A Formalization of the First Order Theory of Rewriting (FORT) *

Alexander Lochmann Bertram Felgenhauer

March 17, 2025

Abstract

The first-order theory of rewriting (FORT) is a decidable theory for linear variable-separated rewrite systems. The decision procedure is based on tree automata technique and an inference system presented in [4]. This AFP entry provides a formalization of the underlying decision procedure. Moreover it allows to generate a function that can verify each inference step via the code generation facility of Isabelle/HOL.

Additionally it contains the specification of a certificate language (that allows to state proofs in FORT) and a formalized function that allows to verify the validity of the proof. This gives software tool authors, that implement the decision procedure, the possibility to verify their output.

Contents

*Supported by FWF (Austrian Science Fund) project P30301.

1 Introduction

The first-order theory of rewriting (FORT) is a fragment of first-order predicate logic with predefined predicates. The language allows to state many interesting properties of term rewrite systems and is decidable for left-linear right-ground systems. This was proven by Dauchet and Tison [2].

In this AFP entry we provide a formalized proof of an improved decision procedure for the first-order theory of rewriting. We introduce basic definitions to represent the rewrite semantics and connect FORT to first-order logic via the AFP entry "First-Order Logic According to Fitting" by Stefan Berghofer [1]. To prove the decidability and more importantly to allow code generation a relation between formulas in FORT and regular tree language is constructed. The tree language contains all witnesses of free variables satisfying the formula, details can be found in [3].

Moreover we present a certificate language which is rich enough to express the various au- tomata operations in decision procedures for the first-order theory of rewrit- ing as well as numerous predicate symbols that may appear in formulas in this theory, for more details see [4].

theory *Utils*
imports *Regular-Tree-Relations.Term-Context*
Regular-Tree-Relations.FSet-Utils
begin

1.1 Misc

definition *funas-trs* $\mathcal{R} = \bigcup\ ((\lambda\ (s,\ t).\ \textit{funas-term}\ s \cup \textit{funas-term}\ t)\ `\ \mathcal{R})$

fun *linear-term* :: ('*f*, '*v*) *term* ⇒ *bool* **where**
linear-term (*Var* -) = *True* |
linear-term (*Fun* - *ts*) = (*is-partition* (*map vars-term ts*) ∧ (∀ *t*∈*set ts*. *linear-term t*))

fun *vars-term-list* :: ('*f*, '*v*) *term* ⇒ '*v list* **where**
vars-term-list (*Var x*) = [*x*] |
vars-term-list (*Fun* - *ts*) = *concat* (*map vars-term-list ts*)

fun *varposs* :: ('*f*, '*v*) *term* ⇒ *pos set* **where**
varposs (*Var x*) = {[]} |
varposs (*Fun f ts*) = (⋃ *i*<*length ts*. {*i* # *p* | *p*. *p* ∈ *varposs* (*ts* ! *i*)})

abbreviation *poss-args f ts* ≡ *map2* (λ *i t*. *map* ((#) *i*) (*f t*)) ([*0* ..< *length ts*]) *ts*

fun *varposs-list* :: ('*f*, '*v*) *term* ⇒ *pos list* **where**
varposs-list (*Var x*) = [[]] |
varposs-list (*Fun f ts*) = *concat* (*poss-args varposs-list ts*)

fun *concat-index-split* **where**
concat-index-split (*o-idx*, *i-idx*) (*x* # *xs*) =
(*if i-idx* < *length x*
then (*o-idx*, *i-idx*)
else concat-index-split (*Suc o-idx*, *i-idx* − *length x*) *xs*)

inductive-set *trancl-list* **for** $\mathcal{R}$ **where**
base[*intro*, *Pure.intro*] : *length xs* = *length ys* ⟹
(∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}$) ⟹ (*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$
| *list-trancl* [*Pure.intro*]: (*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$ ⟹ *i* < *length ys* ⟹ (*ys* ! *i*, *z*) ∈ $\mathcal{R}$ ⟹
(*xs*, *ys*[*i* := *z*]) ∈ *trancl-list* $\mathcal{R}$

lemma *sorted-append-bigger*:
sorted xs ⟹ ∀ *x* ∈ *set xs*. *x* ≤ *y* ⟹ *sorted* (*xs* @ [*y*])
proof (*induct xs*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs*)
then have *s*: *sorted xs* **by** (*cases xs*) *simp-all*
from *Cons* **have** *a*: ∀ *x*∈*set xs*. *x* ≤ *y* **by** *simp*
from *Cons*(*1*)[*OF s a*] *Cons*(*2*−) **show** *?case* **by** (*cases xs*) *simp-all*
qed

lemma *find-SomeD*:
List.find P xs = *Some x* ⟹ *P x*

List.find P xs = Some x ⟹ x∈set xs
by (*auto simp add*: *find-Some-iff*)

lemma *sum-list-replicate-length'* [*simp*]:
sum-list (*replicate n* (*Suc 0*)) = *n*
by (*induct n*) *simp-all*

lemma *arg-subteq* [*simp*]:
assumes *t* ∈ *set ts* **shows** *Fun f ts* ⊵ *t*
using *assms* **by** *auto*

lemma *finite-funas-term*: *finite* (*funas-term s*)
by (*induct s*) *auto*

lemma *finite-funas-trs*:
finite $\mathcal{R}$ ⟹ *finite* (*funas-trs* $\mathcal{R}$)
by (*induct rule*: *finite.induct*) (*auto simp*: *finite-funas-term funas-trs-def*)

fun *subterms* **where**
subterms (*Var x*) = {*Var x*}|
subterms (*Fun f ts*) = {*Fun f ts*} ∪ (⋃ (*subterms* ‘ *set ts*))

lemma *finite-subterms-fun*: *finite* (*subterms s*)
by (*induct s*) *auto*

lemma *subterms-supteq-conv*: *t* ∈ *subterms s* ⟷ *s* ⊵ *t*
by (*induct s*) (*auto elim*: *supteq.cases*)

lemma *set-all-subteq-subterms*:
subterms s = {*t*. *s* ⊵ *t*}
using *subterms-supteq-conv* **by** *auto*

lemma *finite-subterms*: *finite* {*t*. *s* ⊵ *t*}
unfolding *set-all-subteq-subterms*[*symmetric*]
by (*simp add*: *finite-subterms-fun*)

lemma *finite-strict-subterms*: *finite* {*t*. *s* ▷ *t*}
by (*intro finite-subset*[*OF - finite-subterms*]) *auto*

lemma *finite-UN-I2*:
finite A ⟹ (∀ *B* ∈ *A*. *finite B*) ⟹ *finite* (⋃ *A*)
by *blast*

lemma *root-substerms-funas-term*:
the ‘ (*root* ‘ (*subterms s*) − {*None*}) = *funas-term s* (**is** *?Ls* = *?Rs*)
proof −
thm *subterms.induct*
{fix *x* **assume** *x* ∈ *?Ls* **then have** *x* ∈ *?Rs*
proof (*induct s arbitrary*: *x*)

```
    case (Fun f ts)
    then show ?case
      by auto (metis DiffI Fun.hyps imageI option.distinct(1) singletonD)
  qed auto}
 moreover
 {fix g assume g ∈ ?Rs then have g ∈ ?Ls
  proof (induct s arbitrary: g)
    case (Fun f ts)
    from Fun(2) consider g = (f, length ts) | ∃ t ∈ set ts. g ∈ funas-term t
      by (force simp: in-set-conv-nth)
    then show ?case
    proof cases
      case 1 then show ?thesis
        by (auto simp: image-iff intro: bexI[of - Some (f, length ts)])
    next
      case 2
      then obtain t where wit: t ∈ set ts g ∈ funas-term t by blast
      have g ∈ the ` (root ` subterms t − {None}) using Fun(1)[OF wit] .
      then show ?thesis using wit(1)
        by (auto simp: image-iff)
    qed
  qed auto}
 ultimately show ?thesis by auto
qed

lemma root-substerms-funas-term-set:
 the ` (root ` ⋃ (subterms ` R) − {None}) = ⋃ (funas-term ` R)
 using root-substerms-funas-term
 by fastforce


lemma subst-merge:
 assumes part: is-partition (map vars-term ts)
 shows ∃σ. ∀ i<length ts. ∀ x∈vars-term (ts ! i). σ x = τ i x
proof −
 let ?τ = map τ [0 ..< length ts]
 let ?σ = fun-merge ?τ (map vars-term ts)
 show ?thesis
   by (rule exI[of - ?σ], intro allI impI ballI,
     insert fun-merge-part[OF part, of - - ?τ], auto)
qed


lemma rel-comp-empty-trancl-simp: R O R = {} ⟹ R⁺ = R
 by (metis O-assoc relcomp-empty2 sup-bot-right trancl-unfold trancl-unfold-right)

lemma choice-nat:
 assumes ∀ i<n. ∃ x. P x i
 shows ∃ f. ∀ x<n. P (f x) x using assms
```

proof −
from *assms* **have** ∀ *i*. ∃ *x*. *i* < *n* ⟶ *P x i* **by** *simp*
from *choice*[*OF this*] **show** *?thesis* **by** *auto*
qed

lemma *subseteq-set-conv-nth*:
(∀ *i* < *length ss*. *ss* ! *i* ∈ *T*) ⟷ *set ss* ⊆ *T*
by (*metis in-set-conv-nth subset-code*(*1*))

lemma *singelton-trancl* [*simp*]: {*a*}$^+$ = {*a*}
using *tranclD tranclD2* **by** *fastforce*

context
includes *fset.lifting*
begin
lemmas *frelcomp-empty-ftrancl-simp* = *rel-comp-empty-trancl-simp* [*Transfer.transferred*]
lemmas *in-fset-idx* = *in-set-idx* [*Transfer.transferred*]
lemmas *fsubseteq-fset-conv-nth* = *subseteq-set-conv-nth* [*Transfer.transferred*]
lemmas *singelton-ftrancl* [*simp*] = *singelton-trancl* [*Transfer.transferred*]
end

lemma *set-take-nth*:
assumes *x* ∈ *set* (*take i xs*)
shows ∃ *j* < *length xs*. *j* < *i* ∧ *xs* ! *j* = *x* **using** *assms*
by (*metis in-set-conv-nth length-take min-less-iff-conj nth-take*)

lemma *nth-sum-listI*:
assumes *length xs* = *length ys*
and ∀ *i* < *length xs*. *xs* ! *i* = *ys* ! *i*
shows *sum-list xs* = *sum-list ys*
using *assms nth-equalityI* **by** *blast*

lemma *concat-nth-length*:
i < *length uss* ⟹ *j* < *length* (*uss* ! *i*) ⟹
sum-list (*map length* (*take i uss*)) + *j* < *length* (*concat uss*)
by (*induct uss arbitrary*: *i j*) (*simp*, *case-tac i*, *auto*)

lemma *sum-list-1-E* [*elim*]:
assumes *sum-list xs* = *Suc 0*
obtains *i* **where** *i* < *length xs xs* ! *i* = *Suc 0* ∀ *j* < *length xs*. *j* ≠ *i* ⟶ *xs* ! *j*
= *0*
proof −
have ∃ *i* < *length xs*. *xs* ! *i* = *Suc 0* ∧ (∀ *j* < *length xs*. *j* ≠ *i* ⟶ *xs* ! *j* = *0*)
using *assms*
proof (*induct xs*)
case (*Cons a xs*) **show** *?case*
proof (*cases a*)
case [*simp*]: *0*

obtain i **where** $i < length\ xs\ xs\ !\ i = Suc\ 0\ (\forall\ j < length\ xs.\ j \neq i \longrightarrow xs\ !\ j = 0)$
using *Cons* **by** *auto*
then show *?thesis* **using** *less-Suc-eq-0-disj*
by (*intro exI*[*of* - *Suc i*]) *auto*
next
case (*Suc nat*) **then show** *?thesis* **using** *Cons* **by** *auto*
qed
qed *auto*
then show $(\bigwedge i.\ i < length\ xs \Longrightarrow xs\ !\ i = Suc\ 0 \Longrightarrow \forall j{<}length\ xs.\ j \neq i \longrightarrow xs\ !\ j = 0 \Longrightarrow thesis) \Longrightarrow thesis$
by *blast*
qed

lemma *nth-equalityE*:
$xs = ys \Longrightarrow (length\ xs = length\ ys \Longrightarrow (\bigwedge i.\ i < length\ xs \Longrightarrow xs\ !\ i = ys\ !\ i) \Longrightarrow P) \Longrightarrow P$
by *simp*

lemma *map-cons-presv-distinct*:
$distinct\ t \Longrightarrow distinct\ (map\ ((\#)\ i)\ t)$
by (*simp add*: *distinct-conv-nth*)

lemma *concat-nth-nthI*:
assumes $length\ ss = length\ ts\ \forall\ i < length\ ts.\ length\ (ss\ !\ i) = length\ (ts\ !\ i)$
and $\forall\ i < length\ ts.\ \forall\ j < length\ (ts\ !\ i).\ P\ (ss\ !\ i\ !\ j)\ (ts\ !\ i\ !\ j)$
shows $\forall\ i < length\ (concat\ ts).\ P\ (concat\ ss\ !\ i)\ (concat\ ts\ !\ i)$
using *assms* **by** (*metis nth-concat-two-lists*)

lemma *last-nthI*:
assumes $i < length\ ts\ \neg\ i < length\ ts - Suc\ 0$
shows $ts\ !\ i = last\ ts$ **using** *assms*
by (*induct ts arbitrary*: *i*)
(*auto*, *metis last-conv-nth length-0-conv less-antisym nth-Cons'*)

lemma *trancl-list-appendI* [*simp*, *intro*]:
$(xs, ys) \in trancl\text{-}list\ \mathcal{R} \Longrightarrow (x, y) \in \mathcal{R} \Longrightarrow (x\ \#\ xs, y\ \#\ ys) \in trancl\text{-}list\ \mathcal{R}$
proof (*induct rule*: *trancl-list.induct*)
case (*base xs ys*)
then show *?case* **using** *less-Suc-eq-0-disj*
by (*intro trancl-list.base*) *auto*
next
case (*list-trancl xs ys i z*)
from *list-trancl*(*3*) **have** $*$: $y\ \#\ ys[i := z] = (y\ \#\ ys)[Suc\ i := z]$ **by** *auto*
show *?case* **using** *list-trancl* **unfolding** $*$
by (*intro trancl-list.list-trancl*) *auto*

qed

lemma *trancl-list-append-tranclI* [*intro*]:
(*x*, *y*) ∈ $\mathcal{R}^+$ ⟹ (*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$ ⟹ (*x* # *xs*, *y* # *ys*) ∈ *trancl-list* $\mathcal{R}$
proof (*induct rule*: *trancl.induct*)
case (*trancl-into-trancl a b c*)
then have (*a* # *xs*, *b* # *ys*) ∈ *trancl-list* $\mathcal{R}$ **by** *auto*
from *trancl-list.list-trancl*[*OF this*, *of 0 c*]
show *?case* **using** *trancl-into-trancl*(*3*)
by *auto*
qed *auto*

lemma *trancl-list-conv*:
(*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$ ⟷ *length xs* = *length ys* ∧ (∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}^+$) (**is** *?Ls* ⟷ *?Rs*)
proof
assume *?Ls* **then show** *?Rs*
proof (*induct*)
case (*list-trancl xs ys i z*)
then show *?case*
by *auto* (*metis nth-list-update trancl.trancl-into-trancl*)
qed *auto*
next
assume *?Rs* **then show** *?Ls*
proof (*induct ys arbitrary*: *xs*)
case *Nil*
then show *?case* **by** (*cases xs*) *auto*
next
case (*Cons y ys*)
from *Cons*(*2*) **obtain** *x xs'* **where** ∗: *xs* = *x* # *xs'* **and**
inv: (*x*, *y*) ∈ $\mathcal{R}^+$
by (*cases xs*) *auto*
show *?case* **using** *Cons*(*1*)[*of tl xs*] *Cons*(*2*) **unfolding** ∗
by (*intro trancl-list-append-tranclI*[*OF inv*]) *force*
qed
qed

lemma *trancl-list-induct* [*consumes 2*, *case-names base step*]:
assumes *length ss* = *length ts* ∀ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) ∈ $\mathcal{R}^+$
and ⋀*xs ys*. *length xs* = *length ys* ⟹ ∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}$ ⟹ *P xs ys*
and ⋀*xs ys i z*. *length xs* = *length ys* ⟹ ∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}^+$ ⟹ *P xs ys*
⟹ *i* < *length ys* ⟹ (*ys* ! *i*, *z*) ∈ $\mathcal{R}$ ⟹ *P xs* (*ys*[*i* := *z*])
shows *P ss ts* **using** *assms*
by (*intro trancl-list.induct*[*of ss ts* $\mathcal{R}$ *P*]) (*auto simp*: *trancl-list-conv*)

lemma *swap-trancl*:

$(prod.swap\ `\ R)^+ = prod.swap\ `\ (R^+)$
proof −
have [*simp*]: $prod.swap\ `\ X = X^{-1}$ **for** X **by** *auto*
show *?thesis* **by** (*simp add*: *trancl-converse*)
qed

lemma *swap-rtrancl*:
$(prod.swap\ `\ R)^* = prod.swap\ `\ (R^*)$
proof −
have [*simp*]: $prod.swap\ `\ X = X^{-1}$ **for** X **by** *auto*
show *?thesis* **by** (*simp add*: *rtrancl-converse*)
qed

lemma *Restr-simps*:
$R \subseteq X \times X \Longrightarrow Restr\ (R^+)\ X = R^+$
$R \subseteq X \times X \Longrightarrow Restr\ Id\ X\ O\ R = R$
$R \subseteq X \times X \Longrightarrow R\ O\ Restr\ Id\ X = R$
$R \subseteq X \times X \Longrightarrow S \subseteq X \times X \Longrightarrow Restr\ (R\ O\ S)\ X = R\ O\ S$
$R \subseteq X \times X \Longrightarrow R^+ \subseteq X \times X$
subgoal using *trancl-mono-set*[*of R X* × *X*] **by** (*auto simp*: *trancl-full-on*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *trancl-subset-Sigma* **.**
done

lemma *Restr-tracl-comp-simps*:
$\mathcal{R} \subseteq X \times X \Longrightarrow \mathcal{L} \subseteq X \times X \Longrightarrow \mathcal{L}^+\ O\ \mathcal{R} \subseteq X \times X$
$\mathcal{R} \subseteq X \times X \Longrightarrow \mathcal{L} \subseteq X \times X \Longrightarrow \mathcal{L}\ O\ \mathcal{R}^+ \subseteq X \times X$
$\mathcal{R} \subseteq X \times X \Longrightarrow \mathcal{L} \subseteq X \times X \Longrightarrow \mathcal{L}^+\ O\ \mathcal{R}\ O\ \mathcal{L}^+ \subseteq X \times X$
by (*auto dest*: *subsetD*[*OF Restr-simps*(*5*)[*of* $\mathcal{L}$]] *subsetD*[*OF Restr-simps*(*5*)[*of* $\mathcal{R}$]])

Conversions of the Nth function between lists and a spliting of the list into lists of lists

lemma *concat-index-split-mono-first-arg*:
$i < length\ (concat\ xs) \Longrightarrow o\text{-}idx \leq fst\ (concat\text{-}index\text{-}split\ (o\text{-}idx,\ i)\ xs)$
by (*induct xs arbitrary*: *o-idx i*) (*auto*, *metis Suc-leD add-diff-inverse-nat nat-add-left-cancel-less*)

lemma *concat-index-split-sound-fst-arg-aux*:
$i < length\ (concat\ xs) \Longrightarrow fst\ (concat\text{-}index\text{-}split\ (o\text{-}idx,\ i)\ xs) < length\ xs + o\text{-}idx$
by (*induct xs arbitrary*: *o-idx i*) (*auto*, *metis add-Suc-right add-diff-inverse-nat nat-add-left-cancel-less*)

lemma *concat-index-split-sound-fst-arg*:
$i < length\ (concat\ xs) \Longrightarrow fst\ (concat\text{-}index\text{-}split\ (0,\ i)\ xs) < length\ xs$
using *concat-index-split-sound-fst-arg-aux*[*of i xs 0*] **by** *auto*

lemma *concat-index-split-sound-snd-arg-aux*:
assumes $i < length\ (concat\ xs)$
shows $snd\ (concat\text{-}index\text{-}split\ (n, i)\ xs) < length\ (xs\ !\ (fst\ (concat\text{-}index\text{-}split\ (n, i)\ xs) - n))$ **using** *assms*
proof (*induct xs arbitrary*: *i n*)
case (*Cons x xs*)
show *?case* **proof** (*cases i < length x*)
case *False* **then have** *size*: $i - length\ x < length\ (concat\ xs)$
using *Cons*(*2*) *False* **by** *auto*
obtain *k j* **where** [*simp*]: $concat\text{-}index\text{-}split\ (Suc\ n, i - length\ x)\ xs = (k, j)$
using *old.prod.exhaust* **by** *blast*
show *?thesis* **using** *False Cons*(*1*)[*OF size*, *of Suc n*] *concat-index-split-mono-first-arg*[*OF size*, *of Suc n*]
by (*auto simp*: *nth-append*)
qed (*auto simp add*: *nth-append*)
qed *auto*

lemma *concat-index-split-sound-snd-arg*:
assumes $i < length\ (concat\ xs)$
shows $snd\ (concat\text{-}index\text{-}split\ (0, i)\ xs) < length\ (xs\ !\ fst\ (concat\text{-}index\text{-}split\ (0, i)\ xs))$
using *concat-index-split-sound-snd-arg-aux*[*OF assms*, *of 0*] **by** *auto*

lemma *reconstr-1d-concat-index-split*:
assumes $i < length\ (concat\ xs)$
shows $i = (\lambda\ (m, j).\ sum\text{-}list\ (map\ length\ (take\ (m - n)\ xs)) + j)\ (concat\text{-}index\text{-}split\ (n, i)\ xs)$ **using** *assms*
proof (*induct xs arbitrary*: *i n*)
case (*Cons x xs*) **show** *?case*
proof (*cases i < length x*)
case *False*
obtain *m k* **where** *res*: $concat\text{-}index\text{-}split\ (Suc\ n, i - length\ x)\ xs = (m, k)$
using *prod-decode-aux.cases* **by** *blast*
then have *unf-ind*: $concat\text{-}index\text{-}split\ (n, i)\ (x\ \#\ xs) = concat\text{-}index\text{-}split\ (Suc\ n, i - length\ x)\ xs$ **and**
size: $i - length\ x < length\ (concat\ xs)$ **using** *Cons*(*2*) *False* **by** *auto*
have $Suc\ n \leq m$ **using** *concat-index-split-mono-first-arg*[*OF size*, *of Suc n*] **by** (*auto simp*: *res*)
then have [*simp*]: $sum\text{-}list\ (map\ length\ (take\ (m - n)\ (x\ \#\ xs))) = sum\text{-}list\ (map\ length\ (take\ (m - Suc\ n)\ xs)) + length\ x$
by (*simp add*: *take-Cons'*)
show *?thesis* **using** *Cons*(*1*)[*OF size*, *of Suc n*] *False* **unfolding** *unf-ind res* **by** *auto*
qed *auto*
qed *auto*

lemma *concat-index-split-larger-lists* [*simp*]:
assumes $i < length\ (concat\ xs)$
shows $concat\text{-}index\text{-}split\ (n, i)\ (xs\ @\ ys) = concat\text{-}index\text{-}split\ (n, i)\ xs$ **using**

assms
by (*induct xs arbitrary*: *ys n i*) *auto*

lemma *concat-index-split-split-sound-aux*:
assumes $i < length\ (concat\ xs)$
shows $concat\ xs\ !\ i = (\lambda\ (k,\ j).\ xs\ !\ (k - n)\ !\ j)\ (concat\text{-}index\text{-}split\ (n,\ i)\ xs)$
using *assms*
proof (*induct xs arbitrary*: *i n*)
case (*Cons x xs*)
show *?case* **proof** (*cases* $i < length\ x$)
case *False* **then have** *size*: $i - length\ x < length\ (concat\ xs)$
using *Cons*(*2*) *False* **by** *auto*
obtain *k j* **where** [*simp*]: $concat\text{-}index\text{-}split\ (Suc\ n,\ i - length\ x)\ xs = (k,\ j)$
using *prod-decode-aux.cases* **by** *blast*
show *?thesis* **using** *False Cons*(*1*)[*OF size*, *of Suc n*]
using *concat-index-split-mono-first-arg*[*OF size*, *of Suc n*]
by (*auto simp*: *nth-append*)
qed (*auto simp add*: *nth-append*)
qed *auto*

lemma *concat-index-split-sound*:
assumes $i < length\ (concat\ xs)$
shows $concat\ xs\ !\ i = (\lambda\ (k,\ j).\ xs\ !\ k\ !\ j)\ (concat\text{-}index\text{-}split\ (0,\ i)\ xs)$
using *concat-index-split-split-sound-aux*[*OF assms*, *of 0*] **by** *auto*

lemma *concat-index-split-sound-bounds*:
assumes $i < length\ (concat\ xs)$ **and** $concat\text{-}index\text{-}split\ (0,\ i)\ xs = (m,\ n)$
shows $m < length\ xs\ n < length\ (xs\ !\ m)$
using *concat-index-split-sound-fst-arg*[*OF assms*(*1*)] *concat-index-split-sound-snd-arg*[*OF assms*(*1*)]
by (*auto simp*: *assms*(*2*))

lemma *concat-index-split-less-length-concat*:
assumes $i < length\ (concat\ xs)$ **and** $concat\text{-}index\text{-}split\ (0,\ i)\ xs = (m,\ n)$
shows $i = sum\text{-}list\ (map\ length\ (take\ m\ xs)) + n\ m < length\ xs\ n < length\ (xs\ !\ m)$
$concat\ xs\ !\ i = xs\ !\ m\ !\ n$
using *concat-index-split-sound*[*OF assms*(*1*)] *reconstr-1d-concat-index-split*[*OF assms*(*1*), *of 0*]
using *concat-index-split-sound-fst-arg*[*OF assms*(*1*)] *concat-index-split-sound-snd-arg*[*OF assms*(*1*)] *assms*(*2*)
by *auto*

lemma *nth-concat-split′*:
assumes $i < length\ (concat\ xs)$
obtains *j k* **where** $j < length\ xs\ k < length\ (xs\ !\ j)\ concat\ xs\ !\ i = xs\ !\ j\ !\ k\ i = sum\text{-}list\ (map\ length\ (take\ j\ xs)) + k$
using *concat-index-split-less-length-concat*[*OF assms*]
by (*meson old.prod.exhaust*)

lemma *sum-list-split* [*dest*!, *consumes 1*]:
  assumes *sum-list* (*map length* (*take i xs*)) + *j* = *sum-list* (*map length* (*take k xs*)) + *l*
   and *i* < *length xs k* < *length xs*
   and *j* < *length* (*xs* ! *i*) *l* < *length* (*xs* ! *k*)
 shows *i* = *k* ∧ *j* = *l* **using** *assms*
proof (*induct xs rule*: *rev-induct*)
  case (*snoc x xs*)
  then show *?case*
    by (*auto simp*: *nth-append split*: *if-splits*)
      (*metis concat-nth-length length-concat not-add-less1*)+
qed *auto*

lemma *concat-index-split-unique*:
  assumes *i* < *length* (*concat xs*) **and** *length xs* = *length ys*
    and ∀ *i* < *length xs*. *length* (*xs* ! *i*) = *length* (*ys* ! *i*)
  shows *concat-index-split* (*n*, *i*) *xs* = *concat-index-split* (*n*, *i*) *ys* **using** *assms*
proof (*induct xs arbitrary*: *ys n i*)
  case (*Cons x xs*) **note** *IH* = *this* **show** *?case*
  proof (*cases ys*)
    case *Nil* **then show** *?thesis* **using** *Cons*(*3*) **by** *auto*
  next
    case [*simp*]: (*Cons y ys'*)
    have [*simp*]: *length y* = *length x* **using** *IH*(*4*) **by** *force*
    have [*simp*]: ¬ *i* < *length x* ⟹ *i* − *length x* < *length* (*concat xs*) **using** *IH*(*2*)
by *auto*
    have [*simp*]: *i* < *length ys'* ⟹ *length* (*xs* ! *i*) = *length* (*ys'* ! *i*) **for** *i* **using** *IH*(*3*, *4*)
      by (*auto simp*: *All-less-Suc*) (*metis IH*(*4*) *Suc-less-eq length-Cons Cons nth-Cons-Suc*)
    show *?thesis* **using** *IH*(*2*−) *IH*(*1*)[*of i* − *length x ys' Suc n*] **by** *auto*
  qed
qed *auto*

lemma *set-vars-term-list* [*simp*]:
  set (*vars-term-list t*) = *vars-term t*
  by (*induct t*) *simp-all*

lemma *vars-term-list-empty-ground* [*simp*]:
  vars-term-list t = [] ⟷ *ground t*
  by (*induct t*) *auto*

lemma *varposs-imp-poss*:
  assumes *p* ∈ *varposs t*
  shows *p* ∈ *poss t*
  using *assms* **by** (*induct t arbitrary*: *p*) *auto*

lemma *vaposs-list-fun*:

assumes $p \in$ *set* (*varposs-list* (*Fun f ts*))
obtains *i ps* **where** $i <$ *length ts* $p = i \# ps$
using *assms set-zip-leftD* **by** *fastforce*

lemma *varposs-list-distinct*:
distinct (*varposs-list t*)
proof (*induct t*)
case (*Fun f ts*)
then show *?case* **proof** (*induct ts rule*: *rev-induct*)
case (*snoc x xs*)
then have *distinct* (*varposs-list* (*Fun f xs*)) *distinct* (*varposs-list x*) **by** *auto*
then show *?case* **using** *snoc* **by** (*auto simp add*: *map-cons-presv-distinct dest*: *set-zip-leftD*)
qed *auto*
qed *auto*

lemma *varposs-append*:
varposs (*Fun f* (*ts* @ [*t*])) = *varposs* (*Fun f ts*) ∪ ((#) (*length ts*)) ‘ *varposs t*
by (*auto simp*: *nth-append split*: *if-splits*)

lemma *varposs-eq-varposs-list*:
set (*varposs-list t*) = *varposs t*
proof (*induct t*)
case (*Fun f ts*)
then show *?case* **proof** (*induct ts rule*: *rev-induct*)
case (*snoc x xs*)
then have *varposs* (*Fun f xs*) = *set* (*varposs-list* (*Fun f xs*))
varposs x = *set* (*varposs-list x*) **by** *auto*
then show *?case* **using** *snoc* **unfolding** *varposs-append*
by *auto*
qed *auto*
qed *auto*

lemma *varposs-list-var-terms-length*:
length (*varposs-list t*) = *length* (*vars-term-list t*)
by (*induct t*) (*auto simp*: *vars-term-list.simps intro*: *eq-length-concat-nth*)

lemma *vars-term-list-nth*:
assumes $i <$ *length* (*vars-term-list* (*Fun f ts*))
and *concat-index-split* (*0*, *i*) (*map vars-term-list ts*) = (*k*, *j*)
shows $k <$ *length ts* $\wedge$ $j <$ *length* (*vars-term-list* (*ts* ! *k*)) $\wedge$
vars-term-list (*Fun f ts*) ! *i* = *map vars-term-list ts* ! *k* ! *j* $\wedge$
i = *sum-list* (*map length* (*map vars-term-list* (*take k ts*))) + *j*
using *assms concat-index-split-less-length-concat*[*of i map vars-term-list ts k j*]
by (*auto simp*: *vars-term-list.simps comp-def take-map*)

lemma *varposs-list-nth*:
assumes $i <$ *length* (*varposs-list* (*Fun f ts*))
and *concat-index-split* (*0*, *i*) (*poss-args varposs-list ts*) = (*k*, *j*)

shows $k < length\ ts \wedge j < length\ (varposs\text{-}list\ (ts\ !\ k)) \wedge$
$varposs\text{-}list\ (Fun\ f\ ts)\ !\ i = k\ \#\ (map\ varposs\text{-}list\ ts)\ !\ k\ !\ j \wedge$
$i = sum\text{-}list\ (map\ length\ (map\ varposs\text{-}list\ (take\ k\ ts))) + j$
using *assms concat-index-split-less-length-concat*[*of i poss-args varposs-list ts k j*]
by (*auto simp*: *comp-def take-map intro*: *nth-sum-listI*)

lemma *varposs-list-to-var-term-list*:
assumes $i < length\ (varposs\text{-}list\ t)$
shows *the-Var* $(t\ |\text{-}\ (varposs\text{-}list\ t\ !\ i)) = (vars\text{-}term\text{-}list\ t)\ !\ i$ **using** *assms*
proof (*induct t arbitrary*: *i*)
case (*Fun f ts*)
have *concat-index-split* (*0*, *i*) (*poss-args varposs-list ts*) = *concat-index-split* (*0*, *i*) (*map vars-term-list ts*)
using *Fun*(*2*) *concat-index-split-unique*[*of i poss-args varposs-list ts map vars-term-list ts 0*]
using *varposs-list-var-terms-length*[*of ts* ! *i* **for** *i*]
by (*auto simp*: *vars-term-list.simps*)
then obtain *k j* **where** *concat-index-split* (*0*, *i*) (*poss-args varposs-list ts*) = (*k*, *j*)
concat-index-split (*0*, *i*) (*map vars-term-list ts*) = (*k*, *j*) **by** *fastforce*
from *varposs-list-nth*[*OF Fun*(*2*) *this*(*1*)] *vars-term-list-nth*[*OF - this*(*2*)]
show *?case* **using** *Fun*(*2*) *Fun*(*1*)[*OF nth-mem*] *varposs-list-var-terms-length*[*of Fun f ts*] **by** *auto*
qed (*auto simp*: *vars-term-list.simps*)

end

2 Preliminaries

2.1 Multihole Contexts

theory *Multihole-Context*
imports
Utils
begin

unbundle *lattice-syntax*

2.1.1 Partitioning lists into chunks of given length

lemma *concat-nth*:
assumes $m < length\ xs$ **and** $n < length\ (xs\ !\ m)$
and $i = sum\text{-}list\ (map\ length\ (take\ m\ xs)) + n$
shows $concat\ xs\ !\ i = xs\ !\ m\ !\ n$
using *assms*
proof (*induct xs arbitrary*: *m n i*)
case (*Cons x xs*)
show *?case*
proof (*cases m*)

```
    case 0
    then show ?thesis using Cons by (simp add: nth-append)
  next
    case (Suc k)
    with Cons(1) [of k n i − length x] and Cons(2−)
      show ?thesis by (simp-all add: nth-append)
  qed
qed simp

lemma sum-list-take-eq:
  fixes xs :: nat list
  shows k < i ⟹ i < length xs ⟹ sum-list (take i xs) =
    sum-list (take k xs) + xs ! k + sum-list (take (i − Suc k) (drop (Suc k) xs))
  by (subst id-take-nth-drop [of k]) (auto simp: min-def drop-take)

fun partition-by where
  partition-by xs [] = [] |
  partition-by xs (y#ys) = take y xs # partition-by (drop y xs) ys

lemma partition-by-map0-append [simp]:
  partition-by xs (map (λx. 0) ys @ zs) = replicate (length ys) [] @ partition-by xs
zs
by (induct ys) simp-all

lemma concat-partition-by [simp]:
  sum-list ys = length xs ⟹ concat (partition-by xs ys) = xs
by (induct ys arbitrary: xs) simp-all

definition partition-by-idx where
  partition-by-idx l ys i j = partition-by [0..<l] ys ! i ! j

lemma partition-by-nth-nth-old:
  assumes i < length (partition-by xs ys)
    and j < length (partition-by xs ys ! i)
    and sum-list ys = length xs
  shows partition-by xs ys ! i ! j = xs ! (sum-list (map length (take i (partition-by
xs ys))) + j)
  using concat-nth [OF assms(1, 2) refl]
  unfolding concat-partition-by [OF assms(3)] by simp

lemma map-map-partition-by:
  map (map f) (partition-by xs ys) = partition-by (map f xs) ys
by (induct ys arbitrary: xs) (auto simp: take-map drop-map)

lemma length-partition-by [simp]:
  length (partition-by xs ys) = length ys
  by (induct ys arbitrary: xs) simp-all

lemma partition-by-Nil [simp]:
```

partition-by [] *ys* = *replicate* (*length ys*) []
by (*induct ys*) *simp-all*

lemma *partition-by-concat-id* [*simp*]:
assumes *length xss* = *length ys*
and ⋀*i*. *i* < *length ys* ⟹ *length* (*xss* ! *i*) = *ys* ! *i*
shows *partition-by* (*concat xss*) *ys* = *xss*
using *assms* **by** (*induct ys arbitrary*: *xss*) (*simp*, *case-tac xss*, *simp*, *fastforce*)

lemma *partition-by-nth*:
i < *length ys* ⟹ *partition-by xs ys* ! *i* = *take* (*ys* ! *i*) (*drop* (*sum-list* (*take i ys*)) *xs*)
by (*induct ys arbitrary*: *xs i*) (*simp*, *case-tac i*, *simp-all add*: *ac-simps*)

lemma *partition-by-nth-less*:
assumes *k* < *i* **and** *i* < *length zs*
and *length xs* = *sum-list* (*take i zs*) + *j*
shows *partition-by* (*xs* @ *y* # *ys*) *zs* ! *k* = *take* (*zs* ! *k*) (*drop* (*sum-list* (*take k zs*)) *xs*)
proof −
have *partition-by* (*xs* @ *y* # *ys*) *zs* ! *k* =
take (*zs* ! *k*) (*drop* (*sum-list* (*take k zs*)) (*xs* @ *y* # *ys*))
using *assms* **by** (*auto simp*: *partition-by-nth*)
moreover have *zs* ! *k* + *sum-list* (*take k zs*) ≤ *length xs*
using *assms* **by** (*simp add*: *sum-list-take-eq*)
ultimately show *?thesis* **by** *simp*
qed

lemma *partition-by-nth-greater*:
assumes *i* < *k* **and** *k* < *length zs* **and** *j* < *zs* ! *i*
and *length xs* = *sum-list* (*take i zs*) + *j*
shows *partition-by* (*xs* @ *y* # *ys*) *zs* ! *k* =
take (*zs* ! *k*) (*drop* (*sum-list* (*take k zs*) − *1*) (*xs* @ *ys*))
proof −
have *partition-by* (*xs* @ *y* # *ys*) *zs* ! *k* =
take (*zs* ! *k*) (*drop* (*sum-list* (*take k zs*)) (*xs* @ *y* # *ys*))
using *assms* **by** (*auto simp*: *partition-by-nth*)
moreover have *sum-list* (*take k zs*) > *length xs*
using *assms* **by** (*auto simp*: *sum-list-take-eq*)
ultimately show *?thesis* **by** (*auto*) (*metis Suc-diff-Suc drop-Suc-Cons*)
qed

lemma *length-partition-by-nth*:
sum-list ys = *length xs* ⟹ *i* < *length ys* ⟹ *length* (*partition-by xs ys* ! *i*) = *ys* ! *i*
by (*induct ys arbitrary*: *xs i*; *case-tac i*) *auto*

lemma *partition-by-nth-nth-elem*:
assumes *sum-list ys* = *length xs i* < *length ys j* < *ys* ! *i*

shows *partition-by xs ys ! i ! j ∈ set xs*
proof −
from *assms* **have** *j < length* (*partition-by xs ys ! i*) **by** (*simp only*: *length-partition-by-nth*)
then have *partition-by xs ys ! i ! j ∈ set* (*partition-by xs ys ! i*) **by** *auto*
with *assms*(*2*) **have** *partition-by xs ys ! i ! j ∈ set* (*concat* (*partition-by xs ys*))
by *auto*
then show *?thesis* **using** *assms* **by** *simp*
qed

lemma *partition-by-nth-nth*:
assumes *sum-list ys = length xs i < length ys j < ys ! i*
shows *partition-by xs ys ! i ! j = xs ! partition-by-idx* (*length xs*) *ys i j*
partition-by-idx (*length xs*) *ys i j < length xs*
unfolding *partition-by-idx-def*
proof −
let *?n = partition-by* [*0..<length xs*] *ys ! i ! j*
show *?n < length xs*
using *partition-by-nth-nth-elem*[*OF - assms*(*2*,*3*), *of* [*0..<length xs*]] *assms*(*1*)
by *simp*
have *li*: *i < length* (*partition-by* [*0..<length xs*] *ys*) **using** *assms*(*2*) **by** *simp*
have *lj*: *j < length* (*partition-by* [*0..<length xs*] *ys ! i*)
using *assms* **by** (*simp add*: *length-partition-by-nth*)
have *partition-by* (*map* ((!) *xs*) [*0..<length xs*]) *ys ! i ! j = xs ! ?n*
by (*simp only*: *map-map-partition-by*[*symmetric*] *nth-map*[*OF li*] *nth-map*[*OF lj*])
then show *partition-by xs ys ! i ! j = xs ! ?n* **by** (*simp add*: *map-nth*)
qed

lemma *map-length-partition-by* [*simp*]:
sum-list ys = length xs ⟹ map length (*partition-by xs ys*) = *ys*
by (*intro nth-equalityI*, *auto simp*: *length-partition-by-nth*)

lemma *map-partition-by-nth* [*simp*]:
i < length ys ⟹ map f (*partition-by xs ys ! i*) = *partition-by* (*map f xs*) *ys ! i*
by (*induct ys arbitrary*: *i xs*) (*simp*, *case-tac i*, *simp-all add*: *take-map drop-map*)

lemma *sum-list-partition-by* [*simp*]:
sum-list ys = length xs ⟹
sum-list (*map* (*λx. sum-list* (*map f x*)) (*partition-by xs ys*)) = *sum-list* (*map f xs*)
by (*induct ys arbitrary*: *xs*) (*simp-all*, *metis append-take-drop-id sum-list-append map-append*)

lemma *partition-by-map-conv*:
partition-by xs ys = map (*λi. take* (*ys ! i*) (*drop* (*sum-list* (*take i ys*)) *xs*)) [*0 ..< length ys*]
by (*rule nth-equalityI*) (*simp-all add*: *partition-by-nth*)

lemma *UN-set-partition-by-map*:

sum-list ys = length xs $\Longrightarrow$ $(\bigcup x{\in}set\ (partition\text{-}by\ (map\ f\ xs)\ ys).\ \bigcup\ (set\ x)) = \bigcup(set\ (map\ f\ xs))$
by (*induct ys arbitrary*: *xs*)
(*simp-all add*: *drop-map take-map*, *metis UN-Un append-take-drop-id set-append*)

lemma *UN-set-partition-by*:
sum-list ys = length xs $\Longrightarrow$ $(\bigcup zs \in set\ (partition\text{-}by\ xs\ ys).\ \bigcup x \in set\ zs.\ f\ x) = (\bigcup x \in set\ xs.\ f\ x)$
by (*induct ys arbitrary*: *xs*) (*simp-all*, *metis UN-Un append-take-drop-id set-append*)

lemma *Ball-atLeast0LessThan-partition-by-conv*:
$(\forall\, i{\in}\{0..{<}length\ ys\}.\ \forall\, x{\in}set\ (partition\text{-}by\ xs\ ys\ !\ i).\ P\ x) =$
$(\forall\, x \in \bigcup(set\ (map\ set\ (partition\text{-}by\ xs\ ys))).\ P\ x)$
by *auto* (*metis atLeast0LessThan in-set-conv-nth length-partition-by lessThan-iff*)

lemma *Ball-set-partition-by*:
sum-list ys = length xs $\Longrightarrow$
$(\forall\, x \in set\ (partition\text{-}by\ xs\ ys).\ \forall\, y \in set\ x.\ P\ y) = (\forall\, x \in set\ xs.\ P\ x)$
proof (*induct ys arbitrary*: *xs*)
case (*Cons y ys*)
then show *?case*
apply (*subst* (*2*) *append-take-drop-id* [*of y xs*, *symmetric*])
apply (*simp only*: *set-append*)
apply *auto*
done
qed *simp*

lemma *partition-by-append2*:
partition-by xs (*ys* @ *zs*) = *partition-by* (*take* (*sum-list ys*) *xs*) *ys* @ *partition-by* (*drop* (*sum-list ys*) *xs*) *zs*
by (*induct ys arbitrary*: *xs*) (*auto simp*: *drop-take ac-simps split*: *split-min*)

lemma *partition-by-concat2*:
partition-by xs (*concat ys*) =
concat (*map* (λ*i* . *partition-by* (*partition-by xs* (*map sum-list ys*) ! *i*) (*ys* ! *i*)) [*0..<length ys*])
proof −
have ∗: *map* (λ*i* . *partition-by* (*partition-by xs* (*map sum-list ys*) ! *i*) (*ys* ! *i*)) [*0..<length ys*] =
map (λ(*x*,*y*). *partition-by x y*) (*zip* (*partition-by xs* (*map sum-list ys*)) *ys*)
using *zip-nth-conv*[*of partition-by xs* (*map sum-list ys*) *ys*] **by** *auto*
show *?thesis* **unfolding** ∗ **by** (*induct ys arbitrary*: *xs*) (*auto simp*: *partition-by-append2*)
qed

lemma *partition-by-partition-by*:
length xs = sum-list (*map sum-list ys*) $\Longrightarrow$
partition-by (*partition-by xs* (*concat ys*)) (*map length ys*) =
map (λ*i*. *partition-by* (*partition-by xs* (*map sum-list ys*) ! *i*) (*ys* ! *i*)) [*0..<length ys*]

by (*auto simp*: *partition-by-concat2 intro*: *partition-by-concat-id*)

2.1.2 Multihole contexts definition and functionalities

datatype (*'f*, *vars-mctxt* : *'v*) *mctxt* = *MVar 'v* | *MHole* | *MFun 'f* (*'f*, *'v*) *mctxt list*

2.1.3 Conversions from and to multihole contexts

primrec *mctxt-of-term* :: (*'f*, *'v*) *term* ⇒ (*'f*, *'v*) *mctxt* **where**
mctxt-of-term (*Var x*) = *MVar x* |
mctxt-of-term (*Fun f ts*) = *MFun f* (*map mctxt-of-term ts*)

primrec *term-of-mctxt* :: (*'f*, *'v*) *mctxt* ⇒ (*'f*, *'v*) *term* **where**
term-of-mctxt (*MVar x*) = *Var x* |
term-of-mctxt (*MFun f Cs*) = *Fun f* (*map term-of-mctxt Cs*)

fun *num-holes* :: (*'f*, *'v*) *mctxt* ⇒ *nat* **where**
num-holes (*MVar* -) = *0* |
num-holes MHole = *1* |
num-holes (*MFun* - *ctxts*) = *sum-list* (*map num-holes ctxts*)

fun *ground-mctxt* :: (*'f*, *'v*) *mctxt* ⇒ *bool* **where**
ground-mctxt (*MVar* -) = *False* |
ground-mctxt MHole = *True* |
ground-mctxt (*MFun f Cs*) = *Ball* (*set Cs*) *ground-mctxt*

fun *map-mctxt* :: (*'f* ⇒ *'g*) ⇒ (*'f*, *'v*) *mctxt* ⇒ (*'g*, *'v*) *mctxt*
where
map-mctxt - (*MVar x*) = (*MVar x*) |
map-mctxt - (*MHole*) = *MHole* |
map-mctxt fg (*MFun f Cs*) = *MFun* (*fg f*) (*map* (*map-mctxt fg*) *Cs*)

abbreviation *partition-holes xs Cs* ≡ *partition-by xs* (*map num-holes Cs*)
abbreviation *partition-holes-idx l Cs* ≡ *partition-by-idx l* (*map num-holes Cs*)

fun *fill-holes* :: (*'f*, *'v*) *mctxt* ⇒ (*'f*, *'v*) *term list* ⇒ (*'f*, *'v*) *term* **where**
fill-holes (*MVar x*) - = *Var x* |
fill-holes MHole [*t*] = *t* |
fill-holes (*MFun f cs*) *ts* = *Fun f* (*map* (λ *i*. *fill-holes* (*cs* ! *i*)
(*partition-holes ts cs* ! *i*)) [*0* ..< *length cs*])

fun *fill-holes-mctxt* :: (*'f*, *'v*) *mctxt* ⇒ (*'f*, *'v*) *mctxt list* ⇒ (*'f*, *'v*) *mctxt* **where**
fill-holes-mctxt (*MVar x*) - = *MVar x* |
fill-holes-mctxt MHole [] = *MHole* |
fill-holes-mctxt MHole [*t*] = *t* |
fill-holes-mctxt (*MFun f cs*) *ts* = (*MFun f* (*map* (λ *i*. *fill-holes-mctxt* (*cs* ! *i*)
(*partition-holes ts cs* ! *i*)) [*0* ..< *length cs*]))

fun *unfill-holes* :: ('*f*, '*v*) *mctxt* ⇒ ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *term list* **where**
unfill-holes MHole t = [*t*]
| *unfill-holes* (*MVar w*) (*Var v*) = (*if v* = *w then* [] *else undefined*)
| *unfill-holes* (*MFun g Cs*) (*Fun f ts*) = (*if f* = *g* ∧ *length ts* = *length Cs then*
concat (*map* (λ*i*. *unfill-holes* (*Cs* ! *i*) (*ts* ! *i*)) [*0*..<*length ts*]) *else undefined*)

fun *funas-mctxt* **where**
funas-mctxt (*MFun f Cs*) = {(*f*, *length Cs*)} ∪ ⋃(*funas-mctxt* ‘ *set Cs*) |
funas-mctxt - = {}

fun *split-vars* :: ('*f*, '*v*) *term* ⇒ (('*f*, '*v*) *mctxt* × '*v list*) **where**
split-vars (*Var x*) = (*MHole*, [*x*]) |
split-vars (*Fun f ts*) = (*MFun f* (*map* (*fst* ∘ *split-vars*) *ts*), *concat* (*map* (*snd* ∘ *split-vars*) *ts*))

fun *hole-poss-list* :: ('*f*, '*v*) *mctxt* ⇒ *pos list* **where**
hole-poss-list (*MVar x*) = [] |
hole-poss-list MHole = [[]] |
hole-poss-list (*MFun f cs*) = *concat* (*poss-args hole-poss-list cs*)

fun *map-vars-mctxt* :: ('*v* ⇒ '*w*) ⇒ ('*f*, '*v*) *mctxt* ⇒ ('*f*, '*w*) *mctxt*
where
map-vars-mctxt vw MHole = *MHole* |
map-vars-mctxt vw (*MVar v*) = (*MVar* (*vw v*)) |
map-vars-mctxt vw (*MFun f Cs*) = *MFun f* (*map* (*map-vars-mctxt vw*) *Cs*)

inductive *eq-fill* :: ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *mctxt* × ('*f*, '*v*) *term list* ⇒ *bool* (‹(-/ =*f* -)› [*51*, *51*] *50*)
where
eqfI [*intro*]: *t* = *fill-holes D ss* ⟹ *num-holes D* = *length ss* ⟹ *t* =*f* (*D*, *ss*)

2.1.4 Semilattice Structures

instantiation *mctxt* :: (*type*, *type*) *inf*

begin

fun *inf-mctxt* :: ('*a*, '*b*) *mctxt* ⇒ ('*a*, '*b*) *mctxt* ⇒ ('*a*, '*b*) *mctxt*
where
MHole ⊓ *D* = *MHole* |
C ⊓ *MHole* = *MHole* |
MVar x ⊓ *MVar y* = (*if x* = *y then MVar x else MHole*) |
MFun f Cs ⊓ *MFun g Ds* =
(*if f* = *g* ∧ *length Cs* = *length Ds then MFun f* (*map* (*case-prod* (⊓)) (*zip Cs Ds*))
else MHole) |
C ⊓ *D* = *MHole*

instance ..

end

lemma *inf-mctxt-idem* [*simp*]:
fixes $C :: ('f, 'v)\ mctxt$
shows $C \sqcap C = C$
by (*induct C*) (*auto simp*: *zip-same-conv-map intro*: *map-idI*)

lemma *inf-mctxt-MHole2* [*simp*]:
$C \sqcap MHole = MHole$
by (*induct C*) *simp-all*

lemma *inf-mctxt-comm* [*ac-simps*]:
$(C :: ('f, 'v)\ mctxt) \sqcap D = D \sqcap C$
by (*induct C D rule*: *inf-mctxt.induct*) (*fastforce simp*: *in-set-conv-nth intro*!: *nth-equalityI*)+

lemma *inf-mctxt-assoc* [*ac-simps*]:
fixes $C :: ('f, 'v)\ mctxt$
shows $C \sqcap D \sqcap E = C \sqcap (D \sqcap E)$
apply (*induct C D arbitrary*: *E rule*: *inf-mctxt.induct*)
apply *auto*
apply (*case-tac E*, *auto*)+
apply (*fastforce simp*: *in-set-conv-nth intro*!: *nth-equalityI*)
apply (*case-tac E*, *auto*)+
done

instantiation *mctxt* :: (*type*, *type*) *order*
begin

definition $(C :: ('a, 'b)\ mctxt) \leq D \longleftrightarrow C \sqcap D = C$
definition $(C :: ('a, 'b)\ mctxt) < D \longleftrightarrow C \leq D \wedge \neg\ D \leq C$

instance
by (*standard*, *simp-all add*: *less-eq-mctxt-def less-mctxt-def ac-simps*, *metis inf-mctxt-assoc*)

end

inductive *less-eq-mctxt'* :: $('f, 'v)\ mctxt \Rightarrow ('f,'v)\ mctxt \Rightarrow bool$ **where**
less-eq-mctxt' MHole u
| *less-eq-mctxt'* (*MVar v*) (*MVar v*)
| *length cs* = *length ds* $\Longrightarrow$ ($\bigwedge i.\ i <$ *length cs* $\Longrightarrow$ *less-eq-mctxt'* (*cs* ! *i*) (*ds* ! *i*))
$\Longrightarrow$ *less-eq-mctxt'* (*MFun f cs*) (*MFun f ds*)

2.1.5 Lemmata

lemma *partition-holes-fill-holes-conv*:
fill-holes (*MFun f cs*) *ts* =

Fun f [*fill-holes* (*cs* ! *i*) (*partition-holes ts cs* ! *i*). *i* ← [*0* ..< *length cs*]]
by (*simp add*: *partition-by-nth take-map*)

lemma *partition-holes-fill-holes-mctxt-conv*:
fill-holes-mctxt (*MFun f Cs*) *ts* =
MFun f [*fill-holes-mctxt* (*Cs* ! *i*) (*partition-holes ts Cs* ! *i*). *i* ← [*0* ..< *length Cs*]]
by (*simp add*: *partition-by-nth take-map*)

The following induction scheme provides the *MFun* case with the list argument split according to the argument contexts. This feature is quite delicate: its benefit can be destroyed by premature simplification using the *sum-list ?ys* = *length ?xs* ⟹ *concat* (*partition-by ?xs ?ys*) = *?xs* simplification rule.

lemma *fill-holes-induct2*[*consumes 2*, *case-names MHole MVar MFun*]:
fixes *P* :: ('*f*,'*v*) *mctxt* ⇒ '*a list* ⇒ '*b list* ⇒ *bool*
assumes *len1*: *num-holes C* = *length xs* **and** *len2*: *num-holes C* = *length ys*
and *Hole*: ⋀*x y*. *P MHole* [*x*] [*y*]
and *Var*: ⋀*v*. *P* (*MVar v*) [] []
and *Fun*: ⋀*f Cs xs ys*. *sum-list* (*map num-holes Cs*) = *length xs* ⟹
sum-list (*map num-holes Cs*) = *length ys* ⟹
(⋀*i*. *i* < *length Cs* ⟹ *P* (*Cs* ! *i*) (*partition-holes xs Cs* ! *i*) (*partition-holes ys Cs* ! *i*)) ⟹
P (*MFun f Cs*) (*concat* (*partition-holes xs Cs*)) (*concat* (*partition-holes ys Cs*))
shows *P C xs ys*
proof (*insert len1 len2*, *induct C arbitrary*: *xs ys*)
case *MHole* **then show** *?case* **using** *Hole* **by** (*cases xs*; *cases ys*) *auto*
next
case (*MVar v*) **then show** *?case* **using** *Var* **by** *auto*
next
case (*MFun f Cs*) **then show** *?case* **using** *Fun*[*of Cs xs ys f*] **by** (*auto simp*: *length-partition-by-nth*)
qed

lemma *fill-holes-induct*[*consumes 1*, *case-names MHole MVar MFun*]:
fixes *P* :: ('*f*,'*v*) *mctxt* ⇒ '*a list* ⇒ *bool*
assumes *len*: *num-holes C* = *length xs*
and *Hole*: ⋀*x*. *P MHole* [*x*]
and *Var*: ⋀*v*. *P* (*MVar v*) []
and *Fun*: ⋀*f Cs xs*. *sum-list* (*map num-holes Cs*) = *length xs* ⟹
(⋀*i*. *i* < *length Cs* ⟹ *P* (*Cs* ! *i*) (*partition-holes xs Cs* ! *i*)) ⟹
P (*MFun f Cs*) (*concat* (*partition-holes xs Cs*))
shows *P C xs*
using *fill-holes-induct2*[*of C xs xs* λ *C xs* -. *P C xs*] *assms* **by** *simp*

lemma *length-partition-holes-nth* [*simp*]:
assumes *sum-list* (*map num-holes cs*) = *length ts*
and *i* < *length cs*
shows *length* (*partition-holes ts cs* ! *i*) = *num-holes* (*cs* ! *i*)

using *assms* **by** (*simp add*: *length-partition-by-nth*)

lemmas
map-partition-holes-nth [*simp*] =
map-partition-by-nth [*of* - *map num-holes Cs* **for** *Cs*, *unfolded length-map*] **and**
length-partition-holes [*simp*] =
length-partition-by [*of* - *map num-holes Cs* **for** *Cs*, *unfolded length-map*]

lemma *fill-holes-term-of-mctxt*:
num-holes C = *0* ⟹ *fill-holes C* [] = *term-of-mctxt C*
by (*induct C*) (*auto simp add*: *map-eq-nth-conv*)

lemma *fill-holes-MHole*:
length ts = *Suc 0* ⟹ *ts* ! *0* = *u* ⟹ *fill-holes MHole ts* = *u*
by (*cases ts*) *simp-all*

lemma *fill-holes-arbitrary*:
assumes *lCs*: *length Cs* = *length ts*
and *lss*: *length ss* = *length ts*
and *rec*: ⋀ *i*. *i* < *length ts* ⟹ *num-holes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧ *f* (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *map* (λ*i*. *f* (*Cs* ! *i*) (*partition-holes* (*concat ss*) *Cs* ! *i*)) [*0* ..< *length Cs*] = *ts*
proof −
have *sum-list* (*map num-holes Cs*) = *length* (*concat ss*) **using** *assms*
by (*auto simp*: *length-concat map-nth-eq-conv intro*: *arg-cong*[*of* - - *sum-list*])
moreover have *partition-holes* (*concat ss*) *Cs* = *ss*
using *assms* **by** (*auto intro*: *partition-by-concat-id*)
ultimately show *?thesis* **using** *assms* **by** (*auto intro*: *nth-equalityI*)
qed

lemma *fill-holes-MFun*:
assumes *lCs*: *length Cs* = *length ts*
and *lss*: *length ss* = *length ts*
and *rec*: ⋀ *i*. *i* < *length ts* ⟹ *num-holes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧ *fill-holes* (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *fill-holes* (*MFun f Cs*) (*concat ss*) = *Fun f ts*
unfolding *fill-holes.simps term.simps*
by (*rule conjI*[*OF refl*], *rule fill-holes-arbitrary*[*OF lCs lss rec*])

lemma *eqfE*:
assumes *t* =$_f$ (*D*, *ss*) **shows** *t* = *fill-holes D ss num-holes D* = *length ss*
using *assms*[*unfolded eq-fill.simps*] **by** *auto*

lemma *eqf-MFunE*:
assumes *s* =$_f$ (*MFun f Cs*,*ss*)
obtains *ts sss* **where** *s* = *Fun f ts length ts* = *length Cs length sss* = *length Cs*
⋀ *i*. *i* < *length Cs* ⟹ *ts* ! *i* =$_f$ (*Cs* ! *i*, *sss* ! *i*)

```
  ss = concat sss
proof −
  from eqfE[OF assms] have fh: s = fill-holes (MFun f Cs) ss
    and nh: sum-list (map num-holes Cs) = length ss by auto
  from fh obtain ts where s: s = Fun f ts by (cases s, auto)
  from fh[unfolded s]
  have ts: ts = map (λi. fill-holes (Cs ! i) (partition-holes ss Cs ! i)) [0..<length
Cs]
    (is - = map (?f Cs ss) -)
    by auto
  let ?sss = partition-holes ss Cs
  from nh
  have ∗: length ?sss = length Cs ⋀i. i < length Cs ⟹ ts ! i =f (Cs ! i, ?sss !
i) ss = concat ?sss
    by (auto simp: ts)
  have len: length ts = length Cs unfolding ts by auto
  assume ass: ⋀ts sss. s = Fun f ts ⟹
            length ts = length Cs ⟹
           length sss = length Cs ⟹ (⋀i. i < length Cs ⟹ ts ! i =f (Cs ! i, sss
! i)) ⟹ ss = concat sss ⟹ thesis
  show thesis
    by (rule ass[OF s len ∗])
qed

lemma eqf-MFunI:
  assumes length sss = length Cs
    and length ts = length Cs
    and⋀ i. i < length Cs ⟹ ts ! i =f (Cs ! i, sss ! i)
  shows Fun f ts =f (MFun f Cs, concat sss)
proof
  have num-holes (MFun f Cs) = sum-list (map num-holes Cs) by simp
  also have map num-holes Cs = map length sss
    by (rule nth-equalityI, insert assms eqfE[OF assms(3)], auto)
  also have sum-list (...) = length (concat sss) unfolding length-concat ..
  finally show num-holes (MFun f Cs) = length (concat sss) .
  show Fun f ts = fill-holes (MFun f Cs) (concat sss)
     by (rule fill-holes-MFun[symmetric], insert assms(1,2) eqfE[OF assms(3)],
auto)
qed

lemma split-vars-ground-vars:
  assumes ground-mctxt C and num-holes C = length xs
  shows split-vars (fill-holes C (map Var xs)) = (C, xs) using assms
proof (induct C arbitrary: xs)
  case (MHole xs)
  then show ?case by (cases xs, auto)
next
  case (MFun f Cs xs)
  have fill-holes (MFun f Cs) (map Var xs) =f (MFun f Cs, map Var xs)
```

```
    by (rule eqfI, insert MFun(3), auto)
  from eqf-MFunE[OF this]
  obtain ts xss where fh: fill-holes (MFun f Cs) (map Var xs) = Fun f ts
    and lent: length ts = length Cs
    and lenx: length xss = length Cs
    and args: ⋀i. i < length Cs ⟹ ts ! i =_f (Cs ! i, xss ! i)
    and id: map Var xs = concat xss by auto
  from arg-cong[OF id, of map the-Var] have id2: xs = concat (map (map the-Var)
xss)
    by (metis map-concat length-map map-nth-eq-conv term.sel(1))
  {
    fix i
    assume i: i < length Cs
    then have mem: Cs ! i ∈ set Cs by auto
    with MFun(2) have ground: ground-mctxt (Cs ! i) by auto
    have map Var (map the-Var (xss ! i)) = map id (xss ! i) unfolding map-map
o-def map-eq-conv
    proof
      fix x
      assume x ∈ set (xss ! i)
      with lenx i have x ∈ set (concat xss) by auto
      from this[unfolded id[symmetric]] show Var (the-Var x) = id x by auto
    qed
    then have idxss: map Var (map the-Var (xss ! i)) = xss ! i by auto
    note rec = eqfE[OF args[OF i]]
   note IH = MFun(1)[OF mem ground, of map the-Var (xss ! i), unfolded rec(2)
idxss rec(1)[symmetric]]
    from IH have split-vars (ts ! i) = (Cs ! i, map the-Var (xss ! i)) by auto
    note this idxss
  }
  note IH = this
  have ?case = (map fst (map split-vars ts) = Cs ∧ concat (map snd (map split-vars
ts)) = concat (map (map the-Var) xss))
    unfolding fh unfolding id2 by auto
  also have ...
   proof (rule conjI[OF nth-equalityI arg-cong[of - - concat, OF nth-equalityI,
rule-format]], unfold length-map lent lenx)
    fix i
    assume i: i < length Cs
    with arg-cong[OF IH(2)[OF this], of map the-Var]
      IH[OF this] show map snd (map split-vars ts) ! i = map (map the-Var) xss
! i using lent lenx by auto
  qed (insert IH lent, auto)
  finally show ?case .
qed auto


lemma split-vars-vars-term-list: snd (split-vars t) = vars-term-list t
proof (induct t)
```

case (*Fun f ts*)
then show *?case* **by** (*auto simp*: *vars-term-list.simps o-def*, *induct ts*, *auto*)
qed (*auto simp*: *vars-term-list.simps*)

lemma *split-vars-num-holes*: *num-holes* (*fst* (*split-vars t*)) = *length* (*snd* (*split-vars t*))
proof (*induct t*)
case (*Fun f ts*)
then show *?case* **by** (*induct ts*, *auto*)
qed *simp*

lemma *ground-eq-fill*: $t =_f (C,ss) \Longrightarrow$ *ground t* = (*ground-mctxt C* ∧ (∀ *s* ∈ *set ss*. *ground s*))
proof (*induct C arbitrary*: *t ss*)
case (*MVar x*)
from *eqfE*[*OF this*] **show** *?case* **by** *simp*
next
case (*MHole t ss*)
from *eqfE*[*OF this*] **show** *?case* **by** (*cases ss*, *auto*)
next
case (*MFun f Cs s ss*)
from *eqf-MFunE*[*OF MFun*(*2*)] **obtain** *ts sss* **where** *s*: *s* = *Fun f ts* **and** *len*: *length ts* = *length Cs length sss* = *length Cs*
and *IH*: ⋀ *i*. *i* < *length Cs* ⟹ *ts* ! *i* =$_f$ (*Cs* ! *i*, *sss* ! *i*) **and** *ss*: *ss* = *concat sss* **by** *metis*
{
fix *i*
assume *i*: *i* < *length Cs*
then have *Cs* ! *i* ∈ *set Cs* **by** *simp*
from *MFun*(*1*)[*OF this IH*[*OF i*]]
have *ground* (*ts* ! *i*) = (*ground-mctxt* (*Cs* ! *i*) ∧ (∀ *a*∈*set* (*sss* ! *i*). *ground a*)) **.**
} note *IH* = *this*
note *conv* = *set-conv-nth*
have *?case* = ((∀ *x*∈*set ts*. *ground x*) = ((∀ *x*∈*set Cs*. *ground-mctxt x*) ∧ (∀ *a*∈*set sss*. ∀ *x*∈*set a*. *ground x*)))
unfolding *s ss* **by** *simp*
also have ... **unfolding** *conv*[*of ts*] *conv*[*of Cs*] *conv*[*of sss*] *len* **using** *IH* **by** *auto*
finally show *?case* **by** *simp*
qed

lemma *ground-fill-holes*:
assumes *nh*: *num-holes C* = *length ss*
shows *ground* (*fill-holes C ss*) = (*ground-mctxt C* ∧ (∀ *s* ∈ *set ss*. *ground s*))
by (*rule ground-eq-fill*[*OF eqfI*[*OF refl nh*]])

lemma *split-vars-ground'* [*simp*]:
ground-mctxt (*fst* (*split-vars t*))

by (*induct t*) *auto*

lemma *split-vars-funas-mctxt* [*simp*]:
funas-mctxt (*fst* (*split-vars t*)) = *funas-term t*
by (*induct t*) *auto*

lemma *less-eq-mctxt-prime*: $C \leq D \longleftrightarrow$ *less-eq-mctxt'* C D
proof
assume *less-eq-mctxt'* C D **then show** $C \leq D$
by (*induct C D rule*: *less-eq-mctxt'.induct*) (*auto simp*: *less-eq-mctxt-def intro*: *nth-equalityI*)
next
assume $C \leq D$ **then show** *less-eq-mctxt'* C D **unfolding** *less-eq-mctxt-def*
by (*induct C D rule*: *inf-mctxt.induct*)
(*auto split*: *if-splits simp*: *set-zip intro*!: *less-eq-mctxt'.intros nth-equalityI elim*!: *nth-equalityE*, *metis*)
qed

lemmas *less-eq-mctxt-induct* = *less-eq-mctxt'.induct*[*folded less-eq-mctxt-prime*, *consumes 1*]
lemmas *less-eq-mctxt-intros* = *less-eq-mctxt'.intros*[*folded less-eq-mctxt-prime*]

lemma *less-eq-mctxt-MHoleE2*:
assumes $C \leq$ *MHole*
obtains (*MHole*) C = *MHole*
using *assms* **unfolding** *less-eq-mctxt-prime* **by** (*cases C*, *auto*)

lemma *less-eq-mctxt-MVarE2*:
assumes $C \leq$ *MVar v*
obtains (*MHole*) C = *MHole* | (*MVar*) C = *MVar v*
using *assms* **unfolding** *less-eq-mctxt-prime* **by** (*cases C*) *auto*

lemma *less-eq-mctxt-MFunE2*:
assumes $C \leq$ *MFun f ds*
obtains (*MHole*) C = *MHole*
| (*MFun*) *cs* **where** C = *MFun f cs length cs* = *length ds* $\bigwedge i.\ i <$ *length cs* $\Longrightarrow$ *cs* ! $i \leq$ *ds* ! i
using *assms* **unfolding** *less-eq-mctxt-prime* **by** (*cases C*) *auto*

lemmas *less-eq-mctxtE2* = *less-eq-mctxt-MHoleE2 less-eq-mctxt-MVarE2 less-eq-mctxt-MFunE2*

lemma *less-eq-mctxt-MVarE1*:
assumes *MVar v* $\leq D$
obtains (*MVar*) D = *MVar v*
using *assms* **by** (*cases D*) (*auto elim*: *less-eq-mctxtE2*)

lemma *MHole-Bot* [*simp*]: *MHole* $\leq D$

by (*simp add*: *less-eq-mctxt-intros*(*1*))

lemma *less-eq-mctxt-MFunE1*:
assumes *MFun f cs* ≤ *D*
obtains (*MFun*) *ds* **where** *D* = *MFun f ds length cs* = *length ds* ⋀*i*. *i* < *length cs* ⟹ *cs* ! *i* ≤ *ds* ! *i*
using *assms* **by** (*cases D*) (*auto elim*: *less-eq-mctxtE2*)

lemma *length-unfill-holes* [*simp*]:
assumes *C* ≤ *mctxt-of-term t*
shows *length* (*unfill-holes C t*) = *num-holes C*
using *assms*
proof (*induct C t rule*: *unfill-holes.induct*)
case (*3 f Cs g ts*) **with** *3*(*1*)[*OF - nth-mem*] *3*(*2*) **show** *?case*
by (*auto simp*: *less-eq-mctxt-def length-concat*
intro!: *cong*[*of sum-list*, *OF refl*] *nth-equalityI elim*!: *nth-equalityE*)
qed (*auto simp*: *less-eq-mctxt-def*)

lemma *map-vars-mctxt-id* [*simp*]:
map-vars-mctxt (λ *x*. *x*) *C* = *C*
by (*induct C*, *auto intro*: *nth-equalityI*)

lemma *split-vars-eqf-subst-map-vars-term*:
t · *σ* =*f* (*map-vars-mctxt vw* (*fst* (*split-vars t*)), *map σ* (*snd* (*split-vars t*)))
proof (*induct t*)
case (*Fun f ts*)
have *?case* = (*Fun f* (*map* (λ*t*. *t* · *σ*) *ts*)
=*f* (*MFun f* (*map* (*map-vars-mctxt vw* ∘ (*fst* ∘ *split-vars*)) *ts*), *concat* (*map* (*map σ* ∘ (*snd* ∘ *split-vars*)) *ts*)))
by (*simp add*: *map-concat*)
also have ...
proof (*rule eqf-MFunI*, *simp*, *simp*, *unfold length-map*)
fix *i*
assume *i*: *i* < *length ts*
then have *mem*: *ts* ! *i* ∈ *set ts* **by** *auto*
show *map* (λ*t*. *t* · *σ*) *ts* ! *i* =*f* (*map* (*map-vars-mctxt vw* ∘ (*fst* ∘ *split-vars*)) *ts* ! *i*, *map* (*map σ* ∘ (*snd* ∘ *split-vars*)) *ts* ! *i*)
using *Fun*[*OF mem*] *i* **by** *auto*
qed
finally show *?case* **by** *simp*
qed *auto*

lemma *split-vars-eqf-subst*: *t* · *σ* =*f* (*fst* (*split-vars t*), (*map σ* (*snd* (*split-vars t*))))
using *split-vars-eqf-subst-map-vars-term*[*of t σ* λ *x*. *x*] **by** *simp*

lemma *split-vars-fill-holes*:

assumes $C = fst\ (split\text{-}vars\ s)$ **and** $ss = map\ Var\ (snd\ (split\text{-}vars\ s))$
shows $fill\text{-}holes\ C\ ss = s$ **using** *assms*
by (*metis eqfE*(*1*) *split-vars-eqf-subst subst-apply-term-empty*)

lemma *fill-unfill-holes*:
assumes $C \leq mctxt\text{-}of\text{-}term\ t$
shows $fill\text{-}holes\ C\ (unfill\text{-}holes\ C\ t) = t$
using *assms*
proof (*induct C t rule*: *unfill-holes.induct*)
case (*3 f Cs g ts*) **with** *3*(*1*)[*OF - nth-mem*] *3*(*2*) **show** *?case*
by (*auto simp*: *less-eq-mctxt-def intro*!: *fill-holes-arbitrary elim*!: *nth-equalityE*)
qed (*auto simp*: *less-eq-mctxt-def split*: *if-splits*)

lemma *hole-poss-list-length*:
$length\ (hole\text{-}poss\text{-}list\ D) = num\text{-}holes\ D$
by (*induct D*) (*auto simp*: *length-concat intro*!: *nth-sum-listI*)

lemma *unfill-holles-hole-poss-list-length*:
assumes $C \leq mctxt\text{-}of\text{-}term\ t$
shows $length\ (unfill\text{-}holes\ C\ t) = length\ (hole\text{-}poss\text{-}list\ C)$ **using** *assms*
proof (*induct C arbitrary*: *t*)
case (*MVar x*)
then have [*simp*]: $t = Var\ x$ **by** (*cases t*) (*auto dest*: *less-eq-mctxt-MVarE1*)
show *?case* **by** *simp*
next
case (*MFun f ts*) **then show** *?case*
by (*cases t*) (*auto simp*: *length-concat comp-def*
elim!: *less-eq-mctxt-MFunE1 less-eq-mctxt-MVarE1 intro*!: *nth-sum-listI*)
qed *auto*

lemma *unfill-holes-to-subst-at-hole-poss*:
assumes $C \leq mctxt\text{-}of\text{-}term\ t$
shows $unfill\text{-}holes\ C\ t = map\ ((|\text{-})\ t)\ (hole\text{-}poss\text{-}list\ C)$ **using** *assms*
proof (*induct C arbitrary*: *t*)
case (*MVar x*)
then show *?case* **by** (*cases t*) (*auto elim*: *less-eq-mctxt-MVarE1*)
next
case (*MFun f ts*)
from *MFun*(*2*) **obtain** *ss* **where** [*simp*]: $t = Fun\ f\ ss$ **and** *l*: $length\ ts = length\ ss$
by (*cases t*) (*auto elim*: *less-eq-mctxt-MFunE1*)
let *?ts* = $map\ (\lambda i.\ unfill\text{-}holes\ (ts\ !\ i)\ (ss\ !\ i))\ [0..<length\ ts]$
let *?ss* = $map\ (\lambda\ x.\ map\ ((|\text{-})\ (Fun\ f\ ss))\ (case\ x\ of\ (x,\ y) \Rightarrow map\ ((\#)\ x)\ (hole\text{-}poss\text{-}list\ y)))\ (zip\ [0..<length\ ts]\ ts)$
have *eq-l* [*simp*]: $length\ (concat\ ?ts) = length\ (concat\ ?ss)$ **using** *MFun*
by (*auto simp*: *length-concat comp-def elim*!: *less-eq-mctxt-MFunE1 split*!: *prod.splits intro*!: *nth-sum-listI*)

{fix *i* **assume** *ass*: *i* < *length* (*concat ?ts*)
then have *lss*: *i* < *length* (*concat ?ss*) **by** *auto*
obtain *m n* **where** [*simp*]: *concat-index-split* (*0*, *i*) *?ts* = (*m*, *n*) **by** *fastforce*
then have [*simp*]: *concat-index-split* (*0*, *i*) *?ss* = (*m*, *n*) **using** *concat-index-split-unique*[*OF ass*, *of ?ss 0*] *MFun*(*2*)
by (*auto simp*: *unfill-holles-hole-poss-list-length*[*of ts* ! *i ss* ! *i* **for** *i*]
simp del: *length-unfill-holes elim*!: *less-eq-mctxt-MFunE1*)
from *concat-index-split-less-length-concat*(*2*−)[*OF ass*] *concat-index-split-less-length-concat*(*2*−)[*OF lss*]
have *concat ?ts* ! *i* = *concat ?ss*! *i* **using** *MFun*(*1*)[*OF nth-mem*, *of m ss* ! *m*] *MFun*(*2*)
by (*auto elim*!: *less-eq-mctxt-MFunE1*)**}** **note** *nth* = *this*
show *?case* **using** *MFun*
by (*auto simp*: *comp-def map-concat length-concat*
elim!: *less-eq-mctxt-MFunE1 split*!: *prod.splits*
intro!: *nth-equalityI nth-sum-listI nth*)
qed *auto*

lemma *hole-poss-split-varposs-list-length* [*simp*]:
length (*hole-poss-list* (*fst* (*split-vars t*))) = *length* (*varposs-list t*)
by (*induct t*)(*auto simp*: *length-concat comp-def intro*!: *nth-sum-listI*)

lemma *hole-poss-split-vars-varposs-list*:
hole-poss-list (*fst* (*split-vars t*)) = *varposs-list t*
proof (*induct t*)
case (*Fun f ts*)
let *?ts* = *poss-args hole-poss-list* (*map* (*fst* ∘ *split-vars*) *ts*)
let *?ss* = *poss-args varposs-list ts*
have *len*: *length* (*concat ?ts*) = *length* (*concat ?ss*) *length ?ts* = *length ?ss*
∀ *i* < *length ?ts*. *length* (*?ts* ! *i*) = *length* (*?ss* ! *i*) **by** (*auto intro*: *eq-length-concat-nth*)
{fix *i* **assume** *ass*: *i* < *length* (*concat ?ts*)
then have *lss*: *i* < *length* (*concat ?ss*) **using** *len* **by** *auto*
obtain *m n* **where** *int*: *concat-index-split* (*0*, *i*) *?ts* = (*m*, *n*) **by** *fastforce*
then have [*simp*]: *concat-index-split* (*0*, *i*) *?ss* = (*m*, *n*) **using** *concat-index-split-unique*[*OF ass len*(*2*−)] **by** *auto*
from *concat-index-split-less-length-concat*(*2*−)[*OF ass int*] *concat-index-split-less-length-concat*(*2*−)[*OF lss*]
have *concat ?ts* ! *i* = *concat ?ss*! *i* **using** *Fun*[*OF nth-mem*, *of m*] **by** *auto***}**
then show *?case* **using** *len* **by** (*auto intro*: *nth-equalityI*)
qed *auto*

lemma *funas-term-fill-holes-iff*: *num-holes C* = *length ts* ⟹
g ∈ *funas-term* (*fill-holes C ts*) ⟷ *g* ∈ *funas-mctxt C* ∨ (∃ *t* ∈ *set ts*. *g* ∈ *funas-term t*)
proof (*induct C ts rule*: *fill-holes-induct*)
case (*MFun f Cs ts*)
have (∃ *i* < *length Cs*. *g* ∈ *funas-term* (*fill-holes* (*Cs* ! *i*) (*partition-holes* (*concat*

(*partition-holes ts Cs*)) *Cs* ! *i*)))
  ⟷ (∃ *C* ∈ *set Cs*. *g* ∈ *funas-mctxt C*) ∨ (∃ *us* ∈ *set* (*partition-holes ts Cs*). ∃ *t* ∈ *set us*. *g* ∈ *funas-term t*)
  using *MFun* **by** (*auto simp*: *ex-set-conv-ex-nth*) *blast*
 then show *?case* **by** *auto*
qed *auto*

lemma *vars-term-fill-holes* [*simp*]:
 num-holes C = *length ts* ⟹ *ground-mctxt C* ⟹
  vars-term (*fill-holes C ts*) = ⋃(*vars-term* ‘ *set ts*)
proof (*induct C arbitrary*: *ts*)
 case *MHole*
 then show *?case* **by** (*cases ts*) *simp-all*
next
 case (*MFun f Cs*)
 then have ∗: *length* (*partition-holes ts Cs*) = *length Cs* **by** *simp*
 let *?f* = λ*x*. ⋃ *y* ∈ *set x*. *vars-term y*
 show *?case*
  using *MFun*
  unfolding *partition-holes-fill-holes-conv*
  by (*simp add*: *UN-upt-len-conv* [*OF* ∗, *of ?f*] *UN-set-partition-by*)
qed *simp*

lemma *funas-mctxt-fill-holes* [*simp*]:
 assumes *num-holes C* = *length ts*
 shows *funas-term* (*fill-holes C ts*) = *funas-mctxt C* ∪ ⋃(*set* (*map funas-term ts*))
 using *funas-term-fill-holes-iff*[*OF assms*] **by** *auto*

lemma *funas-mctxt-fill-holes-mctxt* [*simp*]:
 assumes *num-holes C* = *length Ds*
 shows *funas-mctxt* (*fill-holes-mctxt C Ds*) = *funas-mctxt C* ∪ ⋃(*set* (*map funas-mctxt Ds*))
 (**is** *?f C Ds* = *?g C Ds*)
using *assms*
proof (*induct C arbitrary*: *Ds*)
 case *MHole*
 then show *?case* **by** (*cases Ds*) *simp-all*
next
 case (*MFun f Cs*)
 then have *num-holes*: *sum-list* (*map num-holes Cs*) = *length Ds* **by** *simp*
 let *?ys* = *partition-holes Ds Cs*
 have ⋀*i*. *i* < *length Cs* ⟹ *?f* (*Cs* ! *i*) (*?ys* ! *i*) = *?g* (*Cs* ! *i*) (*?ys* ! *i*)
  using *MFun* **by** (*metis nth-mem num-holes.simps*(3) *length-partition-holes-nth*)
 then have (⋃ *i* ∈ {*0* ..< *length Cs*}. *?f* (*Cs* ! *i*) (*?ys* ! *i*)) =
  (⋃ *i* ∈ {*0* ..< *length Cs*}. *?g* (*Cs* ! *i*) (*?ys* ! *i*)) **by** *simp*
 then show *?case*

using *num-holes*
unfolding *partition-holes-fill-holes-mctxt-conv*
by (*simp add*: *UN-Un-distrib UN-upt-len-conv* [*of* - - $\lambda x. \bigcup$ (*set x*)] *UN-set-partition-by-map*)
qed *simp*

end
theory *Ground-MCtxt*
imports
Multihole-Context
Regular-Tree-Relations.Ground-Terms
Regular-Tree-Relations.Ground-Ctxt
begin

2.2 Ground multihole context

datatype (*gfuns-mctxt*: *'f*) *gmctxt* = *GMHole* | *GMFun 'f 'f gmctxt list*

2.2.1 Basic function on ground mutlihole contexts

primrec *gmctxt-of-gterm* :: *'f gterm* ⇒ *'f gmctxt* **where**
gmctxt-of-gterm (*GFun f ts*) = *GMFun f* (*map gmctxt-of-gterm ts*)

fun *num-gholes* :: *'f gmctxt* ⇒ *nat* **where**
num-gholes GMHole = *Suc 0*
| *num-gholes* (*GMFun - ctxts*) = *sum-list* (*map num-gholes ctxts*)

primrec *gterm-of-gmctxt* :: *'f gmctxt* ⇒ *'f gterm* **where**
gterm-of-gmctxt (*GMFun f Cs*) = *GFun f* (*map gterm-of-gmctxt Cs*)

primrec *term-of-gmctxt* :: *'f gmctxt* ⇒ (*'f*, *'v*) *term* **where**
term-of-gmctxt (*GMFun f Cs*) = *Fun f* (*map term-of-gmctxt Cs*)

primrec *gmctxt-of-gctxt* :: *'f gctxt* ⇒ *'f gmctxt* **where**
gmctxt-of-gctxt $\Box_G$ = *GMHole*
| *gmctxt-of-gctxt* (*GMore f ss C ts*) =
GMFun f (*map gmctxt-of-gterm ss* @ *gmctxt-of-gctxt C* # *map gmctxt-of-gterm ts*)

fun *gctxt-of-gmctxt* :: *'f gmctxt* ⇒ *'f gctxt* **where**
gctxt-of-gmctxt GMHole = $\Box_G$
| *gctxt-of-gmctxt* (*GMFun f Cs*) = (*let n* = *length* (*takeWhile* (λ *C*. *num-gholes C* = *0*) *Cs*) *in*
(*if n* < *length Cs then*
GMore f (*map gterm-of-gmctxt* (*take n Cs*)) (*gctxt-of-gmctxt* (*Cs* ! *n*)) (*map gterm-of-gmctxt* (*drop* (*Suc n*) *Cs*))
else undefined))

primrec *gmctxt-of-mctxt* :: (*'f*, *'v*) *mctxt* ⇒ *'f gmctxt* **where**
gmctxt-of-mctxt MHole = *GMHole*
| *gmctxt-of-mctxt* (*MFun f Cs*) = *GMFun f* (*map gmctxt-of-mctxt Cs*)

```
primrec mctxt-of-gmctxt :: 'f gmctxt ⇒ ('f, 'v) mctxt where
  mctxt-of-gmctxt GMHole = MHole
| mctxt-of-gmctxt (GMFun f Cs) = MFun f (map mctxt-of-gmctxt Cs)

fun funas-gmctxt where
  funas-gmctxt (GMFun f Cs) = {(f, length Cs)} ∪ ⋃(funas-gmctxt ‘ set Cs) |
  funas-gmctxt - = {}

abbreviation partition-gholes xs Cs ≡ partition-by xs (map num-gholes Cs)

fun fill-gholes :: 'f gmctxt ⇒ 'f gterm list ⇒ 'f gterm where
  fill-gholes GMHole [t] = t
| fill-gholes (GMFun f cs) ts = GFun f (map (λ i. fill-gholes (cs ! i)
    (partition-gholes ts cs ! i)) [0 ..< length cs])

fun fill-gholes-gmctxt :: 'f gmctxt ⇒ 'f gmctxt list ⇒ 'f gmctxt where
  fill-gholes-gmctxt GMHole [] = GMHole |
  fill-gholes-gmctxt GMHole [t] = t |
  fill-gholes-gmctxt (GMFun f cs) ts = (GMFun f (map (λ i. fill-gholes-gmctxt (cs
! i)
    (partition-gholes ts cs ! i)) [0 ..< length cs]))
```

2.2.2 An inverse of *fill-gholes*

```
fun unfill-gholes :: 'f gmctxt ⇒ 'f gterm ⇒ 'f gterm list where
  unfill-gholes GMHole t = [t]
| unfill-gholes (GMFun g Cs) (GFun f ts) = (if f = g ∧ length ts = length Cs then
    concat (map (λi. unfill-gholes (Cs ! i) (ts ! i)) [0..<length ts]) else undefined)

fun sup-gmctxt-args :: 'f gmctxt ⇒ 'f gmctxt ⇒ 'f gmctxt list where
  sup-gmctxt-args GMHole D = [D] |
  sup-gmctxt-args C GMHole = replicate (num-gholes C) GMHole |
  sup-gmctxt-args (GMFun f Cs) (GMFun g Ds) =
   (if f = g ∧ length Cs = length Ds then concat (map (case-prod sup-gmctxt-args)
(zip Cs Ds))
    else undefined)

fun ghole-poss :: 'f gmctxt ⇒ pos set where
  ghole-poss GMHole = {[]} |
  ghole-poss (GMFun f cs) = ⋃(set (map (λ i. (λ p. i # p) ‘ ghole-poss (cs ! i))
[0 ..< length cs]))

abbreviation poss-rec f ts ≡ map2 (λ i t. map ((#) i) (f t)) ([0 ..< length ts]) ts
fun ghole-poss-list :: 'f gmctxt ⇒ pos list where
  ghole-poss-list GMHole = [[]]
| ghole-poss-list (GMFun f cs) = concat (poss-rec ghole-poss-list cs)
```

fun *poss-gmctxt* :: *'f gmctxt* ⇒ *pos set* **where**
poss-gmctxt GMHole = {} |
poss-gmctxt (*GMFun f cs*) = {[]} ∪ ⋃(*set* (*map* (λ *i*. (λ *p*. *i* # *p*) ‘ *poss-gmctxt* (*cs* ! *i*)) [*0* ..< *length cs*]))

lemma *poss-simps* [*simp*]:
ghole-poss (*GMFun f Cs*) = {*i* # *p* | *i p*. *i* < *length Cs* ∧ *p* ∈ *ghole-poss* (*Cs* ! *i*)}
poss-gmctxt (*GMFun f Cs*) = {[]} ∪ {*i* # *p* | *i p*. *i* < *length Cs* ∧ *p* ∈ *poss-gmctxt* (*Cs* ! *i*)}
by *auto*

fun *ghole-num-bef-pos* **where**
ghole-num-bef-pos [] - = *0* |
ghole-num-bef-pos (*i* # *q*) (*GMFun f Cs*) = *sum-list* (*map num-gholes* (*take i Cs*)) + *ghole-num-bef-pos q* (*Cs* ! *i*)

fun *ghole-num-at-pos* **where**
ghole-num-at-pos [] *C* = *num-gholes C* |
ghole-num-at-pos (*i* # *q*) (*GMFun f Cs*) = *ghole-num-at-pos q* (*Cs* ! *i*)

fun *subgm-at* :: *'f gmctxt* ⇒ *pos* ⇒ *'f gmctxt* **where**
subgm-at C [] = *C*
| *subgm-at* (*GMFun f Cs*) (*i* # *p*) = *subgm-at* (*Cs* ! *i*) *p*

definition *gmctxt-subtgm-at-fill-args* **where**
gmctxt-subtgm-at-fill-args p C ts = *take* (*ghole-num-at-pos p C*) (*drop* (*ghole-num-bef-pos p C*) *ts*)

instantiation *gmctxt* :: (*type*) *inf*
begin

fun *inf-gmctxt* :: *'a gmctxt* ⇒ *'a gmctxt* ⇒ *'a gmctxt* **where**
GMHole ⊓ *D* = *GMHole* |
C ⊓ *GMHole* = *GMHole* |
GMFun f Cs ⊓ *GMFun g Ds* =
(*if f* = *g* ∧ *length Cs* = *length Ds then GMFun f* (*map* (*case-prod* (⊓)) (*zip Cs Ds*))
else GMHole)

instance ..
end

instantiation *gmctxt* :: (*type*) *sup*
begin

fun *sup-gmctxt* :: *'a gmctxt* ⇒ *'a gmctxt* ⇒ *'a gmctxt* **where**

GMHole ⊔ *D* = *D* |
C ⊔ *GMHole* = *C* |
GMFun f Cs ⊔ *GMFun g Ds* =
(*if f* = *g* ∧ *length Cs* = *length Ds then GMFun f* (*map* (*case-prod* (⊔)) (*zip Cs Ds*))
else undefined)

instance ..
end

2.2.3 Orderings and compatibility of ground multihole contexts

inductive *less-eq-gmctxt* :: *'f gmctxt* ⇒ *'f gmctxt* ⇒ *bool* **where**
base [*simp*]: *less-eq-gmctxt GMHole u*
| *ind*[*intro*]: *length cs* = *length ds* ⟹ (⋀*i*. *i* < *length cs* ⟹ *less-eq-gmctxt* (*cs* ! *i*) (*ds* ! *i*)) ⟹
less-eq-gmctxt (*GMFun f cs*) (*GMFun f ds*)

inductive-set *comp-gmctxt* :: (*'f gmctxt* × *'f gmctxt*) *set* **where**
GMHole1 [*simp*]: (*GMHole*, *D*) ∈ *comp-gmctxt* |
GMHole2 [*simp*]: (*C*, *GMHole*) ∈ *comp-gmctxt* |
GMFun [*intro*]: *f* = *g* ⟹ *length Cs* = *length Ds* ⟹ ∀ *i* < *length Ds*. (*Cs* ! *i*, *Ds* ! *i*) ∈ *comp-gmctxt* ⟹
(*GMFun f Cs*, *GMFun g Ds*) ∈ *comp-gmctxt*

definition *gmctxt-closing* **where**
gmctxt-closing C D ⟷ *less-eq-gmctxt C D* ∧ *ghole-poss D* ⊆ *ghole-poss C*

inductive *eq-gfill* (‹(-/ =$_{Gf}$ -)› [*51*, *51*] *50*) **where**
eqfI [*intro*]: *t* = *fill-gholes D ss* ⟹ *num-gholes D* = *length ss* ⟹ *t* =$_{Gf}$ (*D*, *ss*)

2.2.4 Conversions from and to ground multihole contexts

lemma *num-gholes-o-gmctxt-of-gterm* [*simp*]:
num-gholes ∘ *gmctxt-of-gterm* = (λ*x*. *0*)
by (*rule ext*, *induct-tac x*) *simp-all*

lemma *mctxt-of-term-term-of-mctxt-id* [*simp*]:
num-gholes C = *0* ⟹ *gmctxt-of-gterm* (*gterm-of-gmctxt C*) = *C*
by (*induct C*) (*simp-all add*: *map-idI*)

lemma *num-holes-mctxt-of-term* [*simp*]:
num-gholes (*gmctxt-of-gterm t*) = *0*
by (*induct t*) *simp-all*

lemma *num-gholes-gmctxt-of-mctxt* [*simp*]:
ground-mctxt C ⟹ *num-gholes* (*gmctxt-of-mctxt C*) = *num-holes C*
by (*induct C*) (*auto intro*: *nth-sum-listI*)

lemma *num-holes-mctxt-of-gmctxt* [*simp*]:
num-holes (*mctxt-of-gmctxt C*) = *num-gholes C*
by (*induct C*) (*auto intro*: *nth-sum-listI*)

lemma *num-holes-mctxt-of-gmctxt-fun-comp* [*simp*]:
num-holes ∘ *mctxt-of-gmctxt* = *num-gholes*
by (*auto simp*: *comp-def*)

lemma *gmctxt-of-gctxt-num-gholes* [*simp*]:
num-gholes (*gmctxt-of-gctxt C*) = *Suc 0*
by (*induct C*) *auto*

lemma *ground-mctxt-list-num-gholes-gmctxt-of-mctxt-conv* [*simp*]:
∀ *x*∈*set Cs*. *ground-mctxt x* ⟹ *map* (*num-gholes* ∘ *gmctxt-of-mctxt*) *Cs* = *map num-holes Cs*
by *auto*

lemma *num-gholes-map-gmctxt* [*simp*]:
num-gholes (*map-gmctxt f C*) = *num-gholes C*
by (*induct C*) (*auto simp*: *comp-def*, *metis* (*no-types*, *lifting*) *map-eq-conv*)

lemma *map-num-gholes-map-gmctxt* [*simp*]:
map (*num-gholes* ∘ *map-gmctxt f*) *Cs* = *map num-gholes Cs*
by (*induct Cs*) *auto*

lemma *gterm-of-gmctxt-gmctxt-of-gterm-id* [*simp*]:
gterm-of-gmctxt (*gmctxt-of-gterm t*) = *t*
by (*induct t*) (*simp-all add*: *map-idI*)

lemma *no-gholes-gmctxt-of-gterm-gterm-of-gmctxt-id* [*simp*]:
num-gholes C = *0* ⟹ *gmctxt-of-gterm* (*gterm-of-gmctxt C*) = *C*
by (*induct C*) (*auto simp*: *comp-def intro*: *nth-equalityI*)

lemma *no-gholes-term-of-gterm-gterm-of-gmctxt* [*simp*]:
num-gholes C = *0* ⟹ *term-of-gterm* (*gterm-of-gmctxt C*) = *term-of-gmctxt C*
by (*induct C*) (*auto simp*: *comp-def intro*: *nth-equalityI*)

lemma *no-gholes-term-of-mctxt-mctxt-of-gmctxt* [*simp*]:
num-gholes C = *0* ⟹ *term-of-mctxt* (*mctxt-of-gmctxt C*) = *term-of-gmctxt C*
by (*induct C*) (*auto simp*: *comp-def intro*: *nth-equalityI*)

lemma *nthWhile-gmctxt-of-gctxt* [*simp*]:
length (*takeWhile* (λ*C*. *num-gholes C* = *0*) (*map gmctxt-of-gterm ss* @ *gmctxt-of-gctxt C* # *ts*)) = *length ss*
by (*induct ss*) *auto*

lemma *sum-list-nthWhile-length* [*simp*]:
sum-list (*map num-gholes Cs*) = *Suc 0* ⟹ *length* (*takeWhile* (λ*C*. *num-gholes*

```
C = 0) Cs) < length Cs
  by (induct Cs) auto

lemma gctxt-of-gmctxt-gmctxt-of-gctxt [simp]:
  gctxt-of-gmctxt (gmctxt-of-gctxt C) = C
  by (induct C) (auto simp: Let-def comp-def nth-append)

lemma gmctxt-of-gctxt-GMHole-Hole:
  gmctxt-of-gctxt C = GMHole ⟹ C = □G
  by (metis gctxt-of-gmctxt.simps(1) gctxt-of-gmctxt-gmctxt-of-gctxt)

lemma gmctxt-of-gctxt-gctxt-of-gmctxt:
  num-gholes C = Suc 0 ⟹ gmctxt-of-gctxt (gctxt-of-gmctxt C) = C
proof (induct C)
  case (GMFun f Cs)
  then obtain i where nth: i < length Cs i = length (takeWhile (λC. num-gholes
C = 0) Cs)
    using sum-list-nthWhile-length by auto
 then have 0 < num-gholes (Cs ! i) unfolding nth(2) using nth-length-takeWhile
    by auto
  from nth(1) this have num: num-gholes (Cs ! i) = Suc 0 using GMFun(2)
    by (auto elim!: sum-list-1-E)
  then have [simp]: map (gmctxt-of-gterm ∘ gterm-of-gmctxt) (drop (Suc i) Cs)
= drop (Suc i) Cs using GMFun(2) nth(1)
    by (auto elim!: sum-list-1-E simp: comp-def intro!: nth-equalityI)
   (metis add.commute add-Suc-right lessI less-diff-conv no-gholes-gmctxt-of-gterm-gterm-of-gmctxt-id
not-add-less1)
  have [simp]: map (gmctxt-of-gterm ∘ gterm-of-gmctxt) (take i Cs) = take i Cs
    using nth(1) unfolding nth(2) by (induct Cs) auto
  show ?case using id-take-nth-drop[OF nth(1)]
    by (auto simp: Let-def GMFun(1)[OF nth-mem[OF nth(1)] num] simp flip:
nth(2))
qed auto

lemma inj-gmctxt-of-gctxt: inj gmctxt-of-gctxt
  unfolding inj-def by (metis gctxt-of-gmctxt-gmctxt-of-gctxt)

lemma inj-gctxt-of-gmctxt-on-single-hole:
  inj-on gctxt-of-gmctxt (Collect (λ C. num-gholes C = Suc 0))
 by (metis (mono-tags, lifting) gmctxt-of-gctxt-gctxt-of-gmctxt inj-onI mem-Collect-eq)

lemma gctxt-of-gmctxt-hole-dest:
  num-gholes C = Suc 0 ⟹ gctxt-of-gmctxt C = □G ⟹ C = GMHole
  by (cases C) (auto simp: Let-def split!: if-splits)

lemma mctxt-of-gmctxt-inv [simp]:
  gmctxt-of-mctxt (mctxt-of-gmctxt C) = C
  by (induct C) (simp-all add: map-idI)
```

lemma *ground-mctxt-of-gmctxt* [*simp*]:
ground-mctxt (*mctxt-of-gmctxt C*)
by (*induct C*) *auto*

lemma *ground-mctxt-of-gmctxt'* [*simp*]:
mctxt-of-gmctxt C = *MFun f D* ⟹ *ground-mctxt* (*MFun f D*)
by (*induct C*) *auto*

lemma *gmctxt-of-mctxt-inv* [*simp*]:
ground-mctxt C ⟹ *mctxt-of-gmctxt* (*gmctxt-of-mctxt C*) = *C*
by (*induct C*) (*auto 0 0 intro*!: *nth-equalityI*)

lemma *ground-mctxt-of-gmctxtD*:
ground-mctxt C ⟹ ∃ *D*. *C* = *mctxt-of-gmctxt D*
by (*metis gmctxt-of-mctxt-inv*)

lemma *inj-mctxt-of-gmctxt*: *inj-on mctxt-of-gmctxt X*
by (*metis inj-on-def mctxt-of-gmctxt-inv*)

lemma *inj-gmctxt-of-mctxt-ground*:
inj-on gmctxt-of-mctxt (*Collect ground-mctxt*)
using *gmctxt-of-mctxt-inv inj-on-def* **by** *force*

lemma *map-gmctxt-comp* [*simp*]:
map-gmctxt f (*map-gmctxt g C*) = *map-gmctxt* (*f* ∘ *g*) *C*
by (*induct C*) *auto*

lemma *map-mctxt-of-gmctxt*:
map-mctxt f (*mctxt-of-gmctxt C*) = *mctxt-of-gmctxt* (*map-gmctxt f C*)
by (*induct C*) *auto*

lemma *map-gmctxt-of-mctxt*:
ground-mctxt C ⟹ *map-gmctxt f* (*gmctxt-of-mctxt C*) = *gmctxt-of-mctxt* (*map-mctxt f C*)
by (*induct C*) *auto*

lemma *map-gmctxt-nempty* [*simp*]:
C ≠ *GMHole* ⟹ *map-gmctxt f C* ≠ *GMHole*
by (*cases C*) *auto*

lemma *vars-mctxt-of-gmctxt* [*simp*]:
vars-mctxt (*mctxt-of-gmctxt C*) = {}
by (*induct C*) *auto*

lemma *vars-mctxt-of-gmctxt-subseteq* [*simp*]:
vars-mctxt (*mctxt-of-gmctxt C*) ⊆ *Q* ⟷ *True*
by *auto*

2.2.5 Equivalences and simplification rules

lemma *eqgfE*:
assumes $t =_{Gf} (D, ss)$ **shows** *t = fill-gholes D ss num-gholes D = length ss*
using *assms*[*unfolded eq-gfill.simps*] **by** *auto*

lemma *eqgf-GMHoleE*:
assumes $t =_{Gf} (GMHole, ss)$ **shows** $ss = [t]$ **using** *eqgfE*[*OF assms*]
by (*cases ss*) *auto*

lemma *eqgf-GMFunE*:
assumes $s =_{Gf} (GMFun\ f\ Cs, ss)$
obtains *ts sss* **where** *s = GFun f ts length ts = length Cs length sss = length Cs*
$\bigwedge i.\ i < length\ Cs \Longrightarrow ts\ !\ i =_{Gf} (Cs\ !\ i, sss\ !\ i)\ ss = concat\ sss$
proof −
from *eqgfE*[*OF assms*] **have** *fh*: *s = fill-gholes* (*GMFun f Cs*) *ss*
and *nh*: *sum-list* (*map num-gholes Cs*) = *length ss* **by** *auto*
from *fh* **obtain** *ts* **where** *s*: *s = GFun f ts* **by** (*cases s*, *auto*)
from *fh*[*unfolded s*]
have *ts*: *ts = map* (*λi. fill-gholes* (*Cs ! i*) (*partition-gholes ss Cs ! i*)) [*0..<length Cs*]
(**is** - = *map* (*?f Cs ss*) -)
by *auto*
let *?sss = partition-gholes ss Cs*
from *nh* **have** ∗: *length ?sss = length Cs*
$\bigwedge i.\ i < length\ Cs \Longrightarrow ts\ !\ i =_{Gf} (Cs\ !\ i, ?sss\ !\ i)\ ss = concat\ ?sss$
by (*auto simp*: *ts length-partition-by-nth*)
have *len*: *length ts = length Cs* **unfolding** *ts* **by** *auto*
assume *ass*: $\bigwedge ts\ sss.\ s = GFun\ f\ ts \Longrightarrow$
length ts = length Cs $\Longrightarrow$
length sss = length Cs $\Longrightarrow (\bigwedge i.\ i < length\ Cs \Longrightarrow ts\ !\ i =_{Gf} (Cs\ !\ i, sss\ !\ i)) \Longrightarrow ss = concat\ sss \Longrightarrow thesis$
show *thesis* **by** (*rule ass*[*OF s len* ∗])
qed

lemma *partition-holes-subseteq* [*simp*]:
assumes *sum-list* (*map num-holes Cs*) = *length xs i < length Cs*
and *x ∈ set* (*partition-holes xs Cs ! i*)
shows *x ∈ set xs*
using *assms partition-by-nth-nth-elem length-partition-by-nth*
by (*auto simp*: *in-set-conv-nth*) *fastforce*

lemma *partition-gholes-subseteq* [*simp*]:
assumes *sum-list* (*map num-gholes Cs*) = *length xs i < length Cs*
and *x ∈ set* (*partition-gholes xs Cs ! i*)
shows *x ∈ set xs*
using *assms partition-by-nth-nth-elem length-partition-by-nth*
by (*auto simp*: *in-set-conv-nth*) *fastforce*

lemma *list-elem-to-partition-nth* [*elim*]:
assumes *sum-list* (*map num-gholes Cs*) = *length xs x* ∈ *set xs*
obtains *i* **where** *i* < *length Cs x* ∈ *set* (*partition-gholes xs Cs* ! *i*) **using** *assms*
by (*metis concat-partition-by in-set-idx length-map length-partition-by nth-concat-split nth-mem*)

lemma *partition-holes-fill-gholes-conv'*:
fill-gholes (*GMFun f Cs*) *ts* =
GFun f (*map* (*case-prod fill-gholes*) (*zip Cs* (*partition-gholes ts Cs*)))
unfolding *zip-nth-conv* [*of Cs partition-gholes ts Cs*, *simplified*]
and *partition-holes-fill-holes-conv* **by** *simp*

lemma *unfill-gholes-conv*:
assumes *length Cs* = *length ts*
shows *unfill-gholes* (*GMFun f Cs*) (*GFun f ts*) =
concat (*map* (*case-prod unfill-gholes*) (*zip Cs ts*)) **using** *assms*
by (*auto simp*: *zip-nth-conv* [*of Cs ts*, *simplified*] *comp-def*)

lemma *partition-holes-fill-gholes-gmctxt-conv*:
fill-gholes-gmctxt (*GMFun f Cs*) *ts* =
GMFun f [*fill-gholes-gmctxt* (*Cs* ! *i*) (*partition-gholes ts Cs* ! *i*). *i* ← [*0* ..< *length Cs*]]
by (*simp add*: *partition-by-nth take-map*)

lemma *partition-holes-fill-gholes-gmctxt-conv'*:
fill-gholes-gmctxt (*GMFun f Cs*) *ts* =
GMFun f (*map* (*case-prod fill-gholes-gmctxt*) (*zip Cs* (*partition-gholes ts Cs*)))
unfolding *zip-nth-conv* [*of Cs partition-gholes ts Cs*, *simplified*]
and *partition-holes-fill-gholes-gmctxt-conv* **by** *simp*

lemma *fill-gholes-no-holes* [*simp*]:
num-gholes C = *0* ⟹ *fill-gholes C* [] = *gterm-of-gmctxt C*
by (*induct C*) (*auto simp*: *partition-holes-fill-gholes-conv'*
simp del: *fill-gholes.simps intro*: *nth-equalityI*)

lemma *fill-gholes-gmctxt-no-holes* [*simp*]:
num-gholes C = *0* ⟹ *fill-gholes-gmctxt C* [] = *C*
by (*induct C*) (*auto intro*: *nth-equalityI*)

lemma *eqgf-GMFunI*:
assumes ⋀ *i*. *i* < *length Cs* ⟹ *ss* ! *i* =$_{Gf}$ (*Cs* ! *i*, *ts* ! *i*)
and *length Cs* = *length ss length ss* = *length ts*
shows *GFun f ss* =$_{Gf}$ (*GMFun f Cs*, *concat ts*) **using** *assms*
apply (*auto simp del*: *fill-gholes.simps*
simp: *partition-holes-fill-gholes-conv' intro*!: *eq-gfill.intros nth-equalityI*)
apply (*metis eqgfE length-map map-nth-eq-conv partition-by-concat-id*)
by (*metis eqgfE*(*2*) *length-concat nth-map-conv*)

lemma *length-partition-gholes-nth*:

assumes *sum-list* (*map num-gholes cs*) = *length ts*
and *i* < *length cs*
shows *length* (*partition-gholes ts cs* ! *i*) = *num-gholes* (*cs* ! *i*)
using *assms* **by** (*simp add*: *length-partition-by-nth*)

lemma *fill-gholes-induct2*[*consumes 2*, *case-names GMHole GMFun*]:
fixes *P* :: *'f gmctxt* ⇒ *'a list* ⇒ *'b list* ⇒ *bool*
assumes *len1*: *num-gholes C* = *length xs* **and** *len2*: *num-gholes C* = *length ys*
and *Hole*: ⋀*x y*. *P GMHole* [*x*] [*y*]
and *Fun*: ⋀*f Cs xs ys*. *sum-list* (*map num-gholes Cs*) = *length xs* ⟹
sum-list (*map num-gholes Cs*) = *length ys* ⟹
(⋀*i*. *i* < *length Cs* ⟹ *P* (*Cs* ! *i*) (*partition-gholes xs Cs* ! *i*) (*partition-gholes ys Cs* ! *i*)) ⟹
P (*GMFun f Cs*) (*concat* (*partition-gholes xs Cs*)) (*concat* (*partition-gholes ys Cs*))
shows *P C xs ys*
proof (*insert len1 len2*, *induct C arbitrary*: *xs ys*)
case *GMHole*
then show *?case* **using** *Hole* **by** (*cases xs*; *cases ys*) *auto*
next
case (*GMFun f Cs*)
then show *?case* **using** *Fun*[*of Cs xs ys f*] **by** (*auto simp*: *length-partition-by-nth*)
qed

lemma *fill-gholes-induct*[*consumes 1*, *case-names GMHole GMFun*]:
fixes *P* :: *'f gmctxt* ⇒ *'a list* ⇒ *bool*
assumes *len*: *num-gholes C* = *length xs*
and *Hole*: ⋀*x*. *P GMHole* [*x*]
and *Fun*: ⋀*f Cs xs*. *sum-list* (*map num-gholes Cs*) = *length xs* ⟹
(⋀*i*. *i* < *length Cs* ⟹ *P* (*Cs* ! *i*) (*partition-gholes xs Cs* ! *i*)) ⟹
P (*GMFun f Cs*) (*concat* (*partition-gholes xs Cs*))
shows *P C xs*
using *fill-gholes-induct2*[*of C xs xs* λ *C xs* -. *P C xs*] *assms* **by** *simp*

lemma *eq-gfill-induct* [*consumes 1*, *case-names GMHole GMFun*]:
assumes *t* =$_{Gf}$ (*C*, *ts*)
and ⋀*t*. *P t GMHole* [*t*]
and ⋀*f ss Cs ts*. ⟦*length Cs* = *length ss*; *sum-list* (*map num-gholes Cs*) = *length ts*;
∀ *i* < *length ss*. *ss* ! *i* =$_{Gf}$ (*Cs* ! *i*, *partition-gholes ts Cs* ! *i*) ∧
P (*ss* ! *i*) (*Cs* ! *i*) (*partition-gholes ts Cs* ! *i*)⟧
⟹ *P* (*GFun f ss*) (*GMFun f Cs*) *ts*
shows *P t C ts* **using** *assms*(*1*)
proof (*induct t arbitrary*: *C ts*)
case (*GFun f ss*)
{assume *C* = *GMHole* **and** *ts* = [*GFun f ss*]
then have *?case* **using** *assms*(*2*) **by** *auto***}**
moreover
{fix *Cs*

assume C: $C = GMFun\ f\ Cs$ **and** $sum\text{-}list\ (map\ num\text{-}gholes\ Cs) = length\ ts$
and $length\ Cs = length\ ss$
and $GFun\ f\ ss = fill\text{-}gholes\ (GMFun\ f\ Cs)\ ts$
moreover then have $\forall\, i < length\ ss.\ ss\ !\ i =_{Gf} (Cs\ !\ i, partition\text{-}gholes\ ts\ Cs\ !\ i)$
by (*auto simp del*: *fill-gholes.simps*
simp: *partition-holes-fill-gholes-conv′ length-partition-gholes-nth intro*!: *eq-gfill.intros*)
moreover have $\forall\, i < length\ ss.\ P\ (ss\ !\ i)\ (Cs\ !\ i)\ (partition\text{-}gholes\ ts\ Cs\ !\ i)$
using *GFun calculation*(5) *nth-mem* **by** *blast*
ultimately have *?case* **using** *assms*(3)[*of Cs ss ts f*] **by** *auto*}
ultimately show *?case* **using** *GFun*
by (*elim eq-gfill.cases*) (*auto simp del*: *fill-gholes.simps*,
metis GFun.prems eqgf-GMFunE eqgf-GMHoleE gterm.inject num-gholes.elims)
qed

lemma *nempty-ground-mctxt-gmctxt* [*simp*]:
$C \neq MHole \Longrightarrow ground\text{-}mctxt\ C \Longrightarrow gmctxt\text{-}of\text{-}mctxt\ C \neq GMHole$
by (*induct C*) *auto*

lemma *mctxt-of-gmctxt-fill-holes* [*simp*]:
assumes *num-gholes C* = *length ss*
shows *gterm-of-term* (*fill-holes* (*mctxt-of-gmctxt C*) (*map term-of-gterm ss*)) = *fill-gholes C ss* **using** *assms*
by (*induct rule*: *fill-gholes-induct*) *auto*

lemma *mctxt-of-gmctxt-terms-fill-holes*:
assumes *num-gholes C* = *length ss*
shows *gterm-of-term* (*fill-holes* (*mctxt-of-gmctxt C*) *ss*) = *fill-gholes C* (*map gterm-of-term ss*) **using** *assms*
by (*induct rule*: *fill-gholes-induct*) *auto*

lemma *ground-gmctxt-of-mctxt-gterm-fill-holes*:
assumes *num-holes C* = *length ss* **and** *ground-mctxt C*
shows *term-of-gterm* (*fill-gholes* (*gmctxt-of-mctxt C*) *ss*) = *fill-holes C* (*map term-of-gterm ss*) **using** *assms*
by (*induct rule*: *fill-holes-induct*)
(*auto simp*: *comp-def*, *metis* (*no-types*, *lifting*) *map-eq-conv num-gholes-gmctxt-of-mctxt*)

lemma *ground-gmctxt-of-gterm-of-term*:
assumes *num-holes C* = *length ss* **and** *ground-mctxt C*
shows *gterm-of-term* (*fill-holes C* (*map term-of-gterm ss*)) = *fill-gholes* (*gmctxt-of-mctxt C*) *ss* **using** *assms*
by (*induct rule*: *fill-holes-induct*)
(*auto simp*: *comp-def*, *metis* (*no-types*, *lifting*) *map-eq-conv num-gholes-gmctxt-of-mctxt*)

lemma *ground-gmctxt-of-mctxt-fill-holes* [*simp*]:
assumes *num-holes C* = *length ss* **and** *ground-mctxt C* $\forall\ s \in set\ ss.\ ground\ s$
shows *term-of-gterm* (*fill-gholes* (*gmctxt-of-mctxt C*) (*map gterm-of-term ss*)) =

fill-holes C ss **using** *assms*
by (*induct rule*: *fill-holes-induct*) *auto*

lemma *fill-holes-mctxt-of-gmctxt-to-fill-gholes*:
assumes *num-gholes C* = *length ss*
shows *fill-holes* (*mctxt-of-gmctxt C*) (*map term-of-gterm ss*) = *term-of-gterm* (*fill-gholes C ss*)
using *assms*
by (*metis ground-gmctxt-of-mctxt-gterm-fill-holes ground-mctxt-of-gmctxt mctxt-of-gmctxt-inv num-holes-mctxt-of-gmctxt*)

lemma *fill-gholes-gmctxt-of-gterm* [*simp*]:
fill-gholes (*gmctxt-of-gterm s*) [] = *s*
by (*induct s*) (*auto simp add*: *map-nth-eq-conv*)

lemma *fill-gholes-GMHole* [*simp*]:
length ss = *Suc 0* ⟹ *fill-gholes GMHole ss* = *ss* ! *0*
by (*cases ss*) *auto*

lemma *apply-gctxt-fill-gholes*:
C⟨*s*⟩$_G$ = *fill-gholes* (*gmctxt-of-gctxt C*) [*s*]
by (*induct C*) (*auto simp*: *partition-holes-fill-gholes-conv*′
simp del: *fill-gholes.simps intro*!: *nth-equalityI*)

lemma *fill-gholes-apply-gctxt*:
num-gholes C = *Suc 0* ⟹ *fill-gholes C* [*s*] = (*gctxt-of-gmctxt C*)⟨*s*⟩$_G$
by (*simp add*: *apply-gctxt-fill-gholes gmctxt-of-gctxt-gctxt-of-gmctxt*)

lemma *ctxt-of-gctxt-gctxt-of-gmctxt-apply*:
num-gholes C = *Suc 0* ⟹ *fill-holes* (*mctxt-of-gmctxt C*) [*s*] = (*ctxt-of-gctxt* (*gctxt-of-gmctxt C*))⟨*s*⟩
proof (*induct C*)
case (*GMFun f Cs*)
obtain *i* **where** *split*: *i* < *length Cs num-gholes* (*Cs* ! *i*) = *Suc 0*
∀ *j* < *length Cs*. *j* ≠ *i* ⟶ *num-gholes* (*Cs* ! *j*) = *0* **using** *GMFun*(*2*)
by *auto*
then have [*simp*]: *sum-list* (*take i* (*map num-gholes Cs*)) = *0*
by (*auto simp*: *sum-list-eq-0-iff dest*: *set-take-nth*)
from *split* **have** [*simp*]: *j* < *length Cs* ⟹ *j* ≠ *i* ⟹
fill-holes (*mctxt-of-gmctxt* (*Cs* ! *j*)) [] = *term-of-mctxt* (*mctxt-of-gmctxt* (*Cs* ! *j*)) **for** *j*
by (*intro fill-holes-term-of-mctxt*) *auto*
from *split* **have** [*simp*]: *gctxt-of-gmctxt* (*GMFun f Cs*) =
GMore f (*map gterm-of-gmctxt* (*take i Cs*)) (*gctxt-of-gmctxt* (*Cs* ! *i*)) (*map gterm-of-gmctxt* (*drop* (*Suc i*) *Cs*))
using *nth-length-takeWhile GMFun*(*2*) *sum-list-nthWhile-length* **by** (*auto simp*: *Let-def*)
show *?case* **using** *GMFun*(*1*)[*OF nth-mem*[*OF split*(*1*)] *split*(*2*)] *split*

by (*auto simp*: *min-def nth-append-Cons partition-by-nth simp del*: *gctxt-of-gmctxt.simps intro*!: *nth-equalityI*)
qed *auto*

lemma *fill-gholes-replicate* [*simp*]:
n = *length ss* ⟹ *fill-gholes* (*GMFun f* (*replicate n GMHole*)) *ss* = *GFun f ss*
unfolding *partition-holes-fill-gholes-conv*′
by (*induct ss arbitrary*: *n*) *auto*

lemma *fill-gholes-gmctxt-replicate-MHole* [*simp*]:
fill-gholes-gmctxt C (*replicate* (*num-gholes C*) *GMHole*) = *C*
proof (*induct C*)
case (*GMFun f Cs*)
{fix *i* **assume** *i* < *length Cs*
then have *partition-gholes* (*replicate* (*sum-list* (*map num-gholes Cs*)) *GMHole*) *Cs* ! *i* =
replicate (*num-gholes* (*Cs* ! *i*)) *GMHole*
using *partition-by-nth-nth*[*of map num-gholes Cs replicate* (*sum-list* (*map num-gholes Cs*)) *MHole*]
by (*auto simp*: *length-partition-by-nth partition-by-nth-nth intro*!: *nth-equalityI*)**}**
note ∗ = *this*
show *?case* **using** *GMFun*[*OF nth-mem*] **by** (*auto simp*: ∗ *intro*!: *nth-equalityI*)
qed *auto*

lemma *fill-gholes-gmctxt-GMFun-replicate-length* [*simp*]:
fill-gholes-gmctxt (*GMFun f* (*replicate* (*length Cs*) *GMHole*)) *Cs* = *GMFun f Cs*
unfolding *partition-holes-fill-gholes-gmctxt-conv*′
by (*induct Cs*) *simp-all*

lemma *fill-gholes-gmctxt-MFun*:
assumes *lCs*: *length Cs* = *length ts*
and *lss*: *length ss* = *length ts*
and *rec*: ⋀ *i*. *i* < *length ts* ⟹ *num-gholes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧
fill-gholes-gmctxt (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *fill-gholes-gmctxt* (*GMFun f Cs*) (*concat ss*) = *GMFun f ts*
using *assms* **unfolding** *fill-gholes-gmctxt.simps gmctxt.simps*
by (*auto intro*!: *nth-equalityI*)
(*metis length-map map-nth-eq-conv partition-by-concat-id*)

lemma *fill-gholes-gmctxt-nHole* [*simp*]:
C ≠ *GMHole* ⟹ *num-gholes C* = *length Ds* ⟹ *fill-gholes-gmctxt C Ds* ≠ *GMHole*
using *fill-gholes-gmctxt.elims* **by** *blast*

lemma *num-gholes-fill-gholes-gmctxt* [*simp*]:
assumes *num-gholes C* = *length Ds*
shows *num-gholes* (*fill-gholes-gmctxt C Ds*) = *sum-list* (*map num-gholes Ds*)
using *assms*

proof (*induct C arbitrary*: *Ds*)
case *GMHole* **then show** *?case*
by (*cases Ds*) *simp-all*
next
case (*GMFun f Cs*)
then have ∗: *map* (*num-gholes* ∘ (λ*i*. *fill-gholes-gmctxt* (*Cs* ! *i*) (*partition-gholes Ds Cs* ! *i*))) [*0*..<*length Cs*] =
map (λ*i*. *sum-list* (*map num-gholes* (*partition-gholes Ds Cs* ! *i*))) [*0* ..< *length Cs*]
and *sum-list* (*map num-gholes Cs*) = *length Ds*
by (*auto simp add*: *length-partition-by-nth*)
then show *?case*
using *map-upt-len-conv* [*of* λ*x*. *sum-list* (*map num-gholes x*) *partition-gholes Ds Cs*]
unfolding *partition-holes-fill-holes-mctxt-conv* **by** (*simp add*: ∗)
qed

lemma *num-gholes-greater0-fill-gholes-gmctxt* [*intro*!]:
assumes *num-gholes C* = *length Ds*
and ∃ *D* ∈ *set Ds*. *0* < *num-gholes D*
shows *0* < *sum-list* (*map num-gholes Ds*)
using *assms gr-zeroI* **by** *force*

lemma *fill-gholes-gmctxt-fill-gholes*:
assumes *len-ds*: *length Ds* = *num-gholes C*
and *nh*: *num-gholes* (*fill-gholes-gmctxt C Ds*) = *length ss*
shows *fill-gholes* (*fill-gholes-gmctxt C Ds*) *ss* =
fill-gholes C [*fill-gholes* (*Ds* ! *i*) (*partition-gholes ss Ds* ! *i*). *i* ← [*0* ..< *num-gholes C*]]
using *assms*(*1*)[*symmetric*] *assms*(*2*)
proof (*induct C Ds arbitrary*: *ss rule*: *fill-gholes-induct*)
case (*GMFun f Cs ds ss*)
define *qs* **where** *qs* = *map* (λ*i*. *fill-gholes-gmctxt* (*Cs* ! *i*) (*partition-gholes ds Cs* ! *i*)) [*0*..<*length Cs*]
then have *qs*: ⋀*i*. *i* < *length Cs* ⟹ *fill-gholes-gmctxt* (*Cs* ! *i*) (*partition-gholes ds Cs* ! *i*) = *qs* ! *i*
length qs = *length Cs* **by** *auto*
define *zs* **where** *zs* = *map* (λ*i*. *fill-gholes* (*ds* ! *i*) (*partition-gholes ss ds* ! *i*)) [*0*..<*length ds*]
{fix *i* **assume** *i*: *i* < *length Cs*
from *GMFun*(*1*) **have** ∗: *map length* (*partition-gholes ds Cs*) = *map num-gholes Cs* **by** *auto*
have ∗∗: *length ss* = *sum-list* (*map sum-list* (*partition-gholes* (*map num-gholes ds*) *Cs*))
using *GMFun*(*1*) *GMFun*(*3*)[*symmetric*] *num-gholes-fill-gholes-gmctxt*[*of GM-Fun f Cs ds*]
by (*auto simp*: *comp-def map-map-partition-by*[*symmetric*])
have *partition-by* (*partition-by ss*
(*map* (λ*i*. *num-gholes* (*fill-gholes-gmctxt* (*Cs* ! *i*) (*partition-gholes ds Cs* !

```
i))) [0..<length Cs]) ! i)
      (partition-gholes (map num-gholes ds) Cs ! i) = partition-gholes (partition-gholes
ss ds) Cs ! i
      using i GMFun(1) GMFun(3) partition-by-partition-by[OF **]
      by (auto simp: comp-def num-gholes-fill-gholes-gmctxt length-partition-by-nth
        intro!: arg-cong[of - - λx. partition-by (partition-by ss x ! -) -] nth-equalityI)
    then have map (λj. fill-gholes (partition-gholes ds Cs ! i ! j)
        (partition-gholes (partition-gholes ss qs ! i)
        (partition-gholes ds Cs ! i) ! j)) [0..<num-gholes (Cs ! i)] =
        partition-gholes zs Cs ! i using GMFun(1,3)
     by (auto simp: length-partition-by-nth zs-def qs-def i comp-def partition-by-nth-nth
intro: nth-equalityI)}
  then show ?case using GMFun
    by (simp add: qs-def [symmetric] qs zs-def [symmetric] length-partition-by-nth)
qed auto

lemma fill-gholes-gmctxt-sound:
  assumes len-ds: length Ds = num-gholes C
  and len-sss: length sss = num-gholes C
  and len-ts: length ts = num-gholes C
  and insts: ⋀ i. i < length Ds ⟹ ts ! i =Gf (Ds ! i, sss ! i)
  shows fill-gholes C ts =Gf (fill-gholes-gmctxt C Ds, concat sss)
proof (rule eqfI)
  note l-nh-i = eqgfE(2)[OF insts]
  from len-ds len-sss have concat-sss: partition-gholes (concat sss) Ds = sss
    by (metis l-nh-i length-map map-nth-eq-conv partition-by-concat-id)
  then show nh: num-gholes (fill-gholes-gmctxt C Ds) = length (concat sss)
    unfolding num-gholes-fill-gholes-gmctxt [OF len-ds [symmetric]] length-concat
    by (metis l-nh-i len-ds len-sss nth-map-conv)
  have ts: ts = [fill-gholes (Ds ! i) (partition-gholes (concat sss) Ds ! i) . i ←
[0..<num-gholes C]] (is - = ?fhs)
  proof (rule nth-equalityI)
    show l-fhs: length ts = length ?fhs unfolding length-map
      by (metis diff-zero len-ts length-upt)
    fix i
    assume i: i < length ts
    then have i': i < length [0..<num-gholes C]
      by (metis diff-zero len-ts length-upt)
    show ts!i = ?fhs ! i
      unfolding nth-map[OF i']
      using eqgfE(1)[OF insts[unfolded len-ds, OF i[unfolded len-ts]]]
      by (metis concat-sss i' len-ds len-sss map-nth nth-map)
  qed
  note ts = this
  show fill-gholes C ts = fill-gholes (fill-gholes-gmctxt C Ds) (concat sss)
    unfolding fill-gholes-gmctxt-fill-gholes[OF len-ds nh] ts ..
qed
```

2.2.6 Semilattice Structures

lemma *inf-gmctxt-idem* [*simp*]:
(*C* :: *'f gmctxt*) ⊓ *C* = *C*
by (*induct C*) (*auto simp*: *zip-same-conv-map intro*: *map-idI*)

lemma *inf-gmctxt-GMHole2* [*simp*]:
C ⊓ *GMHole* = *GMHole*
by (*induct C*) *simp-all*

lemma *inf-gmctxt-comm* [*ac-simps*]:
(*C* :: *'f gmctxt*) ⊓ *D* = *D* ⊓ *C*
by (*induct C D rule*: *inf-gmctxt.induct*) (*fastforce simp*: *in-set-conv-nth intro*!: *nth-equalityI*)+

lemma *inf-gmctxt-assoc* [*ac-simps*]:
fixes *C* :: *'f gmctxt*
shows *C* ⊓ *D* ⊓ *E* = *C* ⊓ (*D* ⊓ *E*)
apply (*induct C D arbitrary*: *E rule*: *inf-gmctxt.induct*)
apply (*auto*)
apply (*case-tac E*, *auto*)+
apply (*fastforce simp*: *in-set-conv-nth intro*!: *nth-equalityI*)
apply (*case-tac E*, *auto*)+
done

instantiation *gmctxt* :: (*type*) *order*
begin

definition (*C* :: *'a gmctxt*) ≤ *D* ⟷ *C* ⊓ *D* = *C*
definition (*C* :: *'a gmctxt*) < *D* ⟷ *C* ≤ *D* ∧ ¬ *D* ≤ *C*

instance
by (*standard*, *simp-all add*: *less-eq-gmctxt-def less-gmctxt-def ac-simps*, *metis inf-gmctxt-assoc*)

end

lemma *less-eq-gmctxt-prime*: *C* ≤ *D* ⟷ *less-eq-gmctxt C D*
proof
assume *less-eq-gmctxt C D* **then show** *C* ≤ *D*
by (*induct C D rule*: *less-eq-gmctxt.induct*) (*auto simp*: *less-eq-gmctxt-def intro*: *nth-equalityI*)
next
assume *C* ≤ *D* **then show** *less-eq-gmctxt C D* **unfolding** *less-eq-gmctxt-def*
by (*induct C D rule*: *inf-gmctxt.induct*)
(*auto split*: *if-splits simp*: *set-zip intro*!: *less-eq-gmctxt.intros nth-equalityI elim*!: *nth-equalityE*, *metis*)
qed

lemmas *less-eq-gmctxt-induct* = *less-eq-gmctxt.induct*[*folded less-eq-gmctxt-prime*,

consumes 1]
lemmas *less-eq-gmctxt-intros = less-eq-gmctxt.intros*[*folded less-eq-gmctxt-prime*]

lemma *less-eq-gmctxt-Hole*:
less-eq-gmctxt C GMHole ⟹ C = GMHole
using *less-eq-gmctxt.cases* **by** *blast*

lemma *num-gholes-at-least1*:
0 < num-gholes C ⟹ 0 < num-gholes (C ⊓ D)
proof (*induct C arbitrary*: *D*)
case (*GMFun f Cs*)
from *GMFun*(*2*) **obtain** *i* **where** *wit*: *i < length Cs 0 < num-gholes* (*Cs* ! *i*)
by (*auto*, *metis* (*mono-tags*, *lifting*) *in-set-conv-nth length-map map-nth-eq-conv not-less sum-list-nonpos*)
note *IS = GMFun*(*1*)[*OF nth-mem*, *OF wit*]
show *?case*
proof (*cases D*)
case [*simp*]: (*GMFun g Ds*)
{assume *f = g length Cs = length Ds*
then have *0 < num-gholes* (*Cs* ! *i* ⊓ *Ds* ! *i*) **using** *IS*[*of Ds* ! *i*]
by *auto***}**
then show *?thesis* **using** *wit*(*1*)
by (*auto simp*: *split*!: *prod.splits*)
(*smt* (*verit*, *del-insts*) *length-map length-zip map-nth-eq-conv min-less-iff-conj not-gr0 nth-mem nth-zip o-apply prod.simps*(*2*) *sum-list-eq-0-iff*)
qed *auto*
qed *auto*

(⊔) is defined on compatible multihole contexts. Note that compatibility is not transitive.

instance *gmctxt* :: (*type*) *semilattice-inf*
apply (*standard*)
apply (*auto simp*: *less-eq-gmctxt-def inf-gmctxt-assoc* [*symmetric*])
apply (*metis inf-gmctxt-comm inf-gmctxt-assoc inf-gmctxt-idem*)+
done

lemma *sup-gmctxt-idem* [*simp*]:
fixes *C* :: *'f gmctxt*
shows *C ⊔ C = C*
by (*induct C*) (*auto simp*: *zip-same-conv-map intro*: *map-idI*)

lemma *sup-gmctxt-MHole* [*simp*]: *C ⊔ GMHole = C*
by (*induct C*) *simp-all*

lemma *sup-gmctxt-comm* [*ac-simps*]:
fixes *C* :: *'f gmctxt*
shows *C ⊔ D = D ⊔ C*
by (*induct C D rule*: *sup-gmctxt.induct*) (*fastforce simp*: *in-set-conv-nth intro*!:

nth-equalityI)+

lemma *comp-gmctxt-refl*:
(*C*, *C*) ∈ *comp-gmctxt*
by (*induct C*) *auto*

lemma *comp-gmctxt-sym*:
assumes (*C*, *D*) ∈ *comp-gmctxt*
shows (*D*, *C*) ∈ *comp-gmctxt*
using *assms* **by** (*induct*) *auto*

lemma *sup-gmctxt-assoc* [*ac-simps*]:
assumes (*C*, *D*) ∈ *comp-gmctxt* **and** (*D*, *E*) ∈ *comp-gmctxt*
shows *C* ⊔ *D* ⊔ *E* = *C* ⊔ (*D* ⊔ *E*)
using *assms* **by** (*induct C D arbitrary*: *E*) (*auto elim*!: *comp-gmctxt.cases intro*!: *nth-equalityI*)

No instantiation to *semilattice-sup* possible, since (⊔) is only partially defined on terms (e.g., it is not associative in general).

interpretation *gmctxt-order-bot*: *order-bot GMHole* (≤) (<)
by (*standard*) (*simp add*: *less-eq-gmctxt-def*)

lemma *sup-gmctxt-ge1* [*simp*]:
assumes (*C*, *D*) ∈ *comp-gmctxt*
shows *C* ≤ *C* ⊔ *D*
using *assms* **by** (*induct C D*) (*auto simp*: *less-eq-gmctxt-def intro*: *nth-equalityI*)

lemma *sup-gmctxt-ge2* [*simp*]:
assumes (*C*, *D*) ∈ *comp-gmctxt*
shows *D* ≤ *C* ⊔ *D*
using *assms* **by** (*induct*) (*auto simp*: *less-eq-gmctxt-def intro*: *nth-equalityI*)

lemma *sup-gmctxt-least*:
assumes (*D*, *E*) ∈ *comp-gmctxt*
and *D* ≤ *C* **and** *E* ≤ *C*
shows *D* ⊔ *E* ≤ *C*
using *assms*
apply (*induct arbitrary*: *C*)
apply (*auto simp*: *less-eq-gmctxt-def elim*!: *inf-gmctxt.elims intro*!: *nth-equalityI*)
apply (*metis* (*erased*, *lifting*) *length-map nth-map nth-zip split-conv*)
apply (*case-tac fb* = *gb* ∧ *length Csb* = *length Dsb*, *simp-all*)+
done

lemma *sup-gmctxt-args-MHole2* [*simp*]:
sup-gmctxt-args C GMHole = *replicate* (*num-gholes C*) *GMHole*
by (*cases C*) *simp-all*

lemma *num-gholes-sup-gmctxt-args*:

assumes $(C, D) \in$ *comp-gmctxt*
shows *num-gholes* $C =$ *length* (*sup-gmctxt-args* C D)
using *assms* **by** (*induct*) (*auto simp*: *length-concat intro*!: *arg-cong* [*of* - - *sum-list*] *nth-equalityI*)

lemma *sup-gmctxt-sup-gmctxt-args*:
assumes $(C, D) \in$ *comp-gmctxt*
shows *fill-gholes-gmctxt* C (*sup-gmctxt-args* C D) $= C \sqcup D$ **using** *assms*
proof (*induct*)
note *fill-gholes-gmctxt.simps* [*simp del*]
case (*GMFun f g Cs Ds*)
then show *?case*
proof (*cases* $f = g \wedge$ *length Cs* $=$ *length Ds*)
case *True*
with *GMFun* **have** $\forall i <$ *length Cs*.
fill-gholes-gmctxt (*Cs* ! *i*) (*sup-gmctxt-args* (*Cs* ! *i*) (*Ds* ! *i*)) = *Cs* ! *i* $\sqcup$ *Ds* ! *i*
and $*$: $\forall i <$ *length Cs*. (*Cs* ! *i*, *Ds* ! *i*) $\in$ *comp-gmctxt* **by** (*force simp*: *set-zip*)+
moreover have *partition-gholes* (*concat* (*map* (*case-prod sup-gmctxt-args*) (*zip Cs Ds*)))
Cs = *map* (*case-prod sup-gmctxt-args*) (*zip Cs Ds*)
using *True* **and** $*$ **by** (*intro partition-by-concat-id*) (*auto simp*: *num-gholes-sup-gmctxt-args*)
ultimately show *?thesis*
using $*$ **and** *True* **by** (*auto simp*: *partition-holes-fill-gholes-gmctxt-conv intro*!: *nth-equalityI*)
qed *auto*
qed *auto*

lemma *eqgf-comp-gmctxt*:
assumes $s =_{Gf}$ (C, ss) **and** $s =_{Gf}$ (D, ts)
shows $(C, D) \in$ *comp-gmctxt* **using** *assms*
proof (*induct s arbitrary*: *C D ss ts*)
case (*GFun f ss C D us vs*)
{ fix *Cs* **and** *Ds*
assume $*$: *C* $=$ *GMFun f Cs D* $=$ *GMFun f Ds* **and** $**$: *length Cs* $=$ *length Ds*
have *?case*
proof (*unfold* $*$, *intro comp-gmctxt.GMFun* [*OF refl* $**$] *allI impI*)
fix *i*
assume *i* $<$ *length Ds* **then show** (*Cs* ! *i*, *Ds* ! *i*) $\in$ *comp-gmctxt*
using *GFun* **by** (*auto simp*: $*$ $**$ *elim*!: *eqgf-GMFunE*) (*metis nth-mem*)
qed}
from *GFun.prems this* **show** *?case*
by (*cases C D rule*: *gmctxt.exhaust* [*case-product gmctxt.exhaust*])
(*auto simp*: *eq-gfill.simps dest*: *map-eq-imp-length-eq*)
qed

lemma *eqgf-less-eq* [*simp*]:
assumes $s =_{Gf}$ (C, ss)
shows $C \leq$ *gmctxt-of-gterm s* **using** *assms*
by (*induct rule*: *eq-gfill-induct*) (*auto simp*: *less-eq-gmctxt-prime*)

lemma *less-eq-comp-gmctxt* [*simp*]:
  $C \leq D \Longrightarrow (C,\ D) \in$ *comp-gmctxt*
  by (*induct rule*: *less-eq-gmctxt-induct*) *auto*

lemma *gmctxt-less-eq-sup*:
  $(C :: 'f\ gmctxt) \leq D \Longrightarrow C \sqcup D = D$
  by (*induct rule*: *less-eq-gmctxt-induct*) (*auto intro*: *nth-equalityI*)

lemma *fill-gholes-gmctxt-less-eq*:
  assumes *num-gholes C* = *length Ds*
  shows *C* ≤ *fill-gholes-gmctxt C Ds* **using** *assms*
proof (*induct C arbitrary*: *Ds*)
  case (*GMFun f Cs*)
  show *?case* **using** *GMFun*(*1*)[*OF nth-mem*] *GMFun*(*2*)
    unfolding *partition-holes-fill-gholes-gmctxt-conv'*
    by (*intro less-eq-gmctxt-intros*) (*auto simp*: *length-partition-by-nth*)
qed *simp*

lemma *less-eq-to-sup-mctxt-args* [*elim*]:
  assumes *C* ≤ *D*
  obtains *Ds* **where** *num-gholes C* = *length Ds D* = *fill-gholes-gmctxt C Ds*
  using *assms gmctxt-less-eq-sup*[*OF assms*]
  using *sup-gmctxt-sup-gmctxt-args*[*OF less-eq-comp-gmctxt*[*OF assms*]]
  using *less-eq-comp-gmctxt num-gholes-sup-gmctxt-args*
  by *force*

lemma *fill-gholes-gmctxt-sup-mctxt-args* [*simp*]:
  assumes *num-gholes C* = *length Ds*
  shows *sup-gmctxt-args C* (*fill-gholes-gmctxt C Ds*) = *Ds* **using** *assms*
proof (*induct C arbitrary*: *Ds*)
  case *GMHole* **then show** *?case*
    by (*cases Ds*) *auto*
next
  case (*GMFun f Cs*)
  have *map2 sup-gmctxt-args Cs* (*map2 fill-gholes-gmctxt Cs* (*partition-gholes Ds Cs*)) = *partition-gholes Ds Cs*
    using *GMFun*(*1*)[*OF nth-mem*] *GMFun*(*2*)
    by (*auto simp*: *length-partition-by-nth intro*!: *nth-equalityI*)
  then show *?case* **using** *GMFun*(*1*)[*OF nth-mem*] *GMFun*(*2*)
    unfolding *partition-holes-fill-gholes-gmctxt-conv'*
    using *concat-partition-by*[*of map num-gholes Cs Ds*]
    by *auto*
qed

lemma *map2-fill-gholes-gmctxt-id* [*simp*]:
  assumes ⋀ *i*. *i* < *length Ds* ⟹ *num-gholes* (*Ds* ! *i*) = *0*
  shows *map2 fill-gholes-gmctxt Ds* (*replicate* (*length Ds*) []) = *Ds*

using *assms fill-gholes-gmctxt-no-holes*[*of Ds* ! *i* **for** *i*]
by (*auto simp*: *map-nth-eq-conv*)

lemma *fill-gholes-gmctxt-GMFun-replicate-append* [*simp*]:
assumes *length Cs* = *n* **and** ⋀ *t*. *t* ∈ *set Ds* ⟹ *num-gholes t* = *0*
shows *fill-gholes-gmctxt* (*GMFun f* ((*replicate n GMHole*) @ *Ds*)) *Cs* = *GMFun f* (*Cs* @ *Ds*) **using** *assms*
proof (*induct n arbitrary*: *Cs*)
case *0* **note** [*simp*] = *0*(*1*)
have *i* < *length Ds* ⟹ *num-gholes* (*Ds* ! *i*) = *0* **for** *i* **using** *0* **by** *fastforce*
then show *?case* **using** *0* **unfolding** *partition-holes-fill-gholes-gmctxt-conv'*
by (*cases Cs*) *auto*
next
case (*Suc n*) **then show** *?case* **unfolding** *partition-holes-fill-gholes-gmctxt-conv'*
by (*simp add*: *Cons-nth-drop-Suc take-Suc-conv-app-nth*)
qed

lemma *finite-ghole-poss*:
finite (*ghole-poss C*)
by (*induct C*) *auto*

lemma *ghole-poss-simp* [*simp*]:
ghole-poss (*GMFun f cs*) = {*i* # *p* | *i p*. *i* < *length cs* ∧ *p* ∈ *ghole-poss* (*cs* ! *i*)}
by *auto*
declare *ghole-poss.simps*(*2*)[*simp del*]

lemma *num-gholes-zero-ghole-poss*:
num-gholes D = *0* ⟹ *ghole-poss D* = {}
by (*induct D*) *auto*

lemma *ghole-poss-num-gholes-zero*:
ghole-poss D = {} ⟹ *num-gholes D* = *0*
proof (*induct D*)
case (*GMFun f Ds*)
then show *?case*
by (*simp*, *metis Collect-empty-eq Collect-mem-eq in-set-idx*)
qed *simp*

lemma *num-ghloes-nzero-ghole-poss-nempty*:
num-gholes D ≠ *0* ⟹ *ghole-poss D* ≠ {}
by (*induct D*) (*auto simp*: *in-set-conv-nth*, *fastforce*)

lemma *ghole-poss-epsE* [*elim*]:
ghole-poss D = {[]} ⟹ *D* = *GMHole*
by (*induct D*) *auto*

lemma *ghole-poss-gmctxt-of-gterm* [*simp*]:
ghole-poss (*gmctxt-of-gterm t*) = {}
by (*induct t*) *auto*

lemma *ghole-poss-subseteq-args* [*simp*]:
assumes *ghole-poss* (*GMFun f Ds*) ⊆ *ghole-poss* (*GMFun g Cs*)
shows ∀ *i* < *min* (*length Ds*) (*length Cs*). *ghole-poss* (*Ds* ! *i*) ⊆ *ghole-poss* (*Cs* ! *i*) **using** *assms*
by *auto*

lemma *factor-ghole-pos-by-prefix*:
assumes *C* ≤ *D p* ∈ *ghole-poss D*
obtains *q* **where** *q* ≤$_p$ *p q* ∈ *ghole-poss C*
using *assms*
by (*induct C D arbitrary*: *p thesis rule*: *less-eq-gmctxt-induct*)
(*auto*, *metis position-less-eq-Cons*)

lemma *prefix-and-fewer-gholes-implies-equal-gmctxt*:
C ≤ *D* ⟹ *ghole-poss C* ⊆ *ghole-poss D* ⟹ *C* = *D*
proof (*induct C D rule*: *less-eq-gmctxt-induct*)
case (*1 D*) **then show** *?case* **by** (*cases D*) *auto*
next
case (*2 Cs Ds f*)
have *i* < *length Ds* ⟹ *ghole-poss* (*Cs* ! *i*) ⊆ *ghole-poss* (*Ds* ! *i*) **for** *i* **using** *2*(*1*,*4*) **by** *auto*
then show *?case* **using** *2* **by** (*auto intro*!: *nth-equalityI*)
qed

lemma *set-sup-gmctxt-args-split*:
length Cs = *length Ds* ⟹ *set* (*sup-gmctxt-args* (*GMFun f Cs*) (*GMFun f Ds*)) =
(⋃ *i* ∈ {*0*..< *length Ds*}. *set* (*sup-gmctxt-args* (*Cs* ! *i*) (*Ds* ! *i*)))
by (*auto simp*: *atLeast0LessThan in-set-zip*)
(*metis length-map map-fst-zip nth-mem nth-zip*)

lemma *gmctxt-closing-trans*:
gmctxt-closing C D ⟹ *gmctxt-closing D E* ⟹ *gmctxt-closing C E*
unfolding *gmctxt-closing-def* **using** *less-eq-gmctxt-prime*
by (*metis* (*full-types*) *order-trans*)

lemma *gmctxt-closing-sup-args-ghole-or-gterm*:
assumes *gmctxt-closing C D*
shows ∀ *E* ∈ *set* (*sup-gmctxt-args C D*). *E* = *GMHole* ∨ *num-gholes E* = *0*
using *assms* **unfolding** *gmctxt-closing-def*
proof −
from *assms* **have** *C* ≤ *D ghole-poss D* ⊆ *ghole-poss C* **unfolding** *gmctxt-closing-def*
by (*auto simp*: *less-eq-gmctxt-prime*)
then show *?thesis*
proof (*induct rule*: *less-eq-gmctxt-induct*)
case (*1 D*)
then show *?case*
by (*cases D*) (*auto simp*: *in-set-conv-nth intro*!: *ghole-poss-num-gholes-zero*, *blast*)

next
case (*2 cs ds f*) **note** *IS* = *this*
show *?case* **using** *IS set-sup-gmctxt-args-split*[*OF IS*(*1*)]
by *auto*
qed
qed

lemma *inv-imples-ghole-poss-subseteq*:
$C \leq D \Longrightarrow \forall\ E \in$ *set* (*sup-gmctxt-args C D*). $E =$ *GMHole* $\vee$ *num-gholes* $E =$ *0* $\Longrightarrow$ *ghole-poss D* $\subseteq$ *ghole-poss C*
proof (*induct rule*: *less-eq-gmctxt-induct*)
case (*1 D*) **then show** *?case*
by (*cases D*) (*auto simp*: *num-gholes-zero-ghole-poss*)
next
case (*2 cs ds f*)
then show *?case* **using** *set-sup-gmctxt-args-split*[*OF 2*(*1*)]
by *auto* (*metis* (*no-types*, *lifting*) *fst-conv in-set-zip snd-conv subsetD*)
qed

lemma *fill-gholes-gmctxt-ghole-poss-subseteq*:
assumes *num-gholes C* = *length Ds* $\forall$ *i* < *length Ds*. *Ds* ! *i* = *GMHole* $\vee$ *num-gholes* (*Ds* ! *i*) = *0*
shows *ghole-poss* (*fill-gholes-gmctxt C Ds*) $\subseteq$ *ghole-poss C* **using** *assms*
proof (*induct rule*: *fill-gholes-induct*)
case (*GMFun f Cs xs*)
then show *?case* **unfolding** *partition-holes-fill-gholes-gmctxt-conv'*
by *auto* (*metis* (*no-types*, *lifting*) *length-map length-partition-by-nth partition-by-nth-nth*(*1*, *2*) *subsetD*)
qed (*auto simp*: *num-gholes-zero-ghole-poss*)

lemma *ghole-poss-not-in-poss-gmctxt*:
assumes *p* $\in$ *ghole-poss C*
shows *p* $\notin$ *poss-gmctxt C* **using** *assms*
by (*induct C arbitrary*: *p*) *auto*

lemma *comp-gmctxt-inf-ghole-poss-cases*:
assumes (*C*, *D*) $\in$ *comp-gmctxt p* $\in$ *ghole-poss* (*C* $\sqcap$ *D*)
shows *p* $\in$ *ghole-poss C* $\wedge$ *p* $\in$ *ghole-poss D* $\vee$
p $\in$ *ghole-poss C* $\wedge$ *p* $\in$ *poss-gmctxt D* $\vee$
p $\in$ *ghole-poss D* $\wedge$ *p* $\in$ *poss-gmctxt C* **using** *assms*
proof (*induct arbitrary*: *p*)
case (*GMHole1 D*) **then show** *?case*
by (*cases D*) *auto*
next
case (*GMHole2 C*) **then show** *?case*
by (*cases C*) *auto*
next
case (*GMFun f g Cs Ds*)
then show *?case*

```
    by (auto simp: atLeast0LessThan) blast+
qed

lemma length-ghole-poss-list-num-gholes:
  num-gholes C = length (ghole-poss-list C)
  by (induct C) (auto simp: length-concat intro: nth-sum-listI)

lemma ghole-poss-list-distict:
  distinct (ghole-poss-list C)
proof (induct C)
  case (GMFun f Cs)
  then show ?case proof (induct Cs rule: rev-induct)
    case (snoc x xs)
     then have distinct (ghole-poss-list (GMFun f xs)) distinct (ghole-poss-list x)
by auto
   then show ?case using snoc by (auto simp add: map-cons-presv-distinct dest:
set-zip-leftD)
  qed auto
qed auto

lemma ghole-poss-ghole-poss-list-conv:
  ghole-poss C = set (ghole-poss-list C)
proof (induct C)
  case (GMFun f Cs) note IS = GMFun[OF nth-mem]
  {fix p assume p ∈ ghole-poss (GMFun f Cs)
    then obtain i ps where w: p = i # ps i < length Cs
      ps ∈ ghole-poss (Cs ! i) by auto
    then have (i, Cs ! i) ∈ set (zip [0..<length Cs] Cs)
      by (force simp: in-set-zip)
    then have p ∈ set (ghole-poss-list (GMFun f Cs)) using IS[of i] w
      by auto}
  then show ?case using IS
    by (auto simp: in-set-zip)
qed auto

lemma card-ghole-poss-num-gholes:
  card (ghole-poss C) = num-gholes C
  unfolding ghole-poss-ghole-poss-list-conv
  unfolding length-ghole-poss-list-num-gholes
  using ghole-poss-list-distict
  using distinct-card by blast

lemma subgm-at-hole-poss [simp]:
  p ∈ ghole-poss C ⟹ subgm-at C p = GMHole
  by (induct C arbitrary: p) auto

lemma subgm-at-mctxt-of-term:
  p ∈ gposs t ⟹ subgm-at (gmctxt-of-gterm t) p = gmctxt-of-gterm (gsubt-at t p)
  by (induct t arbitrary: p) auto
```

lemma *num-gholes-subgm-at*:
assumes $p \in$ *poss-gmctxt* C
shows *num-gholes* (*subgm-at* C p) = *ghole-num-at-pos* p C **using** *assms*
by (*induct* C *arbitrary*: p) *auto*

lemma *gmctxt-subtgm-at-fill-args-empty-pos* [*simp*]:
assumes *num-gholes* C = *length ts*
shows *gmctxt-subtgm-at-fill-args* [] C *ts* = *ts*
using *assms* **by** (*auto simp*: *gmctxt-subtgm-at-fill-args-def*)

lemma *ghole-num-bef-at-pos-num-gholes-less-eq*:
assumes $p \in$ *poss-gmctxt* C
shows *ghole-num-bef-pos* p C + *ghole-num-at-pos* p $C \leq$ *num-gholes* C **using** *assms*
proof (*induct* C *arbitrary*: p)
case (*GMFun f Cs*)
show *?case*
proof (*cases p*)
case (*Cons i ps*)
have ∗: *num-gholes* (*GMFun f Cs*) = *sum-list* (*map num-gholes* (*take i Cs*)) + *num-gholes* (*Cs* ! *i*) + *sum-list* (*map num-gholes* (*drop* (*Suc i*) *Cs*))
using *GMFun*(*2*) **unfolding** *Cons*
by (*auto simp flip*: *take-map take-drop*)
(*metis Cons-nth-drop-Suc add.assoc append-take-drop-id drop-map length-map nth-map sum-list.Cons sum-list.append*)
from *Cons* **have**
(*sum-list* (*map num-gholes* (*take i Cs*)) + (*ghole-num-bef-pos ps* (*Cs* ! *i*) + *ghole-num-at-pos ps* (*Cs* ! *i*)))
+ *sum-list* (*map num-gholes* (*drop* (*Suc i*) *Cs*)) ≤
(*sum-list* (*map num-gholes* (*take i Cs*)) + *num-gholes* (*Cs* ! *i*)) + *sum-list* (*map num-gholes* (*drop* (*Suc i*) *Cs*))
using *GMFun*(*1*)[*OF nth-mem*, *of i ps*] *GMFun*(*2*)
by *auto*
from *add-le-imp-le-right*[*OF this*] **show** *?thesis* **using** *GMFun*(*2*) ∗
unfolding *Cons*
by *simp*
qed *auto*
qed *auto*

lemma *ghole-num-at-pos-fill-args-length*:
assumes $p \in$ *poss-gmctxt* C *num-gholes* C = *length ts*
shows *ghole-num-at-pos* p C = *length* (*gmctxt-subtgm-at-fill-args* p C *ts*)
using *ghole-num-bef-at-pos-num-gholes-less-eq*[*OF assms*(*1*)] *assms*(*2*)
by (*auto simp*: *gmctxt-subtgm-at-fill-args-def*)

lemma *ghole-poss-nth-subt-at*:
assumes $t =_{Gf}$ (C, *ts*) **and** $p \in$ *ghole-poss* C
shows *ghole-num-bef-pos* p $C <$ *length ts* ∧ *gsubt-at* t p = *ts* ! *ghole-num-bef-pos*

p C **using** *assms*
proof (*induct arbitrary*: *p rule*: *eq-gfill-induct*)
case (*GMFun f ss Cs ts*)
let *?ts* = *partition-gholes ts Cs*
from *GMFun* **obtain** *i* **and** *q* **where** [*simp*]: *p* = *i* # *q*
and *i* < *length ss* **and** *q* ∈ *ghole-poss* (*Cs* ! *i*) **by** *auto*
with *GMFun.hyps* **have** *ss* ! *i* =$_{Gf}$ (*Cs* ! *i*, *?ts* ! *i*)
and *j*: *ghole-num-bef-pos q* (*Cs* ! *i*) < *length* (*?ts* ! *i*) (**is** *?j* < *length* -)
and ∗: *?ts* ! *i* ! *ghole-num-bef-pos q* (*Cs* ! *i*) = *gsubt-at* (*ss* ! *i*) *q*
by *auto*
let *?k* = *sum-list* (*map length* (*take i ?ts*)) + *?j*
have *i* < *length ?ts* **using** ‹*i* < *length ss*› **and** *GMFun* **by** *auto*
with *partition-by-nth-nth-old* [*OF this j*] **and** *GMFun* **and** *concat-nth-length* [*OF this j*]
have *?ts* ! *i* ! *?j* = *ts* ! *?k* **and** *?k* < *length ts* **by** (*auto*)
moreover with ∗ **have** *ts* ! *?k* = *gsubt-at* (*GFun f ss*) *p* **using** ‹*i* < *length ss*› **by** *simp*
ultimately show *?case* **using** *GMFun.hyps*(*2*) **by** (*auto simp*: *take-map* [*symmetric*])
qed *auto*

lemma *poss-gmctxt-fill-gholes-split*:
assumes *t* =$_{Gf}$ (*C*, *ts*) **and** *p* ∈ *poss-gmctxt C*
shows *gsubt-at t p* =$_{Gf}$ (*subgm-at C p* , *gmctxt-subtgm-at-fill-args p C ts*)
using *assms*
proof (*induct arbitrary*: *p rule*: *eq-gfill-induct*)
case (*GMFun f ss Cs ts*)
let *?ts* = *partition-gholes ts Cs*
from *GMFun* **have** ⋀ *i*. *i* < *length Cs* ⟹ *ss* ! *i* =$_{Gf}$ (*Cs* ! *i*, *?ts* ! *i*) **by** *auto*
show *?case*
proof (*cases p*)
case *Nil*
then have *GFun f ss* =$_{Gf}$ (*GMFun f Cs*, *concat ?ts*) **using** *GMFun*
by (*intro eqgf-GMFunI*) (*auto simp del*: *fill-gholes.simps*)
then show *?thesis* **using** *GMFun* **unfolding** *Nil*
by *simp*
next
case (*Cons i q*)
then have *ghole-num-at-pos q* (*Cs* ! *i*) ≤ *num-gholes* (*Cs* ! *i*) − *ghole-num-bef-pos q* (*Cs* ! *i*)
using *GMFun*(*1*, *2*, *4*) *ghole-num-bef-at-pos-num-gholes-less-eq*[*of q Cs* ! *i*]
by *auto*
then show *?thesis* **using** *GMFun*
by (*auto simp*: *Cons add.commute gmctxt-subtgm-at-fill-args-def partition-by-nth drop-take take-map min-def split*!: *if-splits*)
qed
qed *auto*

lemma *fill-gholes-ghole-poss*:
assumes *t* =$_{Gf}$ (*C*, *ts*) **and** *i* < *length ts*

shows *gsubt-at t* (*ghole-poss-list C* ! *i*) = *ts* ! *i* **using** *assms*
proof (*induct arbitrary*: *i rule*: *eq-gfill-induct*)
case (*GMFun f ss Cs ts*)
have ∗: *length* (*concat* (*poss-rec ghole-poss-list Cs*)) = *num-gholes* (*GMFun f Cs*)
using *GMFun*(*1*, *2*, *4*)
unfolding *length-ghole-poss-list-num-gholes*[*of GMFun f Cs*, *symmetric*, *unfolded ghole-poss-list.simps*]
by *auto*
from *GMFun* **have** *b*: *i* < *length* (*concat* (*partition-gholes ts Cs*)) **by** *simp*
then have *ts*: *ts* ! *i* = (λ (*j*, *k*). *partition-gholes ts Cs* ! *j* ! *k*) (*concat-index-split* (*0*, *i*) (*partition-gholes ts Cs*))
by (*metis GMFun.hyps*(*2*) *concat-index-split-sound concat-partition-by*)
obtain *o-idx i-idx* **where** *csp*: *concat-index-split* (*0*, *i*) (*partition-gholes ts Cs*) = (*o-idx*, *i-idx*)
using *old.prod.exhaust* **by** *blast*
have *idx*: *o-idx* < *length Cs i-idx* < *length* (*partition-gholes ts Cs* ! *o-idx*)
using *concat-index-split-sound-bounds*[*OF b csp*] **by** *auto*
have *concat-index-split* (*0*, *i*) (*poss-rec ghole-poss-list Cs*) = (*o-idx*, *i-idx*)
using *GMFun*(*1*, *2*, *4*) ∗ **unfolding** *csp*[*symmetric*]
by (*intro concat-index-split-unique*, *unfold* ∗)
(*auto*, *simp add*: *length-ghole-poss-list-num-gholes length-partition-gholes-nth*)
then have *gp*: *ghole-poss-list* (*GMFun f Cs*) ! *i* = *poss-rec ghole-poss-list Cs* ! *o-idx* ! *i-idx*
by (*simp add*: ∗ *GMFun.hyps*(*2*) *GMFun.prems concat-index-split-less-length-concat*(*4*))
from *idx GMFun* **have** *r*: *o-idx* < *length* (*zip* [*0*..<*length ss*] *Cs*) **by** *auto*
then show *?case* **using** *GMFun idx* **unfolding** *ts csp gp*
by (*auto simp*: *nth-map*[*OF r*] *length-ghole-poss-list-num-gholes length-partition-gholes-nth split*: *prod.splits*)
qed *auto*

lemma *length-unfill-gholes* [*simp*]:
assumes *C* ≤ *gmctxt-of-gterm t*
shows *length* (*unfill-gholes C t*) = *num-gholes C*
using *assms*
proof (*induct C t rule*: *unfill-gholes.induct*)
case (*2 f Cs g ts*) **with** *2*(*1*)[*OF - nth-mem*] *2*(*2*) **show** *?case*
by (*auto simp*: *less-eq-gmctxt-def length-concat*
intro!: *cong*[*of sum-list*, *OF refl*] *nth-equalityI elim*!: *nth-equalityE*)
qed *auto*

lemma *fill-gholes-arbitrary*:
assumes *lCs*: *length Cs* = *length ts*
and *lss*: *length ss* = *length ts*
and *rec*: ⋀ *i*. *i* < *length ts* ⟹ *num-gholes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧ *f* (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *map* (λ*i*. *f* (*Cs* ! *i*) (*partition-gholes* (*concat ss*) *Cs* ! *i*)) [*0* ..< *length Cs*] = *ts*
proof −
have *sum-list* (*map num-gholes Cs*) = *length* (*concat ss*) **using** *assms*

by (*auto simp*: *length-concat map-nth-eq-conv intro*: *arg-cong*[*of* - - *sum-list*])
moreover have *partition-gholes* (*concat ss*) *Cs* = *ss*
using *assms* **by** (*auto intro*: *partition-by-concat-id*)
ultimately show *?thesis* **using** *assms* **by** (*auto intro*: *nth-equalityI*)
qed

lemma *fill-unfill-gholes*:
assumes *C* ≤ *gmctxt-of-gterm t*
shows *fill-gholes C* (*unfill-gholes C t*) = *t*
using *assms*
proof (*induct C t rule*: *unfill-gholes.induct*)
case (*2 f Cs g ts*) **with** *2*(*1*)[*OF* - *nth-mem*] *2*(*2*) **show** *?case*
by (*auto simp*: *less-eq-gmctxt-def unfill-gholes-conv intro*!: *fill-gholes-arbitrary elim*!: *nth-equalityE*)
qed (*auto split*: *if-splits*)

lemma *funas-gmctxt-of-mctxt* [*simp*]:
ground-mctxt C ⟹ *funas-gmctxt* (*gmctxt-of-mctxt C*) = *funas-mctxt C*
by (*induct C*) (*auto simp*: *funas-gterm-gterm-of-term*)

lemma *funas-mctxt-of-gmctxt-conv*:
funas-mctxt (*mctxt-of-gmctxt C*) = *funas-gmctxt C*
by (*induct C*) (*auto simp flip*: *funas-gterm-gterm-of-term*)

lemma *funas-gterm-ctxt-apply* [*simp*]:
assumes *num-gholes C* = *length ss*
shows *funas-gterm* (*fill-gholes C ss*) = *funas-gmctxt C* ∪ ⋃ (*set* (*map funas-gterm ss*)) **using** *assms*
proof (*induct rule*: *fill-gholes-induct*)
case (*GMFun f Cs ss*)
show *?case* **using** *GMFun partition-gholes-subseteq*[*OF GMFun*(*1*)]
by (*auto simp add*: *Un-Union-image simp del*: *map-partition-by-nth*)
qed *simp*

lemma *funas-gmctxt-gmctxt-of-gterm* [*simp*]:
funas-gmctxt (*gmctxt-of-gterm s*) = *funas-gterm s*
by (*induct s*) *auto*

lemma *funas-gmctxt-replicate-GMHole* [*simp*]:
funas-gmctxt (*GMFun f* (*replicate n GMHole*)) = {(*f*, *n*)}
by *auto*

lemma *funas-gmctxt-gmctxt-of-gctxt* [*simp*]:
funas-gmctxt (*gmctxt-of-gctxt C*) = *funas-gctxt C*
by (*induct C*) *auto*

lemma *funas-gmctxt-fill-gholes-gmctxt* [*simp*]:
assumes *num-gholes C* = *length Ds*
shows *funas-gmctxt* (*fill-gholes-gmctxt C Ds*) = *funas-gmctxt C* ∪ ⋃(*set* (*map*

funas-gmctxt Ds))
(**is** *?f C Ds* = *?g C Ds*) **using** *assms*
proof (*induct C arbitrary*: *Ds*)
case *GMHole* **then show** *?case* **by** (*cases Ds*) *simp-all*
next
case (*GMFun f Cs*)
then have *num-gholes*: *sum-list* (*map num-gholes Cs*) = *length Ds* **by** *simp*
let *?ys* = *partition-gholes Ds Cs*
have ⋀*i*. *i* < *length Cs* ⟹ *?f* (*Cs* ! *i*) (*?ys* ! *i*) = *?g* (*Cs* ! *i*) (*?ys* ! *i*)
by (*simp add*: *GMFun.hyps length-partition-by-nth num-gholes*)
then have (⋃*i* ∈ {*0* ..< *length Cs*}. *?f* (*Cs* ! *i*) (*?ys* ! *i*)) =
(⋃*i* ∈ {*0* ..< *length Cs*}. *?g* (*Cs* ! *i*) (*?ys* ! *i*)) **by** *simp*
then show *?case*
using *num-gholes* **unfolding** *partition-holes-fill-holes-mctxt-conv*
by (*simp add*: *UN-Un-distrib UN-upt-len-conv* [*of* - - λ*x*. ⋃(*set x*)] *UN-set-partition-by-map*)
qed

lemma *funas-supremum*:
C ≤ *D* ⟹ *funas-gmctxt D* = *funas-gmctxt C* ∪ ⋃ (*set* (*map funas-gmctxt* (*sup-gmctxt-args C D*)))
using *fill-gholes-gmctxt-sup-mctxt-args*[*of C*]
by (*auto simp*: *fill-gholes-gmctxt-sup-mctxt-args*[*of C*] *elim*!: *less-eq-to-sup-mctxt-args*[*of C D*])

lemma *funas-gctxt-gctxt-of-gmctxt* [*simp*]:
num-gholes D = *Suc 0* ⟹ *funas-gctxt* (*gctxt-of-gmctxt D*) = *funas-gmctxt D*
by (*metis One-nat-def funas-gmctxt-gmctxt-of-gctxt gmctxt-of-gctxt-gctxt-of-gmctxt*)

lemma *funas-gterm-gterm-of-gmctxt* [*simp*]:
num-gholes C = *0* ⟹ *funas-gterm* (*gterm-of-gmctxt C*) = *funas-gmctxt C*
by (*metis funas-gmctxt-gmctxt-of-gterm no-gholes-gmctxt-of-gterm-gterm-of-gmctxt-id*)

lemma *less-sup-gmctxt-args-funas-gmctxt*:
C ≤ *D* ⟹ *funas-gmctxt C* ⊆ *F* ⟹ ∀ *Ds* ∈ *set* (*sup-gmctxt-args C D*). *funas-gmctxt Ds* ⊆ *F* ⟹ *funas-gmctxt D* ⊆ *F*
using *funas-supremum*[*of C D*] **by** *auto*

lemma *funas-gmctxt-poss-gmctxt-subgm-at-funas*:
assumes *funas-gmctxt C* ⊆ *F* *p* ∈ *poss-gmctxt C*
shows *funas-gmctxt* (*subgm-at C p*) ⊆ *F*
using *assms*
proof (*induct C arbitrary*: *p*)
case (*GMFun f Cs*)
then show *?case*
by (*auto*, *blast*) (*metis SUP-le-iff nth-mem subsetD*)
qed *auto*

lemma *inf-funas-gmctxt-subset1*:
funas-gmctxt (*C* ⊓ *D*) ⊆ *funas-gmctxt C*

using *funas-supremum*[*of C C* ⊓ *D*]
by (*metis funas-supremum inf-le1 le-sup-iff order-refl*)

lemma *inf-funas-gmctxt-subset2*:
funas-gmctxt (*C* ⊓ *D*) ⊆ *funas-gmctxt D*
using *funas-supremum*[*of D C* ⊓ *D*]
by (*metis funas-supremum inf-le2 le-sup-iff order-refl*)

end
theory *Bot-Terms*
imports *Utils*
begin

2.3 Bottom terms

datatype *'f bot-term* = *Bot* | *BFun 'f* (*args*: *'f bot-term list*)

fun *term-to-bot-term* :: (*'f*, *'v*) *term* ⇒ *'f bot-term* (‹-⊥› [*80*] *80*) **where**
(*Var* -)⊥ = *Bot*
| (*Fun f ts*)⊥ = *BFun f* (*map term-to-bot-term ts*)

fun *root-bot* **where**
root-bot Bot = *None* |
root-bot (*BFun f ts*) = *Some* (*f*, *length ts*)

fun *funas-bot-term* **where**
funas-bot-term Bot = {}
| *funas-bot-term* (*BFun f ss*) = {(*f*, *length ss*)} ∪ (⋃ (*funas-bot-term* ‘ *set ss*))

lemma *finite-funas-bot-term*:
finite (*funas-bot-term t*)
by (*induct t*) *auto*

lemma *funas-bot-term-funas-term*:
funas-bot-term (*t*⊥) = *funas-term t*
by (*induct t*) *auto*

lemma *term-to-bot-term-root-bot* [*simp*]:
root-bot (*t*⊥) = *root t*
by (*induct t*) *auto*

lemma *term-to-bot-term-root-bot-comp* [*simp*]:
root-bot ∘ *term-to-bot-term* = *root*
using *term-to-bot-term-root-bot* **by** *force*

inductive-set *mergeP* **where**
base-l [*simp*]: (*Bot*, *t*) ∈ *mergeP*
| *base-r* [*simp*]: (*t*, *Bot*) ∈ *mergeP*

| *step* [*intro*]: *length ss* = *length ts* ⟹ (∀ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) ∈ *mergeP*)
⟹
(*BFun f ss*, *BFun f ts*) ∈ *mergeP*

lemma *merge-refl*:
(*s*, *s*) ∈ *mergeP*
by (*induct s*) *auto*

lemma *merge-symmetric*:
assumes (*s*, *t*) ∈ *mergeP*
shows (*t*, *s*) ∈ *mergeP*
using *assms* **by** *induct auto*

fun *merge-terms* :: *'f bot-term* ⇒ *'f bot-term* ⇒ *'f bot-term* (**infixr** ‹↑› *67*) **where**
Bot ↑ *s* = *s*
| *s* ↑ *Bot* = *s*
| (*BFun f ss*) ↑ (*BFun g ts*) = (*if f* = *g* ∧ *length ss* = *length ts*
then BFun f (*map* (*case-prod* (↑)) (*zip ss ts*))
else undefined)

lemma *merge-terms-bot-rhs*[*simp*]:
s ↑ *Bot* = *s* **by** (*cases s*) *auto*

lemma *merge-terms-idem*: *s* ↑ *s* = *s*
by (*induct s*) (*auto simp add*: *map-nth-eq-conv*)

lemma *merge-terms-assoc* [*ac-simps*]:
assumes (*s*, *t*) ∈ *mergeP* **and** (*t*, *u*) ∈ *mergeP*
shows (*s* ↑ *t*) ↑ *u* = *s* ↑ *t* ↑ *u*
using *assms* **by** (*induct s t arbitrary*: *u*) (*auto elim*!: *mergeP.cases intro*!: *nth-equalityI*)

lemma *merge-terms-commutative* [*ac-simps*]:
shows *s* ↑ *t* = *t* ↑ *s*
by (*induct s t rule*: *merge-terms.induct*)
(*fastforce simp*: *in-set-conv-nth intro*!: *nth-equalityI*)+

lemma *merge-dist*:
assumes (*s*, *t* ↑ *u*) ∈ *mergeP* **and** (*t*, *u*) ∈ *mergeP*
shows (*s*, *t*) ∈ *mergeP* **using** *assms*
by (*induct t arbitrary*: *s u*) (*auto elim*!: *mergeP.cases*, *metis mergeP.step nth-mem*)

lemma *megeP-ass*:
(*s*, *t* ↑ *u*) ∈ *mergeP* ⟹ (*t*, *u*) ∈ *mergeP* ⟹ (*s* ↑ *t*, *u*) ∈ *mergeP*
by (*induct t arbitrary*: *s u*) (*auto simp*: *mergeP.step elim*!: *mergeP.cases*)

inductive-set *bless-eq* **where**
base-l [*simp*]: (*Bot*, *t*) ∈ *bless-eq*
| *step* [*intro*]: *length ss* = *length ts* ⟹ (∀ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) ∈ *bless-eq*)
⟹

(*BFun f ss*, *BFun f ts*) ∈ *bless-eq*

Infix syntax.

abbreviation *bless-eq-pred s t* ≡ (*s*, *t*) ∈ *bless-eq*
notation
bless-eq (‹{≤$_b$}›) **and**
bless-eq-pred (‹(-/ ≤$_b$ -)› [*56*, *56*] *55*)

lemma *BFun-leq-Bot-False* [*simp*]:
BFun f ts ≤$_b$ *Bot* ⟷ *False*
using *bless-eq.cases* **by** *auto*

lemma *BFun-lesseqE* [*elim*]:
assumes *BFun f ts* ≤$_b$ *t*
obtains *us* **where** *length ts* = *length us t* = *BFun f us*
using *assms bless-eq.cases* **by** *blast*

lemma *bless-eq-refl*: *s* ≤$_b$ *s*
by (*induct s*) *auto*

lemma *bless-eq-trans* [*trans*]:
assumes *s* ≤$_b$ *t* **and** *t* ≤$_b$ *u*
shows *s* ≤$_b$ *u* **using** *assms*
proof (*induct arbitrary*: *u*)
case (*step ss ts f*)
from *step*(*3*) **obtain** *us* **where** [*simp*]: *u* = *BFun f us length ts* = *length us* **by** *auto*
from *step*(*3*, *1*, *2*) **have** *i* < *length ss* ⟹ *ss* ! *i* ≤$_b$ *us* ! *i* **for** *i*
by (*cases rule*: *bless-eq.cases*) *auto*
then show *?case* **using** *step*(*1*) **by** *auto*
qed *auto*

lemma *bless-eq-anti-sym*:
s ≤$_b$ *t* ⟹ *t* ≤$_b$ *s* ⟹ *s* = *t*
by (*induct rule*: *bless-eq.induct*) (*auto elim*!: *bless-eq.cases intro*: *nth-equalityI*)

lemma *bless-eq-mergeP*:
s ≤$_b$ *t* ⟹ (*s*, *t*) ∈ *mergeP*
by (*induct s arbitrary*: *t*) (*auto elim*!: *bless-eq.cases*)

lemma *merge-bot-args-bless-eq-merge*:
assumes (*s*, *t*) ∈ *mergeP*
shows *s* ≤$_b$ *s* ↑ *t* **using** *assms*
by (*induct s arbitrary*: *t*) (*auto simp*: *bless-eq-refl elim*!: *mergeP.cases intro*!: *step*)

lemma *bless-eq-closued-under-merge*:
assumes (*s*, *t*) ∈ *mergeP* (*u*, *v*) ∈ *mergeP s* ≤$_b$ *u t* ≤$_b$ *v*
shows *s* ↑ *t* ≤$_b$ *u* ↑ *v* **using** *assms*(*3*, *4*, *1*, *2*)
proof (*induct arbitrary*: *t v*)

```
  case (base-l t)
  then show ?case using bless-eq-trans merge-bot-args-bless-eq-merge
    by (metis merge-symmetric merge-terms.simps(1) merge-terms-commutative)
next
  case (step ss ts f)
  then show ?case apply (auto elim!: mergeP.cases intro!: bless-eq.step)
    using bless-eq-trans merge-bot-args-bless-eq-merge apply blast
    by (metis bless-eq.cases bot-term.distinct(1) bot-term.sel(2))
qed

lemma bless-eq-closued-under-supremum:
  assumes s ≤b u t ≤b u
  shows s ↑ t ≤b u using assms
  by (induct arbitrary: t) (auto elim!: bless-eq.cases intro!: bless-eq.step)

lemma linear-term-comb-subst:
  assumes linear-term (Fun f ss)
    and length ss = length ts
    and ⋀ i. i < length ts ⟹ ss ! i · σ i = ts ! i
  shows ∃ σ. Fun f ss · σ = Fun f ts
  using assms subst-merge[of ss σ]
  apply auto apply (rule-tac x = σ' in exI)
  apply (intro nth-equalityI) apply auto
  by (metis term-subst-eq)

lemma bless-eq-to-instance:
  assumes s⊥ ≤b t⊥ and linear-term s
  shows ∃ σ. s · σ = t using assms
proof (induct s arbitrary: t)
  case (Fun f ts)
  from Fun(2) obtain us where [simp]: t = Fun f us length ts = length us by
(cases t) auto
  have i < length ts ⟹ ∃σ. ts ! i · σ = us ! i for i
    using Fun(2, 3) Fun(1)[OF nth-mem, of i us ! i for i]
    by (auto elim: bless-eq.cases)
  then show ?case using Ex-list-of-length-P[of length ts λ σ i. ts ! i · σ = us ! i]
    using linear-term-comb-subst[OF Fun(3)] by auto
qed auto

lemma instance-to-bless-eq:
  assumes s · σ = t
  shows s⊥ ≤b t⊥ using assms
proof (induct s arbitrary: t)
  case (Fun f ts) then show ?case
    by (cases t) auto
qed auto

end
theory Saturation
```

imports *Main*
begin

2.4 Set operation closure for idempotent, associative, and commutative functions

lemma *inv-to-set*:
(∀ *i* < *length ss*. *ss* ! *i* ∈ *S*) ⟷ *set ss* ⊆ *S*
by (*induct ss*) (*auto simp*: *nth-Cons split*: *nat.splits*)

lemma *ac-comp-fun-commute*:
assumes ⋀ *x y*. *f x y* = *f y x* **and** ⋀ *x y z*. *f x* (*f y z*) = *f* (*f x y*) *z*
shows *comp-fun-commute f* **using** *assms* **unfolding** *comp-fun-commute-def*
by (*auto simp*: *comp-def*) *fastforce*

lemma (**in** *comp-fun-commute*) *fold-list-swap*:
fold f xs (*fold f ys y*) = *fold f ys* (*fold f xs y*)
by (*metis comp-fun-commute fold-commute fold-commute-apply*)

lemma (**in** *comp-fun-commute*) *foldr-list-swap*:
foldr f xs (*foldr f ys y*) = *foldr f ys* (*foldr f xs y*)
by (*simp add*: *fold-list-swap foldr-conv-fold*)

lemma (**in** *comp-fun-commute*) *foldr-to-fold*:
foldr f xs = *fold f xs*
using *comp-fun-commute foldr-fold*[*of - f*]
by (*auto simp*: *comp-def*)

lemma (**in** *comp-fun-commute*) *fold-commute-f*:
f x (*foldr f xs y*) = *foldr f xs* (*f x y*)
using *comp-fun-commute* **unfolding** *foldr-to-fold*
by (*auto simp*: *comp-def intro*: *fold-commute-apply*)

lemma *closure-sound*:
assumes *cl*: ⋀ *s t*. *s* ∈ *S* ⟹ *t* ∈ *S* ⟹ *f s t* ∈ *S*
and *com*: ⋀ *x y*. *f x y* = *f y x* **and** *ass*: ⋀ *x y z*. *f x* (*f y z*) = *f* (*f x y*) *z*
and *fin*: *set ss* ⊆ *S ss* ≠ []
shows *fold f* (*tl ss*) (*hd ss*) ∈ *S* **using** *assms*(*4* −)
proof (*induct ss*)
case (*Cons s ss*) **note** *IS* = *this* **show** *?case*
proof (*cases ss*)
case *Nil*
then show *?thesis* **using** *IS* **by** *auto*
next
case (*Cons t ts*) **show** *?thesis*
using *IS assms*(*1*) *ac-comp-fun-commute*[*of f*, *OF com ass*] **unfolding** *Cons*
by (*auto simp flip*: *comp-fun-commute.foldr-to-fold*) (*metis com comp-fun-commute.fold-commute-f*)
qed
qed *auto*

locale *set-closure-oprator* =
  fixes *f*
  assumes *com* [*ac-simps*]: $\bigwedge x\ y.\ f\ x\ y = f\ y\ x$
    and *ass* [*ac-simps*]: $\bigwedge x\ y\ z.\ f\ x\ (f\ y\ z) = f\ (f\ x\ y)\ z$
    and *idem*: $\bigwedge x.\ f\ x\ x = x$

sublocale *set-closure-oprator* $\subseteq$ *comp-fun-idem*
  using *set-closure-oprator-axioms ac-comp-fun-commute*
 by (*auto simp*: *comp-fun-idem-def comp-fun-idem-axioms-def comp-def set-closure-oprator-def*)

context *set-closure-oprator*
begin

inductive-set *closure* **for** *S* **where**
  base [*simp*]: $s \in S \Longrightarrow s \in closure\ S$
| *step* [*intro*]: $s \in closure\ S \Longrightarrow t \in closure\ S \Longrightarrow f\ s\ t \in closure\ S$

lemma *closure-idem* [*simp*]:
  closure (*closure S*) = *closure S* (**is** *?LS* = *?RS*)
proof −
  {fix *s* **assume** $s \in$ *?LS* **then have** $s \in$ *?RS* **by** *induct auto***}**
  moreover
  {fix *s* **assume** $s \in$ *?RS* **then have** $s \in$ *?LS* **by** *induct auto***}**
  ultimately show *?thesis* **by** *blast*
qed

lemma *fold-dist*:
  assumes $xs \neq []$
  shows *f* (*fold f* (*tl xs*) (*hd xs*)) *t* = *fold f xs t* **using** *assms*
proof (*induct xs*)
  case (*Cons a xs*)
  show *?case* **using** *Cons com ass fold-commute-f*
    by (*auto simp*: *comp-def foldr-to-fold*)
qed *auto*

lemma *closure-to-cons-list*:
  assumes $s \in closure\ S$
  shows $\exists\ ss \neq [].\ fold\ f\ (tl\ ss)\ (hd\ ss) = s \wedge (\forall\ i < length\ ss.\ ss\ !\ i \in S)$ **using**
assms
proof (*induct*)
  case (*base s*) **then show** *?case* **by** (*auto intro*: *exI*[*of* - [*s*]])
next
  case (*step s t*)
  then obtain *ss ts* **where**
    s: *fold f* (*tl ss*) (*hd ss*) = *s* **and** *inv-s*: $ss \neq [] \forall\ i < length\ ss.\ ss\ !\ i \in S$ **and**
    t: *fold f* (*tl ts*) (*hd ts*) = *t* **and** *inv-t*: $ts \neq [] \forall\ i < length\ ts.\ ts\ !\ i \in S$
    by *auto*

then show *?case*
by (*auto simp*: *fold-dist nth-append intro*!: *exI*[*of - ss @ ts*]) (*metis com fold-dist*)
qed

lemma *sound-fold*:
assumes *set ss* ⊆ *closure S* **and** *ss* ≠ []
shows *fold f* (*tl ss*) (*hd ss*) ∈ *closure S* **using** *assms*
using *closure-sound*[*of closure S f*] *assms step*
by (*auto simp add*: *com fun-left-comm*)

lemma *closure-empty* [*simp*]: *closure* {} = {}
using *closure-to-cons-list* **by** *auto*

lemma *closure-mono*:
S ⊆ *T* ⟹ *closure S* ⊆ *closure T*
proof
fix *s* **assume** *ass*: *s* ∈ *closure S*
then show *S* ⊆ *T* ⟹ *s* ∈ *closure T*
by (*induct*) (*auto simp*: *closure-to-cons-list*)
qed

lemma *closure-insert*:
closure (*insert x S*) = {*x*} ∪ *closure S* ∪ {*f x s* | *s*. *s* ∈ *closure S*}
proof −
{fix *t* **assume** *ass*: *t* ∈ *closure* (*insert x S*) *t* ≠ *x t* ∉ *closure S*
from *closure-to-cons-list*[*OF ass*(*1*)] **obtain** *ss* **where**
t: *fold f* (*tl ss*) (*hd ss*) = *t* **and** *inv-t*: *ss* ≠ [] ∀ *i* < *length ss*. *ss* ! *i* ∈ *insert x S*
by *auto*
then have *mem*: *x* ∈ *set ss* **using** *ass*(*3*) *sound-fold*[*of ss*] *in-set-conv-nth*
by (*force simp add*: *inv-to-set*)
have ∃ *s*. *t* = *f x s* ∧ *s* ∈ *closure S*
proof (*cases set ss* = {*x*})
case *True* **then show** *?thesis* **using** *ass*(*2*) *t*
by (*metis com finite.emptyI fold-dist fold-empty fold-insert-idem fold-set-fold idem inv-t*(*1*))
next
case *False*
from *False inv-t*(*1*) *mem* **obtain** *ts* **where** *split*: *insert x* (*set ts*) = *set ss x* ∉ *set ts ts* ≠ []
by *auto* (*metis False List.finite-set Set.set-insert empty-set finite-insert finite-list*)
then have ∀ *i* < *length ts*. *ts* ! *i* ∈ *S* **using** *inv-t*(*2*) *split* **unfolding** *inv-to-set* **by** *auto*
moreover have *t* = *f x* (*Finite-Set.fold f* (*hd ts*) (*set* (*tl ts*)))
using *split t inv-t*(*1*)
by (*metis List.finite-set com fold-dist fold-insert-idem2 fold-set-fold fun-left-idem idem*)
ultimately show *?thesis* **using** *sound-fold*[*OF - split*(*3*)]

by (*auto simp*: *foldr-to-fold fold-set-fold inv-to-set*) *force*
qed}
then show *?thesis*
by (*auto simp*: *fold-set-fold in-mono*[*OF closure-mono*[*OF subset-insertI*[*of S x*]]])
qed

lemma *finite-S-finite-closure* [*intro*]:
finite S $\Longrightarrow$ *finite* (*closure S*)
by (*induct rule*: *finite.induct*) (*auto simp*: *closure-insert*)

end

locale *semilattice-closure-operator* =
cl: *set-closure-oprator f* **for** *f* :: *'a* $\Rightarrow$ *'a* $\Rightarrow$ *'a* +
fixes *less-eq e*
assumes *neut-fun* [*simp*]:$\bigwedge$ *x*. *f e x* = *x*
and *neut-less* [*simp*]: $\bigwedge$ *x*. *less-eq e x*
and *sup-l*: $\bigwedge$ *x y*. *less-eq x* (*f x y*)
and *sup-r*: $\bigwedge$ *x y*. *less-eq y* (*f x y*)
and *upper-bound*: $\bigwedge$ *x y z*. *less-eq x z* $\Longrightarrow$ *less-eq y z* $\Longrightarrow$ *less-eq* (*f x y*) *z*
and *trans*: $\bigwedge$ *x y z*. *less-eq x y* $\Longrightarrow$ *less-eq y z* $\Longrightarrow$ *less-eq x z*
and *anti-sym*: $\bigwedge$ *x y*. *less-eq x y* $\Longrightarrow$ *less-eq y x* $\Longrightarrow$ *x* = *y*
begin

lemma *unique-neut-elem* [*simp*]:
f x y = *e* $\longleftrightarrow$ *x* = *e* $\wedge$ *y* = *e*
using *neut-fun cl.fun-left-idem*
by (*metis cl.com*)

abbreviation *closure S* $\equiv$ *cl.closure S*

lemma *closure-to-cons-listE*:
assumes *s* $\in$ *closure S*
obtains *ss* **where** *ss* $\neq$ [] *fold f ss e* = *s set ss* $\subseteq$ *S*
using *cl.closure-to-cons-list*[*OF assms*] *cl.fold-dist*
by (*auto simp*: *inv-to-set*) (*metis cl.com neut-fun*)

lemma *sound-fold*:
assumes *set ss* $\subseteq$ *closure S ss* $\neq$ []
shows *fold f ss e* $\in$ *closure S*
using *cl.sound-fold*[*OF assms*] *cl.fold-dist*[*OF assms*(*2*)]
by (*metis cl.com neut-fun*)

abbreviation *supremum S* $\equiv$ *Finite-Set.fold f e S*
definition *smaller-subset x S* $\equiv$ {*y*. *less-eq y x* $\wedge$ *y* $\in$ *S*}

lemma *smaller-subset-empty* [*simp*]:

smaller-subset x {} = {}
by (*auto simp*: *smaller-subset-def*)

lemma *finite-smaller-subset* [*simp*, *intro*]:
finite S ⟹ finite (*smaller-subset x S*)
by (*auto simp*: *smaller-subset-def*)

lemma *smaller-subset-mono*:
smaller-subset x S ⊆ S
by (*auto simp*: *smaller-subset-def*)

lemma *sound-set-fold*:
assumes *set ss ⊆ closure S* **and** *ss ≠ []*
shows *supremum* (*set ss*) *∈ closure S*
using *sound-fold*[*OF assms*]
by (*auto simp*: *cl.fold-set-fold*)

lemma *supremum-neutral* [*simp*]:
assumes *finite S* **and** *supremum S = e*
shows *S ⊆ {e}* **using** *assms*
by (*induct*) *auto*

lemma *supremum-in-closure*:
assumes *finite S* **and** *R ⊆ closure S* **and** *R ≠ {}*
shows *supremum R ∈ closure S*
proof −
obtain *L* **where** [*simp*]: *R = set L*
using *cl.finite-S-finite-closure*[*OF assms*(*1*)] *assms*(*2*) *finite-list*
by (*metis infinite-super*)
then show *?thesis* **using** *sound-set-fold*[*of L S*] *assms*
by (*cases L*) *auto*
qed

lemma *supremum-sound*:
assumes *finite S*
shows ⋀ *t*. *t ∈ S ⟹ less-eq t* (*supremum S*)
using *assms sup-l sup-r trans*
by *induct* (*auto*, *blast*)

lemma *supremum-sound-list*:
∀ *i < length ss*. *less-eq* (*ss* ! *i*) (*fold f ss e*)
unfolding *cl.fold-set-fold*[*symmetric*]
using *supremum-sound*[*of set ss*]
by *auto*

lemma *smaller-subset-insert* [*simp*]:
less-eq y x ⟹ smaller-subset x (*insert y S*) = *insert y* (*smaller-subset x S*)
¬ less-eq y x ⟹ smaller-subset x (*insert y S*) = *smaller-subset x S*
by (*auto simp*: *smaller-subset-def*)

lemma *supremum-smaller-subset*:
assumes *finite S*
shows *less-eq* (*supremum* (*smaller-subset x S*)) *x* **using** *assms*
proof (*induct*)
case (*insert y F*) **then show** *?case*
by (*cases less-eq y x*) (*auto simp*: *upper-bound*)
qed *simp*

lemma *pre-subset-eq-pos-subset* [*simp*]:
shows *smaller-subset x* (*closure S*) = *closure* (*smaller-subset x S*) (**is** *?LS* = *?RS*)
proof −
{fix *s* **assume** *s* ∈ *?RS* **then have** *s* ∈ *?LS*
using *upper-bound* **by** *induct* (*auto simp*: *smaller-subset-def*)**}**
moreover
{fix *s* **assume** *ass*: *s* ∈ *?LS*
then have *s* ∈ *closure S* **using** *smaller-subset-mono* **by** *auto*
then obtain *ss* **where** *wit*: *ss* ≠ [] ∧ *fold f ss e* = *s* ∧ (*set ss* ⊆ *S*)
using *closure-to-cons-listE* **by** *blast*
then have ∀ *i* < *length ss*. *less-eq* (*ss* ! *i*) *x*
using *supremum-sound*[*of set ss*] *supremum-smaller-subset*[*of set ss x*]
unfolding *cl.fold-set-fold*[*symmetric*]
by *auto* (*metis ass local.trans mem-Collect-eq nth-mem smaller-subset-def*)
then have *s* ∈ *?RS* **using** *wit sound-fold*[*of ss*]
by (*auto simp*: *smaller-subset-def*)
(*metis* (*mono-tags*, *lifting*) *cl.closure.base inv-to-set mem-Collect-eq*)**}**
ultimately show *?thesis* **by** *blast*
qed

lemma *supremum-in-smaller-closure*:
assumes *finite S*
shows *supremum* (*smaller-subset x S*) ∈ {*e*} ∪ (*closure S*)
using *supremum-in-closure*[*OF assms*, *of smaller-subset x S*]
by (*metis UnI1 UnI2 cl.closure.base fold-empty singletonI smaller-subset-mono subset-iff*)

lemma *supremum-subset-less-eq*:
assumes *finite S* **and** *R* ⊆ *S*
shows *less-eq* (*supremum R*) (*supremum S*) **using** *assms*
proof (*induct arbitrary*: *R*)
case (*insert x F*)
from *insert*(*1*, *2*, *4*) *insert*(*3*)[*of R* − {*x*}]
have *less-eq* (*supremum* (*R* − {*x*})) (*f x* (*supremum F*))
by (*metis Diff-subset-conv insert-is-Un local.trans sup-r*)
then show *?case* **using** *insert*(*1*, *2*, *4*)
by *auto* (*metis Diff-empty Diff-insert0 cl.fold-rec finite.insertI finite-subset sup-l*

upper-bound)
qed (*auto*)

lemma *supremum-smaller-closure* [*simp*]:
  assumes *finite S*
  shows *supremum* (*smaller-subset x* (*closure S*)) = *supremum* (*smaller-subset x S*)
proof (*cases smaller-subset x S* = {})
  case [*simp*]: *True* **show** *?thesis* **by** *auto*
next
  case *False*
  have *smaller-subset x S* ⊆ *smaller-subset x* (*closure S*)
    unfolding *pre-subset-eq-pos-subset* **by** *auto*
  then have *l*: *less-eq* (*supremum* (*smaller-subset x S*)) (*supremum* (*smaller-subset x* (*closure S*)))
    using *assms* **unfolding** *pre-subset-eq-pos-subset*
    by (*intro supremum-subset-less-eq*) *auto*
  from *False* **have** *supremum* (*closure* (*smaller-subset x S*)) ∈ *closure S*
    using *assms cl.closure-mono*[*OF smaller-subset-mono*]
    using ‹*smaller-subset x S* ⊆ *smaller-subset x* (*closure S*)›
    by (*auto simp add*: *assms intro*!: *supremum-in-closure*)
  from *closure-to-cons-listE*[*OF this*] **obtain** *ss* **where**
    dec : *supremum* (*smaller-subset x* (*closure S*)) = *Finite-Set.fold f e* (*set ss*)
    and *inv*: *ss* ≠ [] *set ss* ⊆ *S*
    by (*auto simp*: *cl.fold-set-fold*) *force*
  then have *set ss* ⊆ *smaller-subset x S*
    using *supremum-sound*[*OF assms*]
    using *supremum-smaller-subset*[*OF assms*]
    by (*auto simp*: *smaller-subset-def*)
    (*metis List.finite-set assms cl.finite-S-finite-closure dec trans supremum-smaller-subset supremum-sound*)
  then have *less-eq* (*supremum* (*smaller-subset x* (*closure S*))) (*supremum* (*smaller-subset x S*))
    using *inv assms* **unfolding** *dec*
    by (*intro supremum-subset-less-eq*) *auto*
  then show *?thesis* **using** *l anti-sym*
    by *auto*
qed

end

fun *lift-f-total* **where**
  lift-f-total P f None - = None
| *lift-f-total P f - None = None*
| *lift-f-total P f* (*Some s*) (*Some t*) = (*if P s t then Some* (*f s t*) *else None*)

fun *lift-less-eq-total* **where**
  lift-less-eq-total f - None = True

```
| lift-less-eq-total f None - = False
| lift-less-eq-total f (Some s) (Some t) = (f s t)


locale set-closure-partial-oprator =
  fixes P f
  assumes refl: ⋀ x. P x x
    and sym: ⋀ x y. P x y ⟹ P y x
    and dist: ⋀ x y z. P y z ⟹ P x (f y z) ⟹ P x y
    and assP: ⋀ x y z. P x (f y z) ⟹ P y z ⟹ P (f x y) z
    and com [ac-simps]: ⋀ x y. P x y ⟹ f x y = f y x
    and ass [ac-simps]: ⋀ x y z. P x y ⟹ P y z ⟹ f x (f y z) = f (f x y) z
    and idem: ⋀ x. f x x = x
begin

lemma lift-f-total-com:
  lift-f-total P f x y = lift-f-total P f y x
  using com by (cases x; cases y) (auto simp: sym)

lemma lift-f-total-ass:
  lift-f-total P f x (lift-f-total P f y z) = lift-f-total P f (lift-f-total P f x y) z
proof (cases x)
  case [simp]: (Some s) show ?thesis
  proof (cases y)
    case [simp]: (Some t) show ?thesis
    proof (cases z)
      case [simp]: (Some u) show ?thesis
        by (auto simp add: ass dist[of t u s])
          (metis com dist assP sym)+
    qed auto
  qed auto
qed auto

lemma lift-f-total-idem:
  lift-f-total P f x x = x
  by (cases x) (auto simp: idem refl)

lemma lift-f-totalE[elim]:
  assumes lift-f-total P f s u = Some t
  obtains v w where s = Some v u = Some w
  using assms by (cases s; cases u) auto

lemma lift-set-closure-oprator:
  set-closure-oprator (lift-f-total P f)
  using lift-f-total-com lift-f-total-ass lift-f-total-idem by unfold-locales blast+

end

sublocale set-closure-partial-oprator ⊆ lift-fun: set-closure-oprator lift-f-total P f
```

by (*simp add*: *lift-set-closure-oprator*)

context *set-closure-partial-oprator* **begin**

abbreviation *lift-closure* $S \equiv$ *lift-fun.closure* (*Some* ‘ S)

inductive-set *pred-closure* **for** S **where**
  base [*simp*]: $s \in S \Longrightarrow s \in$ *pred-closure* S
| *step* [*intro*]: $s \in$ *pred-closure* $S \Longrightarrow t \in$ *pred-closure* $S \Longrightarrow P\ s\ t \Longrightarrow f\ s\ t \in$ *pred-closure* S

lemma *pred-closure-to-some-lift-closure*:
  assumes $s \in$ *pred-closure* S
  shows *Some* $s \in$ *lift-closure* S **using** *assms*
proof (*induct*)
  case (*step* s t)
  then have *lift-f-total* P f (*Some* s) (*Some* t) $\in$ *lift-closure* S
    by (*intro lift-fun.closure.step*) *auto*
  then show *?case* **using** *step*(*5*)
    by (*auto split*: *if-splits*)
qed *auto*

lemma *some-lift-closure-pred-closure*:
  fixes t **defines** $s \equiv$ *Some* t
  assumes *Some* $t \in$ *lift-closure* S
  shows $t \in$ *pred-closure* S **using** *assms*(*2*)
  unfolding *assms*(*1*)[*symmetric*] **using** *assms*(*1*)
proof (*induct arbitrary*: t)
  case (*step* s u)
  from *step*(*5*) **obtain** v w **where** [*simp*]: $s =$ *Some* v $u =$ *Some* w **by** *auto*
  show *?case* **using** *step* **by** (*auto split*: *if-splits*)
qed *auto*

lemma *pred-closure-lift-closure*:
  pred-closure $S =$ *the* ‘ (*lift-closure* $S - \{None\}$) (**is** *?LS* = *?RS*)
proof
  {fix s **assume** $s \in$ *?LS*
    from *pred-closure-to-some-lift-closure*[*OF this*] **have** $s \in$ *?RS*
      by (*metis DiffI empty-iff image-iff insertE option.distinct*(*1*) *option.sel*)**}**
  then show *?LS* $\subseteq$ *?RS* **by** *blast*
next
  {fix s **assume** *ass*: $s \in$ *?RS*
    then have *Some* $s \in$ *lift-closure* S
      using *option.collapse* **by** *fastforce*
    from *some-lift-closure-pred-closure*[*OF this*] **have** $s \in$ *?LS*
      using *option.collapse* **by** *auto***}**
  then show *?RS* $\subseteq$ *?LS* **by** *blast*
qed

lemma *finite-S-finite-closure* [*simp*, *intro*]:
finite S $\Longrightarrow$ *finite* (*pred-closure S*)
using *finite-subset*[*of Some ‘ pred-closure S lift-closure S*]
using *pred-closure-to-some-lift-closure lift-fun.finite-S-finite-closure*[*of Some ‘ S*]
by (*auto simp add*: *pred-closure-lift-closure set-closure-partial-oprator-axioms*)

lemma *closure-mono*:
assumes $S \subseteq T$
shows *pred-closure S* $\subseteq$ *pred-closure T*
proof −
have *Some ‘ S* $\subseteq$ *Some ‘ T* **using** *assms* **by** *auto*
from *lift-fun.closure-mono*[*OF this*] **show** *?thesis*
using *pred-closure-to-some-lift-closure some-lift-closure-pred-closure set-closure-partial-oprator-axioms*
by *fastforce*
qed

lemma *pred-closure-empty* [*simp*]:
pred-closure {} = {}
using *pred-closure-lift-closure* **by** *fastforce*
end

locale *semilattice-closure-partial-operator* =
cl: *set-closure-partial-oprator P f* **for** *P* **and** *f* :: *'a* ⇒ *'a* ⇒ *'a* +
fixes *less-eq e*
assumes *neut-elm* :$\bigwedge$ *x*. *f e x* = *x*
and *neut-pred*: $\bigwedge$ *x*. *P e x*
and *neut-less*: $\bigwedge$ *x*. *less-eq e x*
and *pred-less*: $\bigwedge$ *x y z*. *less-eq x y* $\Longrightarrow$ *less-eq z y* $\Longrightarrow$ *P x z*
and *sup-l*: $\bigwedge$ *x y*. *P x y* $\Longrightarrow$ *less-eq x* (*f x y*)
and *sup-r*: $\bigwedge$ *x y*. *P x y* $\Longrightarrow$ *less-eq y* (*f x y*)
and *upper-bound*: $\bigwedge$ *x y z*. *less-eq x z* $\Longrightarrow$ *less-eq y z* $\Longrightarrow$ *less-eq* (*f x y*) *z*
and *trans*: $\bigwedge$ *x y z*. *less-eq x y* $\Longrightarrow$ *less-eq y z* $\Longrightarrow$ *less-eq x z*
and *anti-sym*: $\bigwedge$ *x y*. *less-eq x y* $\Longrightarrow$ *less-eq y x* $\Longrightarrow$ *x* = *y*
begin

abbreviation *lifted-less-eq* ≡ *lift-less-eq-total less-eq*
abbreviation *lifted-fun* ≡ *lift-f-total P f*

lemma *lift-less-eq-None* [*simp*]:
lifted-less-eq None y ⟷ *y* = *None*
by (*cases y*) *auto*

lemma *lift-less-eq-neut-elm* [*simp*]:
lifted-fun (*Some e*) *s* = *s*
using *neut-elm neut-pred* **by** (*cases s*) *auto*

lemma *lift-less-eq-neut-less* [*simp*]:
lifted-less-eq (*Some e*) *s* ⟷ *True*

using *neut-less* **by** (*cases s*) *auto*

lemma *lift-less-eq-sup-l* [*simp*]:
lifted-less-eq x (*lifted-fun x y*) ⟷ *True*
using *sup-l* **by** (*cases x*; *cases y*) *auto*

lemma *lift-less-eq-sup-r* [*simp*]:
lifted-less-eq y (*lifted-fun x y*) ⟷ *True*
using *sup-r* **by** (*cases x*; *cases y*) *auto*

lemma *lifted-less-eq-trans* [*trans*]:
lifted-less-eq x y ⟹ *lifted-less-eq y z* ⟹ *lifted-less-eq x z*
using *trans* **by** (*auto elim*!: *lift-less-eq-total.elims*)

lemma *lifted-less-eq-anti-sym* [*trans*]:
lifted-less-eq x y ⟹ *lifted-less-eq y x* ⟹ *x* = *y*
using *anti-sym* **by** (*auto elim*!: *lift-less-eq-total.elims*)

lemma *lifted-less-eq-upper*:
lifted-less-eq x z ⟹ *lifted-less-eq y z* ⟹ *lifted-less-eq* (*lifted-fun x y*) *z*
using *upper-bound pred-less* **by** (*auto elim*!: *lift-less-eq-total.elims*)

lemma *semilattice-closure-operator-axioms*:
semilattice-closure-operator-axioms (*lift-f-total P f*) (*lift-less-eq-total less-eq*) (*Some e*)
using *lifted-less-eq-upper lifted-less-eq-trans lifted-less-eq-anti-sym*
by *unfold-locales* (*auto elim*!: *lift-f-total.cases*)

end

sublocale *semilattice-closure-partial-operator* ⊆ *lift-ord*: *semilattice-closure-operator lift-f-total P f lift-less-eq-total less-eq Some e*
by (*simp add*: *cl.lift-set-closure-oprator semilattice-closure-operator.intro semilattice-closure-operator-axioms*)

context *semilattice-closure-partial-operator*
begin

abbreviation *supremum* ≡ *lift-ord.supremum*
abbreviation *smaller-subset* ≡ *lift-ord.smaller-subset*

lemma *supremum-impl*:
assumes *supremum* (*set* (*map Some ss*)) = *Some t*
shows *foldr f ss e* = *t* **using** *assms*
proof (*induct ss arbitrary*: *t*)
case (*Cons a ss*)
then show *?case*

by *auto* (*metis cl.lift-f-totalE lift-f-total.simps*(*3*) *option.distinct*(*1*) *option.sel*)

qed *auto*

lemma *supremum-smaller-exists-unique*:
assumes *finite S*
shows ∃! *p*. *supremum* (*smaller-subset* (*Some t*) (*Some ' S*)) = *Some p* **using** *assms*
proof (*induct*)
case (*insert x F*) **show** *?case*
proof (*cases lifted-less-eq* (*Some x*) (*Some t*))
case *True*
obtain *p* **where** *wit*: *supremum* (*smaller-subset* (*Some t*) (*Some ' F*)) = *Some p*
using *insert* **by** *auto*
then have *pred*: *less-eq p t less-eq x t* **using** *True insert*(*1*)
using *lift-ord.supremum-smaller-subset*
by *auto* (*metis finite-imageI lift-less-eq-total.simps*(*3*))
show *?thesis* **using** *insert pred wit pred-less*
by *auto*
next
case *False* **then show** *?thesis*
using *insert* **by** *auto*
qed
qed *auto*

lemma *supremum-neut-or-in-closure*:
assumes *finite S*
shows *the* (*supremum* (*smaller-subset* (*Some t*) (*Some ' S*))) ∈ {*e*} ∪ *cl.pred-closure S*
using *supremum-smaller-exists-unique*[*OF assms*]
using *lift-ord.supremum-in-smaller-closure*[*of Some ' S Some t*] *assms*
by *auto* (*metis cl.some-lift-closure-pred-closure option.sel*)

end

fun *closure-impl* **where**
closure-impl f [] = []
| *closure-impl f* (*x* # *S*) = (*let cS* = *closure-impl f S in remdups* (*x* # *cS* @ *map* (*f x*) *cS*))

lemma (**in** *set-closure-oprator*) *closure-impl* [*simp*]:
set (*closure-impl f S*) = *closure* (*set S*)
by (*induct S*, *auto simp*: *closure-insert Let-def*)

lemma (**in** *set-closure-partial-oprator*) *closure-impl* [*simp*]:
set (*map the* (*removeAll None* (*closure-impl* (*lift-f-total P f*) (*map Some S*)))) = *pred-closure* (*set S*)

using *lift-set-closure-oprator set-closure-oprator.closure-impl pred-closure-lift-closure*
by *auto*

end

3 Rewriting

theory *Rewriting*
imports *Regular-Tree-Relations.Term-Context*
Regular-Tree-Relations.Ground-Terms
Utils
begin

3.1 Type definitions and rewrite relation definitions

type-synonym *'f sig* = (*'f* × *nat*) *set*
type-synonym (*'f*, *'v*) *rule* = (*'f*, *'v*) *term* × (*'f*, *'v*) *term*
type-synonym (*'f*, *'v*) *trs* = (*'f*, *'v*) *rule set*

definition *sig-step* $\mathcal{F}$ $\mathcal{R}$ = {(*s*, *t*). *funas-term s* ⊆ $\mathcal{F}$ ∧ *funas-term t* ⊆ $\mathcal{F}$ ∧ (*s*, *t*) ∈ $\mathcal{R}$}

inductive-set *rstep* :: - ⇒ (*'f*, *'v*) *term rel* **for** *R* :: (*'f*, *'v*) *trs*
where
rstep: ⋀*C σ l r*. (*l*, *r*) ∈ *R* ⟹ *s* = *C*⟨*l* · *σ*⟩ ⟹ *t* = *C*⟨*r* · *σ*⟩ ⟹ (*s*, *t*) ∈ *rstep R*

definition *rstep-r-p-s* :: (*'f*, *'v*) *trs* ⇒ (*'f*, *'v*) *rule* ⇒ *pos* ⇒ (*'f*, *'v*) *subst* ⇒ (*'f*, *'v*) *trs* **where**
rstep-r-p-s R r p σ = {(*s*, *t*). *p* ∈ *poss s* ∧ *p* ∈ *poss t* ∧ *r* ∈ *R* ∧ *ctxt-at-pos s p* = *ctxt-at-pos t p* ∧
s[*p* ← (*fst r* · *σ*)] = *s* ∧ *t*[*p* ← (*snd r* · *σ*)] = *t*}

Rewriting steps below the root position.

definition *nrrstep* :: (*'f*, *'v*) *trs* ⇒ (*'f*, *'v*) *trs* **where**
nrrstep R = {(*s*,*t*). ∃*r i ps σ*. (*s*,*t*) ∈ *rstep-r-p-s R r* (*i*#*ps*) *σ*}

Rewriting step at the root position.

definition *rrstep* :: (*'f*, *'v*) *trs* ⇒ (*'f*, *'v*) *trs* **where**
rrstep R = {(*s*,*t*). ∃*r σ*. (*s*,*t*) ∈ *rstep-r-p-s R r* [] *σ*}

the parallel rewrite relation

inductive-set *par-rstep* :: (*'f*,*'v*)*trs* ⇒ (*'f*,*'v*)*trs* **for** *R* :: (*'f*,*'v*)*trs*
where *root-step*[*intro*]: (*s*,*t*) ∈ *R* ⟹ (*s* · *σ*,*t* · *σ*) ∈ *par-rstep R*
| *par-step-fun*[*intro*]: ⟦⋀ *i*. *i* < *length ts* ⟹ (*ss* ! *i*,*ts* ! *i*) ∈ *par-rstep R*⟧ ⟹ *length ss* = *length ts*
⟹ (*Fun f ss*, *Fun f ts*) ∈ *par-rstep R*
| *par-step-var*[*intro*]: (*Var x*, *Var x*) ∈ *par-rstep R*

3.2 Ground variants connecting to FORT

definition *grrstep* :: ('*f*, '*v*) *trs* ⇒ '*f gterm rel* **where**
grrstep $\mathcal{R}$ = *inv-image* (*rrstep* $\mathcal{R}$) *term-of-gterm*

definition *gnrrstep* :: ('*f*, '*v*) *trs* ⇒ '*f gterm rel* **where**
gnrrstep $\mathcal{R}$ = *inv-image* (*nrrstep* $\mathcal{R}$) *term-of-gterm*

definition *grstep* :: ('*f*, '*v*) *trs* ⇒ '*f gterm rel* **where**
grstep $\mathcal{R}$ = *inv-image* (*rstep* $\mathcal{R}$) *term-of-gterm*

definition *gpar-rstep* :: ('*f*, '*v*) *trs* ⇒ '*f gterm rel* **where**
gpar-rstep $\mathcal{R}$ = *inv-image* (*par-rstep* $\mathcal{R}$) *term-of-gterm*

An alternative induction scheme that treats the rule-case, the substition-case, and the context-case separately.

lemma *rstep-induct* [*consumes 1*, *case-names rule subst ctxt*]:
assumes (*s*, *t*) ∈ *rstep R*
and *rule*: ⋀*l r*. (*l*, *r*) ∈ *R* ⟹ *P l r*
and *subst*: ⋀*s t* σ. *P s t* ⟹ *P* (*s* · σ) (*t* · σ)
and *ctxt*: ⋀*s t C*. *P s t* ⟹ *P* (*C*⟨*s*⟩) (*C*⟨*t*⟩)
shows *P s t*
using *assms* **by** (*induct*) *auto*

lemmas *rstepI* = *rstep.intros* [*intro*]

lemmas *rstepE* = *rstep.cases* [*elim*]

lemma *rstep-ctxt* [*intro*]: (*s*, *t*) ∈ *rstep R* ⟹ (*C*⟨*s*⟩, *C*⟨*t*⟩) ∈ *rstep R*
by (*force simp flip*: *ctxt-ctxt-compose*)

lemma *rstep-rule* [*intro*]: (*l*, *r*) ∈ *R* ⟹ (*l*, *r*) ∈ *rstep R*
using *rstep.rstep* [**where** *C* = □ **and** σ = *Var* **and** *R* = *R*] **by** *simp*

lemma *rstep-subst* [*intro*]: (*s*, *t*) ∈ *rstep R* ⟹ (*s* · σ, *t* · σ) ∈ *rstep R*
by (*force simp flip*: *subst-subst-compose*)

lemma *nrrstep-def'*:
nrrstep R = {(*s*, *t*). ∃*l r C* σ. (*l*, *r*) ∈ *R* ∧ *C* ≠ □ ∧ *s* = *C*⟨*l*·σ⟩ ∧ *t* = *C*⟨*r*·σ⟩}
(**is** *?Ls* = *?Rs*)
proof
show *?Ls* ⊆ *?Rs*
proof (*rule subrelI*)
fix *s t* **assume** (*s*, *t*) ∈ *?Ls*
then obtain *l r i ps* σ **where** *step*: (*s*, *t*) ∈ *rstep-r-p-s R* (*l*, *r*) (*i* # *ps*) σ
unfolding *nrrstep-def* **by** *best*
let *?C* = *ctxt-at-pos s* (*i* # *ps*)
from *step* **have***i* # *ps* ∈ *poss s* **and** (*l*, *r*) ∈ *R* **and** *s* = *?C*⟨*l*·σ⟩ **and** *t* = *?C*⟨*r*·σ⟩

unfolding *rstep-r-p-s-def Let-def* **by** (*auto simp flip*: *replace-term-at-replace-at-conv*)
moreover from ‹i # $ps \in poss\ s$› **have** $?C \neq \Box$ **by** (*induct s*) *auto*
ultimately show $(s, t) \in ?Rs$ **by** *auto*
qed
next
show $?Rs \subseteq ?Ls$
proof (*rule subrelI*)
fix $s\ t$ **assume** $(s, t) \in ?Rs$
then obtain $l\ r\ C\ \sigma$ **where** *in-R*: $(l, r) \in R$ **and** $C \neq \Box$
and s: $s = C\langle l{\cdot}\sigma\rangle$ **and** t: $t = C\langle r{\cdot}\sigma\rangle$ **by** *auto*
from ‹$C \neq \Box$› **obtain** $i\ p$ **where** *ip*: *hole-pos* $C = i$ # p **by** (*induct C*) *auto*
have i # $p \in poss\ s$ **unfolding** s *ip*[*symmetric*] **by** *simp*
then have C: $C = ctxt\text{-}at\text{-}pos\ s\ (i$ # $p)$
unfolding s *ip*[*symmetric*] **by** *simp*
from ‹i # $p \in poss\ s$› *in-R* $s\ t$ **have** $(s, t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ (l, r)\ (i$ # $p)\ \sigma$
unfolding *rstep-r-p-s-def C*[*symmetric*] *ip*[*symmetric*] **by** *simp*
then show $(s, t) \in nrrstep\ R$ **unfolding** *nrrstep-def* **by** *best*
qed
qed

lemma *rrstep-def'*: $rrstep\ R = \{(s, t).\ \exists\, l\ r\ \sigma.\ (l, r) \in R \wedge s = l{\cdot}\sigma \wedge t = r{\cdot}\sigma\}$
by (*auto simp*: *rrstep-def rstep-r-p-s-def*)

lemma *rstep-imp-C-s-r*:
assumes $(s,t) \in rstep\ R$
shows $\exists\, C\ \sigma\ l\ r.\ (l,r) \in R \wedge s = C\langle l{\cdot}\sigma\rangle \wedge t = C\langle r{\cdot}\sigma\rangle$ **using** *assms*
by (*induct*) *auto*

lemma *rhs-wf*:
assumes *R*: $(l, r) \in R$ **and** $funas\text{-}trs\ R \subseteq F$
shows $funas\text{-}term\ r \subseteq F$
using *assms* **by** (*force simp*: *funas-trs-def*)

abbreviation $linear\text{-}sys\ \mathcal{R} \equiv (\forall\ (l, r) \in \mathcal{R}.\ linear\text{-}term\ l \wedge linear\text{-}term\ r)$
abbreviation $const\text{-}subt\ c \equiv \lambda\ x.\ Fun\ c\ []$

end

4 Primitive constructions

theory *LV-to-GTT*
imports *Regular-Tree-Relations.Pair-Automaton*
Bot-Terms
Rewriting
begin

4.1 Recognizing subterms of linear terms

abbreviation *ffunas-terms* **where**
ffunas-terms $R \equiv |\bigcup|$ (*ffunas-term* $|`|$ R)

definition *states* $R \equiv \{t^{\perp} \mid s\ t.\ s \in R \wedge s \trianglerighteq t\}$

lemma *states-conv*:
states R = *term-to-bot-term* ‘ $(\bigcup\ s \in R.$ *subterms* $s)$
unfolding *states-def set-all-subteq-subterms*
by *auto*

lemma *finite-states*:
assumes *finite* R **shows** *finite* (*states* R)
proof −
have *conv*: *states* R = *term-to-bot-term* ‘ $(\bigcup\ s \in R.\ \{t \mid t.\ s \trianglerighteq t\})$
unfolding *states-def* **by** *auto*
from *assms* **have** *finite* $(\bigcup\ s \in R.\ \{t \mid t.\ s \trianglerighteq t\})$
by (*intro finite-UN-I2 finite-imageI*) (*simp add*: *finite-subterms*)+
then show *?thesis* **using** *conv* **by** *auto*
qed

lemma *root-bot-diff*:
root-bot ‘ $(R - \{Bot\})$ = (*root-bot* ‘ R) − $\{None\}$
using *root-bot.elims* **by** *auto*

lemma *root-bot-states-root-subterms*:
the ‘ (*root-bot* ‘ (*states* R − $\{Bot\}$)) = *the* ‘ (*root* ‘ $(\bigcup\ s \in R.$ *subterms* $s)$ − $\{None\}$)
unfolding *states-conv root-bot-diff*
unfolding *image-comp*
by *simp*

context
includes *fset.lifting*
begin

lift-definition *fstates* :: $('f, 'v)$ *term fset* ⇒ $'f$ *bot-term fset* **is** *states*
by (*simp add*: *finite-states*)

lift-definition *fsubterms* :: $('f, 'v)$ *term* ⇒ $('f, 'v)$ *term fset* **is** *subterms*
by (*simp add*: *finite-subterms-fun*)

lemmas *fsubterms* [*code*] = *subterms.simps*[*Transfer.transferred*]

lift-definition *ffunas-trs* :: $(('f, 'v)$ *term* × $('f, 'v)$ *term*) *fset* ⇒ $('f \times nat)$ *fset* **is** *funas-trs*
by (*simp add*: *finite-funas-trs*)

lemma *fstates-def'*:

t |∈| *fstates* $R \longleftrightarrow (\exists\ s\ u.\ s$ |∈| $R \wedge s \trianglerighteq u \wedge u^{\perp} = t)$
by *transfer* (*auto simp*: *states-def*)

lemma *fstates-fmemberE* [*elim*!]:
assumes t |∈| *fstates* R
obtains $s\ u$ **where** s |∈| $R \wedge s \trianglerighteq u \wedge u^{\perp} = t$
using *assms* **unfolding** *fstates-def'*
by *blast*

lemma *fstates-fmemberI* [*intro*]:
s |∈| $R \Longrightarrow s \trianglerighteq u \Longrightarrow u^{\perp}$ |∈| *fstates* R
unfolding *fstates-def'* **by** *blast*

lemmas *froot-bot-states-root-subterms* = *root-bot-states-root-subterms*[*Transfer.transferred*]
lemmas *root-fsubsterms-ffunas-term-fset* = *root-substerms-funas-term-set*[*Transfer.transferred*]

lemma *fstates*[*code*]:
fstates R = *term-to-bot-term* |‘| (|⋃| (*fsubterms* |‘| R))
by *transfer* (*auto simp*: *states-conv*)

end

definition *ta-rule-sig* **where**
ta-rule-sig = (λ r. (*r-root* r, *length* (*r-lhs-states* r)))

primrec *term-to-ta-rule* **where**
term-to-ta-rule (*BFun* f *ts*) = *TA-rule* f *ts* (*BFun* f *ts*)

lemma *ta-rule-sig-term-to-ta-rule-root*:
$t \neq$ *Bot* $\Longrightarrow$ *ta-rule-sig* (*term-to-ta-rule* t) = *the* (*root-bot* t)
by (*cases* t) (*auto simp*: *ta-rule-sig-def*)

lemma *ta-rule-sig-term-to-ta-rule-root-set*:
assumes *Bot* |∉| R
shows *ta-rule-sig* |‘| (*term-to-ta-rule* |‘| R) = *the* |‘| (*root-bot* |‘| R)
proof −
{fix x **assume** x |∈| R **then have** *ta-rule-sig* (*term-to-ta-rule* x) = *the* (*root-bot* x)
using *ta-rule-sig-term-to-ta-rule-root*[*of* x] *assms*
by *auto***}**
then show *?thesis*
by (*force simp*: *fimage-iff*)
qed

definition *pattern-automaton-rules* **where**
pattern-automaton-rules $\mathcal{F}$ R =
(*let states* = (*fstates* R) − {|*Bot*|} *in*
term-to-ta-rule |‘| *states* |×| (λ (f, n). *TA-rule* f (*replicate* n *Bot*) *Bot*) |‘| $\mathcal{F}$)

lemma *pattern-automaton-rules-BotD*:
assumes *TA-rule f ss Bot* $|\in|$ *pattern-automaton-rules* $\mathcal{F}$ *R*
shows *TA-rule f ss Bot* $|\in|$ $(\lambda\ (f,\ n).$ *TA-rule f* (*replicate n Bot*) *Bot*$)$ $|`|$ $\mathcal{F}$
using *assms*
by (*auto simp*: *pattern-automaton-rules-def*)
(*metis ta-rule.inject term-to-bot-term.elims term-to-ta-rule.simps*)

lemma *pattern-automaton-rules-FunD*:
assumes *TA-rule f ss* (*BFun g ts*) $|\in|$ *pattern-automaton-rules* $\mathcal{F}$ *R*
shows $g = f \wedge ts = ss \wedge$
TA-rule f ss (*BFun g ts*) $|\in|$ *term-to-ta-rule* $|`|$ ((*fstates R*) $-$ $\{|Bot|\}$) **using** *assms*
apply (*auto simp*: *pattern-automaton-rules-def*)
apply (*metis bot-term.exhaust ta-rule.inject term-to-ta-rule.simps*)
by (*metis* (*no-types*, *lifting*) *ta-rule.inject term-to-bot-term.elims term-to-ta-rule.simps*)

definition *pattern-automaton* **where**
pattern-automaton $\mathcal{F}$ *R* $=$ *TA* (*pattern-automaton-rules* $\mathcal{F}$ *R*) $\{||\}$

lemma *ta-sig-pattern-automaton* [*simp*]:
ta-sig (*pattern-automaton* $\mathcal{F}$ *R*) $=$ $\mathcal{F}$ $|\cup|$ *ffunas-terms R*
proof $-$
let *?r* $=$ *ta-rule-sig*
have $*$:*Bot* $|\notin|$ (*fstates R*) $-$ $\{|Bot|\}$ **by** *simp*
have *f*: $\mathcal{F}$ $=$ *?r* $|`|$ $((\lambda\ (f,\ n).$ *TA-rule f* (*replicate n Bot*) *Bot*$)$ $|`|$ $\mathcal{F})$
by (*auto simp*: *image-iff Bex-def ta-rule-sig-def split*!: *prod.splits*)
moreover have *ffunas-terms R* $=$ *?r* $|`|$ (*term-to-ta-rule* $|`|$ ((*fstates R*) $-$ $\{|Bot|\}$))
unfolding *ta-rule-sig-term-to-ta-rule-root-set*[*OF* $*$]
unfolding *froot-bot-states-root-subterms root-fsubsterms-ffunas-term-fset*
by *simp*
ultimately show *?thesis* **unfolding** *pattern-automaton-def ta-sig-def*
unfolding *ta-rule-sig-def pattern-automaton-rules-def*
by (*simp add*: *fimage-funion comp-def*) *blast*
qed

lemma *terms-reach-Bot*:
assumes *ffunas-gterm t* $|\subseteq|$ $\mathcal{F}$
shows *Bot* $|\in|$ *ta-der* (*pattern-automaton* $\mathcal{F}$ *R*) (*term-of-gterm t*) **using** *assms*
proof (*induct t*)
case (*GFun f ts*)
have [*simp*]: *s* $\in$ *set ts* $\Longrightarrow$ *ffunas-gterm s* $|\subseteq|$ $\mathcal{F}$ **for** *s* **using** *GFun*(*2*)
using *in-set-idx* **by** *fastforce*
from *GFun* **show** *?case*
by (*auto simp*: *pattern-automaton-def pattern-automaton-rules-def rev-fimage-eqI*
intro!: *exI*[*of - replicate* (*length ts*) *Bot*])
qed

lemma *pattern-automaton-reach-smaller-term*:
assumes $l \mid\in\mid R\ l \trianglerighteq s\ s^\bot \leq_b (\textit{term-of-gterm}\ t)^\bot$ *ffunas-gterm* $t \mid\subseteq\mid \mathcal{F}$
shows $s^\bot \mid\in\mid$ *ta-der* (*pattern-automaton* $\mathcal{F}\ R$) (*term-of-gterm* t) **using** *assms*(2−)
proof (*induct t arbitrary*: *s*)
case (*GFun f ts*) **show** *?case*
proof (*cases s*)
case (*Var x*)
then show *?thesis* **using** *terms-reach-Bot*[*OF GFun*(4)]
by (*auto simp del*: *ta-der-Fun*)
next
case [*simp*]: (*Fun g ss*)
let *?ss* = *map term-to-bot-term ss*
have [*simp*]: $s \in$ *set ts* $\Longrightarrow$ *ffunas-gterm s* $\mid\subseteq\mid \mathcal{F}$ **for** *s* **using** *GFun*(4)
using *in-set-idx* **by** *fastforce*
from *GFun*(3) **have** *s*: *g* = *f length ss* = *length ts* **by** *auto*
from *GFun*(2) *s*(2) *assms*(1) **have** *rule*: *TA-rule f ?ss* (*BFun f ?ss*) $\mid\in\mid$ *pattern-automaton-rules* $\mathcal{F}\ R$
by (*auto simp*: *s*(1) *pattern-automaton-rules-def image-iff Bex-def*)
{fix *i* **assume** *bound*: *i* < *length ts*
then have *sub*: $l \trianglerighteq$ *ss* ! *i* **using** *GFun*(2) *arg-subteq*[*OF nth-mem*, *of i ss f*]
unfolding *Fun s*(1) **using** *s*(2) **by** (*metis subterm.order.trans*)
have *ss* ! $i^\bot \leq_b$ (*term-of-gterm* (*ts* ! *i*)):: ('*a*, '*c*) *term*)$^\bot$ **using** *GFun*(3) *bound s*(2)
by (*auto simp*: *s elim*!: *bless-eq.cases*)
from *GFun*(1)[*OF nth-mem sub this*] *bound*
have *ss* ! $i^\bot \mid\in\mid$ *ta-der* (*pattern-automaton* $\mathcal{F}\ R$) (*term-of-gterm* (*ts* ! *i*))
by *auto***}**
then show *?thesis* **using** *GFun*(2−) *s*(2) *rule*
by (*auto simp*: *s*(1) *pattern-automaton-def intro*!: *exI*[*of* - *?ss*] *exI*[*of* - *BFun f ?ss*])
qed
qed

lemma *bot-term-of-gterm-conv*:
term-of-gterm $s^\bot$ = *term-of-gterm* $s^\bot$
by (*induct s*) *auto*

lemma *pattern-automaton-ground-instance-reach*:
assumes $l \mid\in\mid R\ l \cdot \sigma = (\textit{term-of-gterm}\ t)$ *ffunas-gterm* $t \mid\subseteq\mid \mathcal{F}$
shows $l^\bot \mid\in\mid$ *ta-der* (*pattern-automaton* $\mathcal{F}\ R$) (*term-of-gterm t*)
proof −
let *?t* = (*term-of-gterm t*) :: ('*a*, '*a bot-term*) *term*
from *instance-to-bless-eq*[*OF assms*(2)] **have** *sm*: $l^\bot \leq_b$ *?t*$^\bot$
using *bot-term-of-gterm-conv* **by** *metis*
show *?thesis* **using** *pattern-automaton-reach-smaller-term*[*OF assms*(1) - *sm*] *assms*(3−)
by *auto*
qed

lemma *pattern-automaton-reach-smallet-term*:
assumes $l^\bot$ $|\in|$ *ta-der* (*pattern-automaton* $\mathcal{F}$ R) t *ground* t
shows $l^\bot \leq_b t^\bot$ **using** *assms*
proof (*induct* t *arbitrary*: l)
case (*Fun f ts*) **note** *IH* = *this* **show** *?case*
proof (*cases* l)
case (*Fun g ss*)
let *?ss* = *map term-to-bot-term ss*
from *IH*(*2*) **have** *rule*: $g = f$ *length ss* = *length ts*
TA-rule f ?ss (*BFun f ?ss*) $|\in|$ *rules* (*pattern-automaton* $\mathcal{F}$ R)
by (*auto simp*: *Fun pattern-automaton-def dest*: *pattern-automaton-rules-FunD*)
{fix i **assume** $i <$ *length ts*
then have *ss* ! $i^\bot \leq_b$ *ts* ! $i^\bot$ **using** *IH*(*2*, *3*) *rule*(*2*)
by (*intro IH*(*1*)) (*auto simp*: *Fun pattern-automaton-def dest*: *pattern-automaton-rules-FunD*)**}**
then show *?thesis* **using** *rule*(*2*)
by (*auto simp*: *Fun rule*(*1*))
qed *auto*
qed *auto*

4.2 Recognizing root step relation of LV-TRSs

definition *lv-trs* :: ('*f*, '*v*) *trs* ⇒ *bool* **where**
lv-trs $R \equiv \forall (l, r) \in R$. *linear-term* $l \wedge$ *linear-term* $r \wedge$ (*vars-term* $l \cap$ *vars-term* $r = \{\}$)

lemma *subst-unification*:
assumes *vars-term* $s \cap$ *vars-term* $t = \{\}$
obtains μ **where** $s \cdot \sigma = s \cdot \mu$ $t \cdot \tau = t \cdot \mu$
using *assms*
by (*auto intro*!: *that*[*of* λx. *if* $x \in$ *vars-term s then* σ x *else* τ x] *simp*: *term-subst-eq-conv*)

lemma *lv-trs-subst-unification*:
assumes *lv-trs* R $(l, r) \in R$ $s = l \cdot \sigma$ $t = r \cdot \tau$
obtains μ **where** $s = l \cdot \mu \wedge t = r \cdot \mu$
using *assms subst-unification*[*of* l r σ τ]
unfolding *lv-trs-def*
by (*force split*!: *prod.splits*)

definition Rel_f **where**
Rel_f R = *map-both term-to-bot-term* $|`|$ R

definition *root-pair-automaton* **where**
root-pair-automaton $\mathcal{F}$ R = (*pattern-automaton* $\mathcal{F}$ (*fst* $|`|$ R),
pattern-automaton $\mathcal{F}$ (*snd* $|`|$ R))

definition *agtt-grrstep* **where**
agtt-grrstep $\mathcal{R}$ $\mathcal{F}$ = *pair-at-to-agtt'* (*root-pair-automaton* $\mathcal{F}$ $\mathcal{R}$) (Rel_f $\mathcal{R}$)

lemma *agtt-grrstep-eps-trancl* [*simp*]:
(*eps* (*fst* (*agtt-grrstep* $\mathcal{R}$ $\mathcal{F}$)))$|^+|$ = *eps* (*fst* (*agtt-grrstep* $\mathcal{R}$ $\mathcal{F}$))
(*eps* (*snd* (*agtt-grrstep* $\mathcal{R}$ $\mathcal{F}$))) = {||}
by (*auto simp add*: *agtt-grrstep-def pair-at-to-agtt'-def*
pair-at-to-agtt-def Let-def root-pair-automaton-def pattern-automaton-def
fmap-states-ta-def intro!: *frelcomp-empty-ftrancl-simp*)

lemma *root-pair-automaton-grrstep*:
fixes R :: ('*f*, '*v*) *rule fset*
assumes *lv-trs* (*fset R*) *ffunas-trs R* $|\subseteq|$ $\mathcal{F}$
shows *pair-at-lang* (*root-pair-automaton* $\mathcal{F}$ *R*) (*Rel*$_f$ *R*) = *Restr* (*grrstep* (*fset*
R)) ($\mathcal{T}_G$ (*fset* $\mathcal{F}$)) (**is** *?Ls* = *?Rs*)
proof
let *?t-o-g* = *term-of-gterm* :: '*f gterm* ⇒ ('*f*, '*v*) *Term.term*
have [*simp*]: $\mathcal{F}$ $|\cup|$ $|\bigcup|$ ((*ffunas-term* ∘ *fst*) $|`|$ *R*) = $\mathcal{F}$
$\mathcal{F}$ $|\cup|$ $|\bigcup|$ ((*ffunas-term* ∘ *snd*) $|`|$ *R*) = $\mathcal{F}$ **using** *assms*(*2*)
by (*force simp*: *less-eq-fset.rep-eq ffunas-trs.rep-eq funas-trs-def ffunas-term.rep-eq*
ffUnion.rep-eq)+
{fix *s t* **assume** (*s*, *t*) ∈ *?Ls*
from *pair-at-langE*[*OF this*] **obtain** *p q* **where** *st*: (*q*, *p*) $|\in|$ *Rel*$_f$ *R*
q $|\in|$ *gta-der* (*fst* (*root-pair-automaton* $\mathcal{F}$ *R*)) *s p* $|\in|$ *gta-der* (*snd* (*root-pair-automaton*
$\mathcal{F}$ *R*)) *t*
by *blast*
from *st*(*1*) **obtain** *l r* **where** *tm*: *q* = $l^\perp$ *p* = $r^\perp$ (*l*, *r*) $|\in|$ *R* **unfolding**
Rel$_f$*-def*
using *assms*(*1*) **by** *auto*
have *sm*: $l^\perp$ $\leq_b$ (*?t-o-g s*)$^\perp$ $r^\perp$ $\leq_b$ (*?t-o-g t*)$^\perp$
using *pattern-automaton-reach-smallet-term*[*of l* $\mathcal{F}$ *fst* $|`|$ *R term-of-gterm s*]
using *pattern-automaton-reach-smallet-term*[*of r* $\mathcal{F}$ *snd* $|`|$ *R term-of-gterm*
t]
using *st*(*2*, *3*) *tm*(*3*) **unfolding** *tm*
by (*auto simp*: *gta-der-def root-pair-automaton-def*) (*smt* (*verit*) *bot-term-of-gterm-conv*)+
have *linear-term l linear-term r* **using** *tm*(*3*) *assms*(*1*)
by (*auto simp*: *lv-trs-def*)
then obtain σ τ **where** *l* · σ = *?t-o-g s r* · τ = *?t-o-g t* **using** *sm*
by (*auto dest*!: *bless-eq-to-instance*)
then obtain μ **where** *subst*: *l* · μ = *?t-o-g s r* · μ = *?t-o-g t*
using *lv-trs-subst-unification*[*OF assms*(*1*) *tm*(*3*), *of ?t-o-g s* σ *?t-o-g t* τ]
by *metis*
moreover have *s* ∈ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) *t* ∈ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) **using** *st*(*2*−) *assms*
using *ta-der-gterm-sig*[*of q pattern-automaton* $\mathcal{F}$ (*fst* $|`|$ *R*) *s*]
using *ta-der-gterm-sig*[*of p pattern-automaton* $\mathcal{F}$ (*snd* $|`|$ *R*) *t*]
by (*auto simp*: *gta-der-def root-pair-automaton-def* $\mathcal{T}_G$*-equivalent-def less-eq-fset.rep-eq*
ffunas-gterm.rep-eq)
ultimately have (*s*, *t*) ∈ *?Rs* **using** *tm*(*3*)
by (*auto simp*: *grrstep-def rrstep-def'*) *metis***}**
then show *?Ls* ⊆ *?Rs* **by** *auto*
next
let *?t-o-g* = *term-of-gterm* :: '*f gterm* ⇒ ('*f*, '*v*) *Term.term*

```
  {fix s t assume (s, t) ∈ ?Rs
    then obtain σ l r where st: (l, r) |∈| R l · σ = ?t-o-g s r · σ = ?t-o-g t s ∈
𝒯_G (fset ℱ) t ∈ 𝒯_G (fset ℱ)
      by (auto simp: grrstep-def rrstep-def')
    have funas: ffunas-gterm s |⊆| ℱ ffunas-gterm t |⊆| ℱ using st(4, 5)
      by (auto simp: 𝒯_G-equivalent-def)
        (metis ffunas-gterm.rep-eq subsetD)+
    from st(1) have (l^⊥, r^⊥) |∈| Rel_f R unfolding Rel_f-def using assms(1)
      by (auto simp: fimage-iff fBex-def)
    then have (s, t) ∈ ?Ls using st
     using pattern-automaton-ground-instance-reach[of l fst |`| R σ, OF - - funas(1)]
      using pattern-automaton-ground-instance-reach[of r snd |`| R σ, OF - - fu-
nas(2)]
      by (auto simp: 𝒯_G-equivalent-def image-iff Bex-def root-pair-automaton-def
gta-der-def pair-at-lang-def)}
  then show ?Rs ⊆ ?Ls by auto
qed


lemma agtt-grrstep:
  fixes R :: ('f, 'v) rule fset
  assumes lv-trs (fset R) ffunas-trs R |⊆| ℱ
  shows agtt-lang (agtt-grrstep R ℱ) = Restr (grrstep (fset R)) (𝒯_G (fset ℱ))
 using root-pair-automaton-grrstep[OF assms] unfolding pair-at-agtt-cost agtt-grrstep-def
  by simp


lemma root-pair-automaton-grrstep-set:
  fixes R :: ('f, 'v) rule set
  assumes finite R finite ℱ lv-trs R funas-trs R ⊆ ℱ
 shows pair-at-lang (root-pair-automaton (Abs-fset ℱ) (Abs-fset R)) (Rel_f (Abs-fset
R)) = Restr (grrstep R) (𝒯_G ℱ)
proof −
  from assms(1, 2, 4) have ffunas-trs (Abs-fset R) |⊆| Abs-fset ℱ
    by (auto simp add: Abs-fset-inverse ffunas-trs.rep-eq subset-eq)
  from root-pair-automaton-grrstep[OF - this] assms
  show ?thesis
    by (auto simp: Abs-fset-inverse)
qed

lemma agtt-grrstep-set:
  fixes R :: ('f, 'v) rule set
  assumes finite R finite ℱ lv-trs R funas-trs R ⊆ ℱ
  shows agtt-lang (agtt-grrstep (Abs-fset R) (Abs-fset ℱ)) = Restr (grrstep R) (𝒯_G
ℱ)
 using root-pair-automaton-grrstep-set[OF assms] unfolding pair-at-agtt-cost agtt-grrstep-def
  by simp

end
```

theory *NF*
imports
Saturation
Bot-Terms
Regular-Tree-Relations.Tree-Automata
begin

4.3 Recognizing normal forms of left linear TRSs

interpretation *lift-total*: *semilattice-closure-partial-operator* $\lambda\ x\ y.\ (x, y) \in mergeP$ $(\uparrow)\ \lambda\ x\ y.\ x \leq_b y\ Bot$
apply *unfold-locales* **apply** (*auto simp*: *merge-refl merge-symmetric merge-terms-assoc merge-terms-idem merge-bot-args-bless-eq-merge*)
using *merge-dist* **apply** *blast*
using *megeP-ass* **apply** *blast*
using *merge-terms-commutative* **apply** *blast*
apply (*metis bless-eq-mergeP bless-eq-trans merge-bot-args-bless-eq-merge merge-dist merge-symmetric merge-terms-commutative*)
apply (*metis merge-bot-args-bless-eq-merge merge-symmetric merge-terms-commutative*)
using *bless-eq-closued-under-supremum bless-eq-trans bless-eq-anti-sym*
by *blast+*

abbreviation *psubt-lhs-bot* $R \equiv \{t^{\perp} \mid s\ t.\ s \in R \wedge s \rhd t\}$
abbreviation *closure* $S \equiv$ *lift-total.cl.pred-closure* S

definition *states* **where**
states $R =$ *insert Bot* (*closure* (*psubt-lhs-bot* R))

lemma *psubt-mono*:
$R \subseteq S \Longrightarrow$ *psubt-lhs-bot* $R \subseteq$ *psubt-lhs-bot* S **by** *auto*

lemma *states-mono*:
$R \subseteq S \Longrightarrow$ *states* $R \subseteq$ *states* S
unfolding *states-def* **using** *lift-total.cl.closure-mono*[*OF psubt-mono*[*of R S*]]
by *auto*

lemma *finite-lhs-subt* [*simp*, *intro*]:
assumes *finite* R
shows *finite* (*psubt-lhs-bot* R)
proof −
have *conv*: *psubt-lhs-bot* $R =$ *term-to-bot-term* ‘ $\{t \mid s\ t\ .\ s \in R \wedge s \rhd t\}$ **by** *auto*
from *assms* **have** *finite* $\{t \mid s\ t\ .\ s \in R \wedge s \rhd t\}$
by (*simp add*: *finite-strict-subterms*)
then show *?thesis* **using** *conv* **by** *auto*
qed

lemma *states-ref-closure*:
states $R \subseteq$ *insert Bot* (*closure* (*psubt-lhs-bot* R))
unfolding *states-def* **by** *auto*

lemma *finite-R-finite-states* [*simp*, *intro*]:
finite R $\Longrightarrow$ *finite* (*states R*)
using *finite-lhs-subt states-ref-closure*
using *lift-total.cl.finite-S-finite-closure finite-subset*
by *fastforce*

abbreviation *lift-sup-small s S* ≡ *lift-total.supremum* (*lift-total.smaller-subset* (*Some s*) (*Some ‘ S*))
abbreviation *bound-max s S* ≡ *the* (*lift-sup-small s S*)

lemma *bound-max-state-set*:
assumes *finite R*
shows *bound-max t* (*psubt-lhs-bot R*) ∈ *states R*
using *lift-total.supremum-neut-or-in-closure*[*OF finite-lhs-subt*[*OF assms*], *of t*]
unfolding *states-def* **by** *auto*

context
includes *fset.lifting*
begin
lift-definition *fstates* :: (′*a*, ′*b*) *term fset* ⇒ ′*a bot-term fset* **is** *states*
by *simp*

lemma *bound-max-state-fset*:
bound-max t (*psubt-lhs-bot* (*fset R*)) |∈| *fstates R*
using *bound-max-state-set*[*of fset R t*]
using *fstates.rep-eq* **by** *fastforce*

end

definition *nf-rules* **where**
nf-rules R $\mathcal{F}$ = {|*TA-rule f qs q* | *f qs q*. (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *fset-of-list qs* |⊆| *fstates R* ∧
¬(∃ *l* |∈| *R*. $l^{\bot}$ ≤$_b$ *BFun f qs*) ∧ *q* = *bound-max* (*BFun f qs*) (*psubt-lhs-bot* (*fset R*))|}

lemma *nf-rules-fmember*:
TA-rule f qs q |∈| *nf-rules R* $\mathcal{F}$ ⟷ (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *fset-of-list qs* |⊆| *fstates R* ∧
¬(∃ *l* |∈| *R*. $l^{\bot}$ ≤$_b$ *BFun f qs*) ∧ *q* = *bound-max* (*BFun f qs*) (*psubt-lhs-bot* (*fset R*))
proof −
let *?subP* = λ *n qs*. *fset-of-list qs* |⊆| *fstates R* ∧ *length qs* = *n*
let *?sub* = λ *n*. *Collect* (*?subP n*)
have ∗: *finite* (*?sub n*) **for** *n*
using *finite-lists-length-eq*[*of fset* (*fstates R*) *n*]
by (*simp add*: *less-eq-fset.rep-eq fset-of-list.rep-eq*)
{fix *f n* **assume** *mem*: (*f*, *n*) ∈ *fset* $\mathcal{F}$
have ∗∗: {*f*} × (*?sub n*) = {(*f*, *qs*) |*qs*. *?subP n qs*} **by** *auto*

from *mem* **have** *finite* {(*f*, *qs*) |*qs*. *?subP n qs*} **using** ∗
using *finite-cartesian-product*[*OF* - ∗[*of n*], *of* {*f*}] **unfolding** ∗∗ **by** *simp*}
then have ∗: *finite* (⋃ (*f*, *n*) ∈ *fset* $\mathcal{F}$. {(*f*, *qs*) | *qs*. *?subP n qs*}) **by** *auto*
have ∗∗: (⋃ (*f*, *n*) ∈ *fset* $\mathcal{F}$. {(*f*, *qs*) | *qs*. *?subP n qs*}) = {(*f*, *qs*) | *f qs*. (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *?subP* (*length qs*) *qs*}
by *auto*
have ∗: *finite* ({(*f*, *qs*) | *f qs*. (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *?subP* (*length qs*) *qs*} × *fset* (*fstates R*))
using ∗ **unfolding** ∗∗ **by** (*intro finite-cartesian-product*) *auto*
have ∗∗: {*TA-rule f qs q* | *f qs q*. (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *fset-of-list qs* |⊆| *fstates R* ∧ *q* |∈| *fstates R*} =
(λ ((*f*, *qs*), *q*). *TA-rule f qs q*) ‘ ({(*f*, *qs*) | *f qs*. (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *?subP* (*length qs*) *qs*} × *fset* (*fstates R*))
by (*auto simp*: *image-def split*!: *prod.splits*)
have *f*: *finite* {*TA-rule f qs q* | *f qs q*. (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *fset-of-list qs* |⊆| *fstates R* ∧ *q* |∈| *fstates R*}
unfolding ∗∗ **using** ∗ **by** *auto*
show *?thesis*
by (*auto simp*: *nf-rules-def bound-max-state-fset intro*!: *finite-subset*[*OF* - *f*])
qed

definition *nf-ta* **where**
nf-ta R $\mathcal{F}$ = *TA* (*nf-rules R* $\mathcal{F}$) {||}

definition *nf-reg* **where**
nf-reg R $\mathcal{F}$ = *Reg* (*fstates R*) (*nf-ta R* $\mathcal{F}$)

lemma *bound-max-sound*:
assumes *finite R*
shows *bound-max t* (*psubt-lhs-bot R*) ≤$_b$ *t*
using *assms lift-total.lift-ord.supremum-smaller-subset*[*of Some* ‘ *psubt-lhs-bot R Some t*]
by *auto* (*metis* (*no-types*, *lifting*) *lift-less-eq-total.elims*(*2*) *option.sel option.simps*(*3*))

lemma *Bot-in-filter*:
Bot ∈ *Set.filter* (λ*s*. *s* ≤$_b$ *t*) (*states R*)
by (*auto simp*: *Set.filter-def states-def*)

lemma *bound-max-exists*:
∃ *p*. *p* = *bound-max t* (*psubt-lhs-bot R*)
by *blast*

lemma *bound-max-unique*:
assumes *p* = *bound-max t* (*psubt-lhs-bot R*) **and** *q* = *bound-max t* (*psubt-lhs-bot R*)
shows *p* = *q* **using** *assms* **by** *force*

lemma *nf-rule-to-bound-max*:
f qs → *q* |∈| *nf-rules R* $\mathcal{F}$ ⟹ *q* = *bound-max* (*BFun f qs*) (*psubt-lhs-bot* (*fset*

R))
by (*auto simp*: *nf-rules-fmember*)

lemma *nf-rules-unique*:
assumes $f\ qs \to q \mid\in\mid$ *nf-rules R* $\mathcal{F}$ **and** $f\ qs \to q' \mid\in\mid$ *nf-rules R* $\mathcal{F}$
shows $q = q'$ **using** *assms* **unfolding** *nf-rules-def*
using *nf-rule-to-bound-max*[*OF assms*(*1*)] *nf-rule-to-bound-max*[*OF assms*(*2*)]
using *bound-max-unique* **by** *blast*

lemma *nf-ta-det*:
shows *ta-det* (*nf-ta R* $\mathcal{F}$)
by (*auto simp add*: *ta-det-def nf-ta-def nf-rules-unique*)

lemma *term-instance-of-reach-state*:
assumes $q \mid\in\mid$ *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars t*) **and** *ground t*
shows $q \leq_b t^\bot$ **using** *assms*(*1*, *2*)
proof(*induct t arbitrary*: *q*)
case (*Fun f ts*)
from *Fun*(*2*) **obtain** *qs* **where** *wit*: $f\ qs \to q \mid\in\mid$ *nf-rules R* $\mathcal{F}$ *length qs* = *length ts*
$\forall\ i <$ *length ts*. *qs* ! *i* $\mid\in\mid$ *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars* (*ts* ! *i*))
by (*auto simp add*: *nf-ta-def*)
then have *BFun f qs* $\leq_b$ *Fun f* $ts^\bot$ **using** *Fun*(*1*)[*OF nth-mem*, *of i qs* !*i* **for** *i*] **using** *Fun*(*3*)
by *auto*
then show *?case* **using** *bless-eq-trans wit*(*1*) *bound-max-sound*[*of fset R*]
by (*auto simp*: *nf-rules-fmember*)
qed *auto*

lemma [*simp*]: *i* < *length ss* $\Longrightarrow$ *l* $\rhd$ *Fun f ss* $\Longrightarrow$ *l* $\rhd$ *ss* ! *i*
by (*meson nth-mem subterm.dual-order.strict-trans supt.arg*)

lemma *subt-less-eq-res-less-eq*:
assumes *ground*: *ground t* **and** *l* $\mid\in\mid$ *R* **and** *l* $\rhd$ *s* **and** $s^\bot \leq_b t^\bot$
and *q* $\mid\in\mid$ *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars t*)
shows $s^\bot \leq_b q$ **using** *assms*(*2*−)
proof (*induction t arbitrary*: *q s*)
case (*Var x*)
then show *?case* **using** *lift-total.anti-sym* **by** *fastforce*
next
case (*Fun f ts*) **note** *IN* = *this*
from *IN* **obtain** *qs* **where** *rule*: $f\ qs \to q \mid\in\mid$ *nf-rules R* $\mathcal{F}$ **and**
reach: *length qs* = *length ts* $\forall\ i <$ *length ts*. *qs* ! *i* $\mid\in\mid$ *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars* (*ts* ! *i*))
by (*auto simp add*: *nf-ta-def*)
have *q*: *lift-sup-small* (*BFun f qs*) (*psubt-lhs-bot* (*fset R*)) = *Some q*
using *nf-rule-to-bound-max*[*OF rule*]
using *lift-total.supremum-smaller-exists-unique*[*OF finite-lhs-subt*, *of fset R*

BFun f qs]
  by *simp* (*metis option.collapse option.distinct(1)*)
 have *subst*: $s^\bot \leq_b$ *BFun f qs* **using** *IN(1)*[*OF nth-mem*, *of i term.args s ! i qs ! i* **for** *i*] *IN(2−) reach*
  by (*cases s*) (*auto elim*!: *bless-eq.cases*)
 have $s^\bot \in$ *psubt-lhs-bot* (*fset R*) **using** *Fun(2 − 4)*
  by *auto*
then have *lift-total.lifted-less-eq* (*Some* ($s^\bot$)) (*lift-sup-small* (*BFun f qs*) (*psubt-lhs-bot* (*fset R*)))
  using *subst*
  by (*intro lift-total.lift-ord.supremum-sound*)
   (*auto simp*: *lift-total.lift-ord.smaller-subset-def*)
 then show *?case* **using** *subst q finite-lhs-subt*
  by *auto*
qed

lemma *ta-nf-sound1*:
 assumes *ground*: *ground t* **and** *lhs*: *l* |∈| *R* **and** *inst*: $l^\bot \leq_b t^\bot$
 shows *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars t*) = {||}
proof (*rule ccontr*)
 assume *ass*: *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars t*) ≠ {||}
 show *False* **proof** (*cases t*)
  case [*simp*]: (*Fun f ts*) **from** *ass*
  obtain *q qs* **where** *fin*: *q* |∈| *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars* (*Fun f ts*)) **and**
   rule: (*f qs* → *q*) |∈| *rules* (*nf-ta R* $\mathcal{F}$) *length qs* = *length ts* **and**
   reach: ∀ *i* < *length ts*. *qs* ! *i* |∈| *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars* (*ts* ! *i*))
   by (*auto simp add*: *nf-ta-def*) *blast*
  have $l^\bot \leq_b$ *BFun f qs* **using** *reach assms(1) inst rule(2)*
   using *subt-less-eq-res-less-eq*[*OF - lhs*, *of ts* ! *i term.args l* ! *i qs* ! *i* $\mathcal{F}$ **for** *i*]
    by (*cases l*) (*auto elim*!: *bless-eq.cases intro*!: *bless-eq.step*)
  then show *?thesis* **using** *lhs rule* **by** (*auto simp*: *nf-ta-def nf-rules-def*)
 qed (*metis ground ground.simps(1)*)
qed

lemma *ta-nf-tr-to-state*:
 assumes *ground t* **and** *q* |∈| *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars t*)
 shows *q* |∈| *fstates R* **using** *assms bound-max-state-fset*
 by (*cases t*) (*auto simp*: *states-def nf-ta-def nf-rules-def*)

lemma *ta-nf-sound2*:
 assumes *linear*: ∀ *l* |∈| *R*. *linear-term l*
  and *ground* (*t* :: ('*f*, '*v*) *term*) **and** *funas-term t* ⊆ *fset* $\mathcal{F}$
  and *NF*: ⋀ *l s*. *l* |∈| *R* ⟹ *t* ⊵ *s* ⟹ ¬ $l^\bot \leq_b s^\bot$
 shows ∃ *q*. *q* |∈| *ta-der* (*nf-ta R* $\mathcal{F}$) (*adapt-vars t*) **using** *assms(2 − 4)*
proof (*induct t*)
 case (*Fun f ts*)
 have *sub*: ⋀ *i*. *i* < *length ts* ⟹ (⋀*l s*. *l* |∈| *R* ⟹ *ts* ! *i* ⊵ *s* ⟹ ¬ $l^\bot \leq_b s^\bot$)
using *Fun(4) nth-mem* **by** *blast*
 from *Fun(1)*[*OF nth-mem*] *this Fun(2, 3)* **obtain** *qs* **where**

```
    reach: (∀ i < length ts. qs ! i |∈| ta-der (nf-ta R F) (adapt-vars (ts ! i))) and
len: length qs = length ts
   using Ex-list-of-length-P[of length ts λ x i. x |∈| (ta-der (nf-ta R F) (adapt-vars
(ts ! i)))]
    by auto (meson UN-subset-iff nth-mem)
  have nt-inst: ¬ (∃ s |∈| R. s⊥ ≤b BFun f qs)
  proof (rule ccontr, simp)
    assume ass: ∃ s |∈| R. s⊥ ≤b BFun f qs
    from term-instance-of-reach-state[of qs ! i R F ts ! i for i] reach Fun(2) len
    have BFun f qs ≤b Fun f ts⊥ by auto
    then show False using ass Fun(4) bless-eq-trans by blast
  qed
  obtain q where q = bound-max (BFun f qs) (psubt-lhs-bot (fset R)) by blast
  then have f qs → q |∈| rules (nf-ta R F) using Fun(2 − 4)
    using ta-nf-tr-to-state[of ts ! i qs ! i R F for i] len nt-inst reach
    by (auto simp: nf-ta-def nf-rules-fmember)
       (metis (no-types, lifting) in-fset-idx nth-mem)
  then show ?case using reach len by auto
qed auto

lemma ta-nf-lang-sound:
  assumes l |∈| R
  shows C⟨l · σ⟩ ∉ ta-lang (fstates R) (nf-ta R F)
proof (rule ccontr, simp del: ta-lang-to-gta-lang)
  assume ∗: C⟨l · σ⟩ ∈ ta-lang (fstates R) (nf-ta R F)
  then have cgr:ground (C⟨l·σ⟩) unfolding ta-lang-def by force
  then have gr: ground (l · σ) by simp
  then have l⊥ ≤b (l · σ)⊥ using instance-to-bless-eq by blast
  from ta-nf-sound1[OF gr assms(1) this] have res: ta-der (nf-ta R F) (adapt-vars
(l · σ)) = {||} .
  from ta-langE ∗ obtain q where q |∈| ta-der (nf-ta R F) (adapt-vars (C⟨l·σ⟩))
    by (metis adapt-vars-adapt-vars)
  with ta-der-ctxt-decompose[OF this[unfolded adapt-vars-ctxt]] res
  show False by blast
qed

lemma ta-nf-lang-complete:
  assumes linear: ∀ l |∈| R. linear-term l
      and ground: ground (t :: ('f, 'v) term) and sig: funas-term t ⊆ fset F
      and nf: ⋀C σ l. l |∈| R ⟹ C⟨l·σ⟩ ≠ t
    shows t ∈ ta-lang (fstates R) (nf-ta R F)
proof −
  from nf have ⋀ l s. l |∈| R ⟹ t ⊵ s ⟹ ¬ l⊥ ≤b s⊥
    using bless-eq-to-instance linear by blast
  from ta-nf-sound2[OF linear ground sig] this
  obtain q where q |∈| ta-der (nf-ta R F) (adapt-vars t) by blast
  from this ta-nf-tr-to-state[OF ground this] ground show ?thesis
    by (intro ta-langI) (auto simp add: nf-ta-def)
qed
```

lemma *ta-nf-$\mathcal{L}$-complete*:
assumes *linear*: $\forall\ l \mid\in\mid R.$ *linear-term* l
and *sig*: *funas-gterm* $t \subseteq$ *fset* $\mathcal{F}$
and *nf*: $\bigwedge C\ \sigma\ l.\ l \mid\in\mid R \Longrightarrow C\langle l{\cdot}\sigma\rangle \neq$ (*term-of-gterm* t)
shows $t \in \mathcal{L}$ (*nf-reg* $R\ \mathcal{F}$)
using *ta-nf-lang-complete*[*of* R *term-of-gterm* $t\ \mathcal{F}$] *assms*
by (*force simp*: $\mathcal{L}$-*def nf-reg-def funas-term-of-gterm-conv*)

lemma *nf-ta-funas*:
assumes *ground* t $q \mid\in\mid$ *ta-der* (*nf-ta* $R\ \mathcal{F}$) t
shows *funas-term* $t \subseteq$ *fset* $\mathcal{F}$ **using** *assms*
proof (*induct* t *arbitrary*: q)
case (*Fun* f *ts*)
from *Fun*(2−) **have** (f, *length ts*) $\mid\in\mid \mathcal{F}$
by (*auto simp*: *nf-ta-def nf-rules-def*)
then show *?case* **using** *Fun*
apply *simp*
by (*smt* (*verit*) *Union-least image-iff in-set-idx*)
qed *auto*

lemma *gta-lang-nf-ta-funas*:
assumes $t \in \mathcal{L}$ (*nf-reg* $R\ \mathcal{F}$)
shows *funas-gterm* $t \subseteq$ *fset* $\mathcal{F}$ **using** *assms nf-ta-funas*[*of term-of-gterm* t - R $\mathcal{F}$]
unfolding *nf-reg-def* $\mathcal{L}$-*def*
by (*auto simp*: *funas-term-of-gterm-conv*)

end
theory *Tree-Automata-Derivation-Split*
imports *Regular-Tree-Relations.Tree-Automata*
Ground-MCtxt
begin

lemma *ta-der′-inf-mctxt*:
assumes $t \mid\in\mid$ *ta-der′* $\mathcal{A}\ s$
shows *fst* (*split-vars* t) $\leq$ (*mctxt-of-term* s) **using** *assms*
proof (*induct* s *arbitrary*: t)
case (*Fun* f *ts*) **then show** *?case*
by (*cases* t) (*auto simp*: *comp-def less-eq-mctxt-prime intro*: *less-eq-mctxt′.intros*)
qed (*auto simp*: *ta-der′.simps*)

lemma *ta-der′-poss-subt-at-ta-der′*:
assumes $t \mid\in\mid$ *ta-der′* $\mathcal{A}\ s$ **and** $p \in$ *poss* t
shows t |- $p \mid\in\mid$ *ta-der′* $\mathcal{A}$ (s |- p) **using** *assms*
by (*induct* s *arbitrary*: $t\ p$) (*auto simp*: *ta-der′.simps*, *blast*+)

lemma *ta-der′-varposs-to-ta-der*:
assumes $t \mid\in\mid$ *ta-der′* $\mathcal{A}\ s$ **and** $p \in$ *varposs* t

shows *the-Var* $(t \mid\!-\, p) \mid\!\in\!\mid$ *ta-der* $\mathcal{A}$ $(s \mid\!-\, p)$ **using** *assms*
by (*induct s arbitrary*: *t p*) (*auto simp*: *ta-der'.simps*, *blast*+)

definition *ta-der'-target-mctxt t* ≡ *fst* (*split-vars t*)
definition *ta-der'-target-args t* ≡ *snd* (*split-vars t*)
definition *ta-der'-source-args t s* ≡ *unfill-holes* (*fst* (*split-vars t*)) *s*

lemmas *ta-der'-mctxt-simps* = *ta-der'-target-mctxt-def ta-der'-target-args-def ta-der'-source-args-def*

lemma *ta-der'-target-mctxt-funas* [*simp*]:
funas-mctxt (*ta-der'-target-mctxt u*) = *funas-term u*
by (*auto simp*: *ta-der'-target-mctxt-def*)

lemma *ta-der'-target-mctxt-ground* [*simp*]:
ground-mctxt (*ta-der'-target-mctxt t*)
by (*auto simp*: *ta-der'-target-mctxt-def*)

lemma *ta-der'-source-args-ground*:
t $\mid\!\in\!\mid$ *ta-der'* $\mathcal{A}$ *s* ⟹ *ground s* ⟹ ∀ *u* ∈ *set* (*ta-der'-source-args t s*). *ground u*
by (*metis fill-unfill-holes ground-fill-holes length-unfill-holes ta-der'-inf-mctxt ta-der'-mctxt-simps*)

lemma *ta-der'-source-args-term-of-gterm*:
t $\mid\!\in\!\mid$ *ta-der'* $\mathcal{A}$ (*term-of-gterm s*) ⟹ ∀ *u* ∈ *set* (*ta-der'-source-args t* (*term-of-gterm s*)). *ground u*
by (*intro ta-der'-source-args-ground*) *auto*

lemma *ta-der'-source-args-length*:
t $\mid\!\in\!\mid$ *ta-der'* $\mathcal{A}$ *s* ⟹ *num-holes* (*ta-der'-target-mctxt t*) = *length* (*ta-der'-source-args t s*)
by (*auto simp*: *ta-der'-mctxt-simps ta-der'-inf-mctxt*)

lemma *ta-der'-target-args-length*:
num-holes (*ta-der'-target-mctxt t*) = *length* (*ta-der'-target-args t*)
by (*auto simp*: *ta-der'-mctxt-simps split-vars-num-holes*)

lemma *ta-der'-target-args-vars-term-conv*:
vars-term t = *set* (*ta-der'-target-args t*)
by (*auto simp*: *ta-der'-target-args-def split-vars-vars-term-list*)

lemma *ta-der'-target-args-vars-term-list-conv*:
ta-der'-target-args t = *vars-term-list t*
by (*auto simp*: *ta-der'-target-args-def split-vars-vars-term-list*)

lemma *mctxt-args-ta-der'*:
assumes *num-holes C* = *length qs num-holes C* = *length ss*
and ∀ *i* < *length ss*. *qs* ! *i* $\mid\!\in\!\mid$ *ta-der* $\mathcal{A}$ (*ss* ! *i*)
shows (*fill-holes C* (*map Var qs*)) $\mid\!\in\!\mid$ *ta-der'* $\mathcal{A}$ (*fill-holes C ss*) **using** *assms*
proof (*induct rule*: *fill-holes-induct2*)

case *MHole* **then show** *?case*
by (*cases ss*; *cases qs*) (*auto simp*: *ta-der-to-ta-der′*)
next
case (*MFun f ts*) **then show** *?case*
by (*simp add*: *partition-by-nth-nth*(*1*, *2*))
qed *auto*

— Splitting derivation into multihole context containing the remaining function symbols and the states, where each state is reached via the automata
lemma *ta-der′-mctxt-structure*:
assumes *t* |∈| *ta-der′ A s*
shows *t* = *fill-holes* (*ta-der′-target-mctxt t*) (*map Var* (*ta-der′-target-args t*)) (**is** *?G1*)
s = *fill-holes* (*ta-der′-target-mctxt t*) (*ta-der′-source-args t s*) (**is** *?G2*)
num-holes (*ta-der′-target-mctxt t*) = *length* (*ta-der′-source-args t s*) ∧
length (*ta-der′-source-args t s*) = *length* (*ta-der′-target-args t*) (**is** *?G3*)
i < *length* (*ta-der′-source-args t s*) ⟹ *ta-der′-target-args t* ! *i* |∈| *ta-der A* (*ta-der′-source-args t s* ! *i*)
proof −
let *?C* = *ta-der′-target-mctxt t* **let** *?ss* = *ta-der′-source-args t s*
let *?qs* = *ta-der′-target-args t*
have *t-split*: *?G1* **by** (*auto simp*: *ta-der′-mctxt-simps split-vars-fill-holes*)
have *s-split*: *?G2* **by** (*auto simp*: *ta-der′-mctxt-simps ta-der′-inf-mctxt*[*OF assms*]
intro!: *fill-unfill-holes*[*symmetric*])
have *len*: *num-holes ?C* = *length ?ss length ?ss* = *length ?qs* **using** *assms*
by (*auto simp*: *ta-der′-mctxt-simps split-vars-num-holes ta-der′-inf-mctxt*)
have *i* < *length* (*ta-der′-target-args t*) ⟹
ta-der′-target-args t ! *i* |∈| *ta-der A* (*ta-der′-source-args t s* ! *i*) **for** *i*
using *ta-der′-poss-subt-at-ta-der′*[*OF assms*, *of varposs-list t* ! *i*]
unfolding *ta-der′-mctxt-simps split-vars-vars-term-list length-map*
by (*auto simp*: *unfill-holes-to-subst-at-hole-poss*[*OF ta-der′-inf-mctxt*[*OF assms*]]
simp flip: *varposs-list-to-var-term-list*[*of i t*, *unfolded varposs-list-var-terms-length*])
(*metis assms hole-poss-split-vars-varposs-list nth-map nth-mem*
ta-der′-varposs-to-ta-der ta-der-to-ta-der′ varposs-eq-varposs-list varposs-list-var-terms-length)
then show *?G1 ?G2 ?G3 i* < *length* (*ta-der′-source-args t s*) ⟹
ta-der′-target-args t ! *i* |∈| *ta-der A* (*ta-der′-source-args t s* ! *i*) **using** *len t-split s-split*
by (*simp-all add*: *ta-der′-mctxt-simps*)
qed

lemma *ta-der′-ground-mctxt-structure*:
assumes *t* |∈| *ta-der′ A* (*term-of-gterm s*)
shows *t* = *fill-holes* (*ta-der′-target-mctxt t*) (*map Var* (*ta-der′-target-args t*))
term-of-gterm s = *fill-holes* (*ta-der′-target-mctxt t*) (*ta-der′-source-args t* (*term-of-gterm s*))
num-holes (*ta-der′-target-mctxt t*) = *length* (*ta-der′-source-args t* (*term-of-gterm s*)) ∧
length (*ta-der′-source-args t* (*term-of-gterm s*)) = *length* (*ta-der′-target-args t*)
i < *length* (*ta-der′-target-args t*) ⟹ *ta-der′-target-args t* ! *i* |∈| *ta-der A*

(*ta-der′-source-args t* (*term-of-gterm s*) ! *i*)
using *ta-der′-mctxt-structure*[*OF assms*]
by *force*+

— Splitting derivation into context containing the remaining function symbols and state

definition *ta-der′-gctxt t* ≡ *gctxt-of-gmctxt* (*gmctxt-of-mctxt* (*fst* (*split-vars t*)))
abbreviation *ta-der′-ctxt t* ≡ *ctxt-of-gctxt* (*ta-der′-gctxt t*)
definition *ta-der′-source-ctxt-arg t s* ≡ *hd* (*unfill-holes* (*fst* (*split-vars t*)) *s*)

abbreviation *ta-der′-source-gctxt-arg t s* ≡ *gterm-of-term* (*ta-der′-source-ctxt-arg t* (*term-of-gterm s*))

lemma *ta-der′-ctxt-structure*:
assumes *t* |∈| *ta-der′ A s vars-term-list t* = [*q*]
shows *t* = (*ta-der′-ctxt t*)⟨*Var q*⟩ (**is** *?G1*)
s = (*ta-der′-ctxt t*)⟨*ta-der′-source-ctxt-arg t s*⟩ (**is** *?G2*)
ground-ctxt (*ta-der′-ctxt t*) (**is** *?G3*)
q |∈| *ta-der A* (*ta-der′-source-ctxt-arg t s*) (**is** *?G4*)
proof −
have ∗: *length xs* = *Suc 0* ⟹ *xs* = [*hd xs*] **for** *xs*
by (*metis length-0-conv length-Suc-conv list.sel*(*1*))
have [*simp*]: *length* (*snd* (*split-vars t*)) = *Suc 0* **using** *assms*(*2*) *ta-der′-inf-mctxt*[*OF assms*(*1*)]
by (*auto simp*: *split-vars-vars-term-list*)
have [*simp*]: *num-gholes* (*gmctxt-of-mctxt* (*fst* (*split-vars t*))) = *Suc 0* **using** *assms*(*2*)
by (*simp add*: *split-vars-num-holes split-vars-vars-term-list*)
have [*simp*]: *ta-der′-source-args t s* = [*ta-der′-source-ctxt-arg t s*]
using *assms*(*2*) *ta-der′-inf-mctxt*[*OF assms*(*1*)]
by (*auto simp*: *ta-der′-source-args-def ta-der′-source-ctxt-arg-def split-vars-num-holes intro*!: ∗)
have *t-split*: *?G1* **using** *assms*(*2*)
by (*auto simp*: *ta-der′-gctxt-def split-vars-fill-holes*
split-vars-vars-term-list simp flip: *ctxt-of-gctxt-gctxt-of-gmctxt-apply*)
have *s-split*: *?G2* **using** *ta-der′-mctxt-structure*[*OF assms*(*1*)] *assms*(*2*)
by (*auto simp*: *ta-der′-gctxt-def ta-der′-target-mctxt-def*
simp flip: *ctxt-of-gctxt-gctxt-of-gmctxt-apply*)
from *ta-der′-mctxt-structure*[*OF assms*(*1*)] **have** *?G4*
by (*auto simp*: *ta-der′-target-args-def assms*(*2*) *split-vars-vars-term-list*)
moreover have *?G3* **unfolding** *ta-der′-gctxt-def* **by** *auto*
ultimately show *?G1 ?G2 ?G3 ?G4* **using** *t-split s-split*
by *force*+
qed

lemma *ta-der′-ground-ctxt-structure*:

assumes t $|\in|$ *ta-der'* $\mathcal{A}$ (*term-of-gterm* s) *vars-term-list* $t = [q]$
shows $t = (\textit{ta-der'-ctxt}\ t)\langle \textit{Var}\ q\rangle$
$s = (\textit{ta-der'-gctxt}\ t)\langle \textit{ta-der'-source-gctxt-arg}\ t\ s\rangle_G$
ground (*ta-der'-source-ctxt-arg* t (*term-of-gterm* s))
q $|\in|$ *ta-der* $\mathcal{A}$ (*ta-der'-source-ctxt-arg* t (*term-of-gterm* s))
using *ta-der'-ctxt-structure*[*OF assms*] *term-of-gterm-ctxt-apply*
by *force+*

4.4 Sufficient condition for splitting the reachability relation induced by a tree automaton

locale *derivation-split* =
fixes A :: ('q, 'f) *ta* **and** $\mathcal{A}$ **and** $\mathcal{B}$
assumes *rule-split*: *rules* A = *rules* $\mathcal{A}$ $|\cup|$ *rules* $\mathcal{B}$
and *eps-split*: *eps* A = *eps* $\mathcal{A}$ $|\cup|$ *eps* $\mathcal{B}$
and *B-target-states*: *rule-target-states* (*rules* $\mathcal{B}$) $|\cup|$ (*snd* $|`|$ (*eps* $\mathcal{B}$)) $|\cap|$
(*rule-arg-states* (*rules* $\mathcal{A}$) $|\cup|$ (*fst* $|`|$ (*eps* $\mathcal{A}$))) = $\{||\}$
begin

abbreviation $\Delta_A \equiv$ *rules* $\mathcal{A}$
abbreviation $\Delta_{\mathcal{E}\,A} \equiv$ *eps* $\mathcal{A}$
abbreviation $\Delta_B \equiv$ *rules* $\mathcal{B}$
abbreviation $\Delta_{\mathcal{E}\,B} \equiv$ *eps* $\mathcal{B}$

abbreviation $\mathcal{Q}_A \equiv \mathcal{Q}\ \mathcal{A}$
definition $\mathcal{Q}_B \equiv$ *rule-target-states* Δ_B $|\cup|$ (*snd* $|`|$ $\Delta_{\mathcal{E}\,B}$)
lemmas *B-target-states'* = *B-target-states*[*folded* $\mathcal{Q}_B$*-def*]

lemma *states-split* [*simp*]: $\mathcal{Q}\ A = \mathcal{Q}\ \mathcal{A}$ $|\cup|$ $\mathcal{Q}\ \mathcal{B}$
by (*auto simp add*: $\mathcal{Q}$*-def rule-split eps-split*)

lemma *A-args-states-not-B*:
TA-rule f qs q $|\in|$ $\Delta_A \Longrightarrow p$ $|\in|$ *fset-of-list qs* $\Longrightarrow p$ $|\notin|$ $\mathcal{Q}_B$
using *B-target-states*
by (*force simp add*: $\mathcal{Q}_B$*-def*)

lemma *rule-statesD*:
r $|\in|$ $\Delta_A \Longrightarrow$ *r-rhs* r $|\in|$ $\mathcal{Q}_A$
r $|\in|$ $\Delta_B \Longrightarrow$ *r-rhs* r $|\in|$ $\mathcal{Q}_B$
r $|\in|$ $\Delta_A \Longrightarrow p$ $|\in|$ *fset-of-list* (*r-lhs-states* r) $\Longrightarrow p$ $|\in|$ $\mathcal{Q}_A$
TA-rule f qs q $|\in|$ $\Delta_A \Longrightarrow q$ $|\in|$ $\mathcal{Q}_A$
TA-rule f qs q $|\in|$ $\Delta_B \Longrightarrow q$ $|\in|$ $\mathcal{Q}_B$
TA-rule f qs q $|\in|$ $\Delta_A \Longrightarrow p$ $|\in|$ *fset-of-list qs* $\Longrightarrow p$ $|\in|$ $\mathcal{Q}_A$
by (*auto simp*: *rule-statesD* $\mathcal{Q}_B$*-def rev-image-eqI*)

lemma *eps-states-dest*:
(p, q) $|\in|$ $\Delta_{\mathcal{E}\,A} \Longrightarrow p$ $|\in|$ $\mathcal{Q}_A$
(p, q) $|\in|$ $\Delta_{\mathcal{E}\,A} \Longrightarrow q$ $|\in|$ $\mathcal{Q}_A$
(p, q) $|\in|$ $\Delta_{\mathcal{E}\,A}|^+| \Longrightarrow p$ $|\in|$ $\mathcal{Q}_A$

$(p,\ q)\ |\in|\ \Delta_{\mathcal{E}\,A}|^+|\Longrightarrow q\ |\in|\ \mathcal{Q}_A$
$(p,\ q)\ |\in|\ \Delta_{\mathcal{E}\,B}\Longrightarrow q\ |\in|\ \mathcal{Q}_B$
$(p,\ q)\ |\in|\ \Delta_{\mathcal{E}\,B}|^+|\Longrightarrow q\ |\in|\ \mathcal{Q}_B$
by (*auto simp*: *eps-dest-all* $\mathcal{Q}_B$*-def rev-image-eqI elim*: *ftranclE*)

lemma *transcl-eps-simp*:
$(eps\ A)|^+|=\Delta_{\mathcal{E}\,A}|^+|\ |\cup|\ \Delta_{\mathcal{E}\,B}|^+|\ |\cup|\ (\Delta_{\mathcal{E}\,A}|^+|\ |O|\ \Delta_{\mathcal{E}\,B}|^+|)$
proof −
have $\Delta_{\mathcal{E}\,B}\ |O|\ \Delta_{\mathcal{E}\,A}=\{||\}$ **using** *B-target-states′*
by (*metis eps-states-dest*(*5*) *ex-fin-conv fimageI finterI frelcompE fst-conv inf-sup-distrib1 sup-eq-bot-iff*)
from *ftrancl-Un2-separatorE*[*OF this*] **show** *?thesis*
unfolding *eps-split* **by** *auto*
qed

lemma *B-rule-eps-A-False*:
$f\ qs\to q\ |\in|\ \Delta_B\Longrightarrow(q,\ p)\ |\in|\ \Delta_{\mathcal{E}\,A}|^+|\Longrightarrow False$
using *B-target-states* **unfolding** $\mathcal{Q}_B$*-def*
by (*metis B-target-states′ equalsffemptyD fimage-eqI finter-iff fst-conv ftranclD funion-iff local.rule-statesD*(*5*))

lemma *to-A-rule-set*:
assumes *TA-rule f qs q* $|\in|$ *rules A* **and** $q=p\vee(q,\ p)\ |\in|\ (eps\ A)|^+|$ **and** $p\ |\notin|\ \mathcal{Q}_B$
shows *TA-rule f qs q* $|\in|\ \Delta_A\ q=p\vee(q,\ p)\ |\in|\ \Delta_{\mathcal{E}\,A}|^+|$ **using** *assms*
unfolding *transcl-eps-simp rule-split*
by (*auto dest*: *rule-statesD eps-states-dest dest*: *B-rule-eps-A-False*)

lemma *to-B-rule-set*:
assumes *TA-rule f qs q* $|\in|$ *rules A* **and** $q\ |\notin|\ \mathcal{Q}_A$
shows *TA-rule f qs q* $|\in|\ \Delta_B$ **using** *assms*
unfolding *transcl-eps-simp rule-split*
by (*auto dest*: *rule-statesD eps-states-dest*)

declare *fsubsetI*[*rule del*]
lemma *ta-der-monos*:
ta-der $\mathcal{A}$ *t* $|\subseteq|$ *ta-der A t ta-der* $\mathcal{B}$ *t* $|\subseteq|$ *ta-der A t*
by (*auto simp*: *sup.coboundedI1 rule-split eps-split intro*!: *ta-der-mono*)
declare *fsubsetI*[*intro*!]

lemma *ta-der-from-*Δ_A:
assumes *q* $|\in|$ *ta-der A* (*term-of-gterm t*) **and** $q\ |\notin|\ \mathcal{Q}_B$
shows *q* $|\in|$ *ta-der* $\mathcal{A}$ (*term-of-gterm t*) **using** *assms*
proof (*induct rule*: *ta-der-gterm-induct*)
case (*GFun f ts ps p q*)
have $i<length\ ts\Longrightarrow ps\ !\ i\ |\notin|\ \mathcal{Q}_B$ **for** i **using** *GFun A-args-states-not-B*
by (*metis fnth-mem to-A-rule-set*(*1*))

```
  then show ?case using GFun(2, 5) to-A-rule-set[OF GFun(1, 3, 6)]
    by (auto simp: transcl-eps-simp)
qed

lemma ta-state:
  assumes q |∈| ta-der A (term-of-gterm s)
  shows q |∈| 𝒬_A ∨ q |∈| 𝒬_B using assms
 by (cases s) (auto simp: rule-split transcl-eps-simp dest: rule-statesD eps-states-dest)



lemma ta-der-split:
  assumes q |∈| ta-der A (term-of-gterm s) and q |∈| 𝒬_B
  shows ∃ t. t |∈| ta-der' 𝒜 (term-of-gterm s) ∧ q |∈| ta-der ℬ t
    (is ∃ t . ?P s q t) using assms
proof (induct rule: ta-der-gterm-induct)
  case (GFun f ts ps p q)
  {fix i assume ass: i < length ts
    then have ∃ t. t |∈| ta-der' 𝒜 (term-of-gterm (ts ! i)) ∧ ps ! i |∈| ta-der ℬ t
    proof (cases ps ! i |∉| 𝒬_B)
      case True then show ?thesis
        using ta-state GFun(2, 4) ta-der-from-Δ_A[of ps ! i ts ! i] ass
        by (intro exI[of - Var (ps ! i)]) (auto simp: ta-der-to-ta-der' 𝒬_B-def)
    next
      case False
      then have ps ! i |∈| 𝒬_B using ta-state[OF GFun(4)[OF ass]]
        by auto
      from GFun(5)[OF ass this] show ?thesis .
    qed}
  then obtain h where IH:
    ∀ i < length ts. h i |∈| ta-der' 𝒜 (term-of-gterm (ts ! i))
    ∀ i < length ts. ps ! i |∈| ta-der ℬ (h i)
    using GFun(1 − 4) choice-nat[of length ts λ t i. ?P (ts ! i) (ps ! i) t]
    by blast
  from GFun(1) consider (A) f ps → p |∈| Δ_A | (B) f ps → p |∈| Δ_B by (auto
simp: rule-split)
  then show ?case
  proof cases
    case A then obtain q' where eps-sp: p = q' ∨ (p, q') |∈| Δ_{ℰ A}|^+|
      q' = q ∨ (q', q) |∈| Δ_{ℰ B}|^+| using GFun(3, 6)
      by (auto simp: transcl-eps-simp dest: eps-states-dest)
    from GFun(4)[THEN ta-der-from-Δ_A] A GFun(2, 4)
    have reach-fst: p |∈| ta-der 𝒜 (term-of-gterm (GFun f ts))
      using A-args-states-not-B by auto
    then have q' |∈| ta-der 𝒜 (term-of-gterm (GFun f ts)) using eps-sp
      by (meson ta-der-trancl-eps)
    then show ?thesis using eps-sp(2)
     by (intro exI[of - Var q']) (auto simp flip: ta-der-to-ta-der' simp del: ta-der'-simps)
  next
```

```
    case B
    then have p = q ∨ (p, q) |∈| Δ_{ℰ B}|^+| using GFun(3)
      by (auto simp: transcl-eps-simp dest: B-rule-eps-A-False)
    then show ?thesis using GFun(2, 4, 6) IH B
      by (auto intro!: exI[of - Fun f (map h [0 ..< length ts])] exI[of - ps])
  qed
qed


lemma ta-der'-split:
  assumes t |∈| ta-der' A (term-of-gterm s)
  shows ∃ u. u |∈| ta-der' 𝒜 (term-of-gterm s) ∧ t |∈| ta-der' ℬ u
    (is ∃ u. ?P s t u) using assms
proof (induct s arbitrary: t)
  case (GFun f ts) show ?case
  proof (cases t)
    case [simp]: (Var q)
    have q |∈| ta-der A (term-of-gterm (GFun f ts)) using GFun(2)
      by (auto simp flip: ta-der-to-ta-der')
    from ta-der-split[OF this] ta-der-from-Δ_A[OF this] ta-state[OF this]
    show ?thesis unfolding Var
      by (metis ta-der'-refl ta-der-to-ta-der')
  next
    case [simp]: (Fun g ss)
    obtain h where IH:
      ∀ i < length ts. h i |∈| ta-der' 𝒜 (term-of-gterm (ts ! i))
      ∀ i < length ts. ss ! i |∈| ta-der' ℬ (h i)
      using GFun choice-nat[of length ts λ t i. ?P (ts ! i) (ss ! i) t]
      by auto
    then show ?thesis using GFun(2)
      by (auto intro!: exI[of - Fun f (map h [0..<length ts])])
  qed
qed


lemma ta-der-to-mcxtx:
  assumes q |∈| ta-der A (term-of-gterm s) and q |∈| 𝒬_B
  shows ∃ C ss qs. length qs = length ss ∧ num-holes C = length ss ∧
    (∀ i < length ss. qs ! i |∈| ta-der 𝒜 (term-of-gterm (ss ! i))) ∧
    q |∈| ta-der ℬ (fill-holes C (map Var qs)) ∧
    ground-mctxt C ∧ fill-holes C (map term-of-gterm ss) = term-of-gterm s
    (is ∃ C ss qs. ?P s q C ss qs)
proof −
  from ta-der-split[OF assms] obtain t where
    wit: t |∈| ta-der' 𝒜 (term-of-gterm s) q |∈| ta-der ℬ t by auto
  let ?C = fst (split-vars t) let ?ss = map (gsubt-at s) (varposs-list t)
  let ?qs = snd (split-vars t)
  have poss [simp]:i < length (varposs-list t) ⟹ varposs-list t ! i ∈ gposs s for i
    by (metis nth-mem ta-der'-poss[OF wit(1)] poss-gposs-conv subset-eq var-
```

poss-eq-varposs-list
varposs-imp-poss varposs-list-var-terms-length)
have *len*: *num-holes ?C* = *length ?ss length ?ss* = *length ?qs*
by (*simp-all add*: *split-vars-num-holes split-vars-vars-term-list varposs-list-var-terms-length*)
from *unfill-holes-to-subst-at-hole-poss*[*OF ta-der'-inf-mctxt*[*OF wit*(*1*)]]
have *unfill-holes* (*fst* (*split-vars t*)) (*term-of-gterm s*) = *map* (*term-of-gterm* ∘ *gsubt-at s*) (*varposs-list t*)
by (*auto simp*: *comp-def hole-poss-split-vars-varposs-list*
dest: *in-set-idx intro*!: *nth-equalityI term-of-gterm-gsubt*)
from *fill-unfill-holes*[*OF ta-der'-inf-mctxt*[*OF wit*(*1*)]] *this*
have *rep*: *fill-holes ?C* (*map term-of-gterm ?ss*) = *term-of-gterm s*
by *simp*
have *reach-int*: *i* < *length ?ss* ⟹ *?qs* ! *i* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm* (*?ss* ! *i*))
for *i*
using *wit*(*1*) *ta-der'-varposs-to-ta-der*
unfolding *split-vars-vars-term-list length-map*
unfolding *varposs-list-to-var-term-list*[*symmetric*]
by (*metis nth-map nth-mem poss term-of-gterm-gsubt varposs-eq-varposs-list*)
have *reach-end*: *q* |∈| *ta-der* $\mathcal{B}$ (*fill-holes ?C* (*map Var ?qs*)) **using** *wit*
using *split-vars-fill-holes*[*of ?C t map Var ?qs*]
by *auto*
show *?thesis* **using** *len rep reach-end reach-int*
by (*metis split-vars-ground'*)
qed

lemma *ta-der-to-gmcxtx*:
assumes *q* |∈| *ta-der A* (*term-of-gterm s*) **and** *q* |∈| $\mathcal{Q}_B$
shows ∃ *C ss qs qs'*. *length qs'* = *length qs* ∧ *length qs* = *length ss* ∧ *num-gholes C* = *length ss* ∧
(∀ *i* < *length ss*. *qs* ! *i* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm* (*ss* ! *i*))) ∧
q |∈| *ta-der* $\mathcal{B}$ (*fill-holes* (*mctxt-of-gmctxt C*) (*map Var qs'*)) ∧
fill-gholes C ss = *s*
using *ta-der-to-mcxtx*[*OF assms*]
by (*metis gmctxt-of-mctxt-inv ground-gmctxt-of-gterm-of-term num-gholes-gmctxt-of-mctxt term-of-gterm-inv*)

lemma *mctxt-const-to-ta-der*:
assumes *num-holes C* = *length ss length ss* = *length qs*
and ∀ *i* < *length qs*. *qs* ! *i* |∈| *ta-der* $\mathcal{A}$ (*ss* ! *i*)
and *q* |∈| *ta-der* $\mathcal{B}$ (*fill-holes C* (*map Var qs*))
shows *q* |∈| *ta-der A* (*fill-holes C ss*)
proof −
have *mid*: *fill-holes C* (*map Var qs*) |∈| *ta-der' A* (*fill-holes C ss*)
using *assms*(*1* − *3*) *ta-der-monos*(*1*)
by (*intro mctxt-args-ta-der'*) *auto*
then show *?thesis* **using** *assms*(*1*, *2*) *ta-der-monos*(*2*)[*THEN fsubsetD*, *OF assms*(*4*)]

using *ta-der'-trans*
by (*simp add*: *ta-der'-ta-der*)
qed

lemma *ctxt-const-to-ta-der*:
assumes $q \mid\in\mid$ *ta-der* $\mathcal{A}$ *s*
and $p \mid\in\mid$ *ta-der* $\mathcal{B}$ $C\langle Var\ q\rangle$
shows $p \mid\in\mid$ *ta-der A* $C\langle s\rangle$ **using** *assms*
by (*meson fin-mono ta-der-ctxt ta-der-monos*(*1*) *ta-der-monos*(*2*))

lemma *gctxt-const-to-ta-der*:
assumes $q \mid\in\mid$ *ta-der* $\mathcal{A}$ (*term-of-gterm s*)
and $p \mid\in\mid$ *ta-der* $\mathcal{B}$ (*ctxt-of-gctxt C*)$\langle Var\ q\rangle$
shows $p \mid\in\mid$ *ta-der A* (*term-of-gterm* $C\langle s\rangle_G$) **using** *assms*
by (*metis ctxt-const-to-ta-der ctxt-of-gctxt-inv ground-ctxt-of-gctxt ground-gctxt-of-ctxt-apply-gterm*)

end
end

5 (Multihole)Context closure of recognized tree languages

theory *TA-Clousure-Const*
imports *Tree-Automata-Derivation-Split*
begin

5.1 Tree Automata closure constructions

declare *ta-union-def* [*simp*]

5.1.1 Reflexive closure over a given signature

definition *reflcl-rules* $\mathcal{F}$ *q* $\equiv$ (λ (*f*, *n*). *TA-rule f* (*replicate n q*) *q*) $\mid`\mid$ $\mathcal{F}$
definition *refl-ta* $\mathcal{F}$ *q* = *TA* (*reflcl-rules* $\mathcal{F}$ *q*) {||}

definition *gen-reflcl-automaton* :: ('*f* × *nat*) *fset* ⇒ ('*q*, '*f*) *ta* ⇒ '*q* ⇒ ('*q*, '*f*) *ta*
where
gen-reflcl-automaton $\mathcal{F}$ $\mathcal{A}$ *q* = *ta-union* $\mathcal{A}$ (*refl-ta* $\mathcal{F}$ *q*)

definition *reflcl-automaton* $\mathcal{F}$ $\mathcal{A}$ = (*let* $\mathcal{B}$ = *fmap-states-ta Some* $\mathcal{A}$ *in*
gen-reflcl-automaton $\mathcal{F}$ $\mathcal{B}$ *None*)

definition *reflcl-reg* $\mathcal{F}$ $\mathcal{A}$ = *Reg* (*finsert None* (*Some* $\mid`\mid$ *fin* $\mathcal{A}$)) (*reflcl-automaton* $\mathcal{F}$ (*ta* $\mathcal{A}$))

5.1.2 Multihole context closure over a given signature

definition *refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q* = *TA* (*reflcl-rules* $\mathcal{F}$ *q*) ((λ *p*. (*p*, *q*)) $\mid`\mid$ (*Q* $\mid\cap\mid$ $\mathcal{Q}$ $\mathcal{A}$))

definition *gen-parallel-closure-automaton* :: *'q fset* ⇒ (*'f* × *nat*) *fset* ⇒ (*'q*, *'f*) *ta* ⇒ *'q* ⇒ (*'q*, *'f*) *ta* **where**
gen-parallel-closure-automaton Q F A q = *ta-union A* (*refl-over-states-ta Q F A q*)

definition *parallel-closure-reg* **where**
parallel-closure-reg F A = (*let B* = *fmap-states-reg Some A in*
Reg {|*None*|} (*gen-parallel-closure-automaton* (*fin B*) *F* (*ta B*) *None*))

5.1.3 Context closure of regular tree language

definition *semantic-path-rules F q_c q_i q_f* ≡
|⋃| ((λ (*f*, *n*). *fset-of-list* (*map* (λ *i*. *TA-rule f* ((*replicate n q_c*)[*i* := *q_i*]) *q_f*) [*0*..< *n*])) |‘| *F*)

definition *reflcl-over-single-ta Q F q_c q_f* ≡
TA (*reflcl-rules F q_c* |∪| *semantic-path-rules F q_c q_f q_f*) ((λ *p*. (*p*, *q_f*)) |‘| *Q*)

definition *gen-ctxt-closure-automaton Q F A q_c q_f* = *ta-union A* (*reflcl-over-single-ta Q F q_c q_f*)

definition *gen-ctxt-closure-reg F A q_c q_f* =
Reg {|*q_f*|} (*gen-ctxt-closure-automaton* (*fin A*) *F* (*ta A*) *q_c q_f*)

definition *ctxt-closure-reg F A* =
(*let B* = *fmap-states-reg Inl* (*reg-Restr-Q_f A*) *in*
gen-ctxt-closure-reg F B (*Inr False*) (*Inr True*))

5.1.4 Not empty context closure of regular tree language

datatype *cl-states* = *cl-state* | *tr-state* | *fin-state* | *fin-clstate*

definition *reflcl-over-nhole-ctxt-ta Q F q_c q_i q_f* ≡
TA (*reflcl-rules F q_c* |∪| *semantic-path-rules F q_c q_i q_f* |∪| *semantic-path-rules F q_c q_f q_f*) ((λ *p*. (*p*, *q_i*)) |‘| *Q*)

definition *gen-nhole-ctxt-closure-automaton Q F A q_c q_i q_f* =
ta-union A (*reflcl-over-nhole-ctxt-ta Q F q_c q_i q_f*)

definition *gen-nhole-ctxt-closure-reg F A q_c q_i q_f* =
Reg {|*q_f*|} (*gen-nhole-ctxt-closure-automaton* (*fin A*) *F* (*ta A*) *q_c q_i q_f*)

definition *nhole-ctxt-closure-reg F A* =
(*let B* = *fmap-states-reg Inl* (*reg-Restr-Q_f A*) *in*
(*gen-nhole-ctxt-closure-reg F B* (*Inr cl-state*) (*Inr tr-state*) (*Inr fin-state*)))

5.1.5 Non empty multihole context closure of regular tree language

abbreviation *add-eps* $\mathcal{A}$ $e \equiv$ *TA* (*rules* $\mathcal{A}$) (*eps* $\mathcal{A}$ |∪| e)
definition *reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i $q_f \equiv$
add-eps (*reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) {|(q_i, q_c)|}

definition *gen-nhole-mctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f =
ta-union $\mathcal{A}$ (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f)

definition *gen-nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f =
Reg {|q_f|} (*gen-nhole-mctxt-closure-automaton* (*fin* $\mathcal{A}$) $\mathcal{F}$ (*ta* $\mathcal{A}$) q_c q_i q_f)

definition *nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *fmap-states-reg Inl* (*reg-Restr-*Q_f $\mathcal{A}$) *in*
(*gen-nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{B}$ (*Inr cl-state*) (*Inr tr-state*) (*Inr fin-state*)))

5.1.6 Not empty multihole context closure of regular tree language

definition *gen-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f =
Reg {|q_f, q_i|} (*gen-nhole-mctxt-closure-automaton* (*fin* $\mathcal{A}$) $\mathcal{F}$ (*ta* $\mathcal{A}$) q_c q_i q_f)

definition *mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *fmap-states-reg Inl* (*reg-Restr-*Q_f $\mathcal{A}$) *in*
(*gen-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{B}$ (*Inr cl-state*) (*Inr tr-state*) (*Inr fin-state*)))

5.1.7 Multihole context closure of regular tree language

definition *nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$ =
reg-union (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) (*Reg* {|*fin-clstate*|} (*refl-ta* $\mathcal{F}$ (*fin-clstate*)))

5.1.8 Lemmas about *ta-der′*

lemma *ta-det′-ground-id*:
t |∈| *ta-der′* $\mathcal{A}$ $s \Longrightarrow$ *ground* $t \Longrightarrow t = s$
by (*induct s arbitrary*: t) (*auto simp add*: *ta-der′.simps nth-equalityI*)

lemma *ta-det′-vars-term-id*:
t |∈| *ta-der′* $\mathcal{A}$ $s \Longrightarrow$ *vars-term* $t \cap$ *fset* ($\mathcal{Q}$ $\mathcal{A}$) = {} $\Longrightarrow t = s$
proof (*induct s arbitrary*: t)
case (*Fun f ss*)
from *Fun*(*2*−) **obtain** *ts* **where** [*simp*]: t = *Fun f ts* **and** *len*: *length ts* = *length ss*
by (*cases t*) (*auto dest*: *rule-statesD eps-dest-all*)
from *Fun*(*1*)[*OF nth-mem*, *of i ts* ! *i* **for** *i*] **show** *?case* **using** *Fun*(*2*−) *len*
by (*auto simp add*: *ta-der′.simps Union-disjoint*
dest: *rule-statesD eps-dest-all intro*!: *nth-equalityI*)
qed (*auto simp add*: *ta-der′.simps dest*: *rule-statesD eps-dest-all*)

lemma *fresh-states-ta-der′-pres*:
assumes *st*: $q \in$ *vars-term s q* $|\notin|$ $\mathcal{Q}$ $\mathcal{A}$
and *reach*: *t* $|\in|$ *ta-der′* $\mathcal{A}$ *s*
shows $q \in$ *vars-term t* **using** *reach st*(*1*)
proof (*induct s arbitrary*: *t*)
case (*Var x*)
then show *?case* **using** *assms*(*2*)
by (*cases t*) (*auto simp*: *ta-der′.simps dest*: *eps-trancl-statesD*)
next
case (*Fun f ss*)
from *Fun*(*3*) **obtain** *i* **where** *w*: $i <$ *length ss q* $\in$ *vars-term* (*ss* ! *i*) **by** (*auto simp*: *in-set-conv-nth*)
have $i <$ *length* (*args t*) $\wedge$ $q \in$ *vars-term* (*args t* ! *i*) **using** *Fun*(*2*) *w assms*(*2*) *Fun*(*1*)[*OF nth-mem*[*OF w*(*1*)] - *w*(*2*)]
using *rule-statesD*(*3*) *ta-der-to-ta-der′*
by (*auto simp*: *ta-der′.simps dest*: *rule-statesD*(*3*)) *fastforce*+
then show *?case* **by** (*cases t*) *auto*
qed

lemma *ta-der′-states*:
t $|\in|$ *ta-der′* $\mathcal{A}$ *s* $\Longrightarrow$ *vars-term t* $\subseteq$ *vars-term s* $\cup$ *fset* ($\mathcal{Q}$ $\mathcal{A}$)
proof (*induct s arbitrary*: *t*)
case (*Var x*) **then show** *?case*
by (*auto simp*: *ta-der′.simps dest*: *eps-dest-all*)
next
case (*Fun f ts*) **then show** *?case*
by (*auto simp*: *ta-der′.simps rule-statesD dest*: *eps-dest-all*)
(*metis* (*no-types*, *opaque-lifting*) *Un-iff in-set-conv-nth subsetD*)
qed

lemma *ta-der′-gterm-states*:
t $|\in|$ *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) $\Longrightarrow$ *vars-term t* $\subseteq$ *fset* ($\mathcal{Q}$ $\mathcal{A}$)
using *ta-der′-states*[*of t* $\mathcal{A}$ *term-of-gterm s*]
by *auto*

lemma *ta-der′-Var-funas*:
Var q $|\in|$ *ta-der′* $\mathcal{A}$ *s* $\Longrightarrow$ *funas-term s* $\subseteq$ *fset* (*ta-sig* $\mathcal{A}$)
by (*auto simp*: *less-eq-fset.rep-eq ffunas-term.rep-eq dest*!: *ta-der-term-sig ta-der′-to-ta-der*)

lemma *ta-sig-fsubsetI*:
assumes $\bigwedge$ *r*. *r* $|\in|$ *rules* $\mathcal{A}$ $\Longrightarrow$ (*r-root r*, *length* (*r-lhs-states r*)) $|\in|$ $\mathcal{F}$
shows *ta-sig* $\mathcal{A}$ $|\subseteq|$ $\mathcal{F}$ **using** *assms*
by (*auto simp*: *ta-sig-def*)

5.1.9 Signature induced by *refl-ta* and *refl-over-states-ta*

lemma *refl-ta-sig* [*simp*]:
ta-sig (*refl-ta* $\mathcal{F}$ *q*) = $\mathcal{F}$

ta-sig (*refl-over-states-ta* $Q\ \mathcal{F}\ \mathcal{A}\ q$) = $\mathcal{F}$
by (*auto simp*: *ta-sig-def refl-ta-def reflcl-rules-def refl-over-states-ta-def image-iff Bex-def*)

5.1.10 Correctness of *refl-ta*, *gen-reflcl-automaton*, and *reflcl-automaton*

lemma *refl-ta-eps* [*simp*]: *eps* (*refl-ta* $\mathcal{F}$ q) = {||}
by (*auto simp*: *refl-ta-def*)

lemma *refl-ta-sound*:
$s \in \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\Longrightarrow$ q |∈| *ta-der* (*refl-ta* $\mathcal{F}$ q) (*term-of-gterm* s)
by (*induct rule*: $\mathcal{T}_G$.*induct*) (*auto simp*: *refl-ta-def reflcl-rules-def image-iff Bex-def*)

lemma *reflcl-rules-args*:
length $ps = n \Longrightarrow f\ ps \rightarrow p$ |∈| *reflcl-rules* $\mathcal{F}$ $q \Longrightarrow ps$ = *replicate* n q
by (*auto simp*: *reflcl-rules-def*)

lemma $\mathcal{Q}$*-refl-ta*:
$\mathcal{Q}$ (*refl-ta* $\mathcal{F}$ q) |⊆| {|q|}
by (*auto simp*: $\mathcal{Q}$*-def refl-ta-def rule-states-def reflcl-rules-def fset-of-list-elem*)

lemma *refl-ta-complete1*:
Var p |∈| *ta-der′* (*refl-ta* $\mathcal{F}$ q) $s \Longrightarrow p \neq q \Longrightarrow s$ = *Var* p
by (*cases* s) (*auto simp*: *ta-der′.simps refl-ta-def reflcl-rules-def*)

lemma *refl-ta-complete2*:
Var q |∈| *ta-der′* (*refl-ta* $\mathcal{F}$ q) $s \Longrightarrow$ *funas-term* $s \subseteq$ *fset* $\mathcal{F} \wedge$ *vars-term* $s \subseteq \{q\}$
unfolding *ta-der-to-ta-der′*[*symmetric*]
using *ta-der-term-sig*[*of* q *refl-ta* $\mathcal{F}$ q s] *ta-der-states′*[*of* q *refl-ta* $\mathcal{F}$ q s]
using *fsubsetD*[*OF* $\mathcal{Q}$*-refl-ta*[*of* $\mathcal{F}$ q]]
by (*auto simp*: *ffunas-term.rep-eq*)
(*metis Term.term.simps*(17) *fresh-states-ta-der′-pres singletonD ta-der-to-ta-der′*)

lemma *gen-reflcl-lang*:
assumes q |∉| $\mathcal{Q}$ $\mathcal{A}$
shows *gta-lang* (*finsert* q Q) (*gen-reflcl-automaton* $\mathcal{F}$ $\mathcal{A}$ q) = *gta-lang* Q $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
(**is** *?Ls* = *?Rs*)
proof −
let *?A* = *gen-reflcl-automaton* $\mathcal{F}$ $\mathcal{A}$ q
interpret *sq*: *derivation-split ?A* $\mathcal{A}$ *refl-ta* $\mathcal{F}$ q
using *assms* **unfolding** *derivation-split-def*
by (*auto simp*: *gen-reflcl-automaton-def refl-ta-def reflcl-rules-def* $\mathcal{Q}$*-def*)
show *?thesis*
proof
{fix s **assume** $s \in$ *?Ls* **then obtain** p u **where**
seq: u |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm* s) *Var* p |∈| *ta-der′* (*refl-ta* $\mathcal{F}$ q) u **and**
fin: p |∈| *finsert* q Q

by (*auto simp*: *ta-der-to-ta-der′ elim*!: *gta-langE dest*!: *sq.ta-der′-split*)
have *vars-term u* ⊆ {*q*} ⟹ *u* = *term-of-gterm s* **using** *assms*
by (*intro ta-det′-vars-term-id*[*OF seq*(*1*)]) *auto*
then have *s* ∈ *?Rs* **using** *assms fin seq funas-term-of-gterm-conv*
using *refl-ta-complete1*[*OF seq*(*2*)]
by (*cases p* = *q*) (*auto simp*: *ta-der-to-ta-der′* $\mathcal{T}_G$*-funas-gterm-conv dest*!: *refl-ta-complete2*)**}**
then show *?Ls* ⊆ *gta-lang Q* $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) **by** *blast*
next
show *gta-lang Q* $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) ⊆ *?Ls*
using *sq.ta-der-monos* **unfolding** *gta-lang-def gta-der-def*
by (*auto dest*: *refl-ta-sound*)
qed
qed

lemma *reflcl-lang*:
gta-lang (*finsert None* (*Some* |‘| *Q*)) (*reflcl-automaton* $\mathcal{F}$ $\mathcal{A}$) = *gta-lang Q* $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
proof −
have *st*: *None* |∉| $\mathcal{Q}$ (*fmap-states-ta Some* $\mathcal{A}$) **by** *auto*
have *gta-lang Q* $\mathcal{A}$ = *gta-lang* (*Some* |‘| *Q*) (*fmap-states-ta Some* $\mathcal{A}$)
by (*simp add*: *finj-Some fmap-states-ta-lang2*)
then show *?thesis*
unfolding *reflcl-automaton-def Let-def gen-reflcl-lang*[*OF st*, *of Some* |‘| *Q* $\mathcal{F}$]
by *simp*
qed

lemma $\mathcal{L}$*-reflcl-reg*:
$\mathcal{L}$ (*reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
by (*simp add*: $\mathcal{L}$*-def reflcl-lang reflcl-reg-def*)

5.1.11 Correctness of *gen-parallel-closure-automaton* and *parallel-closure-reg*

lemma *set-list-subset-nth-conv*:
set xs ⊆ *A* ⟹ *i* < *length xs* ⟹ *xs* ! *i* ∈ *A*
by (*metis in-set-conv-nth subset-code*(*1*))

lemma *ground-gmctxt-of-mctxt-fill-holes′*:
num-holes C = *length ss* ⟹ *ground-mctxt C* ⟹ ∀ *s*∈*set ss*. *ground s* ⟹
fill-gholes (*gmctxt-of-mctxt C*) (*map gterm-of-term ss*) = *gterm-of-term* (*fill-holes C ss*)
using *ground-gmctxt-of-mctxt-fill-holes*
by (*metis term-of-gterm-inv*)

lemma *refl-over-states-ta-eps-trancl* [*simp*]:
(*eps* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*))|$^+$| = *eps* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*)
proof (*intro fequalityI fsubsetI*)
fix *x* **assume** *x* |∈| (*eps* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*))|$^+$|

hence (*fst x*, *snd x*) |∈| (*eps* (*refl-over-states-ta Q F A q*))|^+^|
by (*metis prod.exhaust-sel*)
thus *x* |∈| *eps* (*refl-over-states-ta Q F A q*)
by (*rule ftranclE*) (*auto simp add*: *refl-over-states-ta-def image-iff Bex-def dest*: *ftranclD*)
next
fix *x* **assume** *x* |∈| *eps* (*refl-over-states-ta Q F A q*)
thus *x* |∈| (*eps* (*refl-over-states-ta Q F A q*))|^+^|
by (*metis fr-into-trancl prod.exhaust-sel*)
qed

lemma *refl-over-states-ta-epsD*:
(*p*, *q*) |∈| (*eps* (*refl-over-states-ta Q F A q*)) ⟹ *p* |∈| *Q*
by (*auto simp*: *refl-over-states-ta-def*)

lemma *refl-over-states-ta-vars-term*:
q |∈| *ta-der* (*refl-over-states-ta Q F A q*) *u* ⟹ *vars-term u* ⊆ *insert q* (*fset Q*)
proof (*induct u*)
case (*Fun f ts*)
from *Fun*(*2*) *reflcl-rules-args*[*of - length ts f - F q*]
have *i* < *length ts* ⟹ *q* |∈| *ta-der* (*refl-over-states-ta Q F A q*) (*ts* ! *i*) **for** *i*
by (*fastforce simp*: *refl-over-states-ta-def*)
then have *i* < *length ts* ⟹ *x* ∈ *vars-term* (*ts* ! *i*) ⟹ *x* = *q* ∨ *x* |∈| *Q* **for** *i x*
using *Fun*(*1*)[*OF nth-mem*, *of i*]
by (*meson insert-iff subsetD*)
then show *?case* **by** (*fastforce simp*: *in-set-conv-nth*)
qed (*auto dest*: *refl-over-states-ta-epsD*)

lemmas *refl-over-states-ta-vars-term′* =
refl-over-states-ta-vars-term[*unfolded ta-der-to-ta-der′ ta-der′-target-args-vars-term-conv*,
THEN set-list-subset-nth-conv, *unfolded finsert.rep-eq*[*symmetric*]]

lemma *refl-over-states-ta-sound*:
funas-term u ⊆ *fset F* ⟹ *vars-term u* ⊆ *insert q* (*fset* (*Q* |∩| *Q A*)) ⟹ *q* |∈| *ta-der* (*refl-over-states-ta Q F A q*) *u*
proof (*induct u*)
case (*Fun f ts*)
have *reach*: *i* < *length ts* ⟹ *q* |∈| *ta-der* (*refl-over-states-ta Q F A q*) (*ts* ! *i*) **for** *i*
using *Fun*(*2*−) **by** (*intro Fun*(*1*)[*OF nth-mem*]) (*auto simp*: *SUP-le-iff*)
from *Fun*(*2*) **have** *TA-rule f* (*replicate* (*length ts*) *q*) *q* |∈| *rules* (*refl-over-states-ta Q F A q*)
by (*auto simp*: *refl-over-states-ta-def reflcl-rules-def fimage-iff fBex-def*)
then show *?case* **using** *reach*
by *force*
qed (*auto simp*: *refl-over-states-ta-def*)

lemma *gen-parallelcl-lang*:
fixes *A* :: (′*q*, ′*f*) *ta*

assumes $q \mid\notin\mid \mathcal{Q}\ \mathcal{A}$
shows *gta-lang* $\{\mid q \mid\}$ (*gen-parallel-closure-automaton* $Q\ \mathcal{F}\ \mathcal{A}\ q$) =
$\{$*fill-gholes* C *ss* $\mid$ C *ss. num-gholes* C = *length ss* $\wedge$ *funas-gmctxt* $C \subseteq$ (*fset* $\mathcal{F}$) $\wedge$ ($\forall$ $i <$ *length ss. ss* ! $i \in$ *gta-lang* $Q\ \mathcal{A}$)$\}$
(**is** *?Ls* = *?Rs*)
proof −
let *?A* = *gen-parallel-closure-automaton* $Q\ \mathcal{F}\ \mathcal{A}\ q$ **let** *?B* = *refl-over-states-ta* $Q\ \mathcal{F}\ \mathcal{A}\ q$
interpret *sq*: *derivation-split ?A* $\mathcal{A}$ *?B*
using *assms* **unfolding** *derivation-split-def*
by (*auto simp*: *gen-parallel-closure-automaton-def refl-over-states-ta-def* $\mathcal{Q}$*-def reflcl-rules-def*)
{fix s **assume** $s \in$ *?Ls* **then obtain** u **where**
seq: $u \mid\in\mid$ *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) *Var* $q \mid\in\mid$ *ta-der′?B* u **and**
fin: $q \mid\in\mid$ *finsert* $q\ Q$
by (*auto simp*: *ta-der-to-ta-der′ elim*!: *gta-langE dest*!: *sq.ta-der′-split*)
let *?w* = λ *i. ta-der′-source-args u* (*term-of-gterm s*) ! i
have $s \in$ *?Rs* **using** *seq*(*1*) *ta-der′-Var-funas*[*OF seq*(*2*)] *fin*
using *ground-ta-der-statesD*[*of ?w i ta-der′-target-args u* ! i $\mathcal{A}$ **for** i] *assms*
using *refl-over-states-ta-vars-term′*[*OF seq*(*2*)]
using *ta-der′-ground-mctxt-structure*[*OF seq*(*1*)]
by (*force simp*: *ground-gmctxt-of-mctxt-fill-holes′ ta-der′-source-args-term-of-gterm*
intro!: *exI*[*of - gmctxt-of-mctxt* (*ta-der′-target-mctxt u*)]
exI[*of - map gterm-of-term* (*ta-der′-source-args u* (*term-of-gterm s*))]
gta-langI[*of ta-der′-target-args u* ! i Q $\mathcal{A}$
gterm-of-term (*?w i*) **for** i])**}**
then have *ls*: *?Ls* $\subseteq$ *?Rs* **by** *blast*
{fix t **assume** $t \in$ *?Rs*
then obtain C *ss* **where** *len*: *num-gholes* C = *length ss* **and**
gr-fun: *funas-gmctxt* $C \subseteq$ *fset* $\mathcal{F}$ **and**
reachA: $\forall$ $i <$ *length ss. ss* ! $i \in$ *gta-lang* $Q\ \mathcal{A}$ **and**
const: t = *fill-gholes* C *ss* **by** *auto*
from *reachA* **obtain** *qs* **where** *length ss* = *length qs* $\forall$ $i <$ *length qs. qs* ! $i \mid\in\mid$ $Q \mid\cap\mid \mathcal{Q}\ \mathcal{A}$
$\forall$ $i <$ *length qs. qs* ! $i \mid\in\mid$ *ta-der* $\mathcal{A}$ ((*map term-of-gterm ss*) ! i)
using *Ex-list-of-length-P*[*of length ss* λ q *i.* $q \mid\in\mid$ *ta-der* $\mathcal{A}$ (*term-of-gterm* (*ss* ! i)) $\wedge$ $q \mid\in\mid$ Q]
by (*metis* (*full-types*) *finterI gta-langE gterm-ta-der-states length-map map-nth-eq-conv*)
then have $q \mid\in\mid$ *ta-der ?A* (*fill-holes* (*mctxt-of-gmctxt C*) (*map term-of-gterm ss*))
using *reachA len gr-fun*
by (*intro sq.mctxt-const-to-ta-der*[*of mctxt-of-gmctxt C map term-of-gterm ss qs q*])
(*auto simp*: *funas-mctxt-of-gmctxt-conv*
dest!: *in-set-idx intro*!: *refl-over-states-ta-sound*)
then have $t \in$ *?Ls* **unfolding** *const*
by (*simp add*: *fill-holes-mctxt-of-gmctxt-to-fill-gholes gta-langI len*)**}**
then show *?thesis* **using** *ls* **by** *blast*
qed

lemma *parallelcl-gmctxt-lang*:
fixes $\mathcal{A}$ :: ('*q*, '*f*) *reg*
shows $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{*fill-gholes C ss* |
C ss. num-gholes C = *length ss* ∧ *funas-gmctxt C* ⊆ *fset* $\mathcal{F}$ ∧ (∀ *i* < *length ss. ss* ! *i* ∈ $\mathcal{L}$ $\mathcal{A}$)}
proof −
have ∗: *gta-lang* (*fin* (*fmap-states-reg Some* $\mathcal{A}$)) (*fmap-states-ta Some* (*ta* $\mathcal{A}$)) = *gta-lang* (*fin* $\mathcal{A}$) (*ta* $\mathcal{A}$)
by (*simp add*: *finj-Some fmap-states-reg-def fmap-states-ta-lang2*)
have *None* |∉| $\mathcal{Q}$ (*fmap-states-ta Some* (*ta* $\mathcal{A}$)) **by** *auto*
from *gen-parallelcl-lang*[*OF this*, *of fin* (*fmap-states-reg Some* $\mathcal{A}$) $\mathcal{F}$] **show** *?thesis*
unfolding $\mathcal{L}$*-def parallel-closure-reg-def Let-def* ∗ *fmap-states-reg-def*
by (*simp add*: *finj-Some fmap-states-ta-lang2*)
qed

lemma *parallelcl-mctxt-lang*:
shows $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{(*gterm-of-term* :: ('*f*, '*q option*) *term* ⇒ '*f gterm*) (*fill-holes C* (*map term-of-gterm ss*)) |
C ss. num-holes C = *length ss* ∧ *ground-mctxt C* ∧ *funas-mctxt C* ⊆ *fset* $\mathcal{F}$ ∧ (∀ *i* < *length ss. ss* ! *i* ∈ $\mathcal{L}$ $\mathcal{A}$)}
by (*auto simp*: *parallelcl-gmctxt-lang*) (*metis funas-gmctxt-of-mctxt num-gholes-gmctxt-of-mctxt ground-gmctxt-of-gterm-of-term funas-mctxt-of-gmctxt-conv ground-mctxt-of-gmctxt mctxt-of-gmctxt-fill-holes num-holes-mctxt-of-gmctxt*)+

5.1.12 Correctness of *gen-ctxt-closure-reg* and *ctxt-closure-reg*

lemma *semantic-path-rules-rhs*:
r |∈| *semantic-path-rules Q* q_c q_i q_f ⟹ *r-rhs r* = q_f
by (*auto simp*: *semantic-path-rules-def*)

lemma *reflcl-over-single-ta-transl* [*simp*]:
(*eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f))|$^+$| = *eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f)
proof (*intro fequalityI fsubsetI*)
fix *x* **assume** *x* |∈| (*eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f))|$^+$|
hence (*fst x*, *snd x*) |∈| (*eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f))|$^+$|
by *simp*
thus *x* |∈| *eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f)
by (*smt* (*verit*, *ccfv-threshold*) *fimageE ftranclD ftranclE prod.collapse reflcl-over-single-ta-def snd-conv ta.sel*(*2*))
next
show ⋀*x*. *x* |∈| *eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f) ⟹
x |∈| (*eps* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f))|$^+$|
by *auto*
qed

lemma *reflcl-over-single-ta-epsD*:

$(p,\ q_f) \mid\in\mid$ *eps* (*reflcl-over-single-ta* $Q\ \mathcal{F}\ q_c\ q_f) \Longrightarrow p \mid\in\mid Q$
$(p,\ q) \mid\in\mid$ *eps* (*reflcl-over-single-ta* $Q\ \mathcal{F}\ q_c\ q_f) \Longrightarrow q = q_f$
by (*auto simp*: *reflcl-over-single-ta-def*)

lemma *reflcl-over-single-ta-rules-split*:
$r \mid\in\mid$ *rules* (*reflcl-over-single-ta* $Q\ \mathcal{F}\ q_c\ q_f) \Longrightarrow$
$r \mid\in\mid$ *reflcl-rules* $\mathcal{F}\ q_c \vee r \mid\in\mid$ *semantic-path-rules* $\mathcal{F}\ q_c\ q_f\ q_f$
by (*auto simp*: *reflcl-over-single-ta-def*)

lemma *reflcl-over-single-ta-rules-semantic-path-rulesI*:
$r \mid\in\mid$ *semantic-path-rules* $\mathcal{F}\ q_c\ q_f\ q_f \Longrightarrow r \mid\in\mid$ *rules* (*reflcl-over-single-ta* $Q\ \mathcal{F}$ $q_c\ q_f$)
by (*auto simp*: *reflcl-over-single-ta-def*)

lemma *semantic-path-rules-fmember* [*intro*]:
TA-rule f qs q $\mid\in\mid$ *semantic-path-rules* $\mathcal{F}\ q_c\ q_i\ q_f \longleftrightarrow (\exists\ n\ i.\ (f,\ n) \mid\in\mid \mathcal{F} \wedge i$ $< n \wedge q = q_f \wedge$
$(qs = (replicate\ n\ q_c)[i := q_i]))$ (**is** *?Ls* $\longleftrightarrow$ *?Rs*)
by (*force simp*: *semantic-path-rules-def fBex-def fimage-iff fset-of-list-elem*)

lemma *semantic-path-rules-fmemberD*:
$r \mid\in\mid$ *semantic-path-rules* $\mathcal{F}\ q_c\ q_i\ q_f \Longrightarrow (\exists\ n\ i.$ (*r-root* $r,\ n) \mid\in\mid \mathcal{F} \wedge i < n \wedge$ *r-rhs* $r = q_f \wedge$
(*r-lhs-states* $r = (replicate\ n\ q_c)[i := q_i])$)
by (*cases* r) (*simp add*: *semantic-path-rules-fmember*)

lemma *reflcl-over-single-ta-vars-term-*q_c:
$q_c \neq q_f \Longrightarrow q_c \mid\in\mid$ *ta-der* (*reflcl-over-single-ta* $Q\ \mathcal{F}\ q_c\ q_f)\ u \Longrightarrow$
vars-term-list $u =$ *replicate* (*length* (*vars-term-list* u)) q_c
proof (*induct* u)
case (*Fun f ts*)
have $i <$ *length ts* $\Longrightarrow q_c \mid\in\mid$ *ta-der* (*reflcl-over-single-ta* $Q\ \mathcal{F}\ q_c\ q_f)$ (*ts* ! i) **for** i **using** *Fun*(*2*, *3*)
by (*auto dest*!: *reflcl-over-single-ta-rules-split reflcl-over-single-ta-epsD reflcl-rules-args semantic-path-rules-rhs*)
then have $i <$ *length* (*concat* (*map vars-term-list ts*)) $\Longrightarrow$ *concat* (*map vars-term-list ts*) ! $i = q_c$ **for** i
using *Fun*(*1*)[*OF nth-mem Fun*(*2*)]
by (*metis* (*no-types*, *lifting*) *length-map nth-concat-split nth-map nth-replicate*)
then show *?case* **using** *Fun*(*1*)[*OF nth-mem Fun*(*2*)]
by (*auto intro*: *nth-equalityI*)
qed (*auto dest*: *reflcl-over-single-ta-epsD*)

lemma *reflcl-over-single-ta-vars-term*:
$q_c \mid\notin\mid Q \Longrightarrow q_c \neq q_f \Longrightarrow q_f \mid\in\mid$ *ta-der* (*reflcl-over-single-ta* $Q\ \mathcal{F}\ q_c\ q_f)\ u \Longrightarrow$
length (*vars-term-list* u) $= n \Longrightarrow (\exists\ i\ q.\ i < n \wedge q \mid\in\mid$ *finsert* $q_f\ Q \wedge$ *vars-term-list* $u = (replicate\ n\ q_c)[i := q]$)
proof (*induct u arbitrary*: n)

case (*Var x*) **then show** *?case*
by (*intro exI*[*of - 0*] *exI*[*of - x*]) (*auto dest*: *reflcl-over-single-ta-epsD*(*1*))
next
case (*Fun f ts*)
from *Fun*(*2*, *3*, *4*) **obtain** *qs* **where** *rule*: *TA-rule f qs* q_f |∈| *semantic-path-rules* $\mathcal{F}$ q_c q_f q_f
length qs = *length ts* ∀ *i* < *length ts*. *qs* ! *i* |∈| *ta-der* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f) (*ts* ! *i*)
using *semantic-path-rules-rhs reflcl-over-single-ta-epsD*
by (*fastforce simp*: *reflcl-rules-def dest*!: *reflcl-over-single-ta-rules-split*)
from *rule*(*1*, *2*) **obtain** *i* **where** *states*: *i* < *length ts qs* = (*replicate* (*length ts*) q_c)[*i* := q_f]
by (*auto simp*: *semantic-path-rules-fmember*)
then have *qc*: *j* < *length ts* ⟹ *j* ≠ *i* ⟹ *vars-term-list* (*ts* ! *j*) = *replicate* (*length* (*vars-term-list* (*ts* ! *j*))) q_c **for** *j*
using *reflcl-over-single-ta-vars-term-*q_c[*OF Fun*(*3*)] *rule*
by *force*
from *Fun*(*1*)[*OF nth-mem*, *of i*] *Fun*(*2*, *3*) *rule states* **obtain** *k q* **where**
qf: *k* < *length* (*vars-term-list* (*ts* ! *i*)) *q* |∈| *finsert* q_f *Q*
vars-term-list (*ts* ! *i*) = (*replicate* (*length* (*vars-term-list* (*ts* ! *i*))) q_c)[*k* := *q*]
by (*auto simp*: *nth-list-update split*: *if-splits*)
let *?l* = *sum-list* (*map length* (*take i* (*map vars-term-list ts*))) + *k*
show *?case* **using** *qc qf rule*(*2*) *Fun*(*5*) *states*(*1*)
apply (*intro exI*[*of - ?l*] *exI*[*of - q*])
apply (*auto simp*: *concat-nth-length nth-list-update elim*!: *nth-concat-split*′ *intro*!: *nth-equalityI*)
apply (*metis length-replicate nth-list-update-eq nth-list-update-neq nth-replicate*)+
done
qed

lemma *refl-ta-reflcl-over-single-ta-mono*:
q |∈| *ta-der* (*refl-ta* $\mathcal{F}$ *q*) *t* ⟹ *q* |∈| *ta-der* (*reflcl-over-single-ta Q* $\mathcal{F}$ *q* q_f) *t*
by (*intro ta-der-el-mono*[**where** *?*$\mathcal{B}$ = *reflcl-over-single-ta Q* $\mathcal{F}$ *q* q_f])
(*auto simp*: *refl-ta-def reflcl-over-single-ta-def*)

lemma *reflcl-over-single-ta-sound*:
assumes *funas-gctxt C* ⊆ *fset* $\mathcal{F}$ *q* |∈| *Q*
shows q_f |∈| *ta-der* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f) (*ctxt-of-gctxt C*)⟨*Var q*⟩
using *assms*(*1*)
proof (*induct C*)
case *GHole* **then show** *?case* **using** *assms*(*2*)
by (*auto simp add*: *reflcl-over-single-ta-def*)
next
case (*GMore f ss C ts*)
let *?i* = *length ss* **let** *?n* = *Suc* (*length ss* + *length ts*)
from *GMore* **have** (*f*, *?n*) |∈| $\mathcal{F}$ **by** *auto*
then have *f* ((*replicate ?n* q_c)[*?i* := q_f]) → q_f |∈| *rules* (*reflcl-over-single-ta Q* $\mathcal{F}$ q_c q_f)
using *semantic-path-rules-fmember*[*of f* (*replicate ?n* q_c)[*?i* := q_f] q_f $\mathcal{F}$ q_c q_f

q_f]
using *less-add-Suc1*
by (*intro reflcl-over-single-ta-rules-semantic-path-rulesI*) *blast*
moreover from *GMore*(*2*) **have** $i <$ *length ss* $\Longrightarrow$ q_c $|\in|$ *ta-der* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) (*term-of-gterm* (*ss* ! *i*)) **for** *i*
by (*intro refl-ta-reflcl-over-single-ta-mono refl-ta-sound*) (*auto simp*: *SUP-le-iff* $\mathcal{T}_G$*-funas-gterm-conv*)
moreover from *GMore*(*2*) **have** $i <$ *length ts* $\Longrightarrow$ q_c $|\in|$ *ta-der* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) (*term-of-gterm* (*ts* ! *i*)) **for** *i*
by (*intro refl-ta-reflcl-over-single-ta-mono refl-ta-sound*) (*auto simp*: *SUP-le-iff* $\mathcal{T}_G$*-funas-gterm-conv*)
moreover from *GMore* **have** q_f $|\in|$ *ta-der* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) (*ctxt-of-gctxt* $C\langle Var\ q\rangle$ **by** *auto*
ultimately show *?case*
by (*auto simp*: *nth-append-Cons simp del*: *replicate.simps intro*!: *exI*[*of* - (*replicate ?n* q_c)[*?i* := q_f]] *exI*[*of* - q_f])
qed

lemma *reflcl-over-single-ta-sig*: *ta-sig* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) $|\subseteq|$ $\mathcal{F}$
by (*intro ta-sig-fsubsetI*)
(*auto simp*: *reflcl-rules-def dest*!: *semantic-path-rules-fmemberD reflcl-over-single-ta-rules-split*)

lemma *gen-gctxtcl-lang*:
assumes q_c $|\notin|$ $\mathcal{Q}$ $\mathcal{A}$ **and** q_f $|\notin|$ $\mathcal{Q}$ $\mathcal{A}$ **and** q_c $|\notin|$ Q **and** $q_c \neq q_f$
shows *gta-lang* $\{|q_f|\}$ (*gen-ctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_f) =
$\{C\langle s\rangle_G \mid C\ s.$ *funas-gctxt* $C \subseteq$ *fset* $\mathcal{F} \wedge s \in$ *gta-lang* Q $\mathcal{A}\}$
(**is** *?Ls* = *?Rs*)
proof −
let *?A* = *gen-ctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_f **let** *?B* = *reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f
interpret *sq*: *derivation-split ?A* $\mathcal{A}$ *?B*
using *assms* **unfolding** *derivation-split-def*
by (*auto simp*: *gen-ctxt-closure-automaton-def reflcl-over-single-ta-def* $\mathcal{Q}$*-def reflcl-rules-def*
dest!: *semantic-path-rules-rhs*)
{fix *s* **assume** *s* ∈ *?Ls* **then obtain** *u* **where**
seq: *u* $|\in|$ *ta-der*′ $\mathcal{A}$ (*term-of-gterm s*) *Var* q_f $|\in|$ *ta-der*′*?B u* **using** *sq.ta-der*′*-split*
by (*force simp*: *ta-der-to-ta-der*′ *elim*!: *gta-langE*)
have $q_c \notin$ *vars-term u* $q_f \notin$ *vars-term u*
using *subsetD*[*OF ta-der*′*-gterm-states*[*OF seq*(*1*)]] *assms*(*1*, *2*)
by (*auto simp flip*: *set-vars-term-list*)
then obtain *q* **where** *vars*: *vars-term-list u* = [*q*] **and** *fin*: *q* $|\in|$ Q **unfolding** *set-vars-term-list*[*symmetric*]
using *reflcl-over-single-ta-vars-term*[*unfolded ta-der-to-ta-der*′, *OF assms*(*3*, *4*) *seq*(*2*), *of length* (*vars-term-list u*)]
by (*metis* (*no-types*, *lifting*) *finsertE in-set-conv-nth length-0-conv length-Suc-conv length-replicate lessI less-Suc-eq-0-disj nth-Cons-0 nth-list-update nth-replicate zero-less-Suc*)
have *s* ∈ *?Rs* **using** *fin ta-der*′*-ground-ctxt-structure*[*OF seq*(*1*) *vars*]

using *ta-der'-Var-funas*[*OF seq*(*2*), *THEN subset-trans*, *OF reflcl-over-single-ta-sig*[*unfolded less-eq-fset.rep-eq*]]
by (*auto intro*!: *exI*[*of - ta-der'-gctxt u*] *exI*[*of - ta-der'-source-gctxt-arg u s*])
(*metis Un-iff funas-ctxt-apply funas-ctxt-of-gctxt-conv subset-eq*)
}
then have *ls*: *?Ls* ⊆ *?Rs* **by** *blast*
{fix *t* **assume** *t* ∈ *?Rs*
then obtain *C s* **where** *gr-fun*: *funas-gctxt C* ⊆ *fset* $\mathcal{F}$ **and** *reachA*: *s* ∈ *gta-lang Q* $\mathcal{A}$ **and**
const: *t* = $C\langle s\rangle_G$ **by** *auto*
from *reachA* **obtain** *q* **where** *der-A*: *q* |∈| *Q* |∩| $\mathcal{Q}$ $\mathcal{A}$ *q* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm s*)
by *auto*
have q_f |∈| *ta-der ?B* (*ctxt-of-gctxt C*)⟨*Var q*⟩ **using** *gr-fun der-A*(*1*)
using *reflcl-over-single-ta-sound*[*OF gr-fun*]
by *force*
then have *t* ∈ *?Ls* **unfolding** *const*
by (*meson der-A*(*2*) *finsertI1 gta-langI sq.gctxt-const-to-ta-der*)**}**
then show *?thesis* **using** *ls* **by** *blast*
qed

lemma *gen-gctxt-closure-sound*:
fixes $\mathcal{A}$ :: ('*q*, '*f*) *reg*
assumes q_c |∉| $\mathcal{Q}_r$ $\mathcal{A}$ **and** q_f |∉| $\mathcal{Q}_r$ $\mathcal{A}$ **and** q_c |∉| *fin* $\mathcal{A}$ **and** $q_c \neq q_f$
shows $\mathcal{L}$ (*gen-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ q_c q_f) = {$C\langle s\rangle_G$ | *C s*. *funas-gctxt C* ⊆ *fset* $\mathcal{F}$ ∧ *s* ∈ $\mathcal{L}$ $\mathcal{A}$}
using *gen-gctxtcl-lang*[*OF assms*] **unfolding** $\mathcal{L}$-*def*
by (*simp add*: *gen-ctxt-closure-reg-def*)

lemma *gen-ctxt-closure-sound*:
fixes $\mathcal{A}$ :: ('*q*, '*f*) *reg*
assumes q_c |∉| $\mathcal{Q}_r$ $\mathcal{A}$ **and** q_f |∉| $\mathcal{Q}_r$ $\mathcal{A}$ **and** q_c |∉| *fin* $\mathcal{A}$ **and** $q_c \neq q_f$
shows $\mathcal{L}$ (*gen-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ q_c q_f) =
{(*gterm-of-term* :: ('*f*, '*q*) *term* ⇒ '*f gterm*) $C\langle$*term-of-gterm s*$\rangle$ | *C s*. *ground-ctxt C* ∧ *funas-ctxt C* ⊆ *fset* $\mathcal{F}$ ∧ *s* ∈ $\mathcal{L}$ $\mathcal{A}$}
unfolding *gen-gctxt-closure-sound*[*OF assms*]
by (*metis* (*no-types*, *opaque-lifting*) *ctxt-of-gctxt-apply funas-ctxt-of-gctxt-conv gctxt-of-ctxt-inv ground-ctxt-of-gctxt*)

lemma *gctxt-closure-lang*:
shows $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{ $C\langle s\rangle_G$ | *C s*. *funas-gctxt C* ⊆ *fset* $\mathcal{F}$ ∧ *s* ∈ $\mathcal{L}$ $\mathcal{A}$}
proof −
let *?B* = *fmap-states-reg Inl* (*reg-Restr-Q*$_f$ $\mathcal{A}$)
have *ts*: *Inr False* |∉| $\mathcal{Q}_r$ *?B Inr True* |∉| $\mathcal{Q}_r$ *?B Inr False* |∉| *fin* (*fmap-states-reg Inl* (*reg-Restr-Q*$_f$ $\mathcal{A}$))
by (*auto simp*: *fmap-states-reg-def fmap-states-ta-def' Q-def rule-states-def*)
from *gen-gctxt-closure-sound*[*OF ts*] **show** *?thesis*
by (*simp add*: *ctxt-closure-reg-def*)

qed

lemma *ctxt-closure-lang*:
shows $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{(*gterm-of-term* :: ('*f*, '*q* + *bool*) *term* ⇒ '*f gterm*) $C\langle$*term-of-gterm s*$\rangle$ |
C s. *ground-ctxt C* ∧ *funas-ctxt C* ⊆ *fset* $\mathcal{F}$ ∧ *s* ∈ $\mathcal{L}$ $\mathcal{A}$}
unfolding *gctxt-closure-lang*
by (*metis* (*mono-tags*, *opaque-lifting*) *ctxt-of-gctxt-inv funas-gctxt-of-ctxt ground-ctxt-of-gctxt ground-gctxt-of-ctxt-apply-gterm term-of-gterm-inv*)

5.1.13 Correctness of *gen-nhole-ctxt-closure-automaton* and *nhole-ctxt-closure-reg*

lemma *reflcl-over-nhole-ctxt-ta-vars-term-*q_c:
$q_c \neq q_f \Longrightarrow q_c \neq q_i \Longrightarrow q_c$ |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) *u*
⟹
vars-term-list u = *replicate* (*length* (*vars-term-list u*)) q_c
proof (*induct u*)
case (*Fun f ts*)
have *i* < *length ts* ⟹ q_c |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) (*ts* ! *i*) **for** *i* **using** *Fun*(*2*, *3*, *4*)
by (*auto simp*: *reflcl-over-nhole-ctxt-ta-def dest*!: *ftranclD2 reflcl-rules-args semantic-path-rules-rhs*)
then have *i* < *length* (*concat* (*map vars-term-list ts*)) ⟹ *concat* (*map vars-term-list ts*) ! *i* = q_c **for** *i*
using *Fun*(*1*)[*OF nth-mem Fun*(*2*, *3*)]
by (*metis* (*no-types*, *lifting*) *length-map map-nth-eq-conv nth-concat-split*′ *nth-replicate*)
then show *?case* **using** *Fun*(*1*)[*OF nth-mem Fun*(*2*)]
by (*auto intro*: *nth-equalityI*)
qed (*auto simp*: *reflcl-over-nhole-ctxt-ta-def dest*: *ftranclD2*)

lemma *reflcl-over-nhole-ctxt-ta-vars-term-Var*:
assumes *disj*: q_c |∉| *Q* q_f |∉| *Q* $q_c \neq q_f$ $q_i \neq q_f$ $q_c \neq q_i$
and *reach*: q_i |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) *u*
shows (∃ *q*. *q* |∈| *finsert* q_i *Q* ∧ *u* = *Var q*) **using** *assms*
by (*cases u*) (*fastforce simp*: *reflcl-over-nhole-ctxt-ta-def reflcl-rules-def dest*: *ftranclD semantic-path-rules-rhs*)+

lemma *reflcl-over-nhole-ctxt-ta-vars-term*:
assumes *disj*: q_c |∉| *Q* q_f |∉| *Q* $q_c \neq q_f$ $q_i \neq q_f$ $q_c \neq q_i$
and *reach*: q_f |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) *u*
shows (∃ *i q*. *i* < *length* (*vars-term-list u*) ∧ *q* |∈| {|q_i, q_f|} |∪| *Q* ∧ *vars-term-list u* = (*replicate* (*length* (*vars-term-list u*)) q_c)[*i* := *q*])
using *assms*
proof (*induct u*)
case (*Var q*) **then show** *?case*
by (*intro exI*[*of* - *0*] *exI*[*of* - *q*]) (*auto simp*: *reflcl-over-nhole-ctxt-ta-def dest*: *ftranclD2*)
next
case (*Fun f ts*)

from *Fun*(2 − 7) **obtain** *q qs* **where** *rule*: *TA-rule f qs* q_f |∈| *semantic-path-rules* $\mathcal{F}$ q_c *q* q_f *q* = q_i ∨ *q* = q_f
length qs = *length ts* ∀ *i* < *length ts*. *qs* ! *i* |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) (*ts* ! *i*)
by (*auto simp*: *reflcl-over-nhole-ctxt-ta-def reflcl-rules-def dest*: *ftranclD2*)
from *rule*(1− 3) **obtain** *i* **where** *states*: *i* < *length ts qs* = (*replicate* (*length ts*) q_c)[*i* := *q*]
by (*auto simp*: *semantic-path-rules-fmember*)
then have *qc*: *j* < *length ts* ⟹ *j* ≠ *i* ⟹ *vars-term-list* (*ts* ! *j*) = *replicate* (*length* (*vars-term-list* (*ts* ! *j*))) q_c **for** *j*
using *reflcl-over-nhole-ctxt-ta-vars-term-*q_c[*OF Fun*(4, 6)] *rule*
by *force*
from *rule states* **have** *q* |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) (*ts* ! *i*)
by *auto*
from *this Fun*(1)[*OF nth-mem*, *of i*, *OF* - *Fun*(2 − 6)] *rule*(2) *states*(1) **obtain** *k q*′ **where**
qf: *k* < *length* (*vars-term-list* (*ts* ! *i*)) *q*′ |∈| {|q_i, q_f|} |∪| *Q*
vars-term-list (*ts* ! *i*) = (*replicate* (*length* (*vars-term-list* (*ts* ! *i*))) q_c)[*k* := *q*′]
using *reflcl-over-nhole-ctxt-ta-vars-term-Var*[*OF Fun*(2 − 6), *of* $\mathcal{F}$ *ts* ! *i*]
by (*auto simp*: *nth-list-update split*: *if-splits*) *blast*
let *?l* = *sum-list* (*map length* (*take i* (*map vars-term-list ts*))) + *k*
show *?case* **using** *qc qf rule*(3) *states*(1)
apply (*intro exI*[*of* - *?l*] *exI*[*of* - *q*′])
apply (*auto 0 0 simp*: *concat-nth-length nth-list-update elim*!: *nth-concat-split*′ *intro*!: *nth-equalityI*)
apply (*metis length-replicate nth-list-update-eq nth-list-update-neq nth-replicate*)+
done
qed

lemma *reflcl-over-nhole-ctxt-ta-mono*:
q |∈| *ta-der* (*refl-ta* $\mathcal{F}$ *q*) *t* ⟹ *q* |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ *q* q_i q_f) *t*
by (*intro ta-der-el-mono*[**where** *?*$\mathcal{B}$ = *reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ *q* q_i q_f])
(*auto simp*: *refl-ta-def reflcl-over-nhole-ctxt-ta-def*)

lemma *reflcl-over-nhole-ctxt-ta-sound*:
assumes *funas-gctxt C* ⊆ *fset* $\mathcal{F}$ *C* ≠ *GHole q* |∈| *Q*
shows q_f |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) (*ctxt-of-gctxt C*)⟨*Var q*⟩ **using** *assms*(1, 2)
proof (*induct C*)
case *GHole* **then show** *?case* **using** *assms*(2)
by (*auto simp add*: *reflcl-over-single-ta-def*)
next
case (*GMore f ss C ts*) **note** *IH* = *this*
let *?i* = *length ss* **let** *?n* = *Suc* (*length ss* + *length ts*)
from *GMore* **have** *funas*: (*f*, *?n*) |∈| $\mathcal{F}$ **by** *auto*
from *GMore*(2) **have** *args-ss*: *i* < *length ss* ⟹ q_c |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta*

$Q\ \mathcal{F}\ q_c\ q_i\ q_f$) (*term-of-gterm* (*ss* ! *i*)) **for** *i*
by (*intro reflcl-over-nhole-ctxt-ta-mono refl-ta-sound*) (*auto simp*: *SUP-le-iff* $\mathcal{T}_G$*-funas-gterm-conv*)
from *GMore*(*2*) **have** *args-ts*: $i <$ *length ts* $\Longrightarrow q_c \mid\in\mid$ *ta-der* (*reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$) (*term-of-gterm* (*ts* ! *i*)) **for** *i*
by (*intro reflcl-over-nhole-ctxt-ta-mono refl-ta-sound*) (*auto simp*: *SUP-le-iff* $\mathcal{T}_G$*-funas-gterm-conv*)
note *args* = *this*
show *?case*
proof (*cases C*)
case [*simp*]: *GHole*
from *funas* **have** *f* ((*replicate ?n* q_c)[*?i* := q_i]) → $q_f \mid\in\mid$ *rules* (*reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$)
using *semantic-path-rules-fmember*[*of f* (*replicate ?n* q_c)[*?i* := q_i] $q_f\ \mathcal{F}\ q_c\ q_i\ q_f$]
unfolding *reflcl-over-nhole-ctxt-ta-def*
by (*metis funionCI less-add-Suc1 ta.sel*(*1*))
moreover have $q_i \mid\in\mid$ *ta-der* (*reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$) (*ctxt-of-gctxt* $C\langle Var\ q\rangle$
using *assms*(*3*) **by** (*auto simp*: *reflcl-over-nhole-ctxt-ta-def*)
ultimately show *?thesis* **using** *args-ss args-ts*
by (*auto simp*: *nth-append-Cons simp del*: *replicate.simps intro*!: *exI*[*of* - (*replicate ?n* q_c)[*?i* := q_i]] *exI*[*of* - q_f])
next
case (*GMore x21 x22 x23 x24*)
then have $q_f \mid\in\mid$ *ta-der* (*reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$) (*ctxt-of-gctxt* $C\langle Var\ q\rangle$
using *IH*(*1*, *2*) **by** *auto*
moreover from *funas* **have** *f* ((*replicate ?n* q_c)[*?i* := q_f]) → $q_f \mid\in\mid$ *rules* (*reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$)
using *semantic-path-rules-fmember*[*of f* (*replicate ?n* q_c)[*?i* := q_f] $q_f\ \mathcal{F}\ q_c\ q_f\ q_f$]
unfolding *reflcl-over-nhole-ctxt-ta-def*
by (*metis funionI2 less-add-Suc1 ta.sel*(*1*))
ultimately show *?thesis* **using** *args-ss args-ts*
by (*auto simp*: *nth-append-Cons simp del*: *replicate.simps intro*!: *exI*[*of* - (*replicate ?n* q_c)[*?i* := q_f]] *exI*[*of* - q_f])
qed
qed

lemma *reflcl-over-nhole-ctxt-ta-sig*: *ta-sig* (*reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$) $\mid\subseteq\mid\ \mathcal{F}$
by (*intro ta-sig-fsubsetI*)
(*auto simp*: *reflcl-over-nhole-ctxt-ta-def reflcl-rules-def dest*!: *semantic-path-rules-fmemberD*)

lemma *gen-nhole-gctxt-closure-lang*:
assumes $q_c \mid\notin\mid \mathcal{Q}\ \mathcal{A}\ q_i \mid\notin\mid \mathcal{Q}\ \mathcal{A}\ q_f \mid\notin\mid \mathcal{Q}\ \mathcal{A}$
and $q_c \mid\notin\mid Q\ q_f \mid\notin\mid Q$
and $q_c \neq q_i\ q_c \neq q_f\ q_i \neq q_f$

shows *gta-lang* $\{|q_f|\}$ (*gen-nhole-ctxt-closure-automaton* $Q\ \mathcal{F}\ \mathcal{A}\ q_c\ q_i\ q_f$) =
$\{C\langle s\rangle_G \mid C\ s.\ C \neq GHole \wedge$ *funas-gctxt* $C \subseteq$ *fset* $\mathcal{F} \wedge s \in$ *gta-lang* $Q\ \mathcal{A}\}$
(**is** *?Ls* = *?Rs*)
proof −
let *?A* = *gen-nhole-ctxt-closure-automaton* $Q\ \mathcal{F}\ \mathcal{A}\ q_c\ q_i\ q_f$ **let** *?B* = *reflcl-over-nhole-ctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$
interpret *sq*: *derivation-split ?A* $\mathcal{A}$ *?B*
using *assms* **unfolding** *derivation-split-def*
by (*auto simp*: *gen-nhole-ctxt-closure-automaton-def reflcl-over-nhole-ctxt-ta-def* $\mathcal{Q}$*-def reflcl-rules-def*
dest!: *semantic-path-rules-rhs*)
{fix *s* **assume** *s* ∈ *?Ls* **then obtain** *u* **where**
seq: *u* |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) *Var* q_f |∈| *ta-der′?B u* **using** *sq.ta-der′-split*
by (*force simp*: *ta-der-to-ta-der′ elim*!: *gta-langE*)
have $q_c \notin$ *vars-term u* $q_i \notin$ *vars-term u* $q_f \notin$ *vars-term u*
using *subsetD*[*OF ta-der′-gterm-states*[*OF seq*(*1*)]] *assms*(*1* − *3*)
by (*auto simp flip*: *set-vars-term-list*)
then obtain *q* **where** *vars*: *vars-term-list u* = [*q*] **and** *fin*: *q* |∈| *Q*
unfolding *set-vars-term-list*[*symmetric*]
using *reflcl-over-nhole-ctxt-ta-vars-term*[*unfolded ta-der-to-ta-der′*, *OF assms*(*4*, *5*, *7* − *8*, *6*) *seq*(*2*)]
by (*metis* (*no-types*, *opaque-lifting*) *finsert-iff funion-commute funion-finsert-right length-greater-0-conv lessI list.size*(*3*) *list-update-code*(*2*) *not0-implies-Suc nth-list-update-neq nth-mem nth-replicate replicate-Suc replicate-empty sup-bot.right-neutral*)
from *seq*(*2*) **have** *ta-der′-gctxt u* ≠ *GHole* **using** *ta-der′-ground-ctxt-structure*(*1*)[*OF seq*(*1*) *vars*]
using *fin assms*(*4*, *5*, *8*) **by** (*auto simp*: *reflcl-over-nhole-ctxt-ta-def dest*!: *ftranclD2*)
then have *s* ∈ *?Rs* **using** *fin ta-der′-ground-ctxt-structure*[*OF seq*(*1*) *vars*] *seq*(*2*)
using *ta-der′-Var-funas*[*OF seq*(*2*), *THEN subset-trans*, *OF reflcl-over-nhole-ctxt-ta-sig*[*unfolded less-eq-fset.rep-eq*]]
by (*auto intro*!: *exI*[*of - ta-der′-gctxt u*] *exI*[*of - ta-der′-source-gctxt-arg u s*])
(*metis Un-iff funas-ctxt-apply funas-ctxt-of-gctxt-conv in-mono*)**}**
then have *ls*: *?Ls* ⊆ *?Rs* **by** *blast*
{fix *t* **assume** *t* ∈ *?Rs*
then obtain *C s* **where** *gr-fun*: *funas-gctxt C* ⊆ *fset* $\mathcal{F}$ *C* ≠ *GHole* **and** *reachA*: *s* ∈ *gta-lang Q* $\mathcal{A}$ **and**
const: *t* = $C\langle s\rangle_G$ **by** *auto*
from *reachA* **obtain** *q* **where** *der-A*: *q* |∈| *Q* |∩| $\mathcal{Q}$ $\mathcal{A}$ *q* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm s*)
by *auto*
have q_f |∈| *ta-der ?B* (*ctxt-of-gctxt C*)⟨*Var q*⟩ **using** *gr-fun der-A*(*1*)
using *reflcl-over-nhole-ctxt-ta-sound*[*OF gr-fun*]
by *force*
then have *t* ∈ *?Ls* **unfolding** *const*
by (*meson der-A*(*2*) *finsertI1 gta-langI sq.gctxt-const-to-ta-der*)**}**
then show *?thesis* **using** *ls* **by** *blast*

qed

lemma *gen-nhole-gctxt-closure-sound*:
assumes $q_c \mid\notin\mid \mathcal{Q}_r\ \mathcal{A}\ q_i \mid\notin\mid \mathcal{Q}_r\ \mathcal{A}\ q_f \mid\notin\mid \mathcal{Q}_r\ \mathcal{A}$
and $q_c \mid\notin\mid (\mathit{fin}\ \mathcal{A})\ q_f \mid\notin\mid (\mathit{fin}\ \mathcal{A})$
and $q_c \neq q_i\ q_c \neq q_f\ q_i \neq q_f$
shows $\mathcal{L}\ (\mathit{gen\text{-}nhole\text{-}ctxt\text{-}closure\text{-}reg}\ \mathcal{F}\ \mathcal{A}\ q_c\ q_i\ q_f) =$
$\{\ C\langle s\rangle_G \mid C\ s.\ C \neq \mathit{GHole} \wedge \mathit{funas\text{-}gctxt}\ C \subseteq \mathit{fset}\ \mathcal{F} \wedge s \in \mathcal{L}\ \mathcal{A}\}$
using *gen-nhole-gctxt-closure-lang*[*OF assms*] **unfolding** $\mathcal{L}$-*def*
by (*auto simp*: *gen-nhole-ctxt-closure-reg-def*)

lemma *nhole-ctxtcl-lang*:
$\mathcal{L}\ (\mathit{nhole\text{-}ctxt\text{-}closure\text{-}reg}\ \mathcal{F}\ \mathcal{A}) =$
$\{\ C\langle s\rangle_G \mid C\ s.\ C \neq \mathit{GHole} \wedge \mathit{funas\text{-}gctxt}\ C \subseteq \mathit{fset}\ \mathcal{F} \wedge s \in \mathcal{L}\ \mathcal{A}\}$
proof −
let *?B* = *fmap-states-reg* (*Inl* :: *'b* ⇒ *'b* + *cl-states*) (*reg-Restr-Q*$_f$ $\mathcal{A}$)
have *ts*: *Inr cl-state* $\mid\notin\mid$ $\mathcal{Q}_r$ *?B Inr tr-state* $\mid\notin\mid$ $\mathcal{Q}_r$ *?B Inr fin-state* $\mid\notin\mid$ $\mathcal{Q}_r$ *?B*
by (*auto simp*: *fmap-states-reg-def*)
then have *Inr cl-state* $\mid\notin\mid$ *fin* (*fmap-states-reg Inl* (*reg-Restr-Q*$_f$ $\mathcal{A}$))
Inr fin-state $\mid\notin\mid$ *fin* (*fmap-states-reg Inl* (*reg-Restr-Q*$_f$ $\mathcal{A}$))
using *finj-Inl-Inr*(*1*) *fmap-states-reg-Restr-Q*$_f$*-fin* **by** *blast*+
from *gen-nhole-gctxt-closure-sound*[*OF ts this*] **show** *?thesis*
by (*simp add*: *nhole-ctxt-closure-reg-def Let-def*)
qed

5.1.14 Correctness of *gen-nhole-mctxt-closure-automaton*

lemmas *reflcl-over-nhole-mctxt-ta-simp* = *reflcl-over-nhole-mctxt-ta-def reflcl-over-nhole-ctxt-ta-def*

lemma *reflcl-rules-rhsD*:
$f\ ps \to q \mid\in\mid \mathit{reflcl\text{-}rules}\ \mathcal{F}\ q_c \Longrightarrow q = q_c$
by (*auto simp*: *reflcl-rules-def*)

lemma *reflcl-over-nhole-mctxt-ta-vars-term*:
assumes $q \mid\in\mid \mathit{ta\text{-}der}\ (\mathit{reflcl\text{-}over\text{-}nhole\text{-}mctxt\text{-}ta}\ Q\ \mathcal{F}\ q_c\ q_i\ q_f)\ t$
and $q_c \mid\notin\mid Q\ q \neq q_c\ q_f \neq q_c\ q_i \neq q_c$
shows *vars-term t* ≠ {} **using** *assms*
proof (*induction t arbitrary*: *q*)
case (*Fun f ts*)
let *?A* = *reflcl-over-nhole-mctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$
from *Fun*(*2*) **obtain** *p ps* **where** *rule*: *TA-rule f ps p* $\mid\in\mid$ *rules ?A*
length ps = *length ts* ∀ *i* < *length ts*. *ps* ! *i* $\mid\in\mid$ *ta-der ?A* (*ts* ! *i*)
$p = q \vee (p, q) \mid\in\mid (\mathit{eps}\ ?A)|^+|$
by *force*
from *rule*(*1*, *4*) *Fun*(*3*−) **have** $p \neq q_c$
by (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp dest*: *ftranclD*)
then have ∃ *i* < *length ts*. *ps* ! *i* ≠ q_c **using** *rule*(*1*, *2*) *Fun*(*4*−)
using *semantic-path-rules-fmemberD*

by (*force simp*: *reflcl-over-nhole-mctxt-ta-simp dest*: *reflcl-rules-rhsD*)
then show *?case* **using** *Fun*(*1*)[*OF nth-mem - Fun*(*3*) *- Fun*(*5*, *6*)] *rule*(*2*, *3*)
by *fastforce*
qed *auto*

lemma *reflcl-over-nhole-mctxt-ta-Fun*:
assumes q_f |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) *t t* ≠ *Var* q_f
and $q_f \neq q_c$ $q_f \neq q_i$
shows *is-Fun t* **using** *assms*
by (*cases t*) (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp dest*: *ftranclD2*)

lemma *rule-states-reflcl-rulesD*:
p |∈| *rule-states* (*reflcl-rules* $\mathcal{F}$ *q*) ⟹ *p* = *q*
by (*auto simp*: *reflcl-rules-def rule-states-def fset-of-list-elem*)

lemma *rule-states-semantic-path-rulesD*:
p |∈| *rule-states* (*semantic-path-rules* $\mathcal{F}$ q_c q_i q_f) ⟹ *p* = q_c ∨ *p* = q_i ∨ *p* = q_f
by (*auto simp*: *rule-states-def dest*!: *semantic-path-rules-fmemberD*)
(*metis in-fset-conv-nth length-list-update length-replicate nth-list-update nth-replicate*)

lemma $\mathcal{Q}$*-reflcl-over-nhole-mctxt-ta*:
$\mathcal{Q}$ (*reflcl-over-nhole-mctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) |⊆| *Q* |∪| {|q_c, q_i, q_f|}
by (*auto 0 0 simp*: *eps-states-def reflcl-over-nhole-mctxt-ta-simp* $\mathcal{Q}$*-def*
dest!: *rule-states-reflcl-rulesD rule-states-semantic-path-rulesD*)

lemma *reflcl-over-nhole-mctxt-ta-vars-term-subset-eq*:
assumes *q* |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) *t q* = q_f ∨ *q* = q_i
shows *vars-term t* ⊆ {q_c, q_i, q_f} ∪ *fset Q*
using *fresh-states-ta-der'-pres*[*OF - - assms*(*1*)[*unfolded ta-der-to-ta-der'*]] *assms*(*2*)
using *fsubsetD*[*OF* $\mathcal{Q}$*-reflcl-over-nhole-mctxt-ta*[*of Q* $\mathcal{F}$ q_c q_i q_f]]
by *auto*

lemma *sig-reflcl-over-nhole-mctxt-ta* [*simp*]:
ta-sig (*reflcl-over-nhole-mctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) = $\mathcal{F}$
by (*force simp*: *reflcl-over-nhole-mctxt-ta-simp reflcl-rules-def*
dest!: *semantic-path-rules-fmemberD intro*!: *ta-sig-fsubsetI*)

lemma *reflcl-over-nhole-mctxt-ta-aux-sound*:
assumes *funas-term t* ⊆ *fset* $\mathcal{F}$ *vars-term t* ⊆ *fset Q*
shows q_c |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta Q* $\mathcal{F}$ q_c q_i q_f) *t* **using** *assms*
proof (*induct t*)
case (*Var x*)
then show *?case*
by (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp fimage-iff*)
(*meson finsertI1 finsertI2 fr-into-trancl ftrancl-into-trancl rev-fimage-eqI*)
next
case (*Fun f ts*)
from *Fun*(*2*) **have** *TA-rule f* (*replicate* (*length ts*) q_c) q_c |∈| *rules* (*reflcl-over-nhole-mctxt-ta*

$Q\ \mathcal{F}\ q_c\ q_i\ q_f$)
by (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp reflcl-rules-def fimage-iff fBall-def split*: *prod.splits*)
then show *?case* **using** *Fun(1)*[*OF nth-mem*] *Fun(2−)*
by (*auto simp*: *SUP-le-iff*) (*metis length-replicate nth-replicate*)
qed

lemma *reflcl-over-nhole-mctxt-ta-sound*:
assumes *funas-term* $t \subseteq$ *fset* $\mathcal{F}$ *vars-term* $t \subseteq$ *fset* Q *vars-term* $t \neq \{\}$
shows (*is-Var* $t \longrightarrow q_i \mid\in\mid$ *ta-der* (*reflcl-over-nhole-mctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$) t) $\wedge$
(*is-Fun* $t \longrightarrow q_f \mid\in\mid$ *ta-der* (*reflcl-over-nhole-mctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$) t) **using** *assms*
proof (*induct* t)
case (*Fun* f *ts*)
let *?A* = *reflcl-over-nhole-mctxt-ta* $Q\ \mathcal{F}\ q_c\ q_i\ q_f$
from *Fun(4)* **obtain** i **where** *vars*: $i <$ *length ts vars-term* (*ts* ! i) $\neq \{\}$
by (*metis SUP-le-iff in-set-conv-nth subset-empty term.set(4)*)
consider (*v*) *is-Var* (*ts* ! i) | (*f*) *is-Fun* (*ts* ! i) **by** *blast*
then show *?case*
proof *cases*
case *v*
from *v Fun(1)*[*OF nth-mem*[*OF vars(1)*]] **have** $q_i \mid\in\mid$ *ta-der ?A* (*ts* ! i)
using *vars Fun(2−)* **by** (*auto simp*: *SUP-le-iff*)
moreover have f (*replicate* (*length ts*) q_c)[$i := q_i$] $\rightarrow q_f \mid\in\mid$ *rules ?A*
using *Fun(2) vars(1)*
by (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp semantic-path-rules-fmember*)
moreover have $j <$ *length ts* $\Longrightarrow q_c \mid\in\mid$ *ta-der ?A* (*ts* ! j) **for** j **using** *Fun(2−)*
by (*intro reflcl-over-nhole-mctxt-ta-aux-sound*) (*auto simp*: *SUP-le-iff*)
ultimately show *?thesis* **using** *vars*
by *auto* (*metis length-list-update length-replicate nth-list-update nth-replicate*)
next
case f
from f *Fun(1)*[*OF nth-mem*[*OF vars(1)*]] **have** $q_f \mid\in\mid$ *ta-der ?A* (*ts* ! i)
using *vars Fun(2−)* **by** (*auto simp*: *SUP-le-iff*)
moreover have f (*replicate* (*length ts*) q_c)[$i := q_f$] $\rightarrow q_f \mid\in\mid$ *rules ?A*
using *Fun(2) vars(1)*
by (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp semantic-path-rules-fmember*)
moreover have $j <$ *length ts* $\Longrightarrow q_c \mid\in\mid$ *ta-der ?A* (*ts* ! j) **for** j **using** *Fun(2−)*
by (*intro reflcl-over-nhole-mctxt-ta-aux-sound*) (*auto simp*: *SUP-le-iff*)
ultimately show *?thesis* **using** *vars*
by *auto* (*metis length-list-update length-replicate nth-list-update nth-replicate*)

qed
qed (*auto simp*: *reflcl-over-nhole-mctxt-ta-simp dest*!: *ftranclD2*)

lemma *gen-nhole-gmctxt-closure-lang*:
assumes $q_c \mid\notin\mid \mathcal{Q}\ \mathcal{A}$ **and** $q_i \mid\notin\mid \mathcal{Q}\ \mathcal{A}\ q_f \mid\notin\mid \mathcal{Q}\ \mathcal{A}$
and $q_c \mid\notin\mid Q\ q_f \neq q_c\ q_f \neq q_i\ q_i \neq q_c$

shows *gta-lang* $\{|q_f|\}$ (*gen-nhole-mctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f) =
{ *fill-gholes C ss* |
C ss. 0 < num-gholes C ∧ *num-gholes C = length ss* ∧ *C* ≠ *GMHole* ∧
funas-gmctxt C ⊆ *fset* $\mathcal{F}$ ∧ (∀ *i < length ss. ss* ! *i* ∈ *gta-lang Q* $\mathcal{A}$)}
(**is** *?Ls = ?Rs*)
proof −
let *?A = gen-nhole-mctxt-closure-automaton Q* $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f **let** *?B = reflcl-over-nhole-mctxt-ta Q* $\mathcal{F}$ q_c q_i q_f
interpret *sq*: *derivation-split ?A* $\mathcal{A}$ *?B*
using *assms* **unfolding** *derivation-split-def*
by (*auto simp*: *gen-nhole-mctxt-closure-automaton-def reflcl-over-nhole-mctxt-ta-def reflcl-over-nhole-ctxt-ta-def Q-def reflcl-rules-def dest*!: *semantic-path-rules-rhs*)
{fix *s* **assume** *s* ∈ *?Ls* **then obtain** *u* **where**
seq: *u* |∈| *ta-der*′ $\mathcal{A}$ (*term-of-gterm s*) *Var* q_f |∈| *ta-der*′*?B u*
by (*auto simp*: *ta-der-to-ta-der*′ *elim*!: *gta-langE dest*!: *sq.ta-der*′*-split*)
note *der = seq*(*2*)[*unfolded ta-der-to-ta-der*′[*symmetric*]]
have *vars-term u* ⊆ *fset Q vars-term u* ≠ {}
using *ta-der*′*-gterm-states*[*OF seq*(*1*)] *assms*(*1* − *3*)
using *reflcl-over-nhole-mctxt-ta-vars-term*[*OF der assms*(*4*) *assms*(*5*) *assms*(*5*) *assms*(*7*)]
using *reflcl-over-nhole-mctxt-ta-vars-term-subset-eq*[*OF der*]
by (*metis Un-insert-left insert-is-Un subset-iff subset-insert*)+
then have *vars*: ¬ *ground u i < length* (*ta-der*′*-target-args u*) ⟹ *ta-der*′*-target-args u* ! *i* |∈| *Q* **for** *i*
by (*auto simp*: *ta-der*′*-target-args-def split-vars-vars-term-list set-list-subset-nth-conv simp flip*: *set-vars-term-list*)
have *hole*: *ta-der*′*-target-mctxt u* ≠ *MHole* **using** *vars assms*(*3*−)
using *reflcl-over-nhole-mctxt-ta-Fun*[*OF der*]
using *ta-der*′*-mctxt-structure*(*1*, *3*)[*OF seq*(*1*)]
by *auto* (*metis fill-holes-MHole gterm-ta-der-states length-map lessI map-nth-eq-conv seq*(*1*) *ta-der-to-ta-der*′ *term.disc*(*1*))
let *?w* = λ *i. ta-der*′*-source-args u* (*term-of-gterm s*) ! *i*
have *s* ∈ *?Rs* **using** *seq*(*1*) *ta-der*′*-Var-funas*[*OF seq*(*2*)]
using *ground-ta-der-statesD*[*of ?w i ta-der*′*-target-args u* ! *i* $\mathcal{A}$ **for** *i*] *assms vars*
using *ta-der*′*-ground-mctxt-structure*[*OF seq*(*1*)] *hole*
by (*force simp*: *ground-gmctxt-of-mctxt-fill-holes*′ *ta-der*′*-source-args-term-of-gterm intro*!: *exI*[*of - gmctxt-of-mctxt* (*ta-der*′*-target-mctxt u*)] *exI*[*of - map gterm-of-term* (*ta-der*′*-source-args u* (*term-of-gterm s*))] *gta-langI*[*of ta-der*′*-target-args u* ! *i Q* $\mathcal{A}$ *gterm-of-term* (*?w i*) **for** *i*])**}**
then have *ls*: *?Ls* ⊆ *?Rs* **by** *blast*
{fix *t* **assume** *t* ∈ *?Rs*
then obtain *C ss* **where** *len*: *0 < num-gholes C num-gholes C = length ss C* ≠ *GMHole* **and**
gr-fun: *funas-gmctxt C* ⊆ *fset* $\mathcal{F}$ **and**
reachA: ∀ *i < length ss. ss* ! *i* ∈ *gta-lang Q* $\mathcal{A}$ **and**
const: *t = fill-gholes C ss* **by** *auto*
from *reachA* **obtain** *qs* **where** *states*: *length ss = length qs* ∀ *i < length qs.*

qs ! i |∈| Q |∩| $\mathcal{Q}$ $\mathcal{A}$
∀ i < *length qs*. qs ! i |∈| *ta-der* $\mathcal{A}$ ((*map term-of-gterm ss*) ! i)
using *Ex-list-of-length-P*[*of length ss* λ q i. q |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm* (*ss* ! i)) ∧ q |∈| Q]
by (*metis* (*full-types*) *finterI gta-langE gterm-ta-der-states length-map map-nth-eq-conv*)
have [*simp*]: *is-Fun* (*fill-holes* (*mctxt-of-gmctxt C*) (*map Var qs*)) ⟷ *True*
is-Var (*fill-holes* (*mctxt-of-gmctxt C*) (*map Var qs*)) ⟷ *False*
using *len*(*3*) **by** (*cases C*, *auto*)+
have q_f |∈| *ta-der ?A* (*fill-holes* (*mctxt-of-gmctxt C*) (*map term-of-gterm ss*))
using *reachA len gr-fun states*
using *reflcl-over-nhole-mctxt-ta-sound*[*of fill-holes* (*mctxt-of-gmctxt C*) (*map Var qs*)]
by (*intro sq.mctxt-const-to-ta-der*[*of mctxt-of-gmctxt C map term-of-gterm ss qs* q_f])
(*auto simp*: *funas-mctxt-of-gmctxt-conv set-list-subset-eq-nth-conv*
dest!: *in-set-idx*)
then have t ∈ *?Ls* **unfolding** *const*
by (*simp add*: *fill-holes-mctxt-of-gmctxt-to-fill-gholes gta-langI len*)**}**
then show *?thesis* **using** *ls* **by** *blast*
qed

lemma *nhole-gmctxt-closure-lang*:
$\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{ *fill-gholes C ss* | *C ss*. *num-gholes C* = *length ss* ∧ *0* < *num-gholes C* ∧ *C* ≠ *GMHole* ∧
funas-gmctxt C ⊆ *fset* $\mathcal{F}$ ∧ (∀ i < *length ss*. *ss* ! i ∈ $\mathcal{L}$ $\mathcal{A}$)}
(**is** *?Ls* = *?Rs*)
proof −
let *?B* = *fmap-states-reg* (*Inl* :: *'b* ⇒ *'b* + *cl-states*) (*reg-Restr-Q*$_f$ $\mathcal{A}$)
have *ts*: *Inr cl-state* |∉| $\mathcal{Q}_r$ *?B Inr tr-state* |∉| $\mathcal{Q}_r$ *?B Inr fin-state* |∉| $\mathcal{Q}_r$ *?B Inr cl-state* |∉| *fin ?B*
by (*auto simp*: *fmap-states-reg-def*)
have [*simp*]: *gta-lang* (*fin* (*fmap-states-reg Inl* (*reg-Restr-Q*$_f$ $\mathcal{A}$))) (*ta* (*fmap-states-reg Inl* (*reg-Restr-Q*$_f$ $\mathcal{A}$)))
= *gta-lang* (*fin* $\mathcal{A}$) (*ta* $\mathcal{A}$)
by (*metis* $\mathcal{L}$*-def* $\mathcal{L}$*-fmap-states-reg-Inl-Inr*(*1*) *reg-Rest-fin-states*)
from *gen-nhole-gmctxt-closure-lang*[*OF ts*] **show** *?thesis*
by (*auto simp add*: *nhole-mctxt-closure-reg-def gen-nhole-mctxt-closure-reg-def Let-def* $\mathcal{L}$*-def*)
qed

5.1.15 Correctness of *gen-mctxt-closure-reg* and *mctxt-closure-reg*

lemma *gen-gmctxt-closure-lang*:
assumes q_c |∉| $\mathcal{Q}$ $\mathcal{A}$ **and** q_i |∉| $\mathcal{Q}$ $\mathcal{A}$ q_f |∉| $\mathcal{Q}$ $\mathcal{A}$
and *disj*: q_c |∉| Q q_f ≠ q_c q_f ≠ q_i q_i ≠ q_c
shows *gta-lang* {|q_f, q_i|} (*gen-nhole-mctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f)
=
{ *fill-gholes C ss* |

C ss. 0 < num-gholes C ∧ num-gholes C = length ss ∧
funas-gmctxt C ⊆ fset ℱ ∧ (∀ i < length ss. ss ! i ∈ gta-lang Q 𝒜)}
(**is** *?Ls = ?Rs*)
proof −
let *?A = gen-nhole-mctxt-closure-automaton Q ℱ 𝒜 q_c q_i q_f* **let** *?B = reflcl-over-nhole-mctxt-ta Q ℱ q_c q_i q_f*
interpret *sq*: *derivation-split ?A 𝒜 ?B*
using *assms* **unfolding** *derivation-split-def*
by (*auto simp*: *gen-nhole-mctxt-closure-automaton-def reflcl-over-nhole-mctxt-ta-def reflcl-over-nhole-ctxt-ta-def 𝒬-def reflcl-rules-def dest*!: *semantic-path-rules-rhs*)
{fix *s* **assume** *s ∈ ?Ls* **then obtain** *u q* **where**
seq: *u |∈| ta-der′ 𝒜 (term-of-gterm s) Var q |∈| ta-der′?B u q = q_f ∨ q = q_i*
by (*auto simp*: *ta-der-to-ta-der′ elim*!: *gta-langE dest*!: *sq.ta-der′-split*)
have *vars-term u ⊆ fset Q vars-term u ≠ {}*
using *ta-der′-gterm-states*[*OF seq(1)*] *assms seq(3)*
using *reflcl-over-nhole-mctxt-ta-vars-term*[*OF seq(2)*[*unfolded ta-der-to-ta-der′*[*symmetric*]] *disj(1) - disj(2, 4)*]
using *reflcl-over-nhole-mctxt-ta-vars-term-subset-eq*[*OF seq(2)*[*unfolded ta-der-to-ta-der′*[*symmetric*]] *seq(3)*]
by (*metis Un-insert-left subsetD subset-insert sup-bot-left*)+
then have *vars*: *¬ ground u i < length (ta-der′-target-args u) ⟹ ta-der′-target-args u ! i |∈| Q* **for** *i*
by (*auto simp*: *ta-der′-target-args-def split-vars-vars-term-list set-list-subset-nth-conv simp flip*: *set-vars-term-list*)
let *?w = λ i. ta-der′-source-args u (term-of-gterm s) ! i*
have *s ∈ ?Rs* **using** *seq(1) ta-der′-Var-funas*[*OF seq(2)*]
using *ground-ta-der-statesD*[*of ?w i ta-der′-target-args u ! i 𝒜* **for** *i*] *assms vars*
using *ta-der′-ground-mctxt-structure*[*OF seq(1)*]
by (*force simp*: *ground-gmctxt-of-mctxt-fill-holes′ ta-der′-source-args-term-of-gterm intro*!: *exI*[*of - gmctxt-of-mctxt (ta-der′-target-mctxt u)*] *exI*[*of - map gterm-of-term (ta-der′-source-args u (term-of-gterm s))*] *gta-langI*[*of ta-der′-target-args u ! i Q 𝒜 gterm-of-term (?w i)* **for** *i*])**}**
then have *?Ls ⊆ ?Rs* **by** *blast*
moreover
{fix *t* **assume** *t ∈ ?Rs*
then obtain *C ss* **where** *len*: *0 < num-gholes C num-gholes C = length ss* **and**
gr-fun: *funas-gmctxt C ⊆ fset ℱ* **and**
reachA: *∀ i < length ss. ss ! i ∈ gta-lang Q 𝒜* **and**
const: *t = fill-gholes C ss* **by** *auto*
from *const* **have** *const′*: *term-of-gterm t = fill-holes (mctxt-of-gmctxt C) (map term-of-gterm ss)*
by (*simp add*: *fill-holes-mctxt-of-gmctxt-to-fill-gholes len(2)*)
from *reachA* **obtain** *qs* **where** *states*: *length ss = length qs ∀ i < length qs. qs ! i |∈| Q |∩| 𝒬 𝒜*
∀ i < length qs. qs ! i |∈| ta-der 𝒜 ((map term-of-gterm ss) ! i)
using *Ex-list-of-length-P*[*of length ss λ q i. q |∈| ta-der 𝒜 (term-of-gterm (ss*

```
! i)) ∧ q |∈| Q]
    by (metis (full-types) finterI gta-langE gterm-ta-der-states length-map map-nth-eq-conv)
   have C = GMHole ⟹ is-Var (fill-holes (mctxt-of-gmctxt C) (map Var qs)) =
True using len states(1)
       by (metis fill-holes-MHole length-map mctxt-of-gmctxt.simps(1) nth-map
num-gholes.simps(1) term.disc(1))
   then have hole: C = GMHole ⟹ q_i |∈| ta-der ?A (fill-holes (mctxt-of-gmctxt
C) (map term-of-gterm ss))
     using reachA len gr-fun states len
     using reflcl-over-nhole-mctxt-ta-sound[of fill-holes (mctxt-of-gmctxt C) (map
Var qs)]
     by (intro sq.mctxt-const-to-ta-der[of mctxt-of-gmctxt C map term-of-gterm ss
qs q_i])
        (auto simp: funas-mctxt-of-gmctxt-conv set-list-subset-eq-nth-conv
         dest!: in-set-idx)
    have C ≠ GMHole ⟹ is-Fun (fill-holes (mctxt-of-gmctxt C) (map Var qs))
= True
     by (cases C) auto
   then have C ≠ GMHole ⟹ q_f |∈| ta-der ?A (fill-holes (mctxt-of-gmctxt C)
(map term-of-gterm ss))
     using reachA len gr-fun states
     using reflcl-over-nhole-mctxt-ta-sound[of fill-holes (mctxt-of-gmctxt C) (map
Var qs)]
     by (intro sq.mctxt-const-to-ta-der[of mctxt-of-gmctxt C map term-of-gterm ss
qs q_f])
        (auto simp: funas-mctxt-of-gmctxt-conv set-list-subset-eq-nth-conv
         dest!: in-set-idx)
   then have t ∈ ?Ls using hole const' unfolding gta-lang-def gta-der-def
     by (metis (mono-tags, lifting) fempty-iff finsert-iff finterI mem-Collect-eq)}
 ultimately show ?thesis
   by (meson subsetI subset-antisym)
qed


lemma gmctxt-closure-lang:
 L (mctxt-closure-reg F A) =
   { fill-gholes C ss | C ss. num-gholes C = length ss ∧ 0 < num-gholes C ∧
     funas-gmctxt C ⊆ fset F ∧ (∀ i < length ss. ss ! i ∈ L A)}
 (is ?Ls = ?Rs)
proof −
 let ?B = fmap-states-reg (Inl :: 'b ⇒ 'b + cl-states) (reg-Restr-Q_f A)
 have ts: Inr cl-state |∉| Q_r ?B Inr tr-state |∉| Q_r ?B Inr fin-state |∉| Q_r ?B
   Inr cl-state |∉| fin ?B
   by (auto simp: fmap-states-reg-def)
 have [simp]: gta-lang (fin (fmap-states-reg Inl (reg-Restr-Q_f A))) (ta (fmap-states-reg
Inl (reg-Restr-Q_f A)))
   = gta-lang (fin A) (ta A)
   by (metis L-def L-fmap-states-reg-Inl-Inr(1) reg-Rest-fin-states)
 from gen-gmctxt-closure-lang[OF ts] show ?thesis
```

by (*auto simp add*: *mctxt-closure-reg-def gen-mctxt-closure-reg-def Let-def* $\mathcal{L}$*-def*)
qed

5.1.16 Correctness of *nhole-mctxt-reflcl-reg*

lemma *nhole-mctxt-reflcl-lang*:
$\mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) $\cup$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
proof −
let *?refl* = *Reg* {|*fin-clstate*|} (*refl-ta* $\mathcal{F}$ (*fin-clstate*))
{fix *t* **assume** *t* ∈ $\mathcal{L}$ *?refl* **then have** *t* ∈ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
using *reg-funas* **by** *fastforce***}**
moreover
{fix *t* **assume** *t* ∈ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) **then have** *t* ∈ $\mathcal{L}$ *?refl*
by (*simp add*: $\mathcal{L}$*-def gta-langI refl-ta-sound*)**}**
ultimately have ∗: $\mathcal{L}$ *?refl* = $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
by *blast*
show *?thesis* **unfolding** *nhole-mctxt-reflcl-reg-def* $\mathcal{L}$*-union* ∗ **by** *simp*
qed
declare *ta-union-def* [*simp del*]
end
theory *Type-Instances-Impl*
imports *Bot-Terms*
TA-Clousure-Const
Regular-Tree-Relations.Tree-Automata-Class-Instances-Impl
begin

6 Type class instantiations for the implementation

derive *linorder sum*
derive *linorder bot-term*
derive *linorder cl-states*

derive *compare bot-term*
derive *compare cl-states*

derive (*eq*) *ceq bot-term mctxt cl-states*

derive (*compare*) *ccompare bot-term cl-states*

derive (*rbt*) *set-impl bot-term cl-states*

derive (*no*) *cenum bot-term*

instantiation *cl-states* :: *cenum*
begin
abbreviation *cl-all-list* ≡ [*cl-state*, *tr-state*, *fin-state*, *fin-clstate*]
definition *cEnum-cl-states* :: (*cl-states list* × ((*cl-states* ⇒ *bool*) ⇒ *bool*) × ((*cl-states* ⇒ *bool*) ⇒ *bool*)) *option*
where *cEnum-cl-states* = *Some* (*cl-all-list*, (λ *P*. *list-all P cl-all-list*), (λ *P*.

list-ex P cl-all-list))
instance
apply *intro-classes* **apply** (*auto simp*: *cEnum-cl-states-def elim*!: *cl-states.induct*)
using *cl-states.exhaust* **apply** *blast*
by (*metis cl-states.exhaust*)
end

lemma *infinite-bot-term-UNIV*[*simp*, *intro*]: *infinite* (*UNIV* :: *'f bot-term set*)
proof −
fix *f* :: *'f*
let *?inj* = *λn. BFun f* (*replicate n Bot*)
have *inj ?inj* **unfolding** *inj-on-def* **by** *auto*
from *infinite-super*[*OF - range-inj-infinite*[*OF this*]]
show *?thesis* **by** *blast*
qed

lemma *finite-cl-states*: (*UNIV* :: *cl-states set*) = {*cl-state*, *tr-state*, *fin-state*, *fin-clstate*}
using *cl-states.exhaust*
by *auto*

instantiation *cl-states* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*cl-states*) *True*
definition *card-UNIV* = *Phantom*(*cl-states*) *4*
instance
by *intro-classes* (*simp-all add*: *card-UNIV-cl-states-def finite-UNIV-cl-states-def finite-cl-states*)
end

instantiation *bot-term* :: (*type*) *finite-UNIV*
begin
definition *finite-UNIV* = *Phantom*(*'a bot-term*) *False*
instance
by (*intro-classes*, *unfold finite-UNIV-bot-term-def*, *simp*)
end

instantiation *bot-term* :: (*compare*) *cproper-interval*
begin
definition *cproper-interval* = (*λ* (- :: *'a bot-term option*) - . *False*)
instance by (*intro-classes*, *auto*)
end

instantiation *cl-states* :: *cproper-interval*
begin

definition *cproper-interval-cl-states* :: *cl-states option* ⇒ *cl-states option* ⇒ *bool*
where *cproper-interval-cl-states x y* =
(*case ID CCOMPARE*(*cl-states*) *of Some f* ⇒

```
   (case x of None ⇒
     (case y of None ⇒ True | Some c ⇒ list-ex (λ x. (lt-of-comp f) x c) cl-all-list)
   | Some c ⇒
     (case y of None ⇒ list-ex (λ x. (lt-of-comp f) c x) cl-all-list
      | Some d ⇒ (filter (λ x. (lt-of-comp f) x d ∧ (lt-of-comp f) c x) cl-all-list) ≠
[])))

instance
proof (intro-classes)
  assume ass: (ID ccompare :: (cl-states ⇒ cl-states ⇒ order) option) ≠ None
  from ass obtain f where comp: (ID ccompare :: (cl-states ⇒ cl-states ⇒ order)
option) = Some f by auto
  let ?g = cproper-interval :: cl-states option ⇒ cl-states option ⇒ bool
  have [simp]: x < y ⟷ lt-of-comp f x y for x y
   by (metis ID-Some ccompare-cl-states-def comp compare-cl-states-def less-cl-states-def
option.sel)
  {fix c d x assume lt-of-comp f x d lt-of-comp f c x
    then have c < x x < d by simp-all}
  moreover
  {fix c d assume ∃ z. (c ::cl-states) < z ∧ z < d
    then obtain z where w: c < z ∧ z < d by blast
    then have z ∈ set cl-all-list by (cases z) auto
    moreover have lt-of-comp f z d ∧ lt-of-comp f c z using w comp
      by auto
    ultimately have filter (λx. lt-of-comp f x d ∧ lt-of-comp f c x) cl-all-list ≠ []
using w
      by auto}
  ultimately have filter (λx. lt-of-comp f x d ∧ lt-of-comp f c x) cl-all-list ≠ []
⟷ (∃ z. c < z ∧ z < d) for d c using comp
    unfolding filter-empty-conv by simp blast
  then have ?g (Some x) (Some y) = (∃ z. x < z ∧ z < y) for x y
    by (simp add: comp cproper-interval-cl-states-def)
 moreover have ?g None None = True by (simp add: comp cproper-interval-cl-states-def)
  moreover have ?g None (Some y) = (∃ z. z < y) for y using comp
    by (auto simp add: cproper-interval-cl-states-def ccompare-cl-states-def) (metis
cl-states.exhaust)+
  moreover have ?g (Some y) None = (∃ z. y < z) for y using comp
    by (auto simp add: cproper-interval-cl-states-def) (metis cl-states.exhaust)+
  ultimately show class.proper-interval cless ?g
    unfolding class.proper-interval-def comp
    by simp
qed
end

derive (rbt) mapping-impl cl-states
derive (rbt) mapping-impl bot-term

end
theory NF-Impl
```

imports *NF*
Type-Instances-Impl
begin

6.0.1 Implementation of normal form construction

fun *supteq-list* :: ('*f*, '*v*) *Term.term* ⇒ ('*f*, '*v*) *Term.term list*
where
supteq-list (*Var x*) = [*Var x*] |
supteq-list (*Fun f ts*) = *Fun f ts* # *concat* (*map supteq-list ts*)

fun *supt-list* :: ('*f*, '*v*) *Term.term* ⇒ ('*f*, '*v*) *Term.term list*
where
supt-list (*Var x*) = [] |
supt-list (*Fun f ts*) = *concat* (*map supteq-list ts*)

lemma *supteq-list* [*simp*]:
set (*supteq-list t*) = {*s*. *t* ⊵ *s*}
proof (*rule set-eqI*, *simp*)
fix *s*
show *s* ∈ *set*(*supteq-list t*) = (*t* ⊵ *s*)
proof (*induct t*, *simp add*: *supteq-var-imp-eq*)
case (*Fun f ss*)
show *?case*
proof (*cases Fun f ss* = *s*, (*auto*)[*1*])
case *False*
show *?thesis*
proof
assume *Fun f ss* ⊵ *s*
with *False* **have** *sup*: *Fun f ss* ▷ *s* **using** *supteq-supt-conv* **by** *auto*
obtain *C* **where** *C* ≠ □ **and** *Fun f ss* = *C*⟨*s*⟩ **using** *sup* **by** *auto*
then obtain *b D a* **where** *Fun f ss* = *Fun f* (*b* @ *D*⟨*s*⟩ # *a*) **by** (*cases C*, *auto*)
then have *D*: *D*⟨*s*⟩ ∈ *set ss* **by** *auto*
with *Fun*[*OF D*] *ctxt-imp-supteq*[*of D s*] **obtain** *t* **where** *t* ∈ *set ss* **and** *s* ∈ *set* (*supteq-list t*) **by** *auto*
then show *s* ∈ *set* (*supteq-list* (*Fun f ss*)) **by** *auto*
next
assume *s* ∈ *set* (*supteq-list* (*Fun f ss*))
with *False* **obtain** *t* **where** *t*: *t* ∈ *set ss* **and** *s* ∈ *set* (*supteq-list t*) **by** *auto*
with *Fun*[*OF t*] **have** *t* ⊵ *s* **by** *auto*
with *t* **show** *Fun f ss* ⊵ *s* **by** *auto*
qed
qed
qed
qed

lemma *supt-list-sound* [*simp*]:
set (*supt-list t*) = {*s*. *t* ▷ *s*}

by (*cases t*) *auto*

fun *mergeP-impl* **where**
mergeP-impl Bot t = *True*
| *mergeP-impl t Bot* = *True*
| *mergeP-impl* (*BFun f ss*) (*BFun g ts*) =
(*if f* = *g* ∧ *length ss* = *length ts then list-all* (λ (*x*, *y*). *mergeP-impl x y*) (*zip ss ts*) *else False*)

lemma [*simp*]: *mergeP-impl s Bot* = *True* **by** (*cases s*) *auto*

lemma [*simp*]: *mergeP-impl s t* ⟷ (*s*, *t*) ∈ *mergeP* (**is** *?LS* = *?RS*)
proof
show *?LS* ⟹ *?RS*
by (*induct rule*: *mergeP-impl.induct*, *auto split*: *if-splits intro*!: *step*)
(*smt* (*verit*) *length-zip list-all-length mergeP.step min-less-iff-conj nth-mem nth-zip old.prod.case*)
next
show *?RS* ⟹ *?LS* **by** (*induct rule*: *mergeP.induct*, *auto simp add*: *list-all-length*)
qed

fun *bless-eq-impl* **where**
bless-eq-impl Bot t = *True*
| *bless-eq-impl* (*BFun f ss*) (*BFun g ts*) =
(*if f* = *g* ∧ *length ss* = *length ts then list-all* (λ (*x*, *y*). *bless-eq-impl x y*) (*zip ss ts*) *else False*)
| *bless-eq-impl* - - = *False*

lemma [*simp*]: *bless-eq-impl s t* ⟷ (*s*, *t*) ∈ *bless-eq* (**is** *?RS* = *?LS*)
proof
show *?LS* ⟹ *?RS* **by** (*induct rule*: *bless-eq.induct*, *auto simp add*: *list-all-length*)
next
show *?RS* ⟹ *?LS*
by (*induct rule*: *bless-eq-impl.induct*, *auto split*: *if-splits intro*!: *bless-eq.step*)
(*metis* (*full-types*) *length-zip list-all-length min-less-iff-conj nth-mem nth-zip old.prod.case*)
qed

definition *psubt-bot-impl R* ≡ *remdups* (*map term-to-bot-term* (*concat* (*map supt-list R*)))
lemma *psubt-bot-impl*[*simp*]: *set* (*psubt-bot-impl R*) = *psubt-lhs-bot* (*set R*)
by (*induct R*, *auto simp*: *psubt-bot-impl-def*)

definition *states-impl R* = *List.insert Bot* (*map the* (*removeAll None*
(*closure-impl* (*lift-f-total mergeP-impl* (↑)) (*map Some* (*psubt-bot-impl R*)))))

lemma *states-impl* [*simp*]: *set* (*states-impl R*) = *states* (*set R*)
proof −

have [*simp*]: *lift-f-total mergeP-impl* (↑) = *lift-f-total* (λ *x y*. *mergeP-impl x y*) (↑) **by** *blast*
show *?thesis* **unfolding** *states-impl-def states-def*
using *lift-total.cl.closure-impl*
by (*simp add*: *lift-total.cl.pred-closure-lift-closure*)
qed

abbreviation *check-intance-lhs* **where**
check-intance-lhs qs f R ≡ *list-all* (λ *u*. ¬ *bless-eq-impl u* (*BFun f qs*)) *R*

definition *min-elem* **where**
min-elem s ss = (*let ts* = *filter* (λ *x*. *bless-eq-impl x s*) *ss in*
foldr (↑) *ts Bot*)

lemma *bound-impl* [*simp*, *code*]:
bound-max s (*set ss*) = *min-elem s ss*
proof −
have [*simp*]: {*y*. *lift-total.lifted-less-eq y* (*Some s*) ∧ *y* ∈ *Some* ‘ *set ss*} = *Some* ‘ {*x* ∈ *set ss*. *x* ≤$_b$ *s*}
by *auto*
then show *?thesis*
using *lift-total.supremum-impl*[*of filter* (λ *x*. *bless-eq-impl x s*) *ss*]
using *lift-total.supremum-smaller-exists-unique*[*of set ss s*]
by (*auto simp*: *min-elem-def Let-def lift-total.lift-ord.smaller-subset-def*)
qed

definition *nf-rule-impl* **where**
nf-rule-impl S R SR h = (*let* (*f*, *n*) = *h in*
let states = *List.n-lists n S in*
let nlhs-inst = *filter* (λ *qs*. *check-intance-lhs qs f R*) *states in*
map (λ *qs*. *TA-rule f qs* (*min-elem* (*BFun f qs*) *SR*)) *nlhs-inst*)

abbreviation *nf-rules-impl* **where**
nf-rules-impl R $\mathcal{F}$ ≡ *concat* (*map* (*nf-rule-impl* (*states-impl R*) (*map term-to-bot-term R*) (*psubt-bot-impl R*)) $\mathcal{F}$)

lemma *nf-rules-in-impl*:
assumes *TA-rule f qs q* |∈| *nf-rules* (*fset-of-list R*) (*fset-of-list* $\mathcal{F}$)
shows *TA-rule f qs q* |∈| *fset-of-list* (*nf-rules-impl R* $\mathcal{F}$)
proof −
have *funas*: (*f*, *length qs*) ∈ *set* $\mathcal{F}$ **and** *st*: *fset-of-list qs* |⊆| *fstates* (*fset-of-list R*)
and *nlhs*: ¬(∃ *s* ∈ (*set R*). *s*$^\bot$ ≤$_b$ *BFun f qs*)
and *min*: *q* = *bound-max* (*BFun f qs*) (*psubt-lhs-bot* (*set R*))
using *assms* **by** (*auto simp add*: *nf-rules-fmember simp flip*: *fset-of-list-elem*)
then have *st-impl*: *qs* |∈| *fset-of-list* (*List.n-lists* (*length qs*) (*states-impl R*))

by (*auto simp add*: *fset-of-list-elem subset-code*(*1*) *set-n-lists*
fset-of-list.rep-eq less-eq-fset.rep-eq fstates.rep-eq)
from *nlhs* **have** *nlhs-impl*: *check-intance-lhs qs f* (*map term-to-bot-term R*)
by (*auto simp*: *list.pred-set*)
have *min-impl*: *q* = *min-elem* (*BFun f qs*) (*psubt-bot-impl R*)
using *bound-impl min*
by (*auto simp flip*: *psubt-bot-impl*)
then show *?thesis* **using** *funas nlhs-impl funas st-impl* **unfolding** *nf-rule-impl-def*
by (*auto simp*: *fset-of-list-elem*)
qed

lemma *nf-rules-impl-in-rules*:
assumes *TA-rule f qs q* |∈| *fset-of-list* (*nf-rules-impl R* $\mathcal{F}$)
shows *TA-rule f qs q* |∈| *nf-rules* (*fset-of-list R*) (*fset-of-list* $\mathcal{F}$)
proof −
have *funas*: (*f*, *length qs*) ∈ *set* $\mathcal{F}$
and *st-impl*: *qs* |∈| *fset-of-list* (*List.n-lists* (*length qs*) (*states-impl R*))
and *nlhs-impl*: *check-intance-lhs qs f* (*map term-to-bot-term R*)
and *min*: *q* = *min-elem* (*BFun f qs*) (*psubt-bot-impl R*) **using** *assms*
by (*auto simp add*: *set-n-lists nf-rule-impl-def fset-of-list-elem*)
from *st-impl* **have** *st*: *fset-of-list qs* |⊆| *fstates* (*fset-of-list R*)
by (*force simp*: *set-n-lists fset-of-list-elem fstates.rep-eq fset-of-list.rep-eq*)
from *nlhs-impl* **have** *nlhs*: ¬(∃ *l* ∈ (*set R*). $l^{\bot} \leq_b$ *BFun f qs*)
by *auto* (*metis* (*no-types*, *lifting*) *Ball-set-list-all in-set-idx length-map nth-map nth-mem*)
have *q* = *bound-max* (*BFun f qs*) (*psubt-lhs-bot* (*set R*))
using *bound-impl min*
by (*auto simp flip*: *psubt-bot-impl*)
then show *?thesis* **using** *funas st nlhs*
by (*auto simp add*: *nf-rules-fmember fset-of-list-elem fset-of-list.rep-eq*)
qed

lemma *rule-set-eq*:
shows *nf-rules* (*fset-of-list R*) (*fset-of-list* $\mathcal{F}$) = *fset-of-list* (*nf-rules-impl R* $\mathcal{F}$)
(**is** *?Ls* = *?Rs*)
proof −
{fix *r* **assume** *r* |∈| *?Ls* **then have** *r* |∈| *?Rs*
using *nf-rules-in-impl*[**where** *?R* = *R* **and** *?*$\mathcal{F}$ = $\mathcal{F}$]
by (*cases r*) *auto***}**
moreover
{fix *r* **assume** *r* |∈| *?Rs* **then have** *r* |∈| *?Ls*
using *nf-rules-impl-in-rules*[**where** *?R* = *R* **and** *?*$\mathcal{F}$ = $\mathcal{F}$]
by (*cases r*) *auto***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *fstates-code*[*code*]:
fstates R = *fset-of-list* (*states-impl* (*sorted-list-of-fset R*))
by (*auto simp*: *fstates.rep-eq fset-of-list.rep-eq*)

lemma *nf-ta-code* [*code*]:
nf-ta R $\mathcal{F}$ = *TA* (*fset-of-list* (*nf-rules-impl* (*sorted-list-of-fset R*) (*sorted-list-of-fset* $\mathcal{F}$))) {||}
unfolding *nf-ta-def* **using** *rule-set-eq*[*of sorted-list-of-fset R sorted-list-of-fset* $\mathcal{F}$]
by (*intro TA-equalityI*) *auto*

end
theory *Context-Extensions*
imports *Regular-Tree-Relations.Ground-Ctxt*
Regular-Tree-Relations.Ground-Closure
Ground-MCtxt
begin

7 Multihole context and context closures over predicates

definition *gctxtex-onp* **where**
gctxtex-onp P $\mathcal{R}$ = {($C\langle s\rangle_G$, $C\langle t\rangle_G$) | *C s t*. *P C* ∧ (*s*, *t*) ∈ $\mathcal{R}$}

definition *gmctxtex-onp* **where**
gmctxtex-onp P $\mathcal{R}$ = {(*fill-gholes C ss*, *fill-gholes C ts*) | *C ss ts*.
num-gholes C = *length ss* ∧ *length ss* = *length ts* ∧ *P C* ∧ (∀ *i* < *length ts*. (*ss* ! *i* , *ts* ! *i*) ∈ $\mathcal{R}$)}

definition *compatible-p* **where**
compatible-p P Q ≡ (∀ *C*. *P C* ⟶ *Q* (*gmctxt-of-gctxt C*))

7.1 Elimination and introduction rules for the extensions

lemma *gctxtex-onpE* [*elim*]:
assumes (*s*, *t*) ∈ *gctxtex-onp P* $\mathcal{R}$
obtains *C u v* **where** *s* = $C\langle u\rangle_G$ *t* = $C\langle v\rangle_G$ *P C* (*u*, *v*) ∈ $\mathcal{R}$
using *assms* **unfolding** *gctxtex-onp-def* **by** *auto*

lemma *gctxtex-onp-neq-rootE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gctxtex-onp P* $\mathcal{R}$ **and** *f* ≠ *g*
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
proof −
obtain *C u v* **where** *GFun f ss* = $C\langle u\rangle_G$ *GFun g ts* = $C\langle v\rangle_G$ (*u*, *v*) ∈ $\mathcal{R}$
using *assms*(*1*) **by** *auto*
then show *?thesis* **using** *assms*(*2*)
by (*cases C*) *auto*

qed

lemma *gctxtex-onp-neq-lengthE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gctxtex-onp P* $\mathcal{R}$ **and** *length ss* ≠ *length ts*
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
proof −
obtain *C u v* **where** *GFun f ss* = $C\langle u\rangle_G$ *GFun g ts* = $C\langle v\rangle_G$ (*u*, *v*) ∈ $\mathcal{R}$
using *assms*(*1*) **by** *auto*
then show *?thesis* **using** *assms*(*2*)
by (*cases C*) *auto*
qed

lemma *gmctxtex-onpE* [*elim*]:
assumes (*s*, *t*) ∈ *gmctxtex-onp P* $\mathcal{R}$
obtains *C us vs* **where** *s* = *fill-gholes C us t* = *fill-gholes C vs num-gholes C* = *length us*
length us = *length vs P C* ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
using *assms* **unfolding** *gmctxtex-onp-def* **by** *auto*

lemma *gmctxtex-onpE2* [*elim*]:
assumes (*s*, *t*) ∈ *gmctxtex-onp P* $\mathcal{R}$
obtains *C us vs* **where** *s* =$_{Gf}$ (*C*, *us*) *t* =$_{Gf}$ (*C*, *vs*)
P C ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
using *gmctxtex-onpE*[*OF assms*] **by** (*metis eq-gfill.intros*)

lemma *gmctxtex-onp-neq-rootE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gmctxtex-onp P* $\mathcal{R}$ **and** *f* ≠ *g*
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
proof −
obtain *C us vs* **where** *GFun f ss* = *fill-gholes C us GFun g ts* = *fill-gholes C vs*
num-gholes C = *length us length us* = *length vs* ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
using *assms*(*1*) **by** *auto*
then show *?thesis* **using** *assms*(*2*)
by (*cases C*; *cases us*; *cases vs*) *auto*
qed

lemma *gmctxtex-onp-neq-lengthE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gmctxtex-onp P* $\mathcal{R}$ **and** *length ss* ≠ *length ts*
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
proof −
obtain *C us vs* **where** *GFun f ss* = *fill-gholes C us GFun g ts* = *fill-gholes C vs*
num-gholes C = *length us length us* = *length vs* ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
using *assms*(*1*) **by** *auto*
then show *?thesis* **using** *assms*(*2*)
by (*cases C*; *cases us*; *cases vs*) *auto*
qed

lemma *gmctxtex-onp-listE*:
assumes $\forall\ i < length\ ts.\ (ss\ !\ i,\ ts\ !\ i) \in gmctxtex\text{-}onp\ Q\ \mathcal{R}\ length\ ss = length\ ts$
obtains *Ds sss tss* **where** *length ts = length Ds length Ds = length sss length sss = length tss*
$\forall\ i < length\ tss.\ length\ (sss\ !\ i) = length\ (tss\ !\ i)\ \forall\ D \in set\ Ds.\ Q\ D$
$\forall\ i < length\ tss.\ ss\ !\ i =_{Gf} (Ds\ !\ i,\ sss\ !\ i)\ \forall\ i < length\ tss.\ ts\ !\ i =_{Gf} (Ds\ !\ i,\ tss\ !\ i)$
$\forall\ i < length\ (concat\ tss).\ (concat\ sss\ !\ i,\ concat\ tss\ !\ i) \in \mathcal{R}$
proof −
let *?P* $= \lambda\ W\ i.\ ss\ !\ i =_{Gf} (fst\ W,\ fst\ (snd\ W)) \wedge ts\ !\ i =_{Gf} (fst\ W,\ snd\ (snd\ W)) \wedge$
$Q\ (fst\ W) \wedge (\forall\ i < length\ (snd\ (snd\ W)).\ (fst\ (snd\ W)\ !\ i,\ snd\ (snd\ W)\ !\ i) \in \mathcal{R})$
have $\forall\ i < length\ ts.\ \exists\ x.\ ?P\ x\ i$ **using** *assms gmctxtex-onpE2*[*of ss ! i ts ! i Q* $\mathcal{R}$ **for** *i*]
by *auto metis*
from *Ex-list-of-length-P*[*OF this*] **obtain** *W* **where**
P: *length W = length ts* $\forall\ i < length\ ts.\ ?P\ (W\ !\ i)\ i$ **by** *blast*
define *Ds sss tss* **where** *Ds* $\equiv$ *map fst W* **and** *sss* $\equiv$ *map* $(fst \circ snd)$ *W* **and** *tss* $\equiv$ *map* $(snd \circ snd)$ *W*
from *P* **have** *len*: *length ts = length Ds length Ds = length sss length sss = length tss* **and**
pred: $\forall\ D \in set\ Ds.\ Q\ D$ **and**
split: $\forall\ i < length\ Ds.\ ss\ !\ i =_{Gf} (Ds\ !\ i,\ sss\ !\ i) \wedge ts\ !\ i =_{Gf} (Ds\ !\ i,\ tss\ !\ i)$**and**
rec: $\forall\, i < length\ Ds.\ \forall\ j < length\ (tss\ !\ i).\ (sss\ !\ i\ !\ j,\ tss\ !\ i\ !\ j) \in \mathcal{R}$
using *assms*(*2*) **by** (*auto simp*: *Ds-def sss-def tss-def dest*!: *in-set-idx*)
from *len split* **have** *inn*: $\forall\ i < length\ tss.\ length\ (sss\ !\ i) = length\ (tss\ !\ i)$
by *auto* (*metis eqgfE*(*2*))
from *inn len rec* **have** $\forall\ i < length\ (concat\ tss).\ (concat\ sss\ !\ i,\ concat\ tss\ !\ i) \in \mathcal{R}$
by (*intro concat-nth-nthI*) *auto*
then show ($\bigwedge$*Ds sss tss. length ts = length Ds* $\Longrightarrow$ *length Ds = length sss* $\Longrightarrow$ *length sss = length tss* $\Longrightarrow$
$\forall\, i{<}length\ tss.\ length\ (sss\ !\ i) = length\ (tss\ !\ i) \Longrightarrow \forall\, D{\in}set\ Ds.\ Q\ D \Longrightarrow$
$\forall\, i{<}length\ tss.\ ss\ !\ i =_{Gf} (Ds\ !\ i,\ sss\ !\ i) \Longrightarrow \forall\, i{<}length\ tss.\ ts\ !\ i =_{Gf} (Ds\ !\ i,\ tss\ !\ i) \Longrightarrow$
$\forall\, i{<}length\ (concat\ tss).\ (concat\ sss\ !\ i,\ concat\ tss\ !\ i) \in \mathcal{R} \Longrightarrow thesis) \Longrightarrow thesis$
using *pred split inn len* **by** *auto*
qed

lemma *gmctxtex-onp-doubleE* [*elim*]:
assumes $(s,\ t) \in gmctxtex\text{-}onp\ P\ (gmctxtex\text{-}onp\ Q\ \mathcal{R})$
obtains *C Ds ss ts us vs* **where** $s =_{Gf} (C,\ ss)\ t =_{Gf} (C,\ ts)\ P\ C\ \forall\ D \in set\ Ds.\ Q\ D$
num-gholes C = length Ds length Ds = length ss length ss = length ts length ts = length us length us = length vs

$\forall\ i < \mathit{length}\ \mathit{Ds}.\ \mathit{ss}\ !\ i =_{Gf} (\mathit{Ds}\ !\ i, \mathit{us}\ !\ i) \wedge \mathit{ts}\ !\ i =_{Gf} (\mathit{Ds}\ !\ i, \mathit{vs}\ !\ i)$
$\forall\ i < \mathit{length}\ \mathit{Ds}.\ \forall\ j < \mathit{length}\ (\mathit{vs}\ !\ i).\ (\mathit{us}\ !\ i\ !\ j, \mathit{vs}\ !\ i\ !\ j) \in \mathcal{R}$
proof −
from *gmctxtex-onpE2*[*OF assms*] **obtain** *C ss ts* **where**
split: $s =_{Gf} (C, ss)\ t =_{Gf} (C, ts)$ **and**
len: *num-gholes C* = *length ss length ss* = *length ts* **and**
pred: *P C* **and** *rec*: $\forall\ i < \mathit{length}\ \mathit{ts}.\ (\mathit{ss}\ !\ i, \mathit{ts}\ !\ i) \in \mathit{gmctxtex\text{-}onp}\ Q\ \mathcal{R}$
by (*metis eqgfE*(2))
let *?P* = $\lambda\ W\ i.\ \mathit{ss}\ !\ i =_{Gf} (\mathit{fst}\ W, \mathit{fst}\ (\mathit{snd}\ W)) \wedge \mathit{ts}\ !\ i =_{Gf} (\mathit{fst}\ W, \mathit{snd}\ (\mathit{snd}\ W)) \wedge$
$Q\ (\mathit{fst}\ W) \wedge (\forall\ i < \mathit{length}\ (\mathit{snd}\ (\mathit{snd}\ W)).\ (\mathit{fst}\ (\mathit{snd}\ W)\ !\ i, \mathit{snd}\ (\mathit{snd}\ W)\ !\ i) \in \mathcal{R})$
have $\forall\ i < \mathit{length}\ \mathit{ts}.\ \exists\ x.\ \mathit{?P}\ x\ i$ **using** *rec gmctxtex-onpE2*[*of ss* ! *i ts* ! *i Q* $\mathcal{R}$ **for** *i*]
by *auto metis*
from *Ex-list-of-length-P*[*OF this*] **obtain** *W* **where**
P: *length W* = *length ts* $\forall\ i < \mathit{length}\ \mathit{ts}.\ \mathit{?P}\ (W\ !\ i)\ i$ **by** *blast*
define *Ds us vs* **where** *Ds* ≡ *map fst W* **and** *us* ≡ *map* (*fst* ∘ *snd*) *W* **and** *vs* ≡ *map* (*snd* ∘ *snd*) *W*
from *P* **have** *len′*: *length Ds* = *length ss length ss* = *length ts length ts* = *length us length us* = *length vs* **and**
pred′: $\forall\ D \in \mathit{set}\ \mathit{Ds}.\ Q\ D$ **and**
split′: $\forall\ i < \mathit{length}\ \mathit{Ds}.\ \mathit{ss}\ !\ i =_{Gf} (\mathit{Ds}\ !\ i, \mathit{us}\ !\ i) \wedge \mathit{ts}\ !\ i =_{Gf} (\mathit{Ds}\ !\ i, \mathit{vs}\ !\ i)$**and**
rec′: $\forall i < \mathit{length}\ \mathit{Ds}.\ \forall\ j < \mathit{length}\ (\mathit{vs}\ !\ i).\ (\mathit{us}\ !\ i\ !\ j, \mathit{vs}\ !\ i\ !\ j) \in \mathcal{R}$
using *len* **by** (*auto simp*: *Ds-def us-def vs-def dest*!: *in-set-idx*)
from *len′ len* **have** *num-gholes C* = *length Ds* **by** *simp*
from *this split pred pred′ len′ split′ rec′ len*
show $(\bigwedge C\ ss\ ts\ Ds\ us\ vs.\ s =_{Gf} (C, ss) \Longrightarrow t =_{Gf} (C, ts) \Longrightarrow P\ C \Longrightarrow$
$\forall D{\in}\mathit{set}\ \mathit{Ds}.\ Q\ D \Longrightarrow \mathit{num\text{-}gholes}\ C = \mathit{length}\ \mathit{Ds} \Longrightarrow \mathit{length}\ \mathit{Ds} = \mathit{length}\ \mathit{ss} \Longrightarrow \mathit{length}\ \mathit{ss} = \mathit{length}\ \mathit{ts} \Longrightarrow$
$\mathit{length}\ \mathit{ts} = \mathit{length}\ \mathit{us} \Longrightarrow \mathit{length}\ \mathit{us} = \mathit{length}\ \mathit{vs} \Longrightarrow$
$\forall i{<}\mathit{length}\ \mathit{Ds}.\ \mathit{ss}\ !\ i =_{Gf} (\mathit{Ds}\ !\ i, \mathit{us}\ !\ i) \wedge \mathit{ts}\ !\ i =_{Gf} (\mathit{Ds}\ !\ i, \mathit{vs}\ !\ i) \Longrightarrow$
$\forall i{<}\mathit{length}\ \mathit{Ds}.\ \forall j{<}\mathit{length}\ (\mathit{vs}\ !\ i).\ (\mathit{us}\ !\ i\ !\ j, \mathit{vs}\ !\ i\ !\ j) \in \mathcal{R} \Longrightarrow \mathit{thesis}) \Longrightarrow \mathit{thesis}$
by *blast*
qed

lemma *gctxtex-onpI* [*intro*]:
assumes *P C* **and** $(s, t) \in \mathcal{R}$
shows $(C\langle s\rangle_G, C\langle t\rangle_G) \in \mathit{gctxtex\text{-}onp}\ P\ \mathcal{R}$
using *assms* **by** (*auto simp*: *gctxtex-onp-def*)

lemma *gmctxtex-onpI* [*intro*]:
assumes *P C* **and** *num-gholes C* = *length us* **and** *length us* = *length vs*
and $\forall\ i < \mathit{length}\ \mathit{vs}.\ (\mathit{us}\ !\ i, \mathit{vs}\ !\ i) \in \mathcal{R}$
shows (*fill-gholes C us*, *fill-gholes C vs*) ∈ *gmctxtex-onp P* $\mathcal{R}$
using *assms* **unfolding** *gmctxtex-onp-def*
by *force*

lemma *gmctxtex-onp-arg-monoI*:
assumes P *GMHole*
shows $\mathcal{R} \subseteq$ *gmctxtex-onp* P $\mathcal{R}$ **using** *assms*
proof (*intro subsetI*)
fix s **assume** *mem*: $s \in \mathcal{R}$
have $*$: (*fill-gholes GMHole* [*fst s*], *fill-gholes GMHole* [*snd s*]) $= s$ **by** *auto*
have (*fill-gholes GMHole* [*fst s*], *fill-gholes GMHole* [*snd s*]) $\in$ *gmctxtex-onp* P $\mathcal{R}$
by (*intro gmctxtex-onpI*) (*auto simp*: *assms mem*)
then show $s \in$ *gmctxtex-onp* P $\mathcal{R}$ **unfolding** $*$ **.**
qed

lemma *gmctxtex-onpI2* [*intro*]:
assumes P C **and** $s =_{Gf}$ (C, *ss*) $t =_{Gf}$ (C, *ts*)
and $\forall$ $i <$ *length ts*. (*ss* ! i, *ts* ! i) $\in \mathcal{R}$
shows (s, t) $\in$ *gmctxtex-onp* P $\mathcal{R}$
using *eqgfE*[*OF assms*(*2*)] *eqgfE*[*OF assms*(*3*)]
using *gmctxtex-onpI*[*of* P, *OF assms*(*1*) - - *assms*(*4*)]
by (*simp add*: ‹*num-gholes* C = *length ss*›)

lemma *gctxtex-onp-hold-cond* [*simp*]:
(s, t) $\in$ *gctxtex-onp* P $\mathcal{R}$ $\Longrightarrow$ *groot* $s \neq$ *groot* $t \Longrightarrow$ P $\Box_G$
(s, t) $\in$ *gctxtex-onp* P $\mathcal{R}$ $\Longrightarrow$ *length* (*gargs* s) $\neq$ *length* (*gargs* t) $\Longrightarrow$ P $\Box_G$
by (*auto elim*!: *gctxtex-onpE*, *case-tac* C; *auto*)+

7.2 Monotonicity rules for the extensions

lemma *gctxtex-onp-rel-mono*:
$\mathcal{L} \subseteq \mathcal{R} \Longrightarrow$ *gctxtex-onp* P $\mathcal{L} \subseteq$ *gctxtex-onp* P $\mathcal{R}$
unfolding *gctxtex-onp-def* **by** *auto*

lemma *gmctxtex-onp-rel-mono*:
$\mathcal{L} \subseteq \mathcal{R} \Longrightarrow$ *gmctxtex-onp* P $\mathcal{L} \subseteq$ *gmctxtex-onp* P $\mathcal{R}$
unfolding *gmctxtex-onp-def*
by *auto* (*metis subsetD*)

lemma *compatible-p-gctxtex-gmctxtex-subseteq* [*dest*]:
compatible-p P $Q \Longrightarrow$ *gctxtex-onp* P $\mathcal{R} \subseteq$ *gmctxtex-onp* Q $\mathcal{R}$
unfolding *compatible-p-def*
by (*auto simp*: *apply-gctxt-fill-gholes gmctxtex-onpI*)

lemma *compatible-p-mono1*:
$P \leq R \Longrightarrow$ *compatible-p* R $Q \Longrightarrow$ *compatible-p* P Q
unfolding *compatible-p-def* **by** *auto*

lemma *compatible-p-mono2*:
$Q \leq R \Longrightarrow$ *compatible-p* P $Q \Longrightarrow$ *compatible-p* P R
unfolding *compatible-p-def* **by** *auto*

lemma *gctxtex-onp-mono* [*intro*]:
$P \leq Q \Longrightarrow$ *gctxtex-onp* $P\ \mathcal{R} \subseteq$ *gctxtex-onp* $Q\ \mathcal{R}$
by *auto*

lemma *gctxtex-onp-mem*:
$P \leq Q \Longrightarrow (s,\ t) \in$ *gctxtex-onp* $P\ \mathcal{R} \Longrightarrow (s,\ t) \in$ *gctxtex-onp* $Q\ \mathcal{R}$
by *auto*

lemma *gmctxtex-onp-mono* [*intro*]:
$P \leq Q \Longrightarrow$ *gmctxtex-onp* $P\ \mathcal{R} \subseteq$ *gmctxtex-onp* $Q\ \mathcal{R}$
by (*auto elim*!: *gmctxtex-onpE*)

lemma *gmctxtex-onp-mem*:
$P \leq Q \Longrightarrow (s,\ t) \in$ *gmctxtex-onp* $P\ \mathcal{R} \Longrightarrow (s,\ t) \in$ *gmctxtex-onp* $Q\ \mathcal{R}$
by (*auto dest*!: *gmctxtex-onp-mono*)

lemma *gctxtex-eqI* [*intro*]:
$P = Q \Longrightarrow \mathcal{R} = \mathcal{L} \Longrightarrow$ *gctxtex-onp* $P\ \mathcal{R} =$ *gctxtex-onp* $Q\ \mathcal{L}$
by *auto*

lemma *gmctxtex-eqI* [*intro*]:
$P = Q \Longrightarrow \mathcal{R} = \mathcal{L} \Longrightarrow$ *gmctxtex-onp* $P\ \mathcal{R} =$ *gmctxtex-onp* $Q\ \mathcal{L}$
by *auto*

7.3 Relation swap and converse

lemma *swap-gctxtex-onp*:
gctxtex-onp P (*prod.swap* ‘ $\mathcal{R}$) = *prod.swap* ‘ *gctxtex-onp* $P\ \mathcal{R}$
by (*auto simp*: *gctxtex-onp-def image-def*) *force*+

lemma *swap-gmctxtex-onp*:
gmctxtex-onp P (*prod.swap* ‘ $\mathcal{R}$) = *prod.swap* ‘ *gmctxtex-onp* $P\ \mathcal{R}$
by (*auto simp*: *gmctxtex-onp-def image-def*) *force*+

lemma *converse-gctxtex-onp*:
$($*gctxtex-onp* $P\ \mathcal{R})^{-1} =$ *gctxtex-onp* $P\ (\mathcal{R}^{-1})$
by (*auto simp*: *gctxtex-onp-def*)

lemma *converse-gmctxtex-onp*:
$($*gmctxtex-onp* $P\ \mathcal{R})^{-1} =$ *gmctxtex-onp* $P\ (\mathcal{R}^{-1})$
by (*auto simp*: *gmctxtex-onp-def*) *force*+

7.4 Subset equivalence for context extensions over predicates

lemma *gctxtex-onp-closure-predI*:
assumes $\bigwedge C\ s\ t.\ P\ C \Longrightarrow (s,\ t) \in \mathcal{R} \Longrightarrow (C\langle s\rangle_G,\ C\langle t\rangle_G) \in \mathcal{R}$
shows *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$
using *assms* **by** *auto*

lemma *gmctxtex-onp-closure-predI*:
assumes $\bigwedge$ *C ss ts. P C* $\Longrightarrow$ *num-gholes C = length ss* $\Longrightarrow$ *length ss = length ts* $\Longrightarrow$
($\forall$ *i < length ts.* (*ss ! i*, *ts ! i*) $\in \mathcal{R}$) $\Longrightarrow$ (*fill-gholes C ss*, *fill-gholes C ts*) $\in \mathcal{R}$
shows *gmctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$
using *assms* **by** *auto*

lemma *gctxtex-onp-closure-predE*:
assumes *gctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$
shows $\bigwedge$ *C s t. P C* $\Longrightarrow$ (*s*, *t*) $\in \mathcal{R} \Longrightarrow$ ($C\langle s\rangle_G$, $C\langle t\rangle_G$) $\in \mathcal{R}$
using *assms* **by** *auto*

lemma *gctxtex-closure* [*intro*]:
P $\square_G \Longrightarrow \mathcal{R} \subseteq$ *gctxtex-onp P* $\mathcal{R}$
by (*auto simp*: *gctxtex-onp-def*) *force*

lemma *gmctxtex-closure* [*intro*]:
assumes *P GMHole*
shows $\mathcal{R} \subseteq$ (*gmctxtex-onp P* $\mathcal{R}$)
proof −
{fix *s t* **assume** (*s*, *t*) $\in \mathcal{R}$ **then have** (*s*, *t*) $\in$ *gmctxtex-onp P* $\mathcal{R}$
using *gmctxtex-onpI*[*of P GMHole* [*s*] [*t*]] *assms* **by** *auto***}**
then show *?thesis* **by** *auto*
qed

lemma *gctxtex-pred-cmp-subseteq*:
assumes $\bigwedge$ *C D. P C* $\Longrightarrow$ *Q D* $\Longrightarrow$ *Q* (*C* $\circ_{Gc}$ *D*)
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gctxtex-onp Q* $\mathcal{R}$
using *assms* **by** (*auto simp*: *gctxtex-onp-def*) (*metis ctxt-ctxt-compose*)

lemma *gctxtex-pred-cmp-subseteq2*:
assumes $\bigwedge$ *C D. P C* $\Longrightarrow$ *Q D* $\Longrightarrow$ *P* (*C* $\circ_{Gc}$ *D*)
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gctxtex-onp P* $\mathcal{R}$
using *assms* **by** (*auto simp*: *gctxtex-onp-def*) (*metis ctxt-ctxt-compose*)

lemma *gmctxtex-pred-cmp-subseteq*:
assumes $\bigwedge$ *C D. C* $\leq$ *D* $\Longrightarrow$ *P C* $\Longrightarrow$ ($\forall$ *Ds* $\in$ *set* (*sup-gmctxt-args C D*). *Q Ds*) $\Longrightarrow$ *Q D*
shows *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gmctxtex-onp Q* $\mathcal{R}$ (**is** *?Ls* $\subseteq$ *?Rs*)
proof −
{fix *s t* **assume** (*s*, *t*) $\in$ *?Ls*
then obtain *C Ds ss ts us vs* **where**
split: *s* $=_{Gf}$ (*C*, *ss*) *t* $=_{Gf}$ (*C*, *ts*) **and**
len: *num-gholes C = length Ds length Ds = length ss length ss = length ts*
length ts = length us length us = length vs **and**
pred: *P C* $\forall$ *D* $\in$ *set Ds. Q D* **and**
split': $\forall$ *i < length Ds. ss ! i* $=_{Gf}$ (*Ds ! i*, *us ! i*) $\wedge$ *ts ! i* $=_{Gf}$ (*Ds ! i*, *vs ! i*)
and

rec: $\forall i{<}length\ Ds.\ \forall j{<}length\ (vs\ !\ i).\ (us\ !\ i\ !\ j,\ vs\ !\ i\ !\ j) \in \mathcal{R}$
by *auto*
from *pred*(*2*) *assms*[*OF* - *pred*(*1*), *of fill-gholes-gmctxt C Ds*] *len*
have *P*: *Q* (*fill-gholes-gmctxt C Ds*)
by (*simp add*: *fill-gholes-gmctxt-less-eq*)
have *mem*: $\forall\ i < length\ (concat\ vs).\ (concat\ us\ !\ i,\ concat\ vs\ !\ i) \in \mathcal{R}$
using *rec split′ len*
by (*intro concat-nth-nthI*) (*auto*, *metis eqgfE*(*2*))
have $(s,\ t) \in ?Rs$ **using** *split′ split len*
by (*intro gmctxtex-onpI2*[*of Q*, *OF P* - - *mem*])
(*metis eqgfE*(*1*) *fill-gholes-gmctxt-sound*)+**}**
then show *?thesis* **by** *auto*
qed

lemma *gmctxtex-pred-cmp-subseteq2*:
assumes $\bigwedge\ C\ D.\ C \leq D \Longrightarrow P\ C \Longrightarrow (\forall\ Ds \in set\ (sup\text{-}gmctxt\text{-}args\ C\ D).\ Q\ Ds) \Longrightarrow P\ D$
shows *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gmctxtex-onp P* $\mathcal{R}$ (**is** *?Ls* $\subseteq$ *?Rs*)
proof −
{fix *s t* **assume** $(s,\ t) \in ?Ls$
then obtain *C Ds ss ts us vs* **where**
split: $s =_{Gf} (C,\ ss)\ t =_{Gf} (C,\ ts)$ **and**
len: *num-gholes C* = *length Ds length Ds* = *length ss length ss* = *length ts*
length ts = *length us length us* = *length vs* **and**
pred: *P C* $\forall$ *D* $\in$ *set Ds*. *Q D* **and**
split′: $\forall\ i < length\ Ds.\ ss\ !\ i =_{Gf} (Ds\ !\ i,\ us\ !\ i) \wedge ts\ !\ i =_{Gf} (Ds\ !\ i,\ vs\ !\ i)$
and
rec: $\forall i{<}length\ Ds.\ \forall j{<}length\ (vs\ !\ i).\ (us\ !\ i\ !\ j,\ vs\ !\ i\ !\ j) \in \mathcal{R}$
by *auto*
from *pred*(*2*) *assms*[*OF* - *pred*(*1*), *of fill-gholes-gmctxt C Ds*] *len*
have *P*: *P* (*fill-gholes-gmctxt C Ds*)
by (*simp add*: *fill-gholes-gmctxt-less-eq*)
have *mem*: $\forall\ i < length\ (concat\ vs).\ (concat\ us\ !\ i,\ concat\ vs\ !\ i) \in \mathcal{R}$ **using** *rec split′ len*
by (*intro concat-nth-nthI*) (*auto*, *metis eqgfE*(*2*))
have $(s,\ t) \in ?Rs$ **using** *split′ split len*
by (*intro gmctxtex-onpI2*[*of P*, *OF P* - - *mem*])
(*metis eqgfE*(*1*) *fill-gholes-gmctxt-sound*)+**}**
then show *?thesis* **by** *auto*
qed

lemma *gctxtex-onp-idem* [*simp*]:
assumes *P* $\square_G$ **and** $\bigwedge\ C\ D.\ P\ C \Longrightarrow Q\ D \Longrightarrow Q\ (C \circ_{Gc} D)$
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) = *gctxtex-onp Q* $\mathcal{R}$ (**is** *?Ls* = *?Rs*)
by (*simp add*: *assms gctxtex-pred-cmp-subseteq gctxtex-closure subset-antisym*)

lemma *gctxtex-onp-idem2* [*simp*]:
assumes *Q* $\square_G$ **and** $\bigwedge\ C\ D.\ P\ C \Longrightarrow Q\ D \Longrightarrow P\ (C \circ_{Gc} D)$
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) = *gctxtex-onp P* $\mathcal{R}$ (**is** *?Ls* = *?Rs*)

using *gctxtex-pred-cmp-subseteq2*[*of P Q*, *OF assms*(*2*)]
using *gctxtex-closure*[*of Q*, *OF assms*(*1*)] *in-mono*
by *auto fastforce*

lemma *gmctxtex-onp-idem* [*simp*]:
assumes *P GMHole*
and $\bigwedge$ *C D*. *C* $\leq$ *D* $\Longrightarrow$ *P C* $\Longrightarrow$ ($\forall$ *Ds* $\in$ *set* (*sup-gmctxt-args C D*). *Q Ds*) $\Longrightarrow$ *Q D*
shows *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$) = *gmctxtex-onp Q* $\mathcal{R}$
using *gmctxtex-pred-cmp-subseteq*[*of P Q* $\mathcal{R}$] *gmctxtex-closure*[*of P*] *assms*
by *auto*

7.5 *gmctxtex-onp* subset equivalence *gctxtex-onp* transitive closure

The following definition demands that if we arbitrarily fill a multihole context C with terms induced by signature F such that one hole remains then the predicate Q holds

definition *gmctxt-p-inv C* $\mathcal{F}$ *Q* $\equiv$ ($\forall$ *D*. *gmctxt-closing C D* $\longrightarrow$ *num-gholes D* = *1* $\longrightarrow$ *funas-gmctxt D* $\subseteq$ $\mathcal{F}$
$\longrightarrow$ *Q* (*gctxt-of-gmctxt D*))

lemma *gmctxt-p-invE*:
gmctxt-p-inv C $\mathcal{F}$ *Q* $\Longrightarrow$ *C* $\leq$ *D* $\Longrightarrow$ *ghole-poss D* $\subseteq$ *ghole-poss C* $\Longrightarrow$ *num-gholes D* = *1* $\Longrightarrow$
funas-gmctxt D $\subseteq$ $\mathcal{F}$ $\Longrightarrow$ *Q* (*gctxt-of-gmctxt D*)
unfolding *gmctxt-closing-def gmctxt-p-inv-def*
using *less-eq-gmctxt-prime* **by** *blast*

lemma *gmctxt-closing-gmctxt-p-inv-comp*:
gmctxt-closing C D $\Longrightarrow$ *gmctxt-p-inv C* $\mathcal{F}$ *Q* $\Longrightarrow$ *gmctxt-p-inv D* $\mathcal{F}$ *Q*
unfolding *gmctxt-closing-def gmctxt-p-inv-def*
by *auto* (*meson less-eq-gmctxt-prime order-trans*)

lemma *GMHole-gmctxt-p-inv-GHole* [*simp*]:
gmctxt-p-inv GMHole $\mathcal{F}$ *Q* $\Longrightarrow$ *Q* $\Box_G$
by (*auto dest*: *gmctxt-p-invE*)

lemma *gmctxtex-onp-gctxtex-onp-trancl*:
assumes *sig*: $\bigwedge$ *C*. *P C* $\Longrightarrow$ *0* < *num-gholes C* $\wedge$ *funas-gmctxt C* $\subseteq$ $\mathcal{F}$ $\mathcal{R}$ $\subseteq$ $\mathcal{T}_G$ $\mathcal{F}$ $\times$ $\mathcal{T}_G$ $\mathcal{F}$
and $\bigwedge$ *C*. *P C* $\Longrightarrow$ *gmctxt-p-inv C* $\mathcal{F}$ *Q*
shows *gmctxtex-onp P* $\mathcal{R}$ $\subseteq$ (*gctxtex-onp Q* $\mathcal{R}$)$^+$
proof
fix *s t* **assume** (*s*, *t*) $\in$ *gmctxtex-onp P* $\mathcal{R}$
then obtain *C ss ts* **where**
split: *s* = *fill-gholes C ss t* = *fill-gholes C ts* **and**
inv: *num-gholes C* = *length ss num-gholes C* = *length ts* **and**

pred: $P\ C$ **and** *rec*: $\forall\ i < \textit{length ts}.\ (\textit{ss} ! i, \textit{ts} ! i) \in \mathcal{R}$
by *auto*
from *pred* **have** $0 < \textit{num-gholes}\ C\ \textit{funas-gmctxt}\ C \subseteq \mathcal{F}$ **using** *sig* **by** *auto*
from *this inv assms*(3)[*OF pred*] *rec* **show** $(s, t) \in (\textit{gctxtex-onp}\ Q\ \mathcal{R})^+$ **unfolding** *split*
proof (*induct num-gholes C arbitrary*: *C ss ts*)
case (*Suc x*) **note** *IS* = *this* **then show** *?case*
proof (*cases C*)
case *GMHole* **then show** *?thesis*
using *IS*(2−) *gctxtex-closure* **unfolding** *gmctxt-p-inv-def gmctxt-closing-def*
by (*metis One-nat-def fill-gholes-GMHole gctxt-of-gmctxt.simps*(1)
gmctxt-order-bot.bot.extremum-unique less-eq-gmctxt-prime num-gholes.simps(1)
r-into-trancl′ subsetD subsetI)
next
case [*simp*]: (*GMFun f Cs*) **note** *IS* = *IS*[*unfolded GMFun*]
let *?rep* = $\lambda\ x.$ *replicate* (*num-gholes* (*GMFun f Cs*) − 1) *x*
let *?Ds1* = *?rep GMHole* @ [*gmctxt-of-gterm* (*last ss*)]
let *?Ds2* = *map gmctxt-of-gterm* (*butlast ts*) @ [*GMHole*]
let *?D1* = *fill-gholes-gmctxt* (*GMFun f Cs*) *?Ds1*
let *?D2* = *fill-gholes-gmctxt* (*GMFun f Cs*) *?Ds2*
have *holes*: *num-gholes* (*GMFun f Cs*) = *length ?Ds1 num-gholes* (*GMFun f Cs*) = *length ?Ds2*
using *IS*(2, 5, 6) **by** *auto*
from *holes*(2) **have** [*simp*]: *num-gholes ?D2* = *Suc 0*
by (*auto simp*: *num-gholes-fill-gholes-gmctxt simp del*: *fill-gholes-gmctxt.simps*)
from *holes*(1) **have** *h*: *x* = *num-gholes ?D1* **using** *IS*(2)
by (*auto simp*: *num-gholes-fill-gholes-gmctxt simp del*: *fill-gholes-gmctxt.simps*)
from *holes* **have** *less*: *GMFun f Cs* ≤ *?D1 GMFun f Cs* ≤ *?D2*
by (*auto simp del*: *fill-gholes-gmctxt.simps intro*: *fill-gholes-gmctxt-less-eq*)
have *ghole-poss ?D1* ⊆ *ghole-poss* (*GMFun f Cs*) **using** *less*(1) *IS*(2, 3)
by (*intro fill-gholes-gmctxt-ghole-poss-subseteq*) (*auto simp*: *nth-append*)
then have *ext*: *gmctxt-p-inv ?D1* $\mathcal{F}$ *Q* **using** *less*(1) *IS*(7)
using *gmctxt-closing-def gmctxt-closing-gmctxt-p-inv-comp less-eq-gmctxt-prime*
by *blast*
have *split-last-D1-ss*: *fill-gholes C* (*butlast ts* @ [*last ss*]) $=_{Gf}$ (*?D1*, *concat* (*map* (λ *x*. [*x*]) (*butlast ts*) @ [[]]))
using *holes*(1) *IS*(2, 5, 6) **unfolding** *GMFun*
by (*intro fill-gholes-gmctxt-sound*)
(*auto simp*: *nth-append eq-gfill.simps nth-butlast*)
have *split-last-D2-ss*: *fill-gholes C* (*butlast ts* @ [*last ss*]) $=_{Gf}$ (*?D2*, *concat* (*?rep* [] @ [[*last ss*]]))
using *holes*(2) *IS*(2, 5, 6) **unfolding** *GMFun*
by (*intro fill-gholes-gmctxt-sound*) (*auto simp*: *nth-append*
eq-gfill.simps nth-butlast last-conv-nth intro: *last-nthI*)
have *split-last-ts*: *fill-gholes C ts* $=_{Gf}$ (*?D2*, *concat* (*?rep* [] @ [[*last ts*]]))
using *holes*(2) *IS*(2, 5, 6) **unfolding** *GMFun*
by (*intro fill-gholes-gmctxt-sound*) (*auto simp*: *nth-append*
eq-gfill.simps nth-butlast last-conv-nth intro: *last-nthI*)
from *eqgfE*[*OF split-last-ts*] **have** *last-eq*: *fill-gholes C ts* = *fill-gholes ?D2*

[*last ts*]
by (*auto simp del*: *fill-gholes.simps fill-gholes-gmctxt.simps*)
have *trans*: *fill-gholes ?D1* (*butlast ts*) = *fill-gholes ?D2* [*last ss*]
using *eqgfE*[*OF split-last-D1-ss*] *eqgfE*[*OF split-last-D2-ss*]
by (*auto simp del*: *fill-gholes.simps fill-gholes-gmctxt.simps*)
have *ghole-poss ?D2* ⊆ *ghole-poss* (*GMFun f Cs*) **using** *less*(*2*) *IS*(*2*, *3*, *6*)
by (*intro fill-gholes-gmctxt-ghole-poss-subseteq*) (*auto simp*: *nth-append*)
then have *Q* (*gctxt-of-gmctxt ?D2*) **using** *less*(*2*)
using *subsetD*[*OF assms*(*2*)] *IS*(*2* − *6*, *8*) *holes*(*2*)
by (*intro gmctxt-p-invE*[*OF IS*(*7*)])
(*auto simp del*: *fill-gholes-gmctxt.simps simp*: *num-gholes-fill-gholes-gmctxt in-set-conv-nth* $\mathcal{T}_G$*-equivalent-def nth-butlast*, *metis less-SucI subsetD*)
from *gctxtex-onpI*[*of Q - last ss last ts* $\mathcal{R}$, *OF this*] *IS*(*2*, *3*, *5*, *6*, *8*)
have *mem*: (*fill-gholes ?D2* [*last ss*], *fill-gholes ?D2* [*last ts*]) ∈ *gctxtex-onp Q* $\mathcal{R}$
using *fill-gholes-apply-gctxt*[*of ?D2 last ss*]
using *fill-gholes-apply-gctxt*[*of ?D2 last ts*]
by (*auto simp del*: *gctxt-of-gmctxt.simps fill-gholes-gmctxt.simps fill-gholes.simps*)
(*metis IS*(*2*) *IS*(*3*) *append-butlast-last-id diff-Suc-1 length-butlast length-greater-0-conv lessI nth-append-length*)
show *?thesis*
proof (*cases x*)
case *0* **then show** *?thesis* **using** *mem IS*(*2* − *6*) *eqgfE*[*OF split-last-D2-ss*] *last-eq*
by (*cases ss*; *cases ts*)
(*auto simp del*: *gctxt-of-gmctxt.simps fill-gholes-gmctxt.simps fill-gholes.simps*, *metis IS*(*3*, *5*) *length-0-conv less-not-refl*)
next
case [*simp*]: (*Suc nat*)
have *fill-gholes C ss* =$_{Gf}$ (*?D1*, *concat* (*map* (λ *x*. [*x*]) (*butlast ss*) @ [[]]))
using *holes*(*1*) *IS*(*2*, *5*, *6*) **unfolding** *GMFun*
by (*intro fill-gholes-gmctxt-sound*)
(*auto simp del*: *fill-gholes-gmctxt.simps fill-gholes.simps simp*: *nth-append nth-butlast eq-gfill.intros last-nthI*)
from *eqgfE*[*OF this*] **have** *l*: *fill-gholes C ss* = *fill-gholes ?D1* (*butlast ss*)
by (*auto simp del*: *fill-gholes-gmctxt.simps fill-gholes.simps*)
from *IS*(*1*)[*OF h - - - - ext*, *of butlast ss butlast ts*] *IS*(*2*−) *holes*(*2*) *h assms*(*2*)
have (*fill-gholes ?D1* (*butlast ss*), *fill-gholes ?D1* (*butlast ts*)) ∈ (*gctxtex-onp Q* $\mathcal{R}$)$^+$
by (*auto simp del*: *gctxt-of-gmctxt.simps fill-gholes-gmctxt.simps fill-gholes.simps simp*: $\mathcal{T}_G$*-equivalent-def*)
(*smt* (*verit*) *Suc.prems*(*1*) *Suc.prems*(*4*) *diff-Suc-1 last-conv-nth length-butlast length-greater-0-conv lessI less-SucI mem-Sigma-iff nth-butlast sig*(*2*) *subset-iff* $\mathcal{T}_G$*-funas-gterm-conv*)
then have (*fill-gholes ?D1* (*butlast ss*), *fill-gholes ?D2* [*last ts*]) ∈ (*gctxtex-onp Q* $\mathcal{R}$)$^+$
using *mem* **unfolding** *trans*
by (*auto simp del*: *gctxt-of-gmctxt.simps fill-gholes-gmctxt.simps fill-gholes.simps*)

```
      then show ?thesis unfolding last-eq l
        by (auto simp del: fill-gholes-gmctxt.simps fill-gholes.simps)
     qed
    qed
  qed auto
qed

lemma gmctxtex-onp-gctxtex-onp-rtrancl:
  assumes sig: ⋀ C. P C ⟹ funas-gmctxt C ⊆ ℱ ℛ ⊆ 𝒯_G ℱ × 𝒯_G ℱ
    and ⋀ C D. P C ⟹ gmctxt-p-inv C ℱ Q
  shows gmctxtex-onp P ℛ ⊆ (gctxtex-onp Q ℛ)*
proof
  fix s t assume (s, t) ∈ gmctxtex-onp P ℛ
  then obtain C ss ts where
    split: s = fill-gholes C ss t = fill-gholes C ts and
    inv: num-gholes C = length ss num-gholes C = length ts and
    pred: P C and rec: ∀ i < length ts. (ss ! i, ts ! i) ∈ ℛ
    by auto
  then show (s, t) ∈ (gctxtex-onp Q ℛ)*
  proof (cases num-gholes C)
    case 0 then show ?thesis using inv unfolding split
      by auto
  next
    case (Suc nat)
    from split inv pred rec assms
    have (s, t) ∈ gmctxtex-onp (λ C. P C ∧ 0 < num-gholes C) ℛ unfolding split
      by auto (metis (no-types, lifting) Suc gmctxtex-onpI zero-less-Suc)
   moreover have gmctxtex-onp (λ C. P C ∧ 0 < num-gholes C) ℛ ⊆ (gctxtex-onp
Q ℛ)+ using assms
      by (intro gmctxtex-onp-gctxtex-onp-trancl) auto
    ultimately show ?thesis by auto
  qed
qed

lemma rtrancl-gmctxtex-onp-rtrancl-gctxtex-onp-eq:
  assumes sig: ⋀ C. P C ⟹ funas-gmctxt C ⊆ ℱ ℛ ⊆ 𝒯_G ℱ × 𝒯_G ℱ
    and ⋀ C D. P C ⟹ gmctxt-p-inv C ℱ Q
    and compatible-p Q P
  shows (gmctxtex-onp P ℛ)* = (gctxtex-onp Q ℛ)* (is ?Ls* = ?Rs*)
proof −
  from assms(4) have ?Rs ⊆ ?Ls by auto
  then have ?Rs* ⊆ ?Ls*
    by (simp add: rtrancl-mono)
  moreover from gmctxtex-onp-gctxtex-onp-rtrancl[OF assms(1 − 3), of P]
  have ?Ls* ⊆ ?Rs*
    by (simp add: rtrancl-subset-rtrancl)
  ultimately show ?thesis by blast
qed
```

7.6 Extensions to reflexive transitive closures

lemma *gctxtex-onp-substep-trancl*:
assumes *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$
shows *gctxtex-onp* $P\ (\mathcal{R}^+) \subseteq \mathcal{R}^+$
proof −
{fix $s\ t$ **assume** $(s, t) \in$ *gctxtex-onp* $P\ (\mathcal{R}^+)$
then obtain $C\ u\ v$ **where** *rec*: $(u, v) \in \mathcal{R}^+\ P\ C$ **and** t: $s = C\langle u\rangle_G\ t = C\langle v\rangle_G$
by *auto*
from *rec* **have** $(s, t) \in \mathcal{R}^+$ **unfolding** t
proof (*induct*)
case (*base* y)
then show *?case* **using** *assms* **by** *auto*
next
case (*step* $y\ z$)
from *assms step*(*2*, *4*) **have** $(C\langle y\rangle_G, C\langle z\rangle_G) \in \mathcal{R}$ **by** *auto*
then show *?case* **using** *step* **by** *auto*
qed}
then show *?thesis* **by** *auto*
qed

lemma *gctxtex-onp-substep-rtrancl*:
assumes *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$
shows *gctxtex-onp* $P\ (\mathcal{R}^*) \subseteq \mathcal{R}^*$
using *gctxtex-onp-substep-trancl*[*OF assms*]
by (*smt* (*verit*) *gctxtex-onpE gctxtex-onpI rtrancl-eq-or-trancl subrelI subset-eq*)

lemma *gctxtex-onp-substep-trancl-diff-pred* [*intro*]:
assumes $\bigwedge\ C\ D.\ P\ C \Longrightarrow Q\ D \Longrightarrow Q\ (D \circ_{Gc} C)$
shows *gctxtex-onp* $Q\ ((\textit{gctxtex-onp}\ P\ \mathcal{R})^+) \subseteq (\textit{gctxtex-onp}\ Q\ \mathcal{R})^+$
proof
fix $s\ t$ **assume** $(s, t) \in$ *gctxtex-onp* $Q\ ((\textit{gctxtex-onp}\ P\ \mathcal{R})^+)$
from *gctxtex-onpE*[*OF this*] **obtain** $C\ u\ v$ **where**
$*$: $s = C\langle u\rangle_G\ t = C\langle v\rangle_G$ **and** *inv*: $Q\ C$ **and** *mem*: $(u, v) \in (\textit{gctxtex-onp}\ P\ \mathcal{R})^+$
by *blast*
show $(s, t) \in (\textit{gctxtex-onp}\ Q\ \mathcal{R})^+$ **using** *mem* $*$ *inv*
proof (*induct arbitrary*: $s\ t$)
case (*base* y)
then show *?case* **using** *assms*
by (*auto elim*!: *gctxtex-onpE intro*!: *r-into-trancl*) (*metis ctxt-ctxt-compose gctxtex-onpI*)
next
case (*step* $y\ z$)
from *step*(*2*) **have** $(C\langle y\rangle_G, C\langle z\rangle_G) \in$ *gctxtex-onp* $Q\ \mathcal{R}$
using *assms*[*OF - step*(*6*)]
by (*auto elim*!: *gctxtex-onpE*) (*metis ctxt-ctxt-compose gctxtex-onpI*)
then show *?case* **using** *step*(*3*)[*of s* $C\langle y\rangle_G$] *step*(*1*, *2*, *4* −)
by *auto*
qed
qed

lemma *gctxtcl-pres-trancl*:
assumes $(s, t) \in \mathcal{R}^+$ **and** *gctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$ **and** *P C*
shows $(C\langle s\rangle_G, C\langle t\rangle_G) \in \mathcal{R}^+$
using *gctxtex-onp-substep-trancl* [*OF assms*(*2*)] *assms*(*1*, *3*)
by *auto*

lemma *gctxtcl-pres-rtrancl*:
assumes $(s, t) \in \mathcal{R}^*$ **and** *gctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$ **and** *P C*
shows $(C\langle s\rangle_G, C\langle t\rangle_G) \in \mathcal{R}^*$
using *assms*(*1*) *gctxtcl-pres-trancl*[*OF - assms*(*2*, *3*)]
unfolding *rtrancl-eq-or-trancl*
by (*cases s* = *t*) *auto*

lemma *gmctxtex-onp-substep-trancl*:
assumes *gmctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$
and *Id-on* (*snd* ‘ $\mathcal{R}$) $\subseteq \mathcal{R}$
shows *gmctxtex-onp P* $(\mathcal{R}^+) \subseteq \mathcal{R}^+$
proof −
{fix *s t* **assume** $(s, t) \in$ *gmctxtex-onp P* $(\mathcal{R}^+)$
from *gmctxtex-onpE*[*OF this*] **obtain** *C us vs* **where**
∗: *s* = *fill-gholes C us t* = *fill-gholes C vs* **and**
len: *num-gholes C* = *length us length us* = *length vs* **and**
inv: *P C* ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) $\in \mathcal{R}^+$ **by** *auto*
have $(s, t) \in \mathcal{R}^+$ **using** *len*(*2*) *inv*(*2*) *len*(*1*) *inv*(*1*) **unfolding** ∗
proof (*induction rule*: *trancl-list-induct*)
case (*base xs ys*)
then have (*fill-gholes C xs*, *fill-gholes C ys*) $\in \mathcal{R}$ **using** *assms*(*1*)
by *blast*
then show *?case* **by** *auto*
next
case (*step xs ys i z*)
have *sub*: *set ys* ⊆ *snd* ‘ $\mathcal{R}$ **using** *step*(*1*, *2*)
by (*auto simp*: *image-def*) (*metis in-set-idx snd-conv tranclD2*)
from *step* **have** *lft*: (*fill-gholes C xs*, *fill-gholes C ys*) $\in \mathcal{R}^+$ **by** *auto*
have (*fill-gholes C ys*, *fill-gholes C* (*ys*[*i* := *z*])) $\in$ *gmctxtex-onp P* $\mathcal{R}$
using *step*(*3*, *4*) *sub assms step*(*1*, *6*)
by (*intro gmctxtex-onpI*[*of P*, *OF step*(*7*), *of ys ys*[*i* := *z*] $\mathcal{R}$])
(*simp add*: *Id-on-eqI nth-list-update subset-iff*)+
then have (*fill-gholes C ys*, *fill-gholes C* (*ys*[*i* := *z*])) $\in \mathcal{R}$ **using** *assms*(*1*)
by *blast*
then show *?case* **using** *lft* **by** *auto*
qed}
then show *?thesis* **by** *auto*
qed

lemma *gmctxtex-onp-substep-tranclE*:
assumes *trans* $\mathcal{R}$ **and** *gmctxtex-onp Q* $\mathcal{R}$ *O* $\mathcal{R} \subseteq \mathcal{R}$ **and** $\mathcal{R}$ *O gmctxtex-onp Q*

$\mathcal{R} \subseteq \mathcal{R}$
  and $\bigwedge p\ C.\ P\ C \Longrightarrow p \in$ *poss-gmctxt* $C \Longrightarrow Q$ (*subgm-at* $C\ p$)
  and $\bigwedge C\ D.\ P\ C \Longrightarrow P\ D \Longrightarrow (C,\ D) \in$ *comp-gmctxt* $\Longrightarrow P\ (C \sqcap D)$
 shows (*gmctxtex-onp* $P\ \mathcal{R})^+ =$ *gmctxtex-onp* $P\ \mathcal{R}$ (**is** *?Ls* = *?Rs*)
proof
 show *?Rs* $\subseteq$ *?Ls* **using** *trancl-mono-set* **by** *fastforce*
next
 {fix $s\ t$ **assume** $(s,\ t) \in$ *?Ls* **then have** $(s,\ t) \in$ *?Rs*
  proof *induction*
   case (*step* $t\ u$)
   from *step*(*3*) **obtain** $C\ us\ vs$ **where**
    $*$: $s =$ *fill-gholes* $C\ us\ t =$ *fill-gholes* $C\ vs$ **and**
    l: *num-gholes* $C =$ *length* us *length* $us =$ *length* vs **and**
    inv: $P\ C\ \forall\, i<$*length vs*. $(us\ !\ i,\ vs\ !\ i) \in \mathcal{R}$
    by *auto*
   from *step*(*2*) **obtain** $D\ xs\ ys$ **where**
    $**$: $t =$ *fill-gholes* $D\ xs\ u =$ *fill-gholes* $D\ ys$ **and**
    l': *num-gholes* $D =$ *length* xs *length* $xs =$ *length* ys **and**
    inv': $P\ D\ \forall\, i<$*length ys*. $(xs\ !\ i,\ ys\ !\ i) \in \mathcal{R}$
    by *auto*
   let *?C'* = $C \sqcap D$
   let *?sss* = *unfill-gholes ?C' s* **let** *?uss* = *unfill-gholes ?C' u*
   have *less*: *?C'* $\leq$ *gmctxt-of-gterm s ?C'* $\leq$ *gmctxt-of-gterm u*
    using *eq-gfill.intros eqgf-less-eq inf.coboundedI1 inf.coboundedI2* $l(1)\ l'(1)$
    unfolding $*$ $**$ **unfolding** $l'(2)$
    by *metis*+
   from $*(2)$ $**(1)$ **have** *comp*: $(C,\ D) \in$ *comp-gmctxt* **using** $l\ l'$
    using *eqgf-comp-gmctxt* **by** *fastforce*
   then have *P*: P *?C'* **using** *inv*(*1*) *inv'*(*1*) *assms*(*5*) **by** *blast*
   moreover have l'': *num-gholes ?C'* = *length ?sss length ?sss* = *length ?uss*
    using *less* **by** *auto*
   moreover have *fill*: *fill-gholes ?C' ?sss* = *s fill-gholes ?C' ?uss* = *u*
    using *less* **by** (*simp add*: *fill-unfill-gholes*)+
   moreover have $\forall\ i <$ *length ?uss*. (*?sss* ! *i*, *?uss* ! *i*) $\in \mathcal{R}$
   proof (*rule*, *rule*)
    fix i **assume** i: $i <$ *length* (*unfill-gholes ?C' u*)
    then obtain p **where** *pos*: $p \in$ *ghole-poss ?C'*
      unfill-gholes ?C' s ! $i =$ *gsubt-at* (*fill-gholes ?C' ?sss*) p
      unfill-gholes ?C' u ! $i =$ *gsubt-at* (*fill-gholes ?C' ?uss*) p
      using *fill* l'' *fill-gholes-ghole-poss*
    by (*metis eq-gfill.intros ghole-poss-ghole-poss-list-conv length-ghole-poss-list-num-gholes nth-mem*)
   from *comp-gmctxt-inf-ghole-poss-cases*[*OF comp pos*(*1*)]
   consider (*a*) $p \in$ *ghole-poss* $C \wedge p \in$ *ghole-poss* D |
        (*b*) $p \in$ *ghole-poss* $C \wedge p \in$ *poss-gmctxt* D |
        (*c*) $p \in$ *ghole-poss* $D \wedge p \in$ *poss-gmctxt* C **by** *blast*
   then show (*unfill-gholes ?C' s* ! *i*, *unfill-gholes ?C' u* ! *i*) $\in \mathcal{R}$ **unfolding** *pos fill*
   proof *cases*

```
        case a
        then show (gsubt-at s p, gsubt-at u p) ∈ R
          using assms(1) ∗(2) l l' inv(2) inv'(2) unfolding ∗ ∗∗
          using ghole-poss-nth-subt-at
          by (metis ∗(2) ∗∗(1) eq-gfill.intros trancl-id trancl-into-trancl2)
      next
        case b
        then have sp: gsubt-at t p =Gf (subgm-at D p, gmctxt-subtgm-at-fill-args
p D xs)
          gsubt-at u p =Gf (subgm-at D p, gmctxt-subtgm-at-fill-args p D ys)
          using poss-gmctxt-fill-gholes-split[of - D - p] ∗∗ l'
          by force+
        have (gsubt-at t p, gsubt-at u p) ∈ gmctxtex-onp Q R using inv'(2)
          using assms(4)[OF inv'(1) conjunct2[OF b]] eqgfE[OF sp(1)] eqgfE[OF
sp(2)]
          by (auto simp: gmctxt-subtgm-at-fill-args-def intro!: gmctxtex-onpI)
        moreover have (gsubt-at s p, gsubt-at t p) ∈ R
          using ∗ l inv(2)
          using ghole-poss-nth-subt-at[OF - conjunct1[OF b]]
          by auto (metis eq-gfill.intros)
        ultimately show (gsubt-at s p, gsubt-at u p) ∈ R
          using assms(3) by auto
      next
       case c
       then have sp: gsubt-at s p =Gf (subgm-at C p, gmctxt-subtgm-at-fill-args
p C us)
          gsubt-at t p =Gf (subgm-at C p, gmctxt-subtgm-at-fill-args p C vs)
          using poss-gmctxt-fill-gholes-split[of - C - p] ∗ l
          by force+
        have (gsubt-at s p, gsubt-at t p) ∈ gmctxtex-onp Q R using inv(2)
          using assms(4)[OF inv(1) conjunct2[OF c]] eqgfE[OF sp(1)] eqgfE[OF
sp(2)]
          by (auto simp: gmctxt-subtgm-at-fill-args-def intro!: gmctxtex-onpI)
        moreover have (gsubt-at t p, gsubt-at u p) ∈ R
          using ∗∗ l' inv'(2)
          using ghole-poss-nth-subt-at[OF - conjunct1[OF c]]
          by auto (metis eq-gfill.intros)
        ultimately show (gsubt-at s p, gsubt-at u p) ∈ R
          using assms(2) by auto
      qed
    qed
    ultimately show ?case by (metis gmctxtex-onpI)
  qed simp}
 then show ?Ls ⊆ ?Rs by auto
qed
```

7.7 Restr to set, union and predicate distribution

lemma *Restr-gctxtex-onp-dist* [*simp*]:

Restr (*gctxtex-onp P* $\mathcal{R}$) ($\mathcal{T}_G$ $\mathcal{F}$) =
gctxtex-onp (λ *C*. *funas-gctxt C* ⊆ $\mathcal{F}$ ∧ *P C*) (*Restr* $\mathcal{R}$ ($\mathcal{T}_G$ $\mathcal{F}$))
by (*auto simp*: *gctxtex-onp-def* $\mathcal{T}_G$*-equivalent-def*) *blast*

lemma *Restr-gmctxtex-onp-dist* [*simp*]:
Restr (*gmctxtex-onp P* $\mathcal{R}$) ($\mathcal{T}_G$ $\mathcal{F}$) =
gmctxtex-onp (λ *C*. *funas-gmctxt C* ⊆ $\mathcal{F}$ ∧ *P C*) (*Restr* $\mathcal{R}$ ($\mathcal{T}_G$ $\mathcal{F}$))
by (*auto elim*!: *gmctxtex-onpE simp*: $\mathcal{T}_G$*-equivalent-def SUP-le-iff gmctxtex-onpI*)
(*metis in-set-idx subsetD*)+

lemma *Restr-id-subset-gmctxtex-onp* [*intro*]:
assumes ⋀ *C*. *num-gholes C* = *0* ∧ *funas-gmctxt C* ⊆ $\mathcal{F}$ ⟹ *P C*
shows *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) ⊆ *gmctxtex-onp P* $\mathcal{R}$
proof
fix *s t* **assume** (*s*, *t*) ∈ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$)
then show (*s*, *t*) ∈ *gmctxtex-onp P* $\mathcal{R}$ **using** *assms*[*of gmctxt-of-gterm t*]
using *gmctxtex-onpI*[*of P gmctxt-of-gterm t* [] [] $\mathcal{R}$]
by (*auto simp*: $\mathcal{T}_G$*-equivalent-def*)
qed

lemma *Restr-id-subset-gmctxtex-onp2* [*intro*]:
assumes ⋀ *f n*. (*f*, *n*) ∈ $\mathcal{F}$ ⟹ *P* (*GMFun f* (*replicate n GMHole*))
and ⋀ *C Ds*. *num-gholes C* = *length Ds* ⟹ *P C* ⟹ ∀ *D* ∈ *set Ds*. *P D* ⟹ *P* (*fill-gholes-gmctxt C Ds*)
shows *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) ⊆ *gmctxtex-onp P* $\mathcal{R}$
proof
fix *s t* **assume** (*s*, *t*) ∈ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$)
then have ∗: *s* = *t t* ∈ $\mathcal{T}_G$ $\mathcal{F}$ **by** *auto*
have *P* (*gmctxt-of-gterm t*) **using** ∗(*2*)
proof (*induct*)
case (*const a*)
show *?case* **using** *assms*(*1*)[*OF const*] **by** *auto*
next
case (*ind f n ss*)
let *?C* = *GMFun f* (*replicate* (*length ss*) *GMHole*)
have *P* (*fill-gholes-gmctxt ?C* (*map gmctxt-of-gterm ss*))
using *assms*(*1*)[*OF ind*(*1*)] *ind*
by (*intro assms*(*2*)) (*auto simp*: *in-set-conv-nth*)
then show *?case*
by (*metis fill-gholes-gmctxt-GMFun-replicate-length gmctxt-of-gterm.simps length-map*)
qed
from *gmctxtex-onpI*[*of P*, *OF this*] **show** (*s*, *t*) ∈ *gmctxtex-onp P* $\mathcal{R}$ **unfolding** ∗
by *auto*
qed

lemma *gctxtex-onp-union* [*simp*]:
gctxtex-onp P ($\mathcal{R} \cup \mathcal{L}$) = *gctxtex-onp P* $\mathcal{R}$ ∪ *gctxtex-onp P* $\mathcal{L}$
by *auto*

lemma *gctxtex-onp-pred-dist*:
assumes ⋀ *C*. *P C* ⟷ *Q C* ∨ *R C*
shows *gctxtex-onp P* $\mathcal{R}$ = *gctxtex-onp Q* $\mathcal{R}$ ∪ *gctxtex-onp R* $\mathcal{R}$
using *assms* **by** *auto fastforce*

lemma *gmctxtex-onp-pred-dist*:
assumes ⋀ *C*. *P C* ⟷ *Q C* ∨ *R C*
shows *gmctxtex-onp P* $\mathcal{R}$ = *gmctxtex-onp Q* $\mathcal{R}$ ∪ *gmctxtex-onp R* $\mathcal{R}$
using *assms* **by** (*auto elim*!: *gmctxtex-onpE*)

lemma *trivial-gctxtex-onp* [*simp*]: *gctxtex-onp* (λ *C*. *C* = $\square_G$) $\mathcal{R}$ = $\mathcal{R}$
using *gctxtex-closure* **by** *force*

lemma *trivial-gmctxtex-onp* [*simp*]: *gmctxtex-onp* (λ *C*. *C* = *GMHole*) $\mathcal{R}$ = $\mathcal{R}$
proof
show *gmctxtex-onp* (λ*C*. *C* = *GMHole*) $\mathcal{R}$ ⊆ $\mathcal{R}$
by (*auto elim*!: *gmctxtex-onpE*) *force*
next
show $\mathcal{R}$ ⊆ *gmctxtex-onp* (λ*C*. *C* = *GMHole*) $\mathcal{R}$
by (*intro gmctxtex-closure*) *auto*
qed

7.8 Distribution of context closures over relation composition

lemma *gctxtex-onp-relcomp-inner*:
gctxtex-onp P ($\mathcal{R}$ *O* $\mathcal{L}$) ⊆ *gctxtex-onp P* $\mathcal{R}$ *O gctxtex-onp P* $\mathcal{L}$
by *auto*

lemma *gmctxtex-onp-relcomp-inner*:
gmctxtex-onp P ($\mathcal{R}$ *O* $\mathcal{L}$) ⊆ *gmctxtex-onp P* $\mathcal{R}$ *O gmctxtex-onp P* $\mathcal{L}$
proof
fix *s t*
assume (*s*, *t*) ∈ *gmctxtex-onp P* ($\mathcal{R}$ *O* $\mathcal{L}$)
from *gmctxtex-onpE*[*OF this*] **obtain** *C us vs* **where**
∗: *s* = *fill-gholes C us t* = *fill-gholes C vs* **and**
len: *num-gholes C* = *length us length us* = *length vs* **and**
inv: *P C* ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$ *O* $\mathcal{L}$ **by** *blast*
obtain *zs* **where** *l*: *length vs* = *length zs* **and**
rel: ∀ *i* < *length zs*. (*us* ! *i*, *zs* ! *i*) ∈ $\mathcal{R}$ ∀ *i* < *length zs*. (*zs* ! *i*, *vs* ! *i*) ∈ $\mathcal{L}$
using *len*(*2*) *inv*(*2*) *Ex-list-of-length-P*[*of* - λ *y i*. (*us* ! *i*, *y*) ∈ $\mathcal{R}$ ∧ (*y*, *vs* ! *i*) ∈ $\mathcal{L}$]
by (*auto simp*: *relcomp-unfold*) *metis*
from *len l rel inv* **have** (*s*, *fill-gholes C zs*) ∈ *gmctxtex-onp P* $\mathcal{R}$ **unfolding** ∗
by *auto*

moreover from *len l rel inv* **have** (*fill-gholes C zs*, *t*) ∈ *gmctxtex-onp P* $\mathcal{L}$
unfolding ∗
by *auto*
ultimately show (*s*, *t*) ∈ *gmctxtex-onp P* $\mathcal{R}$ *O gmctxtex-onp P* $\mathcal{L}$
by *auto*
qed

7.9 Signature preserving and signature closed

definition *function-closed* **where**
function-closed $\mathcal{F}$ $\mathcal{R}$ ⟷ (∀ *f ss ts*. (*f*, *length ts*) ∈ $\mathcal{F}$ ⟶ *0* ≠ *length ts* ⟶
length ss = *length ts* ⟶ (∀ *i*. *i* < *length ts* ⟶ (*ss* ! *i*, *ts* ! *i*) ∈ $\mathcal{R}$) ⟶
(*GFun f ss*, *GFun f ts*) ∈ $\mathcal{R}$)

lemma *function-closedD*: *function-closed* $\mathcal{F}$ $\mathcal{R}$ ⟹
(*f*,*length ts*) ∈ $\mathcal{F}$ ⟹ *0* ≠ *length ts* ⟹ *length ss* = *length ts* ⟹
⟦⋀ *i*. *i* < *length ts* ⟹ (*ss* ! *i*, *ts* ! *i*) ∈ $\mathcal{R}$⟧ ⟹
(*GFun f ss*, *GFun f ts*) ∈ $\mathcal{R}$
unfolding *function-closed-def* **by** *blast*

lemma *all-ctxt-closed-imp-function-closed*:
all-ctxt-closed $\mathcal{F}$ $\mathcal{R}$ ⟹ *function-closed* $\mathcal{F}$ $\mathcal{R}$
unfolding *all-ctxt-closed-def function-closed-def*
by *auto*

lemma *all-ctxt-closed-imp-reflx-on-sig*:
assumes *all-ctxt-closed* $\mathcal{F}$ $\mathcal{R}$
shows *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) ⊆ $\mathcal{R}$
proof −
{fix *s* **assume** (*s*, *s*) ∈ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) **then have** (*s*, *s*) ∈ $\mathcal{R}$
proof (*induction s*)
case (*GFun f ts*)
then show *?case* **using** *all-ctxt-closedD*[*OF assms*]
by (*auto simp*: $\mathcal{T}_G$*-equivalent-def UN-subset-iff*)
qed}
then show *?thesis* **by** *auto*
qed

lemma *function-closed-un-id-all-ctxt-closed*:
function-closed $\mathcal{F}$ $\mathcal{R}$ ⟹ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) ⊆ $\mathcal{R}$ ⟹ *all-ctxt-closed* $\mathcal{F}$ $\mathcal{R}$
unfolding *all-ctxt-closed-def*
by (*auto dest*: *function-closedD simp*: *subsetD*)

lemma *gctxtex-onp-in-signature* [*intro*]:
assumes ⋀ *C*. *P C* ⟹ *funas-gctxt C* ⊆ $\mathcal{F}$ ⋀ *C*. *P C* ⟹ *funas-gctxt C* ⊆ $\mathcal{G}$
and $\mathcal{R}$ ⊆ $\mathcal{T}_G$ $\mathcal{F}$ × $\mathcal{T}_G$ $\mathcal{G}$
shows *gctxtex-onp P* $\mathcal{R}$ ⊆ $\mathcal{T}_G$ $\mathcal{F}$ × $\mathcal{T}_G$ $\mathcal{G}$ **using** *assms*
by (*auto simp*: *gctxtex-onp-def* $\mathcal{T}_G$*-equivalent-def*) *blast*+

lemma *gmctxtex-onp-in-signature* [*intro*]:
assumes $\bigwedge C.\ P\ C \Longrightarrow$ *funas-gmctxt* $C \subseteq \mathcal{F} \bigwedge C.\ P\ C \Longrightarrow$ *funas-gmctxt* $C \subseteq \mathcal{G}$
and $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$
shows *gmctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$ **using** *assms*
by (*auto simp*: *gmctxtex-onp-def* $\mathcal{T}_G$*-equivalent-def in-set-conv-nth*) *force*+

lemma *gctxtex-onp-in-signature-tranc* [*intro*]:
gctxtex-onp $P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ (*gctxtex-onp* $P\ \mathcal{R})^+ \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
by (*auto simp*: *Restr-simps*)

lemma *gmctxtex-onp-in-signature-tranc* [*intro*]:
gmctxtex-onp $P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ (*gmctxtex-onp* $P\ \mathcal{R})^+ \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
by (*auto simp*: *Restr-simps*)

lemma *gmctxtex-onp-fun-closed* [*intro*!]:
assumes $\bigwedge f\ n.\ (f,\ n) \in \mathcal{F} \Longrightarrow n \neq 0 \Longrightarrow P$ (*GMFun f* (*replicate n GMHole*))
and $\bigwedge C\ Ds.\ P\ C \Longrightarrow$ *num-gholes* $C =$ *length* $Ds \Longrightarrow 0 <$ *num-gholes* $C \Longrightarrow$
$\forall\ D \in$ *set* $Ds.\ P\ D \Longrightarrow P$ (*fill-gholes-gmctxt C Ds*)
shows *function-closed* $\mathcal{F}$ (*gmctxtex-onp* $P\ \mathcal{R}$) **unfolding** *function-closed-def*
proof (*rule allI*, *intro allI*, *intro impI*)
fix *f ss ts* **assume** *sig*: $(f,$ *length ts*$) \in \mathcal{F}$
and *len*: $0 \neq$ *length ts length ss* = *length ts*
and *mem*: $\forall\ i <$ *length ts*. (*ss* ! *i*, *ts* ! *i*) $\in$ *gmctxtex-onp* $P\ \mathcal{R}$
let *?C* = *GMFun f* (*replicate* (*length ts*) *GMHole*)
from *mem len* **obtain** *Ds sss tss* **where**
l': *length ts* = *length Ds length Ds* = *length sss length sss* = *length tss* **and**
inn: $\forall\ i <$ *length tss*. *length* (*sss* ! *i*) = *length* (*tss* ! *i*) **and**
eq: $\forall\ i <$ *length tss*. *ss* ! $i =_{Gf}$ (*Ds* ! *i*, *sss* ! *i*) $\forall\ i <$ *length tss*. *ts* ! $i =_{Gf}$ (*Ds* ! *i*, *tss* ! *i*) **and**
inv: $\forall\ i <$ *length* (*concat tss*). (*concat sss* ! *i*, *concat tss* ! *i*) $\in \mathcal{R} \forall\ D \in$ *set Ds*. *P D*
by (*auto elim*!: *gmctxtex-onp-listE*)
have $*$: *fill-gholes ?C ss* = *GFun f ss fill-gholes ?C ts* = *GFun f ts*
using *len assms*(*1*) **by** (*auto simp del*: *fill-gholes.simps*)
have *s*: *GFun f ss* $=_{Gf}$ (*fill-gholes-gmctxt ?C Ds*, *concat sss*)
using *assms*(*1*) l' *eq*(*1*) *inn len inv*(*1*) **unfolding** $*$[*symmetric*]
by (*intro fill-gholes-gmctxt-sound*) *auto*
have *t*: *GFun f ts* $=_{Gf}$ (*fill-gholes-gmctxt ?C Ds*, *concat tss*)
using *assms*(*1*) *eq* l' *inn len inv*(*1*) **unfolding** $*$[*symmetric*]
by (*intro fill-gholes-gmctxt-sound*) *auto*
then show (*GFun f ss*, *GFun f ts*) $\in$ *gmctxtex-onp* $P\ \mathcal{R}$
unfolding *eqgfE*[*OF s*] *eqgfE*[*OF t*]
using *eqgfE*(*2*)[*OF s*] *eqgfE*(*2*)[*OF t*] *sig len* l' *inv*
using *assms*(*1*)[*OF sig*] *assms*(*2*)[*of GMFun f* (*replicate* (*length ts*) *GMHole*) *Ds*]
using *gmctxtex-onpI*[*of P fill-gholes-gmctxt* (*GMFun f* (*replicate* (*length ts*) *GMHole*)) *Ds concat sss concat tss* $\mathcal{R}$]

by (*auto simp del*: *fill-gholes-gmctxt.simps fill-gholes.simps*)
qed

declare *subsetI*[*rule del*]
lemma *gmctxtex-onp-sig-closed* [*intro*]:
assumes $\bigwedge f\ n.\ (f,\ n) \in \mathcal{F} \Longrightarrow P\ (GMFun\ f\ (replicate\ n\ GMHole))$
and $\bigwedge C\ Ds.\ num\text{-}gholes\ C = length\ Ds \Longrightarrow P\ C \Longrightarrow \forall\ D \in set\ Ds.\ P\ D \Longrightarrow P\ (fill\text{-}gholes\text{-}gmctxt\ C\ Ds)$
shows *all-ctxt-closed* $\mathcal{F}$ (*gmctxtex-onp* P $\mathcal{R}$) **using** *assms*
by (*intro function-closed-un-id-all-ctxt-closed*) *auto*
declare *subsetI*[*intro*!]

lemma *gmctxt-cl-gmctxtex-onp-conv*:
gmctxt-cl $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-onp* (λ C. *funas-gmctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$ (**is** *?Ls* = *?Rs*)
proof −
have *sig-cl*: *all-ctxt-closed* $\mathcal{F}$ (*?Rs*) **by** (*intro gmctxtex-onp-sig-closed*) *auto*
{fix s t **assume** $(s,\ t) \in$ *?Ls* **then have** $(s,\ t) \in$ *?Rs*
proof *induct*
case (*step ss ts f*)
then show *?case* **using** *all-ctxt-closedD*[*OF sig-cl*]
by *force*
qed (*intro subsetD*[*OF gmctxtex-onp-arg-monoI*], *auto*)**}**
moreover
{fix s t **assume** $(s,\ t) \in$ *?Rs*
from *gmctxtex-onpE*[*OF this*] **obtain** C *us vs* **where**
terms: s = *fill-gholes* C *us* t = *fill-gholes* C *vs* **and**
fill-inv: *num-gholes* C = *length us length us* = *length vs* **and**
rel: *funas-gmctxt* $C \subseteq \mathcal{F}\ \forall\ i <$ *length vs*. $(us\ !\ i,\ vs\ !\ i) \in \mathcal{R}$ **by** *blast*
have $(s,\ t) \in$ *?Ls* **unfolding** *terms* **using** *fill-inv rel*
proof (*induct* C *arbitrary*: *us vs*)
case *GMHole*
then show *?case* **using** *rel*(*2*) **by** (*cases vs*; *cases us*) *auto*
next
case (*GMFun f Ds*)
show *?case* **using** *GMFun*(*2*−) **unfolding** *partition-holes-fill-gholes-conv′*
by (*intro all-ctxt-closedD*[*OF gmctxt-cl-is-all-ctxt-closed*[*of* $\mathcal{F}$ $\mathcal{R}$]])
(*auto simp*: *partition-by-nth-nth SUP-le-iff length-partition-gholes-nth intro*!: *GMFun*(*1*))
qed}
ultimately show *?thesis* **by** *auto*
qed

end
theory *FOR-Certificate*
imports *Rewriting*
begin

8 Certificate syntax and type declarations

type-alias *fvar* = *nat* — variable id
datatype *ftrs* = *Fwd nat* | *Bwd nat* — TRS id and direction

definition *map-ftrs* **where**
map-ftrs f = *case-ftrs* (*Fwd* ∘ *f*) (*Bwd* ∘ *f*)

8.1 GTT relations

datatype *'trs gtt-rel* — GTT relations
= *ARoot 'trs list* — root steps
| *GInv 'trs gtt-rel* — inverse of anchored or ordinary GTT relation
| *AUnion 'trs gtt-rel 'trs gtt-rel* — union of anchored GTT relation
| *ATrancl 'trs gtt-rel* — transitive closure of anchored GTT relation
| *GTrancl 'trs gtt-rel* — transitive closure of ordinary GTT relation
| *AComp 'trs gtt-rel 'trs gtt-rel* — composition of anchored GTT relations
| *GComp 'trs gtt-rel 'trs gtt-rel* — composition of ordinary GTT relations

definition *GSteps* **where** *GSteps trss* = *GTrancl* (*ARoot trss*)

8.2 RR1 and RR2 relations

datatype *pos-step* — position specification for lifting anchored GTT relation
= *PRoot* — allow only root steps
| *PNonRoot* — allow only non-root steps
| *PAny* — allow any position

datatype *ext-step* — kind of rewrite steps for lifting anchored GTT relation
= *ESingle* — single steps
| *EParallel* — parallel steps, allowing the empty step
| *EStrictParallel* — parallel steps, no allowing the empty step

datatype *'trs rr1-rel* — RR1 relations, aka regular tree languages
= *R1Terms* — all terms as RR1 relation (regular tree languages)
| *R1NF 'trs list* — direct normal form construction wrt. single steps
| *R1Inf 'trs rr2-rel* — infiniteness predicate
| *R1Proj nat 'trs rr2-rel* — projection of RR2 relation
| *R1Union 'trs rr1-rel 'trs rr1-rel* — union of RR1 relations
| *R1Inter 'trs rr1-rel 'trs rr1-rel* — intersection of RR1 relations
| *R1Diff 'trs rr1-rel 'trs rr1-rel* — difference of RR1 relations
and *'trs rr2-rel* — RR2 relations
= *R2GTT-Rel 'trs gtt-rel pos-step ext-step* — lifted GTT relations
| *R2Diag 'trs rr1-rel* — diagonal relation
| *R2Prod 'trs rr1-rel 'trs rr1-rel* — Cartesian product
| *R2Inv 'trs rr2-rel* — inverse of RR2 relation
| *R2Union 'trs rr2-rel 'trs rr2-rel* — union of RR2 relations

| *R2Inter 'trs rr2-rel 'trs rr2-rel* — intersection of RR2 relations
| *R2Diff 'trs rr2-rel 'trs rr2-rel* — difference of RR2 relations
| *R2Comp 'trs rr2-rel 'trs rr2-rel* — composition of RR2 relations

definition *R1Fin* **where** — finiteness predicate
R1Fin r = R1Diff R1Terms (R1Inf r)
definition *R2Eq* **where** — equality
R2Eq = R2Diag R1Terms
definition *R2Reflc* **where** — reflexive closure
R2Reflc r = R2Union r R2Eq
definition *R2Step* **where** — single step $\to$
R2Step trss = R2GTT-Rel (ARoot trss) PAny ESingle
definition *R2StepEq* **where** — at most one step $\to^{=}$
R2StepEq trss = R2Reflc (R2Step trss)
definition *R2Steps* **where** — at least one step $\to^{+}$
R2Steps trss = R2GTT-Rel (GSteps trss) PAny EStrictParallel
definition *R2StepsEq* **where** — many steps $\to^{*}$
R2StepsEq trss = R2GTT-Rel (GSteps trss) PAny EParallel
definition *R2StepsNF* **where** — rewrite to normal form $\to^{!}$
R2StepsNF trss = R2Inter (R2StepsEq trss) (R2Prod R1Terms (R1NF trss))
definition *R2ParStep* **where** — parallel step
R2ParStep trss = R2GTT-Rel (ARoot trss) PAny EParallel
definition *R2RootStep* **where** — root step $\to_{\epsilon}$
R2RootStep trss = R2GTT-Rel (ARoot trss) PRoot ESingle
definition *R2RootStepEq* **where** — at most one root step $\to_{\epsilon}^{=}$
R2RootStepEq trss = R2Reflc (R2RootStep trss)

definition *R2RootSteps* **where** — at least one root step $\to_{\epsilon}^{+}$
R2RootSteps trss = R2GTT-Rel (ATrancl (ARoot trss)) PRoot ESingle
definition *R2RootStepsEq* **where** — many root steps $\to_{\epsilon}^{*}$
R2RootStepsEq trss = R2Reflc (R2RootSteps trss)
definition *R2NonRootStep* **where** — non-root step $\to_{>\epsilon}$
R2NonRootStep trss = R2GTT-Rel (ARoot trss) PNonRoot ESingle
definition *R2NonRootStepEq* **where** — at most one non-root step $\to_{>\epsilon}^{=}$
R2NonRootStepEq trss = R2Reflc (R2NonRootStep trss)
definition *R2NonRootSteps* **where** — at least one non-root step $\to_{>\epsilon}^{+}$
R2NonRootSteps trss = R2GTT-Rel (GSteps trss) PNonRoot EStrictParallel
definition *R2NonRootStepsEq* **where** — many non-root steps $\to_{>\epsilon}^{*}$
R2NonRootStepsEq trss = R2GTT-Rel (GSteps trss) PNonRoot EParallel
definition *R2Meet* **where** — meet $\uparrow$
R2Meet trss = R2GTT-Rel (GComp (GInv (GSteps trss)) (GSteps trss)) PAny EParallel
definition *R2Join* **where** — join $\downarrow$
R2Join trss = R2GTT-Rel (GComp (GSteps trss) (GInv (GSteps trss))) PAny EParallel

8.3 Formulas

datatype *'trs formula* — formulas
= *FRR1 'trs rr1-rel fvar* — application of RR1 relation
| *FRR2 'trs rr2-rel fvar fvar* — application of RR2 relation
| *FAnd ('trs formula) list* — conjunction
| *FOr ('trs formula) list* — disjunction
| *FNot 'trs formula* — negation
| *FExists 'trs formula* — existential quantification
| *FForall 'trs formula* — universal quantification

definition *FTrue* **where** — true
FTrue ≡ *FAnd* []
definition *FFalse* **where** — false
FFalse ≡ *FOr* []

definition *FRestrict* **where** — reorder/rename/restrict TRSs for subformula
FRestrict f trss ≡ *map-formula* (*map-ftrs* (λ*n*. *if n* ≥ *length trss then 0 else trss* ! *n*)) *f*

8.4 Signatures and Problems

datatype (*'f*, *'v*, *'t*) *many-sorted-sig*
= *Many-Sorted-Sig* (*ms-functions*: (*'f* × *'t list* × *'t*) *list*) (*ms-variables*: (*'v* × *'t*) *list*)

datatype (*'f*, *'v*, *'t*) *problem*
= *Problem* (*p-signature*: (*'f*, *'v*, *'t*) *many-sorted-sig*)
(*p-trss*: (*'f*, *'v*) *trs list*)
(*p-formula*: *ftrs formula*)

8.5 Proofs

datatype *equivalence* — formula equivalences
= *EDistribAndOr* — distributivity: conjunction over disjunction
| *EDistribOrAnd* — distributivity: disjunction over conjunction

datatype *'trs inference* — inference rules for formula creation
= *IRR1 'trs rr1-rel fvar* — formula from RR1 relation
| *IRR2 'trs rr2-rel fvar fvar* — formula from RR2 relation
| *IAnd nat list* — conjunction
| *IOr nat list* — disjunction
| *INot nat* — negation
| *IExists nat* — existential quantification
| *IRename nat fvar list* — permute variables
| *INNFPlus nat* — equivalence modulo negation normal form plus ACIU0 for ∧ and ∨
| *IRepl equivalence nat list nat* — replacement according to given equivalence

datatype *claim* = *Empty* | *Nonempty*

datatype *info* = *Size nat nat nat*

datatype *'trs certificate*
= *Certificate* (*nat* × *'trs inference* × *'trs formula* × *info list*) *list claim nat*

8.6 Example

definition *no-normal-forms-cert* :: *ftrs certificate* **where**
no-normal-forms-cert = *Certificate*
[(*0*, (*IRR2* (*R2Step* [*Fwd 0*]) *1 0*),
(*FRR2* (*R2Step* [*Fwd 0*]) *1 0*), [])
, (*1*, (*IExists 0*),
(*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*)), [])
, (*2*, (*INot 1*),
(*FNot* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*))), [])
, (*3*, (*IExists 2*),
(*FExists* (*FNot* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*)))), [])
, (*4*, (*INot 3*),
(*FNot* (*FExists* (*FNot* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*))))), [])
, (*5*, (*INNFPlus 4*),
(*FForall* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*))), [])
] *Nonempty 5*

definition *no-normal-forms-problem* :: (*string*, *string*, *unit*) *problem* **where**
no-normal-forms-problem = *Problem*
(*Many-Sorted-Sig* [(*''f''*,[()],()), (*''a''*,[],())] [(*''x''*,())])
[{(*Fun ''f''* [*Var ''x''*],*Fun ''a''* [])}]
(*FForall* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*)))

end

9 Lifting root steps to single/parallel root/non-root steps

theory *Lift-Root-Step*
imports
Rewriting
FOR-Certificate
Context-Extensions
Multihole-Context
begin

Closure under all contexts

abbreviation *gctxtcl* $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. *True*) $\mathcal{R}$
abbreviation *gmctxtcl* $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. *True*) $\mathcal{R}$

Extension under all non empty contexts

abbreviation *gctxtex-nempty* $\mathcal{R} \equiv$ *gctxtex-onp* $(\lambda\ C.\ C \neq \Box_G)\ \mathcal{R}$
abbreviation *gmctxtex-nempty* $\mathcal{R} \equiv$ *gmctxtex-onp* $(\lambda\ C.\ C \neq GMHole)\ \mathcal{R}$

Closure under all contexts respecting the signature

abbreviation *gctxtcl-funas* $\mathcal{F}\ \mathcal{R} \equiv$ *gctxtex-onp* $(\lambda\ C.$ *funas-gctxt* $C \subseteq \mathcal{F})\ \mathcal{R}$
abbreviation *gmctxtcl-funas* $\mathcal{F}\ \mathcal{R} \equiv$ *gmctxtex-onp* $(\lambda\ C.$ *funas-gmctxt* $C \subseteq \mathcal{F})\ \mathcal{R}$

Closure under all multihole contexts with at least one hole respecting the signature

abbreviation *gmctxtcl-funas-strict* $\mathcal{F}\ \mathcal{R} \equiv$ *gmctxtex-onp* $(\lambda\ C.\ 0 <$ *num-gholes* $C \wedge$ *funas-gmctxt* $C \subseteq \mathcal{F})\ \mathcal{R}$

Extension under all non empty contexts respecting the signature

abbreviation *gctxtex-funas-nroot* $\mathcal{F}\ \mathcal{R} \equiv$ *gctxtex-onp* $(\lambda\ C.$ *funas-gctxt* $C \subseteq \mathcal{F} \wedge C \neq \Box_G)\ \mathcal{R}$
abbreviation *gmctxtex-funas-nroot* $\mathcal{F}\ \mathcal{R} \equiv$ *gmctxtex-onp* $(\lambda\ C.$ *funas-gmctxt* $C \subseteq \mathcal{F} \wedge C \neq GMHole)\ \mathcal{R}$

Extension under all non empty contexts respecting the signature

abbreviation *gmctxtex-funas-nroot-strict* $\mathcal{F}\ \mathcal{R} \equiv$
gmctxtex-onp $(\lambda\ C.\ 0 <$ *num-gholes* $C \wedge$ *funas-gmctxt* $C \subseteq \mathcal{F} \wedge C \neq GMHole)\ \mathcal{R}$

9.1 Rewrite steps equivalent definitions

definition *gsubst-cl* :: $('f, 'v)$ *trs* $\Rightarrow$ $'f$ *gterm rel* **where**
gsubst-cl $\mathcal{R} = \{($*gterm-of-term* $(l \cdot \sigma),$ *gterm-of-term* $(r \cdot \sigma)) \mid$
$l\ r\ (\sigma :: 'v \Rightarrow ('f, 'v)$ *Term.term*$).\ (l, r) \in \mathcal{R} \wedge$ *ground* $(l \cdot \sigma) \wedge$ *ground* $(r \cdot \sigma)\}$

definition *gnrrstepD* :: $'f$ *sig* $\Rightarrow$ $'f$ *gterm rel* $\Rightarrow$ $'f$ *gterm rel* **where**
gnrrstepD $\mathcal{F}\ \mathcal{R} =$ *gctxtex-funas-nroot* $\mathcal{F}\ \mathcal{R}$

definition *grstepD* :: $'f$ *sig* $\Rightarrow$ $'f$ *gterm rel* $\Rightarrow$ $'f$ *gterm rel* **where**
grstepD $\mathcal{F}\ \mathcal{R} =$ *gctxtcl-funas* $\mathcal{F}\ \mathcal{R}$

definition *gpar-rstepD* :: $'f$ *sig* $\Rightarrow$ $'f$ *gterm rel* $\Rightarrow$ $'f$ *gterm rel* **where**
gpar-rstepD $\mathcal{F}\ \mathcal{R} =$ *gmctxtcl-funas* $\mathcal{F}\ \mathcal{R}$

inductive-set *gpar-rstepD'* :: $'f$ *sig* $\Rightarrow$ $'f$ *gterm rel* $\Rightarrow$ $'f$ *gterm rel* **for** $\mathcal{F}$ :: $'f$ *sig* **and** $\mathcal{R}$ :: $'f$ *gterm rel*
where *groot-step* [*intro*]: $(s, t) \in \mathcal{R} \Longrightarrow (s, t) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R}$
| *gpar-step-fun* [*intro*]: $[\![\bigwedge i.\ i <$ *length ts* $\Longrightarrow (ss\ !\ i, ts\ !\ i) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R}]\!] \Longrightarrow$ *length ss* $=$ *length ts*
$\Longrightarrow (f,$ *length ts*$) \in \mathcal{F} \Longrightarrow ($*GFun f ss*, *GFun f ts*$) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R}$

9.2 Interface between rewrite step definitions and sets

fun *lift-root-step* :: $('f \times nat)$ *set* $\Rightarrow$ *pos-step* $\Rightarrow$ *ext-step* $\Rightarrow$ $'f$ *gterm rel* $\Rightarrow$ $'f$ *gterm rel* **where**

lift-root-step $\mathcal{F}$ *PAny ESingle* $\mathcal{R}$ = *gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PAny EStrictParallel* $\mathcal{R}$ = *gmctxtcl-funas-strict* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PAny EParallel* $\mathcal{R}$ = *gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PNonRoot ESingle* $\mathcal{R}$ = *gctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PNonRoot EStrictParallel* $\mathcal{R}$ = *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PNonRoot EParallel* $\mathcal{R}$ = *gmctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PRoot ESingle* $\mathcal{R}$ = $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PRoot EStrictParallel* $\mathcal{R}$ = $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PRoot EParallel* $\mathcal{R}$ = $\mathcal{R}$ ∪ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$)

9.3 Compatibility of used predicate extensions and signature closure

lemma *compatible-p* [*simp*]:
compatible-p (λ *C*. *C* ≠ $\Box_G$) (λ *C*. *C* ≠ *GMHole*)
compatible-p (λ *C*. *funas-gctxt C* ⊆ $\mathcal{F}$) (λ *C*. *funas-gmctxt C* ⊆ $\mathcal{F}$)
compatible-p (λ *C*. *funas-gctxt C* ⊆ $\mathcal{F}$ ∧ *C* ≠ $\Box_G$) (λ *C*. *funas-gmctxt C* ⊆ $\mathcal{F}$ ∧ *C* ≠ *GMHole*)
unfolding *compatible-p-def*
by *rule* (*case-tac C*, *auto*)+

lemma *gmctxtcl-funas-sigcl*:
all-ctxt-closed $\mathcal{F}$ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)
by (*intro gmctxtex-onp-sig-closed*) *auto*

lemma *gctxtex-funas-nroot-sigcl*:
all-ctxt-closed $\mathcal{F}$ (*gmctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$)
by (*intro gmctxtex-onp-sig-closed*) *auto*

lemma *gmctxtcl-funas-strict-funcl*:
function-closed $\mathcal{F}$ (*gmctxtcl-funas-strict* $\mathcal{F}$ $\mathcal{R}$)
by (*intro gmctxtex-onp-fun-closed*) (*auto dest*: *list.set-sel*)

lemma *gmctxtex-funas-nroot-strict-funcl*:
function-closed $\mathcal{F}$ (*gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$)
by (*intro gmctxtex-onp-fun-closed*) (*auto dest*: *list.set-sel*)

lemma *gctxtcl-funas-dist*:
gctxtcl-funas $\mathcal{F}$ $\mathcal{R}$ = *gctxtex-onp* (λ *C*. *C* = $\Box_G$) $\mathcal{R}$ ∪ *gctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$
by (*intro gctxtex-onp-pred-dist*) *auto*

lemma *gmctxtex-funas-nroot-dist*:
gmctxtex-funas-nroot $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$ ∪
gmctxtex-onp (λ *C*. *num-gholes C* = *0* ∧ *funas-gmctxt C* ⊆ $\mathcal{F}$) $\mathcal{R}$
by (*intro gmctxtex-onp-pred-dist*) *auto*

lemma *gmctxtcl-funas-dist*:
gmctxtcl-funas $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-onp* (λ *C*. *num-gholes C* = *0* ∧ *funas-gmctxt C* ⊆ $\mathcal{F}$) $\mathcal{R}$ ∪

gmctxtex-onp (λ *C*. *0* < *num-gholes C* $\wedge$ *funas-gmctxt C* $\subseteq$ $\mathcal{F}$) $\mathcal{R}$
by (*intro gmctxtex-onp-pred-dist*) *auto*

lemma *gmctxtcl-funas-strict-dist*:
gmctxtcl-funas-strict $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$ $\cup$ *gmctxtex-onp* (λ *C*. *C* = *GMHole*) $\mathcal{R}$
by (*intro gmctxtex-onp-pred-dist*) *auto*

lemma *gmctxtex-onpzero-num-gholes-id* [*simp*]:
gmctxtex-onp (λ *C*. *num-gholes C* = *0* $\wedge$ *funas-gmctxt C* $\subseteq$ $\mathcal{F}$) $\mathcal{R}$ = *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) (**is** *?Ls* = *?Rs*)
proof −
{fix *s t* **assume** (*s*, *t*) ∈ *?Ls* **from** *gmctxtex-onpE*[*OF this*] **obtain** *C us vs* **where**
∗: *s* = *fill-gholes C us t* = *fill-gholes C vs* **and**
len: *num-gholes C* = *length us length us* = *length vs* **and**
inv: *num-gholes C* = *0* $\wedge$ *funas-gmctxt C* $\subseteq$ $\mathcal{F}$ **by** *auto*
then have (*s*, *t*) ∈ *?Rs* **using** *len inv* **unfolding** ∗
by (*cases us*; *cases vs*) (*auto simp*: $\mathcal{T}_G$*-funas-gterm-conv*)**}**
moreover have *?Rs* ⊆ *?Ls*
by (*intro Restr-id-subset-gmctxtex-onp*) *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *gctxtex-onp-sign-trans-fst*:
assumes (*s*, *t*) ∈ *gctxtex-onp P R* **and** *s* ∈ $\mathcal{T}_G$ $\mathcal{F}$
shows (*s*, *t*) ∈ *gctxtex-onp* (λ *C*. *funas-gctxt C* ⊆ $\mathcal{F}$ $\wedge$ *P C*) *R*
using *assms*
by (*auto simp*: $\mathcal{T}_G$*-equivalent-def elim*!: *gctxtex-onpE*)

lemma *gctxtex-onp-sign-trans-snd*:
assumes (*s*, *t*) ∈ *gctxtex-onp P R* **and** *t* ∈ $\mathcal{T}_G$ $\mathcal{F}$
shows (*s*, *t*) ∈ *gctxtex-onp* (λ *C*. *funas-gctxt C* ⊆ $\mathcal{F}$ $\wedge$ *P C*) *R*
using *assms*
by (*auto simp*: $\mathcal{T}_G$*-equivalent-def elim*!: *gctxtex-onpE*)

lemma *gmctxtex-onp-sign-trans-fst*:
assumes (*s*, *t*) ∈ *gmctxtex-onp P R* **and** *s* ∈ $\mathcal{T}_G$ $\mathcal{F}$
shows (*s*, *t*) ∈ *gmctxtex-onp* (λ *C*. *P C* $\wedge$ *funas-gmctxt C* ⊆ $\mathcal{F}$) *R*
using *assms*
by (*auto simp*: $\mathcal{T}_G$*-equivalent-def simp add*: *gmctxtex-onpI*)

lemma *gmctxtex-onp-sign-trans-snd*:
assumes (*s*, *t*) ∈ *gmctxtex-onp P R* **and** *t* ∈ $\mathcal{T}_G$ $\mathcal{F}$
shows (*s*, *t*) ∈ *gmctxtex-onp* (λ *C*. *P C* $\wedge$ *funas-gmctxt C* ⊆ $\mathcal{F}$) *R*
using *assms*
by (*auto simp*: $\mathcal{T}_G$*-equivalent-def simp add*: *gmctxtex-onpI*)

9.4 Basic lemmas

lemma *gsubst-cl*:
fixes $\mathcal{R}$ :: $('f, 'v)$ *trs* **and** σ :: $'v \Rightarrow ('f, 'v)$ *term*
assumes $(l, r) \in \mathcal{R}$ **and** *ground* $(l \cdot \sigma)$ *ground* $(r \cdot \sigma)$
shows $(gterm\text{-}of\text{-}term\ (l \cdot \sigma), gterm\text{-}of\text{-}term\ (r \cdot \sigma)) \in gsubst\text{-}cl\ \mathcal{R}$
using *assms* **unfolding** *gsubst-cl-def* **by** *auto*

lemma *grstepD* [*simp*]:
$(s, t) \in \mathcal{R} \Longrightarrow (s, t) \in grstepD\ \mathcal{F}\ \mathcal{R}$
by (*auto simp*: *grstepD-def gctxtex-onp-def intro*!: *exI*[*of* - $\Box_G$])

lemma *grstepD-ctxtI* [*intro*]:
$(l, r) \in \mathcal{R} \Longrightarrow funas\text{-}gctxt\ C \subseteq \mathcal{F} \Longrightarrow (C\langle l\rangle_G, C\langle r\rangle_G) \in grstepD\ \mathcal{F}\ \mathcal{R}$
by (*auto simp*: *grstepD-def gctxtex-onp-def intro*!: *exI*[*of* - *C*])

lemma *gctxtex-funas-nroot-gctxtcl-funas-subseteq*:
gctxtex-funas-nroot $\mathcal{F}$ $(grstepD\ \mathcal{F}\ \mathcal{R}) \subseteq grstepD\ \mathcal{F}\ \mathcal{R}$
unfolding *grstepD-def*
by (*intro gctxtex-pred-cmp-subseteq*) *auto*

lemma *Restr-gnrrstepD-dist* [*simp*]:
Restr $(gnrrstepD\ \mathcal{F}\ \mathcal{R})\ (\mathcal{T}_G\ \mathcal{G}) = gnrrstepD\ (\mathcal{F} \cap \mathcal{G})\ (Restr\ \mathcal{R}\ (\mathcal{T}_G\ \mathcal{G}))$
by (*auto simp add*: *gnrrstepD-def*)

lemma *Restr-grstepD-dist* [*simp*]:
Restr $(grstepD\ \mathcal{F}\ \mathcal{R})\ (\mathcal{T}_G\ \mathcal{G}) = grstepD\ (\mathcal{F} \cap \mathcal{G})\ (Restr\ \mathcal{R}\ (\mathcal{T}_G\ \mathcal{G}))$
by (*auto simp add*: *grstepD-def*)

lemma *Restr-gpar-rstepD-dist* [*simp*]:
Restr $(gpar\text{-}rstepD\ \mathcal{F}\ \mathcal{R})\ (\mathcal{T}_G\ \mathcal{G}) = gpar\text{-}rstepD\ (\mathcal{F} \cap \mathcal{G})\ (Restr\ \mathcal{R}\ (\mathcal{T}_G\ \mathcal{G}))$ (**is** *?Ls* = *?Rs*)
by (*auto simp*: *gpar-rstepD-def*)

9.5 Equivalence lemmas

lemma *grrstep-subst-cl-conv*:
grrstep $\mathcal{R}$ = *gsubst-cl* $\mathcal{R}$
unfolding *gsubst-cl-def grrstep-def rrstep-def rstep-r-p-s-def*
by (*auto*, *metis ground-substI ground-term-of-gterm term-of-gterm-inv*) *blast*

lemma *gnrrstepD-gnrrstep-conv*:
gnrrstep $\mathcal{R}$ = *gnrrstepD UNIV* (*gsubst-cl* $\mathcal{R}$) (**is** *?Ls* = *?Rs*)
proof −
{fix *s t* **assume** $(s, t) \in$ *?Ls* **then obtain** *l r C* σ **where**
mem: $(l, r) \in \mathcal{R}$ $C \neq \Box$ *term-of-gterm* $s = C\langle l \cdot (\sigma :: 'b \Rightarrow ('a, 'b)\ term)\rangle$ *term-of-gterm* $t = C\langle r \cdot \sigma\rangle$
unfolding *gnrrstep-def inv-image-def nrrstep-def*′ **by** *auto*
then have $(s, t) \in$ *?Rs* **using** *gsubst-cl*[*OF mem*(*1*)]
using *gctxtex-onpI*[*of* λ *C*. *funas-gctxt* $C \subseteq UNIV \wedge C \neq \Box_G$ *gctxt-of-ctxt*

```
C gterm-of-term (l · σ)
      gterm-of-term (r · σ) gsubst-cl R]
    by (auto simp: gnrrstepD-def)}
 moreover
 {fix s t assume (s, t) ∈ ?Rs then have (s, t) ∈ ?Ls
  unfolding gnrrstepD-def gctxtex-onp-def gnrrstep-def inv-image-def nrrstep-def'
gsubst-cl-def
    by auto (metis ctxt-of-gctxt.simps(1) ctxt-of-gctxt-inv ground-ctxt-of-gctxt
ground-gctxt-of-ctxt-apply ground-substI)}
 ultimately show ?thesis by auto
qed

lemma grstepD-grstep-conv:
 grstep R = grstepD UNIV (gsubst-cl R) (is ?Ls = ?Rs)
proof −
 {fix s t assume (s, t) ∈ ?Ls then obtain C l r σ where
  mem: (l, r) ∈ R term-of-gterm s = C⟨l · (σ :: 'b ⇒ ('a, 'b) term)⟩ term-of-gterm
t = C⟨r · σ⟩
    unfolding grstep-def inv-image-def by auto
  then have (s, t) ∈ ?Rs using grstepD-ctxtI[OF gsubst-cl[OF mem(1)], of σ
gctxt-of-ctxt C UNIV]
    by (auto simp: grstepD-def gctxtex-onp-def)}
 moreover
 {fix s t assume (s, t) ∈ ?Rs then have (s, t) ∈ ?Ls
    by (auto simp: gctxtex-onp-def grstepD-def grstep-def gsubst-cl-def)
     (metis ctxt-of-gctxt-apply-gterm ground-ctxt-apply
      ground-ctxt-of-gctxt ground-substI gterm-of-term-inv rstep.intros)}
 ultimately show ?thesis by auto
qed

lemma gpar-rstep-gpar-rstepD-conv:
 gpar-rstep R = gpar-rstepD' UNIV (gsubst-cl R) (is ?Ls = ?Rs)
proof −
 {fix s t assume (s, t) ∈ ?Rs
  then have (s, t) ∈ gpar-rstep R
    by induct (auto simp: gpar-rstep-def gsubst-cl-def)}
 moreover
 {fix s t assume ass: (s, t) ∈ ?Ls then obtain u v where
    (u, v) ∈ par-rstep R u = term-of-gterm s v = term-of-gterm t
      by (simp add: gpar-rstep-def inv-image-def)
  then have (s, t) ∈ ?Rs
  proof (induct arbitrary: s t)
    case (root-step u v σ)
    then have (s, t) ∈ gsubst-cl R unfolding gsubst-cl-def
      by auto (metis ground-substI ground-term-of-gterm term-of-gterm-inv)
    then show ?case by auto
  next
    case (par-step-fun ts ss f)
    then show ?case by (cases s; cases t) auto
```

```
    next
      case (par-step-var x)
      then show ?case by (cases s) auto
  qed}
  ultimately show ?thesis by auto
qed

lemma gmctxtcl-funas-idem:
  gmctxtcl-funas F (gmctxtcl-funas F R) ⊆ gmctxtcl-funas F R
  by (intro gmctxtex-pred-cmp-subseteq)
    (auto elim!: less-eq-to-sup-mctxt-args, blast+)

lemma gpar-rstepD-gpar-rstepD'-conv:
  gpar-rstepD F R = gpar-rstepD' F R (is ?Ls = ?Rs)
proof −
  {fix s t assume (s, t) ∈ ?Rs then have (s, t) ∈ ?Ls
    proof induct
      case (groot-step s t) then show ?case unfolding gpar-rstepD-def
        using gmctxtex-onpI[of - GMHole  [s] [t]]
        by auto
    next
      case (gpar-step-fun ts ss f)
      show ?case using gpar-step-fun(2−) unfolding gpar-rstepD-def
        using subsetD[OF gmctxtcl-funas-idem, of (GFun f ss, GFun f ts) F R]
          using gmctxtex-onpI[of - GMFun f (replicate (length ss) GMHole) ss ts
gmctxtcl-funas F R]
        by (auto simp del: fill-gholes.simps)
    qed}
  moreover
  {fix s t assume (s, t) ∈ ?Ls then obtain C ss ts where
    t: s = fill-gholes C ss t = fill-gholes C ts and
    inv: num-gholes C = length ss num-gholes C = length ts and
    pred: funas-gmctxt C ⊆ F and rel: ∀ i < length ts. (ss ! i, ts ! i) ∈ R
      unfolding gpar-rstepD-def by auto
    have (s, t) ∈ ?Rs using inv pred rel unfolding t
    proof (induct rule: fill-gholes-induct2)
      case (GMHole x) then show ?case
        by (cases ts) auto
    next
      case (GMFun f Cs xs ys)
     from GMFun(1, 2, 5) have i < length Cs ⟹ ∀ j < length (partition-gholes
ys Cs ! i).
        (partition-gholes xs Cs ! i ! j, partition-gholes ys Cs ! i ! j) ∈ R for i
        by (auto simp: length-partition-by-nth partition-by-nth-nth(1, 2))
      from GMFun this show ?case unfolding partition-holes-fill-gholes-conv'
        by (intro gpar-step-fun) (auto, meson UN-I nth-mem subset-iff)
    qed}
  ultimately show ?thesis by auto
qed
```

9.6 Signature preserving lemmas

lemma $\mathcal{T}_G$*-trans-closure-id* [*simp*]:
$(\mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F})^+ = \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
by (*auto simp*: *trancl-full-on*)

lemma *signature-pres-funas-cl* [*simp*]:
$\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *gctxtcl-funas* $\mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
$\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *gmctxtcl-funas* $\mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
apply (*intro gctxtex-onp-in-signature*) **apply** *blast*+
apply (*intro gmctxtex-onp-in-signature*) **apply** *blast*+
done

lemma *relf-on-gmctxtcl-funas*:
assumes $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *refl-on* $(\mathcal{T}_G\ \mathcal{F})$ (*gmctxtcl-funas* $\mathcal{F}\ \mathcal{R}$)
proof −
have $t \in \mathcal{T}_G\ \mathcal{F} \Longrightarrow (t, t) \in$ *gmctxtcl-funas* $\mathcal{F}\ \mathcal{R}$ **for** t
using *gmctxtex-onpI*[*of* - *gmctxt-of-gterm t*]
by (*auto simp*: $\mathcal{T}_G$*-funas-gterm-conv*)
then show *?thesis* **using** *assms*
by (*auto simp*: *refl-on-def*)
qed

lemma *gtrancl-rel-sound*:
$\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *gtrancl-rel* $\mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
unfolding *gtrancl-rel-def*
by (*intro Restr-tracl-comp-simps*(*3*)) (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv*)

9.7 *gcomp-rel* and *gtrancl-rel* lemmas

lemma *gcomp-rel*:
lift-root-step $\mathcal{F}$ *PAny EParallel* (*gcomp-rel* $\mathcal{F}\ \mathcal{R}\ \mathcal{S}$) = *lift-root-step* $\mathcal{F}$ *PAny EParallel* $\mathcal{R}$ *O lift-root-step* $\mathcal{F}$ *PAny EParallel* $\mathcal{S}$ (**is** *?Ls* = *?Rs*)
proof
{ fix *s u* **assume** $(s, u) \in$ *gpar-rstepD'* $\mathcal{F}$ ($\mathcal{R}$ *O gpar-rstepD'* $\mathcal{F}\ \mathcal{S} \cup$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R}$ *O* $\mathcal{S}$)
then have $\exists t.\ (s, t) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R} \wedge (t, u) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{S}$
proof (*induct*)
case (*gpar-step-fun ts ss f*)
from *Ex-list-of-length-P*[*of* - $\lambda\ u\ i.\ (ss\ !\ i, u) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R} \wedge (u, ts\ !\ i) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{S}$]
obtain *us* **where** *l*: *length us* = *length ts* **and**
inv: $\forall\ i < length\ ts.\ (ss\ !\ i, us\ !\ i) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{R} \wedge (us\ !\ i, ts\ !\ i) \in$ *gpar-rstepD'* $\mathcal{F}\ \mathcal{S}$
using *gpar-step-fun*(*2*, *3*) **by** *blast*
then show *?case* **using** *gpar-step-fun*(*3*, *4*)
by (*auto intro*!: *exI*[*of* - *GFun f us*])
qed *auto***}**
then show *?Ls* ⊆ *?Rs* **unfolding** *gcomp-rel-def*

by (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv simp flip*: *gpar-rstepD-gpar-rstepD′-conv*[*unfolded gpar-rstepD-def*])
next
{fix *s t u* **assume** (*s*, *t*) ∈ *gpar-rstepD′ F R* (*t*, *u*) ∈ *gpar-rstepD′ F S*
then have (*s*, *u*) ∈ *gpar-rstepD′ F* (*R O gpar-rstepD′ F S* ∪ *gpar-rstepD′ F R O S*)
proof (*induct arbitrary*: *u rule*: *gpar-rstepD′.induct*)
case (*gpar-step-fun ts ss f*) **note** *IS* = *this*
show *?case*
proof (*cases* (*GFun f ts*, *u*) ∈ *S*)
case *True*
then have (*GFun f ss*, *u*) ∈ *gpar-rstepD′ F R O S* **using** *IS*(*1*, *3*, *4*)
by *auto*
then show *?thesis* **by** *auto*
next
case *False*
then obtain *us* **where** *u*[*simp*]: *u* = *GFun f us* **and** *l*: *length ts* = *length us*
using *IS*(*5*) **by** (*cases u*) (*auto elim*!: *gpar-rstepD′.cases*)
have *i* < *length us* ⟹
(*ss* ! *i*, *us* ! *i*) ∈ *gpar-rstepD′ F* (*R O gpar-rstepD′ F S* ∪ *gpar-rstepD′ F R O S*) **for** *i*
using *IS*(*2*, *5*) *False*
by (*auto elim*!: *gpar-rstepD′.cases*)
then show *?thesis* **using** *l IS*(*3*, *4*) **unfolding** *u*
by *auto*
qed
qed *auto***}**
then show *?Rs* ⊆ *?Ls*
by (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv gcomp-rel-def gpar-rstepD-gpar-rstepD′-conv*[*unfolded gpar-rstepD-def*])
qed

lemma *gmctxtcl-funas-in-rtrancl-gctxtcl-funas*:
assumes *R* ⊆ $\mathcal{T}_G$ *F* × $\mathcal{T}_G$ *F*
shows *gmctxtcl-funas F R* ⊆ (*gctxtcl-funas F R*)* **using** *assms*
by (*intro gmctxtex-onp-gctxtex-onp-rtrancl*) (*auto simp*: *gmctxt-p-inv-def*)

lemma *R-in-gtrancl-rel*:
assumes *R* ⊆ $\mathcal{T}_G$ *F* × $\mathcal{T}_G$ *F*
shows *R* ⊆ *gtrancl-rel F R*
proof
fix *s t* **assume** *ass*: (*s*, *t*) ∈ *R*
then have (*s*, *s*) ∈ *gmctxtcl-funas F R* (*t*, *t*) ∈ *gmctxtcl-funas F R* **using** *assms*
using *all-ctxt-closed-imp-reflx-on-sig*[*OF gmctxtcl-funas-sigcl*, *of F R*]
by *auto*
then show (*s*, *t*) ∈ *gtrancl-rel F R* **using** *ass*
by (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv relcomp-unfold gtrancl-rel-def*)
qed

lemma *trans-gtrancl-rel* [*simp*]:
trans (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$)
proof −
have (*s*, *t*) ∈ $\mathcal{R}$ ⟹ (*s*, *t*) ∈ *gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$ **for** *s t*
by (*metis bot.extremum funas-gmctxt.simps*(*2*) *gmctxtex-closure subsetD*)
then show *?thesis* **unfolding** *trans-def gtrancl-rel-def*
by (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv*, *meson relcomp3-I trancl-into-trancl2 trancl-trans*)
qed

lemma *gtrancl-rel-cl*:
assumes $\mathcal{R}$ ⊆ $\mathcal{T}_G$ $\mathcal{F}$ × $\mathcal{T}_G$ $\mathcal{F}$
shows *gmctxtcl-funas* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) ⊆ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$
proof −
have ∗:(*s*, *t*) ∈ $\mathcal{R}$ ⟹ (*s*, *t*) ∈ *gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$ **for** *s t*
by (*metis bot.extremum funas-gmctxt.simps*(*2*) *gmctxtex-closure subsetD*)
have *gmctxtcl-funas* $\mathcal{F}$ ((*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$) ⊆ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$
unfolding *gtrancl-rel-def* **using** *relf-on-gmctxtcl-funas*[*OF assms*]
by (*intro gmctxtex-onp-substep-trancl*, *intro gmctxtex-pred-cmp-subseteq2*)
(*auto simp*: *less-sup-gmctxt-args-funas-gmctxt refl-on-def*)
moreover have *gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$ ⊆ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$
unfolding *gtrancl-rel-def* **using** ∗
by (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv*, *meson trancl.trancl-into-trancl trancl-trans*)
ultimately show *?thesis* **using** *gmctxtex-onp-rel-mono* **by** *blast*
qed

lemma *gtrancl-rel-aux*:
$\mathcal{R}$ ⊆ $\mathcal{T}_G$ $\mathcal{F}$ × $\mathcal{T}_G$ $\mathcal{F}$ ⟹ *gmctxtcl-funas* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) *O gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$ ⊆ *gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$
$\mathcal{R}$ ⊆ $\mathcal{T}_G$ $\mathcal{F}$ × $\mathcal{T}_G$ $\mathcal{F}$ ⟹ *gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$ *O gmctxtcl-funas* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) ⊆ *gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$
using *subsetD*[*OF gtrancl-rel-cl*[*of* $\mathcal{R}$ $\mathcal{F}$]] **unfolding** *gtrancl-rel-def*
by (*auto simp*: *gmctxt-cl-gmctxtex-onp-conv*) (*meson relcomp3-I trancl-trans*)+

declare *subsetI* [*rule del*]
lemma *gtrancl-rel*:
assumes $\mathcal{R}$ ⊆ $\mathcal{T}_G$ $\mathcal{F}$ × $\mathcal{T}_G$ $\mathcal{F}$ *compatible-p Q P*
and ⋀ *C*. *P C* ⟹ *funas-gmctxt C* ⊆ $\mathcal{F}$
and ⋀ *C D*. *P C* ⟹ *P D* ⟹ (*C*, *D*) ∈ *comp-gmctxt* ⟹ *P* (*C* ⊓ *D*)
shows (*gctxtex-onp Q* $\mathcal{R}$)$^+$ ⊆ *gmctxtex-onp P* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$)
proof −
have *fst*: *gctxtex-onp Q* $\mathcal{R}$ ⊆ *gctxtex-onp Q* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$)
using *R-in-gtrancl-rel*[*OF assms*(*1*)]
by (*simp add*: *gctxtex-onp-rel-mono*)
have *snd*: *gctxtex-onp Q* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) ⊆ *gmctxtex-onp P* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$)
using *assms*(*2*)
by *auto*

have (*gmctxtex-onp P* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$))$^+$ = *gmctxtex-onp P* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$)
by (*intro gmctxtex-onp-substep-tranclE*[*of* - λ *C*. *funas-gmctxt C* $\subseteq$ $\mathcal{F}$])
(*auto simp*: *gtrancl-rel-aux*[*OF assms*(*1*)] *assms*(*3*, *4*) *intro*: *funas-gmctxt-poss-gmctxt-subgm-at-funas*)
then show *?thesis* **using** *subset-trans*[*OF fst snd*]
using *trancl-mono-set* **by** *fastforce*
qed

lemma *gtrancl-rel-subseteq-trancl-gctxtcl-funas*:
assumes $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *gtrancl-rel* $\mathcal{F}$ $\mathcal{R} \subseteq$ (*gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$
proof −
have [*dest*!]: (*s*, *t*) $\in$ $\mathcal{R} \Longrightarrow$ (*s*, *t*) $\in$ (*gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$ **for** *s t*
using *grstepD grstepD-def* **by** *blast*
have [*dest*!]: (*s*, *t*) $\in$ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+ \Longrightarrow$ (*s*, *t*) $\in$ (*gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$
$\cup$ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$)
for *s t*
using *gmctxtcl-funas-in-rtrancl-gctxtcl-funas*[*OF assms*]
using *signature-pres-funas-cl*[*OF assms*]
apply (*auto simp*: *gtrancl-rel-def rtrancl-eq-or-trancl intro*!: *subsetI*)
apply (*metis rtranclD rtrancl-trancl-absorb trancl-mono*)
apply (*metis mem-Sigma-iff trancl-full-on trancl-mono*)+
done
then show *?thesis* **using** *gtrancl-rel-sound*[*OF assms*]
by (*auto simp*: *gtrancl-rel-def rtrancl-eq-or-trancl gmctxt-cl-gmctxtex-onp-conv*
intro!: *subsetI*)
qed

lemma *gmctxtex-onp-gtrancl-rel*:
assumes $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$ **and** $\bigwedge$ *C D*. *Q C* $\Longrightarrow$ *funas-gctxt D* $\subseteq$ $\mathcal{F} \Longrightarrow$ *Q*
(*C* $\circ_{Gc}$ *D*)
and $\bigwedge$*C*. *P C* $\Longrightarrow$ *0* < *num-gholes C* $\wedge$ *funas-gmctxt C* $\subseteq$ $\mathcal{F}$
and $\bigwedge$*C*. *P C* $\Longrightarrow$ *gmctxt-p-inv C* $\mathcal{F}$ *Q*
shows *gmctxtex-onp P* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) $\subseteq$ (*gctxtex-onp Q* $\mathcal{R}$)$^+$
proof −
{fix *s t* **assume** *ass*: (*s*, *t*) $\in$ *gctxtex-onp Q* ((*gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$)
from *gctxtex-onpE*[*OF ass*] **obtain** *C u v* **where**
$*$: *s* = *C*$\langle$*u*$\rangle_G$ *t* = *C*$\langle$*v*$\rangle_G$ **and**
inv: *Q C* (*u*, *v*) $\in$ (*gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$ **by** *blast*
from *inv*(*2*) **have** (*s*, *t*) $\in$ (*gctxtex-onp Q* $\mathcal{R}$)$^+$ **unfolding** $*$
proof *induct*
case (*base y*)
then show *?case* **using** *assms*(*2*)[*OF inv*(*1*)]
by (*auto elim*!: *gctxtex-onpE*) (*metis ctxt-ctxt-compose gctxtex-onpI trancl.r-into-trancl*)
next
case (*step y z*)
from *step*(*2*) **have** (*C*$\langle$*y*$\rangle_G$, *C*$\langle$*z*$\rangle_G$) $\in$ *gctxtex-onp Q* $\mathcal{R}$ **using** *assms*(*2*)[*OF*
inv(*1*)]
by (*auto elim*!: *gctxtex-onpE*) (*metis ctxt-ctxt-compose gctxtex-onpI*)
then show *?case* **using** *step*(*3*)

```
      by auto
    qed}
  then have con: gctxtex-onp Q ((gctxtcl-funas F R)^+) ⊆ (gctxtex-onp Q R)^+
    using subrelI by blast
  have snd: gmctxtex-onp P ((gctxtcl-funas F R)^+) ⊆ (gctxtex-onp Q ((gctxtcl-funas
F R)^+))^+
    using assms(1)
    by (intro gmctxtex-onp-gctxtex-onp-trancl[OF assms(3) - assms(4)]) auto
  have fst: gmctxtex-onp P (gtrancl-rel F R) ⊆ gmctxtex-onp P ((gctxtcl-funas F
R)^+)
    using gtrancl-rel-subseteq-trancl-gctxtcl-funas[OF assms(1)]
    by (simp add: gmctxtex-onp-rel-mono)
  show ?thesis using subset-trans[OF fst snd] con
    by (auto intro!: subsetI)
      (metis (no-types, lifting) in-mono rtrancl-trancl-trancl tranclD2 trancl-mono
trancl-rtrancl-absorb)
qed

lemma gmctxtcl-funas-strict-gtrancl-rel:
  assumes R ⊆ T_G F × T_G F
  shows gmctxtcl-funas-strict F (gtrancl-rel F R) = (gctxtcl-funas F R)^+ (is ?Ls
= ?Rs)
proof
  show ?Ls ⊆ ?Rs
    by (intro gmctxtex-onp-gtrancl-rel[OF assms]) (auto simp: gmctxt-p-inv-def)
next
  show ?Rs ⊆ ?Ls
    by (intro gtrancl-rel[OF assms])
      (auto simp: compatible-p-def num-gholes-at-least1
        intro: subset-trans[OF inf-funas-gmctxt-subset2])
qed

lemma gmctxtex-funas-nroot-strict-gtrancl-rel:
  assumes R ⊆ T_G F × T_G F
  shows gmctxtex-funas-nroot-strict F (gtrancl-rel F R) = (gctxtex-funas-nroot F
R)^+
 (is ?Ls = ?Rs)
proof
  show ?Ls ⊆ ?Rs
    by (intro gmctxtex-onp-gtrancl-rel[OF assms])
      (auto simp: gmctxt-p-inv-def gmctxt-closing-def
       dest!: less-eq-gmctxt-Hole gctxt-of-gmctxt-hole-dest gctxt-compose-HoleE(1))
next
  show ?Rs ⊆ ?Ls
    by (intro gtrancl-rel[OF assms])
      (auto simp: compatible-p-def num-gholes-at-least1
        elim!: comp-gmctxt.cases
        dest: gmctxt-of-gctxt-GMHole-Hole
        intro: subset-trans[OF inf-funas-gmctxt-subset2])
```

qed

lemma *lift-root-step-sig'*:
assumes $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{G} \times \mathcal{T}_G\ \mathcal{H}\ \mathcal{F} \subseteq \mathcal{G}\ \mathcal{F} \subseteq \mathcal{H}$
shows *lift-root-step* $\mathcal{F}\ W\ X\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{G} \times \mathcal{T}_G\ \mathcal{H}$
using *assms* $\mathcal{T}_G$*-mono*
by (*cases* W; *cases* X) (*auto simp add*: *Sigma-mono* $\mathcal{T}_G$*-mono inf.coboundedI2*)

lemmas *lift-root-step-sig* = *lift-root-step-sig'*[*OF - subset-refl subset-refl*]

lemma *lift-root-step-incr*:
$\mathcal{R} \subseteq \mathcal{S} \Longrightarrow$ *lift-root-step* $\mathcal{F}\ W\ X\ \mathcal{R} \subseteq$ *lift-root-step* $\mathcal{F}\ W\ X\ \mathcal{S}$
by (*cases* W; *cases* X) (*auto simp add*: *le-supI1 gctxtex-onp-rel-mono gmctxtex-onp-rel-mono*)

lemma *Restr-id-mono*:
$\mathcal{F} \subseteq \mathcal{G} \Longrightarrow$ *Restr Id* $(\mathcal{T}_G\ \mathcal{F}) \subseteq$ *Restr Id* $(\mathcal{T}_G\ \mathcal{G})$
by (*meson Sigma-mono* $\mathcal{T}_G$*-mono inf-mono subset-refl*)

lemma *lift-root-step-mono*:
$\mathcal{F} \subseteq \mathcal{G} \Longrightarrow$ *lift-root-step* $\mathcal{F}\ W\ X\ \mathcal{R} \subseteq$ *lift-root-step* $\mathcal{G}\ W\ X\ \mathcal{R}$
by (*cases* W; *cases* X) (*auto simp*: *Restr-id-mono intro*: *gmctxtex-onp-mono gctxtex-onp-mono*,
metis Restr-id-mono sup.coboundedI1 sup-commute)

lemma *grstep-lift-root-step*:
lift-root-step $\mathcal{F}$ *PAny ESingle* (*Restr* (*grrstep* $\mathcal{R}$) $(\mathcal{T}_G\ \mathcal{F})$) = *Restr* (*grstep* $\mathcal{R}$) $(\mathcal{T}_G\ \mathcal{F})$
unfolding *grstepD-grstep-conv grstepD-def grrstep-subst-cl-conv*
by *auto*

lemma *prod-swap-id-on-refl* [*simp*]:
Restr Id $(\mathcal{T}_G\ \mathcal{F}) \subseteq$ *prod.swap* ‘ $(\mathcal{R} \cup$ *Restr Id* $(\mathcal{T}_G\ \mathcal{F}))$
by (*auto intro*: *subsetI*)

lemma *swap-lift-root-step*:
lift-root-step $\mathcal{F}\ W\ X$ (*prod.swap* ‘ $\mathcal{R}$) = *prod.swap* ‘ *lift-root-step* $\mathcal{F}\ W\ X\ \mathcal{R}$
by (*cases* W; *cases* X) (*auto simp add*: *image-mono swap-gmctxtex-onp swap-gctxtex-onp intro*: *subsetI*)

lemma *converse-lift-root-step*:
(*lift-root-step* $\mathcal{F}\ W\ X\ R)^{-1}$ = *lift-root-step* $\mathcal{F}\ W\ X\ (R^{-1})$
by (*cases* W; *cases* X) (*auto simp add*: *converse-gctxtex-onp converse-gmctxtex-onp intro*: *subsetI*)

lemma *lift-root-step-sig-transfer*:
assumes $p \in$ *lift-root-step* $\mathcal{F}\ W\ X\ R$ *snd* ‘ $R \subseteq \mathcal{T}_G\ \mathcal{F}$ *funas-gterm* (*fst* p) $\subseteq \mathcal{G}$
shows $p \in$ *lift-root-step* $\mathcal{G}\ W\ X\ R$ **using** *assms*

proof −
from *assms* **have** $p \in$ *lift-root-step* $(\mathcal{F} \cap \mathcal{G})$ W X R
by (*cases p*; *cases* W; *cases* X)
(*auto simp*: *gctxtex-onp-sign-trans-fst*[*of* - - - R $\mathcal{G}$] *gctxtex-onp-sign-trans-snd*[*of* - - - R $\mathcal{G}$]
gmctxtex-onp-sign-trans-fst gmctxtex-onp-sign-trans-snd simp flip: $\mathcal{T}_G$*-equivalent-def* $\mathcal{T}_G$*-funas-gterm-conv*
intro: *basic-trans-rules*(*30*)[*OF gctxtex-onp-sign-trans-fst*[*of* - - - R $\mathcal{G}$],
where *?B* = *gctxtex-onp* P R **for** P]
basic-trans-rules(*30*)[*OF gmctxtex-onp-sign-trans-fst*[*of* - - - R $\mathcal{G}$],
where *?B* = *gmctxtex-onp* P R **for** P])
then show *?thesis*
by (*meson inf.cobounded2 lift-root-step-mono subsetD*)
qed

lemma *lift-root-step-sig-transfer2*:
assumes $p \in$ *lift-root-step* $\mathcal{F}$ W X R *snd* ‘ $R \subseteq \mathcal{T}_G$ $\mathcal{G}$ *funas-gterm* (*fst p*) $\subseteq \mathcal{G}$
shows $p \in$ *lift-root-step* $\mathcal{G}$ W X R
proof −
from *assms* **have** $p \in$ *lift-root-step* $(\mathcal{F} \cap \mathcal{G})$ W X R
by (*cases p*; *cases* W; *cases* X)
(*auto simp*: *gctxtex-onp-sign-trans-fst*[*of* - - - R $\mathcal{G}$] *gctxtex-onp-sign-trans-snd*[*of* - - - R $\mathcal{G}$]
gmctxtex-onp-sign-trans-fst gmctxtex-onp-sign-trans-snd simp flip: $\mathcal{T}_G$*-equivalent-def* $\mathcal{T}_G$*-funas-gterm-conv*
intro: *basic-trans-rules*(*30*)[*OF gctxtex-onp-sign-trans-fst*[*of* - - - R $\mathcal{G}$],
where *?B* = *gctxtex-onp* P R **for** P]
basic-trans-rules(*30*)[*OF gmctxtex-onp-sign-trans-fst*[*of* - - - R $\mathcal{G}$],
where *?B* = *gmctxtex-onp* P R **for** P])
then show *?thesis*
by (*meson inf.cobounded2 lift-root-step-mono subsetD*)
qed

lemma *lift-root-steps-sig-transfer*:
assumes $(s, t) \in$ (*lift-root-step* $\mathcal{F}$ W X R)$^+$ *snd* ‘ $R \subseteq \mathcal{T}_G$ $\mathcal{G}$ *funas-gterm* $s \subseteq \mathcal{G}$
shows $(s, t) \in$ (*lift-root-step* $\mathcal{G}$ W X R)$^+$
using *assms*(*1*,*3*)
proof (*induct rule*: *converse-trancl-induct*)
case (*base s*)
show *?case* **using** *lift-root-step-sig-transfer2*[*OF base*(*1*) *assms*(*2*)] *base*(*2*) **by** (*simp add*: *r-into-trancl*)
next
case (*step s s'*)
show *?case* **using** *lift-root-step-sig-transfer2*[*OF step*(*1*) *assms*(*2*)] *step*(*3*,*4*)
lift-root-step-sig'[*of* R *UNIV* $\mathcal{G}$ $\mathcal{G}$ W X, *THEN subsetD*, *of* (s, s')] *assms*(*2*)
by (*auto simp*: $\mathcal{T}_G$*-funas-gterm-conv* $\mathcal{T}_G$*-equivalent-def*)
(*smt* (*verit*) *SigmaI UNIV-I image-subset-iff snd-conv subrelI trancl-into-trancl2*)
qed

lemma *lift-root-stepseq-sig-transfer*:
assumes $(s, t) \in (\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ X\ R)^*$ $\mathit{snd}\ `\ R \subseteq \mathcal{T}_G\ \mathcal{G}\ \mathit{funas\text{-}gterm}\ s \subseteq \mathcal{G}$
shows $(s, t) \in (\mathit{lift\text{-}root\text{-}step}\ \mathcal{G}\ W\ X\ R)^*$
using *assms* **by** (*auto simp flip*: *reflcl-trancl simp*: *lift-root-steps-sig-transfer*)

lemmas *lift-root-step-sig-transfer'* = *lift-root-step-sig-transfer*[*of prod.swap p* $\mathcal{F}$ *W X prod.swap* ` *R* $\mathcal{G}$ **for** *p* $\mathcal{F}$ *W X* $\mathcal{G}$ *R*,
unfolded swap-lift-root-step, *OF imageI*, *THEN imageI* [*of* - - *prod.swap*],
unfolded image-comp comp-def fst-swap snd-swap swap-swap image-ident]

lemmas *lift-root-steps-sig-transfer'* = *lift-root-steps-sig-transfer*[*of t s* $\mathcal{F}$ *W X prod.swap* ` *R* $\mathcal{G}$ **for** *t s* $\mathcal{F}$ *W X* $\mathcal{G}$ *R*,
THEN imageI [*of* - - *prod.swap*], *unfolded swap-lift-root-step swap-trancl pair-in-swap-image*
image-comp comp-def snd-swap swap-swap swap-simp image-ident]

lemmas *lift-root-stepseq-sig-transfer'* = *lift-root-stepseq-sig-transfer*[*of t s* $\mathcal{F}$ *W X prod.swap* ` *R* $\mathcal{G}$ **for** *t s* $\mathcal{F}$ *W X* $\mathcal{G}$ *R*,
THEN imageI [*of* - - *prod.swap*], *unfolded swap-lift-root-step swap-rtrancl pair-in-swap-image*
image-comp comp-def snd-swap swap-swap swap-simp image-ident]

lemma *lift-root-step-PRoot-ESingle* [*simp*]:
$\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ \mathit{PRoot}\ \mathit{ESingle}\ \mathcal{R} = \mathcal{R}$
by *auto*

lemma *lift-root-step-PRoot-EStrictParallel* [*simp*]:
$\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ \mathit{PRoot}\ \mathit{EStrictParallel}\ \mathcal{R} = \mathcal{R}$
by *auto*

lemma *lift-root-step-Parallel-conv*:
shows $\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ \mathit{EParallel}\ \mathcal{R} = \mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ \mathit{EStrictParallel}\ \mathcal{R} \cup \mathit{Restr}\ \mathit{Id}\ (\mathcal{T}_G\ \mathcal{F})$
by (*cases W*) (*auto simp*: *gmctxtcl-funas-dist gmctxtex-funas-nroot-dist*)

lemma *relax-pos-lift-root-step*:
$\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ X\ R \subseteq \mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ \mathit{PAny}\ X\ R$
by (*cases W*; *cases X*) (*auto simp*: *gctxtex-closure gmctxtex-closure*)

lemma *relax-pos-lift-root-steps*:
$(\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ X\ R)^+ \subseteq (\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ \mathit{PAny}\ X\ R)^+$
by (*simp add*: *relax-pos-lift-root-step trancl-mono-set*)

lemma *relax-ext-lift-root-step*:
$\mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ X\ R \subseteq \mathit{lift\text{-}root\text{-}step}\ \mathcal{F}\ W\ \mathit{EParallel}\ R$
by (*cases W*; *cases X*) (*auto simp*: *compatible-p-gctxtex-gmctxtex-subseteq*)

lemma *lift-root-step-StrictParallel-seq*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$

shows *lift-root-step* $\mathcal{F}$ *PAny EStrictParallel* $R \subseteq$ (*lift-root-step* $\mathcal{F}$ *PAny ESingle* $R)^+$
using *assms*
by (*auto simp*: *gmctxt-p-inv-def intro*!: *gmctxtex-onp-gctxtex-onp-trancl*)

lemma *lift-root-step-Parallel-seq*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *lift-root-step* $\mathcal{F}$ *PAny EParallel* $R \subseteq$ (*lift-root-step* $\mathcal{F}$ *PAny ESingle* $R)^+$ $\cup$ *Restr Id* ($\mathcal{T}_G\ \mathcal{F}$)
unfolding *lift-root-step-Parallel-conv* **using** *lift-root-step-StrictParallel-seq*[*OF assms*]
using *Un-mono* **by** *blast*

lemma *lift-root-step-Single-to-Parallel*:
shows *lift-root-step* $\mathcal{F}$ *PAny ESingle* $R \subseteq$ *lift-root-step* $\mathcal{F}$ *PAny EParallel* R
by (*simp add*: *compatible-p-gctxtex-gmctxtex-subseteq*)

lemma *trancl-partial-reflcl*:
($X \cup$ *Restr Id* $Y)^+ = X^+ \cup$ *Restr Id* Y
proof (*intro equalityI subrelI*, *goal-cases LR RL*)
case (*LR a b*) **then show** *?case* **by** (*induct*) (*auto dest*: *trancl-into-trancl*)
qed (*auto intro*: *trancl-mono*)

lemma *lift-root-step-Parallels-single*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows (*lift-root-step* $\mathcal{F}$ *PAny EParallel* $R)^+ =$ (*lift-root-step* $\mathcal{F}$ *PAny ESingle* $R)^+ \cup$ *Restr Id* ($\mathcal{T}_G\ \mathcal{F}$)
using *trancl-mono-set*[*OF lift-root-step-Parallel-seq*[*OF assms*]]
using *trancl-mono-set*[*OF lift-root-step-Single-to-Parallel*, *of* $\mathcal{F}$ R]
by (*auto simp*: *lift-root-step-Parallel-conv trancl-partial-reflcl*)

lemma *lift-root-Any-Single-eq*:
shows *lift-root-step* $\mathcal{F}$ *PAny ESingle* $R = R \cup$ *lift-root-step* $\mathcal{F}$ *PNonRoot ESingle* R
by (*auto simp*: *gctxtcl-funas-dist intro*!: *gctxtex-closure*)

lemma *lift-root-Any-EStrict-eq* [*simp*]:
shows *lift-root-step* $\mathcal{F}$ *PAny EStrictParallel* $R = R \cup$ *lift-root-step* $\mathcal{F}$ *PNonRoot EStrictParallel* R
by (*auto simp*: *gmctxtcl-funas-strict-dist*)

lemma *gar-rstep-lift-root-step*:
lift-root-step $\mathcal{F}$ *PAny EParallel* (*Restr* (*grrstep* $\mathcal{R}$) ($\mathcal{T}_G\ \mathcal{F}$)) = *Restr* (*gpar-rstep* $\mathcal{R}$) ($\mathcal{T}_G\ \mathcal{F}$)
unfolding *grrstep-subst-cl-conv gpar-rstep-gpar-rstepD-conv*
unfolding *gpar-rstepD-gpar-rstepD'-conv*[*symmetric*]
by (*auto simp*: *gpar-rstepD-def*)

lemma *grrstep-lift-root-gnrrstep*:
lift-root-step $\mathcal{F}$ *PNonRoot ESingle* (*Restr* (*grrstep* $\mathcal{R}$) ($\mathcal{T}_G$ $\mathcal{F}$)) = *Restr* (*gnrrstep* $\mathcal{R}$) ($\mathcal{T}_G$ $\mathcal{F}$)
unfolding *gnrrstepD-gnrrstep-conv grrstep-subst-cl-conv*
by (*simp add*: *gnrrstepD-def*)

declare *subsetI* [*intro*!]
declare *lift-root-step.simps*[*simp del*]

lemma *gpar-rstepD-grstepD-rtrancl-subseteq*:
assumes $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *gpar-rstepD* $\mathcal{F}$ $\mathcal{R} \subseteq$ (*grstepD* $\mathcal{F}$ $\mathcal{R}$)*
using *assms* **unfolding** *gpar-rstepD-def grstepD-def*
by (*intro gmctxtex-onp-gctxtex-onp-rtrancl*) (*auto simp*: $\mathcal{T}_G$*-equivalent-def gmctxt-p-inv-def*)
end
theory *Context-RR2*
imports *Context-Extensions*
Ground-MCtxt
Regular-Tree-Relations.RRn-Automata
begin

9.8 Auxiliary lemmas

lemma *gpair-gctxt*:
assumes *gpair s t* = *u*
shows (*map-gctxt* (λ *f* .(*Some f*, *Some f*)) *C*)⟨*u*⟩$_G$ = *gpair C*⟨*s*⟩$_G$ *C*⟨*t*⟩$_G$ **using** *assms*
by (*induct C arbitrary*: *s t u*) (*auto simp add*: *gpair-context1 comp-def map-funs-term-some-gpair intro*!: *nth-equalityI*)

lemma *gpair-gctxt′*:
assumes *gpair C*⟨*v*⟩$_G$ *C*⟨*w*⟩$_G$ = *u*
shows *u* = (*map-gctxt* (λ *f* .(*Some f*, *Some f*)) *C*)⟨*gpair v w*⟩$_G$
using *assms* **by** (*simp add*: *gpair-gctxt*)

lemma *gpair-gmctxt*:
assumes ∀ *i* < *length us*. *gpair* (*ss* ! *i*) (*ts* ! *i*) = *us* ! *i*
and *num-gholes C* = *length ss length ss* = *length ts length ts* = *length us*
shows *fill-gholes* (*map-gmctxt* (λ*f* . (*Some f*, *Some f*)) *C*) *us* = *gpair* (*fill-gholes C ss*) (*fill-gholes C ts*)
using *assms*
proof (*induct C arbitrary*: *ss ts us*)
case *GMHole*
then show *?case* **by** (*cases ss*; *cases ts*; *cases us*) *auto*
next
case (*GMFun f Cs*)
show *?case* **using** *GMFun*(*2*−)

using *GMFun(1)*[*OF nth-mem, of i partition-gholes us Cs ! i partition-gholes ss Cs ! i partition-gholes ts Cs ! i* **for** *i*]
using *length-partition-gholes-nth*[*of Cs*] *partition-by-nth-nth*[*of map num-gholes Cs us*]
using *partition-by-nth-nth*[*of map num-gholes Cs ss*] *partition-by-nth-nth*[*of map num-gholes Cs ts*]
by (*auto simp*: *partition-holes-fill-gholes-conv' gpair-context1 simp del*: *fill-gholes.simps intro*!: *nth-equalityI*)
(*simp add*: *length-partition-gholes-nth*)
qed

lemma *gctxtex-onp-gpair-set-conv*:
{*gpair t u* |*t u*. (*t*, *u*) ∈ *gctxtex-onp P* ℛ} =
{(*map-gctxt* (λ *f* .(*Some f*, *Some f*)) *C*)⟨*s*⟩$_G$ | *C s*. *P C* ∧ *s* ∈ {*gpair t u* |*t u*. (*t*, *u*) ∈ ℛ}} (**is** *?Ls* = *?Rs*)
proof
show *?Ls* ⊆ *?Rs* **using** *gpair-gctxt'*
by (*auto elim*!: *gctxtex-onpE*) *blast*
next
show *?Rs* ⊆ *?Ls*
by (*auto simp add*: *gctxtex-onpI gpair-gctxt*)
qed

lemma *gmctxtex-onp-gpair-set-conv*:
{*gpair t u* |*t u*. (*t*, *u*) ∈ *gmctxtex-onp P* ℛ} =
{*fill-gholes* (*map-gmctxt* (λ *f* .(*Some f*, *Some f*)) *C*) *ss* | *C ss*. *num-gholes C* = *length ss* ∧ *P C* ∧
(∀ *i* < *length ss*. *ss* ! *i* ∈ {*gpair t u* |*t u*. (*t*, *u*) ∈ ℛ})} (**is** *?Ls* = *?Rs*)
proof
{fix *u* **assume** *u* ∈ *?Ls* **then obtain** *s t*
where ∗: *u* = *gpair s t* (*s*, *t*) ∈ *gmctxtex-onp P* ℛ
by *auto*
from *gmctxtex-onpE*[*OF* ∗(*2*)] **obtain** *C us vs* **where**
∗∗: *s* = *fill-gholes C us t* = *fill-gholes C vs* **and**
inv: *num-gholes C* = *length us length us* = *length vs P C*
∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ ℛ
by *blast*
define *ws* **where** *ws* ≡ *map2 gpair us vs*
from *inv*(*1*, *2*) **have** ∀ *i* < *length ws*. *gpair* (*us* ! *i*) (*vs* ! *i*) = *ws* ! *i*
by(*auto simp*: *ws-def*)
from *gpair-gmctxt*[*OF this inv*(*1*, *2*)] *inv*
have *u* ∈ *?Rs* **unfolding** ∗ ∗∗
by (*auto simp*: *ws-def intro*!: *exI*[*of - ws*] *exI*[*of - C*])**}**
then show *?Ls* ⊆ *?Rs* **by** *blast*
next
{fix *u* **assume** *u* ∈ *?Rs* **then obtain** *C ss* **where**
∗: *u* = *fill-gholes* (*map-gmctxt* (λ*f*. (*Some f*, *Some f*)) *C*) *ss* **and**

inv: *P C num-gholes C = length ss* ∀ *i < length ss*. ∃ *t u*. *ss* ! *i = gpair t u* ∧ (*t*, *u*) ∈ $\mathcal{R}$
by *auto*
define *us* **where** *us* ≡ *map gfst ss* **define** *vs* **where** *vs* ≡ *map gsnd ss*
then have *len*: *length ss = length us length us = length vs* **and**
rec: ∀ *i < length ss*. *gpair* (*us* ! *i*) (*vs* ! *i*) = *ss* ! *i*
∀ *i < length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
by (*auto simp*: *us-def vs-def*) (*metis gfst-gpair gsnd-gpair inv*(*3*))+
from *len* **have** *l*: *length vs = length ss* **by** *auto*
have *u* ∈ *?Ls* **unfolding** ∗ **using** *inv*(*2*) *len*
using *gmctxtex-onpI*[*of P C us vs* $\mathcal{R}$, *OF inv*(*1*) - *len*(*2*) *rec*(*2*)]
using *gpair-gmctxt*[*OF rec*(*1*) - *len*(*2*) *l*, *of C*]
by *simp*}
then show *?Rs* ⊆ *?Ls* **by** *blast*
qed

abbreviation *lift-sig-RR2* ≡ λ (*f*, *n*). ((*Some f*, *Some f*), *n*)
abbreviation *lift-fun* ≡ (λ *f*. (*Some f*, *Some f*))
abbreviation *unlift-fst* ≡ (λ *f*. *the* (*fst f*))
abbreviation *unlift-snd* ≡ (λ *f*. *the* (*snd f*))

lemma *RR2-gterm-unlift-lift-id* [*simp*]:
funas-gterm t ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *map-gterm* (*lift-fun* ∘ *unlift-fst*) *t = t*
by (*induct t*) (*auto simp add*: *SUP-le-iff map-idI*)

lemma *RR2-gterm-unlift-funas* [*simp*]:
funas-gterm t ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *funas-gterm* (*map-gterm unlift-fst t*) ⊆ $\mathcal{F}$
by (*induct t*) (*auto simp add*: *SUP-le-iff map-idI*)

lemma *gterm-funas-lift-RR2-funas* [*simp*]:
funas-gterm t ⊆ $\mathcal{F}$ ⟹ *funas-gterm* (*map-gterm lift-fun t*) ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
by (*induct t*) (*auto simp add*: *SUP-le-iff map-idI*)

lemma *RR2-gctxt-unlift-lift-id* [*simp*, *intro*]:
funas-gctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ (*map-gctxt* (*lift-fun* ∘ *unlift-fst*) *C*) = *C*
by (*induct C*) (*auto simp add*: *all-set-conv-all-nth SUP-le-iff map-idI intro*!: *nth-equalityI*)

lemma *RR2-gctxt-unlift-funas* [*simp*, *intro*]:
funas-gctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *funas-gctxt* (*map-gctxt unlift-fst C*) ⊆ $\mathcal{F}$
by (*induct C*) (*auto simp add*: *all-set-conv-all-nth SUP-le-iff map-idI intro*!: *nth-equalityI*)

lemma *gctxt-funas-lift-RR2-funas* [*simp*, *intro*]:
funas-gctxt C ⊆ $\mathcal{F}$ ⟹ *funas-gctxt* (*map-gctxt lift-fun C*) ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
by (*induct C*) (*auto simp add*: *all-set-conv-all-nth SUP-le-iff map-idI intro*!:

nth-equalityI)

lemma *RR2-gmctxt-unlift-lift-id* [*simp*, *intro*]:
funas-gmctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ (*map-gmctxt* (*lift-fun* ∘ *unlift-fst*) *C*) = *C*
by (*induct C*) (*auto simp add*: *all-set-conv-all-nth SUP-le-iff map-idI intro*!: *nth-equalityI*)

lemma *RR2-gmctxt-unlift-funas* [*simp*, *intro*]:
funas-gmctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *funas-gmctxt* (*map-gmctxt unlift-fst C*) ⊆ $\mathcal{F}$
by (*induct C*) (*auto simp add*: *all-set-conv-all-nth SUP-le-iff map-idI intro*!: *nth-equalityI*)

lemma *gmctxt-funas-lift-RR2-funas* [*simp*, *intro*]:
funas-gmctxt C ⊆ $\mathcal{F}$ ⟹ *funas-gmctxt* (*map-gmctxt lift-fun C*) ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
by (*induct C*) (*auto simp add*: *all-set-conv-all-nth SUP-le-iff map-idI intro*!: *nth-equalityI*)

lemma *RR2-gctxt-cl-to-gctxt*:
assumes ⋀ *C*. *P C* ⟹ *funas-gctxt C* ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
and ⋀ *C*. *P C* ⟹ *R* (*map-gctxt unlift-fst C*)
and ⋀ *C*. *R C* ⟹ *P* (*map-gctxt lift-fun C*)
shows {*C*⟨*s*⟩$_G$ |*C s*. *P C* ∧ *Q s*} = {(*map-gctxt lift-fun C*)⟨*s*⟩$_G$ |*C s*. *R C* ∧ *Q s*} (**is** *?Ls* = *?Rs*)
proof
{fix *u* **assume** *u* ∈ *?Ls* **then obtain** *C s* **where**
∗:*u* = *C*⟨*s*⟩$_G$ **and** *inv*: *P C Q s* **by** *blast*
then have *funas-gctxt C* ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ **using** *assms* **by** *auto*
from *RR2-gctxt-unlift-lift-id*[*OF this*] **have** *u* ∈ *?Rs* **using** *inv assms* **unfolding** ∗
by (*auto intro*!: *exI*[*of - map-gctxt unlift-fst C*] *exI*[*of - s*])**}**
then show *?Ls* ⊆ *?Rs* **by** *blast*
next
{fix *u* **assume** *u* ∈ *?Rs* **then obtain** *C s* **where**
∗:*u* = (*map-gctxt lift-fun C*)⟨*s*⟩$_G$ **and** *inv*: *R C Q s*
by *blast*
have *u* ∈ *?Ls* **unfolding** ∗ **using** *inv assms*
by (*auto intro*!: *exI*[*of - map-gctxt lift-fun C*])**}**
then show *?Rs* ⊆ *?Ls* **by** *blast*
qed

lemma *RR2-gmctxt-cl-to-gmctxt*:
assumes ⋀ *C*. *P C* ⟹ *funas-gmctxt C* ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
and ⋀ *C*. *P C* ⟹ *R* (*map-gmctxt* (λ *f*. *the* (*fst f*)) *C*)
and ⋀ *C*. *R C* ⟹ *P* (*map-gmctxt* (λ *f*. (*Some f*, *Some f*)) *C*)
shows {*fill-gholes C ss* |*C ss*. *num-gholes C* = *length ss* ∧ *P C* ∧ (∀ *i* < *length ss*. *Q* (*ss* ! *i*))} =

{*fill-gholes* (*map-gmctxt* (λ*f*. (*Some f*, *Some f*)) *C*) *ss* |*C ss*. *num-gholes C* = *length ss* ∧
R C ∧ (∀ *i* < *length ss*. *Q* (*ss* ! *i*))} (**is** *?Ls* = *?Rs*)
proof
{fix *u* **assume** *u* ∈ *?Ls* **then obtain** *C ss* **where**
∗:*u* = *fill-gholes C ss* **and** *inv*: *num-gholes C* = *length ss P C* ∀ *i* < *length ss*. *Q* (*ss* ! *i*)
by *blast*
then have *funas-gmctxt C* ⊆ *lift-sig-RR2* ‘ *F* **using** *assms* **by** *auto*
from *RR2-gmctxt-unlift-lift-id*[*OF this*] **have** *u* ∈ *?Rs* **using** *inv assms* **unfolding** ∗
by (*auto intro*!: *exI*[*of - map-gmctxt unlift-fst C*] *exI*[*of - ss*])**}**
then show *?Ls* ⊆ *?Rs* **by** *blast*
next
{fix *u* **assume** *u* ∈ *?Rs* **then obtain** *C ss* **where**
∗:*u* = *fill-gholes* (*map-gmctxt lift-fun C*) *ss* **and** *inv*: *num-gholes C* = *length ss R C*
∀ *i* < *length ss*. *Q* (*ss* ! *i*)
by *blast*
have *u* ∈ *?Ls* **unfolding** ∗ **using** *inv assms*
by (*auto intro*!: *exI*[*of - map-gmctxt lift-fun C*])**}**
then show *?Rs* ⊆ *?Ls* **by** *blast*
qed

lemma *RR2-id-terms-gpair-set* [*simp*]:
$\mathcal{T}_G$ (*lift-sig-RR2* ‘ *F*) = {*gpair t u* |*t u*. (*t*, *u*) ∈ *Restr Id* ($\mathcal{T}_G$ *F*)}
apply (*auto simp*: *map-funs-term-some-gpair* $\mathcal{T}_G$*-equivalent-def*)
apply (*smt* (*verit*) *RR2-gterm-unlift-funas RR2-gterm-unlift-lift-id gterm.map-comp*)
using *funas-gterm-map-gterm* **by** *blast*

end
theory *GTT-RRn*
imports *Regular-Tree-Relations.GTT*
TA-Clousure-Const
Context-RR2
Lift-Root-Step
begin

10 Connecting regular tree languages to set/relation specifications

abbreviation *ggtt-lang* **where**
ggtt-lang F G ≡ *map-both gterm-of-term* ‘ (*Restr* (*gtt-lang-terms G*) {*t*. *funas-term t* ⊆ *fset F*})

lemma *ground-mctxt-map-vars-mctxt* [*simp*]:
ground-mctxt (*map-vars-mctxt f C*) = *ground-mctxt C*
by (*induct C*) *auto*

lemma *root-single-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* $\mathcal{A}$ (*lift-root-step* $\mathcal{F}$ *PRoot ESingle R*)
using *assms* **unfolding** *RR2-spec-def*
by (*auto simp*: *lift-root-step.simps*)

lemma *root-strictparallel-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* $\mathcal{A}$ (*lift-root-step* $\mathcal{F}$ *PRoot EStrictParallel R*)
using *assms* **unfolding** *RR2-spec-def*
by (*auto simp*: *lift-root-step.simps*)

lemma *reflcl-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*reflcl-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PRoot EParallel R*)
unfolding *RR2-spec-def* $\mathcal{L}$*-reflcl-reg*
unfolding *lift-root-step.simps* $\mathcal{T}_G$*-equivalent-def assms*[*unfolded RR2-spec-def*]
by (*auto simp flip*: $\mathcal{T}_G$*-equivalent-def*)

lemma *parallel-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*parallel-closure-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PAny EParallel R*)
unfolding *RR2-spec-def parallelcl-gmctxt-lang lift-root-step.simps*
unfolding *gmctxtex-onp-gpair-set-conv assms*[*unfolded RR2-spec-def*]
by (*intro RR2-gmctxt-cl-to-gmctxt*) *auto*

lemma *ctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*ctxt-closure-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PAny ESingle R*)
unfolding *RR2-spec-def gctxt-closure-lang lift-root-step.simps*
unfolding *gctxtex-onp-gpair-set-conv assms*[*unfolded RR2-spec-def*]
by (*intro RR2-gctxt-cl-to-gctxt*) *auto*

lemma *mctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*mctxt-closure-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PAny EStrictParallel R*)
unfolding *RR2-spec-def gmctxt-closure-lang lift-root-step.simps*
unfolding *gmctxtex-onp-gpair-set-conv assms*[*unfolded RR2-spec-def*] *conj-assoc*
by (*intro RR2-gmctxt-cl-to-gmctxt*[**where** *?P* = λ *C*. *0* < *num-gholes C* $\wedge$ *funas-gmctxt C* $\subseteq$ *fset* (*lift-sig-RR2* |`| $\mathcal{F}$) **and**
?R = λ *C*. *0* < *num-gholes C* $\wedge$ *funas-gmctxt C* $\subseteq$ *fset* $\mathcal{F}$, *unfolded conj-assoc*]) *auto*

lemma *nhole-ctxt-closure-automaton*:

assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*nhole-ctxt-closure-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PNonRoot ESingle R*)
unfolding *RR2-spec-def nhole-ctxtcl-lang lift-root-step.simps*
unfolding *gctxtex-onp-gpair-set-conv assms*[*unfolded RR2-spec-def*]
by (*intro RR2-gctxt-cl-to-gctxt*[**where**
?P = λ *C*. *C* $\neq$ $\square_G$ $\wedge$ *funas-gctxt C* $\subseteq$ *fset* (*lift-sig-RR2* |`| $\mathcal{F}$), *unfolded conj-assoc*]) *auto*

lemma *nhole-mctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*nhole-mctxt-closure-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PNonRoot EStrictParallel R*)
unfolding *RR2-spec-def nhole-gmctxt-closure-lang lift-root-step.simps*
unfolding *gmctxtex-onp-gpair-set-conv assms*[*unfolded RR2-spec-def*]
by (*intro RR2-gmctxt-cl-to-gmctxt*[**where**
?P = λ *C*. *0* < *num-gholes C* $\wedge$ *C* $\neq$ *GMHole* $\wedge$ *funas-gmctxt C* $\subseteq$ *fset* (*lift-sig-RR2* |`| $\mathcal{F}$), *unfolded conj-assoc*])
auto

lemma *nhole-mctxt-reflcl-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*nhole-mctxt-reflcl-reg* (*lift-sig-RR2* |`| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PNonRoot EParallel R*)
using *nhole-mctxt-closure-automaton*[*OF assms*, *of* $\mathcal{F}$]
unfolding *RR2-spec-def lift-root-step-Parallel-conv nhole-mctxt-reflcl-lang*
by (*auto simp flip*: $\mathcal{T}_G$*-equivalent-def*)

definition *GTT-to-RR2-root* :: (*'q*, *'f*) *gtt* $\Rightarrow$ (-, *'f option* $\times$ *'f option*) *ta* **where**
GTT-to-RR2-root $\mathcal{G}$ = *pair-automaton* (*fst* $\mathcal{G}$) (*snd* $\mathcal{G}$)

definition *GTT-to-RR2-root-reg* **where**
GTT-to-RR2-root-reg $\mathcal{G}$ = *Reg* (*map-both Some* |`| *fId-on* (*gtt-states* $\mathcal{G}$)) (*GTT-to-RR2-root* $\mathcal{G}$)

lemma *GTT-to-RR2-root*:
RR2-spec (*GTT-to-RR2-root-reg* $\mathcal{G}$) (*agtt-lang* $\mathcal{G}$)
proof −
{ fix *s* **assume** *s* $\in$ $\mathcal{L}$ (*GTT-to-RR2-root-reg* $\mathcal{G}$)
then obtain *q* **where** *q*: *q* |$\in$| *fin* (*GTT-to-RR2-root-reg* $\mathcal{G}$) *q* |$\in$| *ta-der* (*GTT-to-RR2-root* $\mathcal{G}$) (*term-of-gterm s*)
by (*auto simp*: $\mathcal{L}$*-def gta-lang-def GTT-to-RR2-root-reg-def gta-der-def*)
then obtain *q'* **where** [*simp*]: *q* = (*Some q'*, *Some q'*) **using** *q*(*1*) **by** (*auto simp*: *GTT-to-RR2-root-reg-def*)
have $\exists$ *t u*. *s* = *gpair t u* $\wedge$ *q* |$\in$| *ta-der* (*fst* $\mathcal{G}$) (*term-of-gterm t*) $\wedge$ *q* |$\in$| *ta-der* (*snd* $\mathcal{G}$) (*term-of-gterm u*)
using *fsubsetD*[*OF ta-der-mono' q*(*2*), *of pair-automaton* (*fst* $\mathcal{G}$) (*snd* $\mathcal{G}$)]
by (*auto simp*: *GTT-to-RR2-root-def dest*!: *from-ta-der-pair-automaton*(*4*))
} moreover

{ fix t u q **assume** q: q $|\in|$ *ta-der* (*fst* $\mathcal{G}$) (*term-of-gterm* t) q $|\in|$ *ta-der* (*snd* $\mathcal{G}$) (*term-of-gterm* u)
have *lift-fun* q $|\in|$ *map-both Some* $|`|$ *fId-on* ($\mathcal{Q}$ (*fst* $\mathcal{G}$) $|\cup|$ $\mathcal{Q}$ (*snd* $\mathcal{G}$))
using q[*THEN fsubsetD*[*OF ground-ta-der-states*[*OF ground-term-of-gterm*]]]
by (*auto simp*: *fimage-iff fBex-def*)
then have *gpair* t $u \in \mathcal{L}$ (*GTT-to-RR2-root-reg* $\mathcal{G}$) **using** q
using *fsubsetD*[*OF ta-der-mono to-ta-der-pair-automaton*(*3*)[*OF q*], *of GTT-to-RR2-root* $\mathcal{G}$]
by (*auto simp*: $\mathcal{L}$*-def GTT-to-RR2-root-def gta-lang-def image-def gtt-states-def gta-der-def GTT-to-RR2-root-reg-def*)
} ultimately show *?thesis* **by** (*auto simp*: *RR2-spec-def agtt-lang-def* $\mathcal{L}$*-def gta-der-def*)
qed

lemma *swap-GTT-to-RR2-root*:
gpair s $t \in \mathcal{L}$ (*GTT-to-RR2-root-reg* (*prod.swap* $\mathcal{G}$)) $\longleftrightarrow$
gpair t $s \in \mathcal{L}$ (*GTT-to-RR2-root-reg* $\mathcal{G}$)
by (*auto simp*: *GTT-to-RR2-root*[*unfolded RR2-spec-def*] *agtt-lang-def*)

lemma *funas-mctxt-map-vars-mctxt* [*simp*]:
funas-mctxt (*map-vars-mctxt f C*) = *funas-mctxt C*
by (*induct C*) *auto*

definition *GTT-to-RR2-reg* :: (*'f* × *nat*) *fset* ⇒ (*'q*, *'f*) *gtt* ⇒ (-, *'f option* × *'f option*) *reg* **where**
GTT-to-RR2-reg F G = *parallel-closure-reg* (*lift-sig-RR2* $|`|$ *F*) (*GTT-to-RR2-root-reg G*)

lemma *agtt-lang-syms*:
gtt-syms $\mathcal{G}$ $|\subseteq|$ $\mathcal{F}$ ⟹ *agtt-lang* $\mathcal{G}$ ⊆ {*t*. *funas-gterm t* ⊆ *fset* $\mathcal{F}$} × {*t*. *funas-gterm t* ⊆ *fset* $\mathcal{F}$}
by (*auto simp*: *agtt-lang-def gta-der-def funas-term-of-gterm-conv*)
(*metis ffunas-gterm.rep-eq fin-mono ta-der-gterm-sig*)+

lemma *gtt-lang-from-agtt-lang*:
gtt-lang $\mathcal{G}$ = *lift-root-step UNIV PAny EParallel* (*agtt-lang* $\mathcal{G}$)
unfolding *lift-root-step.simps agtt-lang-def*
by (*auto simp*: *lift-root-step.simps agtt-lang-def gmctxt-cl-gmctxtex-onp-conv*)

lemma *GTT-to-RR2*:
assumes *gtt-syms* $\mathcal{G}$ $|\subseteq|$ $\mathcal{F}$
shows *RR2-spec* (*GTT-to-RR2-reg* $\mathcal{F}$ $\mathcal{G}$) (*ggtt-lang* $\mathcal{F}$ $\mathcal{G}$)
proof −
have ∗: *snd* ` (*X* × *X*) = *X* **for** *X* **by** *auto*
show *?thesis* **unfolding** *gtt-lang-from-agtt-lang GTT-to-RR2-reg-def RR2-spec-def parallel-closure-automaton*[*OF GTT-to-RR2-root*, *of* $\mathcal{F}$ $\mathcal{G}$, *unfolded RR2-spec-def*]
proof (*intro arg-cong*[**where** *f* = λ*X*. {*gpair t u* |*t u*. (*t*,*u*) ∈ *X*}] *equalityI subrelI*, *goal-cases*)

case (*1 s t*) **then show** *?case*
using *subsetD*[*OF equalityD2*[*OF gtt-lang-from-agtt-lang*], *of* (*s*, *t*) $\mathcal{G}$]
by (*intro rev-image-eqI*[*of* (*term-of-gterm s*, *term-of-gterm t*)])
(*auto simp*: *funas-term-of-gterm-conv subsetD*[*OF lift-root-step-mono*]
dest: *subsetD*[*OF lift-root-step-sig*[*unfolded* $\mathcal{T}_G$*-equivalent-def*, *OF agtt-lang-syms*[*OF assms*]]])
next
case (*2 s t*)
from *image-mono*[*OF agtt-lang-syms*[*OF assms*], *of snd*, *unfolded* ∗]
have ∗: *snd* ‘ *agtt-lang* $\mathcal{G}$ ⊆ *gterms UNIV* **by** *auto*
show *?case* **using** *2*
by (*auto intro*!: *lift-root-step-sig-transfer*[*unfolded* $\mathcal{T}_G$*-equivalent-def*, *OF - ∗*, *of - - - fset* $\mathcal{F}$]
simp: *funas-gterm-gterm-of-term funas-term-of-gterm-conv*)
qed
qed

end
theory *FOL-Extra*
imports
Type-Instances-Impl
FOL−Fitting.FOL-Fitting
HOL−Library.FSet
begin

11 Additional support for FOL-Fitting

11.1 Iff

definition *Iff* **where**
Iff p q = *And* (*Impl p q*) (*Impl q p*)

lemma *eval-Iff*:
eval e f g (*Iff p q*) ⟷ (*eval e f g p* ⟷ *eval e f g q*)
by (*auto simp*: *Iff-def*)

11.2 Replacement of subformulas

datatype ('*a*, '*b*) *ctxt*
= *Hole*
| *And1* ('*a*, '*b*) *ctxt* ('*a*, '*b*) *form*
| *And2* ('*a*, '*b*) *form* ('*a*, '*b*) *ctxt*
| *Or1* ('*a*, '*b*) *ctxt* ('*a*, '*b*) *form*
| *Or2* ('*a*, '*b*) *form* ('*a*, '*b*) *ctxt*
| *Impl1* ('*a*, '*b*) *ctxt* ('*a*, '*b*) *form*
| *Impl2* ('*a*, '*b*) *form* ('*a*, '*b*) *ctxt*
| *Neg1* ('*a*, '*b*) *ctxt*
| *Forall1* ('*a*, '*b*) *ctxt*

| *Exists1* (*'a*, *'b*) *ctxt*

primrec *apply-ctxt* :: (*'a*, *'b*) *ctxt* ⇒ (*'a*, *'b*) *form* ⇒ (*'a*, *'b*) *form* **where**
apply-ctxt Hole p = *p*
| *apply-ctxt* (*And1 c v*) *p* = *And* (*apply-ctxt c p*) *v*
| *apply-ctxt* (*And2 u c*) *p* = *And u* (*apply-ctxt c p*)
| *apply-ctxt* (*Or1 c v*) *p* = *Or* (*apply-ctxt c p*) *v*
| *apply-ctxt* (*Or2 u c*) *p* = *Or u* (*apply-ctxt c p*)
| *apply-ctxt* (*Impl1 c v*) *p* = *Impl* (*apply-ctxt c p*) *v*
| *apply-ctxt* (*Impl2 u c*) *p* = *Impl u* (*apply-ctxt c p*)
| *apply-ctxt* (*Neg1 c*) *p* = *Neg* (*apply-ctxt c p*)
| *apply-ctxt* (*Forall1 c*) *p* = *Forall* (*apply-ctxt c p*)
| *apply-ctxt* (*Exists1 c*) *p* = *Exists* (*apply-ctxt c p*)

lemma *replace-subformula*:
assumes ⋀*e*. *eval e f g* (*Iff p q*)
shows *eval e f g* (*Iff* (*apply-ctxt c p*) (*apply-ctxt c q*))
by (*induct c arbitrary*: *e*) (*auto simp*: *assms*[*unfolded eval-Iff*] *Iff-def*)

11.3 Propositional identities

lemma *prop-ids*:
eval e f g (*Iff* (*And p q*) (*And q p*))
eval e f g (*Iff* (*Or p q*) (*Or q p*))
eval e f g (*Iff* (*Or p* (*Or q r*)) (*Or* (*Or p q*) *r*))
eval e f g (*Iff* (*And p* (*And q r*)) (*And* (*And p q*) *r*))
eval e f g (*Iff* (*Neg* (*Or p q*)) (*And* (*Neg p*) (*Neg q*)))
eval e f g (*Iff* (*Neg* (*And p q*)) (*Or* (*Neg p*) (*Neg q*)))

by (*auto simp*: *Iff-def*)

11.4 de Bruijn index manipulation for formulas; cf. *liftt*

primrec *liftti* :: *nat* ⇒ *'a term* ⇒ *'a term* **where**
liftti i (*Var j*) = (*if i* > *j then Var j else Var* (*Suc j*))
| *liftti i* (*App f ts*) = *App f* (*map* (*liftti i*) *ts*)

lemma *liftts-def'*:
liftts ts = *map liftt ts*
by (*induct ts*) *auto*

liftt is a special case of *liftti*

lemma *lifttti-0*:
liftti 0 t = *liftt t*
by (*induct t*) (*auto simp*: *liftts-def'*)

primrec *lifti* :: *nat* ⇒ (*'a*, *'b*) *form* ⇒ (*'a*, *'b*) *form* **where**
lifti i FF = *FF*
| *lifti i TT* = *TT*
| *lifti i* (*Pred b ts*) = *Pred b* (*map* (*liftti i*) *ts*)

| *lifti i* (*And p q*) = *And* (*lifti i p*) (*lifti i q*)
| *lifti i* (*Or p q*) = *Or* (*lifti i p*) (*lifti i q*)
| *lifti i* (*Impl p q*) = *Impl* (*lifti i p*) (*lifti i q*)
| *lifti i* (*Neg p*) = *Neg* (*lifti i p*)
| *lifti i* (*Forall p*) = *Forall* (*lifti* (*Suc i*) *p*)
| *lifti i* (*Exists p*) = *Exists* (*lifti* (*Suc i*) *p*)

abbreviation *lift* **where**
lift ≡ *lifti 0*

interaction of *lifti* and *eval*

lemma *evalts-def′*:
evalts e f ts = *map* (*evalt e f*) *ts*
by (*induct ts*) *auto*

lemma *evalt-liftti*:
evalt (*e*⟨*i*:*z*⟩) *f* (*liftti i t*) = *evalt e f t*
by (*induct t*) (*auto simp*: *evalts-def′ cong*: *map-cong*)

lemma *eval-lifti* [*simp*]:
eval (*e*⟨*i*:*z*⟩) *f g* (*lifti i p*) = *eval e f g p*
by (*induct p arbitrary*: *e i*) (*auto simp*: *evalt-liftti evalts-def′ comp-def*)

11.5 Quantifier Identities

lemma *quant-ids*:
eval e f g (*Iff* (*Neg* (*Exists p*)) (*Forall* (*Neg p*)))
eval e f g (*Iff* (*Neg* (*Forall p*)) (*Exists* (*Neg p*)))
eval e f g (*Iff* (*And p* (*Forall q*)) (*Forall* (*And* (*lift p*) *q*)))
eval e f g (*Iff* (*And p* (*Exists q*)) (*Exists* (*And* (*lift p*) *q*)))
eval e f g (*Iff* (*Or p* (*Forall q*)) (*Forall* (*Or* (*lift p*) *q*)))
eval e f g (*Iff* (*Or p* (*Exists q*)) (*Exists* (*Or* (*lift p*) *q*)))

by (*auto simp*: *Iff-def*)

11.6 Function symbols and predicates, with arities.

primrec *predas-form* :: (′*a*, ′*b*) *form* ⇒ (′*b* × *nat*) *set* **where**
predas-form FF = {}
| *predas-form TT* = {}
| *predas-form* (*Pred b ts*) = {(*b*, *length ts*)}
| *predas-form* (*And p q*) = *predas-form p* ∪ *predas-form q*
| *predas-form* (*Or p q*) = *predas-form p* ∪ *predas-form q*
| *predas-form* (*Impl p q*) = *predas-form p* ∪ *predas-form q*
| *predas-form* (*Neg p*) = *predas-form p*
| *predas-form* (*Forall p*) = *predas-form p*
| *predas-form* (*Exists p*) = *predas-form p*

primrec *funas-term* :: ′*a term* ⇒ (′*a* × *nat*) *set* **where**
funas-term (*Var x*) = {}

| *funas-term* (*App f ts*) = {(*f*, *length ts*)} ∪ ⋃(*set* (*map funas-term ts*))

primrec *terms-form* :: ('*a*, '*b*) *form* ⇒ '*a term set* **where**
  terms-form FF = {}
| *terms-form TT* = {}
| *terms-form* (*Pred b ts*) = *set ts*
| *terms-form* (*And p q*) = *terms-form p* ∪ *terms-form q*
| *terms-form* (*Or p q*) = *terms-form p* ∪ *terms-form q*
| *terms-form* (*Impl p q*) = *terms-form p* ∪ *terms-form q*
| *terms-form* (*Neg p*) = *terms-form p*
| *terms-form* (*Forall p*) = *terms-form p*
| *terms-form* (*Exists p*) = *terms-form p*

definition *funas-form* :: ('*a*, '*b*) *form* ⇒ ('*a* × *nat*) *set* **where**
  funas-form f ≡ ⋃(*funas-term* ‘ *terms-form f*)

11.7 Negation Normal Form

inductive *is-nnf* :: ('*a*, '*b*) *form* ⇒ *bool* **where**
  is-nnf TT
| *is-nnf FF*
| *is-nnf* (*Pred p ts*)
| *is-nnf* (*Neg* (*Pred p ts*))
| *is-nnf p* ⟹ *is-nnf q* ⟹ *is-nnf* (*And p q*)
| *is-nnf p* ⟹ *is-nnf q* ⟹ *is-nnf* (*Or p q*)
| *is-nnf p* ⟹ *is-nnf* (*Forall p*)
| *is-nnf p* ⟹ *is-nnf* (*Exists p*)

primrec *nnf'* :: *bool* ⇒ ('*a*, '*b*) *form* ⇒ ('*a*, '*b*) *form* **where**
  nnf' b TT = (*if b then TT else FF*)
| *nnf' b FF* = (*if b then FF else TT*)
| *nnf' b* (*Pred p ts*) = (*if b then id else Neg*) (*Pred p ts*)
| *nnf' b* (*And p q*) = (*if b then And else Or*) (*nnf' b p*) (*nnf' b q*)
| *nnf' b* (*Or p q*) = (*if b then Or else And*) (*nnf' b p*) (*nnf' b q*)
| *nnf' b* (*Impl p q*) = (*if b then Or else And*) (*nnf'* (¬ *b*) *p*) (*nnf' b q*)
| *nnf' b* (*Neg p*) = *nnf'* (¬ *b*) *p*
| *nnf' b* (*Forall p*) = (*if b then Forall else Exists*) (*nnf' b p*)
| *nnf' b* (*Exists p*) = (*if b then Exists else Forall*) (*nnf' b p*)

lemma *eval-nnf'*:
  eval e f g (*nnf' b p*) ⟷ (*eval e f g p* ⟷ *b*)
  by (*induct p arbitrary*: *e b*) *auto*

lemma *is-nnf-nnf'*:
  is-nnf (*nnf' b p*)
  by (*induct p arbitrary*: *b*) (*auto intro*: *is-nnf.intros*)

abbreviation *nnf* **where**
  nnf ≡ *nnf' True*

lemmas *nnf-simpls* [*simp*] = *eval-nnf'*[**where** *b* = *True*, *unfolded eq-True*] *is-nnf-nnf'*[**where** *b* = *True*]

11.8 Reasoning modulo ACI01

datatype ('*a*, '*b*) *form-aci*
= *TT-aci*
| *FF-aci*
| *Pred-aci bool* '*b* '*a term list*
| *And-aci* ('*a*, '*b*) *form-aci fset*
| *Or-aci* ('*a*, '*b*) *form-aci fset*
| *Forall-aci* ('*a*, '*b*) *form-aci*
| *Exists-aci* ('*a*, '*b*) *form-aci*

evaluation, see *eval*

primrec *eval-aci* :: ‹(*nat* ⇒ '*c*) ⇒ ('*a* ⇒ '*c list* ⇒ '*c*) ⇒
('*b* ⇒ '*c list* ⇒ *bool*) ⇒ ('*a*, '*b*) *form-aci* ⇒ *bool*› **where**
eval-aci e f g FF-aci ⟷ *False*
| *eval-aci e f g TT-aci* ⟷ *True*
| *eval-aci e f g* (*Pred-aci b a ts*) ⟷ (*g a* (*evalts e f ts*) ⟷ *b*)
| *eval-aci e f g* (*And-aci ps*) ⟷ *fBall* (*fimage* (*eval-aci e f g*) *ps*) *id*
| *eval-aci e f g* (*Or-aci ps*) ⟷ *fBex* (*fimage* (*eval-aci e f g*) *ps*) *id*
| *eval-aci e f g* (*Forall-aci p*) ⟷ (∀ *z*. *eval-aci* (*e*⟨*0*:*z*⟩) *f g p*)
| *eval-aci e f g* (*Exists-aci p*) ⟷ (∃ *z*. *eval-aci* (*e*⟨*0*:*z*⟩) *f g p*)

smart constructor: conjunction

fun *and-aci* **where**
and-aci FF-aci - = *FF-aci*
| *and-aci* - *FF-aci* = *FF-aci*
| *and-aci TT-aci q* = *q*
| *and-aci p TT-aci* = *p*
| *and-aci* (*And-aci ps*) (*And-aci qs*) = *And-aci* (*ps* |∪| *qs*)
| *and-aci* (*And-aci ps*) *q* = *And-aci* (*ps* |∪| {|*q*|})
| *and-aci p* (*And-aci qs*) = *And-aci* ({|*p*|} |∪| *qs*)
| *and-aci p q* = (*if p* = *q then p else And-aci* {|*p*,*q*|})

lemma *eval-and-aci* [*simp*]:
eval-aci e f g (*and-aci p q*) ⟷ *eval-aci e f g p* ∧ *eval-aci e f g q*
by (*cases* (*p*, *q*) *rule*: *and-aci.cases*) (*simp-all add*: *ball-Un*, *meson*+)

declare *and-aci.simps* [*simp del*]

smart constructor: disjunction

fun *or-aci* **where**
or-aci TT-aci - = *TT-aci*
| *or-aci* - *TT-aci* = *TT-aci*
| *or-aci FF-aci q* = *q*
| *or-aci p FF-aci* = *p*

```
| or-aci (Or-aci ps) (Or-aci qs) = Or-aci (ps |∪| qs)
| or-aci (Or-aci ps) q           = Or-aci (ps |∪| {|q|})
| or-aci p           (Or-aci qs) = Or-aci ({|p|} |∪| qs)
| or-aci p           q           = (if p = q then p else Or-aci {|p,q|})
```

lemma *eval-or-aci* [*simp*]:
eval-aci e f g (*or-aci p q*) ⟷ *eval-aci e f g p* ∨ *eval-aci e f g q*
by (*cases* (*p*, *q*) *rule*: *or-aci.cases*) (*simp-all add*: *bex-Un*, *meson+*)

declare *or-aci.simps* [*simp del*]

convert negation normal form to ACIU01 normal form

```
fun nnf-to-aci :: ('a, 'b) form ⇒ ('a, 'b) form-aci where
  nnf-to-aci FF                = FF-aci
| nnf-to-aci TT                = TT-aci
| nnf-to-aci (Pred b ts)       = Pred-aci True b ts
| nnf-to-aci (Neg (Pred b ts)) = Pred-aci False b ts
| nnf-to-aci (And p q)         = and-aci (nnf-to-aci p) (nnf-to-aci q)
| nnf-to-aci (Or p q)          = or-aci (nnf-to-aci p) (nnf-to-aci q)
| nnf-to-aci (Forall p)        = Forall-aci (nnf-to-aci p)
| nnf-to-aci (Exists p)        = Exists-aci (nnf-to-aci p)
| nnf-to-aci -                 = undefined
```

lemma *eval-nnf-to-aci*:
is-nnf p ⟹ *eval-aci e f g* (*nnf-to-aci p*) ⟷ *eval e f g p*
by (*induct p arbitrary*: *e rule*: *is-nnf.induct*) *simp-all*

11.9 A (mostly) Propositional Equivalence Check

We reason modulo ∀ = ¬∃¬, de Morgan, double negation, and ACUI01 of ∨ and ∧, by converting to negation normal form, and then collapsing conjunctions and disjunctions taking units, absorption, commutativity, associativity, and idempotence into account. We only need soundness for a certifier.

lemma *check-equivalence-by-nnf-aci*:
nnf-to-aci (*nnf p*) = *nnf-to-aci* (*nnf q*) ⟹ *eval e f g p* ⟷ *eval e f g q*
by (*metis eval-nnf-to-aci is-nnf-nnf' eval-nnf'*)

11.10 Reasoning modulo ACI01

```
datatype ('a, 'b) form-list-aci
  = TT-aci
  | FF-aci
  | Pred-aci bool 'b 'a term list
  | And-aci ('a, 'b) form-list-aci list
  | Or-aci ('a, 'b) form-list-aci list
  | Forall-aci ('a, 'b) form-list-aci
  | Exists-aci ('a, 'b) form-list-aci
```

evaluation, see *eval*

fun *eval-list-aci* :: ‹(*nat* ⇒ *'c*) ⇒ (*'a* ⇒ *'c list* ⇒ *'c*) ⇒
(*'b* ⇒ *'c list* ⇒ *bool*) ⇒ (*'a*, *'b*) *form-list-aci* ⇒ *bool*› **where**
eval-list-aci e f g FF-aci ⟷ *False*
| *eval-list-aci e f g TT-aci* ⟷ *True*
| *eval-list-aci e f g* (*Pred-aci b a ts*) ⟷ (*g a* (*evalts e f ts*) ⟷ *b*)
| *eval-list-aci e f g* (*And-aci ps*) ⟷ *list-all* (λ *fm*. *eval-list-aci e f g fm*) *ps*
| *eval-list-aci e f g* (*Or-aci ps*) ⟷ *list-ex* (λ *fm*. *eval-list-aci e f g fm*) *ps*
| *eval-list-aci e f g* (*Forall-aci p*) ⟷ (∀ *z*. *eval-list-aci* (*e*⟨*0*:*z*⟩) *f g p*)
| *eval-list-aci e f g* (*Exists-aci p*) ⟷ (∃ *z*. *eval-list-aci* (*e*⟨*0*:*z*⟩) *f g p*)

smart constructor: conjunction

fun *and-list-aci* **where**
and-list-aci FF-aci - = *FF-aci*
| *and-list-aci* - *FF-aci* = *FF-aci*
| *and-list-aci TT-aci q* = *q*
| *and-list-aci p TT-aci* = *p*
| *and-list-aci* (*And-aci ps*) (*And-aci qs*) = *And-aci* (*remdups* (*ps* @ *qs*))
| *and-list-aci* (*And-aci ps*) *q* = *And-aci* (*List.insert q ps*)
| *and-list-aci p* (*And-aci qs*) = *And-aci* (*List.insert p qs*)
| *and-list-aci p q* = (*if p* = *q then p else And-aci* [*p*,*q*])

lemma *eval-and-list-aci* [*simp*]:
eval-list-aci e f g (*and-list-aci p q*) ⟷ *eval-list-aci e f g p* ∧ *eval-list-aci e f g q*
apply (*cases* (*p*, *q*) *rule*: *and-list-aci.cases*)
apply (*simp-all add*: *list.pred-set list-ex-iff*)
apply *blast+*
done

declare *and-list-aci.simps* [*simp del*]

smart constructor: disjunction

fun *or-list-aci* **where**
or-list-aci TT-aci - = *TT-aci*
| *or-list-aci* - *TT-aci* = *TT-aci*
| *or-list-aci FF-aci q* = *q*
| *or-list-aci p FF-aci* = *p*
| *or-list-aci* (*Or-aci ps*) (*Or-aci qs*) = *Or-aci* (*remdups* (*ps* @ *qs*))
| *or-list-aci* (*Or-aci ps*) *q* = *Or-aci* (*List.insert q ps*)
| *or-list-aci p* (*Or-aci qs*) = *Or-aci* (*List.insert p qs*)
| *or-list-aci p q* = (*if p* = *q then p else Or-aci* [*p*,*q*])

lemma *eval-or-list-aci* [*simp*]:
eval-list-aci e f g (*or-list-aci p q*) ⟷ *eval-list-aci e f g p* ∨ *eval-list-aci e f g q*
by (*cases* (*p*, *q*) *rule*: *or-list-aci.cases*) (*simp-all add*: *list.pred-set list-ex-iff*, *blast+*)

declare *or-list-aci.simps* [*simp del*]

convert negation normal form to ACIU01 normal form

fun *nnf-to-list-aci* :: (*'a*, *'b*) *form* ⇒ (*'a*, *'b*) *form-list-aci* **where**

```
  nnf-to-list-aci FF              = FF-aci
| nnf-to-list-aci TT              = TT-aci
| nnf-to-list-aci (Pred b ts)     = Pred-aci True b ts
| nnf-to-list-aci (Neg (Pred b ts)) = Pred-aci False b ts
| nnf-to-list-aci (And p q)       = and-list-aci (nnf-to-list-aci p) (nnf-to-list-aci q)
| nnf-to-list-aci (Or p q)        = or-list-aci (nnf-to-list-aci p) (nnf-to-list-aci q)
| nnf-to-list-aci (Forall p)      = Forall-aci (nnf-to-list-aci p)
| nnf-to-list-aci (Exists p)      = Exists-aci (nnf-to-list-aci p)
| nnf-to-list-aci -               = undefined
```

lemma *eval-nnf-to-list-aci*:
is-nnf p $\Longrightarrow$ *eval-list-aci e f g* (*nnf-to-list-aci p*) $\longleftrightarrow$ *eval e f g p*
by (*induct p arbitrary*: *e rule*: *is-nnf.induct*) *simp-all*

11.11 A (mostly) Propositional Equivalence Check

We reason modulo $\forall = \neg\exists\neg$, de Morgan, double negation, and ACUI01 of $\vee$ and $\wedge$, by converting to negation normal form, and then collapsing conjunctions and disjunctions taking units, absorption, commutativity, associativity, and idempotence into account. We only need soundness for a certifier.

derive *linorder term*
derive *compare term*
derive *linorder form-list-aci*
derive *compare form-list-aci*

fun *ord-form-list-aci* **where**
ord-form-list-aci TT-aci = TT-aci
| *ord-form-list-aci FF-aci = FF-aci*
| *ord-form-list-aci* (*Pred-aci bool b ts*) = *Pred-aci bool b ts*
| *ord-form-list-aci* (*And-aci fm*) = (*And-aci* (*sort* (*map ord-form-list-aci fm*)))
| *ord-form-list-aci* (*Or-aci fm*) = (*Or-aci* (*sort* (*map ord-form-list-aci fm*)))
| *ord-form-list-aci* (*Forall-aci fm*) = (*Forall-aci* (*ord-form-list-aci fm*))
| *ord-form-list-aci* (*Exists-aci fm*) = *Exists-aci* (*ord-form-list-aci fm*)

lemma *and-list-aci-simps*:
and-list-aci TT-aci q = q
and-list-aci q FF-aci = FF-aci
by (*cases q*, *auto simp add*: *and-list-aci.simps*)+

lemma *ord-form-list-idemp*:
ord-form-list-aci (*ord-form-list-aci q*) = *ord-form-list-aci q*
apply (*induct q*) **apply** (*auto simp*: *list.set-map*)
apply (*smt* (*verit*) *imageE list.set-map map-idI set-sort sorted-sort-id sorted-sort-key*)+
done

lemma *eval-lsit-aci-ord-form-list-aci*:
eval-list-aci e f g (*ord-form-list-aci p*) $\longleftrightarrow$ *eval-list-aci e f g p*
by (*induct p arbitrary*: *e*) (*auto simp*: *list.pred-set list-ex-iff*)

lemma *check-equivalence-by-nnf-sortedlist-aci*:
ord-form-list-aci (*nnf-to-list-aci* (*nnf p*)) = *ord-form-list-aci* (*nnf-to-list-aci* (*nnf q*)) ⟹ *eval e f g p* ⟷ *eval e f g q*
by (*metis eval-nnf-to-list-aci eval-lsit-aci-ord-form-list-aci is-nnf-nnf' eval-nnf'*)

hide-type (**open**) *term*
hide-const (**open**) *Var*
hide-type (**open**) *ctxt*

end
theory *FOR-Semantics*
imports *FOR-Certificate*
Lift-Root-Step
FOL−Fitting.FOL-Fitting
begin

12 Semantics of Relations

definition *is-to-trs* :: ('*f*, '*v*) *trs list* ⇒ *ftrs list* ⇒ ('*f*, '*v*) *trs* **where**
is-to-trs Rs is = ⋃(*set* (*map* (*case-ftrs* ((!) *Rs*) ((‘) *prod.swap* ∘ (!) *Rs*)) *is*))

primrec *eval-gtt-rel* :: ('*f* × *nat*) *set* ⇒ ('*f*, '*v*) *trs list* ⇒ *ftrs gtt-rel* ⇒ '*f gterm rel* **where**
eval-gtt-rel $\mathcal{F}$ *Rs* (*ARoot is*) = *Restr* (*grrstep* (*is-to-trs Rs is*)) ($\mathcal{T}_G$ $\mathcal{F}$)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*GInv g*) = *prod.swap* ‘ (*eval-gtt-rel* $\mathcal{F}$ *Rs g*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*AUnion g1 g2*) = (*eval-gtt-rel* $\mathcal{F}$ *Rs g1*) ∪ (*eval-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*ATrancl g*) = (*eval-gtt-rel* $\mathcal{F}$ *Rs g*)$^+$
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*AComp g1 g2*) = (*eval-gtt-rel* $\mathcal{F}$ *Rs g1*) *O* (*eval-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*GTrancl g*) = *gtrancl-rel* $\mathcal{F}$ (*eval-gtt-rel* $\mathcal{F}$ *Rs g*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*GComp g1 g2*) = *gcomp-rel* $\mathcal{F}$ (*eval-gtt-rel* $\mathcal{F}$ *Rs g1*) (*eval-gtt-rel* $\mathcal{F}$ *Rs g2*)

primrec *eval-rr1-rel* :: ('*f* × *nat*) *set* ⇒ ('*f*, '*v*) *trs list* ⇒ *ftrs rr1-rel* ⇒ '*f gterm set*
and *eval-rr2-rel* :: ('*f* × *nat*) *set* ⇒ ('*f*, '*v*) *trs list* ⇒ *ftrs rr2-rel* ⇒ '*f gterm rel* **where**
eval-rr1-rel $\mathcal{F}$ *Rs R1Terms* = ($\mathcal{T}_G$ $\mathcal{F}$)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Union R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) ∪ (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Inter R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) ∩ (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Diff R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) − (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Proj n R*) = (*case n of 0* ⇒ *fst* ‘ (*eval-rr2-rel* $\mathcal{F}$ *Rs R*)
| - ⇒ *snd* ‘ (*eval-rr2-rel* $\mathcal{F}$ *Rs R*))
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1NF is*) = *NF* (*Restr* (*grstep* (*is-to-trs Rs is*)) ($\mathcal{T}_G$ $\mathcal{F}$)) ∩ ($\mathcal{T}_G$ $\mathcal{F}$)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Inf R*) = {*s*. *infinite* (*eval-rr2-rel* $\mathcal{F}$ *Rs R* ‘‘ {*s*})}
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2GTT-Rel A W X*) = *lift-root-step* $\mathcal{F}$ *W X* (*eval-gtt-rel* $\mathcal{F}$

Rs A)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Inv R*) = *prod.swap* ‘ (*eval-rr2-rel* $\mathcal{F}$ *Rs R*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Union R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) ∪ (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Inter R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) ∩ (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Diff R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) − (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Comp R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) *O* (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Diag R*) = *Id-on* (*eval-rr1-rel* $\mathcal{F}$ *Rs R*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Prod R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) × (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)

12.1 Semantics of Formulas

fun *eval-formula* :: (*'f* × *nat*) *set* ⇒ (*'f*, *'v*) *trs list* ⇒ (*nat* ⇒ *'f gterm*) ⇒ *ftrs formula* ⇒ *bool* **where**
eval-formula $\mathcal{F}$ *Rs* α (*FRR1 r1 x*) ⟷ α *x* ∈ *eval-rr1-rel* $\mathcal{F}$ *Rs r1*
| *eval-formula* $\mathcal{F}$ *Rs* α (*FRR2 r2 x y*) ⟷ (α *x*, α *y*) ∈ *eval-rr2-rel* $\mathcal{F}$ *Rs r2*
| *eval-formula* $\mathcal{F}$ *Rs* α (*FAnd fs*) ⟷ (∀ *f* ∈ *set fs*. *eval-formula* $\mathcal{F}$ *Rs* α *f*)
| *eval-formula* $\mathcal{F}$ *Rs* α (*FOr fs*) ⟷ (∃ *f* ∈ *set fs*. *eval-formula* $\mathcal{F}$ *Rs* α *f*)
| *eval-formula* $\mathcal{F}$ *Rs* α (*FNot f*) ⟷ ¬ *eval-formula* $\mathcal{F}$ *Rs* α *f*
| *eval-formula* $\mathcal{F}$ *Rs* α (*FExists f*) ⟷ (∃ *z* ∈ $\mathcal{T}_G$ $\mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* (α⟨0 : *z*⟩) *f*)
| *eval-formula* $\mathcal{F}$ *Rs* α (*FForall f*) ⟷ (∀ *z* ∈ $\mathcal{T}_G$ $\mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* (α⟨0 : *z*⟩) *f*)

fun *formula-arity* :: *'trs formula* ⇒ *nat* **where**
formula-arity (*FRR1 r1 x*) = *Suc x*
| *formula-arity* (*FRR2 r2 x y*) = *max* (*Suc x*) (*Suc y*)
| *formula-arity* (*FAnd fs*) = *max-list* (*map formula-arity fs*)
| *formula-arity* (*FOr fs*) = *max-list* (*map formula-arity fs*)
| *formula-arity* (*FNot f*) = *formula-arity f*
| *formula-arity* (*FExists f*) = *formula-arity f* − *1*
| *formula-arity* (*FForall f*) = *formula-arity f* − *1*

lemma *R1NF-reps*:
assumes *funas-trs R* ⊆ $\mathcal{F}$ ∀ *t*. (*term-of-gterm s*, *term-of-gterm t*) ∈ *rstep R* ⟶ ¬*funas-gterm t* ⊆ $\mathcal{F}$
and *funas-gterm s* ⊆ $\mathcal{F}$ (*l*, *r*) ∈ *R term-of-gterm s* = *C*⟨*l* · (σ :: *'b* ⇒ (*'a*, *'b*) *Term.term*)⟩
shows *False*
proof −
obtain *c* **where** *w*: *funas-term* (*c* :: (*'a*, *'b*) *Term.term*) ⊆ $\mathcal{F}$ *ground c*
using *assms*(*3*) *funas-term-of-gterm-conv ground-term-of-gterm* **by** *blast*
define τ **where** τ *x* = (*if x* ∈ *vars-term l then* σ *x else c*) **for** *x*
from *assms*(*4*−) **have** *terms*: *term-of-gterm s* = *C*⟨*l* · τ⟩ (*C*⟨*l* · τ⟩, *C*⟨*r* · τ⟩) ∈ *rstep R*
using τ*-def* **by** *auto* (*metis term-subst-eq*)

from *this*(*1*) **have** [*simp*]: *funas-gterm s* = *funas-term* $C\langle l \cdot \tau \rangle$ **by** (*metis funas-term-of-gterm-conv*)
from *w assms*(*1*, *3*, *4*) **have** [*simp*]: *funas-term* $C\langle r \cdot \tau \rangle \subseteq \mathcal{F}$ **using** τ-*def*
by (*auto simp*: *funas-trs-def funas-term-subst*)
moreover have *ground* $C\langle r \cdot \tau \rangle$ **using** *terms*(*1*) *w* τ-*def*
by (*auto intro*!: *ground-substI*) (*metis term-of-gterm-ctxt-subst-apply-ground*)
ultimately show *?thesis* **using** *assms*(*2*) *terms*(*2*)
by (*metis funas-term-of-gterm-conv ground-term-to-gtermD terms*(*1*))
qed

The central property we are interested in is satisfiability

definition *formula-satisfiable* **where**
formula-satisfiable $\mathcal{F}$ *Rs f* $\longleftrightarrow$ ($\exists \alpha$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *f*)

12.2 Validation

12.3 Defining properties of *gcomp-rel* and *gtrancl-rel*

lemma *gcomp-rel-sig*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$ **and** $S \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *gcomp-rel* $\mathcal{F}$ *R S* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
using *assms subsetD*[*OF signature-pres-funas-cl*(*2*)[*OF assms*(*1*)]]
by (*auto simp*: *gcomp-rel-def lift-root-step.simps gmctxt-cl-gmctxtex-onp-conv*) (*metis refl-onD2 relf-on-gmctxtcl-funas*)

lemma *gtrancl-rel-sig*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *gtrancl-rel* $\mathcal{F}$ *R* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
using *gtrancl-rel-sound*[*OF assms*] **by** *simp*

lemma *gtrancl-rel*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *lift-root-step* $\mathcal{F}$ *PAny EStrictParallel* (*gtrancl-rel* $\mathcal{F}$ *R*) = (*lift-root-step* $\mathcal{F}$ *PAny ESingle R*)$^+$
unfolding *lift-root-step.simps* **using** *gmctxtcl-funas-strict-gtrancl-rel*[*OF assms*]
.

lemma *gtrancl-rel*$'$:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *lift-root-step* $\mathcal{F}$ *PAny EParallel* (*gtrancl-rel* $\mathcal{F}$ *R*) = *Restr* ((*lift-root-step* $\mathcal{F}$ *PAny ESingle R*)*) ($\mathcal{T}_G\ \mathcal{F}$)
using *assms gtrancl-rel*[*OF assms*]
by (*auto simp*: *lift-root-step-Parallel-conv*
simp flip: *reflcl-trancl dest*: *Restr-simps*(*5*)[*OF lift-root-step-sig*, *THEN subsetD*])

GTT relation semantics respects the signature

lemma *eval-gtt-rel-sig*:
eval-gtt-rel $\mathcal{F}$ *Rs g* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
proof −

show *?thesis* **by** (*induct g*) (*auto 0 3 simp*: *gtrancl-rel-sig gcomp-rel-sig dest*: *tranclD tranclD2*)
qed

RR1 and RR2 relation semantics respect the signature

lemma *eval-rr12-rel-sig*:
eval-rr1-rel $\mathcal{F}$ *Rs r1* $\subseteq \mathcal{T}_G\ \mathcal{F}$
eval-rr2-rel $\mathcal{F}$ *Rs r2* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
proof (*induct r1* **and** *r2*)
case (*R1Inf r2*) **then show** *?case* **by** (*auto dest*!: *infinite-imp-nonempty*)
next
case (*R1Proj i r2*) **then show** *?case* **by** (*fastforce split*: *nat.splits*)
next
case (*R2GTT-Rel g W X*) **then show** *?case* **by** (*simp add*: *lift-root-step-sig eval-gtt-rel-sig*)
qed *auto*

12.4 Correctness of derived constructions

lemma *R1Fin*:
eval-rr1-rel $\mathcal{F}$ *Rs* (*R1Fin r*) = {$t \in \mathcal{T}_G\ \mathcal{F}$. *finite* {*s*. (*t*, *s*) $\in$ *eval-rr2-rel* $\mathcal{F}$ *Rs r*}}
by (*auto simp*: *R1Fin-def Image-def*)

lemma *R2Eq*:
eval-rr2-rel $\mathcal{F}$ *Rs R2Eq* = *Id-on* ($\mathcal{T}_G\ \mathcal{F}$)
by (*auto simp*: $\mathcal{T}_G$*-funas-gterm-conv R2Eq-def*)

lemma *R2Reflc*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2Reflc r*) = *eval-rr2-rel* $\mathcal{F}$ *Rs r* $\cup$ *Id-on* ($\mathcal{T}_G\ \mathcal{F}$)
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2Reflc r*) = *Restr* ((*eval-rr2-rel* $\mathcal{F}$ *Rs r*)$^=$) ($\mathcal{T}_G\ \mathcal{F}$)
using *eval-rr12-rel-sig*(*2*)[*of* $\mathcal{F}$ *Rs R2Reflc r*]
by (*auto simp*: *R2Reflc-def R2Eq*)

lemma *R2Step*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2Step ts*) = *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G\ \mathcal{F}$)
by (*auto simp*: *lift-root-step.simps R2Step-def grstep-lift-root-step grrstep-subst-cl-conv grstepD-grstep-conv grstepD-def*)

lemma *R2StepEq*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2StepEq ts*) = *Restr* ((*grstep* (*is-to-trs Rs ts*))$^=$) ($\mathcal{T}_G\ \mathcal{F}$)
by (*auto simp*: *R2StepEq-def R2Step R2Reflc*(*2*))

lemma *R2Steps*:
fixes $\mathcal{F}$ *Rs ts* **defines** *R* $\equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G\ \mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Steps ts*) = R^+
by (*simp add*: *R2Steps-def GSteps-def R-def gtrancl-rel grstep-lift-root-step*)
(*metis FOR-Semantics.gtrancl-rel Sigma-cong grstep-lift-root-step inf.cobounded2 lift-root-Any-EStrict-eq*)

lemma *R2StepsEq*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2StepsEq ts*) = *Restr* (R^*) ($\mathcal{T}_G$ $\mathcal{F}$)
using *R2Steps*[*of* $\mathcal{F}$ *Rs ts*]
by (*simp add*: *R2StepsEq-def R2Steps-def lift-root-step-Parallel-conv Int-Un-distrib2 R-def Restr-simps flip*: *reflcl-trancl*)

lemma *R2StepsNF*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2StepsNF ts*) = *Restr* ($R^* \cap$ *UNIV* $\times$ *NF R*) ($\mathcal{T}_G$ $\mathcal{F}$)
using *R2StepsEq*[*of* $\mathcal{F}$ *Rs ts*]
by (*auto simp*: *R2StepsNF-def R2StepsEq-def R-def*)

lemma *R2ParStep*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2ParStep ts*) = *Restr* (*gpar-rstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
by (*simp add*: *R2ParStep-def gar-rstep-lift-root-step*)

lemma *R2RootStep*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2RootStep ts*) = *Restr* (*grrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
by (*simp add*: *R2RootStep-def lift-root-step.simps*)

lemma *R2RootStepEq*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2RootStepEq ts*) = *Restr* ((*grrstep* (*is-to-trs Rs ts*))$^=$) ($\mathcal{T}_G$ $\mathcal{F}$)
by (*auto simp*: *R2RootStepEq-def R2RootStep R2Reflc*(*2*))

lemma *R2RootSteps*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2RootSteps ts*) = R^+
by (*simp add*: *R2RootSteps-def R-def lift-root-step.simps*)

lemma *R2RootStepsEq*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2RootStepsEq ts*) = *Restr* (R^*) ($\mathcal{T}_G$ $\mathcal{F}$)
by (*auto simp*: *R2RootStepsEq-def R2Reflc-def R2RootSteps R-def R2Eq-def Int-Un-distrib2 Restr-simps simp flip*: *reflcl-trancl*)

lemma *R2NonRootStep*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2NonRootStep ts*) = *Restr* (*gnrrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
by (*simp add*: *R2NonRootStep-def grrstep-lift-root-gnrrstep*)

lemma *R2NonRootStepEq*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2NonRootStepEq ts*) = *Restr* ((*gnrrstep* (*is-to-trs Rs ts*))$^=$) ($\mathcal{T}_G$ $\mathcal{F}$)
by (*auto simp*: *R2NonRootStepEq-def R2Reflc-def R2Eq-def R2NonRootStep Int-Un-distrib2*)

lemma *R2NonRootSteps*:

fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*gnrrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2NonRootSteps ts*) = R^+
apply (*simp add*: *lift-root-step.simps gnrrstepD-gnrrstep-conv gnrrstepD-def grrstep-subst-cl-conv R2NonRootSteps-def R-def GSteps-def lift-root-step-Parallel-conv*)
apply (*intro gmctxtex-funas-nroot-strict-gtrancl-rel*)
by *simp*

lemma *R2NonRootStepsEq*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*gnrrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2NonRootStepsEq ts*) = *Restr* (R^*) ($\mathcal{T}_G$ $\mathcal{F}$)
using *R2NonRootSteps*[*of* $\mathcal{F}$ *Rs ts*]
by (*simp add*: *R2NonRootSteps-def R2NonRootStepsEq-def lift-root-step-Parallel-conv R-def Int-Un-distrib2 Restr-simps flip*: *reflcl-trancl*)

lemma *converse-to-prod-swap*:
R^{-1} = *prod.swap* ‘ *R*
by *auto*

lemma *R2Meet*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Meet ts*) = *Restr* ((R^{-1})* *O* R^*) ($\mathcal{T}_G$ $\mathcal{F}$)
apply (*simp add*: *R2Meet-def R-def GSteps-def converse-to-prod-swap gcomp-rel*[*folded lift-root-step.simps*] *gtrancl-rel′ swap-lift-root-step grstep-lift-root-step*)
apply (*simp add*: *Restr-simps converse-Int converse-Un converse-Times Int-Un-distrib2 flip*: *reflcl-trancl trancl-converse converse-to-prod-swap*)
done

lemma *R2Join*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Join ts*) = *Restr* (R^* *O* (R^{-1})*) ($\mathcal{T}_G$ $\mathcal{F}$)
apply (*simp add*: *R2Join-def R-def GSteps-def converse-to-prod-swap gcomp-rel*[*folded lift-root-step.simps*] *gtrancl-rel′ swap-lift-root-step grstep-lift-root-step*)
apply (*simp add*: *Restr-simps converse-to-prod-swap*[*symmetric*] *converse-Int converse-Un converse-Times Int-Un-distrib2 flip*: *reflcl-trancl trancl-converse*)
done

end
theory *FOR-Check*
imports
FOR-Semantics
FOL-Extra
GTT-RRn
First-Order-Terms.Option-Monad
LV-to-GTT
NF
Regular-Tree-Relations.GTT-Transitive-Closure
Regular-Tree-Relations.AGTT
Regular-Tree-Relations.RR2-Infinite-Q-infinity
Regular-Tree-Relations.RRn-Automata

begin

13 Check inference steps

type-synonym $('f, 'v)$ *fin-trs* $= ('f, 'v)$ *rule fset*

lemma *tl-drop-conv*:
tl xs = *drop 1 xs*
by (*induct xs*) *auto*

definition *rrn-drop-fst* **where**
rrn-drop-fst $\mathcal{A}$ = *relabel-reg* (*trim-reg* (*collapse-automaton-reg* (*fmap-funs-reg* (*drop-none-rule 1*) (*trim-reg* $\mathcal{A}$))))

lemma *rrn-drop-fst-lang*:
assumes *RRn-spec n A T* 1 < *n*
shows *RRn-spec* (*n* − *1*) (*rrn-drop-fst A*) (*drop 1* ‘ *T*)
using *drop-automaton-reg*[*OF - assms*(*2*), *of trim-reg A T*] *assms*(*1*)
unfolding *rrn-drop-fst-def*
by (*auto simp*: *trim-ta-reach*)

definition *liftO1* :: $('a \Rightarrow 'b) \Rightarrow 'a$ *option* $\Rightarrow 'b$ *option* **where**
liftO1 = *map-option*

definition *liftO2* :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a$ *option* $\Rightarrow 'b$ *option* $\Rightarrow 'c$ *option* **where**
liftO2 f a b = *case-option None* ($\lambda a'$. *liftO1* (*f a'*) *b*) *a*

lemma *liftO1-Some* [*simp*]:
liftO1 f x = *Some y* $\longleftrightarrow$ ($\exists\, x'$. *x* = *Some x'*) ∧ *y* = *f* (*the x*)
by (*cases x*) (*auto simp*: *liftO1-def*)

lemma *liftO2-Some* [*simp*]:
liftO2 f x y = *Some z* $\longleftrightarrow$ ($\exists\, x'\ y'$. *x* = *Some x'* ∧ *y* = *Some y'*) ∧ *z* = *f* (*the x*) (*the y*)
by (*cases x*; *cases y*) (*auto simp*: *liftO2-def*)

13.1 Computing TRSs

lemma *is-to-trs-props*:
assumes ∀ *R* ∈ *set Rs*. *finite R* ∧ *lv-trs R* ∧ *funas-trs R* ⊆ $\mathcal{F}$ ∀ *i* ∈ *set is*. *case-ftrs id id i* < *length Rs*
shows *funas-trs* (*is-to-trs Rs is*) ⊆ $\mathcal{F}$ *lv-trs* (*is-to-trs Rs is*) *finite* (*is-to-trs Rs is*)
proof (*goal-cases* $\mathcal{F}$ *lv fin*)
case $\mathcal{F}$ **show** *?case* **using** *assms nth-mem*
apply (*auto simp*: *is-to-trs-def funas-trs-def case-prod-beta split*: *ftrs.splits*)
apply *fastforce*
apply (*metis* (*no-types*, *lifting*) *assms*(*1*) *in-mono rhs-wf*)

apply (*metis* (*no-types*, *lifting*) *assms*(*1*) *in-mono rhs-wf*)
by (*smt* (*verit*) *UN-subset-iff fst-conv in-mono le-sup-iff*)
qed (*insert assms*, (*fastforce simp*: *is-to-trs-def funas-trs-def lv-trs-def split*: *ftrs.splits*)+)

definition *is-to-fin-trs* :: ('*f*, '*v*) *fin-trs list* ⇒ *ftrs list* ⇒ ('*f*, '*v*) *fin-trs* **where**
is-to-fin-trs Rs is = |⋃| (*fset-of-list* (*map* (*case-ftrs* ((!) *Rs*) ((|¦|) *prod.swap* ∘ (!) *Rs*)) *is*))

lemma *is-to-fin-trs-conv*:
assumes ∀ *i* ∈ *set is*. *case-ftrs id id i* < *length Rs*
shows *is-to-trs* (*map fset Rs*) *is* = *fset* (*is-to-fin-trs Rs is*)
using *assms* **unfolding** *is-to-trs-def is-to-fin-trs-def*
by (*auto simp*: *ffUnion.rep-eq fset-of-list.rep-eq split*: *ftrs.splits*)

definition *is-to-trs'* :: ('*f*, '*v*) *fin-trs list* ⇒ *ftrs list* ⇒ ('*f*, '*v*) *fin-trs option* **where**
is-to-trs' Rs is = *do* {
guard (∀ *i* ∈ *set is*. *case-ftrs id id i* < *length Rs*);
Some (*is-to-fin-trs Rs is*)
}

lemma *is-to-trs-conv*:
is-to-trs' Rs is = *Some S* ⟹ *is-to-trs* (*map fset Rs*) *is* = *fset S*
using *is-to-fin-trs-conv* **unfolding** *is-to-trs'-def*
by (*auto simp add*: *guard-simps split*: *bind-splits*)

lemma *is-to-trs'-props*:
assumes ∀ *R* ∈ *set Rs*. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| *F* **and** *is-to-trs' Rs is* = *Some S*
shows *ffunas-trs S* |⊆| *F* *lv-trs* (*fset S*)
proof −
from *assms*(*2*) **have** *well*: ∀ *i* ∈ *set is*. *case-ftrs id id i* < *length Rs is-to-fin-trs Rs is* = *S*
unfolding *is-to-trs'-def*
by (*auto simp add*: *guard-simps split*: *bind-splits*)
have ∀ *R* ∈ *set Rs*. *finite* (*fset R*) ∧ *lv-trs* (*fset R*) ∧ *funas-trs* (*fset R*) ⊆ (*fset F*)
using *assms*(*1*) **by** (*auto simp*: *ffunas-trs.rep-eq less-eq-fset.rep-eq*)
from *is-to-trs-props*[*of map fset Rs fset F is*] *this well*(*1*)
have *lv-trs* (*is-to-trs* (*map fset Rs*) *is*) *funas-trs* (*is-to-trs* (*map fset Rs*) *is*) ⊆ *fset F*
by *auto*
then show *lv-trs* (*fset S*) *ffunas-trs S* |⊆| *F*
using *is-to-fin-trs-conv*[*OF well*(*1*)] **unfolding** *well*(*2*)
by (*auto simp*: *ffunas-trs.rep-eq less-eq-fset.rep-eq*)
qed

13.2 Computing GTTs

fun *gtt-of-gtt-rel* :: (*'f* × *nat*) *fset* ⇒ (*'f* :: *linorder*, *'v*) *fin-trs list* ⇒ *ftrs gtt-rel* ⇒ (*nat*, *'f*) *gtt option* **where**
gtt-of-gtt-rel $\mathcal{F}$ *Rs* (*ARoot is*) = *liftO1* (λ*R*. *relabel-gtt* (*agtt-grrstep R* $\mathcal{F}$)) (*is-to-trs' Rs is*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*GInv g*) = *liftO1 prod.swap* (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*AUnion g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-union' g1 g2*)) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*ATrancl g*) = *liftO1* (*relabel-gtt* ∘ *AGTT-trancl*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*GTrancl g*) = *liftO1 GTT-trancl* (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*AComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-comp' g1 g2*)) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*GComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*GTT-comp' g1 g2*)) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2*)

lemma *gtt-of-gtt-rel-correct*:
assumes ∀ *R* ∈ *set Rs*. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| $\mathcal{F}$
shows *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g* = *Some g'* ⟹ *agtt-lang g'* = *eval-gtt-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *g*
proof (*induct g arbitrary*: *g'*)
note [*simp*] = *bind-eq-Some-conv guard-simps*
have *proj-sq*: *fst* ‘ (*X* × *X*) = *X snd* ‘ (*X* × *X*) = *X* **for** *X* **by** *auto*
{
case (*ARoot is*)
then obtain *w* **where** *w*:*is-to-trs' Rs is* = *Some w* **by** *auto*
then show *?case* **using** *ARoot is-to-trs'-props*[*OF assms w*] *is-to-trs-conv*[*OF w*]
using *agtt-grrstep*
by *auto*
next
case (*GInv g*) **then show** *?case* **by** (*simp add*: *agtt-lang-swap gtt-states-def*)
next
case (*AUnion g1 g2*)
from *AUnion*(*3*)[*simplified*, *THEN conjunct1*] *AUnion*(*3*)[*simplified*, *THEN conjunct2*, *THEN conjunct1*]
obtain *w1 w2* **where**
[*simp*]: *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1* = *Some w1 gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2* = *Some w2*
by *blast*
then show *?case* **using** *AUnion*(*3*)
by (*simp add*: *AGTT-union'-sound AUnion*)
next
case (*ATrancl g*)
from *ATrancl*[*simplified*] **obtain** *w1* **where**
[*simp*]: *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g* = *Some w1 g'* = *relabel-gtt* (*AGTT-trancl w1*) **by** *auto*
then have *fin-lang*: *eval-gtt-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *g* = *agtt-lang w1*
using *ATrancl* **by** *auto*
from *fin-lang* **show** *?case* **using** *AGTT-trancl-sound*[*of w1*]

by *auto*
next
case (*GTrancl g*) **note** ∗ = *GTrancl*(*2*)[*simplified*, *THEN conjunct2*]
show *?case* **unfolding** *gtt-of-gtt-rel.simps GTT-trancl-alang* ∗ *gtrancl-rel-def eval-gtt-rel.simps gmctxt-cl-gmctxtex-onp-conv*
proof ((*intro conjI equalityI subrelI*; (*elim relcompE*)*?*), *goal-cases LR RL*)
case (*LR - - s - z s′ t′ t*)
show *?case* **using** *lift-root-steps-sig-transfer′*[*OF LR*(*2*)[*folded lift-root-step.simps*], *of fset* $\mathcal{F}$]
lift-root-steps-sig-transfer[*OF LR*(*5*)[*folded lift-root-step.simps*], *of fset* $\mathcal{F}$]
image-mono[*OF eval-gtt-rel-sig*[*of fset* $\mathcal{F}$ *map fset Rs g*], *of fst*, *unfolded proj-sq*]
image-mono[*OF eval-gtt-rel-sig*[*of fset* $\mathcal{F}$ *map fset Rs g*], *of snd*, *unfolded proj-sq*]
subsetD[*OF eval-gtt-rel-sig*[*of fset* $\mathcal{F}$ *map fset Rs g*]] *LR*(*1*, *3*, *4*) *GTrancl*
by (*intro relcompI*[*OF - relcompI*, *of - s′ - t′ -*])
(*auto simp*: $\mathcal{T}_G$*-funas-gterm-conv lift-root-step.simps*)
next
case (*RL - - s - z s′ t′ t*)
then show *?case* **using** *GTrancl*
lift-root-step-mono[*of fset* $\mathcal{F}$ *UNIV PAny ESingle eval-gtt-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *g*, *THEN rtrancl-mono*]
unfolding *lift-root-step.simps*[*symmetric*]
by (*intro relcompI*[*OF - relcompI*, *of - s′ - t′ -*])
(*auto simp*: $\mathcal{T}_G$*-funas-gterm-conv lift-root-step-mono trancl-mono*)
qed
next
case (*AComp g1 g2*)
from *AComp*[*simplified*] **obtain** *w1 w2* **where**
[*simp*]: *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1* = *Some w1 gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2* = *Some w2*
g′ = *relabel-gtt* (*AGTT-comp′ w1 w2*) **by** *auto*
then have *fin-lang*: *eval-gtt-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *g1* = *agtt-lang w1*
eval-gtt-rel (*fset* $\mathcal{F}$) (*map fset Rs*) *g2* = *agtt-lang w2*
using *AComp* **by** *auto*
from *fin-lang AGTT-comp′-sound*[*of w1 w2*]
show *?case* **by** *simp*
next
case (*GComp g1 g2*)
let *?r* = λ *g*. *eval-gtt-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *g*
have ∗: *gmctxtex-onp* (λ*C*. *True*) (*?r g1*) = *lift-root-step UNIV PAny EParallel* (*?r g1*)
gmctxtex-onp (λ*C*. *True*) (*?r g2*) = *lift-root-step UNIV PAny EParallel* (*?r g2*)
by (*auto simp*: *lift-root-step.simps*)
show *?case* **using** *GComp*(*3*)
apply (*intro conjI equalityI subrelI*; *simp add*: *gmctxt-cl-gmctxtex-onp-conv GComp*(*1*,*2*) *gtt-comp′-alang gcomp-rel-def* ∗ *flip*: *lift-root-step.simps*; *elim conjE disjE exE relcompE*)
subgoal for *s t - - - - - u*
using *image-mono*[*OF eval-gtt-rel-sig*, *of snd fset* $\mathcal{F}$ *map fset Rs*, *unfolded proj-sq*]

apply (*subst relcompI*[*of* - *u eval-gtt-rel* - - *g1*, *OF* - *lift-root-step-sig-transfer*[*of* - *UNIV PAny EParallel* - *g2 fset* $\mathcal{F}$]])
apply (*force simp add*: *subsetI* $\mathcal{T}_G$*-equivalent-def*)+
done
subgoal for *s t* - - - - - *u*
using *image-mono*[*OF eval-gtt-rel-sig*, *of fst fset* $\mathcal{F}$ *map fset Rs*, *unfolded proj-sq*]
apply (*subst relcompI*[*of* - *u* - - *eval-gtt-rel* - - *g2*, *OF lift-root-step-sig-transfer'*[*of* - *UNIV PAny EParallel* - *g1 fset* $\mathcal{F}$]])
apply (*force simp add*: *subsetI* $\mathcal{T}_G$*-equivalent-def*)+
done
by (*auto intro*: *subsetD*[*OF lift-root-step-mono*[*of fset* $\mathcal{F}$ *UNIV*]])
}
qed

13.3 Computing RR1 and RR2 relations

definition *simplify-reg* $\mathcal{A}$ = (*relabel-reg* (*trim-reg* $\mathcal{A}$))

lemma $\mathcal{L}$*-simplify-reg* [*simp*]: $\mathcal{L}$ (*simplify-reg* $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$
by (*simp add*: *simplify-reg-def* $\mathcal{L}$*-trim*)

lemma *RR1-spec-simplify-reg*[*simp*]:
RR1-spec (*simplify-reg* $\mathcal{A}$) *R* = *RR1-spec* $\mathcal{A}$ *R*
by (*auto simp*: *RR1-spec-def*)
lemma *RR2-spec-simplify-reg*[*simp*]:
RR2-spec (*simplify-reg* $\mathcal{A}$) *R* = *RR2-spec* $\mathcal{A}$ *R*
by (*auto simp*: *RR2-spec-def*)
lemma *RRn-spec-simplify-reg*[*simp*]:
RRn-spec n (*simplify-reg* $\mathcal{A}$) *R* = *RRn-spec n* $\mathcal{A}$ *R*
by (*auto simp*: *RRn-spec-def*)

lemma *RR1-spec-eps-free-reg*[*simp*]:
RR1-spec (*eps-free-reg* $\mathcal{A}$) *R* = *RR1-spec* $\mathcal{A}$ *R*
by (*auto simp*: *RR1-spec-def* $\mathcal{L}$*-eps-free*)
lemma *RR2-spec-eps-free-reg*[*simp*]:
RR2-spec (*eps-free-reg* $\mathcal{A}$) *R* = *RR2-spec* $\mathcal{A}$ *R*
by (*auto simp*: *RR2-spec-def* $\mathcal{L}$*-eps-free*)
lemma *RRn-spec-eps-free-reg*[*simp*]:
RRn-spec n (*eps-free-reg* $\mathcal{A}$) *R* = *RRn-spec n* $\mathcal{A}$ *R*
by (*auto simp*: *RRn-spec-def* $\mathcal{L}$*-eps-free*)

fun *rr1-of-rr1-rel* :: ('*f* × *nat*) *fset* ⇒ ('*f* :: *linorder*, '*v*) *fin-trs list* ⇒ *ftrs rr1-rel* ⇒ (*nat*, '*f*) *reg option*
and *rr2-of-rr2-rel* :: ('*f* × *nat*) *fset* ⇒ ('*f*, '*v*) *fin-trs list* ⇒ *ftrs rr2-rel* ⇒ (*nat*, '*f option* × '*f option*) *reg option* **where**
rr1-of-rr1-rel $\mathcal{F}$ *Rs R1Terms* = *Some* (*relabel-reg* (*term-reg* $\mathcal{F}$))
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1NF is*) = *liftO1* (λ*R*. (*simplify-reg* (*nf-reg* (*fst* |‘| *R*) $\mathcal{F}$))) (*is-to-trs' Rs is*)

```
| rr1-of-rr1-rel F Rs (R1Inf r) = liftO1 (λR.
    let A = trim-reg R in
    simplify-reg (proj-1-reg (Inf-reg-impl A))
  ) (rr2-of-rr2-rel F Rs r)
| rr1-of-rr1-rel F Rs (R1Proj i r) = (case i of 0 ⇒
      liftO1 (trim-reg ∘ proj-1-reg) (rr2-of-rr2-rel F Rs r)
    | - ⇒ liftO1 (trim-reg ∘ proj-2-reg) (rr2-of-rr2-rel F Rs r))
| rr1-of-rr1-rel F Rs (R1Union s1 s2) =
   liftO2 (λ x y. relabel-reg (reg-union x y)) (rr1-of-rr1-rel F Rs s1) (rr1-of-rr1-rel
F Rs s2)
| rr1-of-rr1-rel F Rs (R1Inter s1 s2) =
   liftO2 (λ x y. simplify-reg (reg-intersect x y)) (rr1-of-rr1-rel F Rs s1) (rr1-of-rr1-rel
F Rs s2)
| rr1-of-rr1-rel F Rs (R1Diff s1 s2) = liftO2 (λ x y. relabel-reg (trim-reg (difference-reg
x y))) (rr1-of-rr1-rel F Rs s1) (rr1-of-rr1-rel F Rs s2)

| rr2-of-rr2-rel F Rs (R2GTT-Rel g w x) =
    (case w of PRoot ⇒
     (case x of ESingle ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ GTT-to-RR2-root-reg)
(gtt-of-gtt-rel F Rs g)
       | EParallel ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ reflcl-reg (lift-sig-RR2 |‘|
F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g)
      | EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ GTT-to-RR2-root-reg)
(gtt-of-gtt-rel F Rs g))
      | PNonRoot ⇒
     (case x of ESingle ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ nhole-ctxt-closure-reg
(lift-sig-RR2 |‘| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g)
         | EParallel ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ nhole-mctxt-reflcl-reg
(lift-sig-RR2 |‘| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g)
      | EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ nhole-mctxt-closure-reg
(lift-sig-RR2 |‘| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g))
      | PAny ⇒
       (case x of ESingle ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ ctxt-closure-reg
(lift-sig-RR2 |‘| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g)
         | EParallel ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ parallel-closure-reg
(lift-sig-RR2 |‘| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g)
        | EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-reg ∘ mctxt-closure-reg
(lift-sig-RR2 |‘| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel F Rs g)))
| rr2-of-rr2-rel F Rs (R2Diag s) =
   liftO1 (λ x. fmap-funs-reg (λf. (Some f, Some f)) x) (rr1-of-rr1-rel F Rs s)
| rr2-of-rr2-rel F Rs (R2Prod s1 s2) =
    liftO2 (λ x y. simplify-reg (pair-automaton-reg x y)) (rr1-of-rr1-rel F Rs s1)
(rr1-of-rr1-rel F Rs s2)
| rr2-of-rr2-rel F Rs (R2Inv r) = liftO1 (fmap-funs-reg prod.swap) (rr2-of-rr2-rel
F Rs r)
| rr2-of-rr2-rel F Rs (R2Union r1 r2) =
   liftO2 (λ x y. relabel-reg (reg-union x y)) (rr2-of-rr2-rel F Rs r1) (rr2-of-rr2-rel
F Rs r2)
| rr2-of-rr2-rel F Rs (R2Inter r1 r2) =
```

```
  liftO2 (λ x y. simplify-reg (reg-intersect x y)) (rr2-of-rr2-rel F Rs r1) (rr2-of-rr2-rel
F Rs r2)
| rr2-of-rr2-rel F Rs (R2Diff r1 r2) = liftO2 (λ x y. simplify-reg (difference-reg x
y)) (rr2-of-rr2-rel F Rs r1) (rr2-of-rr2-rel F Rs r2)
| rr2-of-rr2-rel F Rs (R2Comp r1 r2) = liftO2 (λ x y. simplify-reg (rr2-compositon
F x y))
    (rr2-of-rr2-rel F Rs r1) (rr2-of-rr2-rel F Rs r2)


abbreviation lhss where
  lhss R ≡ fst |`| R

lemma rr12-of-rr12-rel-correct:
  fixes Rs :: (('f :: linorder, 'v) Term.term × ('f, 'v) Term.term) fset list
  assumes ∀ R ∈ set Rs. lv-trs (fset R) ∧ ffunas-trs R |⊆| F
  shows ∀ ta1. rr1-of-rr1-rel F Rs r1 = Some ta1 ⟶ RR1-spec ta1 (eval-rr1-rel
(fset F) (map fset Rs) r1)
    ∀ ta2. rr2-of-rr2-rel F Rs r2 = Some ta2 ⟶ RR2-spec ta2 (eval-rr2-rel (fset
F) (map fset Rs) r2)
proof (induct r1 and r2)
  note [simp] = bind-eq-Some-conv guard-simps
  let ?F = fset F let ?Rs = map fset Rs
{
  case R1Terms
  then show ?case using term-automaton[of F]
    by (simp add: T_G-equivalent-def)
next
  case (R1NF r)
  consider (a) ∃ R. is-to-trs' Rs r = Some R | (b) is-to-trs' Rs r = None by auto
  then show ?case
  proof (cases)
    case a
    from a obtain R where [simp]: is-to-trs' Rs r = Some R is-to-fin-trs Rs r =
R
      by (auto simp: is-to-trs'-def)
   from is-to-trs'-props[OF assms this(1)] have inv: ffunas-trs R |⊆| F lv-trs (fset
R) .
    from inv have fl: ∀ l |∈| lhss R. linear-term l
      by (auto simp: lv-trs-def split!: prod.splits)
    {fix s t assume ass: (s, t) ∈ grstep (fset R)
      then obtain C l r σ where step: (l, r) |∈| R term-of-gterm s = (C :: ('f,
'v) ctxt) ⟨l · σ⟩ term-of-gterm t = C⟨r · σ⟩
        unfolding grstep-def by (auto simp: dest!: rstep-imp-C-s-r)
      from step ta-nf-lang-sound[of l lhss R C σ F]
      have s ∉ L (nf-reg (lhss R) F) unfolding L-def
      by (metis fimage-eqI fst-conv nf-reg-def reg.sel(1, 2) term-of-gterm-in-ta-lang-conv)}
    note mem = this
    have funas: funas-trs (fset R) ⊆ ?F using inv(1)
      by (simp add: ffunas-trs.rep-eq less-eq-fset.rep-eq subsetD)
```

{fix s **assume** $s \in \mathcal{L}$ (*nf-reg* (*lhss* R) $\mathcal{F}$)
then have $s \in NF$ (*Restr* (*grstep* (*fset* R)) ($\mathcal{T}_G$ (*fset* $\mathcal{F}$))) $\cap$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
by (*meson IntI NF-I* $\mathcal{T}_G$*-funas-gterm-conv gta-lang-nf-ta-funas inf.cobounded1 mem subset-iff*)**}**
moreover
{fix s **assume** *ass*: $s \in NF$ (*Restr* (*grstep* (*fset* R)) ($\mathcal{T}_G$ (*fset* $\mathcal{F}$))) $\cap$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
then have $*$: (*term-of-gterm* s, *term-of-gterm* t) $\notin$ *rstep* (*fset* R) **for** t **using** *funas*
by (*auto simp*: *funas-trs-def grstep-def NF-iff-no-step* $\mathcal{T}_G$*-funas-gterm-conv*) (*meson R1NF-reps funas rstep.cases*)
then have $s \in \mathcal{L}$ (*nf-reg* (*lhss* R) $\mathcal{F}$) **using** *fl ass*
using *ta-nf-*$\mathcal{L}$*-complete*[*OF fl*, *of* - $\mathcal{F}$] *gta-lang-nf-ta-funas*[*of* - *lhss* R $\mathcal{F}$]
by (*smt* (*verit*, *ccfv-SIG*) *IntE R1NF-reps* $\mathcal{T}_G$*-sound fimageE funas surjective-pairing*)**}**
ultimately have $\mathcal{L}$ (*nf-reg* (*lhss* R) $\mathcal{F}$) = NF (*Restr* (*grstep* (*fset* R)) ($\mathcal{T}_G$ (*fset* $\mathcal{F}$))) $\cap$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
by *blast*
then show *?thesis* **using** *fl*(*1*)
by (*simp add*: *RR1-spec-def is-to-trs-conv*)
qed *auto*
next
case (*R1Inf* r)
consider (*a*) $\exists$ A. *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* r = *Some* A | (*b*) *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* r = *None* **by** *auto*
then show *?case*
proof *cases*
case *a*
have [*simp*]: {u. (t, u) $\in$ *eval-rr2-rel ?F ?Rs* r $\wedge$ *funas-gterm* u $\subseteq$ *?F*} = {u. (t, u) $\in$ *eval-rr2-rel ?F ?Rs* r} **for** t
using *eval-rr12-rel-sig*(*2*)[*of ?F ?Rs* r] **by** (*auto simp*: $\mathcal{T}_G$*-equivalent-def*)
have [*simp*]: *infinite* {u. (t, u) $\in$ *eval-rr2-rel ?F ?Rs* r} $\Longrightarrow$ *funas-gterm* t $\subseteq$ *?F* **for** t
using *eval-rr12-rel-sig*(*2*)[*of ?F ?Rs* r] *not-finite-existsD* **by** (*fastforce simp*: $\mathcal{T}_G$*-equivalent-def*)
from *a* **obtain** A **where** [*simp*]: *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* r = *Some* A **by** *blast*
from *R1Inf this* **have** *spec*: *RR2-spec* A (*eval-rr2-rel ?F ?Rs* r) **by** *auto*
then have *spec-trim*: *RR2-spec* (*trim-reg* A) (*eval-rr2-rel ?F ?Rs* r) **by** *auto*
let *?B* = (*Inf-reg* (*trim-reg* A) (*Q-infty* (*ta* (*trim-reg* A)) $\mathcal{F}$))
have B: *RR2-spec ?B* {(s, t) | s t. *gpair* s t $\in$ $\mathcal{L}$ *?B*}
using *subset-trans*[*OF Inf-automata-subseteq*[*of trim-reg A* $\mathcal{F}$], *of* $\mathcal{L}$ A] *spec*
by (*auto simp*: *RR2-spec-def* $\mathcal{L}$*-trim*)
have $**$: $\mathcal{L}$ (*Inf-reg-impl* (*trim-reg* A)) = $\mathcal{L}$ *?B* **using** *spec*
using *eval-rr12-rel-sig*(*2*)[*of ?F ?Rs* r]
by (*intro Inf-reg-impl-sound*) (*auto simp*: $\mathcal{L}$*-trim RR2-spec-def* $\mathcal{T}_G$*-equivalent-def*)
then have $**$: *RR2-spec* (*Inf-reg-impl* (*trim-reg* A)) {(s, t) | s t. *gpair* s t $\in$ $\mathcal{L}$ *?B*} **using** B
by (*auto simp*: *RR2-spec-def*)
show *?thesis*

```
    using spec eval-rr12-rel-sig(2)[of ?F ?Rs r]
    using ℒ-Inf-reg[OF spec-trim, of ℱ]
    by (auto simp: 𝒯_G-equivalent-def ∗ RR1-spec-def ℒ-trim ℒ-proj(1)[OF ∗∗]
                   Inf-branching-terms-def fImage-singleton)
     (metis (no-types, lifting) SigmaD1 in-mono mem-Collect-eq not-finite-existsD)
  qed auto
next
  case (R1Proj i r)
  then show ?case
  proof (cases i)
    case [simp]:0 show ?thesis using R1Proj
      using proj-automaton-gta-lang(1)[of the (rr2-of-rr2-rel ℱ Rs r) eval-rr2-rel
?F ?Rs r]
      by simp
  next
    case (Suc nat) then show ?thesis using R1Proj
      using proj-automaton-gta-lang(2)[of the (rr2-of-rr2-rel ℱ Rs r) eval-rr2-rel
?F ?Rs r]
      by simp
  qed
next
  case (R1Union s1 s2)
  then show ?case
    by (auto simp: RR1-spec-def ℒ-union)
next
  case (R1Inter s1 s2)
  from R1Inter show ?case
    by (auto simp: ℒ-intersect RR1-spec-def)
next
  case (R1Diff s1 s2)
  then show ?case
    by (auto intro: RR1-difference)
next
  case (R2GTT-Rel g w x)
  note ass = R2GTT-Rel
  consider (a) ∃ A. gtt-of-gtt-rel ℱ Rs g = Some A | (b) gtt-of-gtt-rel ℱ Rs g =
None by blast
  then show ?case
  proof cases
    case a then obtain A where [simp]: gtt-of-gtt-rel ℱ Rs g = Some A by blast
    from gtt-of-gtt-rel-correct[OF assms this]
    have spec [simp]: eval-gtt-rel ?F ?Rs g = agtt-lang A by auto
    let ?B = GTT-to-RR2-root-reg A note [simp] = GTT-to-RR2-root[of A]
    show ?thesis
    proof (cases w)
      case [simp]: PRoot show ?thesis
      proof (cases x)
        case EParallel
        then show ?thesis using reflcl-automaton[of ?B agtt-lang A ℱ]
```

```
        by auto
    qed (auto simp: GTT-to-RR2-root)
  next
    case PNonRoot
    then show ?thesis
      using nhole-ctxt-closure-automaton[of ?B agtt-lang A ℱ]
      using nhole-mctxt-reflcl-automaton[of ?B agtt-lang A ℱ]
      using nhole-mctxt-closure-automaton[of ?B agtt-lang A ℱ]
      by (cases x) auto
  next
    case PAny
    then show ?thesis
      using ctxt-closure-automaton[of ?B agtt-lang A ℱ]
      using parallel-closure-automaton[of ?B agtt-lang A ℱ]
      using mctxt-closure-automaton[of ?B agtt-lang A ℱ]
      by (cases x) auto
  qed
 qed (cases w; cases x, auto)
next
 case (R2Diag s)
 then show ?case
  by (auto simp: RR2-spec-def RR1-spec-def fmap-funs-ℒ Id-on-iff
                 fmap-funs-gta-lang map-funs-term-some-gpair)
next
 case (R2Prod s1 s2)
  then show ?case using pair-automaton[of the (rr1-of-rr1-rel ℱ Rs s1) - the
(rr1-of-rr1-rel ℱ Rs s2)]
  by auto
next
 case (R2Inv r)
 show ?case using R2Inv by (auto simp: swap-RR2-spec)
next
 case (R2Union r1 r2)
 then show ?case using union-automaton
  by (auto simp: RR2-spec-def ℒ-union)
next
 case (R2Inter r1 r2)
 then show ?case
  by (auto simp: ℒ-intersect RR2-spec-def)
next
 case (R2Diff r1 r2)
 then show ?case by (auto intro: RR2-difference)
next
 case (R2Comp r1 r2)
 then show ?case using eval-rr12-rel-sig
  by (auto intro!: rr2-compositon) blast+
}
qed
```

13.4 Misc

lemma *eval-formula-arity-cong*:
assumes $\bigwedge i.\ i < \textit{formula-arity}\ f \Longrightarrow \alpha'\ i = \alpha\ i$
shows *eval-formula* $\mathcal{F}$ *Rs* α' *f* = *eval-formula* $\mathcal{F}$ *Rs* α *f*
proof −
have [*simp*]: *j* < *length fs* ⟹ *i* < *formula-arity* (*fs* ! *j*) ⟹ *i* < *max-list* (*map formula-arity fs*) **for** *i j fs*
by (*simp add*: *less-le-trans max-list*)
show *?thesis* **using** *assms*
proof (*induct f arbitrary*: α α')
case (*FAnd fs*)
show *?case* **using** *FAnd*(*1*)[*OF nth-mem*, *of* - α' α] *FAnd*(*2*) **by** (*auto simp*: *all-set-conv-all-nth*)
next
case (*FOr fs*)
show *?case* **using** *FOr*(*1*)[*OF nth-mem*, *of* - α' α] *FOr*(*2*) **by** (*auto simp*: *ex-set-conv-ex-nth*)
next
case (*FNot f*)
show *?case* **using** *FNot*(*1*)[*of* α' α] *FNot*(*2*) **by** *simp*
next
case (*FExists f*)
show *?case* **using** *FExists*(*1*)[*of* $\alpha'\langle 0 : z\rangle$ $\alpha\langle 0 : z\rangle$ **for** *z*] *FExists*(*2*) **by** (*auto simp*: *shift-def*)
next
case (*FForall f*)
show *?case* **using** *FForall*(*1*)[*of* $\alpha'\langle 0 : z\rangle$ $\alpha\langle 0 : z\rangle$ **for** *z*] *FForall*(*2*) **by** (*auto simp*: *shift-def*)
qed *simp-all*
qed

13.5 Connect semantics to FOL-Fitting

primrec *form-of-formula* :: *'trs formula* ⇒ (*unit*, *'trs rr1-rel* + *'trs rr2-rel*) *form*
where
form-of-formula (*FRR1 r1 x*) = *Pred* (*Inl r1*) [*Var x*]
| *form-of-formula* (*FRR2 r2 x y*) = *Pred* (*Inr r2*) [*Var x*, *Var y*]
| *form-of-formula* (*FAnd fs*) = *foldr And* (*map form-of-formula fs*) *TT*
| *form-of-formula* (*FOr fs*) = *foldr Or* (*map form-of-formula fs*) *FF*
| *form-of-formula* (*FNot f*) = *Neg* (*form-of-formula f*)
| *form-of-formula* (*FExists f*) = *Exists* (*And* (*Pred* (*Inl R1Terms*) [*Var 0*]) (*form-of-formula f*))
| *form-of-formula* (*FForall f*) = *Forall* (*Impl* (*Pred* (*Inl R1Terms*) [*Var 0*]) (*form-of-formula f*))

fun *for-eval-rel* :: (*'f* × *nat*) *set* ⇒ (*'f*, *'v*) *trs list* ⇒ *ftrs rr1-rel* + *ftrs rr2-rel* ⇒ *'f gterm list* ⇒ *bool* **where**
for-eval-rel $\mathcal{F}$ *Rs* (*Inl r1*) [*t*] ⟷ *t* ∈ *eval-rr1-rel* $\mathcal{F}$ *Rs r1*

| *for-eval-rel* $\mathcal{F}$ *Rs* (*Inr r2*) [*t*, *u*] $\longleftrightarrow$ (*t*, *u*) $\in$ *eval-rr2-rel* $\mathcal{F}$ *Rs r2*

lemma *eval-formula-conv*:
eval-formula $\mathcal{F}$ *Rs* α *f* = *eval* α *undefined* (*for-eval-rel* $\mathcal{F}$ *Rs*) (*form-of-formula f*)
proof (*induct f arbitrary*: α)
case (*FAnd fs*) **then show** *?case*
unfolding *eval-formula.simps* **by** (*induct fs*) *auto*
next
case (*FOr fs*) **then show** *?case*
unfolding *eval-formula.simps* **by** (*induct fs*) *auto*
qed *auto*

13.6 RRn relations and formulas

lemma *shift-rangeI* [*intro*!]:
range $\alpha \subseteq T \Longrightarrow x \in T \Longrightarrow$ *range* (*shift* α *i x*) $\subseteq T$
by (*auto simp*: *shift-def*)

definition *formula-relevant* **where**
formula-relevant $\mathcal{F}$ *Rs vs fm* $\longleftrightarrow$
($\forall \alpha\ \alpha'$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \longrightarrow$ *range* $\alpha' \subseteq \mathcal{T}_G\ \mathcal{F} \longrightarrow$ *map* α *vs* = *map* α' *vs* $\longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* α *fm* $\longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* α' *fm*)

lemma *formula-relevant-mono*:
set vs $\subseteq$ *set ws* $\Longrightarrow$ *formula-relevant* $\mathcal{F}$ *Rs vs fm* $\Longrightarrow$ *formula-relevant* $\mathcal{F}$ *Rs ws fm*
unfolding *formula-relevant-def*
by (*meson map-eq-conv subset-code*(*1*))

lemma *formula-relevantD*:
formula-relevant $\mathcal{F}$ *Rs vs fm* $\Longrightarrow$
range $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *range* $\alpha' \subseteq \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *map* α *vs* = *map* α' *vs* $\Longrightarrow$
eval-formula $\mathcal{F}$ *Rs* α *fm* $\Longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* α' *fm*
unfolding *formula-relevant-def*
by *blast*

lemma *trivial-formula-relevant*:
assumes $\bigwedge\alpha$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \Longrightarrow \neg$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
shows *formula-relevant* $\mathcal{F}$ *Rs vs fm*
using *assms* **unfolding** *formula-relevant-def*
by *auto*

lemma *formula-relevant-0-FExists*:
assumes *formula-relevant* $\mathcal{F}$ *Rs* [*0*] *fm*
shows *formula-relevant* $\mathcal{F}$ *Rs* [] (*FExists fm*)
unfolding *formula-relevant-def*
proof (*intro allI*, *intro impI*)
fix α α' **assume** *ass*: *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F}$ *range* (α' :: *fvar* $\Rightarrow$ *'a gterm*) $\subseteq \mathcal{T}_G\ \mathcal{F}$

eval-formula $\mathcal{F}$ *Rs* α (*FExists fm*)
from *ass*(*3*) **obtain** *z* **where** $z \in \mathcal{T}_G$ $\mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* ($\alpha\langle 0 : z\rangle$) *fm*
by *auto*
then show *eval-formula* $\mathcal{F}$ *Rs* α' (*FExists fm*)
using *ass*(*1*, *2*) *formula-relevantD*[*OF assms*, *of* $\alpha\langle 0{:}z\rangle$ $\alpha'\langle 0{:}z\rangle$]
by (*auto simp*: *shift-rangeI intro*!: *exI*[*of* - *z*])
qed

definition *formula-spec* **where**
formula-spec $\mathcal{F}$ *Rs vs A fm* $\longleftrightarrow$ *sorted vs* $\wedge$ *distinct vs* $\wedge$
formula-relevant $\mathcal{F}$ *Rs vs fm* $\wedge$
RRn-spec (*length vs*) *A* {*map* α *vs* |α. *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*}

lemma *formula-spec-RRn-spec*:
formula-spec $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ *RRn-spec* (*length vs*) *A* {*map* α *vs* |α. *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*}
by (*simp add*: *formula-spec-def*)

lemma *formula-spec-nt-empty-form-sat*:
$\neg$ *reg-empty A* $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ $\exists$ α. *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
unfolding *formula-spec-def*
by (*auto simp*: *RRn-spec-def* $\mathcal{L}$*-def*)

lemma *formula-spec-empty*:
reg-empty A $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\Longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* α *fm* $\longleftrightarrow$ *False*
unfolding *formula-spec-def*
by (*auto simp*: *RRn-spec-def* $\mathcal{L}$*-def*)

In each inference step, we obtain a triple consisting of a formula *fm*, a list of relevant variables *vs* (typically a sublist of [*0..<formula-arity fm*]), and an RRn automaton *A*, such that the property *formula-spec* $\mathcal{F}$ *Rs vs A fm* holds.

lemma *false-formula-spec*:
sorted vs $\Longrightarrow$ *distinct vs* $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs empty-reg FFalse*
by (*auto simp*: *formula-spec-def false-RRn-spec FFalse-def formula-relevant-def*)

lemma *true-formula-spec*:
assumes *vs* $\neq$ [] $\vee$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) $\neq$ {} *sorted vs distinct vs*
shows *formula-spec* (*fset* $\mathcal{F}$) *Rs vs* (*true-RRn* $\mathcal{F}$ (*length vs*)) *FTrue*
proof −
have {*ts*. *length ts* = *length vs* $\wedge$ *set ts* $\subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$)} = {*map* α *vs* |α. *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$)}
proof (*intro equalityI subsetI CollectI*, *goal-cases LR RL*)
case (*LR ts*)
moreover obtain *t0* **where** *funas-gterm t0* $\subseteq$ *fset* $\mathcal{F}$ **using** *LR assms*(*1*)
unfolding $\mathcal{T}_G$*-equivalent-def*

by (*cases vs*) *fastforce*+
ultimately show *?case* **using** ‹*distinct vs*›
apply (*intro exI*[*of* - λ*t*. *if t* ∈ *set vs then ts* ! *inv-into* {*0*..<*length vs*} ((!) *vs*) *t else t0*])
apply (*auto intro*!: *nth-equalityI dest*!: *inj-on-nth*[*of vs* {*0*..<*length vs*}] *simp*: *in-set-conv-nth* $\mathcal{T}_G$*-equivalent-def*)
by (*metis inv-to-set mem-Collect-eq subsetD*)
qed *fastforce*
then show *?thesis* **using** *assms true-RRn-spec*[*of length vs* $\mathcal{F}$]
by (*auto simp*: *formula-spec-def FTrue-def formula-relevant-def* $\mathcal{T}_G$*-equivalent-def*)
qed

lemma *relabel-formula-spec*:
formula-spec $\mathcal{F}$ *Rs vs A fm* ⟹ *formula-spec* $\mathcal{F}$ *Rs vs* (*relabel-reg A*) *fm*
by (*simp add*: *formula-spec-def*)

lemma *trim-formula-spec*:
formula-spec $\mathcal{F}$ *Rs vs A fm* ⟹ *formula-spec* $\mathcal{F}$ *Rs vs* (*trim-reg A*) *fm*
by (*simp add*: *formula-spec-def*)

definition *fit-permute* :: *nat list* ⇒ *nat list* ⇒ *nat list* ⇒ *nat list* **where**
fit-permute vs vs' vs'' = *map* (λ*v*. *if v* ∈ *set vs then the* (*mem-idx v vs*) *else length vs* + *the* (*mem-idx v vs''*)) *vs'*

definition *fit-rrn* :: (*'f* × *nat*) *fset* ⇒ *nat list* ⇒ *nat list* ⇒ (*nat*, *'f option list*) *reg* ⇒ (-, *'f option list*) *reg* **where**
fit-rrn $\mathcal{F}$ *vs vs' A* = (*let vs''* = *subtract-list-sorted vs' vs in*
fmap-funs-reg (λ*fs*. *map* ((!) *fs*) (*fit-permute vs vs' vs''*))
(*fmap-funs-reg* (*pad-with-Nones* (*length vs*) (*length vs''*)) (*pair-automaton-reg A* (*true-RRn* $\mathcal{F}$ (*length vs''*)))))

lemma *the-mem-idx-simp* [*simp*]:
distinct xs ⟹ *i* < *length xs* ⟹ *the* (*mem-idx* (*xs* ! *i*) *xs*) = *i*
using *mem-idx-sound*[*THEN iffD1*, *OF nth-mem*, *of i xs*] *mem-idx-sound-output*[*of xs* ! *i xs*] *distinct-conv-nth*
by *fastforce*

lemma *fit-rrn*:
assumes *spec*: *formula-spec* (*fset* $\mathcal{F}$) *Rs vs A fm* **and** *vs*: *sorted vs' distinct vs' set vs* ⊆ *set vs'*
shows *formula-spec* (*fset* $\mathcal{F}$) *Rs vs'* (*fit-rrn* $\mathcal{F}$ *vs vs' A*) *fm*
using *spec* **unfolding** *formula-spec-def formula-relevant-def*
apply (*elim conjE*)
proof (*intro conjI vs*(*1*,*2*) *allI*, *goal-cases rel spec*)
case (*rel* α α') **show** *?case* **using** *vs*(*3*)
by (*fastforce intro*!: *rel*(*3*)[*rule-format*, *of* α α'])
next
case *spec*
define *vs''* **where** *vs''* = *subtract-list-sorted vs' vs*

have *evalI*: *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\Longrightarrow$ *range* $\alpha' \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\Longrightarrow$ *map* α *vs* = *map* α' *vs*
$\Longrightarrow$ *eval-formula* (*fset* $\mathcal{F}$) *Rs* α *fm* $\Longrightarrow$ *eval-formula* (*fset* $\mathcal{F}$) *Rs* α' *fm* **for** α α'
using *spec*(*3*) **by** *blast*
have [*simp*]: *set vs'* = *set vs* $\cup$ *set vs''* *set vs''* $\cap$ *set vs* = {} *set vs* $\cap$ *set vs''* = {} **and** *d*: *distinct vs''*
using *vs spec*(*1*,*2*) **by** (*auto simp*: *vs''-def*)
then have [*dest*]: *v* $\in$ *set vs''* $\Longrightarrow$ *v* $\in$ *set vs* $\Longrightarrow$ *False* **for** *v* **by** *blast*
note $*$ = *permute-automaton*[*OF append-automaton*[*OF spec*(*4*) *true-RRn-spec*, *of length vs''*]]
have [*simp*]: *distinct vs* $\Longrightarrow$ *i* $\in$ *set vs* $\Longrightarrow$ *vs* ! *the* (*mem-idx i vs*) = (*i* :: *nat*) **for** *vs i*
by (*simp add*: *mem-idx-sound mem-idx-sound-output*)
have [*dest*]: *distinct vs* $\Longrightarrow$ *i* $\in$ *set vs* $\Longrightarrow$ $\neg$ *the* (*mem-idx i vs*) < *length vs* $\Longrightarrow$ *False* **for** *i*
by (*meson mem-idx-sound2 mem-idx-sound-output option.exhaust-sel*)
show *?case* **unfolding** *fit-rrn-def Let-def vs''-def*[*symmetric*] $\mathcal{T}_G$*-equivalent-def*
apply (*rule subst*[**where** *P* = λ*l*. *RRn-spec l - -*, *OF - subst*[**where** *P* = λ*ta*. *RRn-spec - - ta*, *OF - ∗*]])
subgoal by (*simp add*: *fit-permute-def*)
subgoal
apply (*intro equalityI subsetI CollectI imageI*; *elim imageE CollectE exE conjE*; *unfold* $\mathcal{T}_G$*-equivalent-def*)
subgoal for *x fs ts us* α
using *spec*(*1*, *2*) *d*
apply (*intro exI*[*of - λv. if v* $\in$ *set vs''* *then us* ! *the* (*mem-idx v vs''*) *else* α *v*])
apply (*auto simp*: *fit-permute-def nth-append* $\mathcal{T}_G$*-equivalent-def*
intro!: *nth-equalityI evalI*[*of* α λ*v*. *if v* $\in$ *set vs''* *then us* ! *the* (*mem-idx v vs''*) *else* α *v*])
apply (*metis distinct-Ex1 in-mono mem-Collect-eq nth-mem the-mem-idx-simp*)
apply (*metis distinct-Ex1 in-mono mem-Collect-eq nth-mem the-mem-idx-simp*)
apply *blast*
by (*meson* ‹$\bigwedge$*va*. ⟦*va* $\in$ *set vs''*; *va* $\in$ *set vs*⟧ $\Longrightarrow$ *False*› *nth-mem*)
subgoal premises *p* **for** *xs* α
apply (*intro rev-image-eqI*[*of map* α (*vs* @ *vs''*)])
subgoal using *p* **by** (*force intro*!: *exI*[*of - map* α *vs*, *OF exI*[*of - map* α *vs''*]])
subgoal using *p*(*1*)
by (*force intro*!: *nth-equalityI simp*: *fit-permute-def comp-def nth-append dest*: *iffD1*[*OF mem-idx-sound*] *mem-idx-sound-output*)
done
done
subgoal using *vs spec*(*1*,*2*) **unfolding** *fit-permute-def*
apply (*intro equalityI subsetI*)
subgoal by (*auto 0 3 dest*: *iffD1*[*OF mem-idx-sound*] *mem-idx-sound-output*)
subgoal for *x*
apply (*simp add*: *Compl-eq*[*symmetric*] *Diff-eq*[*symmetric*] *Un-Diff Diff-triv Int-absorb1*)

apply (*simp add*: *nth-image*[*symmetric*, *of length xs xs* **for** *xs*, *simplified*] *image-iff comp-def*)
using *image-cong*[*OF refl arg-cong*[*OF the-mem-idx-simp*]] ‹*distinct vs''*›
by (*smt* (*verit*) *add-diff-inverse-nat add-less-cancel-left atLeast0LessThan lessThan-iff the-mem-idx-simp*)
done
done
qed

definition *fit-rrns* :: ('*f* × *nat*) *fset* ⇒ (*ftrs formula* × *nat list* × (*nat*, '*f option list*) *reg*) *list* ⇒
nat list × ((*nat*, '*f option list*) *reg*) *list* **where**
fit-rrns $\mathcal{F}$ *rrns* = (*let vs'* = *fold union-list-sorted* (*map* (*fst* ∘ *snd*) *rrns*) [] *in*
(*vs'*, *map* (λ(*fm*, *vs*, *ta*). *relabel-reg* (*trim-reg* (*fit-rrn* $\mathcal{F}$ *vs vs' ta*))) *rrns*))

lemma *sorted-union-list-sortedI* [*simp*]:
sorted xs ⟹ *sorted ys* ⟹ *sorted* (*union-list-sorted xs ys*)
by (*induct xs ys rule*: *union-list-sorted.induct*) *auto*

lemma *distinct-union-list-sortedI* [*simp*]:
sorted xs ⟹ *sorted ys* ⟹ *distinct xs* ⟹ *distinct ys* ⟹ *distinct* (*union-list-sorted xs ys*)
by (*induct xs ys rule*: *union-list-sorted.induct*) *auto*

lemma *fit-rrns*:
assumes *infs*: ⋀*fvA*. *fvA* ∈ *set rrns* ⟹ *formula-spec* (*fset* $\mathcal{F}$) *Rs* (*fst* (*snd fvA*)) (*snd* (*snd fvA*)) (*fst fvA*)
assumes (*vs'*, *tas'*) = *fit-rrns* $\mathcal{F}$ *rrns*
shows *length tas'* = *length rrns* ⋀*i*. *i* < *length rrns* ⟹ *formula-spec* (*fset* $\mathcal{F}$) *Rs vs'* (*tas'* ! *i*) (*fst* (*rrns* ! *i*))
distinct vs' sorted vs'
proof (*goal-cases*)
have *vs'*: *vs'* = *fold union-list-sorted* (*map* (*fst* ∘ *snd*) *rrns*) [] **using** *assms*(*2*) **by** (*simp add*: *fit-rrns-def Let-def*)
have ∗: *sorted vs' distinct vs'* ⋀*fvA*. *fvA* ∈ *set rrns* ⟹ *set* (*fst* (*snd fvA*)) ⊆ *set vs'*
using *infs*[*unfolded formula-spec-def*, *THEN conjunct2*, *THEN conjunct1*]
infs[*unfolded formula-spec-def*, *THEN conjunct1*]
unfolding *vs'* **by** (*induct rrns rule*: *rev-induct*) *auto*
{
case *1* **then show** *?case* **using** *assms*(*2*) **by** (*simp add*: *fit-rrns-def Let-def*)
next
case (*2 i*)
have *tas'*: *tas'* ! *i* = *relabel-reg* (*trim-reg* (*fit-rrn* $\mathcal{F}$ (*fst* (*snd* (*rrns* ! *i*))) *vs'* (*snd* (*snd* (*rrns* ! *i*)))))
using *2 assms*(*2*) **by** (*simp add*: *fit-rrns-def Let-def split*: *prod.splits*)
from ∗(*1*,*2*) ∗(*3*)[*OF nth-mem*] **show** *?case* **using** *2* **unfolding** *tas'*
by (*auto intro*!: *relabel-formula-spec trim-formula-spec fit-rrn 2 assms*(*1*,*2*))
next

case *3* **show** *?case* **by** (*rule* ∗)
next
case *4* **show** *?case* **by** (*rule* ∗)
}
qed

13.7 Building blocks

definition *for-rrn* **where**
for-rrn tas = *fold* (λ*A B*. *relabel-reg* (*reg-union A B*)) *tas* (*Reg* {||} (*TA* {||} {||}))

lemma *for-rrn*:
assumes *length tas* = *length fs* ⋀*i*. *i* < *length fs* ⟹ *formula-spec* ℱ *Rs vs* (*tas* ! *i*) (*fs* ! *i*)
and *vs*: *sorted vs distinct vs*
shows *formula-spec* ℱ *Rs vs* (*for-rrn tas*) (*FOr fs*)
using *assms*(*1*,*2*) **unfolding** *for-rrn-def*
proof (*induct fs arbitrary*: *tas rule*: *rev-induct*)
case *Nil* **then show** *?case* **using** *vs false-formula-spec*[*of vs* ℱ *Rs*] **by** (*auto simp*: *FFalse-def*)
next
case (*snoc fm fs*)
have ∗: *Bex* (*set* [*x*]) *P* = *P x* **for** *P x* **by** *auto*
have [*intro*!]: *formula-spec* ℱ *Rs vs* (*reg-union A B*) (*FOr* (*fs* @ [*fm*])) **if**
formula-spec ℱ *Rs vs A fm formula-spec* ℱ *Rs vs B* (*FOr fs*) **for** *A B* **using** *that*
unfolding *formula-spec-def*
apply (*intro conjI*, *blast*, *blast*)
subgoal unfolding *formula-relevant-def eval-formula.simps set-append bex-Un* ∗ **by** *blast*
apply (*elim conjE*)
subgoal premises *p* **by** (*rule subst*[*of* - - *RRn-spec* - -, *OF* - *union-automaton*[*OF p*(*6*,*8*)]]) *auto*
done
show *?case* **using** *snoc*(*1*)[*of take* (*length fs*) *tas*] *snoc*(*2*) *snoc*(*3*)[*simplified*, *OF less-SucI*] *snoc*(*3*)[*of length fs*] *vs*
by (*cases tas rule*: *rev-exhaust*) (*auto simp*: *min-def nth-append intro*!: *relabel-formula-spec*)
qed

fun *fand-rrn* **where**
fand-rrn ℱ *n* [] = *true-RRn* ℱ *n*
| *fand-rrn* ℱ *n* (*A* # *tas*) = *fold* (λ*A B*. *simplify-reg* (*reg-intersect A B*)) *tas A*

lemma *fand-rrn*:
assumes $\mathcal{T}_G$ (*fset* ℱ) ≠ {} *length tas* = *length fs* ⋀*i*. *i* < *length fs* ⟹ *formula-spec* (*fset* ℱ) *Rs vs* (*tas* ! *i*) (*fs* ! *i*)
and *vs*: *sorted vs distinct vs*

shows *formula-spec* (*fset* $\mathcal{F}$) *Rs vs* (*fand-rrn* $\mathcal{F}$ (*length vs*) *tas*) (*FAnd fs*)
proof (*cases fs*)
case *Nil*
have *tas* = [] **using** *assms*(*2*) **by** (*auto simp*: *Nil*)
then show *?thesis* **using** *true-formula-spec*[*OF - vs*, *of* $\mathcal{F}$ *Rs*] *assms*(*1*) *Nil*
by (*simp add*: *FTrue-def*)
next
case (*Cons fm fs*)
obtain *ta tas'* **where** *tas*: *tas* = *ta* # *tas'* **using** *assms*(*2*) *Cons* **by** (*cases tas*) *auto*
show *?thesis* **using** *assms*(*2*) *assms*(*3*)[*of Suc -*] **unfolding** *tas Cons*
unfolding *list.size add-Suc-right add-0-right nat.inject Suc-less-eq nth-Cons-Suc fand-rrn.simps*
proof (*induct fs arbitrary*: *tas' rule*: *rev-induct*)
case *Nil*
have *formula-relevant* (*fset* $\mathcal{F}$) *Rs vs* (*FAnd* [*fm*]) **using** *assms*(*3*)[*of 0*]
apply (*simp add*: *tas Cons formula-spec-def*)
unfolding *formula-relevant-def eval-formula.simps in-set-simps* **by** *blast*
then show *?case* **using** *assms*(*3*)[*of 0*, *unfolded tas Cons*, *simplified*] *Nil* **by** (*simp add*: *formula-spec-def*)
next
case (*snoc fm' fs*)
have ∗: *Ball* (*insert x X*) *P* ⟷ *P x* ∧ *Ball X P* **for** *P x X* **by** *auto*
have [*intro*!]: *formula-spec* (*fset* $\mathcal{F}$) *Rs vs* (*reg-intersect A B*) (*FAnd* (*fm* # *fs* @ [*fm'*])) **if**
formula-spec (*fset* $\mathcal{F}$) *Rs vs A fm' formula-spec* (*fset* $\mathcal{F}$) *Rs vs B* (*FAnd* (*fm* # *fs*)) **for** *A B* **using** *that*
unfolding *formula-spec-def*
apply (*intro conjI*, *blast*, *blast*)
subgoal unfolding *formula-relevant-def eval-formula.simps set-append set-simps ball-simps ball-Un in-set-simps* ∗
by *blast*
apply (*elim conjE*)
subgoal premises *p*
by (*rule subst*[*of - - RRn-spec - -*, *OF - intersect-automaton*[*OF p*(*6*,*8*)]])
(*auto dest*: *p*(*5*)[*unfolded formula-relevant-def*, *rule-format*])
done
show *?case* **using** *snoc*(*1*)[*of take* (*length fs*) *tas'*] *snoc*(*2*) *snoc*(*3*)[*simplified*, *OF less-SucI*] *snoc*(*3*)[*of length fs*] *vs*
by (*cases tas' rule*: *rev-exhaust*) (*auto simp*: *min-def nth-append simplify-reg-def intro*!: *relabel-formula-spec trim-formula-spec*)
qed
qed

13.7.1 IExists inference rule

lemma *lift-fun-gpairD*:
map-gterm lift-fun s = *gpair t u* ⟹ *t* = *s*
map-gterm lift-fun s = *gpair t u* ⟹ *u* = *s*

by (*metis gfst-gpair gsnd-gpair map-funs-term-some-gpair*)+

definition *upd-bruijn* :: *nat list* ⇒ *nat list* **where**
upd-bruijn vs = *tl* (*map* (λ *x*. *x* − *1*) *vs*)

lemma *upd-bruijn-length*[*simp*]:
length (*upd-bruijn vs*) = *length vs* − *1*
by (*induct vs*) (*auto simp*: *upd-bruijn-def*)

lemma *pres-sorted-dec*:
sorted xs ⟹ *sorted* (*map* (λ*x*. *x* − *Suc 0*) *xs*)
by (*induct xs*) *auto*

lemma *upd-bruijn-pres-sorted*:
sorted xs ⟹ *sorted* (*upd-bruijn xs*)
unfolding *upd-bruijn-def*
by (*intro sorted-tl*) (*auto simp*: *pres-sorted-dec*)

lemma *pres-distinct-not-0-list-dec*:
distinct xs ⟹ *0* ∉ *set xs* ⟹ *distinct* (*map* (λ*x*. *x* − *Suc 0*) *xs*)
by (*induct xs*) (*auto*, *metis Suc-pred neq0-conv*)

lemma *upd-bruijn-pres-distinct*:
assumes *sorted xs distinct xs*
shows *distinct* (*upd-bruijn xs*)
proof −
have *sorted* (*ys* :: *nat list*) ⟹ *distinct ys* ⟹ *0* ∉ *set* (*tl ys*) **for** *ys*
by (*induct ys*) *auto*
from *this*[*OF assms*] **show** *?thesis* **using** *assms*(*2*)
using *pres-distinct-not-0-list-dec*[*OF distinct-tl*, *OF assms*(*2*)]
unfolding *upd-bruijn-def*
by (*simp add*: *map-tl*)
qed

lemma *upd-bruijn-relevant-inv*:
assumes *sorted vs distinct vs 0* ∈ *set vs*
and ⋀ *x*. *x* ∈ *set* (*upd-bruijn vs*) ⟹ α *x* = α′ *x*
shows ⋀ *x*. *x* ∈ *set vs* ⟹ (*shift* α *0 z*) *x* = (*shift* α′ *0 z*) *x*
using *assms* **unfolding** *upd-bruijn-def*
by (*induct vs*) (*auto simp add*: *FOL-Fitting.shift-def*)

lemma *ExistsI-upd-brujin-0*:
assumes *sorted vs distinct vs 0* ∈ *set vs formula-relevant* $\mathcal{F}$ *Rs vs fm*
shows *formula-relevant* $\mathcal{F}$ *Rs* (*upd-bruijn vs*) (*FExists fm*)
unfolding *formula-relevant-def*
proof (*intro allI*, *intro impI*)
fix α α′ **assume** *ass*: *range* α ⊆ $\mathcal{T}_G$ $\mathcal{F}$ *range* (α′ :: *fvar* ⇒ *'a gterm*) ⊆ $\mathcal{T}_G$ $\mathcal{F}$
map α (*upd-bruijn vs*) = *map* α′ (*upd-bruijn vs*) *eval-formula* $\mathcal{F}$ *Rs* α (*FExists fm*)

from *ass*(*4*) **obtain** *z* **where** $z \in \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* ($\alpha\langle 0 : z\rangle$) *fm*
by *auto*
then show *eval-formula* $\mathcal{F}$ *Rs* α' (*FExists fm*)
using *ass*(*1 − 3*) *formula-relevantD*[*OF assms*(*4*), *of* $\alpha\langle 0{:}z\rangle$ $\alpha'\langle 0{:}z\rangle$]
using *upd-bruijn-relevant-inv*[*OF assms*(*1 − 3*), *of* α α']
by (*auto simp*: *shift-rangeI intro*!: *exI*[*of - z*])
qed

declare *subsetI*[*rule del*]
lemma *ExistsI-upd-brujin-no-0*:
assumes *0* $\notin$ *set vs* **and** *formula-relevant* $\mathcal{F}$ *Rs vs fm*
shows *formula-relevant* $\mathcal{F}$ *Rs* (*map* ($\lambda x.\ x - Suc\ 0$) *vs*) (*FExists fm*)
unfolding *formula-relevant-def*
proof ((*intro allI*)+ , (*intro impI*)+, *unfold eval-formula.simps*)
fix α α' **assume** *st*: *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F}$ *range* $\alpha' \subseteq \mathcal{T}_G\ \mathcal{F}$
map α (*map* ($\lambda x.\ x - Suc\ 0$) *vs*) = *map* α' (*map* ($\lambda x.\ x - Suc\ 0$) *vs*)
$\exists\ z \in \mathcal{T}_G\ \mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* (*shift* α *0 z*) *fm*
then obtain *z* **where** *w*: $z \in \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* (*shift* α *0 z*) *fm* **by** *auto*
from *this*(*1*) **have** *eval-formula* $\mathcal{F}$ *Rs* (*shift* α' *0 z*) *fm*
using *st*(*1 − 3*) *assms*(*1*) *FOL-Fitting.shift-def*
apply (*intro formula-relevantD*[*OF assms*(*2*) - - - *w*(*2*), *of shift* α' *0 z*])
by *auto* (*simp add*: *FOL-Fitting.shift-def*)
then show $\exists\ z \in \mathcal{T}_G\ \mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* (*shift* α' *0 z*) *fm* **using** *w*(*1*)
by *blast*
qed

definition *shift-right* **where**
shift-right $\alpha \equiv \lambda\ i.\ \alpha\ (i + 1)$

lemma *shift-right-nt-0*:
$i \neq 0 \Longrightarrow \alpha\ i =$ *shift-right* $\alpha\ (i - Suc\ 0)$
unfolding *shift-right-def*
by *auto*

lemma *shift-shift-right-id* [*simp*]:
shift (*shift-right* α) *0* (α *0*) = α
by (*auto simp*: *shift-def shift-right-def*)

lemma *shift-right-rangeI* [*intro*]:
range $\alpha \subseteq T \Longrightarrow$ *range* (*shift-right* α) $\subseteq T$
by (*auto simp*: *shift-right-def intro*: *subsetI*)

lemma *eval-formula-shift-right-eval*:
eval-formula $\mathcal{F}$ *Rs* α *fm* $\Longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* (*shift* (*shift-right* α) *0* (α *0*)) *fm*
eval-formula $\mathcal{F}$ *Rs* (*shift* (*shift-right* α) *0* (α *0*)) *fm* $\Longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
by (*auto*)
declare *subsetI*[*intro*!]

lemma *nt-rel-0-trivial-shift*:

assumes $0 \notin$ *set vs*
shows $\{$*map* α *vs* $|\alpha$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*$\} =$
$\{$*map* $(\lambda x.\ \alpha\ (x - Suc\ 0))$ *vs* $|\alpha$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z \in \mathcal{T}_G\ \mathcal{F}.$ *eval-formula* $\mathcal{F}$ *Rs* $(\alpha\langle 0{:}z\rangle)$ *fm*$)\}$
(**is** *?Ls* $=$ *?Rs*)
proof
{fix α **assume** *ass*: *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
then have *map* α *vs* $=$ *map* $(\lambda x.$ (*shift-right* α) $(x - Suc\ 0))$ *vs*
range (*shift-right* α) $\subseteq \mathcal{T}_G\ \mathcal{F}\ \alpha\ 0 \in \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* (*shift* (*shift-right* α) *0* (α *0*)) *fm*
using *shift-right-rangeI*[*OF ass*(*1*)] *assms*
by (*auto intro*: *eval-formula-shift-right-eval*(*1*), *metis shift-right-nt-0*)**}**
then show *?Ls* $\subseteq$ *?Rs*
by *blast*
next
show *?Rs* $\subseteq$ *?Ls*
by *auto* (*metis FOL-Fitting.shift-def One-nat-def assms not-less0 shift-rangeI*)
qed

lemma *relevant-vars-upd-bruijn-tl*:
assumes *sorted vs distinct vs*
shows *map* (*shift-right* α) (*upd-bruijn vs*) $=$ *tl* (*map* α *vs*) **using** *assms*
proof (*induct vs*)
case (*Cons a vs*) **then show** *?case*
using *le-antisym*
by (*auto simp*: *upd-bruijn-def shift-right-def*)
(*metis One-nat-def Suc-eq-plus1 le-0-eq shift-right-def shift-right-nt-0*)
qed (*auto simp*: *upd-bruijn-def*)

lemma *drop-upd-bruijn-set*:
assumes *sorted vs distinct vs*
shows *drop 1* ‘ $\{$*map* α *vs* $|\alpha$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*$\} =$
$\{$*map* α (*upd-bruijn vs*) $|\alpha$. *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z \in \mathcal{T}_G\ \mathcal{F}.$ *eval-formula* $\mathcal{F}$ *Rs* $(\alpha\langle 0{:}z\rangle)$ *fm*$)\}$
(**is** *?Ls* $=$ *?Rs*)
proof
{fix α **assume** *ass*: *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
then have *drop 1* (*map* α *vs*) $=$ *map* (*shift-right* α) (*upd-bruijn vs*)
range (*shift-right* α) $\subseteq \mathcal{T}_G\ \mathcal{F}\ \alpha\ 0 \in \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* (*shift* (*shift-right* α) *0* (α *0*)) *fm*
using *shift-right-rangeI*[*OF ass*(*1*)]
by (*auto simp*: *tl-drop-conv relevant-vars-upd-bruijn-tl*[*OF assms*(*1*, *2*)])**}**
then show *?Ls* $\subseteq$ *?Rs*
by *blast*
next
have [*dest*]: $0 \in$ *set* (*tl vs*) $\Longrightarrow$ *False* **using** *assms*(*1*, *2*)
by (*cases vs*) *auto*
{fix α *z* **assume** *ass*: *range* $\alpha \subseteq \mathcal{T}_G\ \mathcal{F}\ z \in \mathcal{T}_G\ \mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* $(\alpha\langle 0{:}z\rangle)$ *fm*

then have *map* α (*upd-bruijn vs*) = *tl* (*map* ($\alpha\langle 0{:}z\rangle$) *vs*)
range ($\alpha\langle 0{:}z\rangle$) $\subseteq \mathcal{T}_G$ $\mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* ($\alpha\langle 0{:}z\rangle$) *fm*
using *shift-rangeI*[*OF ass*(*1*)]
by (*auto simp*: *upd-bruijn-def shift-def simp flip*: *map-tl*)**}**
then show *?Rs* $\subseteq$ *?Ls*
by (*auto simp*: *tl-drop-conv image-def*) *blast*
qed

lemma *closed-sat-form-env-dom*:
assumes *formula-relevant* $\mathcal{F}$ *Rs* [] (*FExists fm*) *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
shows {[α *0*] |α. *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\wedge$ ($\exists$ *z* $\in \mathcal{T}_G$ $\mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* ($\alpha\langle 0{:}z\rangle$) *fm*)} = {[*t*] | *t*. *t* $\in \mathcal{T}_G$ $\mathcal{F}$}
using *formula-relevantD*[*OF assms*(*1*)] *assms*(*2*−)
apply *auto*
apply *blast*
by (*smt* (*verit*) *rangeI shift-eq shift-rangeI shift-right-rangeI shift-shift-right-id subsetD*)

lemma *find-append*:
find P (*xs* @ *ys*) = (*if find P xs* ≠ *None then find P xs else find P ys*)
by (*induct xs arbitrary*: *ys*) (*auto split*!: *if-splits*)

13.8 Checking inferences

derive *linorder ext-step pos-step gtt-rel rr1-rel rr2-rel ftrs*
derive *compare ext-step pos-step gtt-rel rr1-rel rr2-rel ftrs*

fun *check-inference* :: ((*'f* × *nat*) *fset* ⇒ (*'f*, *'v*) *fin-trs list* ⇒ *ftrs rr1-rel* ⇒ (*nat*, *'f*) *reg option*)
⇒ ((*'f* × *nat*) *fset* ⇒ (*'f*, *'v*) *fin-trs list* ⇒ *ftrs rr2-rel* ⇒ (*nat*, *'f option* × *'f option*) *reg option*)
⇒ (*'f* × *nat*) *fset* ⇒ (*'f* :: *compare*, *'v*) *fin-trs list*
⇒ (*ftrs formula* × *nat list* × (*nat*, *'f option list*) *reg*) *list*
⇒ (*nat* × *ftrs inference* × *ftrs formula* × *info list*)
⇒ (*ftrs formula* × *nat list* × (*nat*, *'f option list*) *reg*) *option* **where**
check-inference rr1c rr2c $\mathcal{F}$ *Rs infs* (*l*, *step*, *fm*, *is*) = *do* {
guard (*l* = *length infs*);
case step of
IRR1 s x ⇒ *do* {
guard (*fm* = *FRR1 s x*);
liftO1 (λ*ta*. (*FRR1 s x*, [*x*], *fmap-funs-reg* (λ*f*. [*Some f*]) *ta*)) (*rr1c* $\mathcal{F}$ *Rs s*)
}
| *IRR2 r x y* ⇒ *do* {
guard (*fm* = *FRR2 r x y*);
case compare x y of
Lt ⇒ *liftO1* (λ*ta*. (*FRR2 r x y*, [*x*, *y*], *fmap-funs-reg* (λ(*f*, *g*). [*f*, *g*]) *ta*))

```
(rr2c F Rs r)
      | Eq ⇒ liftO1 (λta. (FRR2 r x y, [x], fmap-funs-reg (λf. [Some f]) ta))
        (liftO1 (simplify-reg ∘ proj-1-reg)
        (liftO2 (λ t1 t2. simplify-reg (reg-intersect t1 t2)) (rr2c F Rs r) (rr2c F
Rs (R2Diag R1Terms))))
      | Gt ⇒ liftO1 (λta. (FRR2 r x y, [y, x], fmap-funs-reg (λ(f, g). [g, f]) ta))
(rr2c F Rs r)
  }
  | IAnd ls ⇒ do {
      guard (∀ l' ∈ set ls. l' < l);
      guard (fm = FAnd (map (λl'. fst (infs ! l')) ls));
      let (vs', tas') = fit-rrns F (map ((!) infs) ls) in
      Some (fm, vs', fand-rrn F (length vs') tas')
  }
  | IOr ls ⇒ do {
      guard (∀ l' ∈ set ls. l' < l);
      guard (fm = FOr (map (λl'. fst (infs ! l')) ls));
      let (vs', tas') = fit-rrns F (map ((!) infs) ls) in
      Some (fm, vs', for-rrn tas')
  }
  | INot l' ⇒ do {
      guard (l' < l);
      guard (fm = FNot (fst (infs ! l')));
      let (vs', tas') = snd (infs ! l');
      Some (fm, vs', simplify-reg (difference-reg (true-RRn F (length vs')) tas'))
  }
  | IExists l' ⇒ do {
      guard (l' < l);
      guard (fm = FExists (fst (infs ! l')));
      let (vs', tas') = snd (infs ! l');
      if length vs' = 0 then Some (fm, [], tas') else
        if reg-empty tas' then Some (fm, [], empty-reg)
        else if 0 ∉ set vs' then Some (fm, map (λ x. x − 1) vs', tas')
        else if 1 = length vs' then Some (fm, [], true-RRn F 0)
        else Some (fm, upd-bruijn vs', rrn-drop-fst tas')
  }
  | IRename l' vs ⇒ guard (l' < l) ≫ None
  | INNFPlus l' ⇒ do {
      guard (l' < l);
      let fm' = fst (infs ! l');
        guard (ord-form-list-aci (nnf-to-list-aci (nnf (form-of-formula fm'))) =
ord-form-list-aci (nnf-to-list-aci (nnf (form-of-formula fm))));
      Some (fm, snd (infs ! l'))
  }
  | IRepl eq pos l' ⇒ guard (l' < l) ≫ None
  }

lemma RRn-spec-true-RRn:
  RRn-spec (Suc 0) (true-RRn F (Suc 0)) {[t] |t. t ∈ T_G (fset F)}
```

apply (*auto simp*: *RRn-spec-def* $\mathcal{T}_G$*-equivalent-def fmap-funs-*$\mathcal{L}$
image-def term-automaton[*of* $\mathcal{F}$, *unfolded RR1-spec-def*])
apply (*metis gencode-singleton*)+
done

lemma *check-inference-correct*:
assumes *sig*: $\mathcal{T}_G$ (*fset* $\mathcal{F}$) $\neq$ {} **and** *Rs*: $\forall$ *R* $\in$ *set Rs*. *lv-trs* (*fset R*) $\wedge$ *ffunas-trs R* $|\subseteq|$ $\mathcal{F}$
assumes *infs*: $\bigwedge$*fvA*. *fvA* $\in$ *set infs* $\Longrightarrow$ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd fvA*)) (*snd* (*snd fvA*)) (*fst fvA*)
assumes *inf*: *check-inference rr1c rr2c* $\mathcal{F}$ *Rs infs* (*l*, *step*, *fm*, *is*) = *Some* (*fm'*, *vs*, *A'*)
assumes *rr1*: $\bigwedge$*r1*. $\forall$ *ta1*. *rr1c* $\mathcal{F}$ *Rs r1* = *Some ta1* $\longrightarrow$ *RR1-spec ta1* (*eval-rr1-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r1*)
assumes *rr2*: $\bigwedge$*r2*. $\forall$ *ta2*. *rr2c* $\mathcal{F}$ *Rs r2* = *Some ta2* $\longrightarrow$ *RR2-spec ta2* (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r2*)
shows *l* = *length infs* $\wedge$ *fm* = *fm'* $\wedge$ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) *vs A' fm'*
using *inf*
proof (*induct step*)
note [*simp*] = *bind-eq-Some-conv guard-simps*
let *?F* = *fset* $\mathcal{F}$ **let** *?Rs* = *map fset Rs*
{
case (*IRR1 s x*)
then show *?case*
using *rr1*[*rule-format*, *of s*]
subsetD[*OF eval-rr12-rel-sig*(*1*), *of* - *?F ?Rs s*]
by (*force simp*: *formula-spec-def formula-relevant-def RR1-spec-def* $\mathcal{T}_G$*-equivalent-def intro*!: *RR1-to-RRn-spec*[*of* - (*λα*. *α x*) ‘ *Collect P* **for** *P*, *unfolded image-comp*, *unfolded image-Collect comp-def One-nat-def*])
next
case (*IRR2 r x y*)
then show *?case* **using** *rr2*[*rule-format*, *of r*]
subsetD[*OF eval-rr12-rel-sig*(*2*), *of* - *?F ?Rs r*]
two-comparisons-into-compare(*1*)[*of x y x* = *y x* < *y x* > *y*]
proof (*induct compare x y*)
note [*intro*!] = *RR1-to-RRn-spec*[*of* - (*λα*. *α y*) ‘ *Collect P* **for** *P*, *unfolded image-comp*,
unfolded image-Collect comp-def One-nat-def prod.simps]
case *Eq*
then obtain *A* **where** *w*[*simp*]: *rr2c* $\mathcal{F}$ *Rs r* = *Some A* **by** *auto*
from *Eq* **obtain** *B* **where** [*simp*]:*rr2c* $\mathcal{F}$ *Rs* (*R2Diag R1Terms*) = *Some B* **by** *auto*
let *?B* = *reg-intersect A B*
from *Eq*(*3*)[*OF w*] **have** *RR2-spec ?B* (*eval-rr2-rel ?F ?Rs r* $\cap$ *Restr Id* ($\mathcal{T}_G$ *?F*))
using *rr2*[*rule-format*, *of R2Diag R1Terms B*]
by (*auto simp add*: $\mathcal{L}$*-intersect RR2-spec-def dest*: *lift-fun-gpairD*)
then have *RR2-spec* (*relabel-reg* (*trim-reg ?B*)) (*eval-rr2-rel ?F ?Rs r* $\cap$ *Restr*

Id ($\mathcal{T}_G$ *?F*)) **by** *simp*
from *proj-1*(*1*)[*OF this*]
have *RR1-spec* (*proj-1-reg* (*relabel-reg* (*trim-reg ?B*))) {α *y* |α. *range* $\alpha \subseteq$ *gterms ?F* $\wedge$ (α *y*, α *y*) $\in$ *eval-rr2-rel ?F ?Rs r*}
apply (*auto simp*: *RR1-spec-def* $\mathcal{T}_G$*-equivalent-def image-iff*)
by (*metis Eq.prems*(*3*) *IdI IntI* $\mathcal{T}_G$*-equivalent-def fst-conv*)
then show *?thesis* **using** *Eq*
by (*auto simp*: *formula-spec-def formula-relevant-def liftO1-def* $\mathcal{T}_G$*-equivalent-def simplify-reg-def RR2-spec-def*
split: *if-splits intro*!: *exI*[*of* - λz. *if z* = *x then* - *else* -])
next
note [*intro*!] = *RR2-to-RRn-spec*[*of* - ($\lambda\alpha$. (α *x*, α *y*)) ‘ *Collect P* **for** *P*, *unfolded image-comp*,
unfolded image-Collect comp-def numeral-2-eq-2 prod.simps]
case *Lt* **then show** *?thesis* **by** (*fastforce simp*: *formula-spec-def formula-relevant-def RR2-spec-def* $\mathcal{T}_G$*-equivalent-def*
split: *if-splits intro*!: *exI*[*of* - λz. *if z* = *x then* - *else* -])
next
note [*intro*!] = *RR2-to-RRn-spec*[*of* - *prod.swap* ‘ ($\lambda\alpha$. (α *x*, α *y*)) ‘ *Collect P* **for** *P*, *OF swap-RR2-spec*,
unfolded image-comp, *unfolded image-Collect comp-def numeral-2-eq-2 prod.simps fmap-funs-reg-comp case-swap*]
case *Gt* **then show** *?thesis*
by (*fastforce simp*: *formula-spec-def formula-relevant-def RR2-spec-def* $\mathcal{T}_G$*-equivalent-def*
split: *if-splits intro*!: *exI*[*of* - λz. *if z* = *x then* - *else* -])
qed
next
case (*IAnd ls*)
have [*simp*]: (*fm*, *vs*, *ta*) $\in$ (!) *infs* ‘ *set ls* $\Longrightarrow$ *formula-spec ?F ?Rs vs ta fm* **for** *fm vs ta*
using *infs IAnd* **by** *auto*
show *?case* **using** *IAnd fit-rrns*[*OF assms*(*3*), *of map* ((!) *infs*) *ls*, *OF* - *prod.collapse*]
by (*force split*: *prod.splits intro*!: *fand-rrn*[*OF assms*(*1*)])
next
case (*IOr ls*)
have [*simp*]: (*fm*, *vs*, *ta*) $\in$ (!) *infs* ‘ *set ls* $\Longrightarrow$ *formula-spec ?F ?Rs vs ta fm* **for** *fm vs ta*
using *infs IOr* **by** *auto*
show *?case* **using** *IOr fit-rrns*[*OF assms*(*3*), *of map* ((!) *infs*) *ls*, *OF* - *prod.collapse*]
by (*force split*: *prod.splits intro*!: *for-rrn*)
next
case (*INot l'*)
obtain *fm vs' ta* **where** [*simp*]: *infs* ! *l'* = (*fm*, *vs'*, *ta*) **by** (*cases infs* ! *l'*) *auto*
then have *spec*: *formula-spec ?F ?Rs vs ta fm* **using** *infs*[*OF nth-mem*, *of l'*] *INot*
by *auto*
have *rel*: *formula-relevant* (*fset* $\mathcal{F}$) (*map fset Rs*) *vs* (*FNot fm*) **using** *spec*
unfolding *formula-spec-def formula-relevant-def*
by (*metis* (*no-types*, *lifting*) *eval-formula.simps*(*5*))

have *vs*: *sorted vs distinct vs* **using** *spec* **by** (*auto simp*: *formula-spec-def*)
{fix *xs* **assume** *ass*: $\forall \alpha$. *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\longrightarrow$ *xs* = *map* α *vs* $\longrightarrow \neg$ *eval-formula* (*fset* $\mathcal{F}$) (*map fset Rs*) α *fm*
length xs = *length vs set xs* $\subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$)
from *sig* **obtain** *s* **where** *mem*: $s \in \mathcal{T}_G$ (*fset* $\mathcal{F}$) **by** *blast*
let *?g* = λ *i*. *find* (λ *p*. *fst p* = *i*) (*zip vs* [*0* ..< *length vs*])
let *?f* = λ *i*. *if ?g i* = *None then s else xs* ! *snd* (*the* (*?g i*))
from *vs*(*1*) **have** $*$: *sorted* (*zip vs* [*0* ..< *length vs*])
by (*induct vs rule*: *rev-induct*) (*auto simp*: *sorted-append elim*!: *in-set-zipE intro*!: *sorted-append-bigger*)
have *i* < *length vs* $\Longrightarrow$ *?g* (*vs* ! *i*) = *Some* (*vs* ! *i*, *i*) **for** *i* **using** *vs*(*2*)
by (*induct vs rule*: *rev-induct*) (*auto simp*: *nth-append find-append find-Some-iff nth-eq-iff-index-eq split*!: *if-splits*)
then have *map ?f vs* = *xs* **using** *vs*(*2*) *ass*(*2*)
by (*intro nth-equalityI*) (*auto simp*: *find-None-iff set-zip*)
moreover have *range ?f* $\subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) **using** *ass*(*2*, *3*) *mem*
using *find-SomeD*(*2*) *set-zip-rightD* **by** *auto fastforce*
ultimately have $\exists \alpha$. *xs* = *map* α *vs* $\wedge$ *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\wedge \neg$ *eval-formula* (*fset* $\mathcal{F}$) (*map fset Rs*) α *fm* **using** *ass*(*1*)
by (*intro exI*[*of* - *?f*]) *auto***}**
then have $*$: {*ts*. *length ts* = *length vs* $\wedge$ *set ts* $\subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$)} −
{*map* α *vs* |α. *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\wedge$ *eval-formula* (*fset* $\mathcal{F}$) (*map fset Rs*) α *fm*} =
{*map* α *vs* |α. *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\wedge \neg$ *eval-formula* (*fset* $\mathcal{F}$) (*map fset Rs*) α *fm*}
apply *auto*
apply *force*
using *formula-relevantD*[*OF rel*] **unfolding** *eval-formula.simps*
by (*meson map-ext*)
have *RRn-spec* (*length vs*) (*difference-reg* (*true-RRn* $\mathcal{F}$ (*length vs*)) *ta*)
{*map* α *vs* |α. *range* $\alpha \subseteq \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\wedge \neg$ *eval-formula* (*fset* $\mathcal{F}$) (*map fset Rs*) α *fm*}
using *RRn-difference*[*OF true-RRn-spec*[*of length vs* $\mathcal{F}$] *formula-spec-RRn-spec*[*OF spec*]]
unfolding $*$ **by** *simp*
then show *?case* **using** *INot spec rel*
by (*auto split*: *prod.splits simp*: *formula-spec-def*)
next
case (*IExists l'*)
obtain *fm vs ta* **where** [*simp*]: *infs* ! *l'* = (*fm*, *vs*, *ta*) **by** (*cases infs* ! *l'*) *auto*
then have *spec*: *formula-spec ?F ?Rs vs ta fm* **using** *infs*[*OF nth-mem*, *of l'*] *IExists*
by *auto*
show *?case*
proof (*cases length vs* = *0*)
case *True*
then show *?thesis* **using** *IExists spec*
apply (*auto simp*: *formula-spec-def*)
subgoal apply (*auto simp*: *formula-relevant-def*)

```
      apply (meson shift-rangeI)
      done
    subgoal apply (auto simp: RRn-spec-def image-iff)
      apply (meson eval-formula-shift-right-eval(1) rangeI shift-right-rangeI sub-
setD)
      apply (meson shift-rangeI)
      done
    done
 next
  case False note len = this
  then have *[simp]: vs ≠ [] by (cases vs) auto
  show ?thesis
  proof (cases reg-empty ta)
    case True
    then show ?thesis using IExists spec formula-spec-empty[OF - spec]
      by (auto simp: 𝒯_G-equivalent-def comp-def formula-spec-def
                     shift-rangeI RRn-spec-def image-iff ℒ-epmty
          intro!: trivial-formula-relevant)
  next
    case False
    then have nt-empty [simp]: ℒ ta ≠ {} by auto
    show ?thesis
    proof (cases 0 ∉ set vs)
      case True
      then have ta: ta = A' using spec IExists
        by (auto simp: formula-spec-def)
       from True have relv: formula-relevant ?F ?Rs (map (λx. x − Suc 0) vs)
(FExists fm)
        using spec IExists
        by (intro ExistsI-upd-brujin-no-0) (auto simp: formula-spec-def)
       then show ?thesis using True spec IExists nt-rel-0-trivial-shift[OF True,
of ?F ?Rs ]
        by (auto simp: formula-spec-def 𝒯_G-equivalent-def comp-def
              elim!: formula-relevantD
              intro!: pres-distinct-not-0-list-dec pres-sorted-dec)
    next
      case False
      then have rel-0: 0 ∈ set vs by simp
      show ?thesis
      proof (cases 1 = length vs)
        case True
        then have [simp]: vs = [0] using rel-0 by (induct vs) auto
        {fix t assume 0 |∈| ta-der (TA {|[] [] → 0|} {||}) (term-of-gterm t)
          then have t = GFun [] [] by (cases t) auto}
        then have [simp]: ℒ (Reg {|0|} (TA {|TA-rule [] [] 0|} {||})) = {GFun []
[]}
          by (auto simp: ℒ-def gta-der-def gta-lang-def)
        have [simp]: GFun [] [] = gencode []
          by (auto simp: gencode-def gunions-def)
```

```
        show ?thesis using IExists spec nt-empty
          by (auto simp: formula-spec-def RRn-spec-true-RRn RRn-spec-def for-
mula-relevant-0-FExists image-iff)
         (meson eval-formula-shift-right-eval(1) in-mono rangeI shift-right-rangeI)
       next
        case False
        from False show ?thesis using spec IExists rel-0 nt-empty
         using rrn-drop-fst-lang[OF formula-spec-RRn-spec[OF spec]]
         by (auto simp: formula-spec-def Suc-lessI simp flip: drop-upd-bruijn-set
                  split: prod.splits
                         intro: upd-bruijn-pres-sorted upd-bruijn-pres-distinct Ex-
istsI-upd-brujin-0)
       qed
     qed
   qed
  qed
next
  case (IRename l' vs)
  then show ?case by simp
next
  case (INNFPlus l')
  show ?case using infs[OF nth-mem, of l'] INNFPlus
    apply (auto simp: formula-spec-def formula-relevant-def eval-formula-conv)
   apply (simp-all only: check-equivalence-by-nnf-sortedlist-aci[of form-of-formula
(fst (infs ! l')) form-of-formula fm])
    done
next
  case (IRepl eq pos l')
  then show ?case by simp
}
qed


end
theory FOR-Check-Impl
  imports FOR-Check
   Regular-Tree-Relations.Regular-Relation-Impl
   NF-Impl
begin
```

14 Inference checking implementation

definition *ftrancl-eps-free-closures* $\mathcal{A}$ = *eps-free-automata* (*eps* $\mathcal{A}$) $\mathcal{A}$
abbreviation *ftrancl-eps-free-reg* $\mathcal{A}$ ≡ *Reg* (*fin* $\mathcal{A}$) (*ftrancl-eps-free-closures* (*ta* $\mathcal{A}$))

lemma *ftrancl-eps-free-ta-derI*:
  (*eps* $\mathcal{A}$)$|^+|$ = *eps* $\mathcal{A}$ ⟹ *ta-der* (*ftrancl-eps-free-closures* $\mathcal{A}$) (*term-of-gterm t*) = *ta-der* $\mathcal{A}$ (*term-of-gterm t*)

using *eps-free*[*of* $\mathcal{A}$] *ta-res-eps-free*[*of* $\mathcal{A}$]
by (*auto simp add*: *ftrancl-eps-free-closures-def*)

lemma $\mathcal{L}$*-ftrancl-eps-free-closuresI*:
($eps\ (ta\ \mathcal{A}))|^+| = eps\ (ta\ \mathcal{A}) \Longrightarrow \mathcal{L}\ (ftrancl\text{-}eps\text{-}free\text{-}reg\ \mathcal{A}) = \mathcal{L}\ \mathcal{A}$
using *ftrancl-eps-free-ta-derI*[*of ta* $\mathcal{A}$]
unfolding $\mathcal{L}$*-def* **by** (*auto simp*: *gta-lang-def gta-der-def*)

definition *root-step* $R\ \mathcal{F} \equiv$ (*let* (*TA1*, *TA2*) = *agtt-grrstep* $R\ \mathcal{F}$ *in*
(*ftrancl-eps-free-closures TA1*, *TA2*))

definition *AGTT-trancl-eps-free* :: $('q, 'f)\ gtt \Rightarrow ('q + 'q, 'f)\ gtt$ **where**
AGTT-trancl-eps-free $\mathcal{G}$ = (*let* ($\mathcal{A}$, $\mathcal{B}$) = *AGTT-trancl* $\mathcal{G}$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$, $\mathcal{B}$))

definition *GTT-trancl-eps-free* **where**
GTT-trancl-eps-free $\mathcal{G}$ = (*let* ($\mathcal{A}$, $\mathcal{B}$) = *GTT-trancl* $\mathcal{G}$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$,
ftrancl-eps-free-closures $\mathcal{B}$))

definition *AGTT-comp-eps-free* **where**
AGTT-comp-eps-free $\mathcal{G}_1\ \mathcal{G}_2$ = (*let* ($\mathcal{A}$, $\mathcal{B}$) = *AGTT-comp'* $\mathcal{G}_1\ \mathcal{G}_2$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$, $\mathcal{B}$))

definition *GTT-comp-eps-free* **where**
GTT-comp-eps-free $\mathcal{G}_1\ \mathcal{G}_2$ =(*let* ($\mathcal{A}$, $\mathcal{B}$) = *GTT-comp'* $\mathcal{G}_1\ \mathcal{G}_2$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$, *ftrancl-eps-free-closures* $\mathcal{B}$))

lemma *eps-free-relable* [*simp*]:
is-gtt-eps-free (*relabel-gtt* $\mathcal{G}$) = *is-gtt-eps-free* $\mathcal{G}$
by (*auto simp*: *is-gtt-eps-free-def relabel-gtt-def fmap-states-gtt-def fmap-states-ta-def*)

lemma *eps-free-prod-swap*:
is-gtt-eps-free ($\mathcal{A}$, $\mathcal{B}$) $\Longrightarrow$ *is-gtt-eps-free* ($\mathcal{B}$, $\mathcal{A}$)
by (*auto simp*: *is-gtt-eps-free-def*)

lemma *eps-free-root-step*:
is-gtt-eps-free (*root-step* $R\ \mathcal{F}$)
by (*auto simp add*: *case-prod-beta is-gtt-eps-free-def root-step-def pair-at-to-agtt'-def ftrancl-eps-free-closures-def*)

lemma *eps-free-AGTT-trancl-eps-free*:
is-gtt-eps-free $\mathcal{G}$ $\Longrightarrow$ *is-gtt-eps-free* (*AGTT-trancl-eps-free* $\mathcal{G}$)
by (*auto simp*: *case-prod-beta is-gtt-eps-free-def AGTT-trancl-def Let-def AGTT-trancl-eps-free-def ftrancl-eps-free-closures-def*)

lemma *eps-free-GTT-trancl-eps-free*:
is-gtt-eps-free $\mathcal{G}$ $\Longrightarrow$ *is-gtt-eps-free* (*GTT-trancl-eps-free* $\mathcal{G}$)

by (*auto simp*: *case-prod-beta is-gtt-eps-free-def GTT-trancl-eps-free-def ftrancl-eps-free-closures-def*)

lemma *eps-free-AGTT-comp-eps-free*:
is-gtt-eps-free $\mathcal{G}_2 \Longrightarrow$ *is-gtt-eps-free* (*AGTT-comp-eps-free* $\mathcal{G}_1$ $\mathcal{G}_2$)
by (*auto simp*: *case-prod-beta is-gtt-eps-free-def AGTT-comp-eps-free-def ftrancl-eps-free-closures-def AGTT-comp-def fmap-states-gtt-def fmap-states-ta-def*)

lemma *eps-free-GTT-comp-eps-free*:
is-gtt-eps-free (*GTT-comp-eps-free* $\mathcal{G}_1$ $\mathcal{G}_2$)
by (*auto simp*: *case-prod-beta is-gtt-eps-free-def GTT-comp-eps-free-def ftrancl-eps-free-closures-def*)

lemmas *eps-free-const* =
eps-free-prod-swap
eps-free-root-step
eps-free-AGTT-trancl-eps-free
eps-free-GTT-trancl-eps-free
eps-free-AGTT-comp-eps-free
eps-free-GTT-comp-eps-free

lemma *agtt-lang-derI*:
assumes $\bigwedge$ *t. ta-der* (*fst* $\mathcal{A}$) (*term-of-gterm t*) = *ta-der* (*fst* $\mathcal{B}$) (*term-of-gterm t*)
and $\bigwedge$ *t. ta-der* (*snd* $\mathcal{A}$) (*term-of-gterm t*) = *ta-der* (*snd* $\mathcal{B}$) (*term-of-gterm t*)
shows *agtt-lang* $\mathcal{A}$ = *agtt-lang* $\mathcal{B}$ **using** *assms*
by (*auto simp*: *agtt-lang-def gta-der-def*)

lemma *agtt-lang-root-step-conv*:
agtt-lang (*root-step R* $\mathcal{F}$) = *agtt-lang* (*agtt-grrstep R* $\mathcal{F}$)
using *ftrancl-eps-free-ta-derI*[*OF agtt-grrstep-eps-trancl*(*1*), *of R* $\mathcal{F}$]
by (*auto simp*: *case-prod-beta root-step-def intro*!: *agtt-lang-derI*)

lemma *agtt-lang-AGTT-trancl-eps-free-conv*:
assumes *is-gtt-eps-free* $\mathcal{G}$
shows *agtt-lang* (*AGTT-trancl-eps-free* $\mathcal{G}$) = *agtt-lang* (*AGTT-trancl* $\mathcal{G}$)
proof −
let *?eps* = *eps* (*fst* (*AGTT-trancl* $\mathcal{G}$))
have *?eps* |*O*| *?eps* = {||} **using** *assms*
by (*auto simp*: *AGTT-trancl-def is-gtt-eps-free-def Let-def fmap-states-ta-def*)
from *ftrancl-eps-free-ta-derI*[*OF frelcomp-empty-ftrancl-simp*[*OF this*]]
show *?thesis*
by (*auto simp*: *case-prod-beta AGTT-trancl-eps-free-def intro*!: *agtt-lang-derI*)
qed

lemma *agtt-lang-GTT-trancl-eps-free-conv*:
assumes *is-gtt-eps-free* $\mathcal{G}$
shows *agtt-lang* (*GTT-trancl-eps-free* $\mathcal{G}$) = *agtt-lang* (*GTT-trancl* $\mathcal{G}$)
proof −
have (*eps* (*fst* (*GTT-trancl* $\mathcal{G}$)))|$^+$| = *eps* (*fst* (*GTT-trancl* $\mathcal{G}$))

(*eps* (*snd* (*GTT-trancl* $\mathcal{G}$)))|$^+$| = *eps* (*snd* (*GTT-trancl* $\mathcal{G}$)) **using** *assms*
by (*auto simp*: *GTT-trancl-def Let-def is-gtt-eps-free-def* Δ*-trancl-inv*)
from *ftrancl-eps-free-ta-derI*[*OF this*(*1*)] *ftrancl-eps-free-ta-derI*[*OF this*(*2*)]
show *?thesis*
by (*auto simp*: *case-prod-beta GTT-trancl-eps-free-def intro*!: *agtt-lang-derI*)
qed

lemma *agtt-lang-AGTT-comp-eps-free-conv*:
assumes *is-gtt-eps-free* $\mathcal{G}_1$ *is-gtt-eps-free* $\mathcal{G}_2$
shows *agtt-lang* (*AGTT-comp-eps-free* $\mathcal{G}_1$ $\mathcal{G}_2$) = *agtt-lang* (*AGTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)
proof −
have (*eps* (*fst* (*AGTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)))|$^+$| = *eps* (*fst* (*AGTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)) **using** *assms*
by (*auto simp*: *is-gtt-eps-free-def fmap-states-gtt-def fmap-states-ta-def case-prod-beta AGTT-comp-def gtt-interface-def* $\mathcal{Q}$*-def intro*!: *frelcomp-empty-ftrancl-simp*)
from *ftrancl-eps-free-ta-derI*[*OF this*] **show** *?thesis*
by (*auto simp*: *case-prod-beta AGTT-comp-eps-free-def intro*!: *agtt-lang-derI*)
qed

lemma *agtt-lang-GTT-comp-eps-free-conv*:
assumes *is-gtt-eps-free* $\mathcal{G}_1$ *is-gtt-eps-free* $\mathcal{G}_2$
shows *agtt-lang* (*GTT-comp-eps-free* $\mathcal{G}_1$ $\mathcal{G}_2$) = *agtt-lang* (*GTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)
proof −
have (*eps* (*fst* (*GTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)))|$^+$| = *eps* (*fst* (*GTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$))
(*eps* (*snd* (*GTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)))|$^+$| = *eps* (*snd* (*GTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$)) **using** *assms*
by (*auto simp*: *is-gtt-eps-free-def fmap-states-gtt-def fmap-states-ta-def* Δ_ε*-fmember case-prod-beta GTT-comp-def gtt-interface-def* $\mathcal{Q}$*-def dest*!: *ground-ta-der-statesD intro*!: *frelcomp-empty-ftrancl-simp*)
from *ftrancl-eps-free-ta-derI*[*OF this*(*1*)] *ftrancl-eps-free-ta-derI*[*OF this*(*2*)]
show *?thesis*
by (*auto simp*: *case-prod-beta GTT-comp-eps-free-def intro*!: *agtt-lang-derI*)
qed

fun *gtt-of-gtt-rel-impl* :: ('*f* × *nat*) *fset* ⇒ ('*f* :: *linorder*, '*v*) *fin-trs list* ⇒ *ftrs gtt-rel* ⇒ (*nat*, '*f*) *gtt option* **where**
gtt-of-gtt-rel-impl $\mathcal{F}$ *Rs* (*ARoot is*) = *liftO1* (λ*R*. *relabel-gtt* (*root-step R* $\mathcal{F}$)) (*is-to-trs' Rs is*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*GInv g*) = *liftO1 prod.swap* (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*AUnion g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-union' g1 g2*)) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*ATrancl g*) = *liftO1* (*relabel-gtt* ∘ *AGTT-trancl-eps-free*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*GTrancl g*) = *liftO1 GTT-trancl-eps-free* (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*AComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-comp-eps-free g1 g2*)) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*GComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*GTT-comp-eps-free g1 g2*)) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)

lemma *gtt-of-gtt-rel-impl-is-gtt-eps-free*:
gtt-of-gtt-rel-impl $\mathcal{F}$ *Rs g* = *Some g'* $\Longrightarrow$ *is-gtt-eps-free g'*
proof (*induct g arbitrary*: *g'*)
case (*AUnion g1 g2*)
then show *?case*
by (*auto simp*: *is-gtt-eps-free-def AGTT-union-def fmap-states-gtt-def fmap-states-ta-def ta-union-def relabel-gtt-def*)
qed (*auto simp*: *eps-free-const*)

lemma *gtt-of-gtt-rel-impl-gtt-of-gtt-rel*:
gtt-of-gtt-rel-impl $\mathcal{F}$ *Rs g* $\neq$ *None* $\longleftrightarrow$ *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g* $\neq$ *None* (**is** *?Ls* $\longleftrightarrow$ *?Rs*)
proof −
have *?Ls* $\Longrightarrow$ *?Rs* **by** (*induct g*) *auto*
moreover have *?Rs* $\Longrightarrow$ *?Ls* **by** (*induct g*) *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *gtt-of-gtt-rel-impl-sound*:
gtt-of-gtt-rel-impl $\mathcal{F}$ *Rs g* = *Some g'* $\Longrightarrow$ *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g* = *Some g''* $\Longrightarrow$ *agtt-lang g'* = *agtt-lang g''*
proof (*induct g arbitrary*: *g' g''*)
case (*ARoot x*)
then show *?case* **by** (*simp add*: *agtt-lang-root-step-conv*)
next
case (*GInv g*)
then have *agtt-lang* (*prod.swap g'*) = *agtt-lang* (*prod.swap g''*) **by** *auto*
then show *?case*
by (*metis converse-agtt-lang converse-converse*)
next
case (*AUnion g1 g2*)
then show *?case*
by *simp* (*metis AGTT-union'-sound option.sel*)
next
case (*ATrancl g*)
then show *?case*
using *agtt-lang-AGTT-trancl-eps-free-conv*[*OF gtt-of-gtt-rel-impl-is-gtt-eps-free*, *of* $\mathcal{F}$ *Rs g*]
by *simp* (*metis AGTT-trancl-sound option.sel*)
next
case (*GTrancl g*)
then show *?case*
using *agtt-lang-GTT-trancl-eps-free-conv*[*OF gtt-of-gtt-rel-impl-is-gtt-eps-free*, *of* $\mathcal{F}$ *Rs g*]
by *simp* (*metis GTT-trancl-alang option.sel*)
next
case (*AComp g1 g2*)
then show *?case*

using *agtt-lang-AGTT-comp-eps-free-conv*[*OF gtt-of-gtt-rel-impl-is-gtt-eps-free*,
of $\mathcal{F}$ *Rs g*
the (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) *the* (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)]
by *simp* (*metis AGTT-comp'-sound agtt-lang-AGTT-comp-eps-free-conv gtt-of-gtt-rel-impl-is-gtt-eps-free
option.sel*)
next
case (*GComp g1 g2*)
then show *?case*
using *agtt-lang-GTT-comp-eps-free-conv*[*OF gtt-of-gtt-rel-impl-is-gtt-eps-free*,
of $\mathcal{F}$ *Rs g*
the (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) *the* (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)]
by *simp* (*metis agtt-lang-GTT-comp-eps-free-conv gtt-comp'-alang gtt-of-gtt-rel-impl-is-gtt-eps-free
option.sel*)
qed

lemma $\mathcal{L}$*-eps-free-nhole-ctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*ftrancl-eps-free-reg* (*nhole-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = $\mathcal{L}$ (*nhole-ctxt-closure-reg*
$\mathcal{F}$ $\mathcal{A}$)
proof −
have *eps* (*ta* (*nhole-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) |*O*| *eps* (*ta* (*nhole-ctxt-closure-reg* $\mathcal{F}$
$\mathcal{A}$)) = {||} **using** *assms*
by (*auto simp*: *nhole-ctxt-closure-reg-def gen-nhole-ctxt-closure-reg-def*
gen-nhole-ctxt-closure-automaton-def ta-union-def reflcl-over-nhole-ctxt-ta-def
fmap-states-reg-def is-ta-eps-free-def fmap-states-ta-def reg-Restr-Q_f*-def*)
from *frelcomp-empty-ftrancl-simp*[*OF this*] **show** *?thesis*
by (*intro* $\mathcal{L}$*-ftrancl-eps-free-closuresI*) *simp*
qed

lemma $\mathcal{L}$*-eps-free-ctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*ftrancl-eps-free-reg* (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)
proof −
have *eps* (*ta* (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) |*O*| *eps* (*ta* (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = {||}
using *assms*
by (*auto simp*: *ctxt-closure-reg-def gen-ctxt-closure-reg-def Let-def*
gen-ctxt-closure-automaton-def ta-union-def reflcl-over-single-ta-def
fmap-states-reg-def is-ta-eps-free-def fmap-states-ta-def reg-Restr-Q_f*-def*)
from *frelcomp-empty-ftrancl-simp*[*OF this*] **show** *?thesis*
by (*intro* $\mathcal{L}$*-ftrancl-eps-free-closuresI*) *simp*
qed

lemma $\mathcal{L}$*-eps-free-parallel-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*ftrancl-eps-free-reg* (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = $\mathcal{L}$ (*parallel-closure-reg*
$\mathcal{F}$ $\mathcal{A}$)
proof −
have *eps* (*ta* (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) |*O*| *eps* (*ta* (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$))

$= \{||\}$ **using** *assms*
by (*auto simp*: *parallel-closure-reg-def gen-parallel-closure-automaton-def Let-def ta-union-def*
refl-over-states-ta-def fmap-states-reg-def is-ta-eps-free-def fmap-states-ta-def reg-Restr-Q_f-def)
from *frelcomp-empty-ftrancl-simp*[*OF this*] **show** *?thesis*
by (*intro $\mathcal{L}$-ftrancl-eps-free-closuresI*) *simp*
qed

abbreviation *eps-free-reg' S R* $\equiv$ *Reg* (*fin R*) (*eps-free-automata S* (*ta R*))

definition *eps-free-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ *in*
eps-free-reg' ((λ *p*. (*fst p*, *Inr cl-state*)) |`| (*eps* (*ta* $\mathcal{B}$)) |∪| *eps* (*ta* $\mathcal{B}$)) $\mathcal{B}$)

definition *eps-free-nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$ *in*
eps-free-reg' ((λ *p*. (*fst p*, *Inl* (*Inr cl-state*))) |`| (*eps* (*ta* $\mathcal{B}$)) |∪| *eps* (*ta* $\mathcal{B}$)) $\mathcal{B}$)

definition *eps-free-nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ *in*
eps-free-reg' ((λ *p*. (*fst p*, (*Inr cl-state*))) |`| (*eps* (*ta* $\mathcal{B}$)) |∪| *eps* (*ta* $\mathcal{B}$)) $\mathcal{B}$)

lemma *$\mathcal{L}$-eps-free-reg'I*:
(*eps* (*ta* $\mathcal{A}$))$|^+|$ = S $\Longrightarrow$ $\mathcal{L}$ (*eps-free-reg' S* $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$
by (*auto simp*: *$\mathcal{L}$-def gta-lang-def gta-der-def ta-res-eps-free simp flip*: *eps-free*)

lemma *$\mathcal{L}$-eps-free-mctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*eps-free-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) **using** *assms*
unfolding *eps-free-mctxt-closure-reg-def Let-def*
apply (*intro $\mathcal{L}$-eps-free-reg'I*)
apply (*auto simp*: *comp-def mctxt-closure-reg-def is-ta-eps-free-def Let-def*
gen-nhole-mctxt-closure-automaton-def reflcl-over-nhole-mctxt-ta-def ta-union-def
reflcl-over-nhole-ctxt-ta-def gen-mctxt-closure-reg-def reg-Restr-Q_f-def
fmap-states-reg-def fmap-states-ta-def dest: *ftranclD ftranclD2*)
by (*meson fimageI finsert-iff finterI fr-into-trancl ftrancl-into-trancl*)

lemma *$\mathcal{L}$-eps-free-nhole-mctxt-reflcl-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*eps-free-nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$)
using *assms*
unfolding *eps-free-nhole-mctxt-reflcl-reg-def Let-def*
apply (*intro $\mathcal{L}$-eps-free-reg'I*)
apply (*auto simp*: *comp-def nhole-mctxt-reflcl-reg-def is-ta-eps-free-def Let-def*
nhole-mctxt-closure-reg-def gen-nhole-mctxt-closure-reg-def reg-union-def ta-union-def
gen-nhole-mctxt-closure-automaton-def reflcl-over-nhole-mctxt-ta-def
reflcl-over-nhole-ctxt-ta-def reg-Restr-Q_f-def fmap-states-reg-def fmap-states-ta-def

dest: *ftranclD ftranclD2*)
by (*meson fimageI finsert-iff finterI fr-into-trancl ftrancl-into-trancl*)

lemma *$\mathcal{L}$-eps-free-nhole-mctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*eps-free-nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) **using** *assms*
unfolding *eps-free-nhole-mctxt-closure-reg-def Let-def*
apply (*intro $\mathcal{L}$-eps-free-reg'I*)
apply (*auto simp*: *comp-def nhole-mctxt-closure-reg-def is-ta-eps-free-def Let-def gen-nhole-mctxt-closure-reg-def reg-Restr-Q_f-def fmap-states-reg-def fmap-states-ta-def gen-nhole-mctxt-closure-automaton-def reflcl-over-nhole-mctxt-ta-def ta-union-def reflcl-over-nhole-ctxt-ta-def dest*: *ftranclD ftranclD2*)
by (*meson fimageI finsert-iff finterI fr-into-trancl ftrancl-into-trancl*)

fun *rr1-of-rr1-rel-impl* :: ('*f* × *nat*) *fset* ⇒ ('*f* :: *linorder*, '*v*) *fin-trs list* ⇒ *ftrs rr1-rel* ⇒ (*nat*, '*f*) *reg option*
and *rr2-of-rr2-rel-impl* :: ('*f* × *nat*) *fset* ⇒ ('*f*, '*v*) *fin-trs list* ⇒ *ftrs rr2-rel* ⇒ (*nat*, '*f option* × '*f option*) *reg option* **where**
rr1-of-rr1-rel-impl $\mathcal{F}$ *Rs R1Terms* = *Some* (*relabel-reg* (*term-reg* $\mathcal{F}$))
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1NF is*) = *liftO1* (λ*R*. (*simplify-reg* (*nf-reg* (*fst* |‘| *R*) $\mathcal{F}$))) (*is-to-trs' Rs is*)
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1Inf r*) = *liftO1* (λ*R*.
let $\mathcal{A}$ = *trim-reg R in*
simplify-reg (*proj-1-reg* (*Inf-reg-impl* $\mathcal{A}$))
) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r*)
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1Proj i r*) = (*case i of 0* ⇒
liftO1 (*trim-reg* ∘ *proj-1-reg*) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r*)
| - ⇒ *liftO1* (*trim-reg* ∘ *proj-2-reg*) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r*))
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1Union s1 s2*) =
liftO2 (λ *x y*. *relabel-reg* (*reg-union x y*)) (*rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs s2*)
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1Inter s1 s2*) =
liftO2 (λ *x y*. *simplify-reg* (*reg-intersect x y*)) (*rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs s2*)
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1Diff s1 s2*) = *liftO2* (λ *x y*. *relabel-reg* (*trim-reg* (*difference-reg x y*))) (*rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs s2*)

| *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs* (*R2GTT-Rel g w x*) =
(*case w of PRoot* ⇒
(*case x of ESingle* ⇒ *liftO1* (*simplify-reg* ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *EParallel* ⇒ *liftO1* (*simplify-reg* ∘ *reflcl-reg* (*lift-sig-RR2* |‘| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *EStrictParallel* ⇒ *liftO1* (*simplify-reg* ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*))
| *PNonRoot* ⇒
(*case x of ESingle* ⇒ *liftO1* (*simplify-reg* ∘ *ftrancl-eps-free-reg* ∘ *nhole-ctxt-closure-reg* (*lift-sig-RR2* |‘| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)

```
      | EParallel ⇒ liftO1 (simplify-reg ∘ eps-free-nhole-mctxt-reflcl-reg (lift-sig-RR2
|`| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl F Rs g)
        | EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-nhole-mctxt-closure-reg
(lift-sig-RR2 |`| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl F Rs g))
      | PAny ⇒
    (case x of ESingle ⇒ liftO1 (simplify-reg ∘ ftrancl-eps-free-reg ∘ ctxt-closure-reg
(lift-sig-RR2 |`| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl F Rs g)
       | EParallel ⇒ liftO1 (simplify-reg ∘ ftrancl-eps-free-reg ∘ parallel-closure-reg
(lift-sig-RR2 |`| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl F Rs g)
      | EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-mctxt-closure-reg (lift-sig-RR2
|`| F) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl F Rs g)))
| rr2-of-rr2-rel-impl F Rs (R2Diag s) =
    liftO1 (λ x. fmap-funs-reg (λf. (Some f, Some f)) x) (rr1-of-rr1-rel-impl F Rs
s)
| rr2-of-rr2-rel-impl F Rs (R2Prod s1 s2) =
    liftO2 (λ x y. simplify-reg (pair-automaton-reg x y)) (rr1-of-rr1-rel-impl F Rs
s1) (rr1-of-rr1-rel-impl F Rs s2)
| rr2-of-rr2-rel-impl F Rs (R2Inv r) = liftO1 (fmap-funs-reg prod.swap) (rr2-of-rr2-rel-impl
F Rs r)
| rr2-of-rr2-rel-impl F Rs (R2Union r1 r2) =
   liftO2 (λ x y. relabel-reg (reg-union x y)) (rr2-of-rr2-rel-impl F Rs r1) (rr2-of-rr2-rel-impl
F Rs r2)
| rr2-of-rr2-rel-impl F Rs (R2Inter r1 r2) =
     liftO2 (λ x y. simplify-reg (reg-intersect x y)) (rr2-of-rr2-rel-impl F Rs r1)
(rr2-of-rr2-rel-impl F Rs r2)
| rr2-of-rr2-rel-impl F Rs (R2Diff r1 r2) = liftO2 (λ x y. simplify-reg (difference-reg
x y)) (rr2-of-rr2-rel-impl F Rs r1) (rr2-of-rr2-rel-impl F Rs r2)
| rr2-of-rr2-rel-impl F Rs (R2Comp r1 r2) = liftO2 (λ x y. simplify-reg (rr2-compositon
F x y))
    (rr2-of-rr2-rel-impl F Rs r1) (rr2-of-rr2-rel-impl F Rs r2)
```

lemmas *ta-simp-unfold = simplify-reg-def relabel-reg-def trim-reg-def relabel-ta-def term-reg-def*

lemma *is-ta-eps-free-trim-reg* [*intro*!]:
  is-ta-eps-free (ta R) ⟹ is-ta-eps-free (ta (trim-reg R))
  by (*simp add*: *is-ta-eps-free-def trim-reg-def trim-ta-def ta-restrict-def*)

lemma *is-ta-eps-free-relabel-reg* [*intro*!]:
  is-ta-eps-free (ta R) ⟹ is-ta-eps-free (ta (relabel-reg R))
  by (*simp add*: *is-ta-eps-free-def relabel-reg-def relabel-ta-def fmap-states-ta-def*)

lemma *is-ta-eps-free-simplify-reg* [*intro*!]:
  is-ta-eps-free (ta R) ⟹ is-ta-eps-free (ta (simplify-reg R))
 by (*simp add*: *is-ta-eps-free-def ta-simp-unfold fmap-states-ta-def trim-ta-def ta-restrict-def*)

lemma *is-ta-emptyI* [*simp*]:
 is-ta-eps-free (TA R {||}) ⟷ True
  by (*simp add*: *is-ta-eps-free-def*)

lemma *is-ta-empty-trim-reg*:
is-ta-eps-free (*ta A*) ⟹ *eps* (*ta* (*trim-reg A*)) = {||}
by (*auto simp*: *is-ta-eps-free-def trim-reg-def trim-ta-def ta-restrict-def*)

lemma *is-proj-ta-eps-empty*:
is-ta-eps-free (*ta R*) ⟹ *is-ta-eps-free* (*ta* (*proj-1-reg R*))
is-ta-eps-free (*ta R*) ⟹ *is-ta-eps-free* (*ta* (*proj-2-reg R*))
by (*auto simp*: *is-ta-eps-free-def proj-1-reg-def proj-2-reg-def collapse-automaton-reg-def collapse-automaton-def*
fmap-funs-reg-def fmap-funs-ta-def trim-reg-def trim-ta-def ta-restrict-def)

lemma *is-pod-ta-eps-empty*:
is-ta-eps-free (*ta R*) ⟹ *is-ta-eps-free* (*ta L*) ⟹ *is-ta-eps-free* (*ta* (*reg-intersect R L*))
by (*auto simp*: *reg-intersect-def prod-ta-def prod-epsRp-def prod-epsLp-def is-ta-eps-free-def*)

lemma *is-fmap-funs-reg-eps-empty*:
is-ta-eps-free (*ta R*) ⟹ *is-ta-eps-free* (*ta* (*fmap-funs-reg f R*))
by (*auto simp*: *fmap-funs-reg-def fmap-funs-ta-def is-ta-eps-free-def*)

lemma *is-collapse-automaton-reg-eps-empty*:
is-ta-eps-free (*ta R*) ⟹ *is-ta-eps-free* (*ta* (*collapse-automaton-reg R*))
by (*auto simp*: *collapse-automaton-reg-def collapse-automaton-def is-ta-eps-free-def*)

lemma *is-pair-automaton-reg-eps-empty*:
is-ta-eps-free (*ta R*) ⟹ *is-ta-eps-free* (*ta L*) ⟹ *is-ta-eps-free* (*ta* (*pair-automaton-reg R L*))
by (*auto simp*: *pair-automaton-reg-def pair-automaton-def is-ta-eps-free-def*)

lemma *is-reflcl-automaton-eps-free*:
is-ta-eps-free A ⟹ *is-ta-eps-free* (*reflcl-automaton* (*lift-sig-RR2* |`| $\mathcal{F}$) *A*)
by (*auto simp*: *is-ta-eps-free-def reflcl-automaton-def ta-union-def gen-reflcl-automaton-def Let-def fmap-states-ta-def*)

lemma *is-GTT-to-RR2-root-eps-empty*:
is-gtt-eps-free $\mathcal{G}$ ⟹ *is-ta-eps-free* (*GTT-to-RR2-root* $\mathcal{G}$)
by (*auto simp*: *is-gtt-eps-free-def GTT-to-RR2-root-def pair-automaton-def is-ta-eps-free-def*)

lemma *is-term-automata-eps-empty*:
is-ta-eps-free (*ta* (*term-reg* $\mathcal{F}$)) ⟷ *True*
by (*auto simp*: *is-ta-eps-free-def term-reg-def term-automaton-def*)

lemma *is-ta-eps-free-eps-free-automata* [*simp*]:
is-ta-eps-free (*eps-free-automata S R*) ⟷ *True*
by (*auto simp*: *eps-free-automata-def is-ta-eps-free-def*)

lemma *rr2-of-rr2-rel-impl-eps-free*:
shows ∀ *A*. *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs r1* = *Some A* ⟶ *is-ta-eps-free* (*ta A*)
∀ *A*. *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2* = *Some A* ⟶ *is-ta-eps-free* (*ta A*)

proof (*induct r1* **and** *r2*)
case *R1Terms*
then show *?case*
by (*auto simp*: *is-ta-eps-free-def term-automaton-def fmap-states-ta-def ta-simp-unfold*)
next
case (*R1NF x*)
then show *?case*
by (*auto simp*: *nf-reg-def nf-ta-def*)
next
case (*R1Inf x*)
then show *?case*
by (*auto simp*: *Inf-reg-impl-def Let-def Inf-reg-def Inf-automata-def is-ta-empty-trim-reg intro*!: *is-proj-ta-eps-empty*)
next
case (*R1Proj n x2*)
then show *?case*
by (*cases n*) (*auto intro*!: *is-proj-ta-eps-empty*)
next
case (*R1Union x1 x2*)
then show *?case*
by (*simp add*: *reg-union-def ta-union-def fmap-states-ta-def is-ta-eps-free-def relabel-reg-def relabel-ta-def*)
next
case (*R1Inter x1 x2*)
then show *?case*
by (*auto intro*: *is-pod-ta-eps-empty*)
next
case (*R1Diff x1 x2*)
then show *?case*
by (*auto simp*: *difference-reg-def Let-def complement-reg-def ps-reg-def ps-ta-def intro*!: *is-pod-ta-eps-empty*)
next
case (*R2GTT-Rel x1 x2 x3*)
then show *?case*
by (*cases x2*; *cases x3*) (*auto simp*: *GTT-to-RR2-root-reg-def ftrancl-eps-free-closures-def eps-free-nhole-mctxt-closure-reg-def eps-free-nhole-mctxt-reflcl-reg-def Let-def eps-free-mctxt-closure-reg-def reflcl-reg-def dest*: *gtt-of-gtt-rel-impl-is-gtt-eps-free intro*!: *is-GTT-to-RR2-root-eps-empty is-reflcl-automaton-eps-free*)
next
case (*R2Diag x*)
then show *?case* **by** (*auto simp*: *fmap-funs-reg-def fmap-funs-ta-def is-ta-eps-free-def*)
next
case (*R2Prod x1 x2*)
then show *?case* **by** (*auto intro*: *is-pair-automaton-reg-eps-empty*)
next
case (*R2Inv x*)
then show *?case* **by** (*auto simp*: *fmap-funs-reg-def fmap-funs-ta-def is-ta-eps-free-def*)
next

case (*R2Union x1 x2*)
then show *?case* **by** (*simp add*: *reg-union-def ta-union-def fmap-states-ta-def is-ta-eps-free-def relabel-reg-def relabel-ta-def*)
next
case (*R2Inter x1 x2*)
then show *?case* **by** (*auto intro*: *is-pod-ta-eps-empty*)
next
case (*R2Diff x1 x2*)
then show *?case* **by** (*auto simp*: *difference-reg-def Let-def complement-reg-def ps-reg-def ps-ta-def intro*!: *is-pod-ta-eps-empty*)
next
case (*R2Comp x1 x2*)
then show *?case* **by** (*auto simp*: *is-term-automata-eps-empty rr2-compositon-def Let-def*
intro!: *is-pod-ta-eps-empty is-fmap-funs-reg-eps-empty is-collapse-automaton-reg-eps-empty is-pair-automaton-reg-eps-empty*)
qed

lemma *rr-of-rr-rel-impl-complete*:
rr1-of-rr1-rel-impl $\mathcal{F}$ *Rs r1* $\neq$ *None* $\longleftrightarrow$ *rr1-of-rr1-rel* $\mathcal{F}$ *Rs r1* $\neq$ *None*
rr2-of-rr2-rel-impl $\mathcal{F}$ *Rs r2* $\neq$ *None* $\longleftrightarrow$ *rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2* $\neq$ *None*
proof (*induct r1* **and** *r2*)
case (*R1Proj n x2*)
then show *?case* **by** (*cases n*) *auto*
next
case (*R2GTT-Rel x1 n p*)
then show *?case* **using** *gtt-of-gtt-rel-impl-gtt-of-gtt-rel*[*of* $\mathcal{F}$ *Rs*]
by (*cases p*; *cases n*) *auto*
qed *auto*

lemma $\mathcal{Q}$*-fmap-funs-reg* [*simp*]:
$\mathcal{Q}_r$ (*fmap-funs-reg f* $\mathcal{A}$) = $\mathcal{Q}_r$ $\mathcal{A}$
by (*auto simp*: *fmap-funs-reg-def*)

lemma *ta-reachable-fmap-funs-reg* [*simp*]:
ta-reachable (*ta* (*fmap-funs-reg f* $\mathcal{A}$)) = *ta-reachable* (*ta* $\mathcal{A}$)
by (*auto simp*: *fmap-funs-reg-def*)

lemma *collapse-reg-cong*:
$\mathcal{Q}_r$ $\mathcal{A}$ |$\subseteq$| *ta-reachable* (*ta* $\mathcal{A}$) $\Longrightarrow$ $\mathcal{Q}_r$ $\mathcal{B}$ |$\subseteq$| *ta-reachable* (*ta* $\mathcal{B}$) $\Longrightarrow$ $\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$
$\Longrightarrow$ $\mathcal{L}$ (*collapse-automaton-reg* $\mathcal{A}$) = $\mathcal{L}$ (*collapse-automaton-reg* $\mathcal{B}$)
by (*auto simp*: *collapse-automaton-reg-def* $\mathcal{L}$*-def collapse-automaton'*)

lemma $\mathcal{L}$*-fmap-funs-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ $\Longrightarrow$ $\mathcal{L}$ (*fmap-funs-reg h* $\mathcal{A}$) = $\mathcal{L}$ (*fmap-funs-reg h* $\mathcal{B}$)
by (*auto simp*: *fmap-funs-*$\mathcal{L}$)

lemma $\mathcal{L}$*-pair-automaton-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ $\Longrightarrow$ $\mathcal{L}$ $\mathcal{C}$ = $\mathcal{L}$ $\mathcal{D}$ $\Longrightarrow$ $\mathcal{L}$ (*pair-automaton-reg* $\mathcal{A}$ $\mathcal{C}$) = $\mathcal{L}$ (*pair-automaton-reg*

$\mathcal{B}$ $\mathcal{D}$)
by (*auto simp*: *pair-automaton'*)

lemma $\mathcal{L}$*-nhole-ctxt-closure-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{F}$ = $\mathcal{G}$ ⟹ $\mathcal{L}$ (*nhole-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-ctxt-closure-reg* $\mathcal{G}$ $\mathcal{B}$)
by (*auto simp*: *nhole-ctxtcl-lang*)

lemma $\mathcal{L}$*-nhole-mctxt-closure-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{F}$ = $\mathcal{G}$ ⟹ $\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{G}$ $\mathcal{B}$)
by (*auto simp*: *nhole-gmctxt-closure-lang*)

lemma $\mathcal{L}$*-ctxt-closure-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{F}$ = $\mathcal{G}$ ⟹ $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{G}$ $\mathcal{B}$)
by (*auto simp*: *gctxt-closure-lang*)

lemma $\mathcal{L}$*-parallel-closure-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{F}$ = $\mathcal{G}$ ⟹ $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{G}$ $\mathcal{B}$)
by (*auto simp*: *parallelcl-gmctxt-lang*)

lemma $\mathcal{L}$*-mctxt-closure-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{F}$ = $\mathcal{G}$ ⟹ $\mathcal{L}$ (*mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*mctxt-closure-reg* $\mathcal{G}$ $\mathcal{B}$)
by (*auto simp*: *gmctxt-closure-lang*)

lemma $\mathcal{L}$*-nhole-mctxt-reflcl-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{F}$ = $\mathcal{G}$ ⟹ $\mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{G}$ $\mathcal{B}$)
unfolding *nhole-mctxt-reflcl-lang*
by (*intro arg-cong2*[**where** *?f* = (∪)] $\mathcal{L}$*-nhole-mctxt-closure-reg-cong*) *auto*

declare *equalityI*[*rule del*]
declare *fsubsetI*[*rule del*]
lemma $\mathcal{L}$*-proj-1-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{L}$ (*proj-1-reg* $\mathcal{A}$) = $\mathcal{L}$ (*proj-1-reg* $\mathcal{B}$)
by (*auto simp*: *proj-1-reg-def* $\mathcal{L}$*-trim intro*!: *collapse-reg-cong* $\mathcal{L}$*-fmap-funs-reg-cong*)

lemma $\mathcal{L}$*-proj-2-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ ⟹ $\mathcal{L}$ (*proj-2-reg* $\mathcal{A}$) = $\mathcal{L}$ (*proj-2-reg* $\mathcal{B}$)
by (*auto simp*: *proj-2-reg-def* $\mathcal{L}$*-trim intro*!: *collapse-reg-cong* $\mathcal{L}$*-fmap-funs-reg-cong*)

lemma *rr2-of-rr2-rel-impl-sound*:
assumes ∀ *R* ∈ *set Rs*. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| $\mathcal{F}$
shows ⋀ *A B*. *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs r1* = *Some A* ⟹ *rr1-of-rr1-rel* $\mathcal{F}$ *Rs r1* = *Some B* ⟹ $\mathcal{L}$ *A* = $\mathcal{L}$ *B*
⋀ *A B*. *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2* = *Some A* ⟹ *rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2* = *Some B* ⟹ $\mathcal{L}$ *A* = $\mathcal{L}$ *B*

proof (*induct r1* **and** *r2*)
case (*R1Inf r*)
then obtain *C D* **where** *inf*: *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r* = *Some C rr2-of-rr2-rel* $\mathcal{F}$ *Rs r* = *Some D*
$\mathcal{L}$ *C* = $\mathcal{L}$ *D* **by** *auto*
have *spec*: *RR2-spec C* (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r*) *RR2-spec D* (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r*)
using *rr12-of-rr12-rel-correct*(2)[*OF assms*, *rule-format*, *OF inf*(2)] *inf*(3)
by (*auto simp*: *RR2-spec-def*)
then have *trim-spec*: *RR2-spec* (*trim-reg C*) (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r*)
RR2-spec (*trim-reg D*) (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r*)
by (*auto simp*: *RR2-spec-def* $\mathcal{L}$-*trim*)
let *?C* = *Inf-reg* (*trim-reg C*) (*Q-infty* (*ta* (*trim-reg C*)) $\mathcal{F}$) **let** *?D* = *Inf-reg* (*trim-reg D*) (*Q-infty* (*ta* (*trim-reg D*)) $\mathcal{F}$)
from *spec* **have** ∗: $\mathcal{L}$ (*Inf-reg-impl* (*trim-reg C*)) = $\mathcal{L}$ *?C*
using *eval-rr12-rel-sig*(2)[*of fset* $\mathcal{F}$ *map fset Rs r*]
by (*intro Inf-reg-impl-sound*) (*auto simp*: *RR2-spec-def* $\mathcal{L}$-*trim* $\mathcal{T}_G$-*equivalent-def*)
from *spec* **have** ∗∗: $\mathcal{L}$ (*Inf-reg-impl* (*trim-reg D*)) = $\mathcal{L}$ *?D*
using *eval-rr12-rel-sig*(2)[*of fset* $\mathcal{F}$ *map fset Rs r*]
by (*intro Inf-reg-impl-sound*) (*auto simp*: *RR2-spec-def* $\mathcal{L}$-*trim* $\mathcal{T}_G$-*equivalent-def*)
then have *C*: *RR2-spec ?C* {(*s*, *t*) | *s t*. *gpair s t* ∈ $\mathcal{L}$ *?C*} **and**
D: *RR2-spec ?D* {(*s*, *t*) | *s t*. *gpair s t* ∈ $\mathcal{L}$ *?D*}
using *subset-trans*[*OF Inf-automata-subseteq*[*of trim-reg C* $\mathcal{F}$], *of* $\mathcal{L}$ *C*] *spec*
using *subset-trans*[*OF Inf-automata-subseteq*[*of trim-reg D* $\mathcal{F}$], *of* $\mathcal{L}$ *D*]
using *eval-rr12-rel-sig*(2)[*of fset* $\mathcal{F}$ *map fset Rs r*]
by (*auto simp*: *RR2-spec-def* $\mathcal{L}$-*trim* $\mathcal{T}_G$-*equivalent-def intro*!: *equalityI fsubsetI*)
from ∗ ∗∗ **have** *r*: $\mathcal{L}$ (*proj-1-reg* (*Inf-reg-impl* (*trim-reg C*))) = $\mathcal{L}$ (*proj-1-reg ?C*)
$\mathcal{L}$ (*proj-1-reg* (*Inf-reg-impl* (*trim-reg D*))) = $\mathcal{L}$ (*proj-1-reg ?D*)
by (*auto intro*: $\mathcal{L}$-*proj-1-reg-cong*)
from $\mathcal{L}$-*Inf-reg*[*OF trim-spec*(1), *of* $\mathcal{F}$] $\mathcal{L}$-*Inf-reg*[*OF trim-spec*(2), *of* $\mathcal{F}$]
show *?case* **using** *R1Inf eval-rr12-rel-sig*(2)[*of fset* $\mathcal{F}$ *map fset Rs r*]
by (*auto simp*: *liftO1-def r inf* $\mathcal{T}_G$-*equivalent-def* $\mathcal{L}$-*proj*(1)[*OF C*] $\mathcal{L}$-*proj*(1)[*OF D*])
next
case (*R1Proj n x2*)
then show *?case* **by** (*cases n*)
(*auto simp*: *liftO1-def* $\mathcal{L}$-*trim proj-1-reg-def proj-2-reg-def intro*!: *fsubsetI* $\mathcal{L}$-*fmap-funs-reg-cong collapse-reg-cong*, (*meson fin-mono trim-reg-reach*)+)
next
case (*R2GTT-Rel g p n*) **note** *IH* = *this*
note *ass* = *R2GTT-Rel*
consider (*a*) ∃ *A*. *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g* = *Some A* | (*b*) *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g* = *None* **by** *blast*
then show *?case*
proof *cases*
case *a* **then obtain** *C D* **where** *gtt* [*simp*]: *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g* = *Some C*
gtt-of-gtt-rel $\mathcal{F}$ *Rs g* = *Some D* **using** *gtt-of-gtt-rel-impl-gtt-of-gtt-rel* **by** *blast*

from *gtt-of-gtt-rel-impl-sound*[*OF this*]
have *spec* [*simp*]: *agtt-lang C* = *agtt-lang D* **by** *auto*
have *eps* [*simp*]: *is-ta-eps-free* (*ta* (*GTT-to-RR2-root-reg C*))
using *gtt-of-gtt-rel-impl-is-gtt-eps-free*[*OF gtt*(*1*)]
by (*auto simp*: *GTT-to-RR2-root-reg-def GTT-to-RR2-root-def pair-automaton-def is-ta-eps-free-def is-gtt-eps-free-def*)
have *lang*: *L* (*GTT-to-RR2-root-reg C*) = *L* (*GTT-to-RR2-root-reg D*)
by (*metis* (*no-types*, *lifting*) *GTT-to-RR2-root RR2-spec-def spec*)
show *?thesis*
proof (*cases p*)
case *PRoot*
then show *?thesis* **using** *IH spec lang*
by (*cases n*) (*auto simp*: *L-eps-free L-reflcl-reg*)
next
case *PNonRoot*
then show *?thesis* **using** *IH*
by (*cases n*) (*auto simp*: *L-eps-free L-eps-free-nhole-ctxt-closure-reg*[*OF eps*] *L-eps-free-nhole-mctxt-reflcl-reg*[*OF eps*] *L-eps-free-nhole-mctxt-closure-reg*[*OF eps*]
lang intro: *L-nhole-ctxt-closure-reg-cong L-nhole-mctxt-reflcl-reg-cong L-nhole-mctxt-closure-reg-cong*)
next
case *PAny*
then show *?thesis* **using** *IH*
by (*cases n*) (*auto simp*: *L-eps-free L-eps-free-ctxt-closure-reg*[*OF eps*] *L-eps-free-parallel-closure-reg*[*OF eps*] *L-eps-free-mctxt-closure-reg*[*OF eps*]
lang
intro!: *L-ctxt-closure-reg-cong L-parallel-closure-reg-cong L-mctxt-closure-reg-cong*)
qed
next
case *b* **then show** *?thesis* **using** *IH*
by (*cases p*; *cases n*) *auto*
qed
next
case (*R2Comp x1 x2*)
then show *?case*
by (*auto simp*: *liftO1-def rr2-compositon-def L-trim L-intersect Let-def intro*!: *L-pair-automaton-reg-cong L-fmap-funs-reg-cong collapse-reg-cong arg-cong2*[**where** *?f* = (∩)])
qed (*auto simp*: *liftO1-def L-intersect L-union L-trim L-difference-reg intro*!: *L-fmap-funs-reg-cong L-pair-automaton-reg-cong*)
declare *equalityI*[*intro*!]
declare *fsubsetI*[*intro*!]

lemma *rr12-of-rr12-rel-impl-correct*:
assumes ∀ *R* ∈ *set Rs*. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| *F*
shows ∀ *ta1*. *rr1-of-rr1-rel-impl F Rs r1* = *Some ta1* ⟶ *RR1-spec ta1* (*eval-rr1-rel* (*fset F*) (*map fset Rs*) *r1*)
∀ *ta2*. *rr2-of-rr2-rel-impl F Rs r2* = *Some ta2* ⟶ *RR2-spec ta2* (*eval-rr2-rel* (*fset F*) (*map fset Rs*) *r2*)

using *rr12-of-rr12-rel-correct*(*1*)[*OF assms*, *of r1*]
using *rr12-of-rr12-rel-correct*(*2*)[*OF assms*, *of r2*]
using *rr2-of-rr2-rel-impl-sound*(*1*)[*OF assms*, *of r1*]
using *rr2-of-rr2-rel-impl-sound*(*2*)[*OF assms*, *of r2*]
using *rr-of-rr-rel-impl-complete*(*1*)[*of* $\mathcal{F}$ *Rs r1*]
using *rr-of-rr-rel-impl-complete*(*2*)[*of* $\mathcal{F}$ *Rs r2*]
by (*force simp*: *RR1-spec-def RR2-spec-def*)+

lemma *check-inference-rrn-impl-correct*:
assumes *sig*: $\mathcal{T}_G$ (*fset* $\mathcal{F}$) $\neq$ {} **and** *Rs*: $\forall R \in$ *set Rs*. *lv-trs* (*fset R*) $\wedge$ *ffunas-trs R* $|\subseteq|$ $\mathcal{F}$
assumes *infs*: $\bigwedge$*fvA*. *fvA* $\in$ *set infs* $\Longrightarrow$ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd fvA*)) (*snd* (*snd fvA*)) (*fst fvA*)
assumes *inf*: *check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs infs* (*l*, *step*, *fm*, *is*) = *Some* (*fm'*, *vs*, *A'*)
shows *l* = *length infs* $\wedge$ *fm* = *fm'* $\wedge$ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) *vs A' fm'*
using *check-inference-correct*[**where** *?rr1c* = *rr1-of-rr1-rel-impl* **and** *?rr2c* = *rr2-of-rr2-rel-impl*, *OF assms*]
using *rr12-of-rr12-rel-impl-correct*[*OF Rs*]
by *auto*

definition *check-sig-nempty* **where**
check-sig-nempty $\mathcal{F}$ = (*0* $|\in|$ *snd* $|`|$ $\mathcal{F}$)

definition *check-trss* **where**
check-trss $\mathcal{R}$ $\mathcal{F}$ = *list-all* (λ *R*. *lv-trs* (*fset R*) $\wedge$ *funas-trs* (*fset R*) $\subseteq$ *fset* $\mathcal{F}$) $\mathcal{R}$

lemma *check-sig-nempty*:
check-sig-nempty $\mathcal{F}$ $\longleftrightarrow$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) $\neq$ {} (**is** *?Ls* $\longleftrightarrow$ *?Rs*)
proof −
{assume *?Ls* **then obtain** *a* **where** (*a*, *0*) $|\in|$ $\mathcal{F}$ **by** (*auto simp*: *check-sig-nempty-def*)
then have *GFun a* [] $\in$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
by (*intro const*) *simp*
then have *?Rs* **by** *blast***}**
moreover
{assume *?Rs* **then obtain** *s* **where** *s* $\in$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) **by** *blast*
then obtain *a* **where** (*a*, *0*) $|\in|$ $\mathcal{F}$
by (*induct s*) (*auto*, *force*)
then have *?Ls* **unfolding** *check-sig-nempty-def*
by (*auto simp*: *image-iff Bex-def*)**}**
ultimately show *?thesis* **by** *blast*
qed

lemma *check-trss*:
check-trss $\mathcal{R}$ $\mathcal{F}$ $\longleftrightarrow$ ($\forall$ *R* $\in$ *set* $\mathcal{R}$. *lv-trs* (*fset R*) $\wedge$ *ffunas-trs R* $|\subseteq|$ $\mathcal{F}$)
unfolding *check-trss-def list-all-iff*
by (*auto simp*: *ffunas-trs.rep-eq less-eq-fset.rep-eq*)

```
fun check-inference-list :: ('f × nat) fset ⇒ ('f :: {compare,linorder}, 'v) fin-trs
list
  ⇒ (nat × ftrs inference × ftrs formula × info list) list
  ⇒ (ftrs formula × nat list × (nat, 'f option list) reg) list option where
  check-inference-list F Rs infs = do {
    guard (check-sig-nempty F);
    guard (check-trss Rs F);
    foldl (λ tas inf. do {
      tas' ← tas;
      r ← check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl F Rs tas' inf;
      Some (tas' @ [r])
    })
    (Some []) infs
  }

lemma check-inference-list-correct:
  assumes check-inference-list F Rs infs = Some fvAs
  shows length infs = length fvAs ∧ (∀ i < length fvAs. fst (snd (snd (infs ! i)))
= fst (fvAs ! i)) ∧
    (∀ i < length fvAs. formula-spec (fset F) (map fset Rs) (fst (snd (fvAs ! i)))
(snd (snd (fvAs ! i))) (fst (fvAs ! i)))
  using assms
proof (induct infs arbitrary: fvAs rule: rev-induct)
  note [simp] = bind-eq-Some-conv guard-simps
  {case Nil
    then show ?case by auto
  next
    case (snoc a infs)
    have inv: T_G (fset F) ≠ {} ∀ R∈set Rs. lv-trs (fset R) ∧ ffunas-trs R |⊆| F
      using snoc(2) by (auto simp: check-sig-nempty check-trss)
    from snoc(2) obtain fvAs' l steps fm fm' is' vs A' where
      ch: check-inference-list F Rs infs = Some fvAs' a = (l, steps, fm, is')
     check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl F Rs fvAs' (l, steps, fm,
is') = Some (fm', vs, A') fvAs = fvAs' @ [(fm', vs, A')]
      by (auto simp del: check-inference.simps) (metis prod-cases4)
    from snoc(1)[OF ch(1)] have fvA ∈ set fvAs' ⟹ formula-spec (fset F) (map
fset Rs) (fst (snd fvA)) (snd (snd fvA)) (fst fvA) for fvA
      by (auto dest: in-set-idx)
    from check-inference-rrn-impl-correct[OF inv this, of fvAs'] this
    show ?case using snoc(1)[OF ch(1)] ch
      by (auto simp del: check-inference.simps simp: nth-append)
  }
qed

fun check-certificate where
  check-certificate F Rs A fm (Certificate infs claim n) = do {
    guard (n < length infs);
    guard (A ⟷ claim = Nonempty);
    guard (fm = fst (snd (snd (infs ! n))));
```

```
    fvA ← check-inference-list F Rs (take (Suc n) infs);
    (let E = reg-empty (snd (snd (last fvA))) in
     case claim of Empty ⇒ Some E
       | - ⇒ Some (¬ E))
  }

definition formula-unsatisfiable where
  formula-unsatisfiable F Rs fm ⟷ (formula-satisfiable F Rs fm = False)

definition correct-certificate where
  correct-certificate F Rs claim infs n ≡
    (claim = Empty ⟷ (formula-unsatisfiable (fset F) (map fset Rs) (fst (snd
(snd (infs ! n))))) ∧
     claim = Nonempty ⟷ formula-satisfiable (fset F) (map fset Rs) (fst (snd
(snd (infs ! n)))))

lemma check-certificate-sound:
  assumes check-certificate F Rs A fm (Certificate infs claim n) = Some B
  shows fm = fst (snd (snd (infs ! n))) A ⟷ claim = Nonempty
  using assms by (auto simp: bind-eq-Some-conv guard-simps)

lemma check-certificate-correct:
  assumes check-certificate F Rs A fm (Certificate infs claim n) = Some B
  shows (B = True ⟶ correct-certificate F Rs claim infs n) ∧
    (B = False ⟶ correct-certificate F Rs (case-claim Nonempty Empty claim)
infs n)
proof −
  note [simp] = bind-eq-Some-conv guard-simps
  from assms obtain fvAs where inf: check-inference-list F Rs (take (Suc n)
infs) = Some fvAs
    by auto
  from assms have len: n < length infs by auto
  from check-inference-list-correct[OF inf] have
    inv: length fvAs = n + 1
    fst (snd (snd (infs ! n))) = fst (fvAs ! n)
   formula-spec (fset F) (map fset Rs) (fst (snd (last fvAs))) (snd (snd (last fvAs)))
(fst (last fvAs))
    using len last-conv-nth[of fvAs] by force+
  have nth: fst (last fvAs) = fst (fvAs ! n) using inv(1)
    using len last-conv-nth[of fvAs] by force
 note spec = formula-spec-empty[OF - inv(3)] formula-spec-nt-empty-form-sat[OF
- inv(3)]
  consider (a) claim = Empty | (b) claim = Nonempty using claim.exhaust by
blast
  then show ?thesis
  proof cases
    case a
    then have ∗: B = reg-empty (snd (snd (last fvAs))) using inv
      using assms len last-conv-nth[of fvAs]
```

by (*auto simp*: *inf simp del*: *check-inference-list.simps*)
show *?thesis* **using** *a inv spec* **unfolding** ∗
by (*auto simp*: *formula-satisfiable-def nth correct-certificate-def formula-unsatisfiable-def simp del*: *reg-empty*)
next
case *b*
then have ∗: *B* ⟷ ¬ (*reg-empty* (*snd* (*snd* (*last fvAs*)))) **using** *inv*
using *assms len last-conv-nth*[*of fvAs*]
by (*auto simp*: *inf simp del*: *check-inference-list.simps*)
show *?thesis* **using** *b inv spec* **unfolding** ∗
by (*auto simp*: *formula-satisfiable-def nth formula-unsatisfiable-def correct-certificate-def simp del*: *reg-empty*)
qed
qed

definition *check-certificate-string* ::
(*integer list* × *fvar*) *fset* ⇒
((*integer list*, *integer list*) *Term.term* × (*integer list*, *integer list*) *Term.term*) *fset list* ⇒
bool ⇒ *ftrs formula* ⇒ *ftrs certificate* ⇒ *bool option*
where *check-certificate-string* = *check-certificate*

export-code *check-certificate-string Var Fun fset-of-list nat-of-integer Certificate*
R2GTT-Rel R2Eq R2Reflc R2Step R2StepEq R2Steps R2StepsEq R2StepsNF R2ParStep R2RootStep
R2RootStepEq R2RootSteps R2RootStepsEq R2NonRootStep R2NonRootStepEq R2NonRootSteps
R2NonRootStepsEq R2Meet R2Join
ARoot GSteps PRoot ESingle Empty Size EDistribAndOr
R1Terms R1Fin
FRR1 FRestrict FTrue FFalse
IRR1 Fwd **in** *Haskell* **module-name** *FOR*

end

References

[1] S. Berghofer. First-order logic according to fitting. *Archive of Formal Proofs*, Aug. 2007. https://isa-afp.org/entries/FOL-Fitting.html, Formal proof development.

[2] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990.

[3] A. Lochmann, A. Middeldorp, F. Mitterwallner, and B. Felgenhauer. A verified decision procedure for the first-order theory of rewriting for linear variable-separated rewrite systems variable-separated rewrite systems in Isabelle/HOL. In C. Hricu and A. Popescu, editors, *Proc. 10th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 250–263, 2021.

[4] F. Mitterwallner, A. Lochmann, A. Middeldorp, and B. Felgenhauer. Certifying proofs in the first-order theory of rewriting. In J. F. Groote and K. G. Larsen, editors, *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12652 of *LNCS*, pages 127–144, 2021.