replace-inside2*

lemma *wfC-replace-cons*:
assumes *wfG P* $\mathcal{B}$ $((x,b,c1) \#_\Gamma\Gamma)$ **and** *wfC P* $\mathcal{B}$ $((x,b,TRUE) \#_\Gamma\Gamma)$ *c2*
shows *wfC P* $\mathcal{B}$ $((x,b,c1) \#_\Gamma\Gamma)$ *c2*
⟨*proof*⟩

lemma *wfC-refl*:
assumes *wfG* Θ $\mathcal{B}$ $((x, b', c') \#_\Gamma\Gamma)$
shows *wfC* Θ $\mathcal{B}$ $((x, b', c') \#_\Gamma\Gamma)$ c'
⟨*proof*⟩

lemma *wfG-wfC-inside*:
assumes $(x, b, c) \in$ *toSet G* **and** *wfG* Θ *B G*
shows *wfC* Θ *B G c*
⟨*proof*⟩

lemma *wfT-wf-cons3*:
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!| z : b \mid c |\!\}$ **and** *atom y* ♯ (c,Γ)
shows $\Theta; \mathcal{B} \vdash_{wf} (y, b, c[z::=V\text{-}var\ y]_{cv}) \#_\Gamma \Gamma$
⟨*proof*⟩

lemma *wfT-wfC-cons*:
assumes *wfT P* $\mathcal{B}$ Γ $\{\!| z1 : b \mid c1 |\!\}$ **and** *wfT P* $\mathcal{B}$ Γ $\{\!| z2 : b \mid c2 |\!\}$ **and** *atom x* ♯ $(c1,c2,\Gamma)$
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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *wfD-unique*:
assumes *wfD P* $\mathcal{B}\ \Gamma\ \Delta$ **and** $(u,\tau') \in$ *setD* Δ **and** $(u,\tau) \in$ *setD* Δ
shows $\tau' = \tau$
⟨*proof*⟩

lemma *replace-in-g-forget*:
fixes $x::x$
assumes *wfG P B G*
shows *atom* $x \notin$ *atom-dom* $G \Longrightarrow (G[x \longmapsto c]) = G$ **and**
atom $x \sharp G \Longrightarrow\ (G[x \longmapsto c]) = G$
⟨*proof*⟩

lemma *replace-in-g-fresh-single*:
fixes $G::\Gamma$ **and** $x::x$
assumes ‹$\Theta;\ \mathcal{B} \vdash_{wf} G[x' \longmapsto c'']$› **and** *atom* $x \sharp G$ **and** ‹$\Theta;\ \mathcal{B} \vdash_{wf} G$ ›
shows *atom* $x \sharp G[x' \longmapsto c'']$
⟨*proof*⟩

8.17 Preservation of well-formedness under substitution

lemma *wfC-cons-switch*:
fixes $c::c$ **and** $c'::c$
assumes $\Theta;\ \mathcal{B};\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma\ \ \vdash_{wf} c'$

shows Θ; $\mathcal{B}$; $(x,\ b,\ c')\ \ \#_\Gamma\ \Gamma\ \ \vdash_{wf}\ c$
⟨*proof*⟩

lemma *subst-g-inside-simple*:
fixes Γ_1::Γ **and** Γ_2::Γ
assumes *wfG P* $\mathcal{B}$ $(\Gamma_1@((x,b,c)\ \#_\Gamma\Gamma_2))$
shows $(\Gamma_1@((x,b,c)\ \#_\Gamma\Gamma_2))[x::=v]_{\Gamma v} = \Gamma_1[x::=v]_{\Gamma v}@\Gamma_2$
⟨*proof*⟩

lemma *subst-c-TRUE-FALSE*:
fixes *c*::*c*
assumes $c \notin \{TRUE,FALSE\}$
shows $c[x::=v']_{cv} \notin \{TRUE,\ FALSE\}$
⟨*proof*⟩

lemma *lookup-subst*:
assumes *Some* $(b,\ c)$ = *lookup* Γ x **and** $x \neq x'$
shows $\exists\, c'$. *Some* (b,c') = *lookup* $\Gamma[x'::=v']_{\Gamma v}\ x$
⟨*proof*⟩

lemma *lookup-subst2*:
assumes *Some* $(b,\ c)$ = *lookup* $(\Gamma'@((x',b_1,c0[z0::=[x']^v]_{cv})\#_\Gamma\Gamma))\ x$ **and** $x \neq x'$ **and**
Θ; $\mathcal{B} \vdash_{wf} (\Gamma'@((x',b_1,c0[z0::=[x']^v]_{cv})\#_\Gamma\Gamma))$
shows $\exists\, c'$. *Some* (b,c') = *lookup* $(\Gamma'[x'::=v']_{\Gamma v}@\Gamma)\ x$
⟨*proof*⟩

lemma *wf-subst1*:
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** *ce*::*ce* **and** *td*::*type-def*
shows *wfV-subst*: Θ; $\mathcal{B}$; $\Gamma\ \vdash_{wf}\ v : b \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$; $\mathcal{B}$;$\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} v[x::=v']_{vv} : b$ **and**
wfC-subst: Θ; $\mathcal{B}$; $\Gamma\ \vdash_{wf}\ c \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$; $\mathcal{B}$; $\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} c[x::=v']_{cv}$ **and**
wfG-subst: Θ; $\mathcal{B} \vdash_{wf} \Gamma \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$; $\mathcal{B}$; $\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; $\mathcal{B} \vdash_{wf} \Gamma[x::=v']_{\Gamma v}$ **and**
wfT-subst: Θ; $\mathcal{B}$; $\Gamma\ \vdash_{wf}\ \tau \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$; $\mathcal{B}$; $\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} \tau[x::=v']_{\tau v}$ **and**
Θ; $\mathcal{B}$; $\Gamma\ \vdash_{wf}\ ts \implies$ *True* **and**
$\vdash_{wf} \Theta \implies$*True* **and**
Θ; $\mathcal{B} \vdash_{wf} b \implies$ *True* **and**
wfCE-subst: Θ; $\mathcal{B}$; $\Gamma \vdash_{wf} ce : b \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$; $\mathcal{B}$; $\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}\ \vdash_{wf}\ ce[x::=v']_{cev} : b$ **and**
$\Theta\ \vdash_{wf}\ td \implies$ *True*
⟨*proof*⟩

lemma *wf-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** *ce*::*ce* **and** *td*::*type-def*
shows Θ; Φ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$; $\mathcal{B}$; $\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; Φ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$; $\Delta[x::=v']_{\Delta v} \vdash_{wf}\ e[x::=v']_{ev} : b$ **and**
Θ; Φ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b \implies \Gamma=\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta$;$\mathcal{B}$; $\Gamma_2 \vdash_{wf} v' : b' \implies \Theta$; Φ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$; $\Delta[x::=v']_{\Delta v} \vdash_{wf}\ s[x::=v']_{sv} : b$ **and**

$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \Longrightarrow \Gamma = \Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b'$
$\Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma[x::=v']_{\Gamma v} ; \Delta[x::=v']_{\Delta v} ; tid ; dc ; t \vdash_{wf}$ *subst-branchv cs x v'* : *b* **and**
$\Theta; \Phi; \mathcal{B}; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \Longrightarrow \Gamma = \Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b'$
$\Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma[x::=v']_{\Gamma v} ; \Delta[x::=v']_{\Delta v} ; tid ; dclist \vdash_{wf}$ *subst-branchlv css x v'* : *b* **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow True$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \Longrightarrow \Gamma = \Gamma_1 @((x,b',c') \#_\Gamma \Gamma_2) \Longrightarrow \Theta; \mathcal{B} ; \Gamma_2 \vdash_{wf} v' : b' \Longrightarrow \Theta ; \mathcal{B} ; \Gamma[x::=v']_{\Gamma v}$
$\vdash_{wf} \Delta[x::=v']_{\Delta v}$ **and**
$\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow True$ **and**
$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow True$
⟨*proof*⟩

lemmas *wf-subst* = *wf-subst1 wf-subst2*

lemma *wfG-subst-wfV*:
assumes $\Theta; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c0[z0::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma$ **and** *wfV* $\Theta\ \mathcal{B}\ \Gamma\ v\ b$
shows $\Theta; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v} @ \Gamma$
⟨*proof*⟩

lemma *wfG-member-subst*:
assumes $(x1,b1,c1) \in toSet\ (\Gamma'@\Gamma)$ **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*⟩

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*⟩

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*⟩

lemma *wfG-inside-fresh-suffix*:
assumes *wfG P B* $(\Gamma'@(x,b,c) \#_\Gamma \Gamma)$
shows *atom x* $\sharp$ Γ
⟨*proof*⟩

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-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 $\Theta ; B' ; \Gamma \vdash_{wf} v : b' \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta ; B \vdash_{wf} b \Longrightarrow \Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$
$v[bv::=b]_{vb} : b'[bv::=b]_{bb}$ **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**
$\vdash_{wf} \Theta \Longrightarrow$ *True* **and**
$\Theta ; B' \vdash_{wf} b' \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta ; B \vdash_{wf} b \Longrightarrow \Theta ; B \vdash_{wf} b'[bv::=b]_{bb}$ **and**
$\Theta ; B' ; \Gamma \vdash_{wf} ce : b' \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta ; B \vdash_{wf} b \Longrightarrow \Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} ce[bv::=b]_{ceb} : b'[bv::=b]_{bb}$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
⟨*proof*⟩

lemma *wf-b-subst2*:
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** $s::s$ **and** $b'::b$ **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows $\Theta ; \Phi ; B' ; \Gamma ; \Delta \vdash_{wf} e : b' \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta ; B \vdash_{wf} b \Longrightarrow \Theta ; \Phi ; B ; \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash_{wf} e[bv::=b]_{eb} : b'[bv::=b]_{bb}$ **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**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow$ *True* **and**
$\Theta ; B' ; \Gamma \vdash_{wf} \Delta \Longrightarrow \{|bv|\} = B' \Longrightarrow \Theta ; B \vdash_{wf} b \Longrightarrow \Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **and**
$\Theta ; \Phi \vdash_{wf} ftq \Longrightarrow$ *True* **and**
$\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \Longrightarrow$ *True*
⟨*proof*⟩

lemmas *wf-b-subst* = *wf-b-subst1 wf-b-subst2*

lemma *wfT-subst-wfT*:
fixes $\tau::\tau$ **and** $b'::b$ **and** $bv::bv$
assumes $\Theta ; \{|bv|\} ; (x,b,c) \#_\Gamma GNil \vdash_{wf} \tau$ **and** $\Theta ; B \vdash_{wf} b'$
shows $\Theta ; B ; (x,b[bv::=b']_{bb},c[bv::=b']_{cb}) \#_\Gamma GNil \vdash_{wf} (\tau[bv::=b']_{\tau b})$
⟨*proof*⟩

lemma *wf-trans*:
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** *cs*::*branch-s* **and** *css*::*branch-list* **and** $\Theta::\Theta$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b' \Longrightarrow \Gamma = (x, b, c2) \#_\Gamma G \Longrightarrow \Theta; \mathcal{B}; (x, b, c1) \#_\Gamma G \vdash_{wf} c2 \Longrightarrow \Theta; \mathcal{B}; (x, b, c1) \#_\Gamma G \vdash_{wf} v : b'$ **and**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \Longrightarrow \Gamma = (x, b, c2) \#_\Gamma G \Longrightarrow \Theta; \mathcal{B}; (x, b, c1) \#_\Gamma G \vdash_{wf} c2 \Longrightarrow \Theta; \mathcal{B}; (x, b, c1) \#_\Gamma G \vdash_{wf} c$ **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**
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b' \Longrightarrow \Gamma = (x, b, c2) \#_\Gamma G \Longrightarrow \Theta; \mathcal{B}; (x, b, c1) \#_\Gamma G \vdash_{wf} c2 \Longrightarrow \Theta; \mathcal{B}; (x, b, c1) \#_\Gamma G \vdash_{wf} ce : b'$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
⟨*proof*⟩

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 ♯ (Θ, $\mathcal{B}$, Γ, z,c,z',c') ;
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ⦃ z : b | c ⦄;
Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ ⦃ z' : b | c' ⦄;
Θ; $\mathcal{B}$; (x,b, $c[z{::=}[x]^v]_v$) $\#_\Gamma$ $\Gamma \models c'[z'{::=}[x]^v]_v$
⟧ $\Longrightarrow$
Θ; $\mathcal{B}$; $\Gamma \vdash$ ⦃ z : b | c ⦄ $\lesssim$ ⦃ z' : b | c' ⦄

equivariance *subtype*
nominal-inductive *subtype*
avoids *subtype-baseI*: x
⟨*proof*⟩

inductive-cases *subtype-elims*:
Θ; $\mathcal{B}$; $\Gamma \vdash$ ⦃ z : b | c ⦄ $\lesssim$ ⦃ z' : b | c' ⦄
Θ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim \tau_2$

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: ⊢ *L-true* ⇒ ⦃ z : *B-bool* | $[[z]^v]^{ce}$ == $[[L\text{-}true]^v]^{ce}$ ⦄
| *infer-falseI*: ⊢ *L-false* ⇒ ⦃ z : *B-bool* | $[[z]^v]^{ce}$ == $[[L\text{-}false]^v]^{ce}$ ⦄
| *infer-natI*: ⊢ *L-num n* ⇒ ⦃ z : *B-int* | $[[z]^v]^{ce}$ == $[[L\text{-}num\ n]^v]^{ce}$ ⦄
| *infer-unitI*: ⊢ *L-unit* ⇒ ⦃ z : *B-unit* | $[[z]^v]^{ce}$ == $[[L\text{-}unit]^v]^{ce}$ ⦄
| *infer-bitvecI*: ⊢ *L-bitvec bv* ⇒ ⦃ z : *B-bitvec* | $[[z]^v]^{ce}$ == $[[L\text{-}bitvec\ bv]^v]^{ce}$ ⦄

nominal-inductive *infer-l* ⟨*proof*⟩
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 $\exists z. \vdash l \Rightarrow (\{\!\!\{ z : \textit{base-for-lit } l \mid [[z]^v]^{ce} == [[l]^v]^{ce} \}\!\!\})$
⟨*proof*⟩

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: ⟦
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma$;
 $Some\ (b,c) = lookup\ \Gamma\ x$;
 $atom\ z \sharp x$; $atom\ z \sharp (\Theta, \mathcal{B}, \Gamma)$
⟧ ⟹
 $\Theta; \mathcal{B}; \Gamma \vdash [x]^v \Rightarrow \{\!\!\{ z : b \mid [[z]^v]^{ce} == [[x]^v]^{ce} \}\!\!\}$

| *infer-v-litI*: ⟦
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma$;
 $\vdash l \Rightarrow \tau$
⟧ ⟹
 $\Theta; \mathcal{B}; \Gamma \vdash [l]^v \Rightarrow \tau$

| *infer-v-pairI*: ⟦
 $atom\ z \sharp (v1, v2)$; $atom\ z \sharp (\Theta, \mathcal{B}, \Gamma)$;
 $\Theta; \mathcal{B}; \Gamma \vdash (v1::v) \Rightarrow t1$;
 $\Theta; \mathcal{B}; \Gamma \vdash (v2::v) \Rightarrow t2$
⟧ ⟹
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-}pair\ v1\ v2 \Rightarrow (\{\!\!\{ z : B\text{-}pair\ (b\text{-}of\ t1)\ (b\text{-}of\ t2) \mid [[z]^v]^{ce} == [[v1,v2]^v]^{ce} \}\!\!\})$

| *infer-v-consI*: ⟦
 $AF\text{-}typedef\ s\ dclist \in set\ \Theta$;
 $(dc, tc) \in set\ dclist$;
 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow tv$;
 $\Theta; \mathcal{B}; \Gamma \vdash tv \lesssim tc$;
 $atom\ z \sharp v$; $atom\ z \sharp (\Theta, \mathcal{B}, \Gamma)$
⟧ ⟹
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-}cons\ s\ dc\ v \Rightarrow (\{\!\!\{ z : B\text{-}id\ s \mid [[z]^v]^{ce} == [\ V\text{-}cons\ s\ dc\ v\ ]^{ce} \}\!\!\})$

| *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 $\sharp$ $(\Theta, \mathcal{B}, \Gamma, v, b)$;
$\Theta; \mathcal{B} \vdash_{wf} b$
$\rrbracket \Longrightarrow$
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-consp s dc b v* $\Rightarrow$ ($\{\!|$ *z* : *B-app s b* | $[[z]^v]^{ce}$ == (*CE-val* (*V-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*⟩

inductive-cases *infer-v-elims*[*elim*!]:
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-var x* $\Rightarrow \tau$
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-lit l* $\Rightarrow \tau$
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-pair v1 v2* $\Rightarrow \tau$
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-cons s dc v* $\Rightarrow \tau$
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-pair v1 v2* $\Rightarrow$ ($\{\!|$ *z* : *b* | *c* $|\!\}$)
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-pair v1 v2* $\Rightarrow$ ($\{\!|$ *z* : $[\ b1\ ,\ b2\ ]^b$ | $[[z]^v]^{ce}$ == $[[v1,v2]^v]^{ce}$ $|\!\}$)
$\Theta; \mathcal{B}; \Gamma \vdash$ *V-consp s dc b v* $\Rightarrow \tau$

inductive *check-v* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow$ *bool* (‹- ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*] *50*) **where**
check-v-subtypeI: $\llbracket\ \Theta; \mathcal{B}; \Gamma \vdash \tau 1 \lesssim \tau 2; \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau 1\ \rrbracket \Longrightarrow \Theta; \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \tau 2$
equivariance *check-v*
nominal-inductive *check-v* ⟨*proof*⟩

inductive-cases *check-v-elims*[*elim*!]:
$\Theta; \mathcal{B}\ ; \Gamma \vdash v \Leftarrow \tau$

9.4 Expressions

Type synthesis for expressions

inductive *infer-e* :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow \tau \Rightarrow$ *bool* (‹- ; - ; - ; - ; - ⊢ - ⇒ -› [*50*, *50*, *50*,*50*] *50*) **where**

infer-e-valI: $\llbracket$
$(\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta)$;
$(\Theta \vdash_{wf} (\Phi::\Phi))$;
$(\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau)\ \rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash$ (*AE-val v*) $\Rightarrow \tau$

| *infer-e-plusI*: $\llbracket$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash$ *v1* $\Rightarrow$ $\{\!|$ *z1* : *B-int* | *c1* $|\!\}$;
$\Theta; \mathcal{B}; \Gamma \vdash$ *v2* $\Rightarrow$ $\{\!|$ *z2* : *B-int* | *c2* $|\!\}$;
atom z3 $\sharp$ (*AE-op Plus v1 v2*); *atom z3* $\sharp$ $\Gamma\ \rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash$ *AE-op Plus v1 v2* $\Rightarrow$ $\{\!|$ *z3* : *B-int* | $[[z3]^v]^{ce}$ == (*CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$) $|\!\}$

| *infer-e-leqI*: $\llbracket$

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{\!\mid z1 : B\text{-}int \mid c1 \mid\!\}$;
$\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{\!\mid z2 : B\text{-}int \mid c2 \mid\!\}$;
$atom\ z3 \sharp (AE\text{-}op\ LEq\ v1\ v2);\ atom\ z3 \sharp \Gamma$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-}op\ LEq\ v1\ v2 \Rightarrow \{\!\mid z3 : B\text{-}bool \mid [[z3]^v]^{ce} == (CE\text{-}op\ LEq\ [v1]^{ce}\ [v2]^{ce}) \mid\!\}$

$|$ *infer-e-eqI*: $[\![$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{\!\mid z1 : b \mid c1 \mid\!\}$;
$\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{\!\mid z2 : b \mid c2 \mid\!\}$;
$atom\ z3 \sharp (AE\text{-}op\ Eq\ v1\ v2);\ atom\ z3 \sharp \Gamma$;
$b \in \{\ B\text{-}bool,\ B\text{-}int,\ B\text{-}unit\ \}$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-}op\ Eq\ v1\ v2 \Rightarrow \{\!\mid z3 : B\text{-}bool \mid [[z3]^v]^{ce} == (CE\text{-}op\ Eq\ [v1]^{ce}\ [v2]^{ce}) \mid\!\}$

$|$ *infer-e-appI*: $[\![$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f;$
$\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{\!\mid x : b \mid c \mid\!\}$;
$atom\ x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v\ , \tau);$
$\tau'[x::=v]_v = \tau$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-}app\ f\ v \Rightarrow \tau$

$|$ *infer-e-appPI*: $[\![$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B} \vdash_{wf} b'$;
$Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f;$
$\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{\!\mid x : b[bv::=b']_b \mid c[bv::=b']_b \mid\!\};\ atom\ x \sharp \Gamma;$
$(\tau'[bv::=b']_b[x::=v]_v) = \tau$;
$atom\ bv \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, b', v, \tau)$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-}appP\ f\ b'\ v \Rightarrow \tau$

$|$ *infer-e-fstI*: $[\![$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\mid z' : [b1,b2]^b \mid c \mid\!\}$;
$atom\ z \sharp AE\text{-}fst\ v$; $atom\ z \sharp \Gamma\ ]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-}fst\ v \Rightarrow \{\!\mid z : b1 \mid [[z]^v]^{ce} == ((CE\text{-}fst\ [v]^{ce})) \mid\!\}$

$|$ *infer-e-sndI*: $[\![$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\mid z' : [\ b1,\ b2]^b \mid c \mid\!\}$;
$atom\ z \sharp AE\text{-}snd\ v$; $atom\ z \sharp \Gamma\ ]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-}snd\ v \Rightarrow \{\!\mid z : b2 \mid [[z]^v]^{ce} == ((CE\text{-}snd\ [v]^{ce})) \mid\!\}$

| *infer-e-lenI*: $\llbracket$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!| z' : \textit{B-bitvec} \mid c |\!\}$;
atom z ♯ *AE-len v* ; *atom z* ♯ $\Gamma\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash \textit{AE-len } v \Rightarrow \{\!| z : \textit{B-int} \mid [[z]^v]^{ce} == ((\textit{CE-len } [v]^{ce}))\ |\!\}$

| *infer-e-mvarI*: $\llbracket$
$\Theta; \mathcal{B} \vdash_{wf} \Gamma$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$(u,\tau) \in \textit{setD } \Delta \rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash\ \textit{AE-mvar } u \Rightarrow \tau$

| *infer-e-concatI*: $\llbracket$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{\!| z1 : \textit{B-bitvec} \mid c1 |\!\}$;
$\Theta; \mathcal{B}; \Gamma \vdash\ v2 \Rightarrow \{\!| z2 : \textit{B-bitvec} \mid c2 |\!\}$;
atom z3 ♯ (*AE-concat v1 v2*); *atom z3* ♯ $\Gamma \rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash \textit{AE-concat}\ \ v1\ v2 \Rightarrow \{\!| z3 : \textit{B-bitvec} \mid [[z3]^v]^{ce} == (\textit{CE-concat } [v1]^{ce}\ [v2]^{ce})\ |\!\}$

| *infer-e-splitI*: $\llbracket$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
$\Theta ; \mathcal{B} ; \Gamma \vdash v1 \Rightarrow \{\!| z1 : \textit{B-bitvec} \mid c1 |\!\}$;
$\Theta ; \mathcal{B} ; \Gamma \vdash v2 \Leftarrow \{\!| z2 : \textit{B-int} \mid$ (*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
$\rrbracket \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash$ (*AE-split v1 v2*) $\Rightarrow \{\!| z3 : \textit{B-pair B-bitvec B-bitvec} \mid$
((*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*⟩

inductive-cases *infer-e-elims*[*elim*!]:
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\textit{AE-op Plus } v1\ v2) \Rightarrow \{\!| z3 : \textit{B-int} \mid [[z3]^v]^{ce} == (\textit{CE-op Plus } [v1]^{ce}\ [v2]^{ce})\ |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\textit{AE-op LEq } v1\ v2) \Rightarrow \{\!| z3 : \textit{B-bool} \mid [[z3]^v]^{ce} == (\textit{CE-op LEq } [v1]^{ce}\ [v2]^{ce})\ |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\textit{AE-op Plus } v1\ v2) \Rightarrow \{\!| z3 : \textit{B-int} \mid c |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\textit{AE-op Plus } v1\ v2) \Rightarrow \{\!| z3 : b \mid c |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\textit{AE-op LEq } v1\ v2) \Rightarrow \{\!| z3 : b \mid c |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\textit{AE-app } f\ v\ ) \Rightarrow \tau$

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}val}\ v) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}fst}\ v) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}snd}\ v) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}mvar}\ u) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ Plus\ v1\ v2}) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ LEq\ v1\ v2}) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ LEq\ v1\ v2}) \Rightarrow \{\!| \ z3 : \mathit{B\text{-}bool} \mid c \ |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}app}\ f\ v\ ) \Rightarrow \tau[x::=v]_{\tau v}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ opp\ v1\ v2}) \Rightarrow \ \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}len}\ v) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}len}\ v) \Rightarrow \{\!| \ z : \mathit{B\text{-}int} \mid [[z]^v]^{ce} == ((\mathit{CE\text{-}len}\ [v]^{ce}))|\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash \mathit{AE\text{-}concat\ v1\ v2} \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash \mathit{AE\text{-}concat\ v1\ v2} \Rightarrow (\{\!| \ z : b \mid \ c \ |\!\})$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash \mathit{AE\text{-}concat\ v1\ v2} \Rightarrow (\{\!| \ z : \mathit{B\text{-}bitvec} \mid \ [[z]^v]^{ce} == (\mathit{CE\text{-}concat}\ [v1]^{ce}\ [v1]^{ce}) \ |\!\})$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}appP}\ f\ b\ v\ ) \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash \mathit{AE\text{-}split\ v1\ v2} \Rightarrow \tau$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ Eq\ v1\ v2}) \Rightarrow \{\!| \ z3 : b \mid c \ |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ Eq\ v1\ v2}) \Rightarrow \{\!| \ z3 : \mathit{B\text{-}bool} \mid c \ |\!\}$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AE\text{-}op\ Eq\ v1\ v2}) \Rightarrow \tau$
nominal-termination (*eqvt*) ⟨*proof*⟩

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* ⇒ $v \Rightarrow$ *branch-list* ⇒ $\tau \Rightarrow$ *bool* (‹ - ; - ; - ; - ; - ; - ; - ; - ⊢ - ⇐ -› [*50*, *50*, *50*,*50*,*50*] *50*) **where**
check-valI: ⟦
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau'$;
$\Theta; \mathcal{B}; \Gamma \vdash \tau' \lesssim \tau$ ⟧ ⟹
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AS\text{-}val}\ v) \Leftarrow \tau$

| *check-letI*: ⟦
atom x ♯ $(\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau)$;
atom z ♯ $(x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau, s)$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash e \Rightarrow \{\!| \ z : b \mid c \ |\!\}$;
$\Theta; \Phi\ ; \mathcal{B}\ ; ((x,b,c[z::=\mathit{V\text{-}var}\ x]_v)\#_\Gamma\Gamma)\ ; \Delta \vdash s \Leftarrow \tau$
⟧ ⟹
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AS\text{-}let}\ x\ e\ s) \Leftarrow \tau$

| *check-assertI*: ⟦
atom x ♯ $(\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, c, \tau, s)$;
$\Theta; \Phi\ ; \mathcal{B}\ ; ((x,\mathit{B\text{-}bool},c)\#_\Gamma\Gamma)\ ; \Delta \vdash s \Leftarrow \tau$;
$\Theta; \mathcal{B}; \Gamma \ \models c$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$
⟧ ⟹
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (\mathit{AS\text{-}assert}\ c\ s) \Leftarrow \tau$

$|$*check-branch-s-branchI*: $[\![$
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$\vdash_{wf} \Theta$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$;
Θ ; $\{||\}$; $GNil \vdash_{wf} const$;
$atom\ x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, tid, cons\ , const, v, \tau)$;
$\Theta; \Phi; \mathcal{B}; ((x, b\text{-}of\ const,\ ([v]^{ce} == [\ V\text{-}cons\ tid\ cons\ [x]^{v}]^{ce}\ )\ AND\ (c\text{-}of\ const\ x))\#_{\Gamma}\Gamma)\ ;\ \Delta \vdash s \Leftarrow \tau$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta\ ;\ tid\ ;\ cons\ ;\ const\ ;\ v \vdash (AS\text{-}branch\ cons\ x\ s) \Leftarrow \tau$

$|$*check-branch-list-consI*: $[\![$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; cons; const; v \vdash cs \Leftarrow \tau$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist; v \vdash css \Leftarrow \tau$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta\ ;\ tid\ ;\ (cons,const)\#dclist\ ;\ v \vdash AS\text{-}cons\ cs\ css \Leftarrow \tau$

$|$*check-branch-list-finalI*: $[\![$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid\ ;\ cons\ ;\ const\ ;\ v \vdash cs \Leftarrow \tau$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta\ ;\ tid\ ;\ [(cons,const)]\ ;\ v \vdash AS\text{-}final\ cs\ \Leftarrow \tau$

$|$ *check-ifI*: $[\![$
$atom\ z \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v\ , s1\ , s2\ , \tau\ )$;
$(\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow (\{\!|\ z : B\text{-}bool \mid TRUE\ |\!\}))$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow (\{\!|\ z : b\text{-}of\ \tau \mid ([v]^{ce} == [[L\text{-}true]^{v}]^{ce})\ IMP\ (c\text{-}of\ \tau\ z)\ |\!\})$;
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow (\{\!|\ z : b\text{-}of\ \tau \mid ([v]^{ce} == [[L\text{-}false]^{v}]^{ce})\ IMP\ (c\text{-}of\ \tau\ z)\ |\!\})$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash IF\ v\ THEN\ s1\ ELSE\ s2 \Leftarrow \tau$

$|$ *check-let2I*: $[\![$
$atom\ x \sharp (\Theta, \Phi, \mathcal{B}, G, \Delta, t, s1, \tau)$;
$\Theta; \Phi\ ;\ \mathcal{B}\ ;\ G; \Delta \vdash s1 \Leftarrow t$;
$\Theta; \Phi\ ;\ \mathcal{B}\ ;\ ((x, b\text{-}of\ t, c\text{-}of\ t\ x)\#_{\Gamma}G)\ ;\ \Delta \vdash s2 \Leftarrow \tau$
$]\!] \Longrightarrow$
$\Theta; \Phi\ ;\ \mathcal{B}\ ;\ G\ ;\ \Delta \vdash (LET\ x : t\ = s1\ IN\ s2) \Leftarrow \tau$

$|$ *check-varI*: $[\![$
$atom\ u \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \tau', v, \tau)$;
$\Theta; \mathcal{B}; \Gamma \vdash\ v \Leftarrow \tau'$;
$\Theta; \Phi; \mathcal{B}; \Gamma\ ;\ ((u,\tau')\ \#_{\Delta}\ \Delta)\ \vdash s \Leftarrow \tau$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (VAR\ u : \tau' = v\ \ IN\ s) \Leftarrow \tau$

$|$ *check-assignI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$;
$(u,\tau) \in setD\ \Delta$;
$\Theta; \mathcal{B}; \Gamma \vdash\ v \Leftarrow \tau$;
$\Theta; \mathcal{B}; \Gamma \vdash (\{\!|\ z : B\text{-}unit \mid TRUE\ |\!\}) \lesssim \tau'$
$]\!] \Longrightarrow$
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash\ (u ::= v) \Leftarrow \tau'$

| *check-whileI*: ⟦
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash s1 \Leftarrow \{\!\mid z : B\text{-}bool \mid TRUE \mid\!\}$;
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash s2 \Leftarrow \{\!\mid z : B\text{-}unit \mid TRUE \mid\!\}$;
  $\Theta;\ \mathcal{B};\ \Gamma \vdash (\{\!\mid z : B\text{-}unit \mid TRUE \mid\!\}) \lesssim \tau'$
⟧ $\Longrightarrow$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash WHILE\ s1\ DO\ \{\ s2\ \} \Leftarrow \tau'$

| *check-seqI*: ⟦
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash s1 \Leftarrow \{\!\mid z : B\text{-}unit \mid TRUE \mid\!\}$;
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash s2 \Leftarrow \tau$
⟧ $\Longrightarrow$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash s1\ ;;\ s2 \Leftarrow \tau$

| *check-caseI*: ⟦
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta;\ tid\ ;\ dclist\ ;\ v \vdash\ cs \Leftarrow \tau$;
  $(AF\text{-}typedef\ tid\ dclist\ ) \in set\ \Theta$;
  $\Theta;\ \mathcal{B};\ \Gamma \vdash v \Leftarrow \{\!\mid z : B\text{-}id\ tid \mid TRUE \mid\!\}$;
  $\vdash_{wf} \Theta$
⟧ $\Longrightarrow$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}match\ v\ cs \Leftarrow \tau$

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*⟩

inductive-cases *check-s-elims*[*elim*!]:
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}val\ v \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}let\ x\ e\ s \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}if\ v\ s1\ s2 \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}let2\ x\ t\ s1\ s2 \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}while\ s1\ s2 \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}var\ u\ t\ v\ s \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}seq\ s1\ s2 \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}assign\ u\ v \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}match\ v\ cs \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta \vdash AS\text{-}assert\ c\ s \Leftarrow \tau$

inductive-cases *check-branch-s-elims*[*elim*!]:
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta;\ tid\ ;\ dclist\ ;\ v \vdash (AS\text{-}final\ cs) \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta;\ tid\ ;\ dclist\ ;\ v \vdash (AS\text{-}cons\ cs\ css) \Leftarrow \tau$
  $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma;\ \Delta;\ tid\ ;\ cons\ ;\ const\ ;\ v \vdash (AS\text{-}branch\ dc\ x\ s\ ) \Leftarrow \tau$

9.6 Programs

Type check function bodies

inductive *check-funtyp* :: $\Theta \Rightarrow \Phi \Rightarrow\ \mathcal{B} \Rightarrow fun\text{-}typ \Rightarrow bool$ (‹ - ; - ; - ⊢ - ›) **where**

check-funtypI: ⟦
atom x ♯ (Θ, Φ, B , b);
Θ; Φ ; B ; ((x,b,c) $\#_\Gamma$ *GNil*) ; $[]_\Delta$ ⊢ $s \Leftarrow \tau$
⟧ ⟹
Θ; Φ ; B ⊢ (*AF-fun-typ* x b c τ s)

equivariance *check-funtyp*
nominal-inductive *check-funtyp*
avoids *check-funtypI*: x
⟨*proof*⟩

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*⟩

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* ⟨*proof*⟩

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* $\Rightarrow$ Δ **where**
 Δ-*of* [] = *DNil*
| Δ-*of* ((*AV-def u t v*)#*vs*) = (*u*,*t*) $\#_\Delta$ (Δ-*of vs*)
 ⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

inductive *check-prog* :: *p* $\Rightarrow$ τ $\Rightarrow$ *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)))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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* ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

Reduction rules

inductive *reduce-stmt* :: $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow bool$ (‹ - ⊢ ⟨ - , -⟩ ⟶ ⟨ - , -⟩› [*50*, *50*, *50*] *50*)
where
reduce-if-trueI: $\Phi \vdash \langle\delta,\ AS\text{-}if\ [L\text{-}true]^v\ s1\ s2\rangle \longrightarrow \langle\delta,\ s1\rangle$
| *reduce-if-falseI*: $\Phi \vdash \langle\delta,\ AS\text{-}if\ [L\text{-}false]^v\ s1\ s2\rangle \longrightarrow \langle\delta,\ s2\rangle$
| *reduce-let-valI*: $\Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ v)\ \ s\rangle \longrightarrow \langle\delta,\ s[x{::=}v]_{sv}\rangle$
| *reduce-let-plusI*: $\Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}op\ Plus\ ((V\text{-}lit\ (L\text{-}num\ n1)))\ ((V\text{-}lit\ (L\text{-}num\ n2))))\ \ s\rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (\ ((\ n1)+(n2))))))\ s\ \rangle$
| *reduce-let-leqI*: $b = (if\ (n1 \le n2)\ then\ L\text{-}true\ else\ L\text{-}false) \Longrightarrow$
$\Phi\ \vdash\ \langle\delta,\ \ AS\text{-}let\ x\ \ ((AE\text{-}op\ LEq\ (V\text{-}lit\ (L\text{-}num\ n1))\ (V\text{-}lit\ (L\text{-}num\ n2))))\ s\rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let\ x\ \ (AE\text{-}val\ (V\text{-}lit\ b))\ s\rangle$
| *reduce-let-eqI*: $b = (if\ (n1 = n2)\ then\ L\text{-}true\ else\ L\text{-}false) \Longrightarrow$
$\Phi\ \vdash\ \langle\delta,\ \ AS\text{-}let\ x\ \ ((AE\text{-}op\ Eq\ (V\text{-}lit\ n1)\ (V\text{-}lit\ n2)))\ s\rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let\ x\ \ (AE\text{-}val\ (V\text{-}lit\ b))\ s\rangle$
| *reduce-let-appI*: $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ z\ b\ c\ \tau\ s'))) = lookup\text{-}fun\ \Phi\ f \Longrightarrow$

$\Phi\ \vdash\ \langle\delta,\ AS\text{-}let\ x\ \ ((AE\text{-}app\ f\ v))\ s\rangle \longrightarrow \langle\delta,\ \ AS\text{-}let2\ x\ \tau[z{::=}v]_{\tau v}\ s'[z{::=}v]_{sv}\ s\rangle$
| *reduce-let-appPI*: $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ z\ b\ c\ \tau\ s'))) = lookup\text{-}fun\ \Phi$
$f \Longrightarrow$
$\Phi\ \vdash\ \langle\delta,\ AS\text{-}let\ x\ \ ((AE\text{-}appP\ f\ b'\ v))\ s\rangle \longrightarrow \langle\delta,\ \ AS\text{-}let2\ x\ \tau[bv{::=}b']_{\tau b}[z{::=}v]_{\tau v}$
$s'[bv{::=}b']_{sb}[z{::=}v]_{sv}\ s\rangle$
| *reduce-let-fstI*: $\Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}fst\ (V\text{-}pair\ v1\ v2))\ \ s\rangle \longrightarrow \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ v1)\ \ s\rangle$
| *reduce-let-sndI*: $\Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}snd\ (V\text{-}pair\ v1\ v2))\ \ s\rangle \longrightarrow \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ v2)\ \ s\rangle$
| *reduce-let-concatI*: $\Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}concat\ (V\text{-}lit\ (L\text{-}bitvec\ v1))\ \ (V\text{-}lit\ (L\text{-}bitvec\ v2)))\ \ s\rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}bitvec\ (v1@v2))))\ \ s\rangle$
| *reduce-let-splitI*: $split\ n\ v\ (v1\ ,\ v2\ ) \Longrightarrow \Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}split\ (V\text{-}lit\ (L\text{-}bitvec\ v))\ \ (V\text{-}lit\ (L\text{-}num$
$n)))\ \ s\rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}pair\ (V\text{-}lit\ (L\text{-}bitvec\ v1))\ (V\text{-}lit\ (L\text{-}bitvec\ v2))))\ \ s\rangle$
| *reduce-let-lenI*: $\Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}len\ (V\text{-}lit\ (L\text{-}bitvec\ v)))\ \ s\rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (int\ (List.length\ v)))))\ \ s\rangle$
| *reduce-let-mvar*: $(u,v) \in set\ \delta \Longrightarrow\ \Phi \vdash \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}mvar\ u)\ \ s\rangle \longrightarrow \langle\delta,\ AS\text{-}let\ x\ (AE\text{-}val\ v)\ s\ \rangle$
| *reduce-assert1I*: $\Phi \vdash \langle\delta,\ AS\text{-}assert\ c\ (AS\text{-}val\ v)\rangle \longrightarrow \langle\delta,\ AS\text{-}val\ v\rangle$
| *reduce-assert2I*: $\Phi\ \vdash \langle\delta,\ s\ \rangle \longrightarrow \langle\ \delta',\ s'\rangle \Longrightarrow \Phi\ \vdash \langle\delta,\ AS\text{-}assert\ c\ s\rangle \longrightarrow \langle\ \delta',\ AS\text{-}assert\ c\ s'\rangle$
| *reduce-varI*: $atom\ u\ \sharp\ \delta \Longrightarrow\ \Phi\ \vdash\ \langle\delta,\ AS\text{-}var\ u\ \tau\ v\ s\ \rangle \longrightarrow \langle\ ((u,v)\#\delta)\ ,\ s\rangle$
| *reduce-assignI*: $\Phi\ \vdash\ \langle\delta,\ AS\text{-}assign\ u\ v\ \rangle \longrightarrow \langle\ update\text{-}d\ \delta\ u\ v\ ,\ AS\text{-}val\ (V\text{-}lit\ L\text{-}unit)\rangle$
| *reduce-seq1I*: $\Phi\ \ \vdash\ \langle\delta,\ AS\text{-}seq\ (AS\text{-}val\ (V\text{-}lit\ L\text{-}unit\ ))\ s\ \rangle \longrightarrow \langle\delta,\ s\rangle$
| *reduce-seq2I*: $[\![s1 \ne AS\text{-}val\ v\ ;\ \Phi\ \vdash\ \langle\delta,\ s1\rangle \longrightarrow \langle\ \delta',\ s1'\rangle\ ]\!] \Longrightarrow$
$\Phi\ \vdash\ \langle\delta,\ AS\text{-}seq\ s1\ s2\rangle \longrightarrow \langle\ \delta',\ AS\text{-}seq\ s1'\ s2\rangle$
| *reduce-let2-valI*: $\Phi\ \vdash\ \langle\delta,\ AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s\rangle \longrightarrow \langle\delta,\ s[x{::=}v]_{sv}\rangle$
| *reduce-let2I*: $\Phi\ \vdash \langle\delta,\ s1\ \rangle \longrightarrow \langle\ \delta',\ s1'\rangle \Longrightarrow \Phi\ \vdash \langle\delta,\ AS\text{-}let2\ x\ t\ s1\ s2\rangle \longrightarrow \langle\ \delta',\ AS\text{-}let2\ x\ t\ s1'\ s2\rangle$

| *reduce-caseI*: $[\![\ Some\ (AS\text{-}branch\ dc\ x'\ s') = lookup\text{-}branch\ dc\ cs\ ]\!] \Longrightarrow\ \Phi \vdash\ \langle\delta,\ AS\text{-}match\ (V\text{-}cons$
$tyid\ dc\ v)\ cs\rangle\ \longrightarrow \langle\delta,\ s'[x'{::=}v]_{sv}\rangle$
| *reduce-whileI*: $[\![\ atom\ x\ \sharp\ (s1,s2);\ atom\ z\ \sharp\ x\ ]\!] \Longrightarrow$
$\Phi\ \vdash\ \langle\delta,\ AS\text{-}while\ s1\ s2\ \rangle \longrightarrow$
$\langle\delta,\ AS\text{-}let2\ x\ (\{\!|\ z : B\text{-}bool\ \mid\ TRUE\ |\!\})\ s1\ (AS\text{-}if\ (V\text{-}var\ x)\ (AS\text{-}seq\ s2\ (AS\text{-}while\ s1\ s2))$
$(AS\text{-}val\ (V\text{-}lit\ L\text{-}unit)))\ \rangle$

equivariance *reduce-stmt*
nominal-inductive *reduce-stmt* ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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* ⟨*proof*⟩

inductive-cases *delta-sim-elims*[*elim*!]:
$\Theta \vdash [] \sim []_\Delta$
$\Theta \vdash ((u,v)\#ds) \sim (u,\tau)\ \#_\Delta\ D$
$\Theta \vdash ((u,v)\#ds) \sim D$

A typing judgement that combines typing of the statement, the store and the condition that definitions are well-typed

inductive *config-type* :: $\Theta \Rightarrow \Phi \Rightarrow \Delta \Rightarrow \delta \Rightarrow s \Rightarrow \tau \Rightarrow$ *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* ⟨*proof*⟩

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*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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]$_{Γv}$ **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]$_{Δv}$ **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]$_{Γv}$ @ Γ **for** *z x v* Γ′ Γ ⇒ ‹*metis add* ›)
| (*match* **conclusion in** *atom z* ♯ (τ::τ)[x::=v]$_{τv}$ **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
⟨*proof*⟩

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
  ⟨*proof*⟩

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*⟩

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*⟩

lemma *wfI-suffix*:
  fixes G::Γ
  assumes *wfI* P $(G'@G)$ i **and** P ; $B \vdash_{wf} G$
  shows P ; $G \vdash i$
  ⟨*proof*⟩

lemma *wfI-replace-inside*:
  assumes *wfI* Θ $(\Gamma'$ @ $(x, b, c) \#_{\Gamma} \Gamma)$ i
  shows *wfI* Θ $(\Gamma'$ @ $(x, b, c') \#_{\Gamma} \Gamma)$ i
  ⟨*proof*⟩

11.2 Existence of evaluation

lemma *eval-l-base*:
  $\Theta \vdash [\![l]\!]$: (*base-for-lit* l)
  ⟨*proof*⟩

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*⟩

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*⟩

lemma *eval-v-exist*:
fixes Γ::Γ **and** *v*::*v* **and** *b*::*b*
assumes $P ; \Gamma \vdash i$ **and** $P ; B ; \Gamma \vdash_{wf} v : b$
shows $\exists s.\ i [\![v]\!] \sim s \wedge P \vdash s : b$
⟨*proof*⟩

lemma *eval-v-uniqueness*:
fixes *v*::*v*
assumes $i [\![v]\!] \sim s$ **and** $i [\![v]\!] \sim s'$
shows $s=s'$
⟨*proof*⟩

lemma *eval-v-base*:
fixes Γ::Γ **and** *v*::*v* **and** *b*::*b*
assumes $P ; \Gamma \vdash i$ **and** $P ; B ; \Gamma \vdash_{wf} v : b$ **and** $i [\![v]\!] \sim s$
shows $P \vdash s : b$
⟨*proof*⟩

lemma *eval-e-uniqueness*:
fixes *e*::*ce*
assumes $i [\![e]\!] \sim s$ **and** $i [\![e]\!] \sim s'$
shows $s=s'$
⟨*proof*⟩

lemma *wfV-eval-bitvec*:
fixes *v*::*v*
assumes $P ; B ; \Gamma \vdash_{wf} v :$ *B-bitvec* **and** $P ; \Gamma \vdash i$
shows $\exists$ *bv*. *eval-v i v* (*SBitvec bv*)
⟨*proof*⟩

lemma *wfV-eval-pair*:
fixes *v*::*v*
assumes $P ; B ; \Gamma \vdash_{wf} v :$ *B-pair b1 b2* **and** $P ; \Gamma \vdash i$
shows $\exists$ *s1 s2*. *eval-v i v* (*SPair s1 s2*)
⟨*proof*⟩

lemma *wfV-eval-int*:
  fixes $v::v$
  assumes P ; B ; $\Gamma \vdash_{wf} v : B\text{-}int$ **and** P ; $\Gamma \vdash i$
  shows $\exists\, n.\ eval\text{-}v\ i\ v\ (SNum\ n)$
⟨*proof*⟩

Well sorted value with a well sorted valuation evaluates

lemma *wfI-wfV-eval-v*:
  fixes $v::v$ **and** $b::b$
  assumes Θ ; B ; $\Gamma \vdash_{wf} v : b$ **and** $wfI\ \Theta\ \Gamma\ i$
  shows $\exists\, s.\ i\ [\![\ v\ ]\!] \sim\ s \wedge \Theta \vdash s : b$
  ⟨*proof*⟩

lemma *wfI-wfCE-eval-e*:
  fixes $e::ce$ **and** $b::b$
  assumes $wfCE\ P\ B\ G\ e\ b$ **and** P ; $G \vdash i$
  shows $\exists\, s.\ i\ [\![\ e\ ]\!] \sim s \wedge P \vdash s : b$
  ⟨*proof*⟩

lemma *eval-e-exist*:
  fixes $\Gamma::\Gamma$ **and** $e::ce$
  assumes P ; $\Gamma \vdash i$ **and** P ; B ; $\Gamma \vdash_{wf} e : b$
  shows $\exists\, s.\ i\ [\![\ e\ ]\!] \sim s$
  ⟨*proof*⟩

lemma *eval-c-exist*:
  fixes $\Gamma::\Gamma$ **and** $c::c$
  assumes P ; $\Gamma \vdash i$ **and** P ; B ; $\Gamma \vdash_{wf} c$
  shows $\exists\, s.\ i\ [\![\ c\ ]\!] \sim s$
  ⟨*proof*⟩

lemma *eval-c-uniqueness*:
  fixes $c::c$
  assumes $i\ [\![\ c\ ]\!] \sim s$ **and** $i\ [\![\ c\ ]\!] \sim s'$
  shows $s{=}s'$
  ⟨*proof*⟩

lemma *wfI-wfC-eval-c*:
  fixes $c::c$
  assumes $wfC\ P\ B\ G\ c$ **and** P ; $G \vdash i$
  shows $\exists\, s.\ i\ [\![\ c\ ]\!] \sim s$
  ⟨*proof*⟩

11.3 Satisfiability

lemma *satis-reflI*:
  fixes $c::c$
  assumes $i \models ((x,\ b,\ c)\ \#_{\Gamma}\ G)$
  shows $i \models c$
  ⟨*proof*⟩

lemma *is-satis-mp*:
  fixes $c1::c$ **and** $c2::c$
  assumes $i \models (c1\ IMP\ c2)$ **and** $i \models c1$
  shows $i \models c2$
  ⟨*proof*⟩

lemma *is-satis-imp*:
  fixes $c1::c$ **and** $c2::c$
  assumes $i \models c1 \longrightarrow i \models c2$ **and** $i\ [\![\ c1\ ]\!] \sim b1$ **and** $i\ [\![\ c2\ ]\!] \sim b2$
  shows $i \models (c1\ IMP\ c2)$
⟨*proof*⟩

lemma *is-satis-iff*:
  $i \models G\ = (\forall\, x\ b\ c.\ (x,b,c) \in toSet\ G \longrightarrow i \models c)$
  ⟨*proof*⟩

lemma *is-satis-g-append*:
  $i \models (G1@G2) = (i \models\ G1 \wedge i \models\ G2)$
  ⟨*proof*⟩

11.4 Substitution for Evaluation

lemma *eval-v-i-upd*:
  fixes $v::v$
  assumes $atom\ x\ \sharp\ v$ **and** $i\ [\![\ v\ ]\!] \sim\ s'$
  shows $eval\text{-}v\ ((i\ (\ x \mapsto s)))\ v\ s'$
  ⟨*proof*⟩

lemma *eval-e-i-upd*:
  fixes $e::ce$
  assumes $i\ [\![\ e\ ]\!] \sim s'$ **and** $atom\ x\ \sharp\ e$
  shows $(i\ (\ x \mapsto s))\ [\![\ e\ ]\!] \sim s'$
  ⟨*proof*⟩

lemma *eval-c-i-upd*:
  fixes $c::c$
  assumes $i\ [\![\ c\ ]\!] \sim s'$ **and** $atom\ x\ \sharp\ c$
  shows $((i\ (\ x \mapsto s)))\ [\![\ c\ ]\!] \sim s'$
  ⟨*proof*⟩

lemma *subst-v-eval-v*[*simp*]:
  fixes $v::v$ **and** $v'::v$
  assumes $i\ [\![\ v\ ]\!] \sim\ s$ **and** $i\ [\![\ (v'[x::=v]_{vv})\ ]\!] \sim s'$
  shows $(i\ (\ x \mapsto s\ ))\ [\![\ v'\ ]\!] \sim s'$
  ⟨*proof*⟩

lemma *subst-e-eval-v*[*simp*]:
  fixes $y::x$ **and** $e::ce$ **and** $v::v$ **and** $e'::ce$
  assumes $i\ [\![\ e'\ ]\!] \sim\ s'$ **and** $e'=(e[y::=v]_{cev})$ **and** $i\ [\![\ v\ ]\!] \sim\ s$
  shows $(i\ (\ y \mapsto s\ ))\ [\![\ e\ ]\!] \sim s'$
  ⟨*proof*⟩

lemma *subst-c-eval-v*[*simp*]:
fixes $v::v$ **and** $c :: c$
assumes $i \; [\![\; v \;]\!] \sim \; s$ **and** $i \; [\![\; c[x::=v]_{cv} \;]\!] \sim \; s1$ **and**
$(i \; (\; x \mapsto s)) \; [\![\; c \;]\!] \sim \; s2$
shows $s1 \; = \; s2$
⟨*proof*⟩

lemma *wfI-upd*:
assumes *wfI* $\Theta \; \Gamma \; i$ **and** *wfRCV* $\Theta \; s \; b$ **and** *wfG* $\Theta \; B \; ((x, \; b, \; c) \; \#_\Gamma \; \Gamma)$
shows *wfI* $\Theta \; ((x, \; b, \; c) \; \#_\Gamma \; \Gamma) \; (i(x \mapsto s))$
⟨*proof*⟩

lemma *wfI-upd-full*:
fixes $v::v$
assumes *wfI* $\Theta \; G \; i$ **and** $G = \; ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ **and** *wfRCV* $\Theta \; s \; b$ **and** *wfG* $\Theta \; B \; (\Gamma'@((x,b,c)\#_\Gamma\Gamma))$
and $\Theta \; ; \; B \; ; \Gamma \vdash_{wf} \; v \; : \; b$
shows *wfI* $\Theta \; (\Gamma'@((x,b,c)\#_\Gamma\Gamma)) \; (i(x \mapsto s))$
⟨*proof*⟩

lemma *subst-c-satis*[*simp*]:
fixes $v::v$
assumes $i \; [\![\; v \;]\!] \sim \; s$ **and** *wfC* $\Theta \; B \; ((x,b,c')\#_\Gamma\Gamma) \; c$ **and** *wfI* $\Theta \; \Gamma \; i$ **and** $\Theta \; ; \; B \; ; \Gamma \vdash_{wf} \; v \; : \; b$
shows $i \models (c[x::=v]_{cv}) \longleftrightarrow (i \; (\; x \mapsto s)) \models c$
⟨*proof*⟩

Key theorem telling us we can replace a substitution with an update to the valuation

lemma *subst-c-satis-full*:
fixes $v::v$ **and** $\Gamma'::\Gamma$
assumes $i \; [\![\; v \;]\!] \sim \; s$ **and** *wfC* $\Theta \; B \; (\Gamma'@((x,b,c')\#_\Gamma\Gamma)) \; c$ **and** *wfI* $\Theta \; ((\Gamma'[x::=v]_{\Gamma v})@\Gamma) \; i$ **and** $\Theta \; ;$ $B \; ; \Gamma \vdash_{wf} \; v \; : \; b$
shows $i \models (c[x::=v]_{cv}) \longleftrightarrow (i \; (\; x \mapsto s)) \models c$
⟨*proof*⟩

11.5 Validity

lemma *validI*[*intro*]:
fixes $c::c$
assumes *wfC* $P \; B \; G \; c$ **and** $\forall \, i. \; P \; ; \; G \vdash i \wedge i \models G \longrightarrow i \models c$
shows $P \; ; \; B \; ; \; G \models c$
⟨*proof*⟩

lemma *valid-g-wf*:
fixes $c::c$ **and** $G::\Gamma$
assumes $P \; ; \; B \; ; \; G \models c$
shows $P \; ; \; B \vdash_{wf} \; G$
⟨*proof*⟩

lemma *valid-reflI* [*intro*]:
fixes $b::b$
assumes $P \; ; \; B \; ; \; ((x,b,c1)\#_\Gamma G) \vdash_{wf} \; c1$ **and** $c1 \; = \; c2$
shows $P \; ; \; B \; ; \; ((x,b,c1)\#_\Gamma G) \models c2$
⟨*proof*⟩

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** $supp\ c \subseteq atom\ `\ d \cup supp\ B$ **and** $i\ [\![\ c\ ]\!]\ \sim\ s$
shows $i'\ [\![\ c\ ]\!]\ \sim\ s$
⟨*proof*⟩

lemma *eval-v-restrict*:
fixes $c::v$ **and** $B::bv\ fset$
assumes $i = i'\ |`\ d$ **and** $supp\ c \subseteq atom\ `\ d \cup supp\ B$ **and** $i'\ [\![\ c\ ]\!]\ \sim\ s$
shows $i\ [\![\ c\ ]\!]\ \sim\ s$
⟨*proof*⟩

lemma *eval-e-weakening*:
fixes $e::ce$ **and** $B::bv\ fset$
assumes $i\ [\![\ e\ ]\!]\ \sim\ s$ **and** $i = i'\ |`\ d$ **and** $supp\ e \subseteq atom\ `\ d \cup supp\ B$
shows $i'\ [\![\ e\ ]\!]\ \sim\ s$
⟨*proof*⟩

lemma *eval-e-restrict* :
fixes $e::ce$ **and** $B::bv\ fset$
assumes $i'\ [\![\ e\ ]\!]\ \sim\ s$ **and** $i = i'\ |`\ d$ **and** $supp\ e \subseteq atom\ `\ d \cup supp\ B$
shows $i\ [\![\ e\ ]\!]\ \sim\ s$
⟨*proof*⟩

lemma *eval-c-i-weakening*:
fixes $c::c$ **and** $B::bv\ fset$
assumes $i\ [\![\ c\ ]\!]\ \sim\ s$ **and** $i = i'\ |`\ d$ **and** $supp\ c \subseteq atom\ `\ d \cup supp\ B$
shows $i'\ [\![\ c\ ]\!]\ \sim\ s$
⟨*proof*⟩

lemma *eval-c-i-restrict*:
fixes $c::c$ **and** $B::bv\ fset$
assumes $i'\ [\![\ c\ ]\!]\ \sim\ s$ **and** $i = i'\ |`\ d$ **and** $supp\ c \subseteq atom\ `\ d \cup supp\ B$
shows $i\ [\![\ c\ ]\!]\ \sim\ s$
⟨*proof*⟩

lemma *is-satis-i-weakening*:
fixes $c::c$ **and** $B::bv\ fset$
assumes $i = i'\ |`\ d$ **and** $supp\ c \subseteq atom\ `\ d\ \cup supp\ B$ **and** $i \models c$
shows $i' \models c$
⟨*proof*⟩

lemma *is-satis-i-restrict*:
fixes $c::c$ **and** $B::bv\ fset$
assumes $i = i'\ |`\ d$ **and** $supp\ c \subseteq atom\ `\ d\ \cup supp\ B$ **and** $i' \models c$
shows $i \models c$
⟨*proof*⟩

lemma *is-satis-g-restrict1* :

fixes $\Gamma'::\Gamma$ **and** $\Gamma::\Gamma$
assumes *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** $i \models \Gamma'$
shows $i \models \Gamma$
⟨*proof*⟩

lemma *is-satis-g-restrict2*:
fixes $\Gamma'::\Gamma$ **and** $\Gamma::\Gamma$
assumes $i \models \Gamma$ **and** $i' = i \mid` d$ **and** *atom-dom* $\Gamma \subseteq$ *atom* ` d **and** Θ ; $B \vdash_{wf} \Gamma$
shows $i' \models \Gamma$
⟨*proof*⟩

lemma *is-satis-g-restrict*:
fixes $\Gamma'::\Gamma$ **and** $\Gamma::\Gamma$
assumes *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** $i' \models \Gamma'$ **and** $i = i' \mid` (fst$ ` *toSet* $\Gamma)$ **and** Θ ; $B \vdash_{wf} \Gamma$
shows $i \models \Gamma$
⟨*proof*⟩

11.5.2 Updating valuation

lemma *is-satis-c-i-upd*:
fixes $c::c$
assumes *atom* $x \sharp c$ **and** $i \models c$
shows $((i (x \mapsto s))) \models c$
⟨*proof*⟩

lemma *is-satis-g-i-upd*:
fixes $G::\Gamma$
assumes *atom* $x \sharp G$ **and** $i \models G$
shows $((i (x \mapsto s))) \models G$
⟨*proof*⟩

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*⟩

lemma *is-satis-g-suffix*:
fixes $G::\Gamma$
assumes $i \models (G'@G)$
shows $i \models G$
⟨*proof*⟩

lemma *wfG-inside-valid2*:
fixes $x::x$ **and** $\Gamma::\Gamma$ **and** $c0::c$ **and** $c0'::c$
assumes *wfG* Θ B $(\Gamma'@((x,b0,c0')\#_{\Gamma}\Gamma))$ **and**
Θ ; B ; $\Gamma'@(x,b0,c0)\#_{\Gamma}\Gamma \models c0'$
shows *wfG* Θ B $(\Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma))$
⟨*proof*⟩

lemma *valid-trans*:
assumes Θ ; $\mathcal{B}$; $\Gamma \models c0[z::=v]_v$ **and** Θ ; $\mathcal{B}$; $(z,b,c0)\#_{\Gamma}\Gamma \models c1$ **and** *atom* $z \sharp \Gamma$ **and** *wfV* Θ $\mathcal{B}$ Γ v b
shows Θ ; $\mathcal{B}$; $\Gamma \models c1[z::=v]_v$

⟨*proof*⟩

lemma *valid-trans-full*:
assumes Θ ; $\mathcal{B}$; $((x,\ b,\ c1[z1::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c2[z2::=V\text{-}var\ x]_v$ **and**
Θ ; $\mathcal{B}$; $((x,\ b,\ c2[z2::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[z3::=V\text{-}var\ x]_v$
shows Θ ; $\mathcal{B}$; $((x,\ b,\ c1[z1::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[z3::=V\text{-}var\ x]_v$
⟨*proof*⟩

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 $i\ [\![\ c\ ]\!]\ {\sim}\ s$
⟨*proof*⟩

lemma *eval-e-weakening-x*:
fixes $c::ce$
assumes $i'\ [\![\ c\ ]\!]\ {\sim}\ s$ **and** $atom\ x\ \sharp\ c$ **and** $i = i'\ (x \mapsto s')$
shows $i\ [\![\ c\ ]\!]\ {\sim}\ s$
⟨*proof*⟩

lemma *eval-c-weakening-x*:
fixes $c::c$
assumes $i'\ [\![\ c\ ]\!]\ {\sim}\ s$ **and** $atom\ x\ \sharp\ c$ **and** $i = i'\ (x \mapsto s')$
shows $i\ [\![\ c\ ]\!]\ {\sim}\ s$
⟨*proof*⟩

lemma *is-satis-weakening-x*:
fixes $c::c$
assumes $i' \models c$ **and** $atom\ x\ \sharp\ c$ **and** $i = i'\ (x \mapsto s)$
shows $i \models c$
⟨*proof*⟩

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 $i \models G$
⟨*proof*⟩

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* ≠ *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* ⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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 ⊢ $i \approx i$
| *boxed-i-GConsI*: ⟦ *Some* $s = i\ x$; *boxed-b* $\Theta\ s\ b\ bv\ b'\ s'$; Θ ; Γ ; b' , $bv \vdash i \approx i'$ ⟧ $\Longrightarrow$ Θ ; $((x,b,c)\#_\Gamma\Gamma)$; b' , $bv \vdash i \approx (i'(x \mapsto s'))$
equivariance *boxed-i*
nominal-inductive *boxed-i* ⟨*proof*⟩

inductive-cases *boxed-i-elims*:
Θ ;*GNil* ; b , $bv \vdash i \approx i'$
Θ ; $((x,b,c)\#_\Gamma\Gamma)$; b' , $bv \vdash i \approx i'$

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* ∧ $T \vdash s : b1[bva::=bdc]_{bb}$ ∧ *atom bva* ♯ *tm*
⟨*proof*⟩

lemma *boxed-b-ex*:
assumes *wfRCV* $T\ s\ b[bv::=b']_{bb}$ **and** *wfTh* T
shows $\exists s'$. *boxed-b* $T\ s\ b\ bv\ b'\ s'$
⟨*proof*⟩

lemma *boxed-i-ex*:
assumes *wfI* $T\ \Gamma[bv::=b]_{\Gamma b}\ i$ **and** *wfTh* T
shows $\exists i'$. T ; Γ ; b , $bv \vdash i \approx i'$
⟨*proof*⟩

lemma *boxed-b-eq*:
assumes *boxed-b* Θ *s1 b bv b' s1'* **and** $\vdash_{wf}$ Θ
shows *wfTh* Θ $\Longrightarrow$ *boxed-b* Θ *s2 b bv b' s2'* $\Longrightarrow$ (*s1* = *s2*) = (*s1'* = *s2'*)
⟨*proof*⟩

lemma *bs-boxed-var*:
assumes *boxed-i* $\Theta\ \Gamma\ b'\ bv\ i\ i'$
shows *Some* (b,c) = *lookup* $\Gamma\ x$ $\Longrightarrow$ *Some* $s = i\ x$ $\Longrightarrow$ *Some* $s' = i'\ x$ $\Longrightarrow$ *boxed-b* $\Theta\ s\ b\ bv\ b'\ s'$
⟨*proof*⟩

lemma *eval-l-boxed-b*:
assumes ⟦ l ⟧ = s
shows *boxed-b* Θ s (*base-for-lit* l) $bv\ b'\ s$
⟨*proof*⟩

lemma *boxed-i-eval-v-boxed-b*:
fixes *v*::*v*
assumes *boxed-i* $\Theta\ \Gamma\ b'\ bv\ i\ i'$ **and** i ⟦ $v[bv::=b']_{vb}$ ⟧ ~ s **and** i' ⟦ v ⟧ ~ s' **and** *wfV* $\Theta\ B\ \Gamma\ v\ b$ **and** *wfI* Θ $\Gamma\ i'$
shows *boxed-b* $\Theta\ s\ b\ bv\ b'\ s'$

⟨*proof*⟩

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′*
⟨*proof*⟩

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′*
⟨*proof*⟩

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′*
⟨*proof*⟩

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*⟩

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*⟩

lemma *bs-boxed-wfi-aux*:
fixes *b*::*b* **and** *bv*::*bv* **and** Θ::Θ **and** *B*::$\mathcal{B}$
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′*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *is-satis-g-bs-boxed*:
fixes $G::\Gamma$
assumes $boxed\text{-}i\ \Theta\ G\ b\ bv\ i\ i'$ **and** $wfI\ \Theta\ G[bv::=b]_{\Gamma b}\ i$ **and** $wfI\ \Theta\ G\ i'$ **and** $wfG\ \Theta\ B\ G$
and $(i \models G[bv::=b]_{\Gamma b})$
shows $(i' \models G)$
⟨*proof*⟩

lemma *subst-b-valid*:
fixes $s::s$ **and** $b::b$
assumes Θ ; $\{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$ **and** Θ ; $\{|bv|\}$;$\Gamma \models c$
shows Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \models c[bv::=b]_{cb}$
⟨*proof*⟩

11.7 Expression Operator Lemmas

lemma *is-satis-len-imp*:
assumes $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (int\ (length\ v)))))$ (**is** *is-satis i ?c1*)
shows $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}len\ [V\text{-}lit\ (L\text{-}bitvec\ v)]^{ce})$
⟨*proof*⟩

lemma *is-satis-plus-imp*:
assumes $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (n1+n2))))$ (**is** *is-satis i ?c1*)
shows $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}op\ Plus\ ([V\text{-}lit\ (L\text{-}num\ n1)]^{ce})\ ([V\text{-}lit\ (L\text{-}num\ n2)]^{ce}))$
⟨*proof*⟩

lemma *is-satis-leq-imp*:
assumes $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}val\ (V\text{-}lit\ (if\ (n1 \le n2)\ then\ L\text{-}true\ else\ L\text{-}false)))$ (**is** *is-satis i ?c1*)
shows $i \models (CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}op\ LEq\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce})$
⟨*proof*⟩

lemma *eval-lit-inj*:
fixes $n1::l$ **and** $n2::l$
assumes $[\![\ n1\ ]\!] = s$ **and** $[\![\ n2\ ]\!] = s$
shows $n1=n2$
⟨*proof*⟩

lemma *eval-e-lit-inj*:
fixes $n1::l$ **and** $n2::l$
assumes $i\ [\![\ [\ [\ n1\ ]^v\ ]^{ce}\ ]\!] \sim s$ **and** $i\ [\![\ [\ [\ n2\ ]^v\ ]^{ce}\ ]\!] \sim s$
shows $n1=n2$
⟨*proof*⟩

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*⟩

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-val* (*V-var x*) == *e1*)$\#_\Gamma$ *GNil* $\models$ *CE-val* (*V-var x*) == *e2*
⟨*proof*⟩

lemma *valid-len*:
assumes $\vdash_{wf}$ Θ
shows Θ ; $\mathcal{B}$; (*x*, *B-int*, $[[x]^v]^{ce}$ == [[*L-num* (*int* (*length v*)) $]^v]^{ce}$) $\#_\Gamma$ *GNil* $\models$ $[[x]^v]^{ce}$ == *CE-len* [[*L-bitvec v* $]^v]^{ce}$ (**is** Θ ; $\mathcal{B}$; *?G* $\models$ *?c*)
⟨*proof*⟩

lemma *valid-arith-bop*:
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*) **and**
atom x ♯ Γ
shows Θ; $\mathcal{B}$; (*x*, *b*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*ll*)))) $\#_\Gamma$ Γ
$\models$ (*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* $\models$ *?c*)
⟨*proof*⟩

lemma *valid-eq-bop*:
assumes *wfG* Θ $\mathcal{B}$ Γ **and** *atom x* ♯ Γ **and** *base-for-lit l1* = *base-for-lit l2*
shows Θ; $\mathcal{B}$; (*x*, *B-bool*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*if l1* = *l2 then L-true else L-false*)))) $\#_\Gamma$ Γ
$\models$ (*CE-val* (*V-var x*) == *CE-op Eq* ([*V-lit* (*l1*)$]^{ce}$) ([*V-lit* (*l2*)$]^{ce}$)) (**is** Θ ; $\mathcal{B}$; *?G* $\models$ *?c*)
⟨*proof*⟩

lemma *valid-fst*:
fixes *x*::*x* **and** v_1::*v* **and** v_2::*v*
assumes *wfTh* Θ **and** *wfV* Θ $\mathcal{B}$ *GNil* (*V-pair* v_1 v_2) (*B-pair* b_1 b_2)
shows Θ ; $\mathcal{B}$; (*x*, b_1, $[[x]^v]^{ce}$ == $[v_1]^{ce}$) $\#_\Gamma$ *GNil* $\models$ $[[x]^v]^{ce}$ == $[\#1[[v_1,v_2]^v]^{ce}]^{ce}$
⟨*proof*⟩

lemma *valid-snd*:
fixes *x*::*x* **and** v_1::*v* **and** v_2::*v*
assumes *wfTh* Θ **and** *wfV* Θ $\mathcal{B}$ *GNil* (*V-pair* v_1 v_2) (*B-pair* b_1 b_2)
shows Θ ; $\mathcal{B}$; (*x*, b_2, $[[x]^v]^{ce}$ == $[v_2]^{ce}$) $\#_\Gamma$ *GNil* $\models$ $[[x]^v]^{ce}$ == $[\#2[[v_1,v_2]^v]^{ce}]^{ce}$
⟨*proof*⟩

lemma *valid-concat*:
fixes *v1*::*bit list* **and** *v2*::*bit list*
assumes $\vdash_{wf}$ Π
shows Π ; $\mathcal{B}$; (*x*, *B-bitvec*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-bitvec* (*v1*@ *v2*))))) $\#_\Gamma$ *GNil* $\models$
(*CE-val* (*V-var x*) == *CE-concat* ([*V-lit* (*L-bitvec v1*)$]^{ce}$) ([*V-lit* (*L-bitvec v2*)$]^{ce}$))
⟨*proof*⟩

lemma *valid-ce-eq*:
fixes *ce*::*ce*
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b*
shows ‹Θ ; $\mathcal{B}$; Γ $\models$ *ce* == *ce* ›
⟨*proof*⟩

lemma *valid-eq-imp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes Θ ; $\mathcal{B}$; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1 IMP c2*
shows Θ ; $\mathcal{B}$; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\models$ *c1 IMP c2*
⟨*proof*⟩

lemma *valid-range*:
assumes $0 \leq n \wedge n \leq m$ **and** $\vdash_{wf}$ Θ
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*⟩

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* $\sharp$ Γ
shows Θ ; {||} ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_\Gamma$ Γ $\models$
(*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* $\models$ *?c1 AND ?c2*)
⟨*proof*⟩

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* $\models$
(*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* $\models$ *?c1 AND ?c2*)
shows $0 \leq n \wedge n \leq$ *int* (*length v*)
⟨*proof*⟩

lemma *wfI-cons*:
fixes *i*::*valuation* **and** Γ::Γ
assumes $i' \models \Gamma$ **and** Θ ; $\Gamma \vdash i'$ **and** $i = i'$ ($x \mapsto s$) **and** Θ $\vdash s : b$ **and** *atom x* $\sharp$ Γ
shows Θ ; (*x*,*b*,*c*) $\#_\Gamma$ $\Gamma \vdash i$
⟨*proof*⟩

lemma *valid-range-length-inv*:
fixes Γ::Γ
assumes Θ ; *B* $\vdash_{wf}$ Γ **and** *atom x* $\sharp$ Γ **and** $\exists\, i.\ i \models \Gamma \wedge \Theta$; $\Gamma \vdash i$
and Θ ; *B* ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_\Gamma$ Γ $\models$
(*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 $0 \leq n \wedge n \leq int\ (length\ v)$
⟨*proof*⟩

lemma *eval-c-conj2I*[*intro*]:
assumes *i* ⟦ *c1* ⟧ ~ *True* **and** *i* ⟦ *c2* ⟧ ~ *True*
shows *i* ⟦ (*C-conj c1 c2*) ⟧ ~ *True*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *is-satis-eq*:
assumes *wfI* Θ *G i* **and** *wfCE* Θ ℬ *G e b*
shows *is-satis i* (*e* == *e*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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* ♯ ⦃ *z* : *b* | *c* ⦄ **and** *atom z* ♯ *x*
shows *atom x* ♯ *c*
⟨*proof*⟩

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* | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP c-of t z* ⦄) **and** *atom z* ♯ (Γ,*t*)
shows *wfT* Θ $\mathcal{B}$ Γ *t*
⟨*proof*⟩

lemma *fresh-u-replace-true*:
fixes *bv*::*bv* **and** Γ::Γ
assumes *atom bv* ♯ Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ
shows *atom bv* ♯ Γ′ @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ
⟨*proof*⟩

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*∗*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 Θ; $\mathcal{B}$; *G* $\vdash_{wf}$ *v* : *b′* ⟹ *G* = Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ ⟹ Θ ; $\mathcal{B}$; Γ′ @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *v* : *b′* **and**
Θ; $\mathcal{B}$; *G* $\vdash_{wf}$ *c″* ⟹ *G* = Γ′ @(*x*, *b*, *c*) $\#_\Gamma$ Γ ⟹ Θ ; $\mathcal{B}$; Γ′ @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *c″* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ *G* ⟹ *G* = Γ′ @(*x*, *b*, *c*) $\#_\Gamma$ Γ ⟹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) **and**
Θ; $\mathcal{B}$; *G* $\vdash_{wf}$ *τ* ⟹ *G* = Γ′ @(*x*, *b*, *c*) $\#_\Gamma$ Γ ⟹ Θ ; $\mathcal{B}$; Γ′ @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ *τ* **and**
Θ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* ⟹ *True* **and**
$\vdash_{wf}$ *P* ⟹ *True* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* ⟹ *True* **and**

Θ ; $\mathcal{B}$; $G \vdash_{wf}$ *ce* : $b' \Longrightarrow$ $G =$ Γ' @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; $\mathcal{B}$; Γ' @ $((x, b, TRUE)$ $\#_\Gamma$ $\Gamma) \vdash_{wf}$ *ce* : b' **and**
Θ $\vdash_{wf}$ *td* $\Longrightarrow$ *True*
⟨*proof*⟩

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*∗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 Θ ; Φ ; $\mathcal{B}$; G ; D $\vdash_{wf}$ *e* : $b' \Longrightarrow G =$ Γ' @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ' @ $((x, b, TRUE)$ $\#_\Gamma$ $\Gamma)$; $D \vdash_{wf}$ *e* : b' **and**
Θ ; Φ ; $\mathcal{B}$; G ; $\Delta \vdash_{wf}$ *s* : $b' \Longrightarrow$ $G =$ Γ' @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ' @ $((x, b, TRUE)$ $\#_\Gamma$ $\Gamma)$; $\Delta \vdash_{wf}$ *s* : b' **and**
Θ ; Φ ; $\mathcal{B}$; G ; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ *cs* : $b' \Longrightarrow$ $G =$ Γ' @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ' @ $((x, b, TRUE)$ $\#_\Gamma$ $\Gamma)$; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ *cs* : b' **and**
Θ ; Φ ; $\mathcal{B}$; G ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : $b' \Longrightarrow$ $G =$ Γ' @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ' @ $((x, b, TRUE)$ $\#_\Gamma$ $\Gamma)$; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : b' **and**

$\Theta \vdash_{wf} \Phi \Longrightarrow$ *True* **and**
Θ; $\mathcal{B}$; G $\vdash_{wf}$ $\Delta \Longrightarrow$ $G =$ Γ' @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; $\mathcal{B}$; Γ' @ $((x, b, TRUE)$ $\#_\Gamma$ $\Gamma) \vdash_{wf} \Delta$ **and**

Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\Longrightarrow$ *True*
⟨*proof*⟩

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*⟩

lemma *subtype-reflI*:
assumes $\{\!|$ *z1* : *b* | *c1* $|\!\}$ = $\{\!|$ *z2* : *b* | *c2* $|\!\}$ **and** *wf1*: Θ; $\mathcal{B}$;$\Gamma \vdash_{wf}$ ($\{\!|$ *z1* : *b* | *c1* $|\!\}$)
shows Θ; $\mathcal{B}$; $\Gamma \vdash$ ($\{\!|$ *z1* : *b* | *c1* $|\!\}$) $\lesssim$ ($\{\!|$ *z2* : *b* | *c2* $|\!\}$)
⟨*proof*⟩

nominal-function *base-eq* :: $\Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow$ *bool* **where**
base-eq - $\{\!|$ *z1* : *b1*| *c1* $|\!\}$ $\{\!|$ *z2* : *b2* | *c2* $|\!\}$ = (*b1* = *b2*)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *subtype-wfT*:
fixes *t1*::τ **and** *t2*::τ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash$ *t1* $\lesssim$ *t2*
shows Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *t1* $\wedge$ Θ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *t2*
⟨*proof*⟩

lemma *subtype-eq-base*:
  assumes $\Theta; \mathcal{B}; \Gamma \vdash (\{\!\mid z1 : b1 \mid c1 \mid\!\}) \lesssim (\{\!\mid z2 : b2 \mid c2 \mid\!\})$
  shows $b1=b2$
  ⟨*proof*⟩

lemma *subtype-eq-base2*:
  assumes $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$
  shows $b\text{-}of\ t1 = b\text{-}of\ t2$
  ⟨*proof*⟩

lemma *subtype-wf*:
  fixes $\tau 1::\tau$ **and** $\tau 2::\tau$
  assumes $\Theta; \mathcal{B}; \Gamma \vdash \tau 1 \lesssim \tau 2$
  shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau 1 \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau 2$
  ⟨*proof*⟩

lemma *subtype-g-wf*:
  fixes $\tau 1::\tau$ **and** $\tau 2::\tau$ **and** $\Gamma::\Gamma$
  assumes $\Theta; \mathcal{B}; \Gamma \vdash \tau 1 \lesssim \tau 2$
  shows $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$
  ⟨*proof*⟩

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 $\Theta; \mathcal{B}; (z, b, c) \#_\Gamma \Gamma \models c'$ **and** $atom\ z \sharp \Gamma$ **and** $atom\ x \sharp (z, c, z, c', \Gamma)$
  shows $\Theta; \mathcal{B}; (x, b, (z \leftrightarrow x) \cdot c) \#_\Gamma \Gamma \models (z \leftrightarrow x) \cdot c'$
⟨*proof*⟩

lemma *valid-wf-all*:
  assumes $\Theta; \mathcal{B}; (z0,b,c0)\#_\Gamma G \models c$
  shows $wfG\ \Theta\ \mathcal{B}\ G$ **and** $wfC\ \Theta\ \mathcal{B}\ ((z0,b,c0)\#_\Gamma G)\ c$ **and** $atom\ z0 \sharp G$
  ⟨*proof*⟩

lemma *valid-wfT*:
  fixes $z::x$
  assumes $\Theta; \mathcal{B}; (z0,b,c0[z::=V\text{-}var\ z0]_v)\#_\Gamma G \models c[z::=V\text{-}var\ z0]_v$ **and** $atom\ z0 \sharp (\Theta, \mathcal{B}, G,c,c0)$
  shows $\Theta; \mathcal{B}; G \vdash_{wf} \{\!\mid z : b \mid c0 \mid\!\}$ **and** $\Theta; \mathcal{B}; G \vdash_{wf} \{\!\mid z : b \mid c \mid\!\}$
⟨*proof*⟩

lemma *valid-flip*:
  fixes $c::c$ **and** $z::x$ **and** $z0::x$ **and** $xx2::x$
  assumes $\Theta; \mathcal{B}; (xx2, b, c0[z0::=V\text{-}var\ xx2]_v) \#_\Gamma \Gamma \models c[z::=V\text{-}var\ xx2]_v$ **and**
    $atom\ xx2 \sharp (c0,\Gamma,c,z)$ **and** $atom\ z0 \sharp (\Gamma,c,z)$
  shows $\Theta; \mathcal{B}; (z0, b, c0) \#_\Gamma \Gamma \models c[z::=V\text{-}var\ z0]_v$
⟨*proof*⟩

lemma *subtype-valid*:
  assumes $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$ **and** $atom\ y \sharp \Gamma$ **and** $t1 = \{\!\mid z1 : b \mid c1 \mid\!\}$ **and** $t2 = \{\!\mid z2 : b \mid c2 \mid\!\}$
  shows $\Theta; \mathcal{B}; ((y, b, c1[z1::=V\text{-}var\ y]_v) \#_\Gamma \Gamma) \models c2[z2::=V\text{-}var\ y]_v$
  ⟨*proof*⟩

lemma *subtype-valid-simple*:
assumes Θ; $\mathcal{B}$; $\Gamma \vdash$ *t1* $\lesssim$ *t2* **and** *atom z* $\sharp$ Γ **and** *t1* = $\{\!\!\{$ *z* : *b* | *c1* $\}\!\!\}$ **and** *t2* = $\{\!\!\{$ *z* : *b* | *c2* $\}\!\!\}$
shows Θ; $\mathcal{B}$; ((*z*, *b*, *c1*) $\#_\Gamma$ Γ) $\models$ *c2*
⟨*proof*⟩

lemma *obtain-for-t-with-fresh*:
assumes *atom x* $\sharp$ *t*
shows $\exists$ *b c*. *t* = $\{\!\!\{$ *x* : *b* | *c* $\}\!\!\}$
⟨*proof*⟩

lemma *subtype-trans*:
assumes Θ; $\mathcal{B}$; $\Gamma \vdash$ $\tau 1$ $\lesssim$ $\tau 2$ **and** Θ; $\mathcal{B}$; $\Gamma \vdash$ $\tau 2$ $\lesssim$ $\tau 3$
shows Θ; $\mathcal{B}$; $\Gamma \vdash$ $\tau 1$ $\lesssim$ $\tau 3$
⟨*proof*⟩

lemma *subtype-eq-e*:
assumes $\forall$ *i s1 s2 G*. *wfG P* $\mathcal{B}$ *G* $\wedge$ *wfI P G i* $\wedge$ *eval-e i e1 s1* $\wedge$ *eval-e i e2 s2* $\longrightarrow$ *s1* = *s2* **and** *atom z1* $\sharp$ *e1* **and** *atom z2* $\sharp$ *e2* **and** *atom z1* $\sharp$ Γ **and** *atom z2* $\sharp$ Γ
and *wfCE P* $\mathcal{B}$ Γ *e1 b* **and** *wfCE P* $\mathcal{B}$ Γ *e2 b*
shows *P*; $\mathcal{B}$; $\Gamma \vdash$ $\{\!\!\{$ *z1* : *b* | *CE-val* (*V-var z1*) == *e1* $\}\!\!\}$ $\lesssim$ ($\{\!\!\{$ *z2* : *b* | *CE-val* (*V-var z2*) == *e2* $\}\!\!\}$)
⟨*proof*⟩

lemma *subtype-eq-e-nil*:
assumes $\forall$ *i s1 s2 G*. *wfG P* $\mathcal{B}$ *G* $\wedge$ *wfI P G i* $\wedge$ *eval-e i e1 s1* $\wedge$ *eval-e i e2 s2* $\longrightarrow$ *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* $\sharp$ *GNil* **and** *atom z2* $\sharp$ *GNil*
shows *P*; $\mathcal{B}$; *GNil* $\vdash$ $\{\!\!\{$ *z1* : *b* | *CE-val* (*V-var z1*) == *e1* $\}\!\!\}$ $\lesssim$ ($\{\!\!\{$ *z2* : *b* | *CE-val* (*V-var z2*) == *e2* $\}\!\!\}$)
⟨*proof*⟩

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* $[V\text{-}pair\ v_1\ v_2]^{ce}$) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-val* v_2) *b2* **and** *atom z1* $\sharp$ *GNil* **and** *atom z2* $\sharp$ *GNil*
shows *P*; $\mathcal{B}$; *GNil* $\vdash$ ($\{\!\!\{$ *z1* : *b* | *CE-val* (*V-var z1*) == *CE-val* v_1 $\}\!\!\}$) $\lesssim$ ($\{\!\!\{$ *z2* : *b* | *CE-val* (*V-var z2*) == *CE-fst* $[V\text{-}pair\ v_1\ v_2]^{ce}$ $\}\!\!\}$)
⟨*proof*⟩

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* $[(V\text{-}pair\ v_1\ v_2)]^{ce}$) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-val* v_1) *b2* **and** *atom z1* $\sharp$ *GNil* **and** *atom z2* $\sharp$ *GNil*
shows *P*; $\mathcal{B}$; *GNil* $\vdash$ ($\{\!\!\{$ *z1* : *b* | *CE-val* (*V-var z1*) == *CE-val* v_2 $\}\!\!\}$) $\lesssim$ ($\{\!\!\{$ *z2* : *b* | *CE-val* (*V-var z2*) == *CE-snd* $[(V\text{-}pair\ v_1\ v_2)]^{ce}$ $\}\!\!\}$)
⟨*proof*⟩

lemma *subtype-gnil-fst*:
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ $[\#1[[v_1,v_2]^v]^{ce}]^{ce}$: *b*
shows Θ ; {||} ; *GNil* $\vdash$ ($\{\!\!\{$ z_1 : *b* | $[[z_1]^v]^{ce}$ == $[v_1]^{ce}$ $\}\!\!\}$) $\lesssim$
($\{\!\!\{$ z_2 : *b* | $[[z_2]^v]^{ce}$ == $[\#1[[v_1, v_2]^v]^{ce}]^{ce}$ $\}\!\!\}$)

⟨*proof*⟩

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*⟩

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*⟩

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-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-var z1*]$_v$ ⦄) ≲ (⦃ *z* : *b* | *c* ⦄)
⟨*proof*⟩

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-var z1*]$_v$ ⦄) **and**
wfT P ℬ Γ (⦃ *z2* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* *c'*[*z'*::=*V-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-var z1*]$_v$ ⦄ ≲ ⦃ *z2* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* *c'*[*z'*::=*V-var z2*]$_v$ ⦄
⟨*proof*⟩

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-lit* (*L-bitvec v1*))]^{ce} [(*V-lit* (*L-bitvec v2*))]^{ce}) *s*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *subtype-len*:
assumes $\vdash_{wf}$ Θ
shows Θ; 𝓑; *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} ⦄ (**is** Θ; 𝓑; *GNil* ⊢ *?t1* ≲ *?t2*)
⟨*proof*⟩

lemma *subtype-base-fresh*:
assumes Θ; 𝓑; Γ $\vdash_{wf}$ ⦃ *z* : *b* | *c* ⦄ **and** Θ; 𝓑; Γ $\vdash_{wf}$ ⦃ *z* : *b* | *c′* ⦄ **and**
atom z ♯ Γ **and** Θ; 𝓑; (*z*, *b*, *c*) $\#_\Gamma$ Γ ⊨ *c′*
shows Θ; 𝓑; Γ ⊢ ⦃ *z* : *b* | *c* ⦄ ≲ ⦃ *z* : *b* | *c′* ⦄
⟨*proof*⟩

lemma *subtype-bop-arith*:
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*)
shows Θ; 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; 𝓑; Γ ⊢ *?T1* ≲ *?T2*)
⟨*proof*⟩

lemma *subtype-bop-eq*:
assumes *wfG* Θ 𝓑 Γ **and** *base-for-lit l1* = *base-for-lit l2*
shows Θ; 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*))) ⦄) ≲
⦃ *z* : *B-bool* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op Eq* [(*V-lit l1*)]^{ce} [(*V-lit l2*)]^{ce}) ⦄ (**is** Θ; 𝓑; Γ ⊢ *?T1* ≲ *?T2*)
⟨*proof*⟩

lemma *subtype-top*:
assumes *wfT* Θ 𝓑 *G* (⦃ *z* : *b* | *c* ⦄)
shows Θ; 𝓑; *G* ⊢ (⦃ *z* : *b* | *c* ⦄) ≲ (⦃ *z* : *b* | *TRUE* ⦄)
⟨*proof*⟩

lemma *if-simp*:
(*if x* = *x then e1 else e2*) = *e1*
⟨*proof*⟩

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*⟩

lemma *subtype-range*:
fixes *n*::*int* **and** Γ::Γ
assumes $0 \leq n \wedge n \leq$ *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*⟩

lemma *check-num-range*:
assumes $0 \leq n \wedge n \leq$ *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* $]^v\ ]^{ce}$ ⦄
⟨*proof*⟩

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*)$]^{ce}$ ⦄)
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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* ⦄))
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

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*⟩

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*)) ⦄)
⟨*proof*⟩

lemma *infer-l-form4* [*simp*]:
  fixes Γ::Γ
  assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma$
  shows $\exists z. \vdash l \Rightarrow (\{\!\!| z : \textit{base-for-lit } l \mid \textit{C-eq } (\textit{CE-val } (\textit{V-var } z)) (\textit{CE-val } (\textit{V-lit } l)) |\!\!\})$
  ⟨*proof*⟩

lemma *infer-v-unit-form*:
  fixes *v*::*v*
  assumes P ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow (\{\!\!| z1 : \textit{B-unit} \mid c1 |\!\!\})$ **and** $\textit{supp } v = \{\}$
  shows $v = \textit{V-lit L-unit}$
  ⟨*proof*⟩

lemma *base-for-lit-wf*:
  assumes $\vdash_{wf} \Theta$
  shows Θ ; $\mathcal{B} \vdash_{wf} \textit{base-for-lit } l$
  ⟨*proof*⟩

lemma *infer-l-t-wf*:
  fixes Γ::Γ
  assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma \wedge \textit{atom } z \sharp \Gamma$
  shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!\!| z : \textit{base-for-lit } l \mid \textit{C-eq } (\textit{CE-val } (\textit{V-var } z)) (\textit{CE-val } (\textit{V-lit } l)) |\!\!\}$
⟨*proof*⟩

lemma *infer-l-wf*:
  fixes *l*::*l* **and** Γ::Γ **and** τ::τ **and** Θ::Θ
  assumes $\vdash l \Rightarrow \tau$ **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma$
  shows $\vdash_{wf} \Theta$ **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$
⟨*proof*⟩

lemma *infer-l-uniqueness*:
  fixes *l*::*l*
  assumes $\vdash l \Rightarrow \tau$ **and** $\vdash l \Rightarrow \tau'$
  shows $\tau = \tau'$
  ⟨*proof*⟩

12.4 Values

lemma *type-v-eq*:
  assumes $\{\!\!| z1 : b1 \mid c1 |\!\!\} = \{\!\!| z : b \mid \textit{C-eq } (\textit{CE-val } (\textit{V-var } z)) (\textit{CE-val } (\textit{V-var } x)) |\!\!\}$ **and** $\textit{atom } z \sharp x$
  shows $b = b1$ **and** $c1 = \textit{C-eq } (\textit{CE-val } (\textit{V-var } z1)) (\textit{CE-val } (\textit{V-var } x))$
  ⟨*proof*⟩

lemma *infer-var2* [*elim*]:
  assumes P; $\mathcal{B}$; $G \vdash \textit{V-var } x \Rightarrow \tau$
  shows $\exists b\ c. \textit{ Some } (b,c) = \textit{lookup } G\ x$
  ⟨*proof*⟩

lemma *infer-var3* [*elim*]:
  assumes $\Theta; \mathcal{B}; \Gamma \vdash \textit{V-var } x \Rightarrow \tau$
  shows $\exists z\ b\ c. \textit{ Some } (b,c) = \textit{lookup } \Gamma\ x \wedge \tau = (\{\!\!| z : b \mid \textit{C-eq } (\textit{CE-val } (\textit{V-var } z)) (\textit{CE-val } (\textit{V-var } x))$
$|\!\!\}) \wedge \textit{atom } z \sharp x \wedge \textit{atom } z \sharp (\Theta, \mathcal{B}, \Gamma)$
  ⟨*proof*⟩

lemma *infer-bool-options2*:
fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\!\{ z : b \mid c \}\!\!\}$ **and** $supp\ v = \{\} \wedge b = B\text{-}bool$
shows $v = V\text{-}lit\ L\text{-}true \vee (v = (V\text{-}lit\ L\text{-}false))$
⟨*proof*⟩

lemma *infer-bool-options*:
fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\!\{ z : B\text{-}bool \mid c \}\!\!\}$ **and** $supp\ v = \{\}$
shows $v = V\text{-}lit\ L\text{-}true \vee (v = (V\text{-}lit\ L\text{-}false))$
⟨*proof*⟩

lemma *infer-int2*:
fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\!\{ z : b \mid c \}\!\!\}$
shows $supp\ v = \{\} \wedge b = B\text{-}int \longrightarrow (\exists n.\ v = V\text{-}lit\ (L\text{-}num\ n))$
⟨*proof*⟩

lemma *infer-bitvec*:
fixes $\Theta::\Theta$ **and** $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\!\{ z' : B\text{-}bitvec \mid c' \}\!\!\}$ **and** $supp\ v = \{\}$
shows $\exists bv.\ v = V\text{-}lit\ (L\text{-}bitvec\ bv)$
⟨*proof*⟩

lemma *infer-int*:
assumes $infer\text{-}v\ \Theta\ \mathcal{B}\ \Gamma\ v\ (\{\!\!\{ z : B\text{-}int \mid c \}\!\!\})$ **and** $supp\ v= \{\}$
shows $\exists n.\ V\text{-}lit\ (L\text{-}num\ n) = v$
⟨*proof*⟩

lemma *infer-lit*:
assumes $infer\text{-}v\ \Theta\ \mathcal{B}\ \Gamma\ v\ (\{\!\!\{ z : b \mid c \}\!\!\})$ **and** $supp\ v= \{\}$ **and** $b \in \{ B\text{-}bool , B\text{-}int , B\text{-}unit \}$
shows $\exists l.\ V\text{-}lit\ l = v$
⟨*proof*⟩

lemma *infer-v-form*[*simp*]:
fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$
shows $\exists z\ b.\ \tau = (\{\!\!\{ z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)\}\!\!\}) \wedge atom\ z \sharp v \wedge atom\ z \sharp (\Theta, \mathcal{B}, \Gamma)$
⟨*proof*⟩

lemma *infer-v-form2*:
fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow (\{\!\!\{ z : b \mid c \}\!\!\})$ **and** $atom\ z \sharp v$
shows $c = C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)$
⟨*proof*⟩

lemma *infer-v-form3*:
fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ **and** $atom\ z \sharp (v,\Gamma)$
shows $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{\!\!\{ z : b\text{-}of\ \tau \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)\}\!\!\}$
⟨*proof*⟩

lemma *infer-v-form4*:
fixes $v::v$
assumes $\Theta;\ \mathcal{B};\ \Gamma \vdash v \Rightarrow \tau$ **and** $atom\ z \sharp (v,\Gamma)$ **and** $b = b\text{-}of\ \tau$
shows $\Theta;\ \mathcal{B};\ \Gamma \vdash v \Rightarrow \{\!\mid z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)\mid\!\}$
⟨*proof*⟩

lemma *infer-v-v-wf*:
fixes $v::v$
shows $\Theta;\ \mathcal{B}\ ;\ G \vdash v \Rightarrow \tau \Longrightarrow \Theta;\ \mathcal{B};\ G \vdash_{wf} v : (b\text{-}of\ \tau)$
⟨*proof*⟩

lemma *infer-v-t-form-wf*:
assumes $wfB\ \Theta\ \mathcal{B}\ b$ **and** $wfV\ \Theta\ \mathcal{B}\ \Gamma\ v\ b$ **and** $atom\ z \sharp \Gamma$
shows $wfT\ \Theta\ \mathcal{B}\ \Gamma\ \{\!\mid z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)\mid\!\}$
⟨*proof*⟩

lemma *infer-v-t-wf*:
fixes $v::v$
assumes $\Theta;\ \mathcal{B};\ G \vdash v \Rightarrow \tau$
shows $wfT\ \Theta\ \mathcal{B}\ G\ \tau \wedge wfB\ \Theta\ \mathcal{B}\ (b\text{-}of\ \tau)$
⟨*proof*⟩

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*⟩

lemma *check-bool-options*:
assumes $\Theta;\ \mathcal{B};\ \Gamma \vdash v \Leftarrow \{\!\mid z : B\text{-}bool \mid TRUE \mid\!\}$ **and** $supp\ v = \{\}$
shows $v = V\text{-}lit\ L\text{-}true \vee v = V\text{-}lit\ L\text{-}false$
⟨*proof*⟩

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*⟩

lemma *infer-v-form-fresh*:
fixes $v::v$ **and** $t::'a::fs$
assumes $\Theta;\ \mathcal{B};\ \Gamma \vdash v \Rightarrow \tau$
shows $\exists z\ b.\ \tau = \{\!\mid z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v)\mid\!\} \wedge atom\ z \sharp (t,v)$
⟨*proof*⟩

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 $b\text{-}of\ \tau = B\text{-}app\ s\ b$
⟨*proof*⟩

lemma *lookup-in-rig-b*:
assumes *Some* (*b2*, *c2*) = *lookup* ($\Gamma[x \longmapsto c']$) *x'* **and**
Some (*b1*, *c1*) = *lookup* Γ *x'*
shows *b1* = *b2*
⟨*proof*⟩

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 $\tau = \tau'$
⟨*proof*⟩

lemma *infer-v-uniqueness*:
assumes *infer-v P B G v τ* **and** *infer-v P B G v τ'*
shows $\tau = \tau'$
⟨*proof*⟩

lemma *infer-v-tid-form*:
fixes *v*::*v*
assumes Θ ; *B* ; Γ ⊢ *v* ⇒ ⦃ *z* : *B-id tid* | *c* ⦄ **and** *AF-typedef tid dclist* ∈ *set* Θ **and** *supp v* = {}
shows ∃ *dc v' t*. *v* = *V-cons tid dc v'* ∧ (*dc* , *t*) ∈ *set dclist*
⟨*proof*⟩

lemma *check-v-tid-form*:
assumes Θ ; *B* ; Γ ⊢ *v* ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄ **and** *AF-typedef tid dclist* ∈ *set* Θ **and** *supp v* = {}
shows ∃ *dc v' t*. *v* = *V-cons tid dc v'* ∧ (*dc* , *t*) ∈ *set dclist*
⟨*proof*⟩

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 ⊢ $[$ *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* $]^{v}$ $]^{ce}$) ⦄
⟨*proof*⟩

lemma *check-int*:
assumes *check-v* Θ $\mathcal{B}$ Γ *v* (⦃ *z* : *B-int* | *c* ⦄) **and** *supp v* = {}
shows ∃ *n*. *V-lit* (*L-num n*) = *v*
⟨*proof*⟩

definition *sble* :: Θ ⇒ Γ ⇒ *bool* **where**
sble Θ Γ = (∃ *i*. *i* ⊨ Γ ∧ Θ ; Γ ⊢ *i*)

lemma *check-v-range*:
assumes Θ ; *B* ; Γ ⊢ *v2* ⇐ ⦃ *z* : *B-int* | [*leq* [[*L-num 0* $]^{v}$ $]^{ce}$ [[*z* $]^{v}$ $]^{ce}$ $]^{ce}$ == [[*L-true* $]^{v}$ $]^{ce}$ *AND*
[*leq* [[*z* $]^{v}$ $]^{ce}$ [| [*v1* $]^{ce}$ |$]^{ce}$ $]^{ce}$ == [[*L-true* $]^{v}$ $]^{ce}$ ⦄
(**is** Θ ; *?B* ; Γ ⊢ *v2* ⇐ ⦃ *z* : *B-int* | *?c1* ⦄)
and *v1* = *V-lit* (*L-bitvec bv*) ∧ *v2* = *V-lit* (*L-num n*) **and** *atom z* ♯ Γ **and** *sble* Θ Γ
shows $0 \leq n \wedge n \leq$ *int* (*length bv*)

⟨*proof*⟩

12.5 Expressions

lemma *infer-e-plus*[*elim*]:
fixes *v1*::*v* **and** *v2*::*v*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *AE-op Plus v1 v2* ⇒ τ
shows ∃ *z* . (⦃ *z* : *B-int* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$) ⦄ = τ)
⟨*proof*⟩

lemma *infer-e-leq*[*elim*]:
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *AE-op LEq v1 v2* ⇒ τ
shows ∃ *z* . (⦃ *z* : *B-bool* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$) ⦄ = τ)
⟨*proof*⟩

lemma *infer-e-eq*[*elim*]:
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *AE-op Eq v1 v2* ⇒ τ
shows ∃ *z* . (⦃ *z* : *B-bool* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op Eq* $[v1]^{ce}$ $[v2]^{ce}$) ⦄ = τ)
⟨*proof*⟩

lemma *infer-e-e-wf*:
fixes *e*::*e*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *e* ⇒ τ
shows Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b-of* τ
⟨*proof*⟩

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
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

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* ⟨*proof*⟩

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 $\vdash$ [*#1*[*v1* , *v2*]^v]^e $\Rightarrow \tau$
shows $\exists\tau'$. Θ ; Φ ; {||} ; *GNil* ; Δ $\vdash$ [*v1*]^e $\Rightarrow \tau' \wedge$
Θ ; {||} ; *GNil* $\vdash \tau' \lesssim \tau$
⟨*proof*⟩

lemma *infer-e-snd-pair*:
assumes Θ ; Φ ; {||} ; *GNil* ; Δ $\vdash$ *AE-snd* (*V-pair v1 v2*) $\Rightarrow \tau$
shows $\exists\tau'$. Θ ; Φ ; {||} ; *GNil* ; Δ $\vdash$ *AE-val v2* $\Rightarrow \tau' \wedge \Theta$; {||} ; *GNil* $\vdash \tau' \lesssim \tau$
⟨*proof*⟩

12.6 Statements

lemma *check-s-v-unit*:
assumes Θ; $\mathcal{B}$; $\Gamma \vdash$ ({| *z* : *B-unit* | *TRUE* |}) $\lesssim \tau$ **and** *wfD* Θ $\mathcal{B}$ Γ Δ **and** *wfPhi* Θ Φ
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-val* (*V-lit L-unit*) $\Leftarrow \tau$
⟨*proof*⟩

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 ; $\Delta \vdash s \Leftarrow \tau$ $\Longrightarrow$ Θ ; *B* $\vdash_{wf}$ Γ $\wedge$ *wfTh* Θ $\wedge$ *wfD* Θ *B* Γ Δ $\wedge$ *wfT* Θ *B* Γ τ $\wedge$ *wfPhi* Θ Φ **and**
check-branch-s Θ Φ *B* Γ Δ *tid cons const v cs* τ $\Longrightarrow$ Θ ; *B* $\vdash_{wf}$ Γ $\wedge$ *wfTh* Θ $\wedge$ *wfD* Θ *B* Γ Δ $\wedge$ *wfT* Θ *B* Γ τ $\wedge$ *wfPhi* Θ Φ
check-branch-list Θ Φ *B* Γ Δ *tid dclist v css* τ $\Longrightarrow$ Θ ; *B* $\vdash_{wf}$ Γ $\wedge$ *wfTh* Θ $\wedge$ *wfD* Θ *B* Γ Δ $\wedge$ *wfT* Θ *B* Γ τ $\wedge$ *wfPhi* Θ Φ
⟨*proof*⟩

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 ; $\Delta \vdash s \Leftarrow \tau$ $\Longrightarrow$ 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 $\Longrightarrow$ *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 $\Longrightarrow$ *wfCSS* Θ Φ *B* Γ Δ *tid dclist css* (*b-of* τ)
⟨*proof*⟩

lemma *check-s-wf*:
fixes *s*::*s*
assumes Θ ; Φ ; *B* ; Γ ; $\Delta \vdash s \Leftarrow \tau$
shows Θ ; *B* $\vdash_{wf}$ Γ $\wedge$ *wfT* Θ *B* Γ τ $\wedge$ *wfPhi* Θ Φ $\wedge$ *wfTh* Θ $\wedge$ *wfD* Θ *B* Γ Δ $\wedge$ *wfS* Θ Φ *B* Γ Δ *s* (*b-of* τ)
⟨*proof*⟩

lemma *check-s-flip-u1*:
fixes *s*::*s* **and** *u*::*u* **and** *u'*::*u*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash s \Leftarrow \tau$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; ($u \leftrightarrow u'$) $\cdot$ Δ $\vdash$ ($u \leftrightarrow u'$) $\cdot$ $s \Leftarrow \tau$
⟨*proof*⟩

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*⟩

lemma *check-s-flip-u*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
shows $\Theta ; \Phi ; B ; \Gamma ; (u \leftrightarrow u') \cdot \Delta \; \vdash (u \leftrightarrow u') \cdot s \Leftarrow \; \tau = (\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau)$
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩
nominal-termination (*eqvt*) ⟨*proof*⟩

lemma *infer-v-pair2I*:
fixes $v_1::v$ **and** $v_2::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow \tau_1$ **and** $\Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow \tau_2$
shows $\exists \tau.\ \Theta; \mathcal{B}; \Gamma \vdash V\text{-}pair\ v_1\ v_2 \Rightarrow \tau \wedge b\text{-}of\ \tau = B\text{-}pair\ (b\text{-}of\ \tau_1)\ (b\text{-}of\ \tau_2)$
⟨*proof*⟩

lemma *infer-v-pair2I-zbc*:
fixes $v_1::v$ **and** $v_2::v$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow \tau_1$ **and** $\Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow \tau_2$
shows $\exists z\ \tau.\ \Theta; \mathcal{B}; \Gamma \vdash V\text{-}pair\ v_1\ v_2 \Rightarrow \tau \wedge \tau = (\{\!\mid z : B\text{-}pair\ (b\text{-}of\ \tau_1)\ (b\text{-}of\ \tau_2) \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ (V\text{-}pair\ v_1\ v_2)) \mid\!\}) \wedge atom\ z \sharp (v_1,v_2) \wedge atom\ z \sharp \Gamma$
⟨*proof*⟩

lemma *infer-v-pair2E*:
assumes $\Theta; \mathcal{B}; \Gamma \vdash V\text{-}pair\ v_1\ v_2 \Rightarrow \tau$
shows $\exists \tau_1\ \tau_2\ z\ .\ \Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow \tau_1 \wedge \Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow \tau_2 \wedge$
$\tau = (\{\!\mid z : B\text{-}pair\ (b\text{-}of\ \tau_1)\ (b\text{-}of\ \tau_2) \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ (V\text{-}pair\ v_1\ v_2)) \mid\!\}) \wedge atom\ z \sharp\ (v_1, v_2)$
⟨*proof*⟩

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; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!\mid x : b \mid c \mid\!\} \wedge \tau = \tau'[x::=v]_{\tau v} \wedge$ *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *v*, τ)
⟨*proof*⟩

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* $\wedge\ \Theta \vdash_{wf} \Phi \wedge\ \Theta$; $\mathcal{B} \vdash_{wf} b \wedge$
(Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!\mid x : ba[bv::=b]_{bb} \mid c[bv::=b]_{cb} \mid\!\}) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge$ *atom x* $\sharp\ \Gamma \wedge$ *atom bv* $\sharp$ *v*
⟨*proof*⟩

12.8 Weakening

Lemmas showing that typing judgements hold when a context is extended

lemma *subtype-weakening*:
fixes Γ'::Γ
assumes Θ; $\mathcal{B}$; $\Gamma \vdash \tau 1 \lesssim \tau 2$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}$; $\Gamma' \vdash \tau 1 \lesssim \tau 2$
⟨*proof*⟩

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; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *toSet* $\Gamma \subseteq$ *toSet* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}$; $\Gamma' \vdash v \Rightarrow \tau$
⟨*proof*⟩

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; $\mathcal{B}$; $\Gamma' \vdash v \Leftarrow \tau$
⟨*proof*⟩

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'; $\Delta \vdash e \Rightarrow \tau$
⟨*proof*⟩

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 ; $\Delta \vdash e \Rightarrow \tau$ **and** *setD* $\Delta \subseteq$ *setD* Δ' **and** *wfD* Θ $\mathcal{B}$ Γ Δ'
shows Θ; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash e \Rightarrow \tau$
⟨*proof*⟩

lemma *wfG-x-fresh-in-v-simple*:
fixes *x*::*x* **and** *v* :: *v*
assumes Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *atom x* ♯ Γ
shows *atom x* ♯ *v*
⟨*proof*⟩

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* $\Longrightarrow$ *toSet* $\Gamma \subseteq$ *toSet* Γ' $\Longrightarrow$ Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ $\Longrightarrow$ *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* $\Longrightarrow$ *toSet* $\Gamma \subseteq$ *toSet* Γ' $\Longrightarrow$ Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ $\Longrightarrow$ *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* $\Longrightarrow$ *toSet* $\Gamma \subseteq$ *toSet* Γ' $\Longrightarrow$ Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ $\Longrightarrow$ *check-branch-list* Θ Φ $\mathcal{B}$ Γ' Δ *tid dclist v css t*
⟨*proof*⟩

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*⟩

lemma *fresh-z-subst-g*:
fixes *G*::Γ
assumes *atom z'* ♯ (*x*,*v*) **and** ‹*atom z'* ♯ *G*›
shows *atom z'* ♯ $G[x{::=}v]_{\Gamma v}$
⟨*proof*⟩

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 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* ♯ (*subst-gv G x v*)
⟨*proof*⟩

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 Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash s \Leftarrow \tau$ $\Longrightarrow$ *setD* $\Delta \subseteq$ *setD* Δ' $\Longrightarrow$ *wfD* Θ $\mathcal{B}$ Γ Δ' $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash s \Leftarrow \tau$ **and**
check-branch-s Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs* τ $\Longrightarrow$ *setD* $\Delta \subseteq$ *setD* Δ' $\Longrightarrow$ *wfD* Θ $\mathcal{B}$ Γ Δ' $\Longrightarrow$ *check-branch-s* Θ Φ $\mathcal{B}$ Γ Δ' *tid cons const v cs* τ **and**
check-branch-list Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v css* τ $\Longrightarrow$ *setD* $\Delta \subseteq$ *setD* Δ' $\Longrightarrow$ *wfD* Θ $\mathcal{B}$ Γ Δ' $\Longrightarrow$ *check-branch-list* Θ Φ $\mathcal{B}$ Γ Δ' *tid dclist v css* τ

⟨*proof*⟩

lemma *valid-ce-eq*:
fixes *v*::*v* **and** *ce2*::*ce*
assumes *ce1* = *ce2*$[x::=v]_{cev}$ **and** *wfV* Θ 𝓑 *GNil v b* **and** *wfCE* Θ 𝓑 ((*x*, *b*, *TRUE*) $\#_\Gamma$ *GNil*) *ce2 b′* **and** *wfCE* Θ 𝓑 *GNil ce1 b′*
shows ‹Θ; 𝓑; (*x*, *b*, ($[[x]^v]^{ce}$ == $[v]^{ce}$)) $\#_\Gamma$ *GNil* ⊨ *ce1* == *ce2* ›
⟨*proof*⟩

lemma *check-v-top*:
fixes *v*::*v*
assumes Θ; 𝓑; *GNil* ⊢ *v* ⇐ τ **and** *ce1* = *ce2*$[z::=v]_{cev}$ **and** Θ; 𝓑; *GNil* $\vdash_{wf}$ ⦃ *z* : *b-of* τ | *ce1* == *ce2* ⦄
and *supp ce1* ⊆ *supp* 𝓑
shows Θ; 𝓑; *GNil* ⊢ *v* ⇐ ⦃ *z* : *b-of* τ | *ce1* == *ce2* ⦄
⟨*proof*⟩

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*
  $\langle proof \rangle$
nominal-termination (*eqvt*) $\langle proof \rangle$

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* $\langle proof \rangle$

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'*
  $\langle proof \rangle$

lemma *replace-in-g-wfG*:
assumes Θ; $\mathcal{B} \vdash G \langle\ xcs\ \rangle \rightsquigarrow G'$ **and** *wfG* Θ $\mathcal{B}$ G
shows *wfG* Θ $\mathcal{B}$ G'
⟨*proof*⟩

lemma *wfD-rig-single*:
fixes Δ::Δ **and** x::x **and** c::c **and** G::Γ
assumes Θ; $\mathcal{B}$; $G \vdash_{wf} \Delta$ **and** *wfG* Θ $\mathcal{B}$ ($G[x \longmapsto c]$)
shows Θ; $\mathcal{B}$; $G[x \longmapsto c] \vdash_{wf} \Delta$
⟨*proof*⟩

lemma *wfD-rig*:
assumes Θ; $\mathcal{B} \vdash G \langle\ xcs\ \rangle \rightsquigarrow G'$ **and** *wfD* Θ $\mathcal{B}$ G Δ
shows *wfD* Θ $\mathcal{B}$ G' Δ
⟨*proof*⟩

lemma *replace-in-g-fresh*:
fixes x::x
assumes Θ; $\mathcal{B} \vdash \Gamma \langle\ xcs\ \rangle \rightsquigarrow \Gamma'$ **and** *wfG* Θ $\mathcal{B}$ Γ **and** *wfG* Θ $\mathcal{B}$ Γ' **and** *atom* x $\sharp$ Γ
shows *atom* x $\sharp$ Γ'
⟨*proof*⟩

lemma *replace-in-g-fresh1*:
fixes x::x
assumes Θ; $\mathcal{B} \vdash \Gamma \langle\ xcs\ \rangle \rightsquigarrow \Gamma'$ **and** *wfG* Θ $\mathcal{B}$ Γ **and** *atom* x $\sharp$ Γ
shows *atom* x $\sharp$ Γ'
⟨*proof*⟩

Wellscoping for an eXchange list

inductive *wsX*:: $\Gamma \Rightarrow (x*c)$ *list* $\Rightarrow$ *bool* **where**
wsX-NilI: *wsX G* []
| *wsX-ConsI*: ⟦ *wsX G xcs* ; *atom x* $\in$ *atom-dom G* ; $x \notin$ *fst ‘ set xcs* ⟧ $\Longrightarrow$ *wsX G* ((*x*,*c*)#*xcs*)
equivariance *wsX*
nominal-inductive *wsX* ⟨*proof*⟩

lemma *wsX-if1*:
assumes *wsX G xcs*
shows ((*atom ‘ fst ‘ set xcs*) $\subseteq$ *atom-dom G*) $\wedge$ *List.distinct* (*List.map fst xcs*)
⟨*proof*⟩

lemma *wsX-if2*:
assumes ((*atom ‘ fst ‘ set xcs*) $\subseteq$ *atom-dom G*) $\wedge$ *List.distinct* (*List.map fst xcs*)
shows *wsX G xcs*
⟨*proof*⟩

lemma *wsX-iff*:
wsX G xcs = (((*atom ‘ fst ‘ set xcs*) $\subseteq$ *atom-dom G*) $\wedge$ *List.distinct* (*List.map fst xcs*))
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *wsX-cons2*:
assumes *wsX* Γ *xcs* **and** $x \notin$ *fst ' set xcs*
shows *wsX* ((x, b, c1) $\#_\Gamma$ Γ) *xcs*
⟨*proof*⟩

lemma *wsX-cons3*:
assumes *wsX* Γ *xcs*
shows *wsX* ((x, b, c1) $\#_\Gamma$ Γ) *xcs*
⟨*proof*⟩

lemma *wsX-fresh*:
assumes *wsX G xcs* **and** *atom x* ♯ *G* **and** *wfG* Θ $\mathcal{B}$ *G*
shows $x \notin$ *fst ' set xcs*
⟨*proof*⟩

lemma *replace-in-g-dist*:
assumes $x' \neq x$
shows *replace-in-g* ((x, b,c) $\#_\Gamma$ G) x' c'' = ((x, b,c) $\#_\Gamma$ (*replace-in-g* G x' c'')) ⟨*proof*⟩

lemma *wfG-replace-inside-rig*:
fixes c''::c
assumes ‹Θ; $\mathcal{B}$ $\vdash_{wf}$ G[x'⟼c'']› ‹Θ; $\mathcal{B}$ $\vdash_{wf}$ (x, b, c) $\#_\Gamma$ G ›
shows Θ; $\mathcal{B}$ $\vdash_{wf}$ (x, b, c) $\#_\Gamma$ G[x'⟼c'']
⟨*proof*⟩

lemma *replace-in-g-valid-weakening*:
assumes Θ; $\mathcal{B}$; Γ[x'⟼c''] ⊨ c' **and** $x' \neq x$ **and** Θ; $\mathcal{B}$ $\vdash_{wf}$ (x, b, c) $\#_\Gamma$ Γ[x'⟼c'']
shows Θ; $\mathcal{B}$; ((x, b,c) $\#_\Gamma$ Γ)[x'⟼ c''] ⊨ c'
⟨*proof*⟩

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'$)
⟨*proof*⟩

lemma *replace-in-g-split*:
fixes G::Γ
assumes Γ = *replace-in-g* Γ′ x c **and** Γ′ = G'@(x,b,c')$\#_\Gamma$G **and** *wfG* Θ $\mathcal{B}$ Γ′
shows Γ = G'@(x,b,c)$\#_\Gamma$G
⟨*proof*⟩

lemma *replace-in-g-subtyped-split0*:
fixes G::Γ
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ′[(x,c)] Γ **and** Γ′ = G'@(x,b,c')$\#_\Gamma$G **and** *wfG* Θ $\mathcal{B}$ Γ′
shows Γ = G'@(x,b,c)$\#_\Gamma$G

⟨*proof*⟩

lemma *replace-in-g-subtyped-split*:
assumes *Some* $(b,\ c') =$ *lookup* $G\ x$ **and** $\Theta;\ \mathcal{B};$ *replace-in-g* $G\ x\ c\ \models c'$ **and** *wfG* $\Theta\ \mathcal{B}\ G$
shows $\exists\ \Gamma\ \Gamma'.\ G = \Gamma'@(x,b,c')\#_\Gamma\Gamma \wedge \Theta;\ \mathcal{B};\ \Gamma'@(x,b,c)\#_\Gamma\Gamma \models c'$
⟨*proof*⟩

13.2 Validity and Subtyping

lemma *wfC-replace-in-g*:
fixes $c::c$ **and** $c0::c$
assumes $\Theta;\ \mathcal{B};\ \Gamma'@(x,b,c0')\#_\Gamma\Gamma \vdash_{wf}\ c$ **and** $\Theta;\ \mathcal{B};\ (x,b,TRUE)\#_\Gamma\Gamma \vdash_{wf}\ c0$
shows $\Theta;\ \mathcal{B};\ \Gamma'\ @\ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma \vdash_{wf}\ c$
⟨*proof*⟩

lemma *ctx-subtype-valid*:
assumes $\Theta;\ \mathcal{B};\ \Gamma'@(x,b,c0')\#_\Gamma\Gamma \models c$ **and**
$\Theta;\ \mathcal{B};\ \Gamma'@(x,b,c0)\#_\Gamma\Gamma \models c0'$
shows $\Theta;\ \mathcal{B};\ \Gamma'@(x,b,c0)\#_\Gamma\Gamma \models c$
⟨*proof*⟩

lemma *ctx-subtype-subtype*:
fixes $\Gamma::\Gamma$
shows $\Theta;\ \mathcal{B};\ G \vdash t1 \lesssim t2 \Longrightarrow G = \Gamma'@(x,b0,c0')\#_\Gamma\Gamma \Longrightarrow \Theta;\ \mathcal{B};\ \Gamma'@(x,b0,c0)\#_\Gamma\Gamma \models c0' \Longrightarrow \Theta;\ \mathcal{B};\ \Gamma'@(x,b0,c0)\#_\Gamma\Gamma \vdash t1 \lesssim t2$
⟨*proof*⟩

lemma *ctx-subtype-subtype-rig*:
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'\ [(x,c0)]\ \Gamma$ **and** $\Theta;\ \mathcal{B};\ \Gamma' \vdash t1 \lesssim t2$
shows $\Theta;\ \mathcal{B};\ \Gamma \vdash t1 \lesssim t2$
⟨*proof*⟩

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* $\Theta\ \mathcal{B}\ \Gamma'\ xcs\ \Gamma$ **and** $\Theta;\ \mathcal{B};\ \Gamma' \vdash t1 \lesssim t2$
shows $\Theta;\ \mathcal{B};\ \Gamma \vdash t1 \lesssim t2$
⟨*proof*⟩

lemma *replace-in-g-inside-valid*:
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'\ [(x,c0)]\ \Gamma$ **and** *wfG* $\Theta\ \mathcal{B}\ \Gamma'$
shows $\exists\, b\ c0'\ G\ G'.\ \Gamma' = G'\ @\ (x,b,c0')\#_\Gamma G \wedge\ \Gamma = G'\ @\ (x,b,c0)\#_\Gamma G \wedge \Theta;\ \mathcal{B};\ G'@\ (x,b,c0)\#_\Gamma G \models c0'$
⟨*proof*⟩

lemma *replace-in-g-valid*:
assumes $\Theta;\ \mathcal{B}\ \vdash G\ \langle\ xcs\ \rangle \rightsquigarrow G'$ **and** $\Theta;\ \mathcal{B};\ G\ \models c$
shows $\langle\Theta;\ \mathcal{B};\ G'\ \models c\ \rangle$
⟨*proof*⟩

13.3 Literals

13.4 Values

lemma *lookup-inside-unique-b*[*simp*]:
assumes Θ ; $B \vdash_{wf}$ (Γ'@(x,$b0$,$c0$)$\#_\Gamma\Gamma$) **and** Θ ; $B \vdash_{wf}$ (Γ'@(x,$b0$,$c0'$)$\#_\Gamma\Gamma$)
and *Some* (b, c) = *lookup* (Γ' @ (x, $b0$, $c0'$) $\#_\Gamma$ Γ) y **and** *Some* ($b0$,$c0$) = *lookup* (Γ'@((x,$b0$,$c0$))$\#_\Gamma\Gamma$) x **and** x=y
shows b = $b0$
⟨*proof*⟩

lemma *ctx-subtype-v-aux*:
fixes v::v
assumes Θ; $\mathcal{B}$; Γ'@((x,$b0$,$c0'$)$\#_\Gamma\Gamma$) $\vdash v \Rightarrow t1$ **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*⟩

lemma *ctx-subtype-v*:
fixes v::v
assumes Θ; $\mathcal{B}$; Γ'@((x,$b0$,$c0'$)$\#_\Gamma\Gamma$) $\vdash v \Rightarrow t1$ **and** Θ; $\mathcal{B}$; Γ'@(x,$b0$,$c0$)$\#_\Gamma\Gamma \models c0'$
shows $\exists t2$. Θ; $\mathcal{B}$; Γ'@((x,$b0$,$c0$)$\#_\Gamma\Gamma$) $\vdash v \Rightarrow t2 \wedge$ Θ; $\mathcal{B}$; Γ'@((x,$b0$,$c0$)$\#_\Gamma\Gamma$) $\vdash t2 \lesssim t1$
⟨*proof*⟩

lemma *ctx-subtype-v-eq*:
fixes v::v
assumes
Θ; $\mathcal{B}$; Γ'@((x,$b0$,$c0'$)$\#_\Gamma\Gamma$) $\vdash v \Rightarrow t1$ **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*⟩

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*⟩

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; $\mathcal{B}$; Γ' $\vdash v \Rightarrow t1$
shows Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow t1$
⟨*proof*⟩

lemma *ctx-subtype-v-rigs-eq*:
fixes v::v
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' *xcs* Γ **and**
Θ; $\mathcal{B}$; Γ' $\vdash v \Rightarrow t1$
shows Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow t1$
⟨*proof*⟩

lemma *ctx-subtype-check-v-rigs-eq*:

assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' *xcs* Γ **and**
Θ; $\mathcal{B}$; Γ' $\vdash v \Leftarrow t1$
shows Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow t1$
⟨*proof*⟩

13.5 Expressions

lemma *valid-wfC*:
fixes *c0*::*c*
assumes Θ; $\mathcal{B}$; $\Gamma'@(x,b0,c0)\#_\Gamma\Gamma \models c0'$
shows Θ; $\mathcal{B}$; $(x,\ b0,\ TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c0$
⟨*proof*⟩

lemma *ctx-subtype-e-eq*:
fixes *G*::Γ
assumes
Θ ; Φ ; $\mathcal{B}$; G ; $\Delta \vdash e \Rightarrow t1$ **and** $G = \Gamma'@((x,b0,c0')\#_\Gamma\Gamma)$
Θ; $\mathcal{B}$; $\Gamma'@(x,b0,c0)\#_\Gamma\Gamma \models c0'$
shows Θ ; Φ ; $\mathcal{B}$; $\Gamma'@((x,b0,c0)\#_\Gamma\Gamma)$; $\Delta \vdash e \Rightarrow t1$
⟨*proof*⟩

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*⟩

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 ; $\Delta \vdash e \Rightarrow t1$
⟨*proof*⟩

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*⟩

lemma *replace-in-g-subtyped-empty*:
assumes *wfG* Θ $\mathcal{B}$ $(\Gamma'\ @\ (x,\ b,\ c[z{::=}V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$
shows *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* $(\Gamma'\ @\ (x,\ b,\ c[z{::=}V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$ x $(c'[z'{::=}V\text{-}var\ x]_{cv})$) [] $(\Gamma'\ @\ (x,\ b,\ c'[z'{::=}V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$
⟨*proof*⟩

lemma *ctx-subtype-s*:
  fixes $s::s$
  assumes Θ ; Φ ; $\mathcal{B}$; $\Gamma'@((x,b,c[z::=V\text{-}var\ x]_{cv})\#_{\Gamma}\Gamma)$; $\Delta \vdash s \Leftarrow \tau$ **and**
    Θ; $\mathcal{B}$; $\Gamma \vdash \{\!\mid z' : b \mid c' \mid\!\} \lesssim \{\!\mid z : b \mid c \mid\!\}$ **and**
    $atom\ x \sharp (z,z',c,c')$
  shows Θ ; Φ ; $\mathcal{B}$; $\Gamma'@(x,b,c'[z'::=V\text{-}var\ x]_{cv})\#_{\Gamma}\Gamma$; $\Delta \vdash s \Leftarrow \tau$
⟨*proof*⟩

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*:
  $\{\!\!\{ z : b \mid TRUE \}\!\!\} = \{\!\!\{ z : b \mid TRUE \}\!\!\}[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *wfD-subst*:
  fixes $\tau_1::\tau$ **and** $v::v$ **and** $\Delta::\Delta$ **and** $\Theta::\Theta$ **and** $\Gamma::\Gamma$
  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** $b\text{-}of\ \tau_1=b_1$
  shows $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\ \ \vdash_{wf}\ \Delta[x::=v]_{\Delta v}$
⟨*proof*⟩

lemma *subst-v-c-of*:
  assumes *atom xa* ♯ (*v*,*x*)
  shows $c\text{-}of\ t[x::=v]_{\tau v}\ xa = (c\text{-}of\ t\ xa)[x::=v]_{cv}$
  ⟨*proof*⟩

14.3 Context

lemma *subst-lookup*:
  assumes $Some\ (b,c) = lookup\ (\Gamma'@((x,b_1,c_1)\#_\Gamma\Gamma))\ y$ **and** $x \neq y$ **and** $wfG\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b_1,c_1)\#_\Gamma\Gamma))$
  shows $\exists\, d.\ Some\ (b,d) = lookup\ ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)\ y$

⟨*proof*⟩

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*⟩

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*⟩

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-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*⟩

lemma *subst-valid*:
fixes $v::v$ **and** $c'::c$ **and** $\Gamma ::\Gamma$
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*⟩

lemma *subst-valid-infer-v*:
fixes $v::v$ **and** $c'::c$
assumes $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{\!| \; z0 : b \mid c0 \; |\!\}$ **and** *atom* $x \sharp c$ **and** *atom* $x \sharp G$ **and** *wfG* $\Theta\ \mathcal{B}$ $(G'@(x,b,c[z::=[x]^v]_{cv}\ )\ \#_\Gamma\ G)$ **and** *atom* $z0 \sharp v$
$\Theta;\mathcal{B};(z0,b,c0)\#_\Gamma G \models c[z::=V\text{-}var\ z0]_{cv}$ **and** *atom* $z0 \sharp c$ **and**
$\Theta;\mathcal{B};G'@(x,b,\ c[z::=[x]^v]_{cv}\ )\ \#_\Gamma\ G \models c'$ (**is** $\Theta ; \mathcal{B};\ ?G \models c'$)
shows $\Theta;\mathcal{B};G'[x::=v]_{\Gamma v}@G \models c'[x::=v]_{cv}$
⟨*proof*⟩

14.5 Subtyping

lemma *subst-subtype*:
fixes $v::v$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow (\{\!|z0:b|c0|\!\})$ **and**
$\Theta;\mathcal{B};\Gamma \vdash (\{\!|z0:b|c0|\!\}) \lesssim (\{\!| \; z : b \mid c \; |\!\})$ **and**
$\Theta;\mathcal{B};\Gamma'@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash (\{\!| \; z1 : b1 \mid c1 \; |\!\}) \lesssim (\{\!| \; z2 : b1 \mid c2 \; |\!\})$ (**is** $\Theta ; \mathcal{B};\ ?G1 \vdash ?t1 \lesssim$ $?t2$) **and**
atom $z \sharp (x,v) \wedge$ *atom* $z0 \sharp (c,x,v,z,\Gamma) \wedge$ *atom* $z1 \sharp (x,v) \wedge$ *atom* $z2 \sharp (x,v)$ **and** *wsV* $\Theta\ \mathcal{B}\ \Gamma\ v$

shows $\Theta;\mathcal{B};\Gamma'[x::=v]_{\Gamma v}@\Gamma \vdash$ $\{\!\!\{$ *z1* : *b1* | *c1* $\}\!\!\}[x::=v]_{\tau v} \lesssim$ $\{\!\!\{$ *z2* : *b1* | *c2* $\}\!\!\}[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *subst-subtype-tau*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau \lesssim$ ($\{\!\!\{$ *z* : *b* | *c* $\}\!\!\}$)
Θ ; $\mathcal{B}$; $\Gamma'@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash \tau 1 \lesssim \tau 2$ **and**
atom z ♯ (*x*,*v*)
shows Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}@\Gamma \vdash$ $\tau 1[x::=v]_{\tau v}$ $\lesssim$ $\tau 2[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *subtype-if1*:
fixes *v*::*v*
assumes *P* ; $\mathcal{B}$; Γ ⊢ *t1* $\lesssim$ *t2* **and** *wfV P* $\mathcal{B}$ 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* ; $\mathcal{B}$; $\Gamma \vdash$ $\{\!\!\{$ *z1* : *b-of t1* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t1 z1*) $\}\!\!\} \lesssim \{\!\!\{$ *z2* : *b-of t2* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t2 z2*) $\}\!\!\}$
⟨*proof*⟩

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*⟩

lemma *subst-t-b-eq*:
fixes *x*::*x* **and** *v*::*v*
shows *b-of* $(\tau[x::=v]_{\tau v})$ = *b-of* τ
⟨*proof*⟩

lemma *fresh-g-fresh-v*:
fixes *x*::*x*
assumes *atom x* ♯ Γ **and** *wfV* Θ $\mathcal{B}$ Γ *v b*
shows *atom x* ♯ *v*
⟨*proof*⟩

lemma *infer-v-fresh-g-fresh-v*:
fixes *x*::*x* **and** Γ::Γ **and** *v*::*v*
assumes *atom x* ♯ $\Gamma'@\Gamma$ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$
shows *atom x* ♯ *v*
⟨*proof*⟩

lemma *infer-v-fresh-g-fresh-xv*:
fixes *xa*::*x* **and** *v*::*v* **and** Γ::Γ
assumes *atom xa* ♯ $\Gamma'@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma)$ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$
shows *atom xa* ♯ (*x*,*v*)
⟨*proof*⟩

lemma *wfG-subst-infer-v*:
fixes $v::v$
assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ @ $(x, b, c0[z0::=[x]^v]_{cv})$ $\#_\Gamma$ Γ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *b-of* $\tau = b$
shows Θ ; $\mathcal{B}\vdash_{wf}$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ
⟨*proof*⟩

lemma *fresh-subst-gv-inside*:
fixes $\Gamma::\Gamma$
assumes *atom* $z \sharp \Gamma'$ @ $(x, b_1, c0[z0::=[\ x\ ]^v]_{cv})$ $\#_\Gamma$ Γ **and** *atom* $z \sharp v$
shows *atom* $z \sharp \Gamma'[x::=v]_{\Gamma v}$@$\Gamma$
⟨*proof*⟩

lemma *subst-t*:
fixes $x::x$ **and** $b::b$ **and** $xa::x$
assumes *atom* $z \sharp x$ **and** *atom* $z \sharp v$
shows $(\{\!\!\{\ z : b\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ v'[x::=v]_{vv}\ ]^{ce}\ \}\!\!\}) = (\{\!\!\{\ z : b\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ v']^{ce}\ \}\!\!\}[x::=v]_{\tau v})$
⟨*proof*⟩

lemma *infer-l-fresh*:
assumes $\vdash l \Rightarrow \tau$
shows *atom* $x \sharp \tau$
⟨*proof*⟩

lemma *subst-infer-v*:
fixes $v::v$ **and** $v'::v$
assumes Θ ; $\mathcal{B}$; Γ'@$((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash v' \Rightarrow \tau_2$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim (\{\!\!\{\ z0 : b_1 \mid c0\ \}\!\!\})$ **and** *atom* $z0 \sharp (x,v)$
shows Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v})$@$\Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau_2[x::=v]_{\tau v}$
⟨*proof*⟩

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 ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim \{\!\!\{\ z0 : b_1 \mid c0\ \}\!\!\}$ **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*⟩

lemma *type-veq-subst*[*simp*]:
assumes *atom* $z \sharp (x,v)$
shows $\{\!\!\{\ z : b \mid$ *CE-val* (*V-var* z) $==$ *CE-val* v' $\}\!\!\}[x::=v]_{\tau v} = \{\!\!\{\ z : b \mid$ *CE-val* (*V-var* z) $==$ *CE-val* $v'[x::=v]_{vv}$ $\}\!\!\}$
⟨*proof*⟩

lemma *subst-infer-v-form*:
fixes $v::v$ **and** $v'::v$ **and** $\Gamma::\Gamma$
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
Θ ; $\mathcal{B}$; Γ'@$((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash v' \Rightarrow \tau_2$ **and** $b=$ *b-of* τ_2
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim (\{\!\!\{\ z0 : b_1 \mid c0\ \}\!\!\})$ **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 ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash v'[x::=v]_{vv} \Rightarrow \{\!\!\{\ z3' : b \mid$ *CE-val* (*V-var* $z3'$) $==$ *CE-val*

$v'[x::=v]_{vv}$ $\}\rangle$
⟨proof⟩

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**
$\Theta; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \{\!\mid z0 : b_1 \mid c0 \mid\!\}$ **and** $atom\ z0 \sharp (x,v)$
shows $\Theta ; \Phi ; \mathcal{B} ; ((\Gamma'[x::=v]_{\Gamma v})@\Gamma) ; (\Delta[x::=v]_{\Delta v}) \vdash (subst\text{-}ev\ e\ x\ v) \Rightarrow \tau_2[x::=v]_{\tau v}$
⟨proof⟩

lemma *infer-e-uniqueness*:
assumes $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash e_1 \Rightarrow \tau_1$ **and** $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash e_1 \Rightarrow \tau_2$
shows $\tau_1 = \tau_2$
⟨proof⟩

14.8 Statements

lemma *subst-infer-check-v1*:
fixes $v::v$ **and** $v'::v$ **and** $\Gamma::\Gamma$
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**
$\Theta ; \mathcal{B} ; \Gamma_2 \vdash \tau_1 \lesssim \{\!\mid z0 : b_1 \mid c0 \mid\!\}$ **and** $atom\ z0 \sharp (x,v)$
shows $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash v'[x::=v]_{vv} \Leftarrow \tau_2[x::=v]_{\tau v}$
⟨proof⟩

lemma *infer-v-c-valid*:
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash \tau \lesssim \{\!\mid z : b \mid c \mid\!\}$
shows $\langle\Theta ; \mathcal{B} ; \Gamma \models c[z::=v]_{cv}\rangle$
⟨proof⟩

Substitution Lemma for Statements

lemma *subst-infer-check-s*:
fixes $v::v$ **and** $s::s$ **and** $cs::branch\text{-}s$ **and** $x::x$ **and** $c::c$ **and** $b::b$ **and**
$\Gamma_1::\Gamma$ **and** $\Gamma_2::\Gamma$ **and** $css::branch\text{-}list$
assumes $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **and** $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \{\!\mid z : b \mid c \mid\!\}$ **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
Θ ; Φ ; $\mathcal{B}$; Γ; Δ; *tid* ; *cons* ; *const* ; $v' \vdash$ *cs* $\Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2 @((x,b,c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1)) \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$;
tid ; *cons* ; *const* ; $v'[x::=v]_{vv} \vdash$ $cs[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$
and
Θ ; Φ ; $\mathcal{B}$; Γ; Δ; *tid* ; *dclist* ; $v' \vdash$ *css* $\Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2 @((x,b,c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1)) \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$; *tid* ; *dclist* ; $v'[x::=v]_{vv} \vdash$
subst-branchlv css x v $\Leftarrow \tau'[x::=v]_{\tau v}$

⟨*proof*⟩

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 ; $\mathcal{B}$; $\Gamma_1 \vdash v \Leftarrow \{\!\!\{ z : b \mid c \}\!\!\}$ **and** *atom z* $\sharp$ (x, v)
and *check-s* Θ Φ $\mathcal{B}$ Γ Δ *s* τ' **and** $\Gamma = (\Gamma_2 @((x,b,c[z::=[x]^v]_{cv}) \#_\Gamma \Gamma_1))$
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*⟩

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*⟩

lemma *subtype-let*:
fixes *s'*::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AS-let x* e_1 *s* $\Leftarrow \tau \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_1 \Rightarrow \tau_1 \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_2 \Rightarrow \tau_2 \Longrightarrow$ Θ ; $\mathcal{B}$; *GNil* $\vdash \tau_2 \lesssim \tau_1 \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AS-let x* e_2 *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*⟩

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* $\sharp$ v
  shows $(v'[x::=v]_{vv})[bv::=b]_{vb} = (v'[bv::=b]_{vb})[x::=v]_{vv}$
  ⟨*proof*⟩

lemma *subst-cev-subst-bb-commute*:
  fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v::v$
  assumes *atom bv* $\sharp$ v
  shows $(ce[x::=v]_v)[bv::=b]_{ceb} = (ce[bv::=b]_{ceb})[x::=v]_v$
  ⟨*proof*⟩

lemma *subst-cv-subst-bb-commute*:
  fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v::v$
  assumes *atom bv* $\sharp$ v
  shows $c[x::=v]_{cv}[bv::=b]_{cb} = (c[bv::=b]_{cb})[x::=v]_{cv}$
  ⟨*proof*⟩

lemma *subst-b-c-of*:
  $(\textit{c-of}\ \tau\ z)[bv::=b]_{cb} = \textit{c-of}\ (\tau[bv::=b]_{\tau b})\ z$
⟨*proof*⟩

lemma *subst-b-b-of*:
  $(\textit{b-of}\ \tau)[bv::=b]_{bb} = \textit{b-of}\ (\tau[bv::=b]_{\tau b})$
  ⟨*proof*⟩

lemma *subst-b-if*:
  $\{\!\mid z : \textit{b-of}\ \tau[bv::=b]_{\tau b} \mid \textit{CE-val}\ (v[bv::=b]_{vb}) == \textit{CE-val}\ (\textit{V-lit}\ ll)\ \ \textit{IMP}\ \ \textit{c-of}\ \tau[bv::=b]_{\tau b}\ z \mid\!\} = \{\!\mid z : \textit{b-of}\ \tau \mid \textit{CE-val}\ (v) == \textit{CE-val}\ (\textit{V-lit}\ ll)\ \ \textit{IMP}\ \ \textit{c-of}\ \tau\ z \mid\!\}[bv::=b]_{\tau b}$
  ⟨*proof*⟩

lemma *subst-b-top-eq*:
  $\{\!\mid z : \textit{B-unit} \mid \textit{TRUE} \mid\!\}[bv::=b]_{\tau b} = \{\!\mid z : \textit{B-unit} \mid \textit{TRUE} \mid\!\}$ **and** $\{\!\mid z : \textit{B-bool} \mid \textit{TRUE} \mid\!\}[bv::=b]_{\tau b} =$

$\{\!\!|\ z : B\text{-}bool\ \mid TRUE\ |\!\!\}$ **and** $\{\!\!|\ z : B\text{-}id\ tid \mid TRUE\ |\!\!\} = \{\!\!|\ z : B\text{-}id\ tid \mid TRUE\ |\!\!\}[bv::=b]_{\tau b}$
⟨proof⟩

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 $(v'[x::=V\text{-}var\ y]_{cv})[bv::=b]_{cb} = (v'[bv::=b]_{cb})[x::=V\text{-}var\ y]_{cv}$
⟨proof⟩

lemma *subst-b-infer-b*:
fixes $l::l$ **and** $b::b$
assumes $\vdash l \Rightarrow \tau$ **and** $\Theta ; \{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
shows $\vdash l \Rightarrow (\tau[bv::=b]_{\tau b})$
⟨proof⟩

lemma *subst-b-subtype*:
fixes $s::s$ **and** $b'::b$
assumes $\Theta ; \{|bv|\} ; \Gamma \vdash \tau 1 \lesssim \tau 2$ **and** $\Theta ; \{||\} \vdash_{wf} b'$ **and** $B = \{|bv|\}$
shows $\Theta ; \{||\} ; \Gamma[bv::=b']_{\Gamma b} \vdash \tau 1[bv::=b']_{\tau b} \lesssim \tau 2[bv::=b']_{\tau b}$
⟨proof⟩

lemma *b-of-subst-bv*:
$(b\text{-}of\ \tau)[x::=v]_{bb} = b\text{-}of\ (\tau[x::=v]_{\tau b})$
⟨proof⟩

lemma *subst-b-infer-v*:
fixes $v::v$ **and** $b::b$
assumes $\Theta ; B ; G \vdash v \Rightarrow \tau$ **and** $\Theta ; \{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
shows $\Theta ; \{||\} ; G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow (\tau[bv::=b]_{\tau b})$
⟨proof⟩

lemma *subst-b-check-v*:
fixes $v::v$ **and** $b::b$
assumes $\Theta ; B ; G \vdash v \Leftarrow \tau$ **and** $\Theta ; \{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
shows $\Theta ; \{||\} ; G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Leftarrow (\tau[bv::=b]_{\tau b})$
⟨proof⟩

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⟩

lemma *subst-cev-subst-vb-switch*:
shows $(ce[bv::=b']_{ceb})[x::=v[bv::=b']_{vb}]_{cev} = (ce[x::=v]_{cev})[bv::=b']_{ceb}$
⟨proof⟩

lemma *subst-cv-subst-vb-switch*:
shows $(c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv} = c[x::=v]_{cv}[bv::=b']_{cb}$
⟨proof⟩

lemma *subst-tv-subst-vb-switch*:
shows $(\tau[bv::=b']_{\tau b})[x::=v[bv::=b']_{vb}]_{\tau v} = \tau[x::=v]_{\tau v}[bv::=b']_{\tau b}$

⟨*proof*⟩

lemma *subst-tb-triple*:
  assumes *atom bv* $\sharp$ 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*⟩

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 ; $\{||\}$; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b} \vdash (e[bv::=b]_{eb}) \Rightarrow (\tau[bv::=b]_{\tau b})$
  ⟨*proof*⟩

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* $\sharp$ *const*
  shows (*c-of const x*)$[bv::=b]_{cb}$ = *c-of const x*
  ⟨*proof*⟩

lemma *subst-b-check-s*:
  fixes $s::s$ **and** $b::b$ **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** $v::v$ **and** $\tau::\tau$
  assumes Θ ; $\{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
  shows Θ ; Φ ; B ; G; $D \vdash s \Leftarrow \tau \Longrightarrow \Theta$; Φ ; $\{||\}$; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b} \vdash (s[bv::=b]_{sb}) \Leftarrow (\tau[bv::=b]_{\tau b})$ **and**
    Θ ; Φ ; B ; G; D ; *tid* ; *cons* ; *const* ; $v \vdash cs \Leftarrow \tau \Longrightarrow \Theta$; Φ ; $\{||\}$; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b}$; *tid* ; *cons* ; *const* ; $v[bv::=b]_{vb} \vdash$ (*subst-branchb cs bv b*) $\Leftarrow (\tau[bv::=b]_{\tau b})$ **and**
    Θ ; Φ ; B ; G; D ; *tid* ; *dclist* ; $v \vdash css \Leftarrow \tau \Longrightarrow \Theta$; Φ ; $\{||\}$; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b}$; *tid* ; *dclist* ; $v[bv::=b]_{vb} \vdash$ (*subst-branchlb css bv b*) $\Leftarrow (\tau[bv::=b]_{\tau b})$
  ⟨*proof*⟩

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* ($\langle$ - $\sqsubseteq$ - $\rangle$) **where**
delta-ext $\Delta\ \Delta' \equiv$ (*setD* $\Delta \subseteq$ *setD* Δ')

nominal-function *dc-of* :: *branch-s* $\Rightarrow$ *string* **where**
dc-of (*AS-branch dc - -*) = *dc*
$\langle$*proof*$\rangle$
nominal-termination (*eqvt*) $\langle$*proof*$\rangle$

lemma *delta-sim-fresh*:
assumes $\Theta \vdash \delta \sim \Delta$ **and** *atom u* $\sharp$ δ
shows *atom u* $\sharp$ Δ
$\langle$*proof*$\rangle$

lemma *delta-sim-v*:
fixes Δ::Δ
assumes $\Theta \vdash \delta \sim \Delta$ **and** $(u,v) \in$ *set* δ **and** $(u,\tau) \in$ *setD* Δ **and** Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ Δ
shows Θ ; $\{||\}$; *GNil* $\vdash v \Leftarrow \tau$
$\langle$*proof*$\rangle$

lemma *delta-sim-delta-lookup*:
assumes $\Theta \vdash \delta \sim \Delta$ **and** $(u, \{\!\!\{\ z : b \mid c\ \}\!\!\}) \in$ *setD* Δ
shows $\exists\, v.\ (u,v) \in$ *set* δ
$\langle$*proof*$\rangle$

lemma *update-d-stable*:
fst ` *set* $\delta$ = *fst* ` *set* (*update-d* δ *u v*)
$\langle$*proof*$\rangle$

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* $\vdash_{wf}$ Δ

shows $\Theta \vdash (\textit{update-d}\ \delta\ u\ v) \sim \Delta$
⟨*proof*⟩

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 $\Theta\ ;\ \Phi\ ;\ B\ ;\ \ \textit{GNil}\ ;\ D\ \vdash\ s \Leftarrow \tau$
shows $\textit{atom}\ x\ \sharp\ s \wedge \textit{atom}\ x\ \sharp\ \tau \wedge \textit{atom}\ x\ \sharp\ D$
⟨*proof*⟩

lemma *check-funtyp-subst-b*:
fixes $b'::b$
assumes $\textit{check-funtyp}\ \Theta\ \Phi\ \{|bv|\}\ (\textit{AF-fun-typ}\ x\ b\ c\ \tau\ s)$ **and** $\langle\ \Theta\ ;\ \{||\}\ \vdash_{wf}\ b'\ \rangle$
shows $\textit{check-funtyp}\ \Theta\ \Phi\ \ \{||\}\ (\textit{AF-fun-typ}\ x\ b[bv::=b']_{bb}\ (c[bv::=b']_{cb})\ \tau[bv::=b']_{\tau b}\ s[bv::=b']_{sb})$
⟨*proof*⟩

lemma *funtyp-simple-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$
assumes $\textit{check-funtyp}\ \Theta\ \Phi\ \ (\{||\}::\textit{bv fset})\ (\textit{AF-fun-typ}\ x\ b\ c\ \tau\ s)$ **and**
$\Theta\ ;\ \{||\}\ ;\ \textit{GNil} \vdash v \Leftarrow \{\!\!\{\ x : b\ \mid\ c\ \}\!\!\}$
shows $\Theta\ ;\ \Phi\ ;\ \ \{||\}\ ;\ \textit{GNil}\ ;\ \textit{DNil} \vdash s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *funtypq-simple-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$
assumes $\textit{check-funtypq}\ \Theta\ \Phi\ \ \ (\textit{AF-fun-typ-none}\ (\textit{AF-fun-typ}\ x\ b\ c\ t\ s))$ **and**
$\Theta\ ;\ \{||\}\ ;\ \textit{GNil} \vdash v \Leftarrow \{\!\!\{\ x : b\ \mid\ c\ \}\!\!\}$
shows $\Theta;\ \Phi;\ \{||\};\ \textit{GNil};\ \textit{DNil} \vdash s[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *funtyp-poly-eq-iff-equalities*:
assumes $[[\textit{atom}\ bv']]\textit{lst}.\ \textit{AF-fun-typ}\ x'\ b''\ c'\ t'\ s' = [[\textit{atom}\ bv]]\textit{lst}.\ \textit{AF-fun-typ}\ x\ b\ c\ t\ s$
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*⟩

lemma *funtypq-poly-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$ **and** $b'::b$
assumes $\textit{check-funtypq}\ \Theta\ \Phi\ \ \ (\textit{AF-fun-typ-some}\ bv\ (\textit{AF-fun-typ}\ x\ b\ c\ t\ s))$ **and**
$\Theta\ ;\ \{||\}\ ;\ \textit{GNil} \vdash v \Leftarrow \{\!\!\{\ x : b[bv::=b']_{bb} \mid c[bv::=b']_{cb}\ \}\!\!\}$ **and**
$\Theta\ ;\ \{||\} \vdash_{wf}\ b'$
shows $\Theta;\ \Phi;\ \{||\};\ \textit{GNil};\ \textit{DNil} \vdash s[bv::=b']_{sb}[x::=v]_{sv} \Leftarrow t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
⟨*proof*⟩

lemma *fundef-simple-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **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 ⊢ $s[x::=v]_{sv}$ ⇐ $t[x::=v]_{\tau v}$
⟨*proof*⟩

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 ⊢ $s[bv::=b']_{sb}[x::=v]_{sv}$ ⇐ $t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
⟨*proof*⟩

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* : ($\tau 1'[x1::=v]_{\tau v}$) = ($s1'[x1::=v]_{sv}$) *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*
⟨*proof*⟩

lemma *fresh-subst-v-subst-b*:
fixes *x2*::*x* **and** *tm*::′*a*::{*has-subst-v*,*has-subst-b*} **and** *x*::*x*
assumes *supp tm* ⊆ { *atom bv*, *atom x* } **and** *atom x2* ♯ *v*
shows *atom x2* ♯ $tm[bv::=b]_b[x::=v]_v$
⟨*proof*⟩

lemma *preservation-poly-app*:
assumes
Some (*AF-fundef f* (*AF-fun-typ-some bv1* (*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-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*
⟨*proof*⟩

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*⟩

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*

$s' \Leftarrow \tau$
⟨*proof*⟩

lemma *check-s-eq*:
assumes Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *LET* x = (*AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*))) *IN* s' $\Leftarrow \tau$
shows Θ; Φ; $\{||\}$; *GNil*; Δ $\vdash$ *LET* x = (*AE-val* (*V-lit* (*if* (*n1* = *n2*) *then L-true else L-false*))) *IN* $s' \Leftarrow \tau$
⟨*proof*⟩

16.2.2 Operators

lemma *preservation-plus*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta$, *LET* x = (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN* s' $\rangle \Leftarrow \tau$

shows Θ; Φ; Δ $\vdash \langle\ \delta$, *LET* x = (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *IN* s' $\rangle \Leftarrow \tau$
⟨*proof*⟩

lemma *preservation-leq*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta$, *AS-let* x (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) s' $\rangle \Leftarrow \tau$
shows Θ; Φ; Δ $\vdash \langle\ \delta$, *AS-let* x (*AE-val* (*V-lit* (((*if* (*n1* $\leq$ *n2*) *then L-true else L-false*))))) s' $\rangle \Leftarrow \tau$
⟨*proof*⟩

lemma *preservation-eq*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta$, *AS-let* x (*AE-op Eq* (*V-lit* (*n1*)) (*V-lit* (*n2*))) s' $\rangle \Leftarrow \tau$
shows Θ; Φ; Δ $\vdash \langle\ \delta$, *AS-let* x (*AE-val* (*V-lit* (((*if* (*n1* = *n2*) *then L-true else L-false*))))) s' $\rangle \Leftarrow \tau$
⟨*proof*⟩

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* $\sharp$ *v* $\wedge$ *atom x* $\sharp$ *v*
shows $s[x{::}{=}v]_{sv}$ = $sa[xa{::}{=}v]_{sv}$
⟨*proof*⟩

lemma *check-let-val*:
fixes *v*::*v* **and** *s*::*s*
shows Θ ; Φ ; *B* ; *G* ; Δ $\vdash$ *ss* $\Leftarrow \tau \Longrightarrow$ *B* = $\{||\}$ $\Longrightarrow$ *G* = *GNil* $\Longrightarrow$
ss = *AS-let* x (*AE-val v*) *s* $\vee$ *ss* = *AS-let2* x *t* (*AS-val v*) *s* $\Longrightarrow$ Θ; Φ; $\{||\}$; *GNil*; Δ $\vdash$ ($s[x{::}{=}v]_{sv}$) $\Leftarrow \tau$ **and**
check-branch-s Θ Φ $\mathcal{B}$ *GNil* Δ *tid dc const v cs* $\tau \Longrightarrow$ *True* **and**
check-branch-list Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *True*
⟨*proof*⟩

lemma *preservation-let-val*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta$, *AS-let* x (*AE-val v*) *s* $\rangle \Leftarrow \tau \vee \Theta$; Φ; $\Delta \vdash \langle\ \delta$, *AS-let2* x *t* (*AS-val v*) *s* $\rangle \Leftarrow \tau$ (**is** *?A* $\vee$ *?B*)
shows $\exists\,\Delta'$. Θ; Φ; Δ' $\vdash \langle\ \delta$, $s[x{::}{=}v]_{sv}$ $\rangle \Leftarrow \tau$ $\wedge$ $\Delta \sqsubseteq \Delta'$
⟨*proof*⟩

lemma *check-s-fst-snd*:
assumes *fst-snd* = *AE-fst* $\wedge$ *v*=*v1* $\vee$ *fst-snd* = *AE-snd* $\wedge$ *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*⟩

lemma *preservation-fst-snd*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta$, *LET x = (fst-snd (V-pair v1 v2)) IN s′* $\rangle \Leftarrow \tau$ **and**
fst-snd = AE-fst $\wedge$ *v=v1* $\vee$ *fst-snd = AE-snd* $\wedge$ *v=v2*
shows $\exists\,\Delta'$. Θ; Φ; $\Delta \vdash \langle\ \delta$, *LET x = (AE-val v) IN s′* $\rangle \Leftarrow \tau\ \ \wedge \Delta \sqsubseteq \Delta'$
⟨*proof*⟩

inductive-cases *check-branch-s-elims2*[*elim*!]:
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *cons* ; *const* ; $v \vdash$ *cs* $\Leftarrow \tau$

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 ; $\mathcal{B}$; $\Gamma \vdash$ $\{\!|$ *z* : *b-of t* | *c-of t z* $|\!\}$ $\lesssim$ $\{\!|$ *z* : *b-of t* | *c IMP c-of t z* $|\!\}$
⟨*proof*⟩

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* $\sharp$ *t*
shows Θ ; $\mathcal{B}$; $\Gamma \vdash$ *t* $\lesssim$ $\{\!|$ *z* : *b-of t* | *c IMP c-of t z* $|\!\}$
⟨*proof*⟩

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*⟩

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*⟩

lemma *preservation-if*:
assumes Θ; Φ; $\Delta \vdash \langle\ \delta$, *IF (V-lit ll) THEN s1 ELSE s2* $\rangle \Leftarrow \tau$ **and**
ll = L-true $\wedge$ *s = s1* $\vee$ *ll = L-false* $\wedge$ *s = s2*
shows Θ; Φ; $\Delta \vdash \langle\delta,\ s\rangle \Leftarrow \tau\ \ \wedge$ *setD* $\Delta \subseteq$ *setD* Δ
⟨*proof*⟩

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⟩

lemma *subtype-conj*:
 assumes $\Theta ; \mathcal{B} ; GNil \vdash t \lesssim \{\!\mid z : b \mid c1 \mid\!\}$ **and** $\Theta ; \mathcal{B} ; GNil \vdash t \lesssim \{\!\mid z : b \mid c2 \mid\!\}$
 shows $\Theta ; \mathcal{B} ; GNil \vdash \{\!\mid z : b \mid c\text{-}of\ t\ z \mid\!\} \lesssim \{\!\mid z : b \mid c1\ AND\ c2 \mid\!\}$
⟨proof⟩

lemma *infer-v-conj*:
 assumes $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{\!\mid z : b \mid c1 \mid\!\}$ **and** $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{\!\mid z : b \mid c2 \mid\!\}$
 shows $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{\!\mid z : b \mid c1\ AND\ c2 \mid\!\}$
⟨proof⟩

lemma *wfG-conj*:
 fixes *c1*::*c*
 assumes $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c1\ AND\ c2) \#_\Gamma \Gamma$
 shows $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c1) \#_\Gamma \Gamma$
⟨proof⟩

lemma *check-match*:
 fixes *s'*::*s* **and** *s*::*s* **and** *css*::*branch-list* **and** *cs*::*branch-s*
 shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau \Longrightarrow True$ **and**
  $\Theta ; \Phi ; B ; G ; \Delta ; tid ; dc ; const ; vcons \vdash cs \Leftarrow \tau \Longrightarrow$
   $vcons = V\text{-}cons\ tid\ dc\ v \Longrightarrow B = \{||\} \Longrightarrow G = GNil \Longrightarrow cs = (dc\ x' \Rightarrow s') \Longrightarrow$
   $\Theta ; \{||\} ; GNil \vdash v \Leftarrow const \Longrightarrow$
   $\Theta; \Phi; \{||\}; GNil; \Delta \vdash s'[x'::=v]_{sv} \Leftarrow \tau$ **and**
  $\Theta ; \Phi ; B ; G ; \Delta ; tid ; dclist ; vcons \vdash css \Leftarrow \tau \Longrightarrow distinct\ (map\ fst\ dclist) \Longrightarrow$
   $vcons = V\text{-}cons\ tid\ dc\ v \Longrightarrow B = \{||\} \Longrightarrow (dc, const) \in set\ dclist \Longrightarrow G = GNil \Longrightarrow$
   $Some\ (AS\text{-}branch\ dc\ x'\ s') = lookup\text{-}branch\ dc\ css \Longrightarrow \Theta ; \{||\} ; GNil \vdash v \Leftarrow const \Longrightarrow$
   $\Theta; \Phi; \{||\}; GNil; \Delta \vdash s'[x'::=v]_{sv} \Leftarrow \tau$
⟨proof⟩

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 $\Theta ; \{||\} ; GNil \vdash \{\!\mid z : B\text{-}unit \mid TRUE \mid\!\} \lesssim \tau'$ **and** $\Theta ; \{||\} ; GNil \vdash_{wf} \Delta$ **and** $\Theta \vdash_{wf} \Phi$ **and** $\Theta ; \{||\} \vdash_{wf} G$
 shows ‹ $\Theta ; \Phi ; \{||\} ; G ; \Delta \vdash [[\ L\text{-}unit\ ]^v]^s \Leftarrow \tau'$›
⟨proof⟩

lemma *preservation-var*:
 shows $\Theta; \Phi; \{||\}; GNil; \Delta \vdash VAR\ u : \tau' = v\ IN\ s \Leftarrow \tau \Longrightarrow \Theta \vdash \delta \sim \Delta \Longrightarrow atom\ u \sharp \delta \Longrightarrow atom\ u \sharp \Delta \Longrightarrow$
  $\Theta; \Phi; \{||\}; GNil; (u,\tau')\#_\Delta\Delta \vdash s \Leftarrow \tau \wedge \Theta \vdash (u,v)\#\delta \sim (u,\tau')\#_\Delta\Delta$
  and
  $check\text{-}branch\text{-}s\ \Theta\ \Phi\ \{||\}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \Longrightarrow True$ **and**
  $check\text{-}branch\text{-}list\ \Theta\ \Phi\ \{||\}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \Longrightarrow True$
⟨proof⟩

lemma *check-while*:
 shows $\Theta; \Phi; \{||\}; GNil; \Delta \vdash WHILE\ s1\ DO\ \{\ s2\ \} \Leftarrow \tau \Longrightarrow atom\ x \sharp (s1, s2) \Longrightarrow atom\ z' \sharp x \Longrightarrow$
  $\Theta; \Phi; \{||\}; GNil; \Delta \vdash LET\ x : (\{\!\mid z' : B\text{-}bool \mid TRUE \mid\!\}) = s1\ IN\ (IF\ (V\text{-}var\ x)\ THEN\ (s2\ ;;$

(*WHILE s1 DO* {*s2*}))
ELSE ([*V-lit L-unit*]^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*⟩

lemma *check-s-narrow*:
fixes *s*::*s* **and** *x*::*x*
assumes *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *c*, τ, *s*) **and** Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ $\vdash s \Leftarrow \tau$ **and**
Θ ; $\mathcal{B}$; Γ $\models c$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash s \Leftarrow \tau$
⟨*proof*⟩

lemma *check-assert-s*:
fixes *s*::*s* **and** *x*::*x*
assumes Θ; Φ; {||}; *GNil*; Δ $\vdash$ *AS-assert c s* $\Leftarrow \tau$
shows Θ; Φ; {||}; *GNil*; Δ $\vdash$ *s* $\Leftarrow \tau \wedge \Theta$; {||} ; *GNil* $\models c$
⟨*proof*⟩

lemma *infer-v-pair2I*:
atom z $\sharp$ (*v1*, *v2*) $\Longrightarrow$
atom z $\sharp$ (Θ, $\mathcal{B}$, Γ) $\Longrightarrow$
Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v1* $\Rightarrow$ *t1* $\Longrightarrow$
Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v2* $\Rightarrow$ *t2* $\Longrightarrow$
b1 = *b-of t1* $\Longrightarrow$ *b2* = *b-of t2* $\Longrightarrow$
Θ ; $\mathcal{B}$; $\Gamma \vdash$ [*v1* , *v2*]^v $\Rightarrow$ ⦃ *z* : [*b1* , *b2*]^b | [[*z*]^v]^{ce} == [[*v1* , *v2*]^v]^{ce} ⦄
⟨*proof*⟩

16.2.5 Main Lemma

lemma *preservation*:
assumes $\Phi \vdash \langle\delta, s\rangle \longrightarrow \langle\delta', s'\rangle$ **and** Θ; Φ; $\Delta \vdash \langle\delta, s\rangle \Leftarrow \tau$
shows $\exists\Delta'$. Θ; Φ; $\Delta' \vdash \langle\delta', s'\rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$
⟨*proof*⟩

lemma *preservation-many*:
assumes $\Phi \vdash \langle\delta, s\rangle \longrightarrow^* \langle \delta' , s' \rangle$
shows Θ; Φ; $\Delta \vdash \langle\delta, s\rangle \Leftarrow \tau \Longrightarrow \exists\Delta'$. Θ; Φ; $\Delta' \vdash \langle \delta' , s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$
⟨*proof*⟩

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 $\vdash$ *LET x* = (*AE-op opp v1 v2*) *IN s* $\Leftarrow \tau$ **and** *supp* (*LET x* = (*AE-op opp v1 v2*) *IN s*) $\subseteq$ *atom*'*fst*'*setD* Δ
shows $\exists$ *z b c*. Θ; Φ; {||}; Γ; $\Delta \vdash$ (*AE-op opp v1 v2*) $\Rightarrow$ ⦃*z*:*b*|*c*⦄
⟨*proof*⟩

lemma *infer-pair*:
assumes Θ ; *B*; $\Gamma \vdash$ *v* $\Rightarrow$ ⦃ *z* : *B-pair b1 b2* | *c* ⦄ **and** *supp v* = {}

obtains *v1* **and** *v2* **where** $v = V\text{-}pair\ v1\ v2$
⟨*proof*⟩

lemma *progress-fst*:
assumes $\Theta; \Phi; \{||\}; \Gamma; \Delta \ \vdash LET\ x = (AE\text{-}fst\ v)\ IN\ s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$ **and**
$supp\ (LET\ x = (AE\text{-}fst\ v)\ IN\ s) \ \subseteq atom`fst`setD\ \Delta$
shows $\exists \delta'\ s'.\ \Phi \ \vdash \langle\ \delta\ ,\ LET\ x = (AE\text{-}fst\ v)\ IN\ s\ \rangle \longrightarrow \langle \delta', s' \rangle$
⟨*proof*⟩

lemma *progress-let*:
assumes $\Theta; \Phi; \{||\}; \Gamma; \Delta \vdash LET\ x = e\ IN\ s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$ **and**
$supp\ (LET\ x = e\ IN\ s) \subseteq atom\ `\ fst\ `\ setD\ \Delta$ **and** $sble\ \Theta\ \Gamma$
shows $\exists \delta'\ s'.\ \Phi \ \vdash \langle\ \delta\ ,\ LET\ x = e\ IN\ s \rangle \longrightarrow \langle\ \delta'\ ,\ s' \rangle$
⟨*proof*⟩

lemma *check-css-lookup-branch-exist*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows
$\Theta; \Phi; B; G; \Delta \vdash \ s \Leftarrow \tau \Longrightarrow True$ **and**
$check\text{-}branch\text{-}s\ \Theta\ \Phi \ \ \{||\}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \Longrightarrow True$ **and**
$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist; v \vdash css \Leftarrow \tau \Longrightarrow (dc, t) \in set\ dclist \Longrightarrow$
$\exists x'\ s'.\ Some\ (AS\text{-}branch\ dc\ x'\ s') = lookup\text{-}branch\ dc\ css$
⟨*proof*⟩

lemma *progress-aux*:
shows $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s \Leftarrow \tau \Longrightarrow \mathcal{B} = \{||\} \Longrightarrow \ sble\ \Theta\ \Gamma \Longrightarrow supp\ s \subseteq atom\ `\ fst\ `\ setD\ \Delta \Longrightarrow$
$\Theta \vdash \delta \sim \Delta \Longrightarrow$
$(\exists v.\ s = [v]^s) \vee (\exists \delta'\ s'.\ \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$ **and**
$\Theta; \Phi; \{||\}; \Gamma; \Delta; tid; dc; const; v2 \vdash cs \Leftarrow \tau \ \Longrightarrow supp\ cs = \{\} \Longrightarrow True$
$\Theta; \Phi; \{||\}; \Gamma; \Delta; tid; dclist; v2 \vdash css \Leftarrow \tau \Longrightarrow supp\ css = \{\} \Longrightarrow True$
⟨*proof*⟩

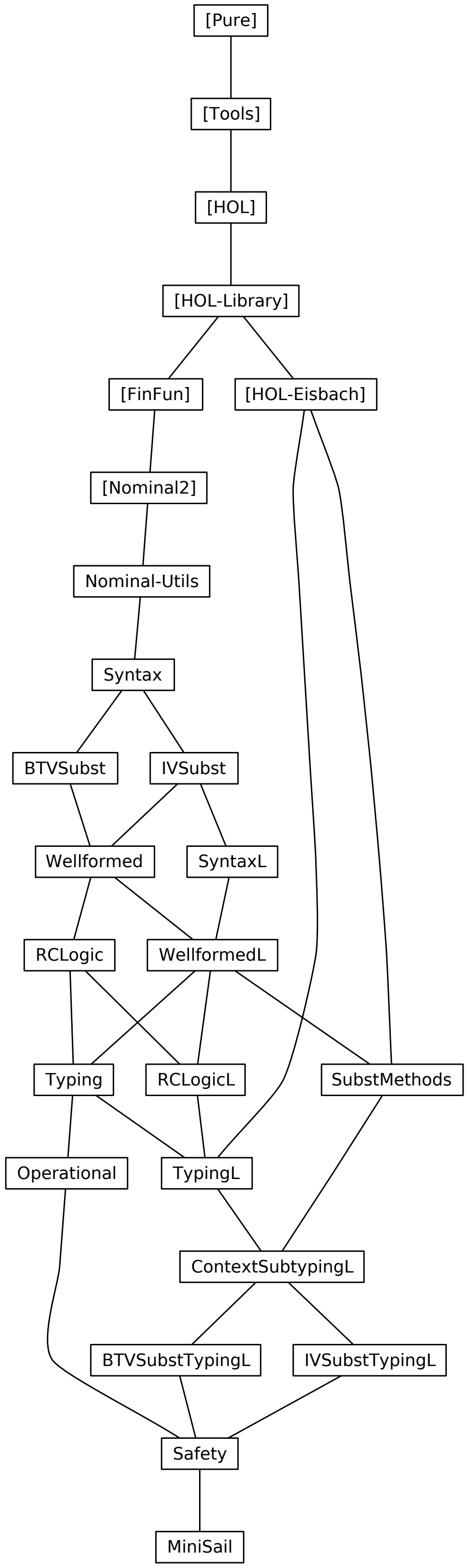
lemma *progress*:
assumes $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$
shows $(\exists v.\ s = [v]^s) \vee (\exists \delta'\ s'.\ \Phi \ \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$
⟨*proof*⟩

16.4 Safety

lemma *safety-stmt*:
assumes $\Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$ **and** $\Theta; \Phi; \Delta \vdash \ \langle \delta, s \rangle \Leftarrow \tau$
shows $(\exists v.\ s' = [v]^s) \vee (\exists \delta''\ s''.\ \Phi \ \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$
⟨*proof*⟩

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 $(\exists v.\ s' = [v]^s) \vee (\exists \delta''\ s''.\ \Phi \ \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$
⟨*proof*⟩

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.