

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

Alexander Lochmann Bertram Felgenhauer

September 13, 2023

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

1	Introduction	4
1.1	Misc	4
2	Preliminaries	12
2.1	Multihole Contexts	12
2.1.1	Partitioning lists into chunks of given length	12
2.1.2	Multihole contexts definition and functionalities	15
2.1.3	Conversions from and to multihole contexts	15
2.1.4	Semilattice Structures	16
2.1.5	Lemmata	18
2.2	Ground multihole context	23
2.2.1	Basic function on ground multihole contexts	23
2.2.2	An inverse of <i>fill-gholes</i>	24
2.2.3	Orderings and compatibility of ground multihole contexts	26

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

2.2.4	Conversions from and to ground multihole contexts . .	26
2.2.5	Equivalences and simplification rules	29
2.2.6	Semilattice Structures	33
2.3	Bottom terms	41
2.4	Set operation closure for idempotent, associative, and com- mutative functions	44
3	Rewriting	52
3.1	Type definitions and rewrite relation definitions	52
3.2	Ground variants connecting to FORT	52
4	Primitive constructions	54
4.1	Recognizing subterms of linear terms	54
4.2	Recognizing root step relation of LV-TRSs	56
4.3	Recognizing normal forms of left linear TRSs	58
4.4	Sufficient condition for splitting the reachability relation in- duced by a tree automaton	63
5	(Multihole)Context closure of recognized tree languages	66
5.1	Tree Automata closure constructions	66
5.1.1	Reflexive closure over a given signature	66
5.1.2	Multihole context closure over a given signature . . .	67
5.1.3	Context closure of regular tree language	67
5.1.4	Not empty context closure of regular tree language . .	67
5.1.5	Non empty multihole context closure of regular tree language	68
5.1.6	Not empty multihole context closure of regular tree language	68
5.1.7	Multihole context closure of regular tree language . .	68
5.1.8	Lemmas about $ta\text{-}der'$	68
5.1.9	Signature induced by $refl\text{-}ta$ and $refl\text{-}over\text{-}states\text{-}ta$. .	69
5.1.10	Correctness of $refl\text{-}ta$, $gen\text{-}reflcl\text{-}automaton$, and $re\text{-}$ $flcl\text{-}automaton$	69
5.1.11	Correctness of $gen\text{-}parallel\text{-}closure\text{-}automaton$ and $par\text{-}$ $allel\text{-}closure\text{-}reg$	70
5.1.12	Correctness of $gen\text{-}ctxt\text{-}closure\text{-}reg$ and $ctxt\text{-}closure\text{-}reg$	71
5.1.13	Correctness of $gen\text{-}nhole\text{-}ctxt\text{-}closure\text{-}automaton$ and $nhole\text{-}ctxt\text{-}closure\text{-}reg$	73
5.1.14	Correctness of $gen\text{-}nhole\text{-}mctxt\text{-}closure\text{-}automaton$. .	74
5.1.15	Correctness of $gen\text{-}mctxt\text{-}closure\text{-}reg$ and $mctxt\text{-}closure\text{-}reg$	75
5.1.16	Correctness of $nhole\text{-}mctxt\text{-}reflcl\text{-}reg$	76
6	Type class instantiations for the implementation	76
6.0.1	Implementation of normal form construction	78

7	Multihole context and context closures over predicates	80
7.1	Elimination and introduction rules for the extensions	80
7.2	Monotonicity rules for the extensions	82
7.3	Relation swap and converse	83
7.4	Subset equivalence for context extensions over predicates . . .	84
7.5	<i>gmctxtex-onp</i> subset equivalence <i>gctxtex-onp</i> transitive closure	85
7.6	Extensions to reflexive transitive closures	86
7.7	Restr to set, union and predicate distribution	87
7.8	Distribution of context closures over relation composition . .	88
7.9	Signature preserving and signature closed	88
8	Certificate syntax and type declarations	89
8.1	GTT relations	89
8.2	RR1 and RR2 relations	90
8.3	Formulas	91
8.4	Signatures and Problems	92
8.5	Proofs	92
8.6	Example	92
9	Lifting root steps to single/parallel root/non-root steps	93
9.1	Rewrite steps equivalent definitions	94
9.2	Interface between rewrite step definitions and sets	94
9.3	Compatibility of used predicate extensions and signature closure	94
9.4	Basic lemmas	96
9.5	Equivalence lemmas	97
9.6	Signature preserving lemmas	97
9.7	<i>gcomp-rel</i> and <i>grancl-rel</i> lemmas	97
9.8	Auxiliary lemmas	102
10	Connecting regular tree languages to set/relation specifications	104
11	Additional support for FOL-Fitting	107
11.1	Iff	107
11.2	Replacement of subformulas	107
11.3	Propositional identities	108
11.4	de Bruijn index manipulation for formulas; cf. <i>liftt</i>	108
11.5	Quantifier Identities	109
11.6	Function symbols and predicates, with arities.	109
11.7	Negation Normal Form	109
11.8	Reasoning modulo ACI01	110
11.9	A (mostly) Propositional Equivalence Check	112
11.10	Reasoning modulo ACI01	112
11.11A	A (mostly) Propositional Equivalence Check	113

12 Semantics of Relations	114
12.1 Semantics of Formulas	115
12.2 Validation	116
12.3 Defining properties of <i>gcomp-rel</i> and <i>gtrancl-rel</i>	116
12.4 Correctness of derived constructions	117
13 Check inference steps	119
13.1 Computing TRSs	120
13.2 Computing GTTs	120
13.3 Computing RR1 and RR2 relations	121
13.4 Misc	123
13.5 Connect semantics to FOL-Fitting	123
13.6 RRn relations and formulas	124
13.7 Building blocks	126
13.7.1 IExists inference rule	127
13.8 Checking inferences	129
14 Inference checking implementation	131

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 automata operations in decision procedures for the first-order theory of rewriting 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* $\mathcal{R} = \bigcup ((\lambda (s, t). \text{funas-term } s \cup \text{funas-term } t) \text{ ` } \mathcal{R})$

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⟩

lemma *find-SomeD*:
List.find P xs = Some x ⇒ P x
List.find P xs = Some x ⇒ x ∈ set xs
⟨proof⟩

lemma *sum-list-replicate-length'* [simp]:
sum-list (replicate n (Suc 0)) = n
⟨proof⟩

lemma *arg-subteq* [simp]:
assumes t ∈ set ts **shows** Fun f ts ≥ t

<proof>

lemma *finite-funas-term*: *finite (funas-term s)*
<proof>

lemma *finite-funas-trs*:
finite $\mathcal{R} \implies \text{finite (funas-trs } \mathcal{R})$
<proof>

fun *subterms where*
subterms (Var x) = {Var x}|
subterms (Fun f ts) = {Fun f ts} \cup (\bigcup (subterms ‘ set ts))

lemma *finite-subterms-fun*: *finite (subterms s)*
<proof>

lemma *subterms-supteq-conv*: *$t \in \text{subterms } s \iff s \supseteq t$*
<proof>

lemma *set-all-subteq-subterms*:
subterms s = {t. s \supseteq t}
<proof>

lemma *finite-subterms*: *finite {t. s \supseteq t}*
<proof>

lemma *finite-strict-subterms*: *finite {t. s \supset t}*
<proof>

lemma *finite-UN-I2*:
finite A \implies ($\forall B \in A. \text{finite } B) \implies \text{finite } (\bigcup A)$
<proof>

lemma *root-substerns-funas-term*:
the ‘ (root ‘ (subterms s) – {None}) = funas-term s (is ?Ls = ?Rs)
<proof>

lemma *root-substerns-funas-term-set*:
the ‘ (root ‘ (\bigcup (subterms ‘ R) – {None}) = \bigcup (funas-term ‘ R)
<proof>

lemma *subst-merge*:
assumes *part*: *is-partition (map vars-term ts)*
shows $\exists \sigma. \forall i < \text{length } ts. \forall x \in \text{vars-term } (ts ! i). \sigma x = \tau i x$
<proof>

lemma *rel-comp-empty-trancl-simp*: *$R \circ R = \{\} \implies R^+ = R$*

$\langle proof \rangle$

lemma *choice-nat*:

assumes $\forall i < n. \exists x. P x i$

shows $\exists f. \forall x < n. P (f x) x$ $\langle proof \rangle$

lemma *subsetq-set-conv-nth*:

$(\forall i < \text{length } ss. ss ! i \in T) \longleftrightarrow \text{set } ss \subseteq T$

$\langle proof \rangle$

lemma *singleton-trancl [simp]*: $\{a\}^+ = \{a\}$

$\langle proof \rangle$

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 *fsubsetq-fset-conv-nth = subsetq-set-conv-nth* [*Transfer.transferred*]

lemmas *singleton-ftrancl [simp] = singleton-trancl* [*Transfer.transferred*]

end

lemma *set-take-nth*:

assumes $x \in \text{set } (take\ i\ xs)$

shows $\exists j < \text{length } xs. j < i \wedge xs ! j = x$ $\langle proof \rangle$

lemma *nth-sum-listI*:

assumes $\text{length } xs = \text{length } ys$

and $\forall i < \text{length } xs. xs ! i = ys ! i$

shows $\text{sum-list } xs = \text{sum-list } ys$

$\langle proof \rangle$

lemma *concat-nth-length*:

$i < \text{length } uss \implies j < \text{length } (uss ! i) \implies$

$\text{sum-list } (\text{map } \text{length } (\text{take } i\ uss)) + j < \text{length } (\text{concat } uss)$

$\langle proof \rangle$

lemma *sum-list-1-E [elim]*:

assumes $\text{sum-list } xs = \text{Suc } 0$

obtains i **where** $i < \text{length } xs$ $xs ! i = \text{Suc } 0$ $\forall j < \text{length } xs. j \neq i \implies xs ! j = 0$

$\langle proof \rangle$

lemma *nth-equalityE*:

$xs = ys \implies (\text{length } xs = \text{length } ys \implies (\bigwedge i. i < \text{length } xs \implies xs ! i = ys ! i) \implies P) \implies P$

$\langle proof \rangle$

lemma *map-cons-presv-distinct*:

$distinct\ t \implies distinct\ (map\ ((\#)\ i)\ t)$
<proof>

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 $\forall\ i < length\ (concat\ ts). P\ (concat\ ss\ !\ i)\ (concat\ ts\ !\ i)$
<proof>

lemma *last-nthI*:

assumes $i < length\ ts \ \neg\ i < length\ ts - Suc\ 0$
shows $ts\ !\ i = last\ ts$ *<proof>*

lemma *trancl-list-appendI* [*simp, intro*]:

$(xs, ys) \in trancl\text{-}list\ \mathcal{R} \implies (x, y) \in \mathcal{R} \implies (x\ \#\ xs, y\ \#\ ys) \in trancl\text{-}list\ \mathcal{R}$
<proof>

lemma *trancl-list-append-tranclI* [*intro*]:

$(x, y) \in \mathcal{R}^+ \implies (xs, ys) \in trancl\text{-}list\ \mathcal{R} \implies (x\ \#\ xs, y\ \#\ ys) \in trancl\text{-}list\ \mathcal{R}$
<proof>

lemma *trancl-list-conv*:

$(xs, ys) \in trancl\text{-}list\ \mathcal{R} \iff length\ xs = length\ ys \wedge (\forall\ i < length\ ys. (xs\ !\ i, ys\ !\ i) \in \mathcal{R}^+)$ (**is** $?Ls \iff ?Rs$)
<proof>

lemma *trancl-list-induct* [*consumes 2, case-names base step*]:

assumes $length\ ss = length\ ts \ \forall\ i < length\ ts. (ss\ !\ i, ts\ !\ i) \in \mathcal{R}^+$
and $\bigwedge xs\ ys. length\ xs = length\ ys \implies \forall\ i < length\ ys. (xs\ !\ i, ys\ !\ i) \in \mathcal{R} \implies P\ xs\ ys$
and $\bigwedge xs\ ys\ i\ z. length\ xs = length\ ys \implies \forall\ i < length\ ys. (xs\ !\ i, ys\ !\ i) \in \mathcal{R}^+ \implies P\ xs\ ys$
 $\implies i < length\ ys \implies (ys\ !\ i, z) \in \mathcal{R} \implies P\ xs\ (ys[i := z])$
shows $P\ ss\ ts$ *<proof>*

lemma *swap-trancl*:

$(prod.swap\ 'R)^+ = prod.swap\ '(R^+)$
<proof>

lemma *swap-rtrancl*:

$(prod.swap\ 'R)^* = prod.swap\ '(R^*)$
<proof>

lemma *Restr-simps*:

$$\begin{aligned}
R \subseteq X \times X &\implies \text{Restr } (R^+) X = R^+ \\
R \subseteq X \times X &\implies \text{Restr Id } X \circ R = R \\
R \subseteq X \times X &\implies R \circ \text{Restr Id } X = R \\
R \subseteq X \times X &\implies S \subseteq X \times X \implies \text{Restr } (R \circ S) X = R \circ S \\
R \subseteq X \times X &\implies R^+ \subseteq X \times X \\
&\langle \text{proof} \rangle
\end{aligned}$$

lemma *Restr-tracl-comp-simps*:

$$\begin{aligned}
\mathcal{R} \subseteq X \times X &\implies \mathcal{L} \subseteq X \times X \implies \mathcal{L}^+ \circ \mathcal{R} \subseteq X \times X \\
\mathcal{R} \subseteq X \times X &\implies \mathcal{L} \subseteq X \times X \implies \mathcal{L} \circ \mathcal{R}^+ \subseteq X \times X \\
\mathcal{R} \subseteq X \times X &\implies \mathcal{L} \subseteq X \times X \implies \mathcal{L}^+ \circ \mathcal{R} \circ \mathcal{L}^+ \subseteq X \times X \\
&\langle \text{proof} \rangle
\end{aligned}$$

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

lemma *concat-index-split-mono-first-arg*:

$$i < \text{length } (\text{concat } xs) \implies o\text{-idx} \leq \text{fst } (\text{concat-index-split } (o\text{-idx}, i) xs)$$

<proof>

lemma *concat-index-split-sound-fst-arg-aux*:

$$i < \text{length } (\text{concat } xs) \implies \text{fst } (\text{concat-index-split } (o\text{-idx}, i) xs) < \text{length } xs + o\text{-idx}$$

<proof>

lemma *concat-index-split-sound-fst-arg*:

$$i < \text{length } (\text{concat } xs) \implies \text{fst } (\text{concat-index-split } (0, i) xs) < \text{length } xs$$

<proof>

lemma *concat-index-split-sound-snd-arg-aux*:

assumes $i < \text{length } (\text{concat } xs)$
shows $\text{snd } (\text{concat-index-split } (n, i) xs) < \text{length } (xs ! (\text{fst } (\text{concat-index-split } (n, i) xs) - n))$ *<proof>*

lemma *concat-index-split-sound-snd-arg*:

assumes $i < \text{length } (\text{concat } xs)$
shows $\text{snd } (\text{concat-index-split } (0, i) xs) < \text{length } (xs ! \text{fst } (\text{concat-index-split } (0, i) xs))$
<proof>

lemma *reconstr-1d-concat-index-split*:

assumes $i < \text{length } (\text{concat } xs)$
shows $i = (\lambda (m, j). \text{sum-list } (\text{map length } (\text{take } (m - n) xs)) + j) (\text{concat-index-split } (n, i) xs)$ *<proof>*

lemma *concat-index-split-larger-lists [simp]*:

assumes $i < \text{length } (\text{concat } xs)$
shows $\text{concat-index-split } (n, i) (xs @ ys) = \text{concat-index-split } (n, i) xs$ *<proof>*

lemma *concat-index-split-split-sound-aux*:

assumes $i < \text{length} (\text{concat } xs)$
shows $\text{concat } xs ! i = (\lambda (k, j). xs ! (k - n) ! j) (\text{concat-index-split } (n, i) xs)$
 $\langle \text{proof} \rangle$

lemma *concat-index-split-sound*:
assumes $i < \text{length} (\text{concat } xs)$
shows $\text{concat } xs ! i = (\lambda (k, j). xs ! k ! j) (\text{concat-index-split } (0, i) xs)$
 $\langle \text{proof} \rangle$

lemma *concat-index-split-sound-bounds*:
assumes $i < \text{length} (\text{concat } xs)$ **and** $\text{concat-index-split } (0, i) xs = (m, n)$
shows $m < \text{length } xs$ $n < \text{length } (xs ! m)$
 $\langle \text{proof} \rangle$

lemma *concat-index-split-less-length-concat*:
assumes $i < \text{length} (\text{concat } xs)$ **and** $\text{concat-index-split } (0, i) xs = (m, n)$
shows $i = \text{sum-list } (\text{map length } (\text{take } m \ xs)) + n$ $m < \text{length } xs$ $n < \text{length } (xs ! m)$
 $\text{concat } xs ! i = xs ! m ! n$
 $\langle \text{proof} \rangle$

lemma *nth-concat-split'*:
assumes $i < \text{length} (\text{concat } xs)$
obtains $j \ k$ **where** $j < \text{length } xs$ $k < \text{length } (xs ! j)$ $\text{concat } xs ! i = xs ! j ! k$ $i = \text{sum-list } (\text{map length } (\text{take } j \ xs)) + k$
 $\langle \text{proof} \rangle$

lemma *sum-list-split* [*dest!*, *consumes* l]:
assumes $\text{sum-list } (\text{map length } (\text{take } i \ xs)) + j = \text{sum-list } (\text{map length } (\text{take } k \ xs)) + l$
and $i < \text{length } xs$ $k < \text{length } xs$
and $j < \text{length } (xs ! i)$ $l < \text{length } (xs ! k)$
shows $i = k \wedge j = l$ $\langle \text{proof} \rangle$

lemma *concat-index-split-unique*:
assumes $i < \text{length} (\text{concat } xs)$ **and** $\text{length } xs = \text{length } ys$
and $\forall i < \text{length } xs. \text{length } (xs ! i) = \text{length } (ys ! i)$
shows $\text{concat-index-split } (n, i) xs = \text{concat-index-split } (n, i) ys$ $\langle \text{proof} \rangle$

lemma *set-vars-term-list* [*simp*]:
 $\text{set } (\text{vars-term-list } t) = \text{vars-term } t$
 $\langle \text{proof} \rangle$

lemma *vars-term-list-empty-ground* [*simp*]:
 $\text{vars-term-list } t = [] \iff \text{ground } t$
 $\langle \text{proof} \rangle$

lemma *varposs-imp-poss*:
assumes $p \in \text{varposs } t$

shows $p \in \text{poss } t$
 $\langle \text{proof} \rangle$

lemma *varposs-list-fun*:
assumes $p \in \text{set } (\text{varposs-list } (\text{Fun } f \text{ } ts))$
obtains $i \text{ } ps$ **where** $i < \text{length } ts$ $p = i \# ps$
 $\langle \text{proof} \rangle$

lemma *varposs-list-distinct*:
 $\text{distinct } (\text{varposs-list } t)$
 $\langle \text{proof} \rangle$

lemma *varposs-append*:
 $\text{varposs } (\text{Fun } f \text{ } (ts \text{ @ } [t])) = \text{varposs } (\text{Fun } f \text{ } ts) \cup ((\#) (\text{length } ts)) \text{ ` } \text{varposs } t$
 $\langle \text{proof} \rangle$

lemma *varposs-eq-varposs-list*:
 $\text{set } (\text{varposs-list } t) = \text{varposs } t$
 $\langle \text{proof} \rangle$

lemma *varposs-list-var-terms-length*:
 $\text{length } (\text{varposs-list } t) = \text{length } (\text{vars-term-list } t)$
 $\langle \text{proof} \rangle$

lemma *vars-term-list-nth*:
assumes $i < \text{length } (\text{vars-term-list } (\text{Fun } f \text{ } ts))$
and $\text{concat-index-split } (0, i) (\text{map } \text{vars-term-list } ts) = (k, j)$
shows $k < \text{length } ts \wedge j < \text{length } (\text{vars-term-list } (ts \text{ ! } k)) \wedge$
 $\text{vars-term-list } (\text{Fun } f \text{ } ts) \text{ ! } i = \text{map } \text{vars-term-list } ts \text{ ! } k \text{ ! } j \wedge$
 $i = \text{sum-list } (\text{map } \text{length } (\text{map } \text{vars-term-list } (\text{take } k \text{ } ts))) + j$
 $\langle \text{proof} \rangle$

lemma *varposs-list-nth*:
assumes $i < \text{length } (\text{varposs-list } (\text{Fun } f \text{ } ts))$
and $\text{concat-index-split } (0, i) (\text{poss-args } \text{varposs-list } ts) = (k, j)$
shows $k < \text{length } ts \wedge j < \text{length } (\text{varposs-list } (ts \text{ ! } k)) \wedge$
 $\text{varposs-list } (\text{Fun } f \text{ } ts) \text{ ! } i = k \# (\text{map } \text{varposs-list } ts) \text{ ! } k \text{ ! } j \wedge$
 $i = \text{sum-list } (\text{map } \text{length } (\text{map } \text{varposs-list } (\text{take } k \text{ } ts))) + j$
 $\langle \text{proof} \rangle$

lemma *varposs-list-to-var-term-list*:
assumes $i < \text{length } (\text{varposs-list } t)$
shows $\text{the-Var } (t \text{ |- } (\text{varposs-list } t \text{ ! } i)) = (\text{vars-term-list } t) \text{ ! } i$ $\langle \text{proof} \rangle$

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  $m < \text{length } xs$  and  $n < \text{length } (xs ! m)$ 
    and  $i = \text{sum-list } (\text{map } \text{length } (\text{take } m \text{ } xs)) + n$ 
  shows  $\text{concat } xs ! i = xs ! m ! n$ 
  <proof>
```

```
lemma sum-list-take-eq:
```

```
  fixes  $xs :: \text{nat list}$ 
  shows  $k < i \implies i < \text{length } xs \implies \text{sum-list } (\text{take } i \text{ } xs) =$ 
     $\text{sum-list } (\text{take } k \text{ } xs) + xs ! k + \text{sum-list } (\text{take } (i - \text{Suc } k) \text{ } (\text{drop } (\text{Suc } k) \text{ } xs))$ 
  <proof>
```

```
fun partition-by where
```

```
  partition-by  $xs [] = []$  |
  partition-by  $xs (y\#ys) = \text{take } y \text{ } xs \# \text{partition-by } (\text{drop } y \text{ } xs) \text{ } ys$ 
```

```
lemma partition-by-map0-append [simp]:
```

```
   $\text{partition-by } xs (\text{map } (\lambda x. 0) \text{ } ys @ zs) = \text{replicate } (\text{length } ys) [] @ \text{partition-by } xs$ 
   $zs$ 
  <proof>
```

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

```
   $\text{sum-list } ys = \text{length } xs \implies \text{concat } (\text{partition-by } xs \text{ } ys) = xs$ 
  <proof>
```

```
definition partition-by-idx where
```

```
  partition-by-idx  $l \text{ } ys \text{ } i \text{ } j = \text{partition-by } [0..<l] \text{ } ys ! i ! j$ 
```

```
lemma partition-by-nth-nth-old:
```

```
  assumes  $i < \text{length } (\text{partition-by } xs \text{ } ys)$ 
    and  $j < \text{length } (\text{partition-by } xs \text{ } ys ! i)$ 
    and  $\text{sum-list } ys = \text{length } xs$ 
  shows  $\text{partition-by } xs \text{ } ys ! i ! j = xs ! (\text{sum-list } (\text{map } \text{length } (\text{take } i \text{ } (\text{partition-by } xs \text{ } ys)))) + j$ 
  <proof>
```

```
lemma map-map-partition-by:
```

$\text{map } (\text{map } f) (\text{partition-by } xs \ ys) = \text{partition-by } (\text{map } f \ xs) \ ys$
(proof)

lemma *length-partition-by* [simp]:
 $\text{length } (\text{partition-by } xs \ ys) = \text{length } ys$
(proof)

lemma *partition-by-Nil* [simp]:
 $\text{partition-by } [] \ ys = \text{replicate } (\text{length } ys) \ []$
(proof)

lemma *partition-by-concat-id* [simp]:
assumes $\text{length } xss = \text{length } ys$
and $\bigwedge i. i < \text{length } ys \implies \text{length } (xss \ ! \ i) = ys \ ! \ i$
shows $\text{partition-by } (\text{concat } xss) \ ys = xss$
(proof)

lemma *partition-by-nth*:
 $i < \text{length } ys \implies \text{partition-by } xs \ ys \ ! \ i = \text{take } (ys \ ! \ i) \ (\text{drop } (\text{sum-list } (\text{take } i \ ys)) \ xs)$
(proof)

lemma *partition-by-nth-less*:
assumes $k < i$ and $i < \text{length } zs$
and $\text{length } xs = \text{sum-list } (\text{take } i \ zs) + j$
shows $\text{partition-by } (xs \ @ \ y \ \# \ ys) \ zs \ ! \ k = \text{take } (zs \ ! \ k) \ (\text{drop } (\text{sum-list } (\text{take } k \ zs)) \ xs)$
(proof)

lemma *partition-by-nth-greater*:
assumes $i < k$ and $k < \text{length } zs$ and $j < zs \ ! \ i$
and $\text{length } xs = \text{sum-list } (\text{take } i \ zs) + j$
shows $\text{partition-by } (xs \ @ \ y \ \# \ ys) \ zs \ ! \ k = \text{take } (zs \ ! \ k) \ (\text{drop } (\text{sum-list } (\text{take } k \ zs) - 1) \ (xs \ @ \ ys))$
(proof)

lemma *length-partition-by-nth*:
 $\text{sum-list } ys = \text{length } xs \implies i < \text{length } ys \implies \text{length } (\text{partition-by } xs \ ys \ ! \ i) = ys \ ! \ i$
(proof)

lemma *partition-by-nth-nth-elem*:
assumes $\text{sum-list } ys = \text{length } xs$ $i < \text{length } ys$ $j < ys \ ! \ i$
shows $\text{partition-by } xs \ ys \ ! \ i \ ! \ j \in \text{set } xs$
(proof)

lemma *partition-by-nth-nth*:
assumes $\text{sum-list } ys = \text{length } xs$ $i < \text{length } ys$ $j < ys \ ! \ i$
shows $\text{partition-by } xs \ ys \ ! \ i \ ! \ j = xs \ ! \ \text{partition-by-idx } (\text{length } xs) \ ys \ i \ j$

partition-by-idx (*length xs*) *ys* *i* *j* < *length xs*
 ⟨*proof*⟩

lemma *map-length-partition-by* [*simp*]:
 $sum-list\ ys = length\ xs \implies map\ length\ (partition-by\ xs\ ys) = ys$
 ⟨*proof*⟩

lemma *map-partition-by-nth* [*simp*]:
 $i < length\ ys \implies map\ f\ (partition-by\ xs\ ys\ !\ i) = partition-by\ (map\ f\ xs)\ ys\ !\ i$
 ⟨*proof*⟩

lemma *sum-list-partition-by* [*simp*]:
 $sum-list\ ys = length\ xs \implies$
 $sum-list\ (map\ (\lambda x. sum-list\ (map\ f\ x))\ (partition-by\ xs\ ys)) = sum-list\ (map\ f\ xs)$
 ⟨*proof*⟩

lemma *partition-by-map-conv*:
 $partition-by\ xs\ ys = map\ (\lambda i. take\ (ys\ !\ i)\ (drop\ (sum-list\ (take\ i\ ys))\ xs))\ [0..<length\ ys]$
 ⟨*proof*⟩

lemma *UN-set-partition-by-map*:
 $sum-list\ ys = length\ xs \implies (\bigcup_{x \in set\ (partition-by\ (map\ f\ xs)\ ys)} set\ x) = \bigcup_{x \in set\ (map\ f\ xs)} set\ x$
 ⟨*proof*⟩

lemma *UN-set-partition-by*:
 $sum-list\ ys = length\ xs \implies (\bigcup_{zs \in set\ (partition-by\ xs\ ys)} set\ zs) = \bigcup_{x \in set\ xs} set\ x$
 ⟨*proof*⟩

lemma *Ball-atLeast0LessThan-partition-by-conv*:
 $(\forall i \in \{0..<length\ ys\}. \forall x \in set\ (partition-by\ xs\ ys\ !\ i). P\ x) = (\forall x \in \bigcup_{i \in \{0..<length\ ys\}} set\ (partition-by\ xs\ ys\ !\ i). P\ x)$
 ⟨*proof*⟩

lemma *Ball-set-partition-by*:
 $sum-list\ ys = length\ xs \implies (\forall x \in set\ (partition-by\ xs\ ys). \forall y \in set\ x. P\ y) = (\forall x \in set\ xs. P\ x)$
 ⟨*proof*⟩

lemma *partition-by-append2*:
 $partition-by\ xs\ (ys\ @\ zs) = partition-by\ (take\ (sum-list\ ys)\ xs)\ ys\ @\ partition-by\ (drop\ (sum-list\ ys)\ xs)\ zs$
 ⟨*proof*⟩

lemma *partition-by-concat2*:
 $partition-by\ xs\ (concat\ ys) =$

```

concat (map (λi . partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i))
[0..<length ys])
⟨proof⟩

```

lemma *partition-by-partition-by*:

```

length xs = sum-list (map sum-list ys) ==>
partition-by (partition-by xs (concat ys)) (map length ys) =
map (λi. partition-by (partition-by xs (map sum-list ys) ! i) (ys ! i)) [0..<length
ys]
⟨proof⟩

```

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 (λi. 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 ((-/
=ₓ -) [51, 51] 50)
where
  eqfI [intro]: t = fill-holes D ss ⇒ num-holes D = length ss ⇒ t =ₓ (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  $\sqcap$  D = MHole |
C  $\sqcap$  MHole = MHole |
MVar x  $\sqcap$  MVar y = (if x = y then MVar x else MHole) |
MFun f Cs  $\sqcap$  MFun g Ds =
  (if f = g  $\wedge$  length Cs = length Ds then MFun f (map (case-prod ( $\sqcap$ )) (zip Cs
Ds))
  else MHole) |
C  $\sqcap$  D = MHole

```

instance \langle proof \rangle

end

lemma *inf-mctxt-idem* [*simp*]:

fixes C :: ('f, 'v) mctxt

shows C \sqcap C = C

\langle proof \rangle

lemma *inf-mctxt-MHole2* [*simp*]:

C \sqcap MHole = MHole

\langle proof \rangle

lemma *inf-mctxt-comm* [*ac-simps*]:

(C :: ('f, 'v) mctxt) \sqcap D = D \sqcap C

\langle proof \rangle

lemma *inf-mctxt-assoc* [*ac-simps*]:

fixes C :: ('f, 'v) mctxt

shows C \sqcap D \sqcap E = C \sqcap (D \sqcap E)

\langle proof \rangle

instantiation mctxt :: (type, type) order

begin

definition (C :: ('a, 'b) mctxt) \leq D \iff C \sqcap D = C

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

instance

\langle proof \rangle

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 \implies (\bigwedge i. i < length cs \implies *less-eq-mctxt'* (cs ! i) (ds ! i))

\implies *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*]]
 ⟨*proof*⟩

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*]]
 ⟨*proof*⟩

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* $?ys = \text{length } ?xs \implies \text{concat } (\text{partition-by } ?xs \ ?ys) = ?xs$ 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*: $\bigwedge x y. P \text{ MHole } [x] [y]$
and *Var*: $\bigwedge v. P (\text{MVar } v) [] []$
and *Fun*: $\bigwedge f Cs xs ys. \text{sum-list } (\text{map } \text{num-holes } Cs) = \text{length } xs \implies$
 $\text{sum-list } (\text{map } \text{num-holes } Cs) = \text{length } ys \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P (Cs ! i) (\text{partition-holes } xs \ Cs ! i) (\text{partition-holes } ys \ Cs ! i)) \implies$
 $P (\text{MFun } f \ Cs) (\text{concat } (\text{partition-holes } xs \ Cs)) (\text{concat } (\text{partition-holes } ys \ Cs))$
shows *P* *C* *xs* *ys*
 ⟨*proof*⟩

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*: $\bigwedge x. P \text{ MHole } [x]$
and *Var*: $\bigwedge v. P (\text{MVar } v) [] []$
and *Fun*: $\bigwedge f Cs xs. \text{sum-list } (\text{map } \text{num-holes } Cs) = \text{length } xs \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P (Cs ! i) (\text{partition-holes } xs \ Cs ! i)) \implies$
 $P (\text{MFun } f \ Cs) (\text{concat } (\text{partition-holes } xs \ Cs))$
shows *P* *C* *xs*
 ⟨*proof*⟩

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*)
 ⟨*proof*⟩

lemmas

$\text{map-partition-holes-nth } [simp] =$
 $\text{map-partition-by-nth } [of - \text{map num-holes } Cs \text{ for } Cs, \text{ unfolded length-map}]$ **and**
 $\text{length-partition-holes } [simp] =$
 $\text{length-partition-by } [of - \text{map num-holes } Cs \text{ for } Cs, \text{ unfolded length-map}]$

lemma *fill-holes-term-of-mctxt*:

$\text{num-holes } C = 0 \implies \text{fill-holes } C [] = \text{term-of-mctxt } C$
 $\langle \text{proof} \rangle$

lemma *fill-holes-MHole*:

$\text{length } ts = \text{Suc } 0 \implies ts ! 0 = u \implies \text{fill-holes } MHole \ ts = u$
 $\langle \text{proof} \rangle$

lemma *fill-holes-arbitrary*:

assumes $lCs: \text{length } Cs = \text{length } ts$
and $lss: \text{length } ss = \text{length } ts$
and $\text{rec}: \bigwedge i. i < \text{length } ts \implies \text{num-holes } (Cs ! i) = \text{length } (ss ! i) \wedge f (Cs ! i) (ss ! i) = ts ! i$
shows $\text{map } (\lambda i. f (Cs ! i) (\text{partition-holes } (\text{concat } ss) \ Cs ! i)) [0 ..< \text{length } Cs] = ts$
 $\langle \text{proof} \rangle$

lemma *fill-holes-MFun*:

assumes $lCs: \text{length } Cs = \text{length } ts$
and $lss: \text{length } ss = \text{length } ts$
and $\text{rec}: \bigwedge i. i < \text{length } ts \implies \text{num-holes } (Cs ! i) = \text{length } (ss ! i) \wedge \text{fill-holes } (Cs ! i) (ss ! i) = ts ! i$
shows $\text{fill-holes } (MFun \ f \ Cs) (\text{concat } ss) = Fun \ f \ ts$
 $\langle \text{proof} \rangle$

lemma *eqfE*:

assumes $t =_f (D, ss)$ **shows** $t = \text{fill-holes } D \ ss \ \text{num-holes } D = \text{length } ss$
 $\langle \text{proof} \rangle$

lemma *eqf-MFunE*:

assumes $s =_f (MFun \ f \ Cs, ss)$
obtains $ts \ sss$ **where** $s = Fun \ f \ ts \ \text{length } ts = \text{length } Cs \ \text{length } sss = \text{length } Cs$
 $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$
 $ss = \text{concat } sss$
 $\langle \text{proof} \rangle$

lemma *eqf-MFunI*:

assumes $\text{length } sss = \text{length } Cs$
and $\text{length } ts = \text{length } Cs$
and $\bigwedge i. i < \text{length } Cs \implies ts ! i =_f (Cs ! i, sss ! i)$
shows $Fun \ f \ ts =_f (MFun \ f \ Cs, \text{concat } sss)$
 $\langle \text{proof} \rangle$

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) <proof>*

lemma *split-vars-vars-term-list*: *snd (split-vars t) = vars-term-list t*
<proof>

lemma *split-vars-num-holes*: *num-holes (fst (split-vars t)) = length (snd (split-vars t))*
<proof>

lemma *ground-eq-fill*: *t =_f (C,ss) \implies ground t = (ground-mctxt C \wedge ($\forall s \in$ set ss. ground s))*
<proof>

lemma *ground-fill-holes*:

assumes *nh: num-holes C = length ss*

shows *ground (fill-holes C ss) = (ground-mctxt C \wedge ($\forall s \in$ set ss. ground s))*

<proof>

lemma *split-vars-ground' [simp]*:

ground-mctxt (fst (split-vars t))

<proof>

lemma *split-vars-funas-mctxt [simp]*:

funas-mctxt (fst (split-vars t)) = funas-term t

<proof>

lemma *less-eq-mctxt-prime*: *C \leq D \longleftrightarrow less-eq-mctxt' C D*

<proof>

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*

<proof>

lemma *less-eq-mctxt-MVarE2*:

assumes *C \leq MVar v*

obtains *(MHole) C = MHole | (MVar) C = MVar v*

<proof>

lemma *less-eq-mctxt-MFunE2*:

assumes $C \leq \text{MFun } f \text{ } ds$
obtains $(\text{MHole}) C = \text{MHole}$
 $\quad | (\text{MFun}) cs \textbf{ where } C = \text{MFun } f \text{ } cs \text{ length } cs = \text{length } ds \wedge i. i < \text{length } cs \implies$
 $cs ! i \leq ds ! i$
 $\langle \text{proof} \rangle$

lemmas $\text{less-eq-mctxtE2} = \text{less-eq-mctxt-MHoleE2} \text{ less-eq-mctxt-MVarE2} \text{ less-eq-mctxt-MFunE2}$

lemma $\text{less-eq-mctxt-MVarE1}$:
assumes $\text{MVar } v \leq D$
obtains $(\text{MVar}) D = \text{MVar } v$
 $\langle \text{proof} \rangle$

lemma $\text{MHole-Bot} [\text{simp}]$: $\text{MHole} \leq D$
 $\langle \text{proof} \rangle$

lemma $\text{less-eq-mctxt-MFunE1}$:
assumes $\text{MFun } f \text{ } cs \leq D$
obtains $(\text{MFun}) ds \textbf{ where } D = \text{MFun } f \text{ } ds \text{ length } cs = \text{length } ds \wedge i. i < \text{length}$
 $cs \implies cs ! i \leq ds ! i$
 $\langle \text{proof} \rangle$

lemma $\text{length-unfill-holes} [\text{simp}]$:
assumes $C \leq \text{mctxt-of-term } t$
shows $\text{length} (\text{unfill-holes } C \text{ } t) = \text{num-holes } C$
 $\langle \text{proof} \rangle$

lemma $\text{map-vars-mctxt-id} [\text{simp}]$:
 $\text{map-vars-mctxt } (\lambda x. x) C = C$
 $\langle \text{proof} \rangle$

lemma $\text{split-vars-egf-subst-map-vars-term}$:
 $t \cdot \sigma =_f (\text{map-vars-mctxt } vw (\text{fst } (\text{split-vars } t)), \text{map } \sigma (\text{snd } (\text{split-vars } t)))$
 $\langle \text{proof} \rangle$

lemma $\text{split-vars-egf-subst}$: $t \cdot \sigma =_f (\text{fst } (\text{split-vars } t), (\text{map } \sigma (\text{snd } (\text{split-vars } t))))$
 $\langle \text{proof} \rangle$

lemma $\text{split-vars-fill-holes}$:
assumes $C = \text{fst } (\text{split-vars } s) \textbf{ and } ss = \text{map } \text{Var } (\text{snd } (\text{split-vars } s))$
shows $\text{fill-holes } C \text{ } ss = s \langle \text{proof} \rangle$

lemma fill-unfill-holes :
assumes $C \leq \text{mctxt-of-term } t$
shows $\text{fill-holes } C (\text{unfill-holes } C \text{ } t) = t$

$\langle \text{proof} \rangle$

lemma *hole-poss-list-length*:

$\text{length } (\text{hole-poss-list } D) = \text{num-holes } D$

$\langle \text{proof} \rangle$

lemma *unfill-holes-hole-poss-list-length*:

assumes $C \leq \text{mctxt-of-term } t$

shows $\text{length } (\text{unfill-holes } C t) = \text{length } (\text{hole-poss-list } C) \langle \text{proof} \rangle$

lemma *unfill-holes-to-subst-at-hole-poss*:

assumes $C \leq \text{mctxt-of-term } t$

shows $\text{unfill-holes } C t = \text{map } ([-] t) (\text{hole-poss-list } C) \langle \text{proof} \rangle$

lemma *hole-poss-split-varposs-list-length* [*simp*]:

$\text{length } (\text{hole-poss-list } (\text{fst } (\text{split-vars } t))) = \text{length } (\text{varposs-list } t)$

$\langle \text{proof} \rangle$

lemma *hole-poss-split-vars-varposs-list*:

$\text{hole-poss-list } (\text{fst } (\text{split-vars } t)) = \text{varposs-list } t$

$\langle \text{proof} \rangle$

lemma *funas-term-fill-holes-iff*: $\text{num-holes } C = \text{length } ts \implies$

$g \in \text{funas-term } (\text{fill-holes } C ts) \iff g \in \text{funas-mctxt } C \vee (\exists t \in \text{set } ts. g \in \text{funas-term } t)$

$\langle \text{proof} \rangle$

lemma *vars-term-fill-holes* [*simp*]:

$\text{num-holes } C = \text{length } ts \implies \text{ground-mctxt } C \implies$

$\text{vars-term } (\text{fill-holes } C ts) = \bigcup (\text{vars-term } ` \text{set } ts)$

$\langle \text{proof} \rangle$

lemma *funas-mctxt-fill-holes* [*simp*]:

assumes $\text{num-holes } C = \text{length } ts$

shows $\text{funas-term } (\text{fill-holes } C ts) = \text{funas-mctxt } C \cup \bigcup (\text{set } (\text{map } \text{funas-term } ts))$

$\langle \text{proof} \rangle$

lemma *funas-mctxt-fill-holes-mctxt* [*simp*]:

assumes $\text{num-holes } C = \text{length } Ds$

shows $\text{funas-mctxt } (\text{fill-holes-mctxt } C Ds) = \text{funas-mctxt } C \cup \bigcup (\text{set } (\text{map } \text{funas-mctxt } Ds))$

(**is** $?f C Ds = ?g C Ds$)

$\langle \text{proof} \rangle$

```

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

```

2.2 Ground multihole context

```

datatype (gfun<math>-m</math>ctxt: 'f) gmctxt = GMHole | GMFun 'f 'f gmctxt list

```

2.2.1 Basic function on ground multihole contexts

```

primrec gmctxt-of-gterm :: 'f gterm  $\Rightarrow$  'f gmctxt where
  gmctxt-of-gterm (GFun f ts) = GMFun f (map gmctxt-of-gterm ts)

```

```

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

```

```

primrec gterm-of-gmctxt :: 'f gmctxt  $\Rightarrow$  'f gterm where
  gterm-of-gmctxt (GMFun f Cs) = GFun f (map gterm-of-gmctxt Cs)

```

```

primrec term-of-gmctxt :: 'f gmctxt  $\Rightarrow$  ('f, 'v) term where
  term-of-gmctxt (GMFun f Cs) = Fun f (map term-of-gmctxt Cs)

```

```

primrec gmctxt-of-gctxt :: 'f gctxt  $\Rightarrow$  'f gmctxt where
  gmctxt-of-gctxt  $\square_G$  = 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  $\Rightarrow$  'f gctxt where
  gctxt-of-gmctxt GMHole =  $\square_G$ 
| gctxt-of-gmctxt (GMFun f Cs) = (let n = length (takeWhile ( $\lambda$  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  $\Rightarrow$  '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  $\Rightarrow$  ('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* (λ *i*. *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*]:
 $ghole-poss (GMFun f Cs) = \{i \# p \mid i p. i < length\ Cs \wedge p \in ghole-poss (Cs ! i)\}$
 $poss-gmctxt (GMFun f Cs) = \{\square\} \cup \{i \# p \mid i p. i < length\ Cs \wedge p \in poss-gmctxt (Cs ! i)\}$
 ⟨*proof*⟩

fun *ghole-num-bef-pos* **where**
 $ghole-num-bef-pos \square = 0 \mid$
 $ghole-num-bef-pos (i \# q) (GMFun f Cs) = sum-list (map\ num-gholes (take\ i\ Cs)) + ghole-num-bef-pos\ q (Cs ! i)$

fun *ghole-num-at-pos* **where**
 $ghole-num-at-pos \square C = num-gholes\ C \mid$
 $ghole-num-at-pos (i \# q) (GMFun f Cs) = ghole-num-at-pos\ q (Cs ! i)$

fun *subgm-at* :: '*f gmctxt* \Rightarrow *pos* \Rightarrow '*f gmctxt* **where**
 $subgm-at\ C \square = C$
 $\mid subgm-at (GMFun f Cs) (i \# p) = subgm-at (Cs ! i) p$

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* \Rightarrow '*a gmctxt* \Rightarrow '*a gmctxt* **where**
 $GMHole \sqcap D = GMHole \mid$
 $C \sqcap GMHole = GMHole \mid$
 $GMFun f Cs \sqcap GMFun g Ds =$
 (*if* $f = g \wedge length\ Cs = length\ Ds$ *then* $GMFun f (map (case-prod (\sqcap)) (zip\ Cs\ Ds))$
else $GMHole$)

instance ⟨*proof*⟩
end

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

fun *sup-gmctxt* :: '*a gmctxt* \Rightarrow '*a gmctxt* \Rightarrow '*a gmctxt* **where**
 $GMHole \sqcup D = D \mid$
 $C \sqcup GMHole = C \mid$
 $GMFun f Cs \sqcup GMFun g Ds =$
 (*if* $f = g \wedge length\ Cs = length\ Ds$ *then* $GMFun f (map (case-prod (\sqcup)) (zip\ Cs\ Ds))$
else $GMHole$)

else undefined)

instance $\langle \text{proof} \rangle$
end

2.2.3 Orderings and compatibility of ground multihole contexts

inductive *less-eq-gmctxt* :: '*f gmctxt* \Rightarrow '*f gmctxt* \Rightarrow *bool* **where**

base [*simp*]: *less-eq-gmctxt* *GMHole* *u*
| *ind*[*intro*]: $\text{length } cs = \text{length } ds \Longrightarrow (\bigwedge i. i < \text{length } cs \Longrightarrow \text{less-eq-gmctxt } (cs ! i) (ds ! i)) \Longrightarrow$
 $\text{less-eq-gmctxt } (GMFun f cs) (GMFun f ds)$

inductive-set *comp-gmctxt* :: ('*f gmctxt* \times '*f gmctxt*) *set* **where**

GMHole1 [*simp*]: $(GMHole, D) \in \text{comp-gmctxt}$ |
GMHole2 [*simp*]: $(C, GMHole) \in \text{comp-gmctxt}$ |
GMFun [*intro*]: $f = g \Longrightarrow \text{length } Cs = \text{length } Ds \Longrightarrow \forall i < \text{length } Ds. (Cs ! i, Ds ! i) \in \text{comp-gmctxt} \Longrightarrow$
 $(GMFun f Cs, GMFun g Ds) \in \text{comp-gmctxt}$

definition *gmctxt-closing* **where**

$\text{gmctxt-closing } C D \longleftrightarrow \text{less-eq-gmctxt } C D \wedge \text{ghole-poss } D \subseteq \text{ghole-poss } C$

inductive *eq-gfill* ((*/ =_{Gf} -*) [*51, 51*] *50*) **where**

eqfI [*intro*]: $t = \text{fill-gholes } D ss \Longrightarrow \text{num-gholes } D = \text{length } ss \Longrightarrow t =_{Gf} (D, ss)$

2.2.4 Conversions from and to ground multihole contexts

lemma *num-gholes-o-gmctxt-of-gterm* [*simp*]:

$\text{num-gholes} \circ \text{gmctxt-of-gterm} = (\lambda x. 0)$
 $\langle \text{proof} \rangle$

lemma *mctxt-of-term-term-of-mctxt-id* [*simp*]:

$\text{num-gholes } C = 0 \Longrightarrow \text{gmctxt-of-gterm } (\text{gterm-of-gmctxt } C) = C$
 $\langle \text{proof} \rangle$

lemma *num-gholes-mctxt-of-term* [*simp*]:

$\text{num-gholes } (\text{gmctxt-of-gterm } t) = 0$
 $\langle \text{proof} \rangle$

lemma *num-gholes-gmctxt-of-mctxt* [*simp*]:

$\text{ground-mctxt } C \Longrightarrow \text{num-gholes } (\text{gmctxt-of-mctxt } C) = \text{num-gholes } C$
 $\langle \text{proof} \rangle$

lemma *num-gholes-mctxt-of-gmctxt* [*simp*]:

$\text{num-gholes } (\text{mctxt-of-gmctxt } C) = \text{num-gholes } C$
 $\langle \text{proof} \rangle$

lemma *num-gholes-mctxt-of-gmctxt-fun-comp* [*simp*]:

$num\text{-}holes \circ mctxt\text{-}of\text{-}gmctxt = num\text{-}gholes$
 $\langle proof \rangle$

lemma $gmctxt\text{-}of\text{-}gctxt\text{-}num\text{-}gholes$ [simp]:
 $num\text{-}gholes (gmctxt\text{-}of\text{-}gctxt C) = Suc\ 0$
 $\langle proof \rangle$

lemma $ground\text{-}mctxt\text{-}list\text{-}num\text{-}gholes\text{-}gmctxt\text{-}of\text{-}mctxt\text{-}conv$ [simp]:
 $\forall x \in set\ Cs. ground\text{-}mctxt\ x \implies map (num\text{-}gholes \circ gmctxt\text{-}of\text{-}mctxt) Cs = map$
 $num\text{-}holes Cs$
 $\langle proof \rangle$

lemma $num\text{-}gholes\text{-}map\text{-}gmctxt$ [simp]:
 $num\text{-}gholes (map\text{-}gmctxt f C) = num\text{-}gholes C$
 $\langle proof \rangle$

lemma $map\text{-}num\text{-}gholes\text{-}map\text{-}gmctxt$ [simp]:
 $map (num\text{-}gholes \circ map\text{-}gmctxt f) Cs = map\ num\text{-}holes Cs$
 $\langle proof \rangle$

lemma $gterm\text{-}of\text{-}gmctxt\text{-}gmctxt\text{-}of\text{-}gterm\text{-}id$ [simp]:
 $gterm\text{-}of\text{-}gmctxt (gmctxt\text{-}of\text{-}gterm t) = t$
 $\langle proof \rangle$

lemma $no\text{-}gholes\text{-}gmctxt\text{-}of\text{-}gterm\text{-}gterm\text{-}of\text{-}gmctxt\text{-}id$ [simp]:
 $num\text{-}gholes C = 0 \implies gmctxt\text{-}of\text{-}gterm (gterm\text{-}of\text{-}gmctxt C) = C$
 $\langle proof \rangle$

lemma $no\text{-}gholes\text{-}term\text{-}of\text{-}gterm\text{-}gterm\text{-}of\text{-}gmctxt$ [simp]:
 $num\text{-}gholes C = 0 \implies term\text{-}of\text{-}gterm (gterm\text{-}of\text{-}gmctxt C) = term\text{-}of\text{-}gmctxt C$
 $\langle proof \rangle$

lemma $no\text{-}gholes\text{-}term\text{-}of\text{-}mctxt\text{-}mctxt\text{-}of\text{-}gmctxt$ [simp]:
 $num\text{-}gholes C = 0 \implies term\text{-}of\text{-}mctxt (mctxt\text{-}of\text{-}gmctxt C) = term\text{-}of\text{-}gmctxt C$
 $\langle proof \rangle$

lemma $nth\text{While}\text{-}gmctxt\text{-}of\text{-}gctxt$ [simp]:
 $length (take\text{While} (\lambda C. num\text{-}gholes C = 0) (map\ gmctxt\text{-}of\text{-}gterm ss @ gmctxt\text{-}of\text{-}gctxt C \# ts)) = length\ ss$
 $\langle proof \rangle$

lemma $sum\text{-}list\text{-}nth\text{While}\text{-}length$ [simp]:
 $sum\text{-}list (map\ num\text{-}gholes Cs) = Suc\ 0 \implies length (take\text{While} (\lambda C. num\text{-}gholes$
 $C = 0) Cs) < length\ Cs$
 $\langle proof \rangle$

lemma $gctxt\text{-}of\text{-}gmctxt\text{-}gmctxt\text{-}of\text{-}gctxt$ [simp]:
 $gctxt\text{-}of\text{-}gmctxt (gmctxt\text{-}of\text{-}gctxt C) = C$

<proof>

lemma *gmctxt-of-gctxt-GMHole-Hole*:

gmctxt-of-gctxt C = GMHole \implies C = \square_G

<proof>

lemma *gmctxt-of-gctxt-gctxt-of-gmctxt*:

num-gholes C = Suc 0 \implies gmctxt-of-gctxt (gctxt-of-gmctxt C) = C

<proof>

lemma *inj-gmctxt-of-gctxt: inj gmctxt-of-gctxt*

<proof>

lemma *inj-gctxt-of-gmctxt-on-single-hole*:

inj-on gctxt-of-gmctxt (Collect (λ C. num-gholes C = Suc 0))

<proof>

lemma *gctxt-of-gmctxt-hole-dest*:

num-gholes C = Suc 0 \implies gctxt-of-gmctxt C = $\square_G \implies$ C = GMHole

<proof>

lemma *mctxt-of-gmctxt-inv [simp]*:

gmctxt-of-mctxt (mctxt-of-gmctxt C) = C

<proof>

lemma *ground-mctxt-of-gmctxt [simp]*:

ground-mctxt (mctxt-of-gmctxt C)

<proof>

lemma *ground-mctxt-of-gmctxt' [simp]*:

mctxt-of-gmctxt C = MFun f D \implies ground-mctxt (MFun f D)

<proof>

lemma *gmctxt-of-mctxt-inv [simp]*:

ground-mctxt C \implies mctxt-of-gmctxt (gmctxt-of-mctxt C) = C

<proof>

lemma *ground-mctxt-of-gmctxtD*:

ground-mctxt C \implies \exists D. C = mctxt-of-gmctxt D

<proof>

lemma *inj-mctxt-of-gmctxt: inj-on mctxt-of-gmctxt X*

<proof>

lemma *inj-gmctxt-of-mctxt-ground*:

inj-on gmctxt-of-mctxt (Collect ground-mctxt)

<proof>

lemma *map-gmctxt-comp [simp]*:

$map-gmctxt\ f\ (map-gmctxt\ g\ C) = map-gmctxt\ (f \circ g)\ C$
 ⟨proof⟩

lemma *map-mctxt-of-gmctxt*:
 $map-mctxt\ f\ (mctxt-of-gmctxt\ C) = mctxt-of-gmctxt\ (map-gmctxt\ f\ C)$
 ⟨proof⟩

lemma *map-gmctxt-of-mctxt*:
 $ground-mctxt\ C \implies map-gmctxt\ f\ (gmctxt-of-mctxt\ C) = gmctxt-of-mctxt\ (map-mctxt\ f\ C)$
 ⟨proof⟩

lemma *map-gmctxt-nempty* [simp]:
 $C \neq GMHole \implies map-gmctxt\ f\ C \neq GMHole$
 ⟨proof⟩

lemma *vars-mctxt-of-gmctxt* [simp]:
 $vars-mctxt\ (mctxt-of-gmctxt\ C) = \{\}$
 ⟨proof⟩

lemma *vars-mctxt-of-gmctxt-subseteq* [simp]:
 $vars-mctxt\ (mctxt-of-gmctxt\ C) \subseteq Q \longleftrightarrow True$
 ⟨proof⟩

2.2.5 Equivalences and simplification rules

lemma *eqgfE*:
assumes $t =_{Gf}\ (D, ss)$ **shows** $t = fill-gholes\ D\ ss\ num-gholes\ D = length\ ss$
 ⟨proof⟩

lemma *eqgf-GMHoleE*:
assumes $t =_{Gf}\ (GMHole, ss)$ **shows** $ss = [t]$ ⟨proof⟩

lemma *eqgf-GMFunE*:
assumes $s =_{Gf}\ (GMFun\ f\ Cs, ss)$
obtains $ts\ sss$ **where** $s = GFun\ f\ ts\ length\ ts = length\ Cs\ length\ sss = length\ Cs$
 $\bigwedge i. i < length\ Cs \implies ts\ !\ i =_{Gf}\ (Cs\ !\ i, sss\ !\ i)\ ss = concat\ sss$
 ⟨proof⟩

lemma *partition-holes-subseteq* [simp]:
assumes $sum-list\ (map\ num-holes\ Cs) = length\ xs\ i < length\ Cs$
and $x \in set\ (partition-holes\ xs\ Cs\ !\ i)$
shows $x \in set\ xs$
 ⟨proof⟩

lemma *partition-gholes-subseteq* [simp]:
assumes $sum-list\ (map\ num-gholes\ Cs) = length\ xs\ i < length\ Cs$

and $x \in \text{set } (\text{partition-gholes } xs \text{ } Cs ! i)$
shows $x \in \text{set } xs$
 $\langle \text{proof} \rangle$

lemma *list-elem-to-partition-nth* [elim]:
assumes $\text{sum-list } (\text{map } \text{num-gholes } Cs) = \text{length } xs \text{ } x \in \text{set } xs$
obtains i **where** $i < \text{length } Cs \text{ } x \in \text{set } (\text{partition-gholes } xs \text{ } Cs ! i)$ $\langle \text{proof} \rangle$

lemma *partition-gholes-fill-gholes-conv'*:
 $\text{fill-gholes } (GMFun \text{ } f \text{ } Cs) \text{ } ts =$
 $GMFun \text{ } f \text{ } (\text{map } (\text{case-prod } \text{fill-gholes}) (\text{zip } Cs \text{ } (\text{partition-gholes } ts \text{ } Cs)))$
 $\langle \text{proof} \rangle$

lemma *unfill-gholes-conv*:
assumes $\text{length } Cs = \text{length } ts$
shows $\text{unfill-gholes } (GMFun \text{ } f \text{ } Cs) \text{ } (GMFun \text{ } f \text{ } ts) =$
 $\text{concat } (\text{map } (\text{case-prod } \text{unfill-gholes}) (\text{zip } Cs \text{ } ts))$ $\langle \text{proof} \rangle$

lemma *partition-gholes-fill-gholes-gmctxt-conv*:
 $\text{fill-gholes-gmctxt } (GMFun \text{ } f \text{ } Cs) \text{ } ts =$
 $GMFun \text{ } f \text{ } [\text{fill-gholes-gmctxt } (Cs ! i) \text{ } (\text{partition-gholes } ts \text{ } Cs ! i). i \leftarrow [0 .. <$
 $\text{length } Cs]]$
 $\langle \text{proof} \rangle$

lemma *partition-gholes-fill-gholes-gmctxt-conv'*:
 $\text{fill-gholes-gmctxt } (GMFun \text{ } f \text{ } Cs) \text{ } ts =$
 $GMFun \text{ } f \text{ } (\text{map } (\text{case-prod } \text{fill-gholes-gmctxt}) (\text{zip } Cs \text{ } (\text{partition-gholes } ts \text{ } Cs)))$
 $\langle \text{proof} \rangle$

lemma *fill-gholes-no-gholes* [simp]:
 $\text{num-gholes } C = 0 \implies \text{fill-gholes } C \text{ } [] = \text{gterm-of-gmctxt } C$
 $\langle \text{proof} \rangle$

lemma *fill-gholes-gmctxt-no-gholes* [simp]:
 $\text{num-gholes } C = 0 \implies \text{fill-gholes-gmctxt } C \text{ } [] = C$
 $\langle \text{proof} \rangle$

lemma *eqgf-GMFunI*:
assumes $\bigwedge i. i < \text{length } Cs \implies ss ! i =_{Gf} (Cs ! i, ts ! i)$
and $\text{length } Cs = \text{length } ss \text{ } \text{length } ss = \text{length } ts$
shows $GMFun \text{ } f \text{ } ss =_{Gf} (GMFun \text{ } f \text{ } Cs, \text{concat } ts)$ $\langle \text{proof} \rangle$

lemma *length-partition-gholes-nth*:
assumes $\text{sum-list } (\text{map } \text{num-gholes } cs) = \text{length } ts$
and $i < \text{length } cs$
shows $\text{length } (\text{partition-gholes } ts \text{ } cs ! i) = \text{num-gholes } (cs ! i)$
 $\langle \text{proof} \rangle$

lemma *fill-gholes-induct2*[consumes 2, case-names GMHole GMFun]:

fixes $P :: 'f \text{ gmctxt} \Rightarrow 'a \text{ list} \Rightarrow 'b \text{ list} \Rightarrow \text{bool}$
assumes $\text{len1: num-gholes } C = \text{length } xs$ **and** $\text{len2: num-gholes } C = \text{length } ys$
and $\text{Hole: } \bigwedge x y. P \text{ GMHole } [x] [y]$
and $\text{Fun: } \bigwedge f Cs xs ys. \text{sum-list } (\text{map num-gholes } Cs) = \text{length } xs \implies$
 $\text{sum-list } (\text{map num-gholes } Cs) = \text{length } ys \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P (Cs ! i) (\text{partition-gholes } xs \text{ } Cs ! i) (\text{partition-gholes } ys \text{ } Cs ! i)) \implies$
 $P (\text{GMFun } f \text{ } Cs) (\text{concat } (\text{partition-gholes } xs \text{ } Cs)) (\text{concat } (\text{partition-gholes } ys \text{ } Cs))$
shows $P \text{ } C \text{ } xs \text{ } ys$
 $\langle \text{proof} \rangle$

lemma $\text{fill-gholes-induct}[\text{consumes } 1, \text{case-names GMHole GMFun}]$:
fixes $P :: 'f \text{ gmctxt} \Rightarrow 'a \text{ list} \Rightarrow \text{bool}$
assumes $\text{len: num-gholes } C = \text{length } xs$
and $\text{Hole: } \bigwedge x. P \text{ GMHole } [x]$
and $\text{Fun: } \bigwedge f Cs xs. \text{sum-list } (\text{map num-gholes } Cs) = \text{length } xs \implies$
 $(\bigwedge i. i < \text{length } Cs \implies P (Cs ! i) (\text{partition-gholes } xs \text{ } Cs ! i)) \implies$
 $P (\text{GMFun } f \text{ } Cs) (\text{concat } (\text{partition-gholes } xs \text{ } Cs))$
shows $P \text{ } C \text{ } xs$
 $\langle \text{proof} \rangle$

lemma $\text{eq-gfill-induct} [\text{consumes } 1, \text{case-names GMHole GMFun}]$:
assumes $t =_{Gf} (C, ts)$
and $\bigwedge t. P \text{ } t \text{ GMHole } [t]$
and $\bigwedge f ss Cs ts. \llbracket \text{length } Cs = \text{length } ss; \text{sum-list } (\text{map num-gholes } Cs) = \text{length } ts; \forall i < \text{length } ss. ss ! i =_{Gf} (Cs ! i, \text{partition-gholes } ts \text{ } Cs ! i) \wedge P (ss ! i) (Cs ! i) (\text{partition-gholes } ts \text{ } Cs ! i) \rrbracket \implies P (\text{GFun } f \text{ } ss) (\text{GMFun } f \text{ } Cs) \text{ } ts$
shows $P \text{ } t \text{ } C \text{ } ts \langle \text{proof} \rangle$

lemma $\text{nempty-ground-mctxt-gmctxt} [\text{simp}]$:
 $C \neq \text{MHole} \implies \text{ground-mctxt } C \implies \text{gmctxt-of-mctxt } C \neq \text{GMHole}$
 $\langle \text{proof} \rangle$

lemma $\text{mctxt-of-gmctxt-fill-holes} [\text{simp}]$:
assumes $\text{num-gholes } C = \text{length } ss$
shows $\text{gterm-of-term } (\text{fill-holes } (\text{mctxt-of-gmctxt } C) (\text{map term-of-gterm } ss)) = \text{fill-gholes } C \text{ } ss \langle \text{proof} \rangle$

lemma $\text{mctxt-of-gmctxt-terms-fill-holes}$:
assumes $\text{num-gholes } C = \text{length } ss$
shows $\text{gterm-of-term } (\text{fill-holes } (\text{mctxt-of-gmctxt } C) \text{ } ss) = \text{fill-gholes } C (\text{map } \text{gterm-of-term } \text{ } ss) \langle \text{proof} \rangle$

lemma $\text{ground-gmctxt-of-mctxt-gterm-fill-holes}$:
assumes $\text{num-gholes } C = \text{length } ss$ **and** $\text{ground-mctxt } C$
shows $\text{term-of-gterm } (\text{fill-gholes } (\text{gmctxt-of-mctxt } C) \text{ } ss) = \text{fill-holes } C (\text{map } \text{term-of-gterm } \text{ } ss) \langle \text{proof} \rangle$

term-of-gterm ss \langle proof \rangle

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* \langle proof \rangle

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* \langle proof \rangle

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)* \langle proof \rangle

lemma *fill-gholes-gmctxt-of-gterm* [simp]:

fill-gholes (gmctxt-of-gterm s) [] = s

\langle proof \rangle

lemma *fill-gholes-GMHole* [simp]:

length ss = Suc 0 \implies fill-gholes GMHole ss = ss ! 0

\langle proof \rangle

lemma *apply-gctxt-fill-gholes*:

C \langle s \rangle_G = fill-gholes (gmctxt-of-gctxt C) [s]

\langle proof \rangle

lemma *fill-gholes-apply-gctxt*:

num-gholes C = Suc 0 \implies fill-gholes C [s] = (gctxt-of-gmctxt C) \langle s \rangle_G

\langle proof \rangle

lemma *ctxt-of-gctxt-gctxt-of-gmctxt-apply*:

num-gholes C = Suc 0 \implies fill-holes (mctxt-of-gmctxt C) [s] = (ctxt-of-gctxt (gctxt-of-gmctxt C)) \langle s \rangle

\langle proof \rangle

lemma *fill-gholes-replicate* [simp]:

n = length ss \implies fill-gholes (GMFun f (replicate n GMHole)) ss = GFun f ss

\langle proof \rangle

lemma *fill-gholes-gmctxt-replicate-MHole* [simp]:

fill-gholes-gmctxt C (replicate (num-gholes C) GMHole) = C

\langle proof \rangle

lemma *fill-gholes-gmctxt-GMFun-replicate-length* [simp]:
fill-gholes-gmctxt (GMFun f (replicate (length Cs) GMHole)) Cs = GMFun f Cs
 ⟨proof⟩

lemma *fill-gholes-gmctxt-MFun*:
assumes *lCs: length Cs = length ts*
and *lss: length ss = length ts*
and *rec: $\bigwedge i. i < \text{length } ts \implies \text{num-gholes } (Cs ! i) = \text{length } (ss ! i) \wedge$*
fill-gholes-gmctxt (Cs ! i) (ss ! i) = ts ! i
shows *fill-gholes-gmctxt (GMFun f Cs) (concat ss) = GMFun f ts*
 ⟨proof⟩

lemma *fill-gholes-gmctxt-nHole* [simp]:
C \neq GMHole $\implies \text{num-gholes } C = \text{length } Ds \implies \text{fill-gholes-gmctxt } C Ds \neq$
GMHole
 ⟨proof⟩

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)*
 ⟨proof⟩

lemma *num-gholes-greater0-fill-gholes-gmctxt* [intro!]:
assumes *num-gholes C = length Ds*
and $\exists D \in \text{set } Ds. 0 < \text{num-gholes } D$
shows $0 < \text{sum-list } (\text{map } \text{num-gholes } Ds)$
 ⟨proof⟩

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 \leftarrow [0 ..< num-gholes
C]]
 ⟨proof⟩

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: $\bigwedge i. i < \text{length } Ds \implies ts ! i =_{Gf} (Ds ! i, sss ! i)$*
shows *fill-gholes C ts =_{Gf} (fill-gholes-gmctxt C Ds, concat sss)*
 ⟨proof⟩

2.2.6 Semilattice Structures

lemma *inf-gmctxt-idem* [simp]:
(C :: 'f gmctxt) \sqcap C = C
 ⟨proof⟩

lemma *inf-gmctxt-GMHole2* [simp]:

$$C \sqcap \text{GMHole} = \text{GMHole}$$

<proof>

lemma *inf-gmctxt-comm* [ac-simps]:

$$(C :: 'f \text{ gmctxt}) \sqcap D = D \sqcap C$$

<proof>

lemma *inf-gmctxt-assoc* [ac-simps]:

fixes $C :: 'f \text{ gmctxt}$

shows $C \sqcap D \sqcap E = C \sqcap (D \sqcap E)$

<proof>

instantiation *gmctxt* :: (type) order

begin

definition $(C :: 'a \text{ gmctxt}) \leq D \iff C \sqcap D = C$

definition $(C :: 'a \text{ gmctxt}) < D \iff C \leq D \wedge \neg D \leq C$

instance

<proof>

end

lemma *less-eq-gmctxt-prime*: $C \leq D \iff \text{less-eq-gmctxt } C \ D$

<proof>

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*:

$$\text{less-eq-gmctxt } C \ \text{GMHole} \implies C = \text{GMHole}$$

<proof>

lemma *num-gholes-at-least1*:

$$0 < \text{num-gholes } C \implies 0 < \text{num-gholes } (C \sqcap D)$$

<proof>

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

instance *gmctxt* :: (type) *semilattice-inf*

<proof>

lemma *sup-gmctxt-idem* [simp]:

fixes $C :: 'f \text{ gmctxt}$

shows $C \sqcup C = C$

<proof>

lemma *sup-gmctxt-MHole* [*simp*]: $C \sqcup \text{GMHole} = C$
<proof>

lemma *sup-gmctxt-comm* [*ac-simps*]:
fixes $C :: 'f \text{ gmctxt}$
shows $C \sqcup D = D \sqcup C$
<proof>

lemma *comp-gmctxt-refl*:
 $(C, C) \in \text{comp-gmctxt}$
<proof>

lemma *comp-gmctxt-sym*:
assumes $(C, D) \in \text{comp-gmctxt}$
shows $(D, C) \in \text{comp-gmctxt}$
<proof>

lemma *sup-gmctxt-assoc* [*ac-simps*]:
assumes $(C, D) \in \text{comp-gmctxt}$ **and** $(D, E) \in \text{comp-gmctxt}$
shows $C \sqcup D \sqcup E = C \sqcup (D \sqcup E)$
<proof>

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

interpretation *gmctxt-order-bot*: *order-bot GMHole* (\leq) ($<$)
<proof>

lemma *sup-gmctxt-ge1* [*simp*]:
assumes $(C, D) \in \text{comp-gmctxt}$
shows $C \leq C \sqcup D$
<proof>

lemma *sup-gmctxt-ge2* [*simp*]:
assumes $(C, D) \in \text{comp-gmctxt}$
shows $D \leq C \sqcup D$
<proof>

lemma *sup-gmctxt-least*:
assumes $(D, E) \in \text{comp-gmctxt}$
and $D \leq C$ **and** $E \leq C$
shows $D \sqcup E \leq C$
<proof>

lemma *sup-gmctxt-args-MHole2* [*simp*]:
sup-gmctxt-args C GMHole = replicate (num-gholes C) GMHole
<proof>

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

assumes $(C, D) \in \text{comp-gmctxt}$

shows $\text{num-gholes } C = \text{length } (\text{sup-gmctxt-args } C D)$

$\langle \text{proof} \rangle$

lemma *sup-gmctxt-sup-gmctxt-args*:

assumes $(C, D) \in \text{comp-gmctxt}$

shows $\text{fill-gholes-gmctxt } C (\text{sup-gmctxt-args } C D) = C \sqcup D \langle \text{proof} \rangle$

lemma *eqgf-comp-gmctxt*:

assumes $s =_{Gf} (C, ss)$ **and** $s =_{Gf} (D, ts)$

shows $(C, D) \in \text{comp-gmctxt} \langle \text{proof} \rangle$

lemma *eqgf-less-eq [simp]*:

assumes $s =_{Gf} (C, ss)$

shows $C \leq \text{gmctxt-of-gterm } s \langle \text{proof} \rangle$

lemma *less-eq-comp-gmctxt [simp]*:

$C \leq D \implies (C, D) \in \text{comp-gmctxt}$

$\langle \text{proof} \rangle$

lemma *gmctxt-less-eq-sup*:

$(C :: 'f \text{ gmctxt}) \leq D \implies C \sqcup D = D$

$\langle \text{proof} \rangle$

lemma *fill-gholes-gmctxt-less-eq*:

assumes $\text{num-gholes } C = \text{length } Ds$

shows $C \leq \text{fill-gholes-gmctxt } C Ds \langle \text{proof} \rangle$

lemma *less-eq-to-sup-mctxt-args [elim]*:

assumes $C \leq D$

obtains Ds **where** $\text{num-gholes } C = \text{length } Ds$ $D = \text{fill-gholes-gmctxt } C Ds$

$\langle \text{proof} \rangle$

lemma *fill-gholes-gmctxt-sup-mctxt-args [simp]*:

assumes $\text{num-gholes } C = \text{length } Ds$

shows $\text{sup-gmctxt-args } C (\text{fill-gholes-gmctxt } C Ds) = Ds \langle \text{proof} \rangle$

lemma *map2-fill-gholes-gmctxt-id [simp]*:

assumes $\bigwedge i. i < \text{length } Ds \implies \text{num-gholes } (Ds ! i) = 0$

shows $\text{map2 fill-gholes-gmctxt } Ds (\text{replicate } (\text{length } Ds) []) = Ds$

$\langle \text{proof} \rangle$

lemma *fill-gholes-gmctxt-GMFun-replicate-append [simp]*:

assumes $\text{length } Cs = n$ **and** $\bigwedge t. t \in \text{set } Ds \implies \text{num-gholes } t = 0$

shows $\text{fill-gholes-gmctxt } (\text{GMFun } f ((\text{replicate } n \text{ GMHole}) @ Ds)) Cs = \text{GMFun } f (Cs @ Ds) \langle \text{proof} \rangle$

lemma *finite-ghole-poss*:

finite (*ghole-poss* *C*)
<proof>

lemma *ghole-poss-simp* [*simp*]:

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

lemma *num-gholes-zero-ghole-poss*:

num-gholes *D* = 0 ⇒ *ghole-poss* *D* = {}
<proof>

lemma *ghole-poss-num-gholes-zero*:

ghole-poss *D* = {} ⇒ *num-gholes* *D* = 0
<proof>

lemma *num-gholes-nzero-ghole-poss-nempty*:

num-gholes *D* ≠ 0 ⇒ *ghole-poss* *D* ≠ {}
<proof>

lemma *ghole-poss-epsE* [*elim*]:

ghole-poss *D* = {} ⇒ *D* = *GMHole*
<proof>

lemma *ghole-poss-gmctxt-of-gterm* [*simp*]:

ghole-poss (*gmctxt-of-gterm* *t*) = {}
<proof>

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) *<proof>*

lemma *factor-ghole-pos-by-prefix*:

assumes *C* ≤ *D* *p* ∈ *ghole-poss* *D*
obtains *q* **where** *q* ≤_{*p*} *p* *q* ∈ *ghole-poss* *C*
<proof>

lemma *prefix-and-fewer-gholes-implies-equal-gmctxt*:

C ≤ *D* ⇒ *ghole-poss* *C* ⊆ *ghole-poss* *D* ⇒ *C* = *D*
<proof>

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*)))
<proof>

lemma *gmctxt-closing-trans*:

$gmctxt-closing\ C\ D \implies gmctxt-closing\ D\ E \implies gmctxt-closing\ C\ E$
<proof>

lemma *gmctxt-closing-sup-args-ghole-or-gterm*:

assumes $gmctxt-closing\ C\ D$

shows $\forall E \in set\ (sup-gmctxt-args\ C\ D). E = GMHole \vee num-gholes\ E = 0$
<proof>

lemma *inv-imples-ghole-poss-subseteq*:

$C \leq D \implies \forall E \in set\ (sup-gmctxt-args\ C\ D). E = GMHole \vee num-gholes\ E = 0 \implies ghole-poss\ D \subseteq ghole-poss\ C$
<proof>

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 $ghole-poss\ (fill-gholes-gmctxt\ C\ Ds) \subseteq ghole-poss\ C$ *<proof>*

lemma *ghole-poss-not-in-poss-gmctxt*:

assumes $p \in ghole-poss\ C$

shows $p \notin poss-gmctxt\ C$ *<proof>*

lemma *comp-gmctxt-inf-ghole-poss-cases*:

assumes $(C, D) \in comp-gmctxt\ p \in ghole-poss\ (C \sqcap D)$

shows $p \in ghole-poss\ C \wedge p \in ghole-poss\ D \vee$

$p \in ghole-poss\ C \wedge p \in poss-gmctxt\ D \vee$

$p \in ghole-poss\ D \wedge p \in poss-gmctxt\ C$ *<proof>*

lemma *length-ghole-poss-list-num-gholes*:

$num-gholes\ C = length\ (ghole-poss-list\ C)$

<proof>

lemma *ghole-poss-list-distinct*:

$distinct\ (ghole-poss-list\ C)$

<proof>

lemma *ghole-poss-ghole-poss-list-conv*:

$ghole-poss\ C = set\ (ghole-poss-list\ C)$

<proof>

lemma *card-ghole-poss-num-gholes*:

$card\ (ghole-poss\ C) = num-gholes\ C$

<proof>

lemma *subgm-at-hole-poss [simp]*:

$p \in ghole-poss\ C \implies subgm-at\ C\ p = GMHole$

<proof>

lemma *subgm-at-mctxt-of-term*:

$p \in gposs\ t \implies subgm-at\ (gmctxt-of-gterm\ t)\ p = gmctxt-of-gterm\ (gsubt-at\ t\ p)$
 $\langle proof \rangle$

lemma *num-gholes-subgm-at*:

assumes $p \in poss-gmctxt\ C$
shows $num-gholes\ (subgm-at\ C\ p) = ghole-num-at-pos\ p\ C\ \langle proof \rangle$

lemma *gmctxt-subtgm-at-fill-args-empty-pos [simp]*:

assumes $num-gholes\ C = length\ ts$
shows $gmctxt-subtgm-at-fill-args\ []\ C\ ts = ts$
 $\langle proof \rangle$

lemma *ghole-num-bef-at-pos-num-gholes-less-eq*:

assumes $p \in poss-gmctxt\ C$
shows $ghole-num-bef-pos\ p\ C + ghole-num-at-pos\ p\ C \leq num-gholes\ C\ \langle proof \rangle$

lemma *ghole-num-at-pos-fill-args-length*:

assumes $p \in poss-gmctxt\ C\ num-gholes\ C = length\ ts$
shows $ghole-num-at-pos\ p\ C = length\ (gmctxt-subtgm-at-fill-args\ p\ C\ ts)$
 $\langle proof \rangle$

lemma *ghole-poss-nth-subt-at*:

assumes $t =_{Gf}\ (C,\ ts)$ **and** $p \in ghole-poss\ C$
shows $ghole-num-bef-pos\ p\ C < length\ ts \wedge gsubt-at\ t\ p = ts\ !\ ghole-num-bef-pos\ p\ C\ \langle proof \rangle$

lemma *poss-gmctxt-fill-gholes-split*:

assumes $t =_{Gf}\ (C,\ ts)$ **and** $p \in poss-gmctxt\ C$
shows $gsubt-at\ t\ p =_{Gf}\ (subgm-at\ C\ p,\ gmctxt-subtgm-at-fill-args\ p\ C\ ts)$
 $\langle proof \rangle$

lemma *fill-gholes-ghole-poss*:

assumes $t =_{Gf}\ (C,\ ts)$ **and** $i < length\ ts$
shows $gsubt-at\ t\ (ghole-poss-list\ C\ !\ i) = ts\ !\ i\ \langle proof \rangle$

lemma *length-unfill-gholes [simp]*:

assumes $C \leq gmctxt-of-gterm\ t$
shows $length\ (unfill-gholes\ C\ t) = num-gholes\ C$
 $\langle proof \rangle$

lemma *fill-gholes-arbitrary*:

assumes $lCs: length\ Cs = length\ ts$
and $lss: length\ ss = length\ ts$
and $rec: \bigwedge i. i < length\ ts \implies num-gholes\ (Cs\ !\ i) = length\ (ss\ !\ i) \wedge f\ (Cs\ !\ i)\ (ss\ !\ i) = ts\ !\ i$
shows $map\ (\lambda i. f\ (Cs\ !\ i)\ (partition-gholes\ (concat\ ss)\ Cs\ !\ i))\ [0..< length\ Cs] = ts$
 $\langle proof \rangle$

lemma *fill-unfill-gholes*:

assumes $C \leq \text{gmctxt-of-gterm } t$

shows $\text{fill-gholes } C (\text{unfill-gholes } C t) = t$

<proof>

lemma *funas-gmctxt-of-mctxt [simp]*:

$\text{ground-mctxt } C \implies \text{funas-gmctxt } (\text{gmctxt-of-mctxt } C) = \text{funas-mctxt } C$

<proof>

lemma *funas-mctxt-of-gmctxt-conv*:

$\text{funas-mctxt } (\text{mctxt-of-gmctxt } C) = \text{funas-gmctxt } C$

<proof>

lemma *funas-gterm-ctxt-apply [simp]*:

assumes $\text{num-gholes } C = \text{length } ss$

shows $\text{funas-gterm } (\text{fill-gholes } C ss) = \text{funas-gmctxt } C \cup \bigcup (\text{set } (\text{map } \text{funas-gterm } ss))$ *<proof>*

lemma *funas-gmctxt-gmctxt-of-gterm [simp]*:

$\text{funas-gmctxt } (\text{gmctxt-of-gterm } s) = \text{funas-gterm } s$

<proof>

lemma *funas-gmctxt-replicate-GMHole [simp]*:

$\text{funas-gmctxt } (\text{GMFun } f (\text{replicate } n \text{ GMHole})) = \{(f, n)\}$

<proof>

lemma *funas-gmctxt-gmctxt-of-gctxt [simp]*:

$\text{funas-gmctxt } (\text{gmctxt-of-gctxt } C) = \text{funas-gctxt } C$

<proof>

lemma *funas-gmctxt-fill-gholes-gmctxt [simp]*:

assumes $\text{num-gholes } C = \text{length } Ds$

shows $\text{funas-gmctxt } (\text{fill-gholes-gmctxt } C Ds) = \text{funas-gmctxt } C \cup \bigcup (\text{set } (\text{map } \text{funas-gmctxt } Ds))$

(is ?f C Ds = ?g C Ds) *<proof>*

lemma *funas-supremum*:

$C \leq D \implies \text{funas-gmctxt } D = \text{funas-gmctxt } C \cup \bigcup (\text{set } (\text{map } \text{funas-gmctxt } (\text{sup-gmctxt-args } C D)))$

<proof>

lemma *funas-gctxt-gctxt-of-gmctxt [simp]*:

$\text{num-gholes } D = \text{Suc } 0 \implies \text{funas-gctxt } (\text{gctxt-of-gmctxt } D) = \text{funas-gmctxt } D$

<proof>

lemma *funas-gterm-gterm-of-gmctxt [simp]*:

$\text{num-gholes } C = 0 \implies \text{funas-gterm } (\text{gterm-of-gmctxt } C) = \text{funas-gmctxt } C$

<proof>

lemma *less-sup-gmctxt-args-funas-gmctxt:*

$C \leq D \implies \text{funas-gmctxt } C \subseteq \mathcal{F} \implies \forall Ds \in \text{set } (\text{sup-gmctxt-args } C D). \text{funas-gmctxt } Ds \subseteq \mathcal{F} \implies \text{funas-gmctxt } D \subseteq \mathcal{F}$
<proof>

lemma *funas-gmctxt-poss-gmctxt-subgm-at-funas:*

assumes $\text{funas-gmctxt } C \subseteq \mathcal{F} \quad p \in \text{poss-gmctxt } C$
shows $\text{funas-gmctxt } (\text{subgm-at } C p) \subseteq \mathcal{F}$
<proof>

lemma *inf-funas-gmctxt-subset1:*

$\text{funas-gmctxt } (C \sqcap D) \subseteq \text{funas-gmctxt } C$
<proof>

lemma *inf-funas-gmctxt-subset2:*

$\text{funas-gmctxt } (C \sqcap D) \subseteq \text{funas-gmctxt } D$
<proof>

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 \Rightarrow '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)} \cup (\bigcup (funas-bot-term ' set ss))

lemma *finite-funas-bot-term:*

finite (funas-bot-term t)
<proof>

lemma *funas-bot-term-funas-term:*

funas-bot-term (t[⊥]) = funas-term t
<proof>

lemma *term-to-bot-term-root-bot* [simp]:

$root\text{-}bot (t^\perp) = root\ t$

$\langle proof \rangle$

lemma *term-to-bot-term-root-bot-comp* [simp]:

$root\text{-}bot \circ term\text{-}to\text{-}bot\text{-}term = root$

$\langle proof \rangle$

inductive-set *mergeP* **where**

base-l [simp]: $(Bot, t) \in mergeP$

| *base-r* [simp]: $(t, Bot) \in mergeP$

| *step* [intro]: $length\ ss = length\ ts \implies (\forall i < length\ ts. (ss ! i, ts ! i) \in mergeP)$

\implies

$(BFun\ f\ ss, BFun\ f\ ts) \in mergeP$

lemma *merge-refl*:

$(s, s) \in mergeP$

$\langle proof \rangle$

lemma *merge-symmetric*:

assumes $(s, t) \in mergeP$

shows $(t, s) \in mergeP$

$\langle proof \rangle$

fun *merge-terms* :: '*f bot-term* \Rightarrow '*f bot-term* \Rightarrow '*f bot-term* (**infixr** \uparrow 67) **where**

Bot $\uparrow s = s$

| *s* $\uparrow Bot = s$

| $(BFun\ f\ ss) \uparrow (BFun\ g\ ts) = (if\ f = g \wedge length\ ss = length\ ts$
 then $BFun\ f\ (map\ (case\text{-}prod\ (\uparrow))\ (zip\ ss\ ts))$
 else *undefined*)

lemma *merge-terms-bot-rhs*[simp]:

$s \uparrow Bot = s$ $\langle proof \rangle$

lemma *merge-terms-idem*: $s \uparrow s = s$

$\langle proof \rangle$

lemma *merge-terms-assoc* [ac-simps]:

assumes $(s, t) \in mergeP$ **and** $(t, u) \in mergeP$

shows $(s \uparrow t) \uparrow u = s \uparrow t \uparrow u$

$\langle proof \rangle$

lemma *merge-terms-commutative* [ac-simps]:

shows $s \uparrow t = t \uparrow s$

$\langle proof \rangle$

lemma *merge-dist*:

assumes $(s, t \uparrow u) \in mergeP$ **and** $(t, u) \in mergeP$

shows $(s, t) \in mergeP$ $\langle proof \rangle$

lemma *mergeP-ass*:

$(s, t \uparrow u) \in \text{mergeP} \implies (t, u) \in \text{mergeP} \implies (s \uparrow t, u) \in \text{mergeP}$
<proof>

inductive-set *bless-eq* **where**

base-l [*simp*]: $(\text{Bot}, t) \in \text{bless-eq}$
| *step* [*intro*]: $\text{length } ss = \text{length } ts \implies (\forall i < \text{length } ts. (ss ! i, ts ! i) \in \text{bless-eq})$
 \implies
 $(\text{BFun } f \text{ } ss, \text{BFun } f \text{ } ts) \in \text{bless-eq}$

Infix syntax.

abbreviation *bless-eq-pred* $s \ t \equiv (s, t) \in \text{bless-eq}$

notation

bless-eq $(\{\leq_b\})$ **and**
bless-eq-pred $((-/ \leq_b -)$ [56, 56] 55)

lemma *BFun-leq-Bot-False* [*simp*]:

$\text{BFun } f \text{ } ts \leq_b \text{Bot} \longleftrightarrow \text{False}$
<proof>

lemma *BFun-lesseqE* [*elim*]:

assumes $\text{BFun } f \text{ } ts \leq_b t$
obtains *us* **where** $\text{length } ts = \text{length } us \ t = \text{BFun } f \text{ } us$
<proof>

lemma *bless-eq-refl*: $s \leq_b s$

<proof>

lemma *bless-eq-trans* [*trans*]:

assumes $s \leq_b t$ **and** $t \leq_b u$
shows $s \leq_b u$ *<proof>*

lemma *bless-eq-anti-sym*:

$s \leq_b t \implies t \leq_b s \implies s = t$
<proof>

lemma *bless-eq-mergeP*:

$s \leq_b t \implies (s, t) \in \text{mergeP}$
<proof>

lemma *merge-bot-args-bless-eq-merge*:

assumes $(s, t) \in \text{mergeP}$
shows $s \leq_b s \uparrow t$ *<proof>*

lemma *bless-eq-closed-under-merge*:

assumes $(s, t) \in \text{mergeP}$ $(u, v) \in \text{mergeP}$ $s \leq_b u$ $t \leq_b v$
shows $s \uparrow t \leq_b u \uparrow v$ *<proof>*

lemma *blees-eq-closed-under-supremum*:

assumes $s \leq_b u$ $t \leq_b u$

shows $s \uparrow t \leq_b u$ *<proof>*

lemma *linear-term-comb-subst*:

assumes *linear-term* (*Fun f ss*)

and $\text{length } ss = \text{length } ts$

and $\bigwedge i. i < \text{length } ts \implies ss ! i \cdot \sigma i = ts ! i$

shows $\exists \sigma. \text{Fun } f \ ss \cdot \sigma = \text{Fun } f \ ts$

<proof>

lemma *blees-eq-to-instance*:

assumes $s^\perp \leq_b t^\perp$ **and** *linear-term s*

shows $\exists \sigma. s \cdot \sigma = t$ *<proof>*

lemma *instance-to-blees-eq*:

assumes $s \cdot \sigma = t$

shows $s^\perp \leq_b t^\perp$ *<proof>*

end

theory *Saturation*

imports *Main*

begin

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

lemma *inv-to-set*:

$(\forall i < \text{length } ss. ss ! i \in S) \longleftrightarrow \text{set } ss \subseteq S$

<proof>

lemma *ac-comp-fun-commute*:

assumes $\bigwedge x y. f x y = f y x$ **and** $\bigwedge x y z. f x (f y z) = f (f x y) z$

shows *comp-fun-commute f* *<proof>*

lemma (**in** *comp-fun-commute*) *fold-list-swap*:

$\text{fold } f \ xs \ (\text{fold } f \ ys \ y) = \text{fold } f \ ys \ (\text{fold } f \ xs \ y)$

<proof>

lemma (**in** *comp-fun-commute*) *foldr-list-swap*:

$\text{foldr } f \ xs \ (\text{foldr } f \ ys \ y) = \text{foldr } f \ ys \ (\text{foldr } f \ xs \ y)$

<proof>

lemma (**in** *comp-fun-commute*) *foldr-to-fold*:

$\text{foldr } f \ xs = \text{fold } f \ xs$

<proof>

lemma (**in** *comp-fun-commute*) *fold-commute-f*:

$f x (\text{foldr } f \ xs \ y) = \text{foldr } f \ xs \ (f x y)$

$\langle proof \rangle$

lemma *closure-sound*:

assumes *cl*: $\bigwedge s t. s \in S \implies t \in S \implies f s t \in S$
and *com*: $\bigwedge x y. f x y = f y x$ **and** *ass*: $\bigwedge x y z. f x (f y z) = f (f x y) z$
and *fin*: $set\ ss \subseteq S\ ss \neq []$
shows $fold\ f\ (tl\ ss)\ (hd\ ss) \in S$ $\langle proof \rangle$

locale *set-closure-operator* =

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-operator* \subseteq *comp-fun-idem*

$\langle proof \rangle$

context *set-closure-operator*

begin

inductive-set *closure* **for** *S* **where**

base [*simp*]: $s \in S \implies s \in closure\ S$
step [*intro*]: $s \in closure\ S \implies t \in closure\ S \implies f s t \in closure\ S$

lemma *closure-idem* [*simp*]:

$closure\ (closure\ S) = closure\ S$ (**is** $?LS = ?RS$)
 $\langle proof \rangle$

lemma *fold-dist*:

assumes $xs \neq []$
shows $f\ (fold\ f\ (tl\ xs)\ (hd\ xs))\ t = fold\ f\ xs\ t$ $\langle proof \rangle$

lemma *closure-to-cons-list*:

assumes $s \in closure\ S$
shows $\exists ss \neq []. fold\ f\ (tl\ ss)\ (hd\ ss) = s \wedge (\forall i < length\ ss. ss ! i \in S)$ $\langle proof \rangle$

lemma *sound-fold*:

assumes $set\ ss \subseteq closure\ S$ **and** $ss \neq []$
shows $fold\ f\ (tl\ ss)\ (hd\ ss) \in closure\ S$ $\langle proof \rangle$

lemma *closure-empty* [*simp*]: $closure\ \{\} = \{\}$

$\langle proof \rangle$

lemma *closure-mono*:

$S \subseteq T \implies closure\ S \subseteq closure\ T$
 $\langle proof \rangle$

lemma *closure-insert*:

$closure (insert\ x\ S) = \{x\} \cup closure\ S \cup \{f\ x\ s \mid s.\ s \in closure\ S\}$
 <proof>

lemma *finite-S-finite-closure* [intro]:
 $finite\ S \implies finite\ (closure\ S)$
 <proof>

end

locale *semilattice-closure-operator* =
cl: set-closure-operator *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]: $\bigwedge x.\ less\text{-}eq\ e\ x$
and *sup-l*: $\bigwedge x\ y.\ less\text{-}eq\ x\ (f\ x\ y)$
and *sup-r*: $\bigwedge x\ y.\ less\text{-}eq\ y\ (f\ x\ y)$
and *upper-bound*: $\bigwedge x\ y\ z.\ less\text{-}eq\ x\ z \implies less\text{-}eq\ y\ z \implies less\text{-}eq\ (f\ x\ y)\ z$
and *trans*: $\bigwedge x\ y\ z.\ less\text{-}eq\ x\ y \implies less\text{-}eq\ y\ z \implies less\text{-}eq\ x\ z$
and *anti-sym*: $\bigwedge x\ y.\ less\text{-}eq\ x\ y \implies less\text{-}eq\ y\ x \implies x = y$
begin

lemma *unique-neut-elem* [simp]:
 $f\ x\ y = e \iff x = e \wedge y = e$
 <proof>

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 $ss \subseteq S$
 <proof>

lemma *sound-fold*:
assumes set $ss \subseteq closure\ S$ $ss \neq []$
shows fold *f* *ss* *e* $\in closure\ S$
 <proof>

abbreviation $supremum\ S \equiv Finite\text{-}Set.fold\ f\ e\ S$

definition $smaller\text{-}subset\ x\ S \equiv \{y.\ less\text{-}eq\ y\ x \wedge y \in S\}$

lemma *smaller-subset-empty* [simp]:
 $smaller\text{-}subset\ x\ \{\} = \{\}$
 <proof>

lemma *finite-smaller-subset* [simp, intro]:
 $finite\ S \implies finite\ (smaller\text{-}subset\ x\ S)$
 <proof>

lemma *smaller-subset-mono*:

smaller-subset $x S \subseteq S$

\langle *proof* \rangle

lemma *sound-set-fold*:

assumes *set* $ss \subseteq \text{closure } S$ **and** $ss \neq []$

shows $\text{supremum } (set\ ss) \in \text{closure } S$

\langle *proof* \rangle

lemma *supremum-neutral* [*simp*]:

assumes *finite* S **and** $\text{supremum } S = e$

shows $S \subseteq \{e\}$ \langle *proof* \rangle

lemma *supremum-in-closure*:

assumes *finite* S **and** $R \subseteq \text{closure } S$ **and** $R \neq \{\}$

shows $\text{supremum } R \in \text{closure } S$

\langle *proof* \rangle

lemma *supremum-sound*:

assumes *finite* S

shows $\bigwedge t. t \in S \implies \text{less-eq } t (\text{supremum } S)$

\langle *proof* \rangle

lemma *supremum-sound-list*:

$\forall i < \text{length } ss. \text{less-eq } (ss\ !\ i) (\text{fold } f\ ss\ e)$

\langle *proof* \rangle

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

$\text{less-eq } y\ x \implies \text{smaller-subset } x (\text{insert } y\ S) = \text{insert } y (\text{smaller-subset } x\ S)$

$\neg \text{less-eq } y\ x \implies \text{smaller-subset } x (\text{insert } y\ S) = \text{smaller-subset } x\ S$

\langle *proof* \rangle

lemma *supremum-smaller-subset*:

assumes *finite* S

shows $\text{less-eq } (\text{supremum } (\text{smaller-subset } x\ S))\ x$ \langle *proof* \rangle

lemma *pre-subset-eq-pos-subset* [*simp*]:

shows $\text{smaller-subset } x (\text{closure } S) = \text{closure } (\text{smaller-subset } x\ S)$ (**is** $?LS = ?RS$)

\langle *proof* \rangle

lemma *supremum-in-smaller-closure*:

assumes *finite* S

shows $\text{supremum } (\text{smaller-subset } x\ S) \in \{e\} \cup (\text{closure } S)$

\langle *proof* \rangle

lemma *supremum-subset-less-eq*:

assumes *finite S and $R \subseteq S$*
shows *less-eq (supremum R) (supremum S) <proof>*

lemma *supremum-smaller-closure [simp]:*

assumes *finite S*
shows *supremum (smaller-subset x (closure S)) = supremum (smaller-subset x S)*
<proof>

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-operator =*

fixes *P f*

assumes *refl: $\bigwedge x. P x x$*

and *sym: $\bigwedge x y. P x y \implies P y x$*

and *dist: $\bigwedge x y z. P y z \implies P x (f y z) \implies P x y$*

and *assP: $\bigwedge x y z. P x (f y z) \implies P y z \implies P (f x y) z$*

and *com [ac-simps]: $\bigwedge x y. P x y \implies f x y = f y x$*

and *ass [ac-simps]: $\bigwedge x y z. P x y \implies P y z \implies f x (f y z) = f (f x y) z$*

and *idem: $\bigwedge x. f x x = x$*

begin

lemma *lift-f-total-com:*

lift-f-total P f x y = lift-f-total P f y x
<proof>

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>

lemma *lift-f-total-idem:*

lift-f-total P f x x = x
<proof>

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*

<proof>

lemma *lift-set-closure-operator*:
 set-closure-operator (lift-f-total P f)
 <proof>

end

sublocale *set-closure-partial-operator* \subseteq *lift-fun*: *set-closure-operator lift-f-total P f*
 <proof>

context *set-closure-partial-operator* **begin**

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

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

lemma *pred-closure-to-some-lift-closure*:
 assumes *s ∈ pred-closure S*
 shows *Some s ∈ lift-closure S* *<proof>*

lemma *some-lift-closure-pred-closure*:
 fixes *t* **defines** *s* \equiv *Some t*
 assumes *Some t ∈ lift-closure S*
 shows *t ∈ pred-closure S* *<proof>*

lemma *pred-closure-lift-closure*:
 pred-closure S = the ‘ (lift-closure S - {None}) (is ?LS = ?RS)
 <proof>

lemma *finite-S-finite-closure [simp, intro]*:
 finite S \implies finite (pred-closure S)
 <proof>

lemma *closure-mono*:
 assumes *S \subseteq T*
 shows *pred-closure S \subseteq pred-closure T*
 <proof>

lemma *pred-closure-empty [simp]*:
 pred-closure {} = {}
 <proof>

end

locale *semilattice-closure-partial-operator* =

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

abbreviation *lifted-less-eq* $\equiv \text{lift-less-eq-total less-eq}$

abbreviation *lifted-fun* $\equiv \text{lift-f-total } P f$

lemma *lift-less-eq-None* [*simp*]:
lifted-less-eq *None* $y \longleftrightarrow y = \text{None}$
 ⟨*proof*⟩

lemma *lift-less-eq-neut-elm* [*simp*]:
lifted-fun (*Some* e) $s = s$
 ⟨*proof*⟩

lemma *lift-less-eq-neut-less* [*simp*]:
lifted-less-eq (*Some* e) $s \longleftrightarrow \text{True}$
 ⟨*proof*⟩

lemma *lift-less-eq-sup-l* [*simp*]:
lifted-less-eq x (*lifted-fun* $x y$) $\longleftrightarrow \text{True}$
 ⟨*proof*⟩

lemma *lift-less-eq-sup-r* [*simp*]:
lifted-less-eq y (*lifted-fun* $x y$) $\longleftrightarrow \text{True}$
 ⟨*proof*⟩

lemma *lifted-less-eq-trans* [*trans*]:
lifted-less-eq $x y \implies \text{lifted-less-eq } y z \implies \text{lifted-less-eq } x z$
 ⟨*proof*⟩

lemma *lifted-less-eq-anti-sym* [*trans*]:
lifted-less-eq $x y \implies \text{lifted-less-eq } y x \implies x = y$
 ⟨*proof*⟩

lemma *lifted-less-eq-upper*:
lifted-less-eq $x z \implies \text{lifted-less-eq } y z \implies \text{lifted-less-eq } (\text{lifted-fun } x y) z$
 ⟨*proof*⟩

lemma *semilattice-closure-operator-axioms*:

semilattice-closure-operator-axioms (*lift-f-total P f*) (*lift-less-eq-total less-eq*) (*Some e*)
 ⟨*proof*⟩

end

sublocale *semilattice-closure-partial-operator* \subseteq *lift-ord: semilattice-closure-operator*
lift-f-total P f lift-less-eq-total less-eq Some e
 ⟨*proof*⟩

context *semilattice-closure-partial-operator*
begin

abbreviation *supremum* \equiv *lift-ord.supremum*

abbreviation *smaller-subset* \equiv *lift-ord.smaller-subset*

lemma *supremum-impl*:

assumes *supremum* (*set* (*map Some ss*)) = *Some t*

shows *foldr f ss e = t* ⟨*proof*⟩

lemma *supremum-smaller-exists-unique*:

assumes *finite S*

shows $\exists!$ *p. supremum* (*smaller-subset* (*Some t*) (*Some ' S*)) = *Some p* ⟨*proof*⟩

lemma *supremum-neut-or-in-closure*:

assumes *finite S*

shows *the* (*supremum* (*smaller-subset* (*Some t*) (*Some ' S*))) \in $\{e\} \cup$ *cl.pred-closure S*
 ⟨*proof*⟩

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*)

⟨*proof*⟩

lemma (**in** *set-closure-partial-operator*) *closure-impl* [*simp*]:

set (*map the* (*removeAll None* (*closure-impl* (*lift-f-total P f*) (*map Some S*)))) = *pred-closure* (*set S*)

⟨*proof*⟩

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* $\mathcal{F} \mathcal{R} = \{(s, t). \text{funas-term } s \subseteq \mathcal{F} \wedge \text{funas-term } t \subseteq \mathcal{F} \wedge (s, t) \in \mathcal{R}\}$

inductive-set *rstep* :: - \Rightarrow (*'f, 'v term rel* **for** *R* :: (*'f, 'v trs*

where

rstep: $\bigwedge C \sigma l r. (l, r) \in R \implies s = C(l \cdot \sigma) \implies t = C(r \cdot \sigma) \implies (s, t) \in \text{rstep } R$

definition *rstep-r-p-s* :: (*'f, 'v trs* \Rightarrow (*'f, 'v rule* \Rightarrow *pos* \Rightarrow (*'f, 'v subst* \Rightarrow (*'f, 'v trs* **where**

rstep-r-p-s $R r p \sigma = \{(s, t). p \in \text{poss } s \wedge p \in \text{poss } t \wedge r \in R \wedge \text{ctxt-at-pos } s p = \text{ctxt-at-pos } t p \wedge$

$s[p \leftarrow (\text{fst } r \cdot \sigma)] = s \wedge t[p \leftarrow (\text{snd } r \cdot \sigma)] = t\}$

Rewriting steps below the root position.

definition *nrrstep* :: (*'f, 'v trs* \Rightarrow (*'f, 'v trs* **where**

nrrstep $R = \{(s, t). \exists r i ps \sigma. (s, t) \in \text{rstep-r-p-s } R r (i\#ps) \sigma\}$

Rewriting step at the root position.

definition *rrstep* :: (*'f, 'v trs* \Rightarrow (*'f, 'v trs* **where**

rrstep $R = \{(s, t). \exists r \sigma. (s, t) \in \text{rstep-r-p-s } R r [] \sigma\}$

the parallel rewrite relation

inductive-set *par-rstep* :: (*'f, 'v trs* \Rightarrow (*'f, 'v trs* **for** *R* :: (*'f, 'v trs*

where *root-step[intro]*: $(s, t) \in R \implies (s \cdot \sigma, t \cdot \sigma) \in \text{par-rstep } R$

| *par-step-fun[intro]*: $\llbracket \bigwedge i. i < \text{length } ts \implies (ss ! i, ts ! i) \in \text{par-rstep } R \rrbracket \implies$

$\text{length } ss = \text{length } ts \implies (\text{Fun } f \text{ } ss, \text{Fun } f \text{ } ts) \in \text{par-rstep } R$

| *par-step-var[intro]*: $(\text{Var } x, \text{Var } x) \in \text{par-rstep } R$

3.2 Ground variants connecting to FORT

definition *grrstep* :: (*'f, 'v trs* \Rightarrow *'f gterm rel* **where**

$grstep \mathcal{R} = \text{inv-image } (rrstep \mathcal{R}) \text{ term-of-gterm}$

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

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

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

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

lemma $rstep\text{-induct}$ [consumes 1, case-names rule subst ctxt]:

assumes $(s, t) \in rstep R$
and rule: $\bigwedge l r. (l, r) \in R \Longrightarrow P l r$
and subst: $\bigwedge s t \sigma. P s t \Longrightarrow P (s \cdot \sigma) (t \cdot \sigma)$
and ctxt: $\bigwedge s t C. P s t \Longrightarrow P (C\langle s \rangle) (C\langle t \rangle)$
shows $P s t$
 $\langle \text{proof} \rangle$

lemmas $rstepI = rstep.intros$ [intro]

lemmas $rstepE = rstep.cases$ [elim]

lemma $rstep\text{-ctxt}$ [intro]: $(s, t) \in rstep R \Longrightarrow (C\langle s \rangle, C\langle t \rangle) \in rstep R$
 $\langle \text{proof} \rangle$

lemma $rstep\text{-rule}$ [intro]: $(l, r) \in R \Longrightarrow (l, r) \in rstep R$
 $\langle \text{proof} \rangle$

lemma $rstep\text{-subst}$ [intro]: $(s, t) \in rstep R \Longrightarrow (s \cdot \sigma, t \cdot \sigma) \in rstep R$
 $\langle \text{proof} \rangle$

lemma $nrrstep\text{-def}'$:

$nrrstep R = \{(s, t). \exists l r C \sigma. (l, r) \in R \wedge C \neq \square \wedge s = C\langle l \cdot \sigma \rangle \wedge t = C\langle r \cdot \sigma \rangle\}$
(is ?Ls = ?Rs)
 $\langle \text{proof} \rangle$

lemma $rrstep\text{-def}'$: $rrstep R = \{(s, t). \exists l r \sigma. (l, r) \in R \wedge s = l \cdot \sigma \wedge t = r \cdot \sigma\}$
 $\langle \text{proof} \rangle$

lemma $rstep\text{-imp-C-s-r}$:

assumes $(s, t) \in rstep R$
shows $\exists C \sigma l r. (l, r) \in R \wedge s = C\langle l \cdot \sigma \rangle \wedge t = C\langle r \cdot \sigma \rangle$ $\langle \text{proof} \rangle$

lemma $rhs\text{-wf}$:

assumes $R: (l, r) \in R$ **and** *funas-trs* $R \subseteq F$
shows *funas-term* $r \subseteq F$
 \langle *proof* \rangle

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

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 $R \equiv |\bigcup| (\text{ffunas-term } | \cdot | R)$

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

lemma *states-conv*:

states $R = \text{term-to-bot-term } \cdot (\bigcup s \in R. \text{subterms } s)$
 \langle *proof* \rangle

lemma *finite-states*:

assumes *finite* R **shows** *finite* (*states* R)

\langle *proof* \rangle

lemma *root-bot-diff*:

root-bot $\cdot (R - \{Bot\}) = (\text{root-bot } \cdot R) - \{None\}$
 \langle *proof* \rangle

lemma *root-bot-states-root-subterms*:

the $\cdot (\text{root-bot } \cdot (\text{states } R - \{Bot\})) = \text{the } \cdot (\text{root } \cdot (\bigcup s \in R. \text{subterms } s) - \{None\})$

\langle *proof* \rangle

context

includes *fset.lifting*

begin

lift-definition *fstates* $:: ('f, 'v) \text{ term } fset \Rightarrow 'f \text{ bot-term } fset \text{ is } \text{states}$

\langle *proof* \rangle

lift-definition $fsubterms :: ('f, 'v) term \Rightarrow ('f, 'v) term fset$ **is** $subterms$
 $\langle proof \rangle$

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

lift-definition $ffunas-trs :: (('f, 'v) term \times ('f, 'v) term) fset \Rightarrow ('f \times nat) fset$
is $funas-trs$
 $\langle proof \rangle$

lemma $fstates-def'$:
 $t \in fstates R \longleftrightarrow (\exists s u. s \in R \wedge s \supseteq u \wedge u^\perp = t)$
 $\langle proof \rangle$

lemma $fstates-fmemberE [elim!]$:
assumes $t \in fstates R$
obtains $s u$ **where** $s \in R \wedge s \supseteq u \wedge u^\perp = t$
 $\langle proof \rangle$

lemma $fstates-fmemberI [intro]$:
 $s \in R \Longrightarrow s \supseteq u \Longrightarrow u^\perp \in fstates R$
 $\langle proof \rangle$

lemmas $root-bot-states-root-subterms = root-bot-states-root-subterms[Transfer.transferred]$
lemmas $root-fsubterms-ffunas-term-fset = root-subterms-funas-term-set[Transfer.transferred]$

lemma $fstates[code]$:
 $fstates R = term-to-bot-term \mid\mid (\mid\bigcup \mid (fsubterms \mid\mid R))$
 $\langle proof \rangle$

end

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

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

lemma $ta-rule-sig-term-to-ta-rule-root$:
 $t \neq Bot \Longrightarrow ta-rule-sig (term-to-ta-rule t) = the (root-bot t)$
 $\langle proof \rangle$

lemma $ta-rule-sig-term-to-ta-rule-root-set$:
assumes $Bot \notin R$
shows $ta-rule-sig \mid\mid (term-to-ta-rule \mid\mid R) = the \mid\mid (root-bot \mid\mid R)$
 $\langle proof \rangle$

definition $pattern-automaton-rules$ **where**

pattern-automaton-rules $\mathcal{F} R =$
 (let states = (fstates R) - $\{|Bot|\}$) in
 term-to-ta-rule $|^{\dagger}$ states \cup $(\lambda (f, n). TA\text{-rule } f \text{ (replicate } n \text{ Bot) Bot}) |^{\dagger} \mathcal{F}$

lemma *pattern-automaton-rules-BotD*:

assumes *TA-rule* $f ss Bot \in$ *pattern-automaton-rules* $\mathcal{F} R$

shows *TA-rule* $f ss Bot \in$ $(\lambda (f, n). TA\text{-rule } f \text{ (replicate } n \text{ Bot) Bot}) |^{\dagger} \mathcal{F}$
 $\langle proof \rangle$

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* $|^{\dagger} ((fstates R) - \{|Bot|\}) \langle proof \rangle$

definition *pattern-automaton where*

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

lemma *ta-sig-pattern-automaton [simp]*:

ta-sig (pattern-automaton $\mathcal{F} R) = \mathcal{F} \cup$ *ffunas-terms* R
 $\langle proof \rangle$

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) $\langle proof \rangle$

lemma *pattern-automaton-reach-smaller-term*:

assumes $l \in$ $R \ l \supseteq \ s \ s^{\perp} \leq_b \ (term\text{-of-gterm } t)^{\perp} \ ffunas\text{-gterm } t \subseteq$ \mathcal{F}

shows $s^{\perp} \in$ *ta-der* (pattern-automaton $\mathcal{F} R$) (term-of-gterm t) $\langle proof \rangle$

lemma *bot-term-of-gterm-conv*:

term-of-gterm $s^{\perp} = term\text{-of-gterm } s^{\perp}$

$\langle proof \rangle$

lemma *pattern-automaton-ground-instance-reach*:

assumes $l \in$ $R \ l \cdot \sigma = (term\text{-of-gterm } t) \ ffunas\text{-gterm } t \subseteq$ \mathcal{F}

shows $l^{\perp} \in$ *ta-der* (pattern-automaton $\mathcal{F} R$) (term-of-gterm t)
 $\langle proof \rangle$

lemma *pattern-automaton-reach-smallest-term*:

assumes $l^{\perp} \in$ *ta-der* (pattern-automaton $\mathcal{F} R$) t ground t

shows $l^{\perp} \leq_b \ t^{\perp} \langle proof \rangle$

4.2 Recognizing root step relation of LV-TRSs

definition *lv-trs* :: $(f, v) \text{ trs} \Rightarrow \text{bool}$ **where**

lv-trs $R \equiv \forall (l, r) \in R. \text{linear-term } l \wedge \text{linear-term } r \wedge (\text{vars-term } l \cap \text{vars-term } r = \{\})$

lemma *subst-unification*:

assumes *vars-term* $s \cap \text{vars-term } t = \{\}$
obtains μ **where** $s \cdot \sigma = s \cdot \mu \ t \cdot \tau = t \cdot \mu$
 $\langle \text{proof} \rangle$

lemma *lv-trs-subst-unification*:

assumes *lv-trs* $R \ (l, r) \in R \ s = l \cdot \sigma \ t = r \cdot \tau$
obtains μ **where** $s = l \cdot \mu \wedge t = r \cdot \mu$
 $\langle \text{proof} \rangle$

definition Rel_f **where**

$Rel_f \ R = \text{map-both term-to-bot-term } |\uparrow| \ R$

definition *root-pair-automaton* **where**

root-pair-automaton $\mathcal{F} \ R = (\text{pattern-automaton } \mathcal{F} \ (\text{fst } |\uparrow| \ R),$
pattern-automaton $\mathcal{F} \ (\text{snd } |\uparrow| \ R))$

definition *agtt-grrstep* **where**

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

lemma *agtt-grrstep-eps-trancl* [*simp*]:

$(\text{eps } (\text{fst } (\text{agtt-grrstep } \mathcal{R} \ \mathcal{F})))^{+} = \text{eps } (\text{fst } (\text{agtt-grrstep } \mathcal{R} \ \mathcal{F}))$
 $(\text{eps } (\text{snd } (\text{agtt-grrstep } \mathcal{R} \ \mathcal{F}))) = \{\|\}$
 $\langle \text{proof} \rangle$

lemma *root-pair-automaton-grrstep*:

fixes $R :: ('f, 'v) \text{ rule fset}$
assumes *lv-trs* $(\text{fset } R) \ \text{ffun-as-trs } R \ |\subseteq| \ \mathcal{F}$
shows *pair-at-lang* $(\text{root-pair-automaton } \mathcal{F} \ R) \ (Rel_f \ R) = \text{Restr } (\text{grrstep } (\text{fset } R)) \ (\mathcal{T}_G \ (\text{fset } \mathcal{F})) \ (\text{is } ?Ls = ?Rs)$
 $\langle \text{proof} \rangle$

lemma *agtt-grrstep*:

fixes $R :: ('f, 'v) \text{ rule fset}$
assumes *lv-trs* $(\text{fset } R) \ \text{ffun-as-trs } R \ |\subseteq| \ \mathcal{F}$
shows *agtt-lang* $(\text{agtt-grrstep } R \ \mathcal{F}) = \text{Restr } (\text{grrstep } (\text{fset } R)) \ (\mathcal{T}_G \ (\text{fset } \mathcal{F}))$
 $\langle \text{proof} \rangle$

lemma *root-pair-automaton-grrstep-set*:

fixes $R :: ('f, 'v) \text{ rule set}$
assumes *finite* $R \ \text{finite } \mathcal{F} \ \text{lv-trs } R \ \text{fun-as-trs } R \ \subseteq \ \mathcal{F}$
shows *pair-at-lang* $(\text{root-pair-automaton } (\text{Abs-fset } \mathcal{F}) \ (\text{Abs-fset } R)) \ (Rel_f \ (\text{Abs-fset } R)) = \text{Restr } (\text{grrstep } R) \ (\mathcal{T}_G \ \mathcal{F})$
 $\langle \text{proof} \rangle$

lemma *agtt-grrstep-set*:

fixes $R :: ('f, 'v) \text{ rule set}$

assumes *finite R finite F lv-trs R funas-trs R* $\subseteq \mathcal{F}$
shows *agtt-lang (agtt-grrstep (Abs-fset R) (Abs-fset F)) = Restr (grrstep R) (\mathcal{T}_G F)*
<proof>

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* $\lambda x y. (x, y) \in \text{merge}P$
 $(\uparrow) \lambda x y. x \leq_b y$ *Bot*
<proof>

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

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

lemma *psubt-mono:*
 $R \subseteq S \implies \text{psubt-lhs-bot } R \subseteq \text{psubt-lhs-bot } S$ *<proof>*

lemma *states-mono:*
 $R \subseteq S \implies \text{states } R \subseteq \text{states } S$
<proof>

lemma *finite-lhs-subt [simp, intro]:*
assumes *finite R*
shows *finite (psubt-lhs-bot R)*
<proof>

lemma *states-ref-closure:*
 $\text{states } R \subseteq \text{insert Bot (closure (psubt-lhs-bot R))}$
<proof>

lemma *finite-R-finite-states [simp, intro]:*
 $\text{finite } R \implies \text{finite (states } R)$
<proof>

abbreviation *lift-sup-small* $s S \equiv \text{lift-total.supremum (lift-total.smaller-subset (Some s) (Some ' S))}$
abbreviation *bound-max* $s S \equiv \text{the (lift-sup-small s S)}$

lemma *bound-max-state-set*:

assumes *finite R*

shows $\text{bound-max } t \text{ (psubt-lhs-bot } R) \in \text{states } R$

<proof>

context

includes *fset.lifting*

begin

lift-definition $\text{fstates} :: ('a, 'b) \text{ term fset} \Rightarrow 'a \text{ bot-term fset}$ **is** *states*

<proof>

lemma *bound-max-state-fset*:

$\text{bound-max } t \text{ (psubt-lhs-bot (fset } R)) \in \text{fstates } R$

<proof>

end

definition *nf-rules where*

$\text{nf-rules } R \mathcal{F} = \{ | \text{TA-rule } f \text{ } q \text{ } q \mid f \text{ } q \text{ } q. (f, \text{length } q) \in \mathcal{F} \wedge \text{fset-of-list } q \subseteq | \text{fstates } R \wedge$

$\neg(\exists l \in | R. l^\perp \leq_b \text{BFun } f \text{ } q) \wedge q = \text{bound-max } (\text{BFun } f \text{ } q) \text{ (psubt-lhs-bot (fset } R)) | \}$

lemma *nf-rules-fmember*:

$\text{TA-rule } f \text{ } q \text{ } q \in \text{nf-rules } R \mathcal{F} \longleftrightarrow (f, \text{length } q) \in \mathcal{F} \wedge \text{fset-of-list } q \subseteq | \text{fstates } R \wedge$

$\neg(\exists l \in | R. l^\perp \leq_b \text{BFun } f \text{ } q) \wedge q = \text{bound-max } (\text{BFun } f \text{ } q) \text{ (psubt-lhs-bot (fset } R))$

<proof>

definition *nf-ta where*

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

definition *nf-reg where*

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

lemma *bound-max-sound*:

assumes *finite R*

shows $\text{bound-max } t \text{ (psubt-lhs-bot } R) \leq_b t$

<proof>

lemma *Bot-in-filter*:

$\text{Bot} \in \text{Set.filter } (\lambda s. s \leq_b t) \text{ (states } R)$

<proof>

lemma *bound-max-exists*:

$\exists p. p = \text{bound-max } t \text{ (psubt-lhs-bot } R)$

<proof>

lemma *bound-max-unique*:
assumes $p = \text{bound-max } t \text{ (psubt-lhs-bot } R)$ **and** $q = \text{bound-max } t \text{ (psubt-lhs-bot } R)$
shows $p = q$ *<proof>*

lemma *nf-rule-to-bound-max*:
 $f \text{ qs} \rightarrow q \mid \in \mid \text{nf-rules } R \mathcal{F} \implies q = \text{bound-max (BFun } f \text{ qs) (psubt-lhs-bot (fset } R))$
<proof>

lemma *nf-rules-unique*:
assumes $f \text{ qs} \rightarrow q \mid \in \mid \text{nf-rules } R \mathcal{F}$ **and** $f \text{ qs} \rightarrow q' \mid \in \mid \text{nf-rules } R \mathcal{F}$
shows $q = q'$ *<proof>*

lemma *nf-ta-det*:
shows *ta-det (nf-ta R F)*
<proof>

lemma *term-instance-of-reach-state*:
assumes $q \mid \in \mid \text{ta-der (nf-ta } R \mathcal{F}) \text{ (adapt-vars } t)$ **and** *ground t*
shows $q \leq_b t^\perp$ *<proof>*

lemma [*simp*]: $i < \text{length } ss \implies l \triangleright \text{Fun } f \text{ ss} \implies l \triangleright ss ! i$
<proof>

lemma *subt-less-eq-res-less-eq*:
assumes *ground: ground t* **and** $l \mid \in \mid R$ **and** $l \triangleright s$ **and** $s^\perp \leq_b t^\perp$
and $q \mid \in \mid \text{ta-der (nf-ta } R \mathcal{F}) \text{ (adapt-vars } t)$
shows $s^\perp \leq_b q$ *<proof>*

lemma *ta-nf-sound1*:
assumes *ground: ground t* **and** *lhs: l |∈| R* **and** *inst: l[⊥] ≤_b t[⊥]*
shows *ta-der (nf-ta R F) (adapt-vars t) = {||}*
<proof>

lemma *ta-nf-tr-to-state*:
assumes *ground t* **and** $q \mid \in \mid \text{ta-der (nf-ta } R \mathcal{F}) \text{ (adapt-vars } t)$
shows $q \mid \in \mid \text{fstates } R$ *<proof>*

lemma *ta-nf-sound2*:
assumes *linear: ∀ l |∈| R. linear-term l*
and *ground (t :: ('f, 'v) term)* **and** *funas-term t ⊆ fset F*
and *NF: ∧ l s. l |∈| R ⟹ t ⊇ s ⟹ ¬ l[⊥] ≤_b s[⊥]*
shows $\exists q. q \mid \in \mid \text{ta-der (nf-ta } R \mathcal{F}) \text{ (adapt-vars } t)$ *<proof>*

lemma *ta-nf-lang-sound*:
assumes $l \mid \in \mid R$
shows $C(l \cdot \sigma) \notin \text{ta-lang (fstates } R) \text{ (nf-ta } R \mathcal{F})$
<proof>

lemma *ta-nf-lang-complete*:

assumes *linear*: $\forall l \mid \in \mid R. \text{linear-term } l$

and *ground*: $\text{ground } (t :: ('f, 'v) \text{ term})$ **and** *sig*: $\text{funas-term } t \subseteq \text{fset } \mathcal{F}$

and *nf*: $\bigwedge C \sigma l. l \mid \in \mid R \implies C \langle l \cdot \sigma \rangle \neq t$

shows $t \in \text{ta-lang } (\text{fstates } R) (\text{nf-ta } R \mathcal{F})$

<proof>

lemma *ta-nf-L-complete*:

assumes *linear*: $\forall l \mid \in \mid R. \text{linear-term } l$

and *sig*: $\text{funas-gterm } t \subseteq \text{fset } \mathcal{F}$

and *nf*: $\bigwedge C \sigma l. l \mid \in \mid R \implies C \langle l \cdot \sigma \rangle \neq (\text{term-of-gterm } t)$

shows $t \in \mathcal{L} (\text{nf-reg } R \mathcal{F})$

<proof>

lemma *nf-ta-funas*:

assumes $\text{ground } t \ q \mid \in \mid \text{ta-der } (\text{nf-ta } R \mathcal{F}) \ t$

shows $\text{funas-term } t \subseteq \text{fset } \mathcal{F}$ *<proof>*

lemma *gta-lang-nf-ta-funas*:

assumes $t \in \mathcal{L} (\text{nf-reg } R \mathcal{F})$

shows $\text{funas-gterm } t \subseteq \text{fset } \mathcal{F}$ *<proof>*

end

theory *Tree-Automata-Derivation-Split*

imports *Regular-Tree-Relations.Tree-Automata*

Ground-MCtxt

begin

lemma *ta-der'-inf-mctxt*:

assumes $t \mid \in \mid \text{ta-der}' \mathcal{A} \ s$

shows $\text{fst } (\text{split-vars } t) \leq (\text{mctxt-of-term } s)$ *<proof>*

lemma *ta-der'-poss-subt-at-ta-der'*:

assumes $t \mid \in \mid \text{ta-der}' \mathcal{A} \ s$ **and** $p \in \text{poss } t$

shows $t \mid - \ p \mid \in \mid \text{ta-der}' \mathcal{A} \ (s \mid - \ p)$ *<proof>*

lemma *ta-der'-varposs-to-ta-der*:

assumes $t \mid \in \mid \text{ta-der}' \mathcal{A} \ s$ **and** $p \in \text{varposs } t$

shows $\text{the-Var } (t \mid - \ p) \mid \in \mid \text{ta-der } \mathcal{A} \ (s \mid - \ p)$ *<proof>*

definition *ta-der'-target-mctxt* $t \equiv \text{fst } (\text{split-vars } t)$

definition *ta-der'-target-args* $t \equiv \text{snd } (\text{split-vars } t)$

definition *ta-der'-source-args* $t \ s \equiv \text{unfill-holes } (\text{fst } (\text{split-vars } t)) \ s$

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*]:

$\text{funas-mctxt } (\text{ta-der'-target-mctxt } u) = \text{funas-term } u$

<proof>

lemma *ta-der'-target-mctxt-ground* [*simp*]:
ground-mctxt (ta-der'-target-mctxt t)
<proof>

lemma *ta-der'-source-args-ground*:
 $t \in | \mathcal{A} s \implies \text{ground } s \implies \forall u \in \text{set } (ta\text{-der}'\text{-source-args } t s). \text{ground } u$
<proof>

lemma *ta-der'-source-args-term-of-gterm*:
 $t \in | \mathcal{A} (\text{term-of-gterm } s) \implies \forall u \in \text{set } (ta\text{-der}'\text{-source-args } t (\text{term-of-gterm } s)). \text{ground } u$
<proof>

lemma *ta-der'-source-args-length*:
 $t \in | \mathcal{A} s \implies \text{num-holes } (ta\text{-der}'\text{-target-mctxt } t) = \text{length } (ta\text{-der}'\text{-source-args } t s)$
<proof>

lemma *ta-der'-target-args-length*:
 $\text{num-holes } (ta\text{-der}'\text{-target-mctxt } t) = \text{length } (ta\text{-der}'\text{-target-args } t)$
<proof>

lemma *ta-der'-target-args-vars-term-conv*:
 $\text{vars-term } t = \text{set } (ta\text{-der}'\text{-target-args } t)$
<proof>

lemma *ta-der'-target-args-vars-term-list-conv*:
 $ta\text{-der}'\text{-target-args } t = \text{vars-term-list } t$
<proof>

lemma *mctxt-args-ta-der'*:

assumes $\text{num-holes } C = \text{length } qs \text{ num-holes } C = \text{length } ss$

and $\forall i < \text{length } ss. qs ! i \in | \mathcal{A} (ss ! i)$

shows $(\text{fill-holes } C (\text{map } \text{Var } qs)) \in | \mathcal{A} (\text{fill-holes } C ss)$ *<proof>*

lemma *ta-der'-mctxt-structure*:

assumes $t \in | \mathcal{A} s$

shows $t = \text{fill-holes } (ta\text{-der}'\text{-target-mctxt } t) (\text{map } \text{Var } (ta\text{-der}'\text{-target-args } t))$ (**is** *?G1*)

$s = \text{fill-holes } (ta\text{-der}'\text{-target-mctxt } t) (ta\text{-der}'\text{-source-args } t s)$ (**is** *?G2*)

$\text{num-holes } (ta\text{-der}'\text{-target-mctxt } t) = \text{length } (ta\text{-der}'\text{-source-args } t s) \wedge$

$\text{length } (ta\text{-der}'\text{-source-args } t s) = \text{length } (ta\text{-der}'\text{-target-args } t)$ (**is** *?G3*)

$i < \text{length } (ta\text{-der}'\text{-source-args } t s) \implies ta\text{-der}'\text{-target-args } t ! i \in | \mathcal{A} (ta\text{-der}'\text{-source-args } t s ! i)$
<proof>

lemma *ta-der'-ground-mctxt-structure*:

assumes $t \in | \mathcal{A} \text{ (term-of-gterm } s)$
shows $t = \text{fill-holes (ta-der'-target-mctxt } t) (\text{map Var (ta-der'-target-args } t))$
 $\text{term-of-gterm } s = \text{fill-holes (ta-der'-target-mctxt } t) (\text{ta-der'-source-args } t (\text{term-of-gterm } s))$
 $\text{num-holes (ta-der'-target-mctxt } t) = \text{length (ta-der'-source-args } t (\text{term-of-gterm } s)) \wedge$
 $\text{length (ta-der'-source-args } t (\text{term-of-gterm } s)) = \text{length (ta-der'-target-args } t)$
 $i < \text{length (ta-der'-target-args } t) \implies \text{ta-der'-target-args } t ! i \in | \mathcal{A}$
 $(\text{ta-der'-source-args } t (\text{term-of-gterm } s) ! i)$
 $\langle \text{proof} \rangle$

definition $\text{ta-der'-gctxt } t \equiv \text{gctxt-of-gmctxt (gmctxt-of-mctxt (fst (split-vars } t))$

abbreviation $\text{ta-der'-ctxt } t \equiv \text{ctxt-of-gctxt (ta-der'-gctxt } t)$

definition $\text{ta-der'-source-ctxt-arg } t s \equiv \text{hd (unfill-holes (fst (split-vars } t)) s)$

abbreviation $\text{ta-der'-source-gctxt-arg } t s \equiv \text{gterm-of-term (ta-der'-source-ctxt-arg } t (\text{term-of-gterm } s))$

lemma $\text{ta-der'-ctxt-structure}$:

assumes $t \in | \mathcal{A} \text{ s vars-term-list } t = [q]$
shows $t = (\text{ta-der'-ctxt } t) \langle \text{Var } q \rangle$ (**is** ?G1)
 $s = (\text{ta-der'-ctxt } t) \langle \text{ta-der'-source-ctxt-arg } t s \rangle$ (**is** ?G2)
 $\text{ground-ctxt (ta-der'-ctxt } t)$ (**is** ?G3)
 $q \in | \mathcal{A} (\text{ta-der'-source-ctxt-arg } t s)$ (**is** ?G4)
 $\langle \text{proof} \rangle$

lemma $\text{ta-der'-ground-ctxt-structure}$:

assumes $t \in | \mathcal{A} \text{ (term-of-gterm } s) \text{ vars-term-list } t = [q]$
shows $t = (\text{ta-der'-ctxt } t) \langle \text{Var } q \rangle$
 $s = (\text{ta-der'-gctxt } t) \langle \text{ta-der'-source-gctxt-arg } t s \rangle_G$
 $\text{ground (ta-der'-source-ctxt-arg } t (\text{term-of-gterm } s))$
 $q \in | \mathcal{A} (\text{ta-der'-source-ctxt-arg } t (\text{term-of-gterm } s))$
 $\langle \text{proof} \rangle$

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

locale $\text{derivation-split} =$

fixes $A :: ('q, 'f) \text{ ta and } \mathcal{A} \text{ and } \mathcal{B}$

assumes $\text{rule-split: rules } A = \text{rules } \mathcal{A} \mid \cup \mid \text{rules } \mathcal{B}$

and $\text{eps-split: eps } A = \text{eps } \mathcal{A} \mid \cup \mid \text{eps } \mathcal{B}$

and $\text{B-target-states: rule-target-states (rules } \mathcal{B}) \mid \cup \mid (\text{snd } | \uparrow (\text{eps } \mathcal{B})) \mid \cap \mid$

$(\text{rule-arg-states (rules } \mathcal{A}) \mid \cup \mid (\text{fst } | \uparrow (\text{eps } \mathcal{A}))) = \{\mid\}$

begin

abbreviation $\Delta_A \equiv \text{rules } \mathcal{A}$

abbreviation $\Delta_{\mathcal{E}A} \equiv \text{eps } \mathcal{A}$

abbreviation $\Delta_B \equiv \text{rules } \mathcal{B}$

abbreviation $\Delta_{\mathcal{E}B} \equiv \text{eps } \mathcal{B}$

abbreviation $\mathcal{Q}_A \equiv \mathcal{Q} \mathcal{A}$

definition $\mathcal{Q}_B \equiv \text{rule-target-states } \Delta_B \mid \cup \mid (\text{snd } \mid \mid \Delta_{\mathcal{E}B})$

lemmas $B\text{-target-states}' = B\text{-target-states}[\text{folded } \mathcal{Q}_B\text{-def}]$

lemma *states-split* [*simp*]: $\mathcal{Q} A = \mathcal{Q} \mathcal{A} \mid \cup \mid \mathcal{Q} \mathcal{B}$
<proof>

lemma *A-args-states-not-B*:

$TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \Delta_A \implies p \mid \in \mid \text{fset-of-list } qs \implies p \mid \notin \mid \mathcal{Q}_B$
<proof>

lemma *rule-statesD*:

$r \mid \in \mid \Delta_A \implies r\text{-rhs } r \mid \in \mid \mathcal{Q}_A$
 $r \mid \in \mid \Delta_B \implies r\text{-rhs } r \mid \in \mid \mathcal{Q}_B$
 $r \mid \in \mid \Delta_A \implies p \mid \in \mid \text{fset-of-list } (r\text{-lhs-states } r) \implies p \mid \in \mid \mathcal{Q}_A$
 $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \Delta_A \implies q \mid \in \mid \mathcal{Q}_A$
 $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \Delta_B \implies q \mid \in \mid \mathcal{Q}_B$
 $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \Delta_A \implies p \mid \in \mid \text{fset-of-list } qs \implies p \mid \in \mid \mathcal{Q}_A$
<proof>

lemma *eps-states-dest*:

$(p, q) \mid \in \mid \Delta_{\mathcal{E}A} \implies p \mid \in \mid \mathcal{Q}_A$
 $(p, q) \mid \in \mid \Delta_{\mathcal{E}A} \implies q \mid \in \mid \mathcal{Q}_A$
 $(p, q) \mid \in \mid \Delta_{\mathcal{E}A}^+ \implies p \mid \in \mid \mathcal{Q}_A$
 $(p, q) \mid \in \mid \Delta_{\mathcal{E}A}^+ \implies q \mid \in \mid \mathcal{Q}_A$
 $(p, q) \mid \in \mid \Delta_{\mathcal{E}B} \implies q \mid \in \mid \mathcal{Q}_B$
 $(p, q) \mid \in \mid \Delta_{\mathcal{E}B}^+ \implies q \mid \in \mid \mathcal{Q}_B$
<proof>

lemma *transcl-eps-simp*:

$(\text{eps } A)^+ \mid \in \mid \Delta_{\mathcal{E}A}^+ \mid \cup \mid \Delta_{\mathcal{E}B}^+ \mid \cup \mid (\Delta_{\mathcal{E}A}^+ \mid \mid \Delta_{\mathcal{E}B}^+)$
<proof>

lemma *B-rule-eps-A-False*:

$f \text{ } qs \text{ } \rightarrow \text{ } q \mid \in \mid \Delta_B \implies (q, p) \mid \in \mid \Delta_{\mathcal{E}A}^+ \implies \text{False}$
<proof>

lemma *to-A-rule-set*:

assumes $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \text{rules } A$ **and** $q = p \vee (q, p) \mid \in \mid (\text{eps } A)^+$ **and** $p \mid \notin \mid \mathcal{Q}_B$
shows $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \Delta_A$ $q = p \vee (q, p) \mid \in \mid \Delta_{\mathcal{E}A}^+$ *<proof>*

lemma *to-B-rule-set*:

assumes $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \text{rules } A$ **and** $q \mid \notin \mid \mathcal{Q}_A$
shows $TA\text{-rule } f \text{ } qs \text{ } q \mid \in \mid \Delta_B$ *<proof>*

declare *fsubsetI*[*rule del*]
lemma *ta-der-monos*:
ta-der $\mathcal{A} t \sqsubseteq | \text{ta-der } A t \text{ ta-der } \mathcal{B} t \sqsubseteq | \text{ta-der } A t$
 $\langle \text{proof} \rangle$
declare *fsubsetI*[*intro!*]

lemma *ta-der-from- Δ_A* :
assumes $q \in | \text{ta-der } A (\text{term-of-gterm } t)$ **and** $q \notin | \mathcal{Q}_B$
shows $q \in | \text{ta-der } \mathcal{A} (\text{term-of-gterm } t)$ $\langle \text{proof} \rangle$

lemma *ta-state*:
assumes $q \in | \text{ta-der } A (\text{term-of-gterm } s)$
shows $q \in | \mathcal{Q}_A \vee q \in | \mathcal{Q}_B$ $\langle \text{proof} \rangle$

lemma *ta-der-split*:
assumes $q \in | \text{ta-der } A (\text{term-of-gterm } s)$ **and** $q \in | \mathcal{Q}_B$
shows $\exists t. t \in | \text{ta-der}' \mathcal{A} (\text{term-of-gterm } s) \wedge q \in | \text{ta-der } \mathcal{B} t$
(is $\exists t. ?P s q t$) $\langle \text{proof} \rangle$

lemma *ta-der'-split*:
assumes $t \in | \text{ta-der}' A (\text{term-of-gterm } s)$
shows $\exists u. u \in | \text{ta-der}' \mathcal{A} (\text{term-of-gterm } s) \wedge t \in | \text{ta-der}' \mathcal{B} u$
(is $\exists u. ?P s t u$) $\langle \text{proof} \rangle$

lemma *ta-der-to-mctxt*:
assumes $q \in | \text{ta-der } A (\text{term-of-gterm } s)$ **and** $q \in | \mathcal{Q}_B$
shows $\exists C ss qs. \text{length } qs = \text{length } ss \wedge \text{num-holes } C = \text{length } ss \wedge$
 $(\forall i < \text{length } ss. qs ! i \in | \text{ta-der } \mathcal{A} (\text{term-of-gterm } (ss ! i))) \wedge$
 $q \in | \text{ta-der } \mathcal{B} (\text{fill-holes } C (\text{map } \text{Var } qs)) \wedge$
 $\text{ground-mctxt } C \wedge \text{fill-holes } C (\text{map } \text{term-of-gterm } ss) = \text{term-of-gterm } s$
(is $\exists C ss qs. ?P s q C ss qs$)
 $\langle \text{proof} \rangle$

lemma *ta-der-to-gmctxt*:
assumes $q \in | \text{ta-der } A (\text{term-of-gterm } s)$ **and** $q \in | \mathcal{Q}_B$
shows $\exists C ss qs qs'. \text{length } qs' = \text{length } qs \wedge \text{length } qs = \text{length } ss \wedge \text{num-gholes}$
 $C = \text{length } ss \wedge$
 $(\forall i < \text{length } ss. qs ! i \in | \text{ta-der } \mathcal{A} (\text{term-of-gterm } (ss ! i))) \wedge$
 $q \in | \text{ta-der } \mathcal{B} (\text{fill-holes } (\text{mctxt-of-gmctxt } C) (\text{map } \text{Var } qs')) \wedge$
 $\text{fill-gholes } C ss = s$
 $\langle \text{proof} \rangle$

lemma *mctxt-const-to-ta-der*:
assumes *num-holes C = length ss length ss = length qs*
and $\forall i < \text{length } qs. qs ! i \in \text{ta-der } \mathcal{A} (ss ! i)$
and $q \in \text{ta-der } \mathcal{B} (\text{fill-holes } C (\text{map } \text{Var } qs))$
shows $q \in \text{ta-der } \mathcal{A} (\text{fill-holes } C ss)$
 $\langle \text{proof} \rangle$

lemma *ctxt-const-to-ta-der*:
assumes $q \in \text{ta-der } \mathcal{A} s$
and $p \in \text{ta-der } \mathcal{B} C \langle \text{Var } q \rangle$
shows $p \in \text{ta-der } \mathcal{A} C \langle s \rangle \langle \text{proof} \rangle$

lemma *gctxt-const-to-ta-der*:
assumes $q \in \text{ta-der } \mathcal{A} (\text{term-of-gterm } s)$
and $p \in \text{ta-der } \mathcal{B} (\text{ctxt-of-gctxt } C) \langle \text{Var } q \rangle$
shows $p \in \text{ta-der } \mathcal{A} (\text{term-of-gterm } C \langle s \rangle_G) \langle \text{proof} \rangle$

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* $\mathcal{F} q \equiv (\lambda (f, n). \text{TA-rule } f (\text{replicate } n q) q) \mid \mathcal{F}$

definition *refl-ta* $\mathcal{F} q = \text{TA} (\text{reflcl-rules } \mathcal{F} q) \{\mid\}$

definition *gen-reflcl-automaton* $:: ('f \times \text{nat}) \text{fset} \Rightarrow ('q, 'f) \text{ta} \Rightarrow 'q \Rightarrow ('q, 'f) \text{ta}$
where

gen-reflcl-automaton $\mathcal{F} \mathcal{A} q = \text{ta-union } \mathcal{A} (\text{refl-ta } \mathcal{F} q)$

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

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

5.1.2 Multihole context closure over a given signature

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

definition *gen-parallel-closure-automaton* $:: 'q \text{ fset} \Rightarrow ('f \times \text{nat}) \text{ fset} \Rightarrow ('q, 'f) \text{ ta} \Rightarrow 'q \Rightarrow ('q, 'f) \text{ ta}$ **where**

gen-parallel-closure-automaton $Q \mathcal{F} \mathcal{A} q = \text{ta-union } \mathcal{A} (\text{refl-over-states-ta } Q \mathcal{F} \mathcal{A} q)$

definition *parallel-closure-reg* **where**

parallel-closure-reg $\mathcal{F} \mathcal{A} = (\text{let } \mathcal{B} = \text{fmap-states-reg } \text{Some } \mathcal{A} \text{ in } \text{Reg } \{\text{None}\} (\text{gen-parallel-closure-automaton } (\text{fin } \mathcal{B}) \mathcal{F} (\text{ta } \mathcal{B}) \text{None}))$

5.1.3 Context closure of regular tree language

definition *semantic-path-rules* $\mathcal{F} q_c q_i q_f \equiv$

$|\cup| ((\lambda (f, n). \text{fset-of-list } (\text{map } (\lambda i. \text{TA-rule } f ((\text{replicate } n \ q_c)[i := q_i]) \ q_f) [0..< n])) \mid^\dagger \mathcal{F}$

definition *reflcl-over-single-ta* $Q \mathcal{F} q_c q_f \equiv$

$TA (\text{reflcl-rules } \mathcal{F} q_c \mid \cup \text{ semantic-path-rules } \mathcal{F} q_c q_f q_f) ((\lambda p. (p, q_f)) \mid^\dagger Q)$

definition *gen-ctxt-closure-automaton* $Q \mathcal{F} \mathcal{A} q_c q_f = \text{ta-union } \mathcal{A} (\text{reflcl-over-single-ta } Q \mathcal{F} q_c q_f)$

definition *gen-ctxt-closure-reg* $\mathcal{F} \mathcal{A} q_c q_f =$

$\text{Reg } \{q_f\} (\text{gen-ctxt-closure-automaton } (\text{fin } \mathcal{A}) \mathcal{F} (\text{ta } \mathcal{A}) q_c q_f)$

definition *ctxt-closure-reg* $\mathcal{F} \mathcal{A} =$

$(\text{let } \mathcal{B} = \text{fmap-states-reg } \text{Inl } (\text{reg-Restr-}Q_f \ \mathcal{A}) \text{ in } \text{gen-ctxt-closure-reg } \mathcal{F} \mathcal{B} (\text{Inr } \text{False}) (\text{Inr } \text{True}))$

5.1.4 Not empty context closure of regular tree language

datatype *cl-states* $= \text{cl-state} \mid \text{tr-state} \mid \text{fin-state} \mid \text{fin-clstate}$

definition *reflcl-over-nhole-ctxt-ta* $Q \mathcal{F} q_c q_i q_f \equiv$

$TA (\text{reflcl-rules } \mathcal{F} q_c \mid \cup \text{ semantic-path-rules } \mathcal{F} q_c q_i q_f \mid \cup \text{ semantic-path-rules } \mathcal{F} q_c q_f q_f) ((\lambda p. (p, q_i)) \mid^\dagger Q)$

definition *gen-nhole-ctxt-closure-automaton* $Q \mathcal{F} \mathcal{A} q_c q_i q_f =$

$\text{ta-union } \mathcal{A} (\text{reflcl-over-nhole-ctxt-ta } Q \mathcal{F} q_c q_i q_f)$

definition *gen-nhole-ctxt-closure-reg* $\mathcal{F} \mathcal{A} q_c q_i q_f =$

$\text{Reg } \{q_f\} (\text{gen-nhole-ctxt-closure-automaton } (\text{fin } \mathcal{A}) \mathcal{F} (\text{ta } \mathcal{A}) q_c q_i q_f)$

definition *nhole-ctxt-closure-reg* $\mathcal{F} \mathcal{A} =$

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

5.1.5 Non empty multihole context closure of regular tree language

abbreviation *add-eps* $\mathcal{A} e \equiv TA$ (rules \mathcal{A}) (eps $\mathcal{A} \mid \cup \mid e$)

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 $\mathcal{B} = \text{fmap-states-reg Inl (reg-Restr-}Q_f \mathcal{A})$ *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 $\mathcal{B} = \text{fmap-states-reg Inl (reg-Restr-}Q_f \mathcal{A})$ *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:*

$t \mid \in \mid ta\text{-der}' \mathcal{A} s \implies \text{ground } t \implies t = s$
 $\langle \text{proof} \rangle$

lemma *ta-det'-vars-term-id:*

$t \mid \in \mid ta\text{-der}' \mathcal{A} s \implies \text{vars-term } t \cap \text{fset } (\mathcal{Q} \mathcal{A}) = \{\} \implies t = s$
 $\langle \text{proof} \rangle$

lemma *fresh-states-ta-der'-pres:*

assumes $st: q \in \text{vars-term } s \mid q \notin \mathcal{Q} \mathcal{A}$
and reach: $t \mid \in \mid ta\text{-der}' \mathcal{A} s$
shows $q \in \text{vars-term } t$ $\langle \text{proof} \rangle$

lemma *ta-der'-states:*

$t \mid \in \mid ta\text{-der}' \mathcal{A} s \implies \text{vars-term } t \subseteq \text{vars-term } s \cup \text{fset } (\mathcal{Q} \mathcal{A})$
 $\langle \text{proof} \rangle$

lemma *ta-der'-gterm-states*:

$t \in | \text{ta-der}' \mathcal{A} \text{ (term-of-gterm } s) \implies \text{vars-term } t \subseteq \text{fset } (\mathcal{Q} \mathcal{A})$
 $\langle \text{proof} \rangle$

lemma *ta-der'-Var-funas*:

$\text{Var } q \in | \text{ta-der}' \mathcal{A} \text{ } s \implies \text{funas-term } s \subseteq \text{fset } (\text{ta-sig } \mathcal{A})$
 $\langle \text{proof} \rangle$

lemma *ta-sig-fsubsetI*:

assumes $\bigwedge r. r \in | \text{rules } \mathcal{A} \implies (r\text{-root } r, \text{length } (r\text{-lhs-states } r)) \in | \mathcal{F}$
shows $\text{ta-sig } \mathcal{A} \subseteq | \mathcal{F} \langle \text{proof} \rangle$

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

lemma *refl-ta-sig [simp]*:

$\text{ta-sig } (\text{refl-ta } \mathcal{F} \ q) = \mathcal{F}$
 $\text{ta-sig } (\text{refl-over-states-ta } \ Q \ \mathcal{F} \ \mathcal{A} \ q) = \mathcal{F}$
 $\langle \text{proof} \rangle$

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

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

$\langle \text{proof} \rangle$

lemma *refl-ta-sound*:

$s \in \mathcal{T}_G (\text{fset } \mathcal{F}) \implies q \in | \text{ta-der } (\text{refl-ta } \mathcal{F} \ q) \text{ (term-of-gterm } s)$
 $\langle \text{proof} \rangle$

lemma *reflcl-rules-args*:

$\text{length } ps = n \implies f \ ps \rightarrow p \in | \text{reflcl-rules } \mathcal{F} \ q \implies ps = \text{replicate } n \ q$
 $\langle \text{proof} \rangle$

lemma *Q-refl-ta*:

$\mathcal{Q} (\text{refl-ta } \mathcal{F} \ q) \subseteq | \{|q|\}$
 $\langle \text{proof} \rangle$

lemma *refl-ta-complete1*:

$\text{Var } p \in | \text{ta-der}' (\text{refl-ta } \mathcal{F} \ q) \text{ } s \implies p \neq q \implies s = \text{Var } p$
 $\langle \text{proof} \rangle$

lemma *refl-ta-complete2*:

$\text{Var } q \in | \text{ta-der}' (\text{refl-ta } \mathcal{F} \ q) \text{ } s \implies \text{funas-term } s \subseteq \text{fset } \mathcal{F} \wedge \text{vars-term } s \subseteq \{q\}$
 $\langle \text{proof} \rangle$

lemma *gen-reflcl-lang*:

assumes $q \notin \mathcal{Q} \ \mathcal{A}$
shows $\text{gta-lang } (\text{finsert } q \ \mathcal{Q}) (\text{gen-reflcl-automaton } \mathcal{F} \ \mathcal{A} \ q) = \text{gta-lang } \mathcal{Q} \ \mathcal{A} \cup \mathcal{T}_G (\text{fset } \mathcal{F})$

(is ?Ls = ?Rs)
 ⟨proof⟩

lemma reflcl-lang:

$gta\text{-}lang (finsert\ None\ (Some\ |\ |\ Q))\ (reflcl\text{-}automaton\ \mathcal{F}\ \mathcal{A}) = gta\text{-}lang\ Q\ \mathcal{A} \cup \mathcal{T}_G (fset\ \mathcal{F})$
 ⟨proof⟩

lemma \mathcal{L} -reflcl-reg:

$\mathcal{L}\ (reflcl\text{-}reg\ \mathcal{F}\ \mathcal{A}) = \mathcal{L}\ \mathcal{A} \cup \mathcal{T}_G (fset\ \mathcal{F})$
 ⟨proof⟩

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

lemma set-list-subset-nth-conv:

$set\ xs \subseteq A \implies i < length\ xs \implies xs\ !\ i \in A$
 ⟨proof⟩

lemma ground-gmctxt-of-mctxt-fill-holes':

$num\text{-}holes\ C = length\ ss \implies ground\text{-}mctxt\ C \implies \forall s \in set\ ss.\ ground\ s \implies fill\text{-}gholes\ (gmctxt\text{-}of\text{-}mctxt\ C)\ (map\ gterm\text{-}of\text{-}term\ ss) = gterm\text{-}of\text{-}term\ (fill\text{-}holes\ C\ ss)$
 ⟨proof⟩

lemma refl-over-states-ta-eps-trancl [simp]:

$(eps\ (refl\text{-}over\text{-}states\text{-}ta\ Q\ \mathcal{F}\ \mathcal{A}\ q))\ |^+| = eps\ (refl\text{-}over\text{-}states\text{-}ta\ Q\ \mathcal{F}\ \mathcal{A}\ q)$
 ⟨proof⟩

lemma refl-over-states-ta-epsD:

$(p, q) \in | (eps\ (refl\text{-}over\text{-}states\text{-}ta\ Q\ \mathcal{F}\ \mathcal{A}\ q)) \implies p \in | Q$
 ⟨proof⟩

lemma refl-over-states-ta-vars-term:

$q \in | ta\text{-}der\ (refl\text{-}over\text{-}states\text{-}ta\ Q\ \mathcal{F}\ \mathcal{A}\ q)\ u \implies vars\text{-}term\ u \subseteq insert\ q\ (fset\ Q)$
 ⟨proof⟩

lemmas refl-over-states-ta-vars-term' =

$refl\text{-}over\text{-}states\text{-}ta\text{-}vars\text{-}term[unfolding\ ta\text{-}der\text{-}to\text{-}ta\text{-}der'\ ta\text{-}der'\text{-}target\text{-}args\text{-}vars\text{-}term\text{-}conv,$
 $THEN\ set\text{-}list\text{-}subset\text{-}nth\text{-}conv,\ unfolding\ finsert.rep\text{-}eq[symmetric]]$

lemma refl-over-states-ta-sound:

$funas\text{-}term\ u \subseteq fset\ \mathcal{F} \implies vars\text{-}term\ u \subseteq insert\ q\ (fset\ (Q\ |\cap|\ \mathcal{Q}\ \mathcal{A})) \implies q \in | ta\text{-}der\ (refl\text{-}over\text{-}states\text{-}ta\ Q\ \mathcal{F}\ \mathcal{A}\ q)\ u$
 ⟨proof⟩

lemma gen-parallelcl-lang:

fixes $\mathcal{A} :: ('q, 'f)\ ta$
assumes $q \notin \mathcal{Q}\ \mathcal{A}$

shows $gta\text{-}lang \{|q|\}$ ($gen\text{-}parallel\text{-}closure\text{-}automaton Q \mathcal{F} \mathcal{A} q$) =
 $\{fill\text{-}gholes C ss \mid C ss. num\text{-}gholes C = length ss \wedge funas\text{-}gmctxt C \subseteq (fset \mathcal{F})$
 $\wedge (\forall i < length ss. ss ! i \in gta\text{-}lang Q \mathcal{A})\}$
 (is ?Ls = ?Rs)
 <proof>

lemma $parallelcl\text{-}gmctxt\text{-}lang$:

fixes $\mathcal{A} :: ('q, 'f) reg$
shows \mathcal{L} ($parallel\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}$) =
 $\{fill\text{-}gholes C ss \mid$
 $C ss. num\text{-}gholes C = length ss \wedge funas\text{-}gmctxt C \subseteq fset \mathcal{F} \wedge (\forall i < length$
 $ss. ss ! i \in \mathcal{L} \mathcal{A})\}$
 <proof>

lemma $parallelcl\text{-}mctxt\text{-}lang$:

shows \mathcal{L} ($parallel\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}$) =
 $\{(gterm\text{-}of\text{-}term :: ('f, 'q option) term \Rightarrow 'f gterm) (fill\text{-}holes C (map term\text{-}of\text{-}gterm$
 $ss)) \mid$
 $C ss. num\text{-}holes C = length ss \wedge ground\text{-}mctxt C \wedge funas\text{-}mctxt C \subseteq fset \mathcal{F}$
 $\wedge (\forall i < length ss. ss ! i \in \mathcal{L} \mathcal{A})\}$
 <proof>

5.1.12 Correctness of $gen\text{-}ctxt\text{-}closure\text{-}reg$ and $ctxt\text{-}closure\text{-}reg$

lemma $semantic\text{-}path\text{-}rules\text{-}rhs$:

$r \mid \in \mid semantic\text{-}path\text{-}rules Q q_c q_i q_f \Longrightarrow r\text{-}rhs r = q_f$
 <proof>

lemma $reflcl\text{-}over\text{-}single\text{-}ta\text{-}transl [simp]$:

$(eps (reflcl\text{-}over\text{-}single\text{-}ta Q \mathcal{F} q_c q_f)) \mid^+ = eps (reflcl\text{-}over\text{-}single\text{-}ta Q \mathcal{F} q_c q_f)$
 <proof>

lemma $reflcl\text{-}over\text{-}single\text{-}ta\text{-}epsD$:

$(p, q_f) \mid \in \mid eps (reflcl\text{-}over\text{-}single\text{-}ta Q \mathcal{F} q_c q_f) \Longrightarrow p \mid \in \mid Q$
 $(p, q) \mid \in \mid eps (reflcl\text{-}over\text{-}single\text{-}ta Q \mathcal{F} q_c q_f) \Longrightarrow q = q_f$
 <proof>

lemma $reflcl\text{-}over\text{-}single\text{-}ta\text{-}rules\text{-}split$:

$r \mid \in \mid rules (reflcl\text{-}over\text{-}single\text{-}ta Q \mathcal{F} q_c q_f) \Longrightarrow$
 $r \mid \in \mid reflcl\text{-}rules \mathcal{F} q_c \vee r \mid \in \mid semantic\text{-}path\text{-}rules \mathcal{F} q_c q_f q_f$
 <proof>

lemma $reflcl\text{-}over\text{-}single\text{-}ta\text{-}rules\text{-}semantic\text{-}path\text{-}rulesI$:

$r \mid \in \mid semantic\text{-}path\text{-}rules \mathcal{F} q_c q_f q_f \Longrightarrow r \mid \in \mid rules (reflcl\text{-}over\text{-}single\text{-}ta Q \mathcal{F} q_c$
 $q_f)$
 <proof>

lemma $semantic\text{-}path\text{-}rules\text{-}fmember [intro]$:

$TA\text{-}rule f qs q \mid \in \mid semantic\text{-}path\text{-}rules \mathcal{F} q_c q_i q_f \longleftrightarrow (\exists n i. (f, n) \mid \in \mid \mathcal{F} \wedge i <$

$n \wedge q = q_f \wedge$
 $(qs = (\text{replicate } n \ q_c)[i := q_i])$ (is ?Ls \longleftrightarrow ?Rs)
 ⟨proof⟩

lemma *semantic-path-rules-fmmemberD*:

$r \mid \in \mid \text{semantic-path-rules } \mathcal{F} \ q_c \ q_i \ q_f \implies (\exists \ n \ i. (r\text{-root } r, n) \mid \in \mid \mathcal{F} \wedge i < n \wedge$
 $r\text{-rhs } r = q_f \wedge$
 $(r\text{-lhs-states } r = (\text{replicate } n \ q_c)[i := q_i]))$
 ⟨proof⟩

lemma *reflcl-over-single-ta-vars-term-qc*:

$q_c \neq q_f \implies q_c \mid \in \mid \text{ta-der } (\text{reflcl-over-single-ta } Q \ \mathcal{F} \ q_c \ q_f) \ u \implies$
 $\text{vars-term-list } u = \text{replicate } (\text{length } (\text{vars-term-list } u)) \ q_c$
 ⟨proof⟩

lemma *reflcl-over-single-ta-vars-term*:

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

lemma *refl-ta-reflcl-over-single-ta-mono*:

$q \mid \in \mid \text{ta-der } (\text{refl-ta } \mathcal{F} \ q) \ t \implies q \mid \in \mid \text{ta-der } (\text{reflcl-over-single-ta } Q \ \mathcal{F} \ q \ q_f) \ t$
 ⟨proof⟩

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

assumes $\text{funas-gctxt } C \subseteq \text{fset } \mathcal{F} \ q \mid \in \mid Q$
shows $q_f \mid \in \mid \text{ta-der } (\text{reflcl-over-single-ta } Q \ \mathcal{F} \ q_c \ q_f) \ (\text{ctxt-of-gctxt } C) \langle \text{Var } q \rangle$
 ⟨proof⟩

lemma *reflcl-over-single-ta-sig*: $\text{ta-sig } (\text{reflcl-over-single-ta } Q \ \mathcal{F} \ q_c \ q_f) \mid \subseteq \mid \mathcal{F}$
 ⟨proof⟩

lemma *gen-gctxtcl-lang*:

assumes $q_c \mid \notin \mid Q \ \mathcal{A}$ **and** $q_f \mid \notin \mid Q \ \mathcal{A}$ **and** $q_c \mid \notin \mid Q$ **and** $q_c \neq q_f$
shows $\text{gta-lang } \{|q_f|\} \ (\text{gen-ctxt-closure-automaton } Q \ \mathcal{F} \ \mathcal{A} \ q_c \ q_f) =$
 $\{C \langle s \rangle_G \mid C \ s. \text{funas-gctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \text{gta-lang } Q \ \mathcal{A}\}$
 (is ?Ls = ?Rs)
 ⟨proof⟩

lemma *gen-gctxt-closure-sound*:

fixes $\mathcal{A} :: ('q, 'f) \text{reg}$
assumes $q_c \mid \notin \mid Q_r \ \mathcal{A}$ **and** $q_f \mid \notin \mid Q_r \ \mathcal{A}$ **and** $q_c \mid \notin \mid \text{fin } \mathcal{A}$ **and** $q_c \neq q_f$
shows $\mathcal{L} \ (\text{gen-ctxt-closure-reg } \mathcal{F} \ \mathcal{A} \ q_c \ q_f) = \{C \langle s \rangle_G \mid C \ s. \text{funas-gctxt } C \subseteq \text{fset}$
 $\mathcal{F} \wedge s \in \mathcal{L} \ \mathcal{A}\}$
 ⟨proof⟩

lemma *gen-ctxt-closure-sound*:

fixes $\mathcal{A} :: ('q, 'f) \text{ reg}$
assumes $q_c \notin \mathcal{Q}_r \mathcal{A}$ **and** $q_f \notin \mathcal{Q}_r \mathcal{A}$ **and** $q_c \notin \text{fin } \mathcal{A}$ **and** $q_c \neq q_f$
shows $\mathcal{L} (\text{gen-ctxt-closure-reg } \mathcal{F} \mathcal{A} q_c q_f) =$
 $\{(gterm\text{-of-term} :: ('f, 'q) \text{ term} \Rightarrow 'f \text{ gterm}) C \langle \text{term-of-gterm } s \rangle \mid C \text{ s. ground-ctxt}$
 $C \wedge \text{funas-ctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \mathcal{L} \mathcal{A}\}$
 $\langle \text{proof} \rangle$

lemma *gctxt-closure-lang*:
shows $\mathcal{L} (\text{ctxt-closure-reg } \mathcal{F} \mathcal{A}) =$
 $\{C \langle s \rangle_G \mid C \text{ s. funas-gctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \mathcal{L} \mathcal{A}\}$
 $\langle \text{proof} \rangle$

lemma *ctxt-closure-lang*:
shows $\mathcal{L} (\text{ctxt-closure-reg } \mathcal{F} \mathcal{A}) =$
 $\{(gterm\text{-of-term} :: ('f, 'q + \text{bool}) \text{ term} \Rightarrow 'f \text{ gterm}) C \langle \text{term-of-gterm } s \rangle \mid$
 $C \text{ s. ground-ctxt } C \wedge \text{funas-ctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \mathcal{L} \mathcal{A}\}$
 $\langle \text{proof} \rangle$

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

lemma *reflcl-over-nhole-ctxt-ta-vars-term-qc*:
 $q_c \neq q_f \implies q_c \neq q_i \implies q_c \in \text{ta-der} (\text{reflcl-over-nhole-ctxt-ta } Q \mathcal{F} q_c q_i q_f) u$
 \implies
 $\text{vars-term-list } u = \text{replicate} (\text{length} (\text{vars-term-list } u)) q_c$
 $\langle \text{proof} \rangle$

lemma *reflcl-over-nhole-ctxt-ta-vars-term-Var*:
assumes *disj*: $q_c \notin Q$ $q_f \notin Q$ $q_c \neq q_f$ $q_i \neq q_f$ $q_c \neq q_i$
and *reach*: $q_i \in \text{ta-der} (\text{reflcl-over-nhole-ctxt-ta } Q \mathcal{F} q_c q_i q_f) u$
shows $(\exists q. q \in \text{finsert } q_i Q \wedge u = \text{Var } q) \langle \text{proof} \rangle$

lemma *reflcl-over-nhole-ctxt-ta-vars-term*:
assumes *disj*: $q_c \notin Q$ $q_f \notin Q$ $q_c \neq q_f$ $q_i \neq q_f$ $q_c \neq q_i$
and *reach*: $q_f \in \text{ta-der} (\text{reflcl-over-nhole-ctxt-ta } Q \mathcal{F} q_c q_i q_f) u$
shows $(\exists i q. i < \text{length} (\text{vars-term-list } u) \wedge q \in \{|q_i, q_f|\} \cup Q \wedge \text{vars-term-list}$
 $u = (\text{replicate} (\text{length} (\text{vars-term-list } u)) q_c)[i := q])$
 $\langle \text{proof} \rangle$

lemma *reflcl-over-nhole-ctxt-ta-mono*:
 $q \in \text{ta-der} (\text{refl-ta } \mathcal{F} q) t \implies q \in \text{ta-der} (\text{reflcl-over-nhole-ctxt-ta } Q \mathcal{F} q q_i$
 $q_f) t$
 $\langle \text{proof} \rangle$

lemma *reflcl-over-nhole-ctxt-ta-sound*:
assumes $\text{funas-gctxt } C \subseteq \text{fset } \mathcal{F}$ $C \neq \text{GHole } q \in Q$
shows $q_f \in \text{ta-der} (\text{reflcl-over-nhole-ctxt-ta } Q \mathcal{F} q_c q_i q_f) (\text{ctxt-of-gctxt } C) \langle \text{Var}$
 $q \rangle \langle \text{proof} \rangle$

lemma *reflcl-over-nhole-ctxt-ta-sig*: *ta-sig* (*reflcl-over-nhole-ctxt-ta* $Q \mathcal{F} q_c q_i q_f$)
 $|\subseteq| \mathcal{F}$
 ⟨*proof*⟩

lemma *gen-nhole-gctxt-closure-lang*:
 assumes $q_c \notin Q \mathcal{A} \quad q_i \notin Q \mathcal{A} \quad q_f \notin Q \mathcal{A}$
 and $q_c \notin Q \quad q_f \notin Q$
 and $q_c \neq q_i \quad q_c \neq q_f \quad q_i \neq q_f$
 shows *gta-lang* $\{|q_f|\}$ (*gen-nhole-ctxt-closure-automaton* $Q \mathcal{F} \mathcal{A} q_c q_i q_f$) =
 $\{C\langle s \rangle_G \mid C \text{ s. } C \neq \text{GHole} \wedge \text{funas-gctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \text{gta-lang } Q \mathcal{A}\}$
 (is $?Ls = ?Rs$)
 ⟨*proof*⟩

lemma *gen-nhole-gctxt-closure-sound*:
 assumes $q_c \notin Q_r \mathcal{A} \quad q_i \notin Q_r \mathcal{A} \quad q_f \notin Q_r \mathcal{A}$
 and $q_c \notin (\text{fin } \mathcal{A}) \quad q_f \notin (\text{fin } \mathcal{A})$
 and $q_c \neq q_i \quad q_c \neq q_f \quad q_i \neq q_f$
 shows \mathcal{L} (*gen-nhole-ctxt-closure-reg* $\mathcal{F} \mathcal{A} q_c q_i q_f$) =
 $\{C\langle s \rangle_G \mid C \text{ s. } C \neq \text{GHole} \wedge \text{funas-gctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \mathcal{L} \mathcal{A}\}$
 ⟨*proof*⟩

lemma *nhole-ctxtcl-lang*:
 \mathcal{L} (*nhole-ctxt-closure-reg* $\mathcal{F} \mathcal{A}$) =
 $\{C\langle s \rangle_G \mid C \text{ s. } C \neq \text{GHole} \wedge \text{funas-gctxt } C \subseteq \text{fset } \mathcal{F} \wedge s \in \mathcal{L} \mathcal{A}\}$
 ⟨*proof*⟩

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*:
 $f \text{ ps} \rightarrow q \mid \in \mid \text{reflcl-rules } \mathcal{F} q_c \implies q = q_c$
 ⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-vars-term*:
 assumes $q \mid \in \mid \text{ta-der} (\text{reflcl-over-nhole-mctxt-ta } Q \mathcal{F} q_c q_i q_f) t$
 and $q_c \notin Q \quad q \neq q_c \quad q_f \neq q_c \quad q_i \neq q_c$
 shows *vars-term* $t \neq \{\}$ ⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-Fun*:
 assumes $q_f \mid \in \mid \text{ta-der} (\text{reflcl-over-nhole-mctxt-ta } Q \mathcal{F} q_c q_i q_f) t \quad t \neq \text{Var } q_f$
 and $q_f \neq q_c \quad q_f \neq q_i$
 shows *is-Fun* t ⟨*proof*⟩

lemma *rule-states-reflcl-rulesD*:
 $p \mid \in \mid \text{rule-states} (\text{reflcl-rules } \mathcal{F} q) \implies p = q$
 ⟨*proof*⟩

lemma *rule-states-semantic-path-rulesD*:

$p \in \text{rule-states (semantic-path-rules } \mathcal{F} \ q_c \ q_i \ q_f) \implies p = q_c \vee p = q_i \vee p = q_f$
 $\langle \text{proof} \rangle$

lemma *Q-reflcl-over-nhole-mctxt-ta*:

$\mathcal{Q} (\text{reflcl-over-nhole-mctxt-ta } Q \ \mathcal{F} \ q_c \ q_i \ q_f) \subseteq \mathcal{Q} \cup \{|q_c, q_i, q_f|\}$
 $\langle \text{proof} \rangle$

lemma *reflcl-over-nhole-mctxt-ta-vars-term-subset-eq*:

assumes $q \in \text{ta-der (reflcl-over-nhole-mctxt-ta } Q \ \mathcal{F} \ q_c \ q_i \ q_f) \ t \ q = q_f \vee q = q_i$
shows $\text{vars-term } t \subseteq \{q_c, q_i, q_f\} \cup \text{fset } Q$
 $\langle \text{proof} \rangle$

lemma *sig-reflcl-over-nhole-mctxt-ta [simp]*:

$\text{ta-sig (reflcl-over-nhole-mctxt-ta } Q \ \mathcal{F} \ q_c \ q_i \ q_f) = \mathcal{F}$
 $\langle \text{proof} \rangle$

lemma *reflcl-over-nhole-mctxt-ta-aux-sound*:

assumes $\text{funas-term } t \subseteq \text{fset } \mathcal{F} \ \text{vars-term } t \subseteq \text{fset } Q$
shows $q_c \in \text{ta-der (reflcl-over-nhole-mctxt-ta } Q \ \mathcal{F} \ q_c \ q_i \ q_f) \ t \ \langle \text{proof} \rangle$

lemma *reflcl-over-nhole-mctxt-ta-sound*:

assumes $\text{funas-term } t \subseteq \text{fset } \mathcal{F} \ \text{vars-term } t \subseteq \text{fset } Q \ \text{vars-term } t \neq \{\}$
shows $(\text{is-Var } t \longrightarrow q_i \in \text{ta-der (reflcl-over-nhole-mctxt-ta } Q \ \mathcal{F} \ q_c \ q_i \ q_f) \ t) \wedge$
 $(\text{is-Fun } t \longrightarrow q_f \in \text{ta-der (reflcl-over-nhole-mctxt-ta } Q \ \mathcal{F} \ q_c \ q_i \ q_f) \ t) \ \langle \text{proof} \rangle$

lemma *gen-nhole-gmctxt-closure-lang*:

assumes $q_c \notin \mathcal{Q} \ \mathcal{A} \ \text{and } q_i \notin \mathcal{Q} \ \mathcal{A} \ q_f \notin \mathcal{Q} \ \mathcal{A}$
and $q_c \notin Q \ q_f \neq q_c \ q_f \neq q_i \ q_i \neq q_c$
shows $\text{gta-lang } \{|q_f|\} (\text{gen-nhole-mctxt-closure-automaton } Q \ \mathcal{F} \ \mathcal{A} \ q_c \ q_i \ q_f) =$
 $\{ \text{fill-gholes } C \ ss \mid$
 $C \ ss. \ 0 < \text{num-gholes } C \wedge \text{num-gholes } C = \text{length } ss \wedge C \neq \text{GMHole} \wedge$
 $\text{funas-gmctxt } C \subseteq \text{fset } \mathcal{F} \wedge (\forall i < \text{length } ss. \ ss ! i \in \text{gta-lang } Q \ \mathcal{A}) \}$
(is ?Ls = ?Rs)
 $\langle \text{proof} \rangle$

lemma *nhole-gmctxt-closure-lang*:

$\mathcal{L} (\text{nhole-mctxt-closure-reg } \mathcal{F} \ \mathcal{A}) =$
 $\{ \text{fill-gholes } C \ ss \mid C \ ss. \ \text{num-gholes } C = \text{length } ss \wedge 0 < \text{num-gholes } C \wedge C \neq$
 $\text{GMHole} \wedge$
 $\text{funas-gmctxt } C \subseteq \text{fset } \mathcal{F} \wedge (\forall i < \text{length } ss. \ ss ! i \in \mathcal{L} \ \mathcal{A}) \}$
(is ?Ls = ?Rs)
 $\langle \text{proof} \rangle$

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

lemma *gen-gmctxt-closure-lang*:

```

assumes  $q_c \notin Q \mathcal{A}$  and  $q_i \notin Q \mathcal{A}$   $q_f \notin Q \mathcal{A}$ 
and  $disj: q_c \notin Q$   $q_f \neq q_c$   $q_f \neq q_i$   $q_i \neq q_c$ 
shows  $gta\text{-}lang \{|q_f, q_i|\}$  (gen-nhole-mctxt-closure-automaton  $Q \mathcal{F} \mathcal{A} q_c q_i q_f$ )
=
{ fill-gholes  $C ss$  |
   $C ss. 0 < num\text{-}gholes C \wedge num\text{-}gholes C = length ss \wedge$ 
   $funas\text{-}gmctxt C \subseteq fset \mathcal{F} \wedge (\forall i < length ss. ss ! i \in gta\text{-}lang Q \mathcal{A})$  }
(is  $?Ls = ?Rs$ )
<proof>

```

```

lemma gmctxt-closure-lang:
 $\mathcal{L} (mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}) =$ 
{ fill-gholes  $C ss$  |  $C ss. num\text{-}gholes C = length ss \wedge 0 < num\text{-}gholes C \wedge$ 
   $funas\text{-}gmctxt C \subseteq fset \mathcal{F} \wedge (\forall i < length ss. ss ! i \in \mathcal{L} \mathcal{A})$  }
(is  $?Ls = ?Rs$ )
<proof>

```

5.1.16 Correctness of *nhole-mctxt-reflcl-reg*

```

lemma nhole-mctxt-reflcl-lang:
 $\mathcal{L} (nhole\text{-}mctxt\text{-}reflcl\text{-}reg \mathcal{F} \mathcal{A}) = \mathcal{L} (nhole\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}) \cup \mathcal{T}_G (fset \mathcal{F})$ 
<proof>
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  $\equiv$  [cl-state, tr-state, fin-state, fin-clstate]
definition cEnum-cl-states :: (cl-states list  $\times$  ((cl-states  $\Rightarrow$  bool)  $\Rightarrow$  bool)  $\times$ 
((cl-states  $\Rightarrow$  bool)  $\Rightarrow$  bool)) option
  where cEnum-cl-states = Some (cl-all-list, ( $\lambda$  P. list-all P cl-all-list), ( $\lambda$  P.
list-ex P cl-all-list))
instance
   $\langle$ proof $\rangle$ 
end

lemma infinite-bot-term-UNIV[simp, intro]: infinite (UNIV :: 'f bot-term set)
   $\langle$ proof $\rangle$ 

lemma finite-cl-states: (UNIV :: cl-states set) = {cl-state, tr-state, fin-state, fin-clstate}
   $\langle$ proof $\rangle$ 

instantiation cl-states :: card-UNIV begin
definition finite-UNIV = Phantom(cl-states) True
definition card-UNIV = Phantom(cl-states) 4
instance
   $\langle$ proof $\rangle$ 
end

instantiation bot-term :: (type) finite-UNIV
begin
definition finite-UNIV = Phantom('a bot-term) False
instance
   $\langle$ proof $\rangle$ 
end

instantiation bot-term :: (compare) cproper-interval
begin
definition cproper-interval = ( $\lambda$  ( - :: 'a bot-term option) - . False)
instance  $\langle$ proof $\rangle$ 
end

instantiation cl-states :: cproper-interval
begin

definition cproper-interval-cl-states :: cl-states option  $\Rightarrow$  cl-states option  $\Rightarrow$  bool
  where cproper-interval-cl-states x y =
    (case ID CCOMPARE(cl-states) of Some f  $\Rightarrow$ 
      (case x of None  $\Rightarrow$ 
        (case y of None  $\Rightarrow$  True | Some c  $\Rightarrow$  list-ex ( $\lambda$  x. (lt-of-comp f) x c) cl-all-list)
      | Some c  $\Rightarrow$ 
        (case y of None  $\Rightarrow$  list-ex ( $\lambda$  x. (lt-of-comp f) c x) cl-all-list
        | Some d  $\Rightarrow$  (filter ( $\lambda$  x. (lt-of-comp f) x d  $\wedge$  (lt-of-comp f) c x) cl-all-list)  $\neq$ 

```

()))

instance

<proof>

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* \Rightarrow (*'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* \Rightarrow (*'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* \geq *s*}

<proof>

lemma *supt-list-sound [simp]*:

set (*supt-list t*) = {*s*. *t* \triangleright *s*}

<proof>

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* \wedge *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* *<proof>*

lemma [*simp*]: *mergeP-impl s t* \longleftrightarrow (*s*, *t*) \in *mergeP* (**is** ?*LS* = ?*RS*)

<proof>

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* \longleftrightarrow (*s*, *t*) ∈ *bless-eq* (**is** ?*RS* = ?*LS*)
 ⟨*proof*⟩

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*)
 ⟨*proof*⟩

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*⟩

abbreviation *check-istance-lhs* **where**

check-istance-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*⟩

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-istance-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* *F* ≡ *concat* (*map* (*nf-rule-impl* (*states-impl* *R*) (*map term-to-bot-term*
R) (*psubt-bot-impl* *R*)) *F*)

lemma *nf-rules-in-impl*:

assumes *TA-rule* *f* *qs* *q* |∈| *nf-rules* (*fset-of-list* *R*) (*fset-of-list* *F*)

shows *TA-rule* *f* *qs* *q* |∈| *fset-of-list* (*nf-rules-impl* *R* *F*)

⟨*proof*⟩

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>

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>

lemma *fstates-code*[*code*]:

fstates R = fset-of-list (states-impl (sorted-list-of-fset R))

<proof>

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}))) {||}

<proof>

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) \mid C s t. P C \wedge (s, t) \in \mathcal{R}\}

definition *gmctxtex-onp* **where**

gmctxtex-onp P \mathcal{R} = \{(fill-gholes C ss, fill-gholes C ts) \mid C ss ts.

num-gholes C = length ss \wedge length ss = length ts \wedge P C \wedge (\forall i < length ts. (ss ! i, ts ! i) \in \mathcal{R})\}

definition *compatible-p* **where**

compatible-p P Q \equiv (\forall C. P C \longrightarrow Q (gmctxtex-of-gctxtex C))

7.1 Elimination and introduction rules for the extensions

lemma *gctxtex-onpE* [*elim*]:

assumes $(s, t) \in gctxtex-onp P \mathcal{R}$
obtains $C u v$ **where** $s = C\langle u \rangle_G t = C\langle v \rangle_G P C (u, v) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gctxtex-onp-neq-rootE* [elim]:
assumes $(GFun f ss, GFun g ts) \in gctxtex-onp P \mathcal{R}$ **and** $f \neq g$
shows $(GFun f ss, GFun g ts) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gctxtex-onp-neq-lengthE* [elim]:
assumes $(GFun f ss, GFun g ts) \in gctxtex-onp P \mathcal{R}$ **and** $length\ ss \neq length\ ts$
shows $(GFun f ss, GFun g ts) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gmctxtex-onpE* [elim]:
assumes $(s, t) \in gmctxtex-onp P \mathcal{R}$
obtains $C us vs$ **where** $s = fill-gholes\ C\ us\ t = fill-gholes\ C\ vs\ num-gholes\ C =$
 $length\ us$
 $length\ us = length\ vs\ P\ C\ \forall\ i < length\ vs.\ (us\ !\ i,\ vs\ !\ i) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gmctxtex-onpE2* [elim]:
assumes $(s, t) \in gmctxtex-onp P \mathcal{R}$
obtains $C us vs$ **where** $s =_{Gf}\ (C,\ us)\ t =_{Gf}\ (C,\ vs)$
 $P\ C\ \forall\ i < length\ vs.\ (us\ !\ i,\ vs\ !\ i) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gmctxtex-onp-neq-rootE* [elim]:
assumes $(GFun f ss, GFun g ts) \in gmctxtex-onp P \mathcal{R}$ **and** $f \neq g$
shows $(GFun f ss, GFun g ts) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gmctxtex-onp-neq-lengthE* [elim]:
assumes $(GFun f ss, GFun g ts) \in gmctxtex-onp P \mathcal{R}$ **and** $length\ ss \neq length\ ts$
shows $(GFun f ss, GFun g ts) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gmctxtex-onp-listE*:
assumes $\forall\ i < length\ ts.\ (ss\ !\ i,\ ts\ !\ i) \in gmctxtex-onp\ Q\ \mathcal{R}\ length\ ss = length\ ts$
obtains $Ds\ sss\ tss$ **where** $length\ ts = length\ Ds\ length\ Ds = length\ sss\ length\ sss = length\ tss$
 $\forall\ i < length\ tss.\ length\ (sss\ !\ i) = length\ (tss\ !\ i)\ \forall\ D \in set\ Ds.\ Q\ D$
 $\forall\ i < length\ tss.\ ss\ !\ i =_{Gf}\ (Ds\ !\ i,\ sss\ !\ i)\ \forall\ i < length\ tss.\ ts\ !\ i =_{Gf}\ (Ds\ !\ i,\ tss\ !\ i)$
 $\forall\ i < length\ (concat\ tss).\ (concat\ sss\ !\ i,\ concat\ tss\ !\ i) \in \mathcal{R}$
 $\langle proof \rangle$

lemma *gmctxtex-onp-doubleE* [elim]:

assumes $(s, t) \in \text{gmctxtex-onp } P \text{ (gmctxtex-onp } Q \mathcal{R})$
obtains $C \text{ } Ds \text{ } ss \text{ } ts \text{ } us \text{ } vs$ **where** $s =_{Gf} (C, ss) \text{ } t =_{Gf} (C, ts) \text{ } P \text{ } C \forall D \in \text{set } Ds. Q \text{ } D$
 $\text{num-gholes } C = \text{length } Ds \text{ } \text{length } Ds = \text{length } ss \text{ } \text{length } ss = \text{length } ts \text{ } \text{length } ts$
 $= \text{length } us \text{ } \text{length } us = \text{length } vs$
 $\forall i < \text{length } Ds. ss ! i =_{Gf} (Ds ! i, us ! i) \wedge ts ! i =_{Gf} (Ds ! i, vs ! i)$
 $\forall i < \text{length } Ds. \forall j < \text{length } (vs ! i). (us ! i ! j, vs ! i ! j) \in \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma gctxtex-onpI [intro]:
assumes $P \text{ } C$ **and** $(s, t) \in \mathcal{R}$
shows $(C \langle s \rangle_G, C \langle t \rangle_G) \in \text{gctxtex-onp } P \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma gmctxtex-onpI [intro]:
assumes $P \text{ } C$ **and** $\text{num-gholes } C = \text{length } us$ **and** $\text{length } us = \text{length } vs$
and $\forall i < \text{length } vs. (us ! i, vs ! i) \in \mathcal{R}$
shows $(\text{fill-gholes } C \text{ } us, \text{fill-gholes } C \text{ } vs) \in \text{gmctxtex-onp } P \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma gmctxtex-onp-arg-monoI:
assumes $P \text{ } GMHole$
shows $\mathcal{R} \subseteq \text{gmctxtex-onp } P \mathcal{R} \langle \text{proof} \rangle$

lemma gmctxtex-onpI2 [intro]:
assumes $P \text{ } C$ **and** $s =_{Gf} (C, ss) \text{ } t =_{Gf} (C, ts)$
and $\forall i < \text{length } ts. (ss ! i, ts ! i) \in \mathcal{R}$
shows $(s, t) \in \text{gmctxtex-onp } P \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma gctxtex-onp-hold-cond [simp]:
 $(s, t) \in \text{gctxtex-onp } P \mathcal{R} \implies \text{groot } s \neq \text{groot } t \implies P \square_G$
 $(s, t) \in \text{gctxtex-onp } P \mathcal{R} \implies \text{length } (\text{gargs } s) \neq \text{length } (\text{gargs } t) \implies P \square_G$
 $\langle \text{proof} \rangle$

7.2 Monotonicity rules for the extensions

lemma gctxtex-onp-rel-mono:
 $\mathcal{L} \subseteq \mathcal{R} \implies \text{gctxtex-onp } P \mathcal{L} \subseteq \text{gctxtex-onp } P \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma gmctxtex-onp-rel-mono:
 $\mathcal{L} \subseteq \mathcal{R} \implies \text{gmctxtex-onp } P \mathcal{L} \subseteq \text{gmctxtex-onp } P \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma compatible-p-gctxtex-gmctxtex-subseteq [dest]:
 $\text{compatible-p } P \text{ } Q \implies \text{gctxtex-onp } P \mathcal{R} \subseteq \text{gmctxtex-onp } Q \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma *compatible-p-mono1*:

$$P \leq R \implies \text{compatible-p } R \ Q \implies \text{compatible-p } P \ Q$$

<proof>

lemma *compatible-p-mono2*:

$$Q \leq R \implies \text{compatible-p } P \ Q \implies \text{compatible-p } P \ R$$

<proof>

lemma *gctxtex-onp-mono* [intro]:

$$P \leq Q \implies \text{gctxtex-onp } P \ \mathcal{R} \subseteq \text{gctxtex-onp } Q \ \mathcal{R}$$

<proof>

lemma *gctxtex-onp-mem*:

$$P \leq Q \implies (s, t) \in \text{gctxtex-onp } P \ \mathcal{R} \implies (s, t) \in \text{gctxtex-onp } Q \ \mathcal{R}$$

<proof>

lemma *gmctxtex-onp-mono* [intro]:

$$P \leq Q \implies \text{gmctxtex-onp } P \ \mathcal{R} \subseteq \text{gmctxtex-onp } Q \ \mathcal{R}$$

<proof>

lemma *gmctxtex-onp-mem*:

$$P \leq Q \implies (s, t) \in \text{gmctxtex-onp } P \ \mathcal{R} \implies (s, t) \in \text{gmctxtex-onp } Q \ \mathcal{R}$$

<proof>

lemma *gctxtex-eqI* [intro]:

$$P = Q \implies \mathcal{R} = \mathcal{L} \implies \text{gctxtex-onp } P \ \mathcal{R} = \text{gctxtex-onp } Q \ \mathcal{L}$$

<proof>

lemma *gmctxtex-eqI* [intro]:

$$P = Q \implies \mathcal{R} = \mathcal{L} \implies \text{gmctxtex-onp } P \ \mathcal{R} = \text{gmctxtex-onp } Q \ \mathcal{L}$$

<proof>

7.3 Relation swap and converse

lemma *swap-gctxtex-onp*:

$$\text{gctxtex-onp } P \ (\text{prod.swap } \mathcal{R}) = \text{prod.swap } \mathcal{R} \ (\text{gctxtex-onp } P \ \mathcal{R})$$

<proof>

lemma *swap-gmctxtex-onp*:

$$\text{gmctxtex-onp } P \ (\text{prod.swap } \mathcal{R}) = \text{prod.swap } \mathcal{R} \ (\text{gmctxtex-onp } P \ \mathcal{R})$$

<proof>

lemma *converse-gctxtex-onp*:

$$(\text{gctxtex-onp } P \ \mathcal{R})^{-1} = \text{gctxtex-onp } P \ (\mathcal{R}^{-1})$$

<proof>

lemma *converse-gmctxtex-onp*:

$$(\text{gmctxtex-onp } P \ \mathcal{R})^{-1} = \text{gmctxtex-onp } P \ (\mathcal{R}^{-1})$$

<proof>

7.4 Subset equivalence for context extensions over predicates

lemma *gctxtex-onp-closure-predI*:

assumes $\bigwedge C s t. P C \implies (s, t) \in \mathcal{R} \implies (C\langle s \rangle_G, C\langle t \rangle_G) \in \mathcal{R}$

shows *gctxtex-onp* $P \mathcal{R} \subseteq \mathcal{R}$

<proof>

lemma *gmctxtex-onp-closure-predI*:

assumes $\bigwedge C ss ts. P C \implies \text{num-gholes } C = \text{length } ss \implies \text{length } ss = \text{length } ts \implies$

$(\forall i < \text{length } ts. (ss ! i, ts ! i) \in \mathcal{R}) \implies (\text{fill-gholes } C ss, \text{fill-gholes } C ts) \in \mathcal{R}$

shows *gmctxtex-onp* $P \mathcal{R} \subseteq \mathcal{R}$

<proof>

lemma *gctxtex-onp-closure-predE*:

assumes *gctxtex-onp* $P \mathcal{R} \subseteq \mathcal{R}$

shows $\bigwedge C s t. P C \implies (s, t) \in \mathcal{R} \implies (C\langle s \rangle_G, C\langle t \rangle_G) \in \mathcal{R}$

<proof>

lemma *gctxtex-closure [intro]*:

$P \sqsubseteq_G \implies \mathcal{R} \subseteq \text{gctxtex-onp } P \mathcal{R}$

<proof>

lemma *gmctxtex-closure [intro]*:

assumes $P \text{ GMHole}$

shows $\mathcal{R} \subseteq (\text{gmctxtex-onp } P \mathcal{R})$

<proof>

lemma *gctxtex-pred-cmp-subseteq*:

assumes $\bigwedge C D. P C \implies Q D \implies Q (C \circ_{Gc} D)$

shows *gctxtex-onp* $P (\text{gctxtex-onp } Q \mathcal{R}) \subseteq \text{gctxtex-onp } Q \mathcal{R}$

<proof>

lemma *gctxtex-pred-cmp-subseteq2*:

assumes $\bigwedge C D. P C \implies Q D \implies P (C \circ_{Gc} D)$

shows *gctxtex-onp* $P (\text{gctxtex-onp } Q \mathcal{R}) \subseteq \text{gctxtex-onp } P \mathcal{R}$

<proof>

lemma *gmctxtex-pred-cmp-subseteq*:

assumes $\bigwedge C D. C \leq D \implies P C \implies (\forall Ds \in \text{set } (\text{sup-gmctxt-args } C D). Q Ds) \implies Q D$

shows *gmctxtex-onp* $P (\text{gmctxtex-onp } Q \mathcal{R}) \subseteq \text{gmctxtex-onp } Q \mathcal{R}$ (**is** $?Ls \subseteq ?Rs$)

<proof>

lemma *gmctxtex-pred-cmp-subseteq2*:

assumes $\bigwedge C D. C \leq D \implies P C \implies (\forall Ds \in \text{set } (\text{sup-gmctxt-args } C D). Q Ds) \implies P D$

shows *gmctxtex-onp* $P (\text{gmctxtex-onp } Q \mathcal{R}) \subseteq \text{gmctxtex-onp } P \mathcal{R}$ (**is** $?Ls \subseteq ?Rs$)

<proof>

lemma *gctxtex-onp-idem* [simp]:

assumes $P \sqsubseteq_G$ and $\bigwedge C D. P C \implies Q D \implies Q (C \circ_{Gc} D)$

shows $gctxtex\text{-onp } P (gctxtex\text{-onp } Q \mathcal{R}) = gctxtex\text{-onp } Q \mathcal{R}$ (is ?Ls = ?Rs)
 ⟨proof⟩

lemma *gctxtex-onp-idem2* [simp]:

assumes $Q \sqsubseteq_G$ and $\bigwedge C D. P C \implies Q D \implies P (C \circ_{Gc} D)$

shows $gctxtex\text{-onp } P (gctxtex\text{-onp } Q \mathcal{R}) = gctxtex\text{-onp } P \mathcal{R}$ (is ?Ls = ?Rs)
 ⟨proof⟩

lemma *gmctxtex-onp-idem* [simp]:

assumes $P \text{ GMHole}$

and $\bigwedge C D. C \leq D \implies P C \implies (\forall Ds \in \text{set } (sup\text{-gmctxt-args } C D). Q Ds) \implies Q D$

shows $gmctxtex\text{-onp } P (gmctxtex\text{-onp } Q \mathcal{R}) = gmctxtex\text{-onp } Q \mathcal{R}$
 ⟨proof⟩

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* $C \mathcal{F} Q \equiv (\forall D. gmctxt\text{-closing } C D \implies num\text{-gholes } D = 1 \implies funas\text{-gmctxt } D \subseteq \mathcal{F} \implies Q (gctxt\text{-of-gmctxt } D))$

lemma *gmctxt-p-invE*:

$gmctxt\text{-p-inv } C \mathcal{F} Q \implies C \leq D \implies ghole\text{-poss } D \subseteq ghole\text{-poss } C \implies num\text{-gholes } D = 1 \implies$

$funas\text{-gmctxt } D \subseteq \mathcal{F} \implies Q (gctxt\text{-of-gmctxt } D)$
 ⟨proof⟩

lemma *gmctxt-closing-gmctxt-p-inv-comp*:

$gmctxt\text{-closing } C D \implies gmctxt\text{-p-inv } C \mathcal{F} Q \implies gmctxt\text{-p-inv } D \mathcal{F} Q$
 ⟨proof⟩

lemma *GMHole-gmctxt-p-inv-GHole* [simp]:

$gmctxt\text{-p-inv } GMHole \mathcal{F} Q \implies Q \sqsubseteq_G$
 ⟨proof⟩

lemma *gmctxtex-onp-gctxtex-onp-trancl*:

assumes $sig: \bigwedge C. P C \implies 0 < num\text{-gholes } C \wedge funas\text{-gmctxt } C \subseteq \mathcal{F} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$

and $\bigwedge C. P C \implies gmctxt\text{-p-inv } C \mathcal{F} Q$

shows $gmctxtex\text{-onp } P \mathcal{R} \subseteq (gctxtex\text{-onp } Q \mathcal{R})^+$
 ⟨proof⟩

lemma *gmctxtex-onp-gctxtex-onp-rtrancl*:
assumes *sig*: $\bigwedge C. P C \implies \text{funas-gmctxt } C \subseteq \mathcal{F} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$
and $\bigwedge C D. P C \implies \text{gmctxt-p-inv } C \mathcal{F} Q$
shows $\text{gmctxtex-onp } P \mathcal{R} \subseteq (\text{gctxtex-onp } Q \mathcal{R})^*$
<proof>

lemma *rtrancl-gmctxtex-onp-rtrancl-gctxtex-onp-eq*:
assumes *sig*: $\bigwedge C. P C \implies \text{funas-gmctxt } C \subseteq \mathcal{F} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$
and $\bigwedge C D. P C \implies \text{gmctxt-p-inv } C \mathcal{F} Q$
and *compatible-p* $Q P$
shows $(\text{gmctxtex-onp } P \mathcal{R})^* = (\text{gctxtex-onp } Q \mathcal{R})^*$ (**is** $?Ls^* = ?Rs^*$)
<proof>

7.6 Extensions to reflexive transitive closures

lemma *gctxtex-onp-substep-trancl*:
assumes $\text{gctxtex-onp } P \mathcal{R} \subseteq \mathcal{R}$
shows $\text{gctxtex-onp } P (\mathcal{R}^+) \subseteq \mathcal{R}^+$
<proof>

lemma *gctxtex-onp-substep-rtrancl*:
assumes $\text{gctxtex-onp } P \mathcal{R} \subseteq \mathcal{R}$
shows $\text{gctxtex-onp } P (\mathcal{R}^*) \subseteq \mathcal{R}^*$
<proof>

lemma *gctxtex-onp-substep-trancl-diff-pred* [*intro*]:
assumes $\bigwedge C D. P C \implies Q D \implies Q (D \circ_{Gc} C)$
shows $\text{gctxtex-onp } Q ((\text{gctxtex-onp } P \mathcal{R})^+) \subseteq (\text{gctxtex-onp } Q \mathcal{R})^+$
<proof>

lemma *gctxtcl-pres-trancl*:
assumes $(s, t) \in \mathcal{R}^+$ **and** $\text{gctxtex-onp } P \mathcal{R} \subseteq \mathcal{R}$ **and** $P C$
shows $(C \langle s \rangle_G, C \langle t \rangle_G) \in \mathcal{R}^+$
<proof>

lemma *gctxtcl-pres-rtrancl*:
assumes $(s, t) \in \mathcal{R}^*$ **and** $\text{gctxtex-onp } P \mathcal{R} \subseteq \mathcal{R}$ **and** $P C$
shows $(C \langle s \rangle_G, C \langle t \rangle_G) \in \mathcal{R}^*$
<proof>

lemma *gmctxtex-onp-substep-trancl*:
assumes $\text{gmctxtex-onp } P \mathcal{R} \subseteq \mathcal{R}$
and *Id-on* $(\text{snd } \cdot \mathcal{R}) \subseteq \mathcal{R}$
shows $\text{gmctxtex-onp } P (\mathcal{R}^+) \subseteq \mathcal{R}^+$
<proof>

lemma *gmctxtex-onp-substep-tranclE*:
assumes *trans* \mathcal{R} **and** $\text{gmctxtex-onp } Q \mathcal{R} O \mathcal{R} \subseteq \mathcal{R}$ **and** $\mathcal{R} O \text{gmctxtex-onp } Q$

$\mathcal{R} \subseteq \mathcal{R}$
and $\bigwedge p C. P C \implies p \in \text{poss-gmctxt } C \implies Q (\text{subgm-at } C p)$
and $\bigwedge C D. P C \implies P D \implies (C, D) \in \text{comp-gmctxt} \implies P (C \sqcap D)$
shows $(\text{gmctxtex-onp } P \mathcal{R})^+ = \text{gmctxtex-onp } P \mathcal{R}$ (is ?Ls = ?Rs)
 ⟨proof⟩

7.7 Restr to set, union and predicate distribution

lemma *Restr-gctxtex-onp-dist* [simp]:

$\text{Restr } (\text{gctxtex-onp } P \mathcal{R}) (\mathcal{T}_G \mathcal{F}) =$
 $\text{gctxtex-onp } (\lambda C. \text{funas-gctxt } C \subseteq \mathcal{F} \wedge P C) (\text{Restr } \mathcal{R} (\mathcal{T}_G \mathcal{F}))$
 ⟨proof⟩

lemma *Restr-gmctxtex-onp-dist* [simp]:

$\text{Restr } (\text{gmctxtex-onp } P \mathcal{R}) (\mathcal{T}_G \mathcal{F}) =$
 $\text{gmctxtex-onp } (\lambda C. \text{funas-gmctxt } C \subseteq \mathcal{F} \wedge P C) (\text{Restr } \mathcal{R} (\mathcal{T}_G \mathcal{F}))$
 ⟨proof⟩

lemma *Restr-id-subset-gmctxtex-onp* [intro]:

assumes $\bigwedge C. \text{num-gholes } C = 0 \wedge \text{funas-gmctxt } C \subseteq \mathcal{F} \implies P C$
shows $\text{Restr Id } (\mathcal{T}_G \mathcal{F}) \subseteq \text{gmctxtex-onp } P \mathcal{R}$
 ⟨proof⟩

lemma *Restr-id-subset-gmctxtex-onp2* [intro]:

assumes $\bigwedge f n. (f, n) \in \mathcal{F} \implies P (\text{GMFun } f (\text{replicate } n \text{ GMHole}))$
and $\bigwedge C Ds. \text{num-gholes } C = \text{length } Ds \implies P C \implies \forall D \in \text{set } Ds. P D \implies$
 $P (\text{fill-gholes-gmctxt } C Ds)$
shows $\text{Restr Id } (\mathcal{T}_G \mathcal{F}) \subseteq \text{gmctxtex-onp } P \mathcal{R}$
 ⟨proof⟩

lemma *gctxtex-onp-union* [simp]:

$\text{gctxtex-onp } P (\mathcal{R} \cup \mathcal{L}) = \text{gctxtex-onp } P \mathcal{R} \cup \text{gctxtex-onp } P \mathcal{L}$
 ⟨proof⟩

lemma *gctxtex-onp-pred-dist*:

assumes $\bigwedge C. P C \longleftrightarrow Q C \vee R C$
shows $\text{gctxtex-onp } P \mathcal{R} = \text{gctxtex-onp } Q \mathcal{R} \cup \text{gctxtex-onp } R \mathcal{R}$
 ⟨proof⟩

lemma *gmctxtex-onp-pred-dist*:

assumes $\bigwedge C. P C \longleftrightarrow Q C \vee R C$
shows $\text{gmctxtex-onp } P \mathcal{R} = \text{gmctxtex-onp } Q \mathcal{R} \cup \text{gmctxtex-onp } R \mathcal{R}$
 ⟨proof⟩

lemma *trivial-gctxtex-onp* [simp]: $\text{gctxtex-onp } (\lambda C. C = \square_G) \mathcal{R} = \mathcal{R}$
 ⟨proof⟩

lemma *trivial-gmctxtex-onp* [simp]: $gmctxtex\text{-onp } (\lambda C. C = GMHole) \mathcal{R} = \mathcal{R}$
 ⟨proof⟩

7.8 Distribution of context closures over relation composition

lemma *gctxtex-onp-relcomp-inner*:
 $gctxtex\text{-onp } P (\mathcal{R} \circ \mathcal{L}) \subseteq gctxtex\text{-onp } P \mathcal{R} \circ gctxtex\text{-onp } P \mathcal{L}$
 ⟨proof⟩

lemma *gmctxtex-onp-relcomp-inner*:
 $gmctxtex\text{-onp } P (\mathcal{R} \circ \mathcal{L}) \subseteq gmctxtex\text{-onp } P \mathcal{R} \circ gmctxtex\text{-onp } P \mathcal{L}$
 ⟨proof⟩

7.9 Signature preserving and signature closed

definition *function-closed where*
 $function\text{-closed } \mathcal{F} \mathcal{R} \iff (\forall f\ ss\ ts. (f, length\ ts) \in \mathcal{F} \implies 0 \neq length\ ts \implies$
 $length\ ss = length\ ts \implies (\forall i. i < length\ ts \implies (ss\ !\ i, ts\ !\ i) \in \mathcal{R}) \implies$
 $(GFun\ f\ ss, GFun\ f\ ts) \in \mathcal{R})$

lemma *function-closedD*: $function\text{-closed } \mathcal{F} \mathcal{R} \implies$
 $(f, length\ ts) \in \mathcal{F} \implies 0 \neq length\ ts \implies length\ ss = length\ ts \implies$
 $\llbracket \bigwedge i. i < length\ ts \implies (ss\ !\ i, ts\ !\ i) \in \mathcal{R} \rrbracket \implies$
 $(GFun\ f\ ss, GFun\ f\ ts) \in \mathcal{R}$
 ⟨proof⟩

lemma *all-ctxt-closed-imp-function-closed*:
 $all\text{-ctxt-closed } \mathcal{F} \mathcal{R} \implies function\text{-closed } \mathcal{F} \mathcal{R}$
 ⟨proof⟩

lemma *all-ctxt-closed-imp-refl-on-sig*:
assumes $all\text{-ctxt-closed } \mathcal{F} \mathcal{R}$
shows $Restr\ Id\ (\mathcal{T}_G\ \mathcal{F}) \subseteq \mathcal{R}$
 ⟨proof⟩

lemma *function-closed-un-id-all-ctxt-closed*:
 $function\text{-closed } \mathcal{F} \mathcal{R} \implies Restr\ Id\ (\mathcal{T}_G\ \mathcal{F}) \subseteq \mathcal{R} \implies all\text{-ctxt-closed } \mathcal{F} \mathcal{R}$
 ⟨proof⟩

lemma *gctxtex-onp-in-signature* [intro]:
assumes $\bigwedge C. P\ C \implies funas\text{-gctxt } C \subseteq \mathcal{F} \bigwedge C. P\ C \implies funas\text{-gctxt } C \subseteq \mathcal{G}$
and $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$
shows $gctxtex\text{-onp } P \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$ ⟨proof⟩

lemma *gmctxtex-onp-in-signature* [intro]:
assumes $\bigwedge C. P\ C \implies funas\text{-gmctxt } C \subseteq \mathcal{F} \bigwedge C. P\ C \implies funas\text{-gmctxt } C \subseteq \mathcal{G}$
and $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$

shows $gmctxtex-onp\ P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$ *<proof>*

lemma *gctxtex-onp-in-signature-tranc* [intro]:
 $gctxtex-onp\ P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \implies (gctxtex-onp\ P\ \mathcal{R})^+ \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
<proof>

lemma *gmctxtex-onp-in-signature-tranc* [intro]:
 $gmctxtex-onp\ P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \implies (gmctxtex-onp\ P\ \mathcal{R})^+ \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
<proof>

lemma *gmctxtex-onp-fun-closed* [intro!]:
assumes $\bigwedge f\ n. (f, n) \in \mathcal{F} \implies n \neq 0 \implies P\ (GMFun\ f\ (replicate\ n\ GMHole))$
and $\bigwedge C\ Ds. P\ C \implies num-gholes\ C = length\ Ds \implies 0 < num-gholes\ C \implies$
 $\forall D \in set\ Ds. P\ D \implies P\ (fill-gholes-gmctxt\ C\ Ds)$
shows *function-closed* $\mathcal{F}\ (gmctxtex-onp\ P\ \mathcal{R})$ *<proof>*

declare *subsetI*[rule del]
lemma *gmctxtex-onp-sig-closed* [intro]:
assumes $\bigwedge f\ n. (f, n) \in \mathcal{F} \implies P\ (GMFun\ f\ (replicate\ n\ GMHole))$
and $\bigwedge C\ Ds. num-gholes\ C = length\ Ds \implies P\ C \implies \forall D \in set\ Ds. P\ D \implies$
 $P\ (fill-gholes-gmctxt\ C\ Ds)$
shows *all-ctxt-closed* $\mathcal{F}\ (gmctxtex-onp\ P\ \mathcal{R})$ *<proof>*
declare *subsetI*[intro!]

lemma *gmctxt-cl-gmctxtex-onp-conv*:
 $gmctxt-cl\ \mathcal{F}\ \mathcal{R} = gmctxtex-onp\ (\lambda C. funas-gmctxt\ C \subseteq \mathcal{F})\ \mathcal{R}$ (**is** $?Ls = ?Rs$)
<proof>

end
theory *FOR-Certificate*
imports *Rewriting*
begin

8 Certificate syntax and type declarations

type-alias $fvar = nat$ — variable id
datatype $ftrs = Fwd\ nat \mid Bwd\ nat$ — TRS id and direction

definition *map-ftrs* **where**
 $map-ftrs\ f = case-ftrs\ (Fwd \circ f)\ (Bwd \circ f)$

8.1 GTT relations

datatype $'trs\ gtt-rel$ — GTT relations
 $= ARoot\ 'trs\ list$ — root steps
 $\mid GInv\ 'trs\ gtt-rel$ — inverse of anchored or ordinary GTT relation
 $\mid AUnion\ 'trs\ gtt-rel\ 'trs\ gtt-rel$ — union of anchored GTT relation
 $\mid ATrancl\ 'trs\ gtt-rel$ — 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 \rightarrow
 $R2Step$ $trss = R2GTT$ -Rel ($ARoot$ $trss$) $PAny$ $ESingle$

definition $R2StepEq$ **where** — at most one step $\rightarrow^=$

$R2StepEq\ trss = R2Refc\ (R2Step\ trss)$
definition $R2Steps$ **where** — at least one step \rightarrow^+
 $R2Steps\ trss = R2GTT-Rel\ (GSteps\ trss)\ PAny\ EStrictParallel$
definition $R2StepsEq$ **where** — many steps \rightarrow^*
 $R2StepsEq\ trss = R2GTT-Rel\ (GSteps\ trss)\ PAny\ EParallel$
definition $R2StepsNF$ **where** — rewrite to normal form $\rightarrow^!$
 $R2StepsNF\ trss = R2Inter\ (R2StepsEq\ trss)\ (R2Prod\ R1Terms\ (R1NF\ trss))$
definition $R2ParStep$ **where** — parallel step
 $R2ParStep\ trss = R2GTT-Rel\ (ARoot\ trss)\ PAny\ EParallel$
definition $R2RootStep$ **where** — root step \rightarrow_ϵ
 $R2RootStep\ trss = R2GTT-Rel\ (ARoot\ trss)\ PRoot\ ESingle$
definition $R2RootStepEq$ **where** — at most one root step $\rightarrow_\epsilon^=$
 $R2RootStepEq\ trss = R2Refc\ (R2RootStep\ trss)$

definition $R2RootSteps$ **where** — at least one root step \rightarrow_ϵ^+
 $R2RootSteps\ trss = R2GTT-Rel\ (ATrancl\ (ARoot\ trss))\ PRoot\ ESingle$
definition $R2RootStepsEq$ **where** — many root steps \rightarrow_ϵ^*
 $R2RootStepsEq\ trss = R2Refc\ (R2RootSteps\ trss)$
definition $R2NonRootStep$ **where** — non-root step $\rightarrow_{>\epsilon}$
 $R2NonRootStep\ trss = R2GTT-Rel\ (ARoot\ trss)\ PNonRoot\ ESingle$
definition $R2NonRootStepEq$ **where** — at most one non-root step $\rightarrow_{>\epsilon}^=$
 $R2NonRootStepEq\ trss = R2Refc\ (R2NonRootStep\ trss)$
definition $R2NonRootSteps$ **where** — at least one non-root step $\rightarrow_{>\epsilon}^+$
 $R2NonRootSteps\ trss = R2GTT-Rel\ (GSteps\ trss)\ PNonRoot\ EStrictParallel$
definition $R2NonRootStepsEq$ **where** — many non-root steps $\rightarrow_{>\epsilon}^*$
 $R2NonRootStepsEq\ trss = R2GTT-Rel\ (GSteps\ trss)\ PNonRoot\ EParallel$
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

$FTrue \equiv FAnd\ []$

definition $FFalse$ **where** — false

$FFalse \equiv FOr\ []$

definition *FRestrict where* — reorder/rename/restrict TRSs for subformula
FRestrict f $trss \equiv \text{map-formula } (\text{map-ftrs } (\lambda n. \text{if } n \geq \text{length } trss \text{ then } 0 \text{ else } trss$
 $! n)) f$

8.4 Signatures and Problems

datatype $(f, 'v, 't)$ *many-sorted-sig*
 $= \text{Many-Sorted-Sig } (ms\text{-functions: } ('f \times 't \text{ list} \times 't) \text{ list}) (ms\text{-variables: } ('v \times 't)$
 $\text{list})$

datatype $(f, 'v, 't)$ *problem*
 $= \text{Problem } (p\text{-signature: } ('f, 'v, 't) \text{ many-sorted-sig})$
 $(p\text{-trss: } ('f, 'v) \text{ trs list})$
 $(p\text{-formula: } ftrs \text{ formula})$

8.5 Proofs

datatype *equivalence* — formula equivalences
 $= \text{EDistribAndOr}$ — distributivity: conjunction over disjunction
 $| \text{EDistribOrAnd}$ — distributivity: disjunction over conjunction

datatype $'trs$ *inference* — inference rules for formula creation
 $= \text{IRR1 } 'trs \text{ rr1-rel } fvar$ — formula from RR1 relation
 $| \text{IRR2 } 'trs \text{ rr2-rel } fvar \text{ fvar}$ — formula from RR2 relation
 $| \text{IAnd } nat \text{ list}$ — conjunction
 $| \text{IOr } nat \text{ list}$ — disjunction
 $| \text{INot } nat$ — negation
 $| \text{IExists } nat$ — existential quantification
 $| \text{IRename } nat \text{ fvar list}$ — permute variables
 $| \text{INNFPlus } nat$ — equivalence modulo negation normal form plus
ACIU0 for \wedge and \vee
 $| \text{IRepl equivalence } nat \text{ list } nat$ — replacement according to given equivalence

datatype *claim* $= \text{Empty} | \text{Nonempty}$

datatype *info* $= \text{Size } nat \text{ nat } nat$

datatype $'trs$ *certificate*
 $= \text{Certificate } (nat \times 'trs \text{ inference} \times 'trs \text{ formula} \times \text{info list}) \text{ list claim } nat$

8.6 Example

definition *no-normal-forms-cert :: ftrs certificate where*
 $\text{no-normal-forms-cert} = \text{Certificate}$
 $[(0, (\text{IRR2 } (\text{R2Step } [\text{Fwd } 0]) 1 0),$
 $(\text{FRR2 } (\text{R2Step } [\text{Fwd } 0]) 1 0), [])$
 $, (1, (\text{IExists } 0),$
 $(\text{FExists } (\text{FRR2 } (\text{R2Step } [\text{Fwd } 0]) 1 0)), [])$
 $, (2, (\text{INot } 1),$

```

      (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, (INNPlus 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* $\mathcal{R} \equiv \text{gctxtex-onp } (\lambda C. \text{True}) \mathcal{R}$

abbreviation *gmctxtcl* $\mathcal{R} \equiv \text{gmctxtex-onp } (\lambda C. \text{True}) \mathcal{R}$

Extension under all non empty contexts

abbreviation *gctxtex-nempty* $\mathcal{R} \equiv \text{gctxtex-onp } (\lambda C. C \neq \square_G) \mathcal{R}$

abbreviation *gmctxtex-nempty* $\mathcal{R} \equiv \text{gmctxtex-onp } (\lambda C. C \neq \text{GMHole}) \mathcal{R}$

Closure under all contexts respecting the signature

abbreviation *gctxtcl-fun* $\mathcal{F} \mathcal{R} \equiv \text{gctxtex-onp } (\lambda C. \text{funas-gctxt } C \subseteq \mathcal{F}) \mathcal{R}$

abbreviation *gmctxtcl-fun* $\mathcal{F} \mathcal{R} \equiv \text{gmctxtex-onp } (\lambda C. \text{funas-gmctxt } C \subseteq \mathcal{F}) \mathcal{R}$

Closure under all multihole contexts with at least one hole respecting the signature

abbreviation *gmctxtcl-fun* $\mathcal{F} \mathcal{R} \equiv \text{gmctxtex-onp } (\lambda C. 0 < \text{num-gholes } C \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}) \mathcal{R}$

Extension under all non empty contexts respecting the signature

abbreviation *gctxtex-fun* $\mathcal{F} \mathcal{R} \equiv \text{gctxtex-onp } (\lambda C. \text{funas-gctxt } C \subseteq \mathcal{F} \wedge C \neq \square_G) \mathcal{R}$

abbreviation $gmctxtex\text{-funas}\text{-nroot } \mathcal{F} \mathcal{R} \equiv gmctxtex\text{-onp } (\lambda C. \text{funas}\text{-gmctxt } C \subseteq \mathcal{F} \wedge C \neq GMHole) \mathcal{R}$

Extension under all non empty contexts respecting the signature

abbreviation $gmctxtex\text{-funas}\text{-nroot}\text{-strict } \mathcal{F} \mathcal{R} \equiv gmctxtex\text{-onp } (\lambda C. 0 < \text{num}\text{-gholes } C \wedge \text{funas}\text{-gmctxt } C \subseteq \mathcal{F} \wedge C \neq GMHole) \mathcal{R}$

9.1 Rewrite steps equivalent definitions

definition $gsubst\text{-cl} :: ('f, 'v) \text{trs} \Rightarrow 'f \text{gterm rel}$ **where**
 $gsubst\text{-cl } \mathcal{R} = \{(gterm\text{-of}\text{-term } (l \cdot \sigma), gterm\text{-of}\text{-term } (r \cdot \sigma)) \mid$
 $l r (\sigma :: 'v \Rightarrow ('f, 'v) \text{Term.term}). (l, r) \in \mathcal{R} \wedge \text{ground } (l \cdot \sigma) \wedge \text{ground } (r \cdot \sigma)\}$

definition $gnrrstepD :: 'f \text{sig} \Rightarrow 'f \text{gterm rel} \Rightarrow 'f \text{gterm rel}$ **where**
 $gnrrstepD \mathcal{F} \mathcal{R} = gctxtex\text{-funas}\text{-nroot } \mathcal{F} \mathcal{R}$

definition $grstepD :: 'f \text{sig} \Rightarrow 'f \text{gterm rel} \Rightarrow 'f \text{gterm rel}$ **where**
 $grstepD \mathcal{F} \mathcal{R} = gctxtcl\text{-funas } \mathcal{F} \mathcal{R}$

definition $gpar\text{-rstepD} :: 'f \text{sig} \Rightarrow 'f \text{gterm rel} \Rightarrow 'f \text{gterm rel}$ **where**
 $gpar\text{-rstepD } \mathcal{F} \mathcal{R} = gmctxtcl\text{-funas } \mathcal{F} \mathcal{R}$

inductive-set $gpar\text{-rstepD}' :: 'f \text{sig} \Rightarrow 'f \text{gterm rel} \Rightarrow 'f \text{gterm rel}$ **for** $\mathcal{F} :: 'f \text{sig}$
and $\mathcal{R} :: 'f \text{gterm rel}$
where $groot\text{-step [intro]: (s, t) \in \mathcal{R} \Longrightarrow (s, t) \in gpar\text{-rstepD}' \mathcal{F} \mathcal{R}$
 $| gpar\text{-step}\text{-fun [intro]: \llbracket \bigwedge i. i < \text{length } ts \Longrightarrow (ss ! i, ts ! i) \in gpar\text{-rstepD}' \mathcal{F} \mathcal{R} \rrbracket \Longrightarrow \text{length } ss = \text{length } ts$
 $\Longrightarrow (f, \text{length } ts) \in \mathcal{F} \Longrightarrow (GFun f ss, GFun f ts) \in gpar\text{-rstepD}' \mathcal{F} \mathcal{R}$

9.2 Interface between rewrite step definitions and sets

fun $lift\text{-root}\text{-step} :: ('f \times \text{nat}) \text{set} \Rightarrow \text{pos}\text{-step} \Rightarrow \text{ext}\text{-step} \Rightarrow 'f \text{gterm rel} \Rightarrow 'f \text{gterm rel}$ **where**

$lift\text{-root}\text{-step } \mathcal{F} PAny ESingle \mathcal{R} = gctxtcl\text{-funas } \mathcal{F} \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PAny EStrictParallel \mathcal{R} = gmctxtcl\text{-funas}\text{-strict } \mathcal{F} \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PAny EParallel \mathcal{R} = gmctxtcl\text{-funas } \mathcal{F} \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PNonRoot ESingle \mathcal{R} = gctxtex\text{-funas}\text{-nroot } \mathcal{F} \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PNonRoot EStrictParallel \mathcal{R} = gmctxtex\text{-funas}\text{-nroot}\text{-strict } \mathcal{F} \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PNonRoot EParallel \mathcal{R} = gmctxtex\text{-funas}\text{-nroot } \mathcal{F} \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PRoot ESingle \mathcal{R} = \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PRoot EStrictParallel \mathcal{R} = \mathcal{R}$
 $| lift\text{-root}\text{-step } \mathcal{F} PRoot EParallel \mathcal{R} = \mathcal{R} \cup \text{Restr Id } (\mathcal{T}_G \mathcal{F})$

9.3 Compatibility of used predicate extensions and signature closure

lemma $compatible\text{-p [simp]:$
 $compatible\text{-p } (\lambda C. C \neq \square_G) (\lambda C. C \neq GMHole)$
 $compatible\text{-p } (\lambda C. \text{funas}\text{-gctxt } C \subseteq \mathcal{F}) (\lambda C. \text{funas}\text{-gmctxt } C \subseteq \mathcal{F})$

compatible-p $(\lambda C. \text{funas-gctxt } C \subseteq \mathcal{F} \wedge C \neq \square_G) (\lambda C. \text{funas-gmctxt } C \subseteq \mathcal{F} \wedge C \neq \text{GMHole})$
 ⟨proof⟩

lemma *gmctxtcl-funas-sigcl*:
all-ctxt-closed $\mathcal{F} (\text{gmctxtcl-funas } \mathcal{F} \mathcal{R})$
 ⟨proof⟩

lemma *gctxtex-funas-nroot-sigcl*:
all-ctxt-closed $\mathcal{F} (\text{gmctxtex-funas-nroot } \mathcal{F} \mathcal{R})$
 ⟨proof⟩

lemma *gmctxtcl-funas-strict-funcl*:
function-closed $\mathcal{F} (\text{gmctxtcl-funas-strict } \mathcal{F} \mathcal{R})$
 ⟨proof⟩

lemma *gmctxtex-funas-nroot-strict-funcl*:
function-closed $\mathcal{F} (\text{gmctxtex-funas-nroot-strict } \mathcal{F} \mathcal{R})$
 ⟨proof⟩

lemma *gctxtcl-funas-dist*:
gctxtcl-funas $\mathcal{F} \mathcal{R} = \text{gctxtex-onp } (\lambda C. C = \square_G) \mathcal{R} \cup \text{gctxtex-funas-nroot } \mathcal{F} \mathcal{R}$
 ⟨proof⟩

lemma *gmctxtex-funas-nroot-dist*:
gmctxtex-funas-nroot $\mathcal{F} \mathcal{R} = \text{gmctxtex-funas-nroot-strict } \mathcal{F} \mathcal{R} \cup$
gmctxtex-onp $(\lambda C. \text{num-gholes } C = 0 \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}) \mathcal{R}$
 ⟨proof⟩

lemma *gmctxtcl-funas-dist*:
gmctxtcl-funas $\mathcal{F} \mathcal{R} = \text{gmctxtex-onp } (\lambda C. \text{num-gholes } C = 0 \wedge \text{funas-gmctxt } C$
 $\subseteq \mathcal{F}) \mathcal{R} \cup$
gmctxtex-onp $(\lambda C. 0 < \text{num-gholes } C \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}) \mathcal{R}$
 ⟨proof⟩

lemma *gmctxtcl-funas-strict-dist*:
gmctxtcl-funas-strict $\mathcal{F} \mathcal{R} = \text{gmctxtex-funas-nroot-strict } \mathcal{F} \mathcal{R} \cup \text{gmctxtex-onp } (\lambda$
 $C. C = \text{GMHole}) \mathcal{R}$
 ⟨proof⟩

lemma *gmctxtex-onpzero-num-gholes-id* [*simp*]:
gmctxtex-onp $(\lambda C. \text{num-gholes } C = 0 \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}) \mathcal{R} = \text{Restr Id}$
 $(\mathcal{T}_G \mathcal{F})$ (**is** $?Ls = ?Rs$)
 ⟨proof⟩

lemma *gctxtex-onp-sign-trans-fst*:
assumes $(s, t) \in \text{gctxtex-onp } P R$ **and** $s \in \mathcal{T}_G \mathcal{F}$
shows $(s, t) \in \text{gctxtex-onp } (\lambda C. \text{funas-gctxt } C \subseteq \mathcal{F} \wedge P C) R$
 ⟨proof⟩

lemma *gctxtex-onp-sign-trans-snd*:

assumes $(s, t) \in \text{gctxtex-onp } P R$ **and** $t \in \mathcal{T}_G \mathcal{F}$
shows $(s, t) \in \text{gctxtex-onp } (\lambda C. \text{funas-gctxt } C \subseteq \mathcal{F} \wedge P C) R$
 $\langle \text{proof} \rangle$

lemma *gmctxtex-onp-sign-trans-fst*:

assumes $(s, t) \in \text{gmctxtex-onp } P R$ **and** $s \in \mathcal{T}_G \mathcal{F}$
shows $(s, t) \in \text{gmctxtex-onp } (\lambda C. P C \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}) R$
 $\langle \text{proof} \rangle$

lemma *gmctxtex-onp-sign-trans-snd*:

assumes $(s, t) \in \text{gmctxtex-onp } P R$ **and** $t \in \mathcal{T}_G \mathcal{F}$
shows $(s, t) \in \text{gmctxtex-onp } (\lambda C. P C \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}) R$
 $\langle \text{proof} \rangle$

9.4 Basic lemmas

lemma *gsubst-cl*:

fixes $\mathcal{R} :: ('f, 'v) \text{ trs}$ **and** $\sigma :: 'v \Rightarrow ('f, 'v) \text{ term}$
assumes $(l, r) \in \mathcal{R}$ **and** $\text{ground } (l \cdot \sigma) \text{ ground } (r \cdot \sigma)$
shows $(\text{gterm-of-term } (l \cdot \sigma), \text{gterm-of-term } (r \cdot \sigma)) \in \text{gsubst-cl } \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma *grstepD [simp]*:

$(s, t) \in \mathcal{R} \implies (s, t) \in \text{grstepD } \mathcal{F} \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma *grstepD-ctxtI [intro]*:

$(l, r) \in \mathcal{R} \implies \text{funas-gctxt } C \subseteq \mathcal{F} \implies (C\langle l \rangle_G, C\langle r \rangle_G) \in \text{grstepD } \mathcal{F} \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma *gctxtex-fun-as-nroot-gctxtcl-fun-as-subseteq*:

$\text{gctxtex-fun-as-nroot } \mathcal{F} (\text{grstepD } \mathcal{F} \mathcal{R}) \subseteq \text{grstepD } \mathcal{F} \mathcal{R}$
 $\langle \text{proof} \rangle$

lemma *Restr-gnrrstepD-dist [simp]*:

$\text{Restr } (\text{gnrrstepD } \mathcal{F} \mathcal{R}) (\mathcal{T}_G \mathcal{G}) = \text{gnrrstepD } (\mathcal{F} \cap \mathcal{G}) (\text{Restr } \mathcal{R} (\mathcal{T}_G \mathcal{G}))$
 $\langle \text{proof} \rangle$

lemma *Restr-grstepD-dist [simp]*:

$\text{Restr } (\text{grstepD } \mathcal{F} \mathcal{R}) (\mathcal{T}_G \mathcal{G}) = \text{grstepD } (\mathcal{F} \cap \mathcal{G}) (\text{Restr } \mathcal{R} (\mathcal{T}_G \mathcal{G}))$
 $\langle \text{proof} \rangle$

lemma *Restr-gpar-rstepD-dist [simp]*:

$\text{Restr } (\text{gpar-rstepD } \mathcal{F} \mathcal{R}) (\mathcal{T}_G \mathcal{G}) = \text{gpar-rstepD } (\mathcal{F} \cap \mathcal{G}) (\text{Restr } \mathcal{R} (\mathcal{T}_G \mathcal{G}))$ (is
 $?Ls = ?Rs$)
 $\langle \text{proof} \rangle$

9.5 Equivalence lemmas

lemma *grrstep-subst-cl-conv*:

$$grrstep \mathcal{R} = gsubst-cl \mathcal{R}$$

<proof>

lemma *gnrrstepD-gnrrstep-conv*:

$$gnrrstep \mathcal{R} = gnrrstepD \text{ UNIV } (gsubst-cl \mathcal{R}) \text{ (is ?Ls = ?Rs)}$$

<proof>

lemma *grstepD-grstep-conv*:

$$grstep \mathcal{R} = grstepD \text{ UNIV } (gsubst-cl \mathcal{R}) \text{ (is ?Ls = ?Rs)}$$

<proof>

lemma *gpar-rstep-gpar-rstepD-conv*:

$$gpar-rstep \mathcal{R} = gpar-rstepD' \text{ UNIV } (gsubst-cl \mathcal{R}) \text{ (is ?Ls = ?Rs)}$$

<proof>

lemma *gmctxtcl-fun-as-idem*:

$$gmctxtcl-fun-as \mathcal{F} (gmctxtcl-fun-as \mathcal{F} \mathcal{R}) \subseteq gmctxtcl-fun-as \mathcal{F} \mathcal{R}$$

<proof>

lemma *gpar-rstepD-gpar-rstepD'-conv*:

$$gpar-rstepD \mathcal{F} \mathcal{R} = gpar-rstepD' \mathcal{F} \mathcal{R} \text{ (is ?Ls = ?Rs)}$$

<proof>

9.6 Signature preserving lemmas

lemma *T_G-trans-closure-id [simp]*:

$$(\mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F})^+ = \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$$

<proof>

lemma *signature-pres-fun-as-cl [simp]*:

$$\begin{aligned} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} &\implies gtxtcl-fun-as \mathcal{F} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} \\ \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} &\implies gmctxtcl-fun-as \mathcal{F} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} \end{aligned}$$

<proof>

lemma *refl-on-gmctxtcl-fun-as*:

assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$
shows *refl-on* $(\mathcal{T}_G \mathcal{F}) (gmctxtcl-fun-as \mathcal{F} \mathcal{R})$

<proof>

lemma *gtrancl-rel-sound*:

$$\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} \implies gtrancl-rel \mathcal{F} \mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$$

<proof>

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⟩

lemma *gmctxtcl-funas-in-rtrancl-gctxtcl-funas*:
 assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$
 shows *gmctxtcl-funas* \mathcal{F} $\mathcal{R} \subseteq (\text{gctxtcl-funas } \mathcal{F} \mathcal{R})^*$ ⟨proof⟩

lemma *R-in-gtrancl-rel*:
 assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$
 shows $\mathcal{R} \subseteq \text{gtrancl-rel } \mathcal{F} \mathcal{R}$
 ⟨proof⟩

lemma *trans-gtrancl-rel [simp]*:
 trans (*gtrancl-rel* \mathcal{F} \mathcal{R})
 ⟨proof⟩

lemma *gtrancl-rel-cl*:
 assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$
 shows *gmctxtcl-funas* \mathcal{F} (*gtrancl-rel* $\mathcal{F} \mathcal{R}$) $\subseteq (\text{gmctxtcl-funas } \mathcal{F} \mathcal{R})^+$
 ⟨proof⟩

lemma *gtrancl-rel-aux*:
 $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} \implies \text{gmctxtcl-funas } \mathcal{F} (\text{gtrancl-rel } \mathcal{F} \mathcal{R}) \text{ O } \text{gtrancl-rel } \mathcal{F} \mathcal{R}$
 $\subseteq \text{gtrancl-rel } \mathcal{F} \mathcal{R}$
 $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F} \implies \text{gtrancl-rel } \mathcal{F} \mathcal{R} \text{ O } \text{gmctxtcl-funas } \mathcal{F} (\text{gtrancl-rel } \mathcal{F} \mathcal{R})$
 $\subseteq \text{gtrancl-rel } \mathcal{F} \mathcal{R}$
 ⟨proof⟩

declare *subsetI* [rule del]

lemma *gtrancl-rel*:
 assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ *compatible-p* $Q P$
 and $\bigwedge C. P C \implies \text{funas-gmctxt } C \subseteq \mathcal{F}$
 and $\bigwedge C D. P C \implies P D \implies (C, D) \in \text{comp-gmctxt} \implies P (C \sqcap D)$
 shows (*gctxtex-onp* $Q \mathcal{R}$)⁺ $\subseteq \text{gmctxtex-onp } P (\text{gtrancl-rel } \mathcal{F} \mathcal{R})$
 ⟨proof⟩

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* $\mathcal{F} \mathcal{R} \subseteq (\text{gctxtcl-funas } \mathcal{F} \mathcal{R})^+$
 ⟨proof⟩

lemma *gmctxtex-onp-gtrancl-rel*:
 assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ and $\bigwedge C D. Q C \implies \text{funas-gctxt } D \subseteq \mathcal{F} \implies Q$
 ($C \circ_{Gc} D$)
 and $\bigwedge C. P C \implies 0 < \text{num-gholes } C \wedge \text{funas-gmctxt } C \subseteq \mathcal{F}$
 and $\bigwedge C. P C \implies \text{gmctxt-p-inv } C \mathcal{F} Q$
 shows *gmctxtex-onp* $P (\text{gtrancl-rel } \mathcal{F} \mathcal{R}) \subseteq (\text{gctxtex-onp } Q \mathcal{R})^+$
 ⟨proof⟩

lemma *gmctxtcl-funas-strict-gtrancl-rel:*

assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$

shows $\text{gmctxtcl-funas-strict } \mathcal{F} (\text{gtrancl-rel } \mathcal{F} \mathcal{R}) = (\text{gctxtcl-funas } \mathcal{F} \mathcal{R})^+ (\text{is } ?Ls = ?Rs)$

<proof>

lemma *gmctxtex-funas-nroot-strict-gtrancl-rel:*

assumes $\mathcal{R} \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$

shows $\text{gmctxtex-funas-nroot-strict } \mathcal{F} (\text{gtrancl-rel } \mathcal{F} \mathcal{R}) = (\text{gctxtex-funas-nroot } \mathcal{F} \mathcal{R})^+$

(is ?Ls = ?Rs)

<proof>

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 $\text{lift-root-step } \mathcal{F} W X \mathcal{R} \subseteq \mathcal{T}_G \mathcal{G} \times \mathcal{T}_G \mathcal{H}$

<proof>

lemmas $\text{lift-root-step-sig} = \text{lift-root-step-sig}'[\text{OF} - \text{subset-refl subset-refl}]$

lemma *lift-root-step-incr:*

$\mathcal{R} \subseteq \mathcal{S} \implies \text{lift-root-step } \mathcal{F} W X \mathcal{R} \subseteq \text{lift-root-step } \mathcal{F} W X \mathcal{S}$

<proof>

lemma *Restr-id-mono:*

$\mathcal{F} \subseteq \mathcal{G} \implies \text{Restr Id } (\mathcal{T}_G \mathcal{F}) \subseteq \text{Restr Id } (\mathcal{T}_G \mathcal{G})$

<proof>

lemma *lift-root-step-mono:*

$\mathcal{F} \subseteq \mathcal{G} \implies \text{lift-root-step } \mathcal{F} W X \mathcal{R} \subseteq \text{lift-root-step } \mathcal{G} W X \mathcal{R}$

<proof>

lemma *grstep-lift-root-step:*

$\text{lift-root-step } \mathcal{F} PAny ESingle (\text{Restr } (\text{grstep } \mathcal{R}) (\mathcal{T}_G \mathcal{F})) = \text{Restr } (\text{grstep } \mathcal{R}) (\mathcal{T}_G \mathcal{F})$

<proof>

lemma *prod-swap-id-on-refl [simp]:*

$\text{Restr Id } (\mathcal{T}_G \mathcal{F}) \subseteq \text{prod.swap } ' (\mathcal{R} \cup \text{Restr Id } (\mathcal{T}_G \mathcal{F}))$

<proof>

lemma *swap-lift-root-step:*

$\text{lift-root-step } \mathcal{F} W X (\text{prod.swap } ' \mathcal{R}) = \text{prod.swap } ' \text{lift-root-step } \mathcal{F} W X \mathcal{R}$

<proof>

lemma *converse-lift-root-step:*

$(\text{lift-root-step } \mathcal{F} W X \mathcal{R})^{-1} = \text{lift-root-step } \mathcal{F} W X (\mathcal{R}^{-1})$

<proof>

lemma *lift-root-step-sig-transfer*:

assumes $p \in \text{lift-root-step } \mathcal{F} \ W \ X \ R \ \text{snd} \ ' \ R \subseteq \mathcal{T}_G \ \mathcal{F} \ \text{funas-gterm} \ (\text{fst } p) \subseteq \mathcal{G}$
shows $p \in \text{lift-root-step } \mathcal{G} \ W \ X \ R \ \langle \text{proof} \rangle$

lemma *lift-root-step-sig-transfer2*:

assumes $p \in \text{lift-root-step } \mathcal{F} \ W \ X \ R \ \text{snd} \ ' \ R \subseteq \mathcal{T}_G \ \mathcal{G} \ \text{funas-gterm} \ (\text{fst } p) \subseteq \mathcal{G}$
shows $p \in \text{lift-root-step } \mathcal{G} \ W \ X \ R$
<proof>

lemma *lift-root-steps-sig-transfer*:

assumes $(s, t) \in (\text{lift-root-step } \mathcal{F} \ W \ X \ R)^+ \ \text{snd} \ ' \ R \subseteq \mathcal{T}_G \ \mathcal{G} \ \text{funas-gterm} \ s \subseteq \mathcal{G}$
shows $(s, t) \in (\text{lift-root-step } \mathcal{G} \ W \ X \ R)^+$
<proof>

lemma *lift-root-stepseq-sig-transfer*:

assumes $(s, t) \in (\text{lift-root-step } \mathcal{F} \ W \ X \ R)^* \ \text{snd} \ ' \ R \subseteq \mathcal{T}_G \ \mathcal{G} \ \text{funas-gterm} \ s \subseteq \mathcal{G}$
shows $(s, t) \in (\text{lift-root-step } \mathcal{G} \ W \ X \ R)^*$
<proof>

lemmas *lift-root-step-sig-transfer' = lift-root-step-sig-transfer*[of *prod.swap* $p \ \mathcal{F} \ W \ X \ \text{prod.swap} \ ' \ R \ \mathcal{G}$ **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 $t \ s \ \mathcal{F} \ W \ X \ \text{prod.swap} \ ' \ R \ \mathcal{G}$ **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 $t \ s \ \mathcal{F} \ W \ X \ \text{prod.swap} \ ' \ R \ \mathcal{G}$ **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*]:

lift-root-step $\mathcal{F} \ \text{PRoot} \ \text{ESingle} \ \mathcal{R} = \mathcal{R}$
<proof>

lemma *lift-root-step-PRoot-EStrictParallel* [*simp*]:

lift-root-step $\mathcal{F} \ \text{PRoot} \ \text{EStrictParallel} \ \mathcal{R} = \mathcal{R}$
<proof>

lemma *lift-root-step-Parallel-conv*:

shows *lift-root-step* $\mathcal{F} \ W \ \text{EParallel} \ \mathcal{R} = \text{lift-root-step } \mathcal{F} \ W \ \text{EStrictParallel} \ \mathcal{R} \cup \text{Restr Id } (\mathcal{T}_G \ \mathcal{F})$

<proof>

lemma *relax-pos-lift-root-step*:

lift-root-step \mathcal{F} W X $R \subseteq$ *lift-root-step* \mathcal{F} $PAny$ X R

<proof>

lemma *relax-pos-lift-root-steps*:

$(\textit{lift-root-step } \mathcal{F} \ W \ X \ R)^+ \subseteq (\textit{lift-root-step } \mathcal{F} \ PAny \ X \ R)^+$

<proof>

lemma *relax-ext-lift-root-step*:

lift-root-step \mathcal{F} W X $R \subseteq$ *lift-root-step* \mathcal{F} W $EParallel$ R

<proof>

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} $PAny$ $EStrictParallel$ $R \subseteq (\textit{lift-root-step } \mathcal{F} \ PAny \ ESingle \ R)^+$

<proof>

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} $PAny$ $EParallel$ $R \subseteq (\textit{lift-root-step } \mathcal{F} \ PAny \ ESingle \ R)^+ \cup \textit{Restr Id } (\mathcal{T}_G \ \mathcal{F})$

<proof>

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

<proof>

lemma *trancl-partial-reflcl*:

$(X \cup \textit{Restr Id } Y)^+ = X^+ \cup \textit{Restr Id } Y$

<proof>

lemma *lift-root-step-Parallels-single*:

assumes $R \subseteq \mathcal{T}_G \ \mathcal{F} \times \mathcal{T}_G \ \mathcal{F}$

shows $(\textit{lift-root-step } \mathcal{F} \ PAny \ EParallel \ R)^+ = (\textit{lift-root-step } \mathcal{F} \ PAny \ ESingle \ R)^+ \cup \textit{Restr Id } (\mathcal{T}_G \ \mathcal{F})$

<proof>

lemma *lift-root-Any-Single-eq*:

shows *lift-root-step* \mathcal{F} $PAny$ $ESingle$ $R = R \cup \textit{lift-root-step } \mathcal{F} \ PNonRoot \ ESingle \ R$

<proof>

lemma *lift-root-Any-EStrict-eq [simp]*:

shows *lift-root-step* \mathcal{F} $PAny$ $EStrictParallel$ $R = R \cup \textit{lift-root-step } \mathcal{F} \ PNonRoot \ EStrictParallel \ R$

<proof>

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}) ($\mathcal{T}_G \mathcal{F}$)
<proof>

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}) ($\mathcal{T}_G \mathcal{F}$)
<proof>

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})*
<proof>

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* ($\lambda f . (Some f, Some f)$) C) $\langle u \rangle_G =$ *gpair* $C\langle s \rangle_G C\langle t \rangle_G$ *<proof>*

lemma *gpair-gctxt'*:

assumes *gpair* $C\langle v \rangle_G C\langle w \rangle_G = u$

shows $u =$ (*map-gctxt* ($\lambda f . (Some f, Some f)$) C) \langle *gpair* $v w \rangle_G$
<proof>

lemma *gpair-gmctxt*:

assumes $\forall i < \text{length } us. \text{gpair } (ss ! i) (ts ! i) = us ! i$

and $\text{num-gholes } C = \text{length } ss \text{ length } ss = \text{length } ts \text{ length } ts = \text{length } us$

shows *fill-gholes* (*map-gmctxt* ($\lambda f . (Some f, Some f)$) C) $us =$ *gpair* (*fill-gholes* $C ss$) (*fill-gholes* $C ts$)
<proof>

lemma *gctxtex-onp-gpair-set-conv*:

$\{ \text{gpair } t u \mid t u. (t, u) \in \text{gctxtex-onp } P \mathcal{R} \} =$

$\{(map-gctxt (\lambda f .(Some f, Some f)) C)\langle s \rangle_G \mid C s. P C \wedge s \in \{gpair\ t\ u \mid t\ u. (t, u) \in \mathcal{R}\}\}$ (is ?Ls = ?Rs)
 ⟨proof⟩

lemma *gmctxtex-onp-gpair-set-conv*:

$\{gpair\ t\ u \mid t\ u. (t, u) \in gmctxtex-onp\ P\ \mathcal{R}\} =$
 $\{fill-gholes\ (map-gmctxt\ (\lambda f .(Some\ f,\ Some\ f))\ C)\ ss \mid C\ ss.\ num-gholes\ C =$
 $length\ ss \wedge P\ C \wedge$
 $(\forall\ i < length\ ss.\ ss\ !\ i \in \{gpair\ t\ u \mid t\ u. (t, u) \in \mathcal{R}\})\}$ (is ?Ls = ?Rs)
 ⟨proof⟩

abbreviation *lift-sig-RR2* $\equiv \lambda (f, n). ((Some\ f,\ Some\ f), n)$

abbreviation *lift-fun* $\equiv (\lambda f. (Some\ f,\ Some\ f))$

abbreviation *unlift-fst* $\equiv (\lambda f. the\ (fst\ f))$

abbreviation *unlift-snd* $\equiv (\lambda f. the\ (snd\ f))$

lemma *RR2-gterm-unlift-lift-id* [simp]:

$funas-gterm\ t \subseteq lift-sig-RR2\ 'F \implies map-gterm\ (lift-fun \circ unlift-fst)\ t = t$
 ⟨proof⟩

lemma *RR2-gterm-unlift-funas* [simp]:

$funas-gterm\ t \subseteq lift-sig-RR2\ 'F \implies funas-gterm\ (map-gterm\ unlift-fst\ t) \subseteq \mathcal{F}$
 ⟨proof⟩

lemma *gterm-funas-lift-RR2-funas* [simp]:

$funas-gterm\ t \subseteq \mathcal{F} \implies funas-gterm\ (map-gterm\ lift-fun\ t) \subseteq lift-sig-RR2\ 'F$
 ⟨proof⟩

lemma *RR2-gctxt-unlift-lift-id* [simp, intro]:

$funas-gctxt\ C \subseteq lift-sig-RR2\ 'F \implies (map-gctxt\ (lift-fun \circ unlift-fst)\ C) = C$
 ⟨proof⟩

lemma *RR2-gctxt-unlift-funas* [simp, intro]:

$funas-gctxt\ C \subseteq lift-sig-RR2\ 'F \implies funas-gctxt\ (map-gctxt\ unlift-fst\ C) \subseteq \mathcal{F}$
 ⟨proof⟩

lemma *gctxt-funas-lift-RR2-funas* [simp, intro]:

$funas-gctxt\ C \subseteq \mathcal{F} \implies funas-gctxt\ (map-gctxt\ lift-fun\ C) \subseteq lift-sig-RR2\ 'F$
 ⟨proof⟩

lemma *RR2-gmctxt-unlift-lift-id* [simp, intro]:

$funas-gmctxt\ C \subseteq lift-sig-RR2\ 'F \implies (map-gmctxt\ (lift-fun \circ unlift-fst)\ C) = C$
 ⟨proof⟩

lemma *RR2-gmctxt-unlift-funas* [simp, intro]:

$funas-gmctxt\ C \subseteq lift-sig-RR2\ 'F \implies funas-gmctxt\ (map-gmctxt\ unlift-fst\ C) \subseteq$

\mathcal{F}
 ⟨proof⟩

lemma *gmctxt-fun-as-lift-RR2-fun-as* [*simp, intro*]:
 $\text{fun-as-gmctxt } C \subseteq \mathcal{F} \implies \text{fun-as-gmctxt } (\text{map-gmctxt lift-fun } C) \subseteq \text{lift-sig-RR2 } \mathcal{F}$
 ⟨proof⟩

lemma *RR2-gctxt-cl-to-gctxt*:
 assumes $\bigwedge C. P C \implies \text{fun-as-gctxt } C \subseteq \text{lift-sig-RR2 } \mathcal{F}$
 and $\bigwedge C. P C \implies R (\text{map-gctxt unlift-fst } C)$
 and $\bigwedge C. R C \implies P (\text{map-gctxt lift-fun } C)$
 shows $\{C\langle s \rangle_G \mid C s. P C \wedge Q s\} = \{(\text{map-gctxt lift-fun } C)\langle s \rangle_G \mid C s. R C \wedge Q s\}$ (is ?Ls = ?Rs)
 ⟨proof⟩

lemma *RR2-gmctxt-cl-to-gmctxt*:
 assumes $\bigwedge C. P C \implies \text{fun-as-gmctxt } C \subseteq \text{lift-sig-RR2 } \mathcal{F}$
 and $\bigwedge C. P C \implies R (\text{map-gmctxt } (\lambda f. \text{the } (\text{fst } f)) C)$
 and $\bigwedge C. R C \implies P (\text{map-gmctxt } (\lambda f. (\text{Some } f, \text{Some } f)) C)$
 shows $\{\text{fill-gholes } C ss \mid C ss. \text{num-gholes } C = \text{length } ss \wedge P C \wedge (\forall i < \text{length } ss. Q (ss ! i))\} =$
 $\{\text{fill-gholes } (\text{map-gmctxt } (\lambda f. (\text{Some } f, \text{Some } f)) C) ss \mid C ss. \text{num-gholes } C = \text{length } ss \wedge$
 $R C \wedge (\forall i < \text{length } ss. Q (ss ! i))\}$ (is ?Ls = ?Rs)
 ⟨proof⟩

lemma *RR2-id-terms-gpair-set* [*simp*]:
 $\mathcal{T}_G (\text{lift-sig-RR2 } \mathcal{F}) = \{\text{gpair } t u \mid t u. (t, u) \in \text{Restr Id } (\mathcal{T}_G \mathcal{F})\}$
 ⟨proof⟩

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**
 $\text{ggtt-lang } F G \equiv \text{map-both } \text{gterm-of-term } \mathcal{F} (\text{Restr } (\text{gtt-lang-terms } G) \{t. \text{fun-as-term } t \subseteq \text{fset } F\})$

lemma *ground-mctxt-map-vars-mctxt* [*simp*]:
 $\text{ground-mctxt } (\text{map-vars-mctxt } f C) = \text{ground-mctxt } C$

<proof>

lemma *root-single-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec A (lift-root-step F PRoot ESingle R)*

<proof>

lemma *root-strictparallel-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec A (lift-root-step F PRoot EStrictParallel R)*

<proof>

lemma *reflcl-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec (reflcl-reg (lift-sig-RR2 |[!] F) A) (lift-root-step (fset F) PRoot EParallel R)*

<proof>

lemma *parallel-closure-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec (parallel-closure-reg (lift-sig-RR2 |[!] F) A) (lift-root-step (fset F) PAny EParallel R)*

<proof>

lemma *ctxt-closure-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec (ctxt-closure-reg (lift-sig-RR2 |[!] F) A) (lift-root-step (fset F) PAny ESingle R)*

<proof>

lemma *mctxt-closure-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec (mctxt-closure-reg (lift-sig-RR2 |[!] F) A) (lift-root-step (fset F) PAny EStrictParallel R)*

<proof>

lemma *nhole-ctxt-closure-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec (nhole-ctxt-closure-reg (lift-sig-RR2 |[!] F) A) (lift-root-step (fset F) PNonRoot ESingle R)*

<proof>

lemma *nhole-mctxt-closure-automaton:*

assumes *RR2-spec A R*

shows *RR2-spec (nhole-mctxt-closure-reg (lift-sig-RR2 |[!] F) A) (lift-root-step (fset F) PNonRoot EStrictParallel R)*

<proof>

lemma *nhole-mctxt-reflcl-automaton:*

assumes $RR2\text{-spec } A R$
shows $RR2\text{-spec } (nhole\text{-mctxt-reflcl-reg } (lift\text{-sig-}RR2 \mid \mathcal{F}) A) (lift\text{-root-step } (fset \mathcal{F}) PNonRoot EParallel R)$
 $\langle proof \rangle$

definition $GTT\text{-to-}RR2\text{-root} :: ('q, 'f) gtt \Rightarrow (-, 'f\text{ option} \times 'f\text{ option}) ta$ **where**
 $GTT\text{-to-}RR2\text{-root } \mathcal{G} = pair\text{-automaton } (fst \mathcal{G}) (snd \mathcal{G})$

definition $GTT\text{-to-}RR2\text{-root-reg}$ **where**
 $GTT\text{-to-}RR2\text{-root-reg } \mathcal{G} = Reg (map\text{-both } Some \mid \mathcal{F} \mid Id\text{-on } (gtt\text{-states } \mathcal{G})) (GTT\text{-to-}RR2\text{-root } \mathcal{G})$

lemma $GTT\text{-to-}RR2\text{-root}$:
 $RR2\text{-spec } (GTT\text{-to-}RR2\text{-root-reg } \mathcal{G}) (agtt\text{-lang } \mathcal{G})$
 $\langle proof \rangle$

lemma $swap\text{-}GTT\text{-to-}RR2\text{-root}$:
 $gpair\ s\ t \in \mathcal{L} (GTT\text{-to-}RR2\text{-root-reg } (prod.swap \mathcal{G})) \longleftrightarrow$
 $gpair\ t\ s \in \mathcal{L} (GTT\text{-to-}RR2\text{-root-reg } \mathcal{G})$
 $\langle proof \rangle$

lemma $funas\text{-mctxt-map-vars-mctxt}$ [simp]:
 $funas\text{-mctxt } (map\text{-vars-mctxt } f\ C) = funas\text{-mctxt } C$
 $\langle proof \rangle$

definition $GTT\text{-to-}RR2\text{-reg} :: ('f \times nat) fset \Rightarrow ('q, 'f) gtt \Rightarrow (-, 'f\text{ option} \times 'f\text{ option}) reg$ **where**
 $GTT\text{-to-}RR2\text{-reg } F\ G = parallel\text{-closure-reg } (lift\text{-sig-}RR2 \mid \mathcal{F}) (GTT\text{-to-}RR2\text{-root-reg } G)$

lemma $agtt\text{-lang-syms}$:
 $gtt\text{-syms } \mathcal{G} \mid \subseteq \mathcal{F} \Longrightarrow agtt\text{-lang } \mathcal{G} \subseteq \{t. funas\text{-gterm } t \subseteq fset \mathcal{F}\} \times \{t. funas\text{-gterm } t \subseteq fset \mathcal{F}\}$
 $\langle proof \rangle$

lemma $gtt\text{-lang-from-agtt-lang}$:
 $gtt\text{-lang } \mathcal{G} = lift\text{-root-step } UNIV PAny EParallel (agtt\text{-lang } \mathcal{G})$
 $\langle proof \rangle$

lemma $GTT\text{-to-}RR2$:
assumes $gtt\text{-syms } \mathcal{G} \mid \subseteq \mathcal{F}$
shows $RR2\text{-spec } (GTT\text{-to-}RR2\text{-reg } \mathcal{F} \mathcal{G}) (ggtt\text{-lang } \mathcal{F} \mathcal{G})$
 $\langle proof \rangle$

end
theory $FOL\text{-Extra}$
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:*

$eval\ e\ f\ g\ (Iff\ p\ q) \longleftrightarrow (eval\ e\ f\ g\ p \longleftrightarrow eval\ e\ f\ g\ q)$
 <proof>

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* \Rightarrow (*'a*, *'b*) *form* \Rightarrow (*'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 $\bigwedge e. eval\ e\ f\ g\ (Iff\ p\ q)$

shows $eval\ e\ f\ g\ (Iff\ (apply-ctxt\ c\ p)\ (apply-ctxt\ c\ q))$

<proof>

11.3 Propositional identities

lemma *prop-ids*:

$eval\ e\ f\ g\ (Iff\ (And\ p\ q)\ (And\ q\ p))$
 $eval\ e\ f\ g\ (Iff\ (Or\ p\ q)\ (Or\ q\ p))$
 $eval\ e\ f\ g\ (Iff\ (Or\ p\ (Or\ q\ r))\ (Or\ (Or\ p\ q)\ r))$
 $eval\ e\ f\ g\ (Iff\ (And\ p\ (And\ q\ r))\ (And\ (And\ p\ q)\ r))$
 $eval\ e\ f\ g\ (Iff\ (Neg\ (Or\ p\ q))\ (And\ (Neg\ p)\ (Neg\ q)))$
 $eval\ e\ f\ g\ (Iff\ (Neg\ (And\ p\ q))\ (Or\ (Neg\ p)\ (Neg\ q)))$

$\langle proof \rangle$

11.4 de Bruijn index manipulation for formulas; cf. *liftt*

primrec *liftti* :: $nat \Rightarrow 'a\ term \Rightarrow 'a\ term$ **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'*:

$liftts\ ts = map\ liftt\ ts$
 $\langle proof \rangle$

liftt is a special case of *liftti*

lemma *lifttti-0*:

$liftti\ 0\ t = liftt\ t$
 $\langle proof \rangle$

primrec *lifti* :: $nat \Rightarrow ('a, 'b)\ form \Rightarrow ('a, 'b)\ form$ **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**

$lift \equiv lifti\ 0$

interaction of *lifti* and *eval*

lemma *evalts-def'*:

$evalts\ e\ f\ ts = map\ (evalt\ e\ f)\ ts$
 $\langle proof \rangle$

lemma *evalt-liftti*:

$evalt\ (e\langle i:z \rangle)\ f\ (liftti\ i\ t) = evalt\ e\ f\ t$
 $\langle proof \rangle$

lemma *eval-lifti* [*simp*]:

$$\text{eval } (e\langle i:z \rangle) f g (\text{lifti } i p) = \text{eval } e f g p$$

<proof>

11.5 Quantifier Identities

lemma *quant-ids*:

$$\text{eval } e f g (\text{Iff } (\text{Neg } (\text{Exists } p)) (\text{Forall } (\text{Neg } p)))$$

$$\text{eval } e f g (\text{Iff } (\text{Neg } (\text{Forall } p)) (\text{Exists } (\text{Neg } p)))$$

$$\text{eval } e f g (\text{Iff } (\text{And } p (\text{Forall } q)) (\text{Forall } (\text{And } (\text{lift } p) q)))$$

$$\text{eval } e f g (\text{Iff } (\text{And } p (\text{Exists } q)) (\text{Exists } (\text{And } (\text{lift } p) q)))$$

$$\text{eval } e f g (\text{Iff } (\text{Or } p (\text{Forall } q)) (\text{Forall } (\text{Or } (\text{lift } p) q)))$$

$$\text{eval } e f g (\text{Iff } (\text{Or } p (\text{Exists } q)) (\text{Exists } (\text{Or } (\text{lift } p) q)))$$

<proof>

11.6 Function symbols and predicates, with arities.

primrec *predas-form* :: ('a, 'b) form \Rightarrow ('b \times nat) set **where**

$$\text{predas-form } FF = \{\}$$

$$| \text{predas-form } TT = \{\}$$

$$| \text{predas-form } (\text{Pred } b \text{ ts}) = \{(b, \text{length } ts)\}$$

$$| \text{predas-form } (\text{And } p q) = \text{predas-form } p \cup \text{predas-form } q$$

$$| \text{predas-form } (\text{Or } p q) = \text{predas-form } p \cup \text{predas-form } q$$

$$| \text{predas-form } (\text{Impl } p q) = \text{predas-form } p \cup \text{predas-form } q$$

$$| \text{predas-form } (\text{Neg } p) = \text{predas-form } p$$

$$| \text{predas-form } (\text{Forall } p) = \text{predas-form } p$$

$$| \text{predas-form } (\text{Exists } p) = \text{predas-form } p$$

primrec *funas-term* :: 'a term \Rightarrow ('a \times nat) set **where**

$$\text{funas-term } (\text{Var } x) = \{\}$$

$$| \text{funas-term } (\text{App } f \text{ ts}) = \{(f, \text{length } ts)\} \cup \bigcup (\text{set } (\text{map } \text{funas-term } ts))$$

primrec *terms-form* :: ('a, 'b) form \Rightarrow 'a term set **where**

$$\text{terms-form } FF = \{\}$$

$$| \text{terms-form } TT = \{\}$$

$$| \text{terms-form } (\text{Pred } b \text{ ts}) = \text{set } ts$$

$$| \text{terms-form } (\text{And } p q) = \text{terms-form } p \cup \text{terms-form } q$$

$$| \text{terms-form } (\text{Or } p q) = \text{terms-form } p \cup \text{terms-form } q$$

$$| \text{terms-form } (\text{Impl } p q) = \text{terms-form } p \cup \text{terms-form } q$$

$$| \text{terms-form } (\text{Neg } p) = \text{terms-form } p$$

$$| \text{terms-form } (\text{Forall } p) = \text{terms-form } p$$

$$| \text{terms-form } (\text{Exists } p) = \text{terms-form } p$$

definition *funas-form* :: ('a, 'b) form \Rightarrow ('a \times nat) set **where**

$$\text{funas-form } f \equiv \bigcup (\text{funas-term } \text{' terms-form } f)$$

11.7 Negation Normal Form

inductive *is-nnf* :: ('a, 'b) form \Rightarrow bool **where**

```

is-nnf TT
| is-nnf FF
| is-nnf (Pred p ts)
| is-nnf (Neg (Pred p ts))
| is-nnf p  $\implies$  is-nnf q  $\implies$  is-nnf (And p q)
| is-nnf p  $\implies$  is-nnf q  $\implies$  is-nnf (Or p q)
| is-nnf p  $\implies$  is-nnf (Forall p)
| is-nnf p  $\implies$  is-nnf (Exists p)

```

primrec *nnf'* :: *bool* \implies ('a, 'b) form \implies ('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' ( $\neg$  b) p) (nnf' b q)
| nnf' b (Neg p)    = nnf' ( $\neg$  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)  $\longleftrightarrow$  (eval e f g p  $\longleftrightarrow$  b)
<proof>

```

lemma *is-nnf-nnf'*:

```

is-nnf (nnf' b p)
<proof>

```

abbreviation *nnf where*

```

nnf  $\equiv$  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* :: \langle (nat \implies 'c) \implies ('a \implies 'c list \implies 'c) \implies

('b \implies 'c list \implies bool) \implies ('a, 'b) form-aci \implies bool **where**

```

eval-aci e f g FF-aci       $\longleftrightarrow$  False
| eval-aci e f g TT-aci     $\longleftrightarrow$  True

```

```

| eval-aci e f g (Pred-aci b a ts)  $\longleftrightarrow$  (g a (evalts e f ts)  $\longleftrightarrow$  b)
| eval-aci e f g (And-aci ps)  $\longleftrightarrow$  fBall (fimage (eval-aci e f g) ps) id
| eval-aci e f g (Or-aci ps)  $\longleftrightarrow$  fBex (fimage (eval-aci e f g) ps) id
| eval-aci e f g (Forall-aci p)  $\longleftrightarrow$  ( $\forall z$ . eval-aci (e⟨0:z⟩) f g p)
| eval-aci e f g (Exists-aci p)  $\longleftrightarrow$  ( $\exists z$ . 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 | $\cup$ | qs)
| and-aci (And-aci ps) q = And-aci (ps | $\cup$ | {|q|})
| and-aci p (And-aci qs) = And-aci ({|p|} | $\cup$ | 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)  $\longleftrightarrow$  eval-aci e f g p  $\wedge$  eval-aci e f g q
⟨proof⟩

```

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 | $\cup$ | qs)
| or-aci (Or-aci ps) q = Or-aci (ps | $\cup$ | {|q|})
| or-aci p (Or-aci qs) = Or-aci ({|p|} | $\cup$ | 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)  $\longleftrightarrow$  eval-aci e f g p  $\vee$  eval-aci e f g q
⟨proof⟩

```

declare or-aci.simps [simp del]

convert negation normal form to ACIU01 normal form

fun nnf-to-aci :: ('a, 'b) form \Rightarrow ('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  $\implies$  eval-aci e f g (nnf-to-aci p)  $\longleftrightarrow$  eval e f g p
<proof>

```

11.9 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.

lemma *check-equivalence-by-nnf-aci*:

```

nnf-to-aci (nnf p) = nnf-to-aci (nnf q)  $\implies$  eval e f g p  $\longleftrightarrow$  eval e f g q
<proof>

```

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* :: $\langle \text{nat} \Rightarrow 'c \rangle \Rightarrow ('a \Rightarrow 'c \text{ list} \Rightarrow 'c) \Rightarrow$

$('b \Rightarrow 'c \text{ list} \Rightarrow \text{bool}) \Rightarrow ('a, 'b) \text{ form-list-aci} \Rightarrow \text{bool}$ **where**

```

eval-list-aci e f g FF-aci       $\longleftrightarrow$  False
| eval-list-aci e f g TT-aci     $\longleftrightarrow$  True
| eval-list-aci e f g (Pred-aci b a ts)  $\longleftrightarrow$  (g a (evalts e f ts)  $\longleftrightarrow$  b)
| eval-list-aci e f g (And-aci ps)   $\longleftrightarrow$  list-all ( $\lambda$  fm. eval-list-aci e f g fm) ps
| eval-list-aci e f g (Or-aci ps)    $\longleftrightarrow$  list-ex ( $\lambda$  fm. eval-list-aci e f g fm) ps
| eval-list-aci e f g (Forall-aci p)  $\longleftrightarrow$  ( $\forall z$ . eval-list-aci (e(0:z)) f g p)
| eval-list-aci e f g (Exists-aci p)  $\longleftrightarrow$  ( $\exists z$ . 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*]:
 $eval-list-aci\ e\ f\ g\ (and-list-aci\ p\ q) \longleftrightarrow eval-list-aci\ e\ f\ g\ p \wedge eval-list-aci\ e\ f\ g\ q$
<proof>

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*]:

$eval-list-aci\ e\ f\ g\ (or-list-aci\ p\ q) \longleftrightarrow eval-list-aci\ e\ f\ g\ p \vee eval-list-aci\ e\ f\ g\ q$
<proof>

declare *or-list-aci.simps* [*simp del*]

convert negation normal form to ACIU01 normal form

fun *nnf-to-list-aci* :: ('a, 'b) *form* \Rightarrow ('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*:

$is-nnf\ p \Longrightarrow eval-list-aci\ e\ f\ g\ (nnf-to-list-aci\ p) \longleftrightarrow eval\ e\ f\ g\ p$
<proof>

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
 ⟨*proof*⟩

lemma *ord-form-list-idemp*:
ord-form-list-aci (ord-form-list-aci q) = ord-form-list-aci q
 ⟨*proof*⟩

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
 ⟨*proof*⟩

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)) \implies eval e f g p \longleftrightarrow eval e f g q
 ⟨*proof*⟩

hide-type (**open**) *term*
hide-const (**open**) *Var*
hide-type (**open**) *ctx*

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* \Rightarrow *ftrs list* \Rightarrow (*f*, *v*) *trs* **where**
is-to-trs Rs is = \bigcup (set (map (case-ftrs (!) Rs) ((\cdot) prod.swap \circ (!) Rs)) is))

primrec *eval-gtt-rel* :: (*f* \times *nat*) *set* \Rightarrow (*f*, *v*) *trs list* \Rightarrow *ftrs gtt-rel* \Rightarrow *f gterm rel* **where**
eval-gtt-rel \mathcal{F} Rs (ARoot is) = Restr (grrstep (is-to-trs Rs is)) ($\mathcal{T}_G \mathcal{F}$)

$| \text{eval-gtt-rel } \mathcal{F} \text{ Rs } (GInv \ g) = \text{prod.swap } ' (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g)$
 $| \text{eval-gtt-rel } \mathcal{F} \text{ Rs } (AUnion \ g1 \ g2) = (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g1) \cup (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g2)$
 $| \text{eval-gtt-rel } \mathcal{F} \text{ Rs } (ATrancl \ g) = (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g)^+$
 $| \text{eval-gtt-rel } \mathcal{F} \text{ Rs } (AComp \ g1 \ g2) = (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g1) \ O \ (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g2)$
 $| \text{eval-gtt-rel } \mathcal{F} \text{ Rs } (GTrancl \ g) = \text{gtrancl-rel } \mathcal{F} \ (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g)$
 $| \text{eval-gtt-rel } \mathcal{F} \text{ Rs } (GComp \ g1 \ g2) = \text{gcomp-rel } \mathcal{F} \ (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g1) \ (\text{eval-gtt-rel } \mathcal{F} \text{ Rs } \ g2)$

primrec $\text{eval-rr1-rel} :: ('f \times \text{nat}) \text{ set} \Rightarrow ('f, 'v) \text{ trs list} \Rightarrow \text{ftrs rr1-rel} \Rightarrow 'f \text{ gterm set}$

and $\text{eval-rr2-rel} :: ('f \times \text{nat}) \text{ set} \Rightarrow ('f, 'v) \text{ trs list} \Rightarrow \text{ftrs rr2-rel} \Rightarrow 'f \text{ gterm rel}$
where

$\text{eval-rr1-rel } \mathcal{F} \text{ Rs } R1Terms = (\mathcal{T}_G \ \mathcal{F})$
 $| \text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1Union \ R \ S) = (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ R) \cup (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1Inter \ R \ S) = (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ R) \cap (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1Diff \ R \ S) = (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ R) - (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1Proj \ n \ R) = (\text{case } n \ \text{of } 0 \Rightarrow \text{fst } ' (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R)$
 $\quad \quad \quad | - \Rightarrow \text{snd } ' (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R))$
 $| \text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1NF \ is) = NF \ (\text{Restr } (\text{grstep } (\text{is-to-trs } \ \text{Rs } \ \text{is})) \ (\mathcal{T}_G \ \mathcal{F})) \cap$
 $\quad (\mathcal{T}_G \ \mathcal{F})$
 $| \text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1Inf \ R) = \{s. \ \text{infinite } (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R \ \text{“ } \{s\})\}$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2GTT-Rel \ A \ W \ X) = \text{lift-root-step } \mathcal{F} \ W \ X \ (\text{eval-gtt-rel } \mathcal{F} \ \text{Rs } \ A)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Inv \ R) = \text{prod.swap } ' (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Union \ R \ S) = (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R) \cup (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Inter \ R \ S) = (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R) \cap (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Diff \ R \ S) = (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R) - (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Comp \ R \ S) = (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ R) \ O \ (\text{eval-rr2-rel } \mathcal{F} \text{ Rs } \ S)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Diag \ R) = \text{Id-on } (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ R)$
 $| \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Prod \ R \ S) = (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ R) \times (\text{eval-rr1-rel } \mathcal{F} \text{ Rs } \ S)$

12.1 Semantics of Formulas

fun $\text{eval-formula} :: ('f \times \text{nat}) \text{ set} \Rightarrow ('f, 'v) \text{ trs list} \Rightarrow (\text{nat} \Rightarrow 'f \text{ gterm}) \Rightarrow$
 $\text{ftrs formula} \Rightarrow \text{bool}$ **where**

$\text{eval-formula } \mathcal{F} \text{ Rs } \alpha \ (FRR1 \ r1 \ x) \longleftrightarrow \alpha \ x \in \text{eval-rr1-rel } \mathcal{F} \ \text{Rs } \ r1$
 $| \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ (FRR2 \ r2 \ x \ y) \longleftrightarrow (\alpha \ x, \alpha \ y) \in \text{eval-rr2-rel } \mathcal{F} \ \text{Rs } \ r2$
 $| \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ (FAnd \ fs) \longleftrightarrow (\forall f \in \text{set } fs. \ \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ f)$
 $| \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ (FOr \ fs) \longleftrightarrow (\exists f \in \text{set } fs. \ \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ f)$
 $| \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ (FNot \ f) \longleftrightarrow \neg \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ f$
 $| \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ (FExists \ f) \longleftrightarrow (\exists z \in \mathcal{T}_G \ \mathcal{F}. \ \text{eval-formula } \mathcal{F} \ \text{Rs } \ (\alpha \langle 0 : z \rangle))$
 $f)$
 $| \text{eval-formula } \mathcal{F} \ \text{Rs } \ \alpha \ (FForall \ f) \longleftrightarrow (\forall z \in \mathcal{T}_G \ \mathcal{F}. \ \text{eval-formula } \mathcal{F} \ \text{Rs } \ (\alpha \langle 0 : z \rangle))$
 $f)$

```

fun formula-arity :: 'trs formula  $\Rightarrow$  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  $R \subseteq \mathcal{F} \forall t. (\text{term-of-gterm } s, \text{term-of-gterm } t) \in \text{rstep } R \longrightarrow$ 
 $\neg \text{funas-gterm } t \subseteq \mathcal{F}$ 
and funas-gterm  $s \subseteq \mathcal{F} (l, r) \in R \text{ term-of-gterm } s = C(l \cdot (\sigma :: 'b \Rightarrow ('a, 'b)$ 
Term.term))
shows False
<proof>

```

The central property we are interested in is satisfiability

definition formula-satisfiable **where**

```

formula-satisfiable  $\mathcal{F} R s f \longleftrightarrow (\exists \alpha. \text{range } \alpha \subseteq \mathcal{T}_G \mathcal{F} \wedge \text{eval-formula } \mathcal{F} R s \alpha f)$ 

```

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  $\text{gcomp-rel } \mathcal{F} R S \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ 
<proof>

```

lemma *gtrancl-rel-sig*:

```

assumes  $R \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ 
shows  $\text{gtrancl-rel } \mathcal{F} R \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ 
<proof>

```

lemma *gtrancl-rel*:

```

assumes  $R \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ 
shows  $\text{lift-root-step } \mathcal{F} \text{ PAny EStrictParallel } (\text{gtrancl-rel } \mathcal{F} R) = (\text{lift-root-step } \mathcal{F}$ 
 $\text{ PAny ESingle } R)^+$ 
<proof>

```

lemma *gtrancl-rel'*:

```

assumes  $R \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$ 
shows  $\text{lift-root-step } \mathcal{F} \text{ PAny EParallel } (\text{gtrancl-rel } \mathcal{F} R) = \text{Restr } ((\text{lift-root-step}$ 
 $\mathcal{F} \text{ PAny ESingle } R)^* (\mathcal{T}_G \mathcal{F}))$ 
<proof>

```

GTT relation semantics respects the signature

lemma *eval-gtt-rel-sig*:

$$\text{eval-gtt-rel } \mathcal{F} \text{ Rs } g \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$$

<proof>

RR1 and RR2 relation semantics respect the signature

lemma *eval-rr12-rel-sig*:

$$\text{eval-rr1-rel } \mathcal{F} \text{ Rs } r1 \subseteq \mathcal{T}_G \mathcal{F}$$

$$\text{eval-rr2-rel } \mathcal{F} \text{ Rs } r2 \subseteq \mathcal{T}_G \mathcal{F} \times \mathcal{T}_G \mathcal{F}$$

<proof>

12.4 Correctness of derived constructions

lemma *R1Fin*:

$$\text{eval-rr1-rel } \mathcal{F} \text{ Rs } (R1Fin \ r) = \{t \in \mathcal{T}_G \mathcal{F}. \text{finite } \{s. (t, s) \in \text{eval-rr2-rel } \mathcal{F} \text{ Rs } r\}\}$$

<proof>

lemma *R2Eq*:

$$\text{eval-rr2-rel } \mathcal{F} \text{ Rs } R2Eq = \text{Id-on } (\mathcal{T}_G \mathcal{F})$$

<proof>

lemma *R2Reflc*:

$$\text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Reflc \ r) = \text{eval-rr2-rel } \mathcal{F} \text{ Rs } r \cup \text{Id-on } (\mathcal{T}_G \mathcal{F})$$

$$\text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Reflc \ r) = \text{Restr } ((\text{eval-rr2-rel } \mathcal{F} \text{ Rs } r)^=) (\mathcal{T}_G \mathcal{F})$$

<proof>

lemma *R2Step*:

$$\text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Step \ ts) = \text{Restr } (\text{grstep } (\text{is-to-trs } \text{Rs } ts)) (\mathcal{T}_G \mathcal{F})$$

<proof>

lemma *R2StepEq*:

$$\text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2StepEq \ ts) = \text{Restr } ((\text{grstep } (\text{is-to-trs } \text{Rs } ts))^=) (\mathcal{T}_G \mathcal{F})$$

<proof>

lemma *R2Steps*:

$$\text{fixes } \mathcal{F} \text{ Rs } ts \text{ defines } R \equiv \text{Restr } (\text{grstep } (\text{is-to-trs } \text{Rs } ts)) (\mathcal{T}_G \mathcal{F})$$

$$\text{shows } \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2Steps \ ts) = R^+$$

<proof>

lemma *R2StepsEq*:

$$\text{fixes } \mathcal{F} \text{ Rs } ts \text{ defines } R \equiv \text{Restr } (\text{grstep } (\text{is-to-trs } \text{Rs } ts)) (\mathcal{T}_G \mathcal{F})$$

$$\text{shows } \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2StepsEq \ ts) = \text{Restr } (R^*) (\mathcal{T}_G \mathcal{F})$$

<proof>

lemma *R2StepsNF*:

$$\text{fixes } \mathcal{F} \text{ Rs } ts \text{ defines } R \equiv \text{Restr } (\text{grstep } (\text{is-to-trs } \text{Rs } ts)) (\mathcal{T}_G \mathcal{F})$$

$$\text{shows } \text{eval-rr2-rel } \mathcal{F} \text{ Rs } (R2StepsNF \ ts) = \text{Restr } (R^* \cap \text{UNIV} \times \text{NF } R) (\mathcal{T}_G \mathcal{F})$$

<proof>

lemma *R2ParStep*:

$eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2ParStep ts) = Restr (gpar\text{-}rstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *R2RootStep*:

$eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2RootStep ts) = Restr (grrstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *R2RootStepEq*:

$eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2RootStepEq ts) = Restr ((grrstep (is\text{-}to\text{-}trs Rs ts))^=) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *R2RootSteps*:

fixes $\mathcal{F} Rs ts$ **defines** $R \equiv Restr (grrstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
shows $eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2RootSteps ts) = R^+$
<proof>

lemma *R2RootStepsEq*:

fixes $\mathcal{F} Rs ts$ **defines** $R \equiv Restr (grrstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
shows $eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2RootStepsEq ts) = Restr (R^*) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *R2NonRootStep*:

$eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2NonRootStep ts) = Restr (gnrrstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *R2NonRootStepEq*:

$eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2NonRootStepEq ts) = Restr ((gnrrstep (is\text{-}to\text{-}trs Rs ts))^=) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *R2NonRootSteps*:

fixes $\mathcal{F} Rs ts$ **defines** $R \equiv Restr (gnrrstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
shows $eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2NonRootSteps ts) = R^+$
<proof>

lemma *R2NonRootStepsEq*:

fixes $\mathcal{F} Rs ts$ **defines** $R \equiv Restr (gnrrstep (is\text{-}to\text{-}trs Rs ts)) (\mathcal{T}_G \mathcal{F})$
shows $eval\text{-}rr2\text{-}rel \mathcal{F} Rs (R2NonRootStepsEq ts) = Restr (R^*) (\mathcal{T}_G \mathcal{F})$
<proof>

lemma *converse-to-prod-swap*:

$R^{-1} = prod.swap \text{ ` } R$
<proof>

lemma *R2Meet*:
fixes \mathcal{F} R ts **defines** $R \equiv \text{Restr } (\text{grstep } (\text{is-to-trs } R \text{ } ts)) (\mathcal{T}_G \mathcal{F})$
shows $\text{eval-rr2-rel } \mathcal{F} \text{ } R \text{ } (R2Meet \text{ } ts) = \text{Restr } ((R^{-1})^* \text{ } O \text{ } R^*) (\mathcal{T}_G \mathcal{F})$
 $\langle \text{proof} \rangle$

lemma *R2Join*:
fixes \mathcal{F} R ts **defines** $R \equiv \text{Restr } (\text{grstep } (\text{is-to-trs } R \text{ } ts)) (\mathcal{T}_G \mathcal{F})$
shows $\text{eval-rr2-rel } \mathcal{F} \text{ } R \text{ } (R2Join \text{ } ts) = \text{Restr } (R^* \text{ } O \text{ } (R^{-1})^*) (\mathcal{T}_G \mathcal{F})$
 $\langle \text{proof} \rangle$

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 $(f, v) \text{ fin-trs} = (f, v) \text{ rule fset}$

lemma *tl-drop-conv*:

$tl \text{ } xs = \text{drop } 1 \text{ } xs$

$\langle \text{proof} \rangle$

definition *rrn-drop-fst where*

$rrn\text{-drop-fst } \mathcal{A} = \text{relabel-reg } (\text{trim-reg } (\text{collapse-automaton-reg } (\text{fmap-funs-reg } (\text{drop-none-rule } 1) (\text{trim-reg } \mathcal{A}))))$

lemma *rrn-drop-fst-lang*:

assumes $RRn\text{-spec } n \text{ } A \text{ } T \text{ } 1 < n$

shows $RRn\text{-spec } (n - 1) \text{ } (rrn\text{-drop-fst } A) \text{ } (\text{drop } 1 \text{ } T)$

$\langle \text{proof} \rangle$

definition *liftO1* :: $(a \Rightarrow b) \Rightarrow 'a \text{ option} \Rightarrow 'b \text{ option}$ **where**

$\text{liftO1} = \text{map-option}$

definition *liftO2* :: $(a \Rightarrow b \Rightarrow c) \Rightarrow 'a \text{ option} \Rightarrow 'b \text{ option} \Rightarrow 'c \text{ option}$ **where**

$\text{liftO2 } f \text{ } a \text{ } b = \text{case-option } \text{None } (\lambda a'. \text{liftO1 } (f \text{ } a') \text{ } b) \text{ } a$

lemma *liftO1-Some* [simp]:
 $\text{liftO1 } f \ x = \text{Some } y \iff (\exists x'. x = \text{Some } x') \wedge y = f \ (\text{the } x)$
 ⟨proof⟩

lemma *liftO2-Some* [simp]:
 $\text{liftO2 } f \ x \ y = \text{Some } z \iff (\exists x' \ y'. x = \text{Some } x' \wedge y = \text{Some } y') \wedge z = f \ (\text{the } x) \ (\text{the } y)$
 ⟨proof⟩

13.1 Computing TRSs

lemma *is-to-trs-props*:
assumes $\forall R \in \text{set } Rs. \text{finite } R \wedge \text{lv-trs } R \wedge \text{funas-trs } R \subseteq \mathcal{F} \ \forall i \in \text{set } is. \text{case-ftrs id id } i < \text{length } Rs$
shows $\text{funas-trs } (\text{is-to-trs } Rs \ is) \subseteq \mathcal{F} \ \text{lv-trs } (\text{is-to-trs } Rs \ is) \ \text{finite } (\text{is-to-trs } Rs \ is)$
 ⟨proof⟩

definition *is-to-fin-trs* :: $(f, 'v) \text{ fin-trs list} \Rightarrow \text{ftrs list} \Rightarrow (f, 'v) \text{ fin-trs}$ **where**
 $\text{is-to-fin-trs } Rs \ is = |\bigcup| \ (\text{fset-of-list } (\text{map } (\text{case-ftrs } (!) \ Rs) \ (|!|) \ \text{prod.swap } \circ (!) \ Rs)) \ is)$

lemma *is-to-fin-trs-conv*:
assumes $\forall i \in \text{set } is. \text{case-ftrs id id } i < \text{length } Rs$
shows $\text{is-to-trs } (\text{map } \text{fset } Rs) \ is = \text{fset } (\text{is-to-fin-trs } Rs \ is)$
 ⟨proof⟩

definition *is-to-trs'* :: $(f, 'v) \text{ fin-trs list} \Rightarrow \text{ftrs list} \Rightarrow (f, 'v) \text{ fin-trs option}$ **where**
 $\text{is-to-trs}' \ Rs \ is = \text{do } \{$
 $\quad \text{guard } (\forall i \in \text{set } is. \text{case-ftrs id id } i < \text{length } Rs);$
 $\quad \text{Some } (\text{is-to-fin-trs } Rs \ is)$
 $\}$

lemma *is-to-trs-conv*:
 $\text{is-to-trs}' \ Rs \ is = \text{Some } S \implies \text{is-to-trs } (\text{map } \text{fset } Rs) \ is = \text{fset } S$
 ⟨proof⟩

lemma *is-to-trs'-props*:
assumes $\forall R \in \text{set } Rs. \text{lv-trs } (\text{fset } R) \wedge \text{ffunas-trs } R \subseteq \mathcal{F}$ **and** $\text{is-to-trs}' \ Rs \ is = \text{Some } S$
shows $\text{ffunas-trs } S \subseteq \mathcal{F} \ \text{lv-trs } (\text{fset } S)$
 ⟨proof⟩

13.2 Computing GTTs

fun *gtt-of-gtt-rel* :: $(f \times \text{nat}) \text{ fset} \Rightarrow (f :: \text{linorder}, 'v) \text{ fin-trs list} \Rightarrow \text{ftrs gtt-rel}$
 $\Rightarrow (\text{nat}, f) \text{ gtt option}$ **where**

$gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (ARoot \text{ } is) = liftO1 \text{ } (\lambda R. \text{ relabel-gtt } (agtt\text{-grrstep } R \text{ } \mathcal{F}))$
 $(is\text{-to-trs}' \text{ } Rs \text{ } is)$
 $| \text{ } gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (GInv \text{ } g) = liftO1 \text{ } prod.swap \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g)$
 $| \text{ } gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (AUnion \text{ } g1 \text{ } g2) = liftO2 \text{ } (\lambda g1 \text{ } g2. \text{ relabel-gtt } (AGTT\text{-union}' \text{ } g1 \text{ } g2)) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g1) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g2)$
 $| \text{ } gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (ATrancl \text{ } g) = liftO1 \text{ } (relabel-gtt \text{ } \circ \text{ } AGTT\text{-trancl}) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g)$
 $| \text{ } gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (GTrancl \text{ } g) = liftO1 \text{ } GTT\text{-trancl} \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g)$
 $| \text{ } gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (AComp \text{ } g1 \text{ } g2) = liftO2 \text{ } (\lambda g1 \text{ } g2. \text{ relabel-gtt } (AGTT\text{-comp}' \text{ } g1 \text{ } g2)) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g1) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g2)$
 $| \text{ } gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } (GComp \text{ } g1 \text{ } g2) = liftO2 \text{ } (\lambda g1 \text{ } g2. \text{ relabel-gtt } (GTT\text{-comp}' \text{ } g1 \text{ } g2)) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g1) \text{ } (gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g2)$

lemma *gtt-of-gtt-rel-correct*:

assumes $\forall R \in \text{set } Rs. \text{ lv-trs } (fset \text{ } R) \wedge \text{ ffunas-trs } R \text{ } |\subseteq| \text{ } \mathcal{F}$
shows $gtt\text{-of-gtt-rel } \mathcal{F} \text{ } Rs \text{ } g = \text{Some } g' \implies agtt\text{-lang } g' = \text{eval-gtt-rel } (fset \text{ } \mathcal{F})$
 $(\text{map } fset \text{ } Rs) \text{ } g$
 $\langle \text{proof} \rangle$

13.3 Computing RR1 and RR2 relations

definition *simplify-reg* $\mathcal{A} = (\text{relabel-reg } (\text{trim-reg } \mathcal{A}))$

lemma *L-simplify-reg [simp]*: $\mathcal{L} \text{ } (simplify\text{-reg } \mathcal{A}) = \mathcal{L} \text{ } \mathcal{A}$
 $\langle \text{proof} \rangle$

lemma *RR1-spec-simplify-reg [simp]*:

$RR1\text{-spec } (simplify\text{-reg } \mathcal{A}) \text{ } R = RR1\text{-spec } \mathcal{A} \text{ } R$
 $\langle \text{proof} \rangle$

lemma *RR2-spec-simplify-reg [simp]*:

$RR2\text{-spec } (simplify\text{-reg } \mathcal{A}) \text{ } R = RR2\text{-spec } \mathcal{A} \text{ } R$
 $\langle \text{proof} \rangle$

lemma *RRn-spec-simplify-reg [simp]*:

$RRn\text{-spec } n \text{ } (simplify\text{-reg } \mathcal{A}) \text{ } R = RRn\text{-spec } n \text{ } \mathcal{A} \text{ } R$
 $\langle \text{proof} \rangle$

lemma *RR1-spec-eps-free-reg [simp]*:

$RR1\text{-spec } (eps\text{-free-reg } \mathcal{A}) \text{ } R = RR1\text{-spec } \mathcal{A} \text{ } R$
 $\langle \text{proof} \rangle$

lemma *RR2-spec-eps-free-reg [simp]*:

$RR2\text{-spec } (eps\text{-free-reg } \mathcal{A}) \text{ } R = RR2\text{-spec } \mathcal{A} \text{ } R$
 $\langle \text{proof} \rangle$

lemma *RRn-spec-eps-free-reg [simp]*:

$RRn\text{-spec } n \text{ } (eps\text{-free-reg } \mathcal{A}) \text{ } R = RRn\text{-spec } n \text{ } \mathcal{A} \text{ } R$
 $\langle \text{proof} \rangle$

fun *rr1-of-rr1-rel* :: $(f \times \text{nat}) \text{ } fset \implies (f :: \text{linorder}, 'v) \text{ } fin\text{-trs } list \implies \text{ftrs } rr1\text{-rel}$
 $\implies (\text{nat}, 'f) \text{ } reg \text{ } option$

and $rr2\text{-of-}rr2\text{-rel} :: ('f \times nat) \text{ fset} \Rightarrow ('f, 'v) \text{ fin-trs list} \Rightarrow \text{ftrs } rr2\text{-rel} \Rightarrow (nat, 'f \text{ option} \times 'f \text{ option}) \text{ reg option}$ **where**

$rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } R1Terms = \text{Some } (\text{relabel-reg } (\text{term-reg } \mathcal{F}))$
 $| rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } (R1NF \text{ is}) = \text{liftO1 } (\lambda R. (\text{simplify-reg } (\text{nf-reg } (\text{fst } |\uparrow| R) \mathcal{F})))$
 $(\text{is-to-trs}' \text{ Rs is})$
 $| rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } (R1Inf r) = \text{liftO1 } (\lambda R.$
 $\quad \text{let } \mathcal{A} = \text{trim-reg } R \text{ in}$
 $\quad \text{simplify-reg } (\text{proj-1-reg } (\text{Inf-reg-impl } \mathcal{A}))$
 $\quad) (rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } r)$
 $| rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } (R1Proj i r) = (\text{case } i \text{ of } 0 \Rightarrow$
 $\quad \text{liftO1 } (\text{trim-reg } \circ \text{proj-1-reg}) (rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } r)$
 $\quad | - \Rightarrow \text{liftO1 } (\text{trim-reg } \circ \text{proj-2-reg}) (rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } r))$
 $| rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } (R1Union s1 s2) =$
 $\quad \text{liftO2 } (\lambda x y. \text{relabel-reg } (\text{reg-union } x y)) (rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s1) (rr1\text{-of-}rr1\text{-rel}$
 $\mathcal{F} \text{ Rs } s2)$
 $| rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } (R1Inter s1 s2) =$
 $\quad \text{liftO2 } (\lambda x y. \text{simplify-reg } (\text{reg-intersect } x y)) (rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s1) (rr1\text{-of-}rr1\text{-rel}$
 $\mathcal{F} \text{ Rs } s2)$
 $| rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } (R1Diff s1 s2) = \text{liftO2 } (\lambda x y. \text{relabel-reg } (\text{trim-reg } (\text{difference-reg}$
 $x y))) (rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s1) (rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s2)$

$| rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } (R2GTT\text{-Rel } g w x) =$
 $\quad (\text{case } w \text{ of } PRoot \Rightarrow$
 $\quad \quad (\text{case } x \text{ of } ESingl e \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{GTT-to-RR2-root-reg})$
 $\quad \quad (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)$
 $\quad \quad | EParallel \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{reflcl-reg } (\text{lift-sig-RR2 } |\uparrow|$
 $\mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)$
 $\quad \quad | EStrictParallel \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{GTT-to-RR2-root-reg})$
 $\quad \quad (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g))$
 $\quad \quad | PNonRoot \Rightarrow$
 $\quad \quad \quad (\text{case } x \text{ of } ESingl e \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{nhole-ctxt-closure-reg}$
 $\quad \quad \quad (\text{lift-sig-RR2 } |\uparrow| \mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)$
 $\quad \quad \quad | EParallel \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{nhole-mctxt-reflcl-reg}$
 $\quad \quad \quad (\text{lift-sig-RR2 } |\uparrow| \mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)$
 $\quad \quad \quad | EStrictParallel \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{nhole-mctxt-closure-reg}$
 $\quad \quad \quad (\text{lift-sig-RR2 } |\uparrow| \mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g))$
 $\quad \quad \quad | PAny \Rightarrow$
 $\quad \quad \quad \quad (\text{case } x \text{ of } ESingl e \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{ctxt-closure-reg}$
 $\quad \quad \quad \quad (\text{lift-sig-RR2 } |\uparrow| \mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)$
 $\quad \quad \quad \quad | EParallel \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{parallel-closure-reg}$
 $\quad \quad \quad \quad (\text{lift-sig-RR2 } |\uparrow| \mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)$
 $\quad \quad \quad \quad | EStrictParallel \Rightarrow \text{liftO1 } (\text{simplify-reg } \circ \text{eps-free-reg } \circ \text{mctxt-closure-reg}$
 $\quad \quad \quad \quad (\text{lift-sig-RR2 } |\uparrow| \mathcal{F}) \circ \text{GTT-to-RR2-root-reg}) (\text{gtt-of-gtt-rel } \mathcal{F} \text{ Rs } g)))$
 $| rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } (R2Diag s) =$
 $\quad \text{liftO1 } (\lambda x. \text{fmap-funs-reg } (\lambda f. (\text{Some } f, \text{Some } f)) x) (rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s)$
 $| rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } (R2Prod s1 s2) =$
 $\quad \text{liftO2 } (\lambda x y. \text{simplify-reg } (\text{pair-automaton-reg } x y)) (rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s1)$
 $(rr1\text{-of-}rr1\text{-rel } \mathcal{F} \text{ Rs } s2)$
 $| rr2\text{-of-}rr2\text{-rel } \mathcal{F} \text{ Rs } (R2Inv r) = \text{liftO1 } (\text{fmap-funs-reg } \text{prod.swap}) (rr2\text{-of-}rr2\text{-rel}$

$\mathcal{F} \text{ Rs } r$
 $| \text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } (R2Union \text{ r1 } \text{ r2}) =$
 $\text{ liftO2 } (\lambda x y. \text{ relabel-reg } (\text{ reg-union } x y)) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r1}) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r2})$
 $| \text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } (R2Inter \text{ r1 } \text{ r2}) =$
 $\text{ liftO2 } (\lambda x y. \text{ simplify-reg } (\text{ reg-intersect } x y)) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r1}) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r2})$
 $| \text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } (R2Diff \text{ r1 } \text{ r2}) = \text{ liftO2 } (\lambda x y. \text{ simplify-reg } (\text{ difference-reg } x y)) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r1}) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r2})$
 $| \text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } (R2Comp \text{ r1 } \text{ r2}) = \text{ liftO2 } (\lambda x y. \text{ simplify-reg } (\text{ rr2-compositon } \mathcal{F} x y)) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r1}) (\text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r2})$

abbreviation *lhss* **where**

$lhss \text{ R} \equiv fst \mid \uparrow \text{ R}$

lemma *rr12-of-rr12-rel-correct*:

fixes $\text{ Rs} :: (('f :: \text{ linorder}, 'v) \text{ Term.term} \times ('f, 'v) \text{ Term.term}) \text{ fset list}$

assumes $\forall R \in \text{ set } \text{ Rs}. \text{ lv-trs } (\text{ fset } R) \wedge \text{ ffunas-trs } R \mid \subseteq \mid \mathcal{F}$

shows $\forall ta1. \text{ rr1-of-rr1-rel } \mathcal{F} \text{ Rs } \text{ r1} = \text{ Some } ta1 \longrightarrow \text{ RR1-spec } ta1 (\text{ eval-rr1-rel } (\text{ fset } \mathcal{F}) (\text{ map } \text{ fset } \text{ Rs}) \text{ r1})$

$\forall ta2. \text{ rr2-of-rr2-rel } \mathcal{F} \text{ Rs } \text{ r2} = \text{ Some } ta2 \longrightarrow \text{ RR2-spec } ta2 (\text{ eval-rr2-rel } (\text{ fset } \mathcal{F}) (\text{ map } \text{ fset } \text{ Rs}) \text{ r2})$

<proof>

13.4 Misc

lemma *eval-formula-arity-cong*:

assumes $\bigwedge i. i < \text{ formula-arity } f \implies \alpha' i = \alpha i$

shows $\text{ eval-formula } \mathcal{F} \text{ Rs } \alpha' f = \text{ eval-formula } \mathcal{F} \text{ Rs } \alpha f$

<proof>

13.5 Connect semantics to FOL-Fitting

primrec *form-of-formula* :: $'\text{trs formula} \Rightarrow (\text{unit}, '\text{trs rr1-rel} + '\text{trs rr2-rel}) \text{ form}$

where

$\text{ form-of-formula } (\text{ FRR1 } \text{ r1 } x) = \text{ Pred } (\text{ Inl } \text{ r1}) [\text{ Var } x]$

$| \text{ form-of-formula } (\text{ FRR2 } \text{ r2 } x y) = \text{ Pred } (\text{ Inr } \text{ r2}) [\text{ Var } x, \text{ Var } y]$

$| \text{ form-of-formula } (\text{ FAnd } \text{ fs}) = \text{ foldr } \text{ And } (\text{ map } \text{ form-of-formula } \text{ fs}) \text{ TT}$

$| \text{ form-of-formula } (\text{ FOr } \text{ fs}) = \text{ foldr } \text{ Or } (\text{ map } \text{ form-of-formula } \text{ fs}) \text{ FF}$

$| \text{ form-of-formula } (\text{ FNot } f) = \text{ Neg } (\text{ form-of-formula } f)$

$| \text{ form-of-formula } (\text{ FExists } f) = \text{ Exists } (\text{ And } (\text{ Pred } (\text{ Inl } \text{ R1Terms}) [\text{ Var } 0])) (\text{ form-of-formula } f)$

$| \text{ form-of-formula } (\text{ FForall } f) = \text{ Forall } (\text{ Impl } (\text{ Pred } (\text{ Inl } \text{ R1Terms}) [\text{ Var } 0])) (\text{ form-of-formula } f)$

fun *for-eval-rel* :: $('f \times \text{ nat}) \text{ set} \Rightarrow ('f, 'v) \text{ trs list} \Rightarrow \text{ ftrs rr1-rel} + \text{ ftrs rr2-rel} \Rightarrow 'f \text{ gterm list} \Rightarrow \text{ bool}$ **where**

for-eval-rel \mathcal{F} Rs (Inl r1) [t] \longleftrightarrow $t \in \text{eval-rr1-rel } \mathcal{F}$ Rs r1
 | for-eval-rel \mathcal{F} Rs (Inr r2) [t, u] \longleftrightarrow $(t, u) \in \text{eval-rr2-rel } \mathcal{F}$ 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⟩

13.6 RRn relations and formulas

lemma shift-rangeI [intro!]:

range $\alpha \subseteq T \implies x \in T \implies \text{range } (\text{shift } \alpha \ i \ x) \subseteq T$
 ⟨proof⟩

definition formula-relevant where

formula-relevant \mathcal{F} Rs vs fm \longleftrightarrow
 $(\forall \alpha \ \alpha'. \text{range } \alpha \subseteq \mathcal{T}_G \ \mathcal{F} \implies \text{range } \alpha' \subseteq \mathcal{T}_G \ \mathcal{F} \implies \text{map } \alpha \ \text{vs} = \text{map } \alpha' \ \text{vs} \implies \text{eval-formula } \mathcal{F} \ \text{Rs } \alpha \ \text{fm} \implies \text{eval-formula } \mathcal{F} \ \text{Rs } \alpha' \ \text{fm})$

lemma formula-relevant-mono:

set vs \subseteq set ws \implies formula-relevant \mathcal{F} Rs vs fm \implies formula-relevant \mathcal{F} Rs ws fm
 ⟨proof⟩

lemma formula-relevantD:

formula-relevant \mathcal{F} Rs vs fm \implies
 range $\alpha \subseteq \mathcal{T}_G \ \mathcal{F} \implies \text{range } \alpha' \subseteq \mathcal{T}_G \ \mathcal{F} \implies \text{map } \alpha \ \text{vs} = \text{map } \alpha' \ \text{vs} \implies$
 eval-formula \mathcal{F} Rs α fm \implies eval-formula \mathcal{F} Rs α' fm
 ⟨proof⟩

lemma trivial-formula-relevant:

assumes $\bigwedge \alpha. \text{range } \alpha \subseteq \mathcal{T}_G \ \mathcal{F} \implies \neg \text{eval-formula } \mathcal{F} \ \text{Rs } \alpha \ \text{fm}$
shows formula-relevant \mathcal{F} Rs vs fm
 ⟨proof⟩

lemma formula-relevant-0-FExists:

assumes formula-relevant \mathcal{F} Rs [0] fm
shows formula-relevant \mathcal{F} Rs [] (FExists fm)
 ⟨proof⟩

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 | $\alpha. \text{range } \alpha \subseteq \mathcal{T}_G \ \mathcal{F} \wedge \text{eval-formula } \mathcal{F} \ \text{Rs } \alpha \ \text{fm}$ }

lemma formula-spec-RRn-spec:

formula-spec \mathcal{F} Rs vs A fm \implies RRn-spec (length vs) A {map α vs | $\alpha. \text{range } \alpha \subseteq \mathcal{T}_G \ \mathcal{F} \wedge \text{eval-formula } \mathcal{F} \ \text{Rs } \alpha \ \text{fm}$ }

<proof>

lemma *formula-spec-nt-empty-form-sat*:

$\neg \text{reg-empty } A \implies \text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } A \text{ } fm \implies \exists \alpha. \text{range } \alpha \subseteq \mathcal{T}_G \mathcal{F} \wedge \text{eval-formula } \mathcal{F} \text{ } Rs \text{ } \alpha \text{ } fm$

<proof>

lemma *formula-spec-empty*:

$\text{reg-empty } A \implies \text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } A \text{ } fm \implies \text{range } \alpha \subseteq \mathcal{T}_G \mathcal{F} \implies \text{eval-formula } \mathcal{F} \text{ } Rs \text{ } \alpha \text{ } fm \longleftrightarrow \text{False}$

<proof>

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*:

$\text{sorted } vs \implies \text{distinct } vs \implies \text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } \text{empty-reg } \text{False}$

<proof>

lemma *true-formula-spec*:

assumes $vs \neq [] \vee \mathcal{T}_G (\text{fset } \mathcal{F}) \neq \{\}$ *sorted vs distinct vs*

shows $\text{formula-spec } (\text{fset } \mathcal{F}) \text{ } Rs \text{ } vs \text{ } (\text{true-RRn } \mathcal{F} \text{ } (\text{length } vs)) \text{ } \text{True}$

<proof>

lemma *relabel-formula-spec*:

$\text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } A \text{ } fm \implies \text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } (\text{relabel-reg } A) \text{ } fm$

<proof>

lemma *trim-formula-spec*:

$\text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } A \text{ } fm \implies \text{formula-spec } \mathcal{F} \text{ } Rs \text{ } vs \text{ } (\text{trim-reg } A) \text{ } fm$

<proof>

definition *fit-permute* :: $\text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{nat list} \Rightarrow \text{nat list}$ **where**

$\text{fit-permute } vs \text{ } vs' \text{ } vs'' = \text{map } (\lambda v. \text{if } v \in \text{set } vs \text{ then the } (\text{mem-idx } v \text{ } vs) \text{ else length } vs + \text{the } (\text{mem-idx } v \text{ } vs'')) \text{ } vs'$

definition *fit-rrn* :: $(\text{'f} \times \text{nat}) \text{ fset} \Rightarrow \text{nat list} \Rightarrow \text{nat list} \Rightarrow (\text{nat}, \text{'f option list}) \text{ reg} \Rightarrow (-, \text{'f option list}) \text{ reg}$ **where**

$\text{fit-rrn } \mathcal{F} \text{ } vs \text{ } vs' \text{ } A = (\text{let } vs'' = \text{subtract-list-sorted } vs' \text{ } vs \text{ in}$

$\text{fmap-funs-reg } (\lambda fs. \text{map } (!) \text{ } fs) \text{ } (\text{fit-permute } vs \text{ } vs' \text{ } vs''))$

$(\text{fmap-funs-reg } (\text{pad-with-Nones } (\text{length } vs) \text{ } (\text{length } vs'')) \text{ } (\text{pair-automaton-reg } A \text{ } (\text{true-RRn } \mathcal{F} \text{ } (\text{length } vs'')))))$

lemma *the-mem-idx-simp* [*simp*]:

$\text{distinct } xs \implies i < \text{length } xs \implies \text{the } (\text{mem-idx } (xs \text{ } ! \text{ } i) \text{ } xs) = i$

<proof>

lemma *fit-rrn*:

assumes *spec*: formula-spec (fset \mathcal{F}) R_s vs A *fm* **and** vs : sorted vs' distinct vs'
set $vs \subseteq$ *set* vs'
shows formula-spec (fset \mathcal{F}) R_s vs' (fit-rrn \mathcal{F} vs vs' A) *fm*
 ⟨*proof*⟩

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 $xs \implies$ sorted $ys \implies$ sorted (union-list-sorted xs ys)
 ⟨*proof*⟩

lemma *distinct-union-list-sortedI* [*simp*]:
 sorted $xs \implies$ sorted $ys \implies$ distinct $xs \implies$ distinct $ys \implies$ distinct (union-list-sorted
 xs ys)
 ⟨*proof*⟩

lemma *fit-rrns*:
assumes *infs*: $\bigwedge fvA. fvA \in$ *set* $rrns \implies$ formula-spec (fset \mathcal{F}) R_s (fst (snd fvA))
 (snd (snd fvA)) (fst fvA)
assumes (vs' , tas') = *fit-rrns* \mathcal{F} $rrns$
shows length $tas' =$ length $rrns \bigwedge i. i <$ length $rrns \implies$ formula-spec (fset \mathcal{F})
 R_s vs' ($tas' ! i$) (fst ($rrns ! i$))
 distinct vs' sorted vs'
 ⟨*proof*⟩

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 $fs \bigwedge i. i <$ length $fs \implies$ formula-spec \mathcal{F} R_s vs (tas
 ! i) ($fs ! i$)
and vs : sorted vs distinct vs
shows formula-spec \mathcal{F} R_s vs (*for-rrn* tas) (*FOr* fs)
 ⟨*proof*⟩

fun *fand-rrn* **where**
fand-rrn \mathcal{F} n [] = true-RRn \mathcal{F} n
 | *fand-rrn* \mathcal{F} n ($A \#$ tas) = fold (λ $A B$. simplify-reg (reg-intersect $A B$)) tas A

lemma *fand-rrn*:
assumes \mathcal{T}_G (fset \mathcal{F}) \neq {} length $tas =$ length $fs \bigwedge i. i <$ length $fs \implies$ for-
 mula-spec (fset \mathcal{F}) R_s 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*⟩

13.7.1 IExists inference rule

lemma *lift-fun-gpairD*:

map-gterm lift-fun s = gpair t u $\implies t = s$
map-gterm lift-fun s = gpair t u $\implies u = s$
 ⟨*proof*⟩

definition *upd-bruijn* :: *nat list* \Rightarrow *nat list* **where**

upd-bruijn vs = tl (map ($\lambda x. x - 1$) vs)

lemma *upd-bruijn-length[simp]*:

length (upd-bruijn vs) = length vs - 1
 ⟨*proof*⟩

lemma *pres-sorted-dec*:

sorted xs \implies *sorted (map ($\lambda x. x - \text{Suc } 0$) xs)*
 ⟨*proof*⟩

lemma *upd-bruijn-pres-sorted*:

sorted xs \implies *sorted (upd-bruijn xs)*
 ⟨*proof*⟩

lemma *pres-distinct-not-0-list-dec*:

distinct xs $\implies 0 \notin \text{set } xs \implies$ *distinct (map ($\lambda x. x - \text{Suc } 0$) xs)*
 ⟨*proof*⟩

lemma *upd-bruijn-pres-distinct*:

assumes *sorted xs distinct xs*
shows *distinct (upd-bruijn xs)*
 ⟨*proof*⟩

lemma *upd-bruijn-relevant-inv*:

assumes *sorted vs distinct vs* $0 \in \text{set } vs$
and $\bigwedge x. x \in \text{set } (\text{upd-bruijn } vs) \implies \alpha x = \alpha' x$
shows $\bigwedge x. x \in \text{set } vs \implies (\text{shift } \alpha \ 0 \ z) x = (\text{shift } \alpha' \ 0 \ z) x$
 ⟨*proof*⟩

lemma *ExistsI-upd-brujin-0*:

assumes *sorted vs distinct vs* $0 \in \text{set } vs$ *formula-relevant* \mathcal{F} *Rs vs fm*
shows *formula-relevant* \mathcal{F} *Rs (upd-bruijn vs) (FExists fm)*
 ⟨*proof*⟩

declare *subsetI[rule del]*

lemma *ExistsI-upd-brujin-no-0*:

assumes $0 \notin \text{set } vs$ **and** *formula-relevant* \mathcal{F} *Rs vs fm*
shows *formula-relevant* \mathcal{F} *Rs (map ($\lambda x. x - \text{Suc } 0$) vs) (FExists fm)*

<proof>

definition *shift-right where*

$shift\text{-}right\ \alpha \equiv \lambda\ i.\ \alpha\ (i + 1)$

lemma *shift-right-nt-0:*

$i \neq 0 \implies \alpha\ i = shift\text{-}right\ \alpha\ (i - Suc\ 0)$

<proof>

lemma *shift-shift-right-id [simp]:*

$shift\ (shift\text{-}right\ \alpha)\ 0\ (\alpha\ 0) = \alpha$

<proof>

lemma *shift-right-rangeI [intro]:*

$range\ \alpha \subseteq T \implies range\ (shift\text{-}right\ \alpha) \subseteq T$

<proof>

lemma *eval-formula-shift-right-eval:*

$eval\text{-}formula\ \mathcal{F}\ Rs\ \alpha\ fm \implies eval\text{-}formula\ \mathcal{F}\ Rs\ (shift\ (shift\text{-}right\ \alpha)\ 0\ (\alpha\ 0))\ fm$

$eval\text{-}formula\ \mathcal{F}\ Rs\ (shift\ (shift\text{-}right\ \alpha)\ 0\ (\alpha\ 0))\ fm \implies eval\text{-}formula\ \mathcal{F}\ Rs\ \alpha\ fm$

<proof>

declare *subsetI[intro!]*

lemma *nt-rel-0-trivial-shift:*

assumes $0 \notin set\ vs$

shows $\{map\ \alpha\ vs\ |\ \alpha.\ range\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge eval\text{-}formula\ \mathcal{F}\ Rs\ \alpha\ fm\} =$

$\{map\ (\lambda x.\ \alpha\ (x - Suc\ 0))\ vs\ |\ \alpha.\ range\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z \in \mathcal{T}_G\ \mathcal{F}.\ eval\text{-}formula\ \mathcal{F}\ Rs\ (\alpha\langle 0:z\rangle)\ fm)\}$

(**is** $?Ls = ?Rs$)

<proof>

lemma *relevant-vars-upd-bruijn-tl:*

assumes *sorted vs distinct vs*

shows $map\ (shift\text{-}right\ \alpha)\ (upd\text{-}bruijn\ vs) = tl\ (map\ \alpha\ vs)$ *<proof>*

lemma *drop-upd-bruijn-set:*

assumes *sorted vs distinct vs*

shows $drop\ 1\ ' \{map\ \alpha\ vs\ |\ \alpha.\ range\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge eval\text{-}formula\ \mathcal{F}\ Rs\ \alpha\ fm\} =$

$\{map\ \alpha\ (upd\text{-}bruijn\ vs)\ |\ \alpha.\ range\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z \in \mathcal{T}_G\ \mathcal{F}.\ eval\text{-}formula\ \mathcal{F}\ Rs\ (\alpha\langle 0:z\rangle)\ fm)\}$

(**is** $?Ls = ?Rs$)

<proof>

lemma *closed-sat-form-env-dom:*

assumes *formula-relevant $\mathcal{F}\ Rs\ []$ (FExists fm) range $\alpha \subseteq \mathcal{T}_G\ \mathcal{F}$ eval-formula $\mathcal{F}\ Rs\ \alpha\ fm$*

shows $\{[\alpha\ 0]\ |\ \alpha.\ range\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z \in \mathcal{T}_G\ \mathcal{F}.\ eval\text{-}formula\ \mathcal{F}\ Rs\ (\alpha\langle 0:z\rangle)\ fm)\} = \{[t]\ |\ t.\ t \in \mathcal{T}_G\ \mathcal{F}\}$

<proof>

lemma *find-append*:

find P (xs @ ys) = (if find P xs ≠ None then find P xs else find P ys)
<proof>

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 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 (λta. (FRR1 s x, [x], fmap-funs-reg (λf. [Some f]) ta)) (rr1c F Rs s)
 }
 | IRR2 r x y ⇒ do {
 guard (fm = FRR2 r x y);
 case compare x y of
 Lt ⇒ liftO1 (λta. (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  $\mathcal{F}$  (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  $\mathcal{F}$  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 \mathcal{F} (Suc 0)) {[t] | t. t ∈ \mathcal{T}_G (fset \mathcal{F})}

⟨proof⟩

lemma *check-inference-correct*:

assumes sig: \mathcal{T}_G (fset \mathcal{F}) ≠ {} **and** Rs: ∀ R ∈ set Rs. lv-trs (fset R) ∧ ffunas-trs R |⊆| \mathcal{F}

assumes infs: $\bigwedge fvA. fvA \in \text{set infs} \implies \text{formula-spec (fset } \mathcal{F}) (\text{map fset Rs}) (\text{fst (snd fvA)}) (\text{snd (snd fvA)}) (\text{fst fvA})$

assumes inf: check-inference rr1c rr2c \mathcal{F} Rs infs (l, step, fm, is) = Some (fm', vs, A')

assumes rr1: $\bigwedge r1. \forall ta1. rr1c \mathcal{F} Rs r1 = \text{Some } ta1 \implies RR1\text{-spec } ta1 (\text{eval-rr1-rel (fset } \mathcal{F}) (\text{map fset Rs}) r1)$

assumes rr2: $\bigwedge r2. \forall ta2. rr2c \mathcal{F} Rs r2 = \text{Some } ta2 \implies RR2\text{-spec } ta2 (\text{eval-rr2-rel (fset } \mathcal{F}) (\text{map fset Rs}) r2)$

shows l = length infs ∧ fm = fm' ∧ formula-spec (fset \mathcal{F}) (map fset Rs) vs A' fm'

⟨proof⟩

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* $\mathcal{A} = \text{eps-free-automata } (\text{eps } \mathcal{A}) \mathcal{A}$

abbreviation *ftrancl-eps-free-reg* $\mathcal{A} \equiv \text{Reg } (\text{fin } \mathcal{A}) (\text{ftrancl-eps-free-closures } (\text{ta } \mathcal{A}))$

lemma *ftrancl-eps-free-ta-derI*:

$(\text{eps } \mathcal{A})|^{+}| = \text{eps } \mathcal{A} \implies \text{ta-der } (\text{ftrancl-eps-free-closures } \mathcal{A}) (\text{term-of-gterm } t) =$
 $\text{ta-der } \mathcal{A} (\text{term-of-gterm } t)$
 ⟨*proof*⟩

lemma *L-ftrancl-eps-free-closuresI*:

$(\text{eps } (\text{ta } \mathcal{A}))|^{+}| = \text{eps } (\text{ta } \mathcal{A}) \implies \mathcal{L} (\text{ftrancl-eps-free-reg } \mathcal{A}) = \mathcal{L } \mathcal{A}$
 ⟨*proof*⟩

definition *root-step* $R \mathcal{F} \equiv (\text{let } (\text{TA1}, \text{TA2}) = \text{agtt-grrstep } R \mathcal{F} \text{ in } (\text{ftrancl-eps-free-closures } \text{TA1}, \text{TA2}))$

definition *AGTT-trancl-eps-free* $:: ('q, 'f) \text{ gtt} \Rightarrow ('q + 'q, 'f) \text{ gtt}$ **where**
AGTT-trancl-eps-free $\mathcal{G} = (\text{let } (\mathcal{A}, \mathcal{B}) = \text{AGTT-trancl } \mathcal{G} \text{ in } (\text{ftrancl-eps-free-closures } \mathcal{A}, \mathcal{B}))$

definition *GTT-trancl-eps-free* **where**

GTT-trancl-eps-free $\mathcal{G} = (\text{let } (\mathcal{A}, \mathcal{B}) = \text{GTT-trancl } \mathcal{G} \text{ in } (\text{ftrancl-eps-free-closures } \mathcal{A}, \text{ftrancl-eps-free-closures } \mathcal{B}))$

definition *AGTT-comp-eps-free* **where**

AGTT-comp-eps-free $\mathcal{G}_1 \mathcal{G}_2 = (\text{let } (\mathcal{A}, \mathcal{B}) = \text{AGTT-comp}' \mathcal{G}_1 \mathcal{G}_2 \text{ in } (\text{ftrancl-eps-free-closures } \mathcal{A}, \mathcal{B}))$

definition *GTT-comp-eps-free* **where**

GTT-comp-eps-free $\mathcal{G}_1 \mathcal{G}_2 = (\text{let } (\mathcal{A}, \mathcal{B}) = \text{GTT-comp}' \mathcal{G}_1 \mathcal{G}_2 \text{ in } (\text{ftrancl-eps-free-closures } \mathcal{A}, \text{ftrancl-eps-free-closures } \mathcal{B}))$

lemma *eps-free-relabel* [*simp*]:

$\text{is-gtt-eps-free } (\text{relabel-gtt } \mathcal{G}) = \text{is-gtt-eps-free } \mathcal{G}$
 ⟨*proof*⟩

lemma *eps-free-prod-swap*:

$\text{is-gtt-eps-free } (\mathcal{A}, \mathcal{B}) \implies \text{is-gtt-eps-free } (\mathcal{B}, \mathcal{A})$

<proof>

lemma *eps-free-root-step:*

is-gtt-eps-free (root-step R F)

<proof>

lemma *eps-free-AGTT-trancl-eps-free:*

is-gtt-eps-free G \implies is-gtt-eps-free (AGTT-trancl-eps-free G)

<proof>

lemma *eps-free-GTT-trancl-eps-free:*

is-gtt-eps-free G \implies is-gtt-eps-free (GTT-trancl-eps-free G)

<proof>

lemma *eps-free-AGTT-comp-eps-free:*

is-gtt-eps-free G₂ \implies is-gtt-eps-free (AGTT-comp-eps-free G₁ G₂)

<proof>

lemma *eps-free-GTT-comp-eps-free:*

is-gtt-eps-free (GTT-comp-eps-free G₁ G₂)

<proof>

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 $\bigwedge t. ta\text{-der } (fst \mathcal{A}) (term\text{-of-gterm } t) = ta\text{-der } (fst \mathcal{B}) (term\text{-of-gterm } t)$

and $\bigwedge t. ta\text{-der } (snd \mathcal{A}) (term\text{-of-gterm } t) = ta\text{-der } (snd \mathcal{B}) (term\text{-of-gterm } t)$

shows *agtt-lang A = agtt-lang B* *<proof>*

lemma *agtt-lang-root-step-conv:*

agtt-lang (root-step R F) = agtt-lang (agtt-grrstep R F)

<proof>

lemma *agtt-lang-AGTT-trancl-eps-free-conv:*

assumes *is-gtt-eps-free G*

shows *agtt-lang (AGTT-trancl-eps-free G) = agtt-lang (AGTT-trancl G)*

<proof>

lemma *agtt-lang-GTT-trancl-eps-free-conv:*

assumes *is-gtt-eps-free G*

shows *agtt-lang (GTT-trancl-eps-free G) = agtt-lang (GTT-trancl G)*

<proof>

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>

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>

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* (*A**Root* *is*) = *liftO1* ($\lambda R.$ *relabel-gtt* (*root-step* R \mathcal{F}))
(*is-to-trs'* *Rs* *is*)

| *gtt-of-gtt-rel-impl* \mathcal{F} *Rs* (*G**Inv* g) = *liftO1* *prod.swap* (*gtt-of-gtt-rel-impl* \mathcal{F} *Rs* g)
| *gtt-of-gtt-rel-impl* \mathcal{F} *Rs* (*A**Union* $g1$ $g2$) = *liftO2* ($\lambda g1$ $g2.$ *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* (*A**Trancl* 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* (*G**Trancl* g) = *liftO1* *GTT-trancl-eps-free* (*gtt-of-gtt-rel-impl*
 \mathcal{F} *Rs* g)

| *gtt-of-gtt-rel-impl* \mathcal{F} *Rs* (*A**Comp* $g1$ $g2$) = *liftO2* ($\lambda g1$ $g2.$ *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* (*G**Comp* $g1$ $g2$) = *liftO2* ($\lambda g1$ $g2.$ *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' ⇒ *is-gtt-eps-free* g'

<proof>

lemma *gtt-of-gtt-rel-impl-gtt-of-gtt-rel:*

gtt-of-gtt-rel-impl \mathcal{F} *Rs* g ≠ *None* ⇔ *gtt-of-gtt-rel* \mathcal{F} *Rs* g ≠ *None* (**is** *?Ls* ⇔ *?Rs*)

<proof>

lemma *gtt-of-gtt-rel-impl-sound:*

gtt-of-gtt-rel-impl \mathcal{F} *Rs* g = *Some* g' ⇒ *gtt-of-gtt-rel* \mathcal{F} *Rs* g = *Some* g'' ⇒
agtt-lang g' = *agtt-lang* g''

<proof>

lemma *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>

lemma \mathcal{L} -eps-free-ctxt-closure-reg:

assumes $is\text{-}ta\text{-}eps\text{-}free (ta \mathcal{A})$

shows $\mathcal{L} (ftrancl\text{-}eps\text{-}free\text{-}reg (ctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A})) = \mathcal{L} (ctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A})$

$\langle proof \rangle$

lemma \mathcal{L} -eps-free-parallel-closure-reg:

assumes $is\text{-}ta\text{-}eps\text{-}free (ta \mathcal{A})$

shows $\mathcal{L} (ftrancl\text{-}eps\text{-}free\text{-}reg (parallel\text{-}closure\text{-}reg \mathcal{F} \mathcal{A})) = \mathcal{L} (parallel\text{-}closure\text{-}reg \mathcal{F} \mathcal{A})$

$\langle proof \rangle$

abbreviation $eps\text{-}free\text{-}reg' S R \equiv Reg (fin R) (eps\text{-}free\text{-}automata S (ta R))$

definition $eps\text{-}free\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A} =$

$(let \mathcal{B} = mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A} in$

$eps\text{-}free\text{-}reg' ((\lambda p. (fst p, Inr cl\text{-}state)) \upharpoonright^{\dagger} (eps (ta \mathcal{B})) \cup eps (ta \mathcal{B})) \mathcal{B})$

definition $eps\text{-}free\text{-}nhole\text{-}mctxt\text{-}reflcl\text{-}reg \mathcal{F} \mathcal{A} =$

$(let \mathcal{B} = nhole\text{-}mctxt\text{-}reflcl\text{-}reg \mathcal{F} \mathcal{A} in$

$eps\text{-}free\text{-}reg' ((\lambda p. (fst p, Inl (Inr cl\text{-}state)))) \upharpoonright^{\dagger} (eps (ta \mathcal{B})) \cup eps (ta \mathcal{B})) \mathcal{B})$

definition $eps\text{-}free\text{-}nhole\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A} =$

$(let \mathcal{B} = nhole\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A} in$

$eps\text{-}free\text{-}reg' ((\lambda p. (fst p, (Inr cl\text{-}state)))) \upharpoonright^{\dagger} (eps (ta \mathcal{B})) \cup eps (ta \mathcal{B})) \mathcal{B})$

lemma \mathcal{L} -eps-free-reg'I:

$(eps (ta \mathcal{A})) \upharpoonright^{\dagger} = S \implies \mathcal{L} (eps\text{-}free\text{-}reg' S \mathcal{A}) = \mathcal{L} \mathcal{A}$

$\langle proof \rangle$

lemma \mathcal{L} -eps-free-mctxt-closure-reg:

assumes $is\text{-}ta\text{-}eps\text{-}free (ta \mathcal{A})$

shows $\mathcal{L} (eps\text{-}free\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}) = \mathcal{L} (mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}) \langle proof \rangle$

lemma \mathcal{L} -eps-free-nhole-mctxt-reflcl-reg:

assumes $is\text{-}ta\text{-}eps\text{-}free (ta \mathcal{A})$

shows $\mathcal{L} (eps\text{-}free\text{-}nhole\text{-}mctxt\text{-}reflcl\text{-}reg \mathcal{F} \mathcal{A}) = \mathcal{L} (nhole\text{-}mctxt\text{-}reflcl\text{-}reg \mathcal{F} \mathcal{A}) \langle proof \rangle$

lemma \mathcal{L} -eps-free-nhole-mctxt-closure-reg:

assumes $is\text{-}ta\text{-}eps\text{-}free (ta \mathcal{A})$

shows $\mathcal{L} (eps\text{-}free\text{-}nhole\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}) = \mathcal{L} (nhole\text{-}mctxt\text{-}closure\text{-}reg \mathcal{F} \mathcal{A}) \langle proof \rangle$

fun $rr1\text{-}of\text{-}rr1\text{-}rel\text{-}impl :: ('f \times nat) fset \Rightarrow ('f :: linorder, 'v) fin\text{-}trs list \Rightarrow ftrs rr1\text{-}rel \Rightarrow (nat, 'f) reg\ option$

and $rr2\text{-}of\text{-}rr2\text{-}rel\text{-}impl :: ('f \times nat) fset \Rightarrow ('f, 'v) fin\text{-}trs list \Rightarrow ftrs rr2\text{-}rel \Rightarrow (nat, 'f option \times 'f option) reg\ option$ **where**

$rr1\text{-}of\text{-}rr1\text{-}rel\text{-}impl \mathcal{F} Rs R1Terms = Some (relabel\text{-}reg (term\text{-}reg \mathcal{F}))$

$| rr1\text{-}of\text{-}rr1\text{-}rel\text{-}impl \mathcal{F} Rs (R1NF is) = liftO1 (\lambda R. (simplify\text{-}reg (nf\text{-}reg (fst \upharpoonright^{\dagger} R))$

$\mathcal{F}))$ (*is-to-trs'* R_s *is*)
 $|$ *rr1-of-rr1-rel-impl* \mathcal{F} R_s (*R1Inf* r) = *liftO1* ($\lambda R.$
 \quad *let* \mathcal{A} = *trim-reg* R *in*
 \quad *simplify-reg* (*proj-1-reg* (*Inf-reg-impl* \mathcal{A}))
 \quad) (*rr2-of-rr2-rel-impl* \mathcal{F} R_s r)
 $|$ *rr1-of-rr1-rel-impl* \mathcal{F} R_s (*R1Proj* i r) = (*case* i *of* $0 \Rightarrow$
 \quad *liftO1* (*trim-reg* \circ *proj-1-reg*) (*rr2-of-rr2-rel-impl* \mathcal{F} R_s r)
 \quad $|$ $- \Rightarrow$ *liftO1* (*trim-reg* \circ *proj-2-reg*) (*rr2-of-rr2-rel-impl* \mathcal{F} R_s r)
 $|$ *rr1-of-rr1-rel-impl* \mathcal{F} R_s (*R1Union* s_1 s_2) =
 \quad *liftO2* ($\lambda x y.$ *relabel-reg* (*reg-union* $x y$)) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s s_1) (*rr1-of-rr1-rel-impl*
 \mathcal{F} R_s s_2)
 $|$ *rr1-of-rr1-rel-impl* \mathcal{F} R_s (*R1Inter* s_1 s_2) =
 \quad *liftO2* ($\lambda x y.$ *simplify-reg* (*reg-intersect* $x y$)) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s s_1)
(*rr1-of-rr1-rel-impl* \mathcal{F} R_s s_2)
 $|$ *rr1-of-rr1-rel-impl* \mathcal{F} R_s (*R1Diff* s_1 s_2) = *liftO2* ($\lambda x y.$ *relabel-reg* (*trim-reg*
(*difference-reg* $x y$)) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s s_1) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s s_2)

 $|$ *rr2-of-rr2-rel-impl* \mathcal{F} R_s (*R2GTT-Rel* g w x) =
 \quad (*case* w *of* *PRoot* \Rightarrow
 \quad (*case* x *of* *ESingle* \Rightarrow *liftO1* (*simplify-reg* \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl*
 \mathcal{F} R_s g)
 \quad $|$ *EParallel* \Rightarrow *liftO1* (*simplify-reg* \circ *reflcl-reg* (*lift-sig-RR2* $| \uparrow \mathcal{F}$) \circ
GTT-to-RR2-root-reg) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g)
 \quad $|$ *EStrictParallel* \Rightarrow *liftO1* (*simplify-reg* \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl*
 \mathcal{F} R_s g))
 \quad $|$ *PNonRoot* \Rightarrow
 \quad (*case* x *of* *ESingle* \Rightarrow *liftO1* (*simplify-reg* \circ *ftrancl-eps-free-reg* \circ *nhole-ctxt-closure-reg*
(*lift-sig-RR2* $| \uparrow \mathcal{F}$) \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g)
 \quad $|$ *EParallel* \Rightarrow *liftO1* (*simplify-reg* \circ *eps-free-nhole-mctxt-reflcl-reg* (*lift-sig-RR2*
 $| \uparrow \mathcal{F}$) \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g)
 \quad $|$ *EStrictParallel* \Rightarrow *liftO1* (*simplify-reg* \circ *eps-free-nhole-mctxt-closure-reg*
(*lift-sig-RR2* $| \uparrow \mathcal{F}$) \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g))
 \quad $|$ *PAny* \Rightarrow
 \quad (*case* x *of* *ESingle* \Rightarrow *liftO1* (*simplify-reg* \circ *ftrancl-eps-free-reg* \circ *ctxt-closure-reg*
(*lift-sig-RR2* $| \uparrow \mathcal{F}$) \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g)
 \quad $|$ *EParallel* \Rightarrow *liftO1* (*simplify-reg* \circ *ftrancl-eps-free-reg* \circ *parallel-closure-reg*
(*lift-sig-RR2* $| \uparrow \mathcal{F}$) \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g)
 \quad $|$ *EStrictParallel* \Rightarrow *liftO1* (*simplify-reg* \circ *eps-free-mctxt-closure-reg* (*lift-sig-RR2*
 $| \uparrow \mathcal{F}$) \circ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel-impl* \mathcal{F} R_s g)))
 $|$ *rr2-of-rr2-rel-impl* \mathcal{F} R_s (*R2Diag* s) =
 \quad *liftO1* ($\lambda x.$ *fmap-funs-reg* ($\lambda f.$ (*Some* f , *Some* f)) x) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s
 s)
 $|$ *rr2-of-rr2-rel-impl* \mathcal{F} R_s (*R2Prod* s_1 s_2) =
 \quad *liftO2* ($\lambda x y.$ *simplify-reg* (*pair-automaton-reg* $x y$)) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s
 s_1) (*rr1-of-rr1-rel-impl* \mathcal{F} R_s s_2)
 $|$ *rr2-of-rr2-rel-impl* \mathcal{F} R_s (*R2Inv* r) = *liftO1* (*fmap-funs-reg* *prod.swap*) (*rr2-of-rr2-rel-impl*
 \mathcal{F} R_s r)
 $|$ *rr2-of-rr2-rel-impl* \mathcal{F} R_s (*R2Union* r_1 r_2) =
 \quad *liftO2* ($\lambda x y.$ *relabel-reg* (*reg-union* $x y$)) (*rr2-of-rr2-rel-impl* \mathcal{F} R_s r_1) (*rr2-of-rr2-rel-impl*

$\mathcal{F} R s r2)$
 $| rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s (R2Inter\ r1\ r2) =$
 $\quad liftO2\ (\lambda\ x\ y.\ simplify\text{-reg}\ (reg\text{-intersect}\ x\ y))\ (rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s\ r1)$
 $(rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s\ r2)$
 $| rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s (R2Diff\ r1\ r2) = liftO2\ (\lambda\ x\ y.\ simplify\text{-reg}\ (difference\text{-reg}$
 $x\ y))\ (rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s\ r1)\ (rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s\ r2)$
 $| rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s (R2Comp\ r1\ r2) = liftO2\ (\lambda\ x\ y.\ simplify\text{-reg}\ (rr2\text{-compositon}$
 $\mathcal{F}\ x\ y))$
 $\quad (rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s\ r1)\ (rr2\text{-of-}rr2\text{-rel-impl } \mathcal{F} R s\ r2)$

lemmas $ta\text{-simp-unfold} = simplify\text{-reg-def}\ relabel\text{-reg-def}\ trim\text{-reg-def}\ relabel\text{-ta-def}\ term\text{-reg-def}$

lemma $is\text{-ta-eps-free-trim-reg}$ [intro!]:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (trim\text{-reg}\ R))$
 $\langle proof \rangle$

lemma $is\text{-ta-eps-free-relabel-reg}$ [intro!]:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (relabel\text{-reg}\ R))$
 $\langle proof \rangle$

lemma $is\text{-ta-eps-free-simplify-reg}$ [intro!]:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (simplify\text{-reg}\ R))$
 $\langle proof \rangle$

lemma $is\text{-ta-emptyI}$ [simp]:

$is\text{-ta-eps-free}\ (TA\ R\ \{\|\}) \longleftrightarrow True$
 $\langle proof \rangle$

lemma $is\text{-ta-empty-trim-reg}$:

$is\text{-ta-eps-free}\ (ta\ A) \implies eps\ (ta\ (trim\text{-reg}\ A)) = \{\|\}$
 $\langle proof \rangle$

lemma $is\text{-proj-ta-eps-empty}$:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (proj\text{-1-reg}\ R))$
 $is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (proj\text{-2-reg}\ R))$
 $\langle proof \rangle$

lemma $is\text{-pod-ta-eps-empty}$:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ L) \implies is\text{-ta-eps-free}\ (ta\ (reg\text{-intersect}$
 $R\ L))$
 $\langle proof \rangle$

lemma $is\text{-fmap-funs-reg-eps-empty}$:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (fmap\text{-funs-reg}\ f\ R))$
 $\langle proof \rangle$

lemma $is\text{-collapse-automaton-reg-eps-empty}$:

$is\text{-ta-eps-free}\ (ta\ R) \implies is\text{-ta-eps-free}\ (ta\ (collapse\text{-automaton-reg}\ R))$
 $\langle proof \rangle$

lemma *is-pair-automaton-reg-eps-empty:*

$is-ta-eps-free (ta R) \implies is-ta-eps-free (ta L) \implies is-ta-eps-free (ta (pair-automaton-reg R L))$
<proof>

lemma *is-reflcl-automaton-eps-free:*

$is-ta-eps-free A \implies is-ta-eps-free (reflcl-automaton (lift-sig-RR2 \mid \cdot \mid \mathcal{F}) A)$
<proof>

lemma *is-GTT-to-RR2-root-eps-empty:*

$is-gtt-eps-free \mathcal{G} \implies is-ta-eps-free (GTT-to-RR2-root \mathcal{G})$
<proof>

lemma *is-term-automata-eps-empty:*

$is-ta-eps-free (ta (term-reg \mathcal{F})) \longleftrightarrow True$
<proof>

lemma *is-ta-eps-free-eps-free-automata [simp]:*

$is-ta-eps-free (eps-free-automata S R) \longleftrightarrow True$
<proof>

lemma *rr2-of-rr2-rel-impl-eps-free:*

shows $\forall A. rr1-of-rr1-rel-impl \mathcal{F} Rs r1 = Some A \longrightarrow is-ta-eps-free (ta A)$
 $\forall A. rr2-of-rr2-rel-impl \mathcal{F} Rs r2 = Some A \longrightarrow is-ta-eps-free (ta A)$
<proof>

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>

lemma *Q-fmap-funs-reg [simp]:*

$Q_r (fmap-funs-reg f \mathcal{A}) = Q_r \mathcal{A}$
<proof>

lemma *ta-reachable-fmap-funs-reg [simp]:*

$ta-reachable (ta (fmap-funs-reg f \mathcal{A})) = ta-reachable (ta \mathcal{A})$
<proof>

lemma *collapse-reg-cong:*

$Q_r \mathcal{A} \mid \subseteq \mid ta-reachable (ta \mathcal{A}) \implies Q_r \mathcal{B} \mid \subseteq \mid ta-reachable (ta \mathcal{B}) \implies \mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B}$
 $\implies \mathcal{L} (collapse-automaton-reg \mathcal{A}) = \mathcal{L} (collapse-automaton-reg \mathcal{B})$
<proof>

lemma *L-fmap-funs-reg-cong:*

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{L} (fmap-funs-reg h \mathcal{A}) = \mathcal{L} (fmap-funs-reg h \mathcal{B})$
<proof>

lemma \mathcal{L} -pair-automaton-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{L} \mathcal{C} = \mathcal{L} \mathcal{D} \implies \mathcal{L} (\text{pair-automaton-reg } \mathcal{A} \mathcal{C}) = \mathcal{L} (\text{pair-automaton-reg } \mathcal{B} \mathcal{D})$
(proof)

lemma \mathcal{L} -nhole-ctxt-closure-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{F} = \mathcal{G} \implies \mathcal{L} (\text{nhole-ctxt-closure-reg } \mathcal{F} \mathcal{A}) = \mathcal{L} (\text{nhole-ctxt-closure-reg } \mathcal{G} \mathcal{B})$
(proof)

lemma \mathcal{L} -nhole-mctxt-closure-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{F} = \mathcal{G} \implies \mathcal{L} (\text{nhole-mctxt-closure-reg } \mathcal{F} \mathcal{A}) = \mathcal{L} (\text{nhole-mctxt-closure-reg } \mathcal{G} \mathcal{B})$
(proof)

lemma \mathcal{L} -ctxt-closure-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{F} = \mathcal{G} \implies \mathcal{L} (\text{ctxt-closure-reg } \mathcal{F} \mathcal{A}) = \mathcal{L} (\text{ctxt-closure-reg } \mathcal{G} \mathcal{B})$
(proof)

lemma \mathcal{L} -parallel-closure-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{F} = \mathcal{G} \implies \mathcal{L} (\text{parallel-closure-reg } \mathcal{F} \mathcal{A}) = \mathcal{L} (\text{parallel-closure-reg } \mathcal{G} \mathcal{B})$
(proof)

lemma \mathcal{L} -mctxt-closure-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{F} = \mathcal{G} \implies \mathcal{L} (\text{mctxt-closure-reg } \mathcal{F} \mathcal{A}) = \mathcal{L} (\text{mctxt-closure-reg } \mathcal{G} \mathcal{B})$
(proof)

lemma \mathcal{L} -nhole-mctxt-reflcl-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{F} = \mathcal{G} \implies \mathcal{L} (\text{nhole-mctxt-reflcl-reg } \mathcal{F} \mathcal{A}) = \mathcal{L} (\text{nhole-mctxt-reflcl-reg } \mathcal{G} \mathcal{B})$
(proof)

declare equalityI[rule del]

declare fsubsetI[rule del]

lemma \mathcal{L} -proj-1-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{L} (\text{proj-1-reg } \mathcal{A}) = \mathcal{L} (\text{proj-1-reg } \mathcal{B})$
(proof)

lemma \mathcal{L} -proj-2-reg-cong:

$\mathcal{L} \mathcal{A} = \mathcal{L} \mathcal{B} \implies \mathcal{L} (\text{proj-2-reg } \mathcal{A}) = \mathcal{L} (\text{proj-2-reg } \mathcal{B})$
(proof)

lemma rr2-of-rr2-rel-impl-sound:

assumes $\forall R \in \text{set } \mathcal{R}s. \text{lv-trs } (\text{fset } R) \wedge \text{ffunas-trs } R \mid \subseteq \mathcal{F}$

shows $\bigwedge A B. \text{rr1-of-rr1-rel-impl } \mathcal{F} \mathcal{R}s \ r1 = \text{Some } A \implies \text{rr1-of-rr1-rel } \mathcal{F} \mathcal{R}s \ r1 = \text{Some } B \implies \mathcal{L} A = \mathcal{L} B$

$\bigwedge A B. \text{rr2-of-rr2-rel-impl } \mathcal{F} \mathcal{R}s \ r2 = \text{Some } A \implies \text{rr2-of-rr2-rel } \mathcal{F} \mathcal{R}s \ r2 =$

Some B $\implies \mathcal{L} A = \mathcal{L} B$
 <proof>

declare *equalityI*[intro!]
declare *fsubsetI*[intro!]

lemma *rr12-of-rr12-rel-impl-correct*:

assumes $\forall R \in \text{set } Rs. \text{lv-trs } (fset R) \wedge \text{ffunas-trs } R \mid\subseteq \mathcal{F}$
shows $\forall ta1. \text{rr1-of-rr1-rel-impl } \mathcal{F} Rs r1 = \text{Some } ta1 \implies \text{RR1-spec } ta1 (\text{eval-rr1-rel } (fset \mathcal{F}) (\text{map } fset Rs) r1)$
 $\forall ta2. \text{rr2-of-rr2-rel-impl } \mathcal{F} Rs r2 = \text{Some } ta2 \implies \text{RR2-spec } ta2 (\text{eval-rr2-rel } (fset \mathcal{F}) (\text{map } fset Rs) r2)$
 <proof>

lemma *check-inference-rrn-impl-correct*:

assumes *sig*: $\mathcal{T}_G (fset \mathcal{F}) \neq \{\}$ **and** *Rs*: $\forall R \in \text{set } Rs. \text{lv-trs } (fset R) \wedge \text{ffunas-trs } R \mid\subseteq \mathcal{F}$
assumes *infs*: $\bigwedge fvA. fvA \in \text{set } infs \implies \text{formula-spec } (fset \mathcal{F}) (\text{map } fset Rs) (\text{fst } (snd fvA)) (\text{snd } (snd fvA)) (\text{fst } fvA)$
assumes *inf*: *check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl* $\mathcal{F} Rs infs (l, \text{step}, fm, is) = \text{Some } (fm', vs, A')$
shows $l = \text{length } infs \wedge fm = fm' \wedge \text{formula-spec } (fset \mathcal{F}) (\text{map } fset Rs) vs A' fm'$
 <proof>

definition *check-sig-nempty where*

check-sig-nempty $\mathcal{F} = (0 \mid\subseteq \text{snd } \mathcal{F})$

definition *check-trss where*

check-trss $\mathcal{R} \mathcal{F} = \text{list-all } (\lambda R. \text{lv-trs } (fset R) \wedge \text{funas-trs } (fset R) \subseteq fset \mathcal{F}) \mathcal{R}$

lemma *check-sig-nempty*:

check-sig-nempty $\mathcal{F} \longleftrightarrow \mathcal{T}_G (fset \mathcal{F}) \neq \{\}$ (**is** $?Ls \longleftrightarrow ?Rs$)
 <proof>

lemma *check-trss*:

check-trss $\mathcal{R} \mathcal{F} \longleftrightarrow (\forall R \in \text{set } \mathcal{R}. \text{lv-trs } (fset R) \wedge \text{ffunas-trs } R \mid\subseteq \mathcal{F})$
 <proof>

fun *check-inference-list* :: $(f \times \text{nat}) fset \Rightarrow (f :: \{\text{compare}, \text{linorder}\}, 'v) \text{fin-trs list}$

$\Rightarrow (\text{nat} \times \text{ftrs inference} \times \text{ftrs formula} \times \text{info list}) \text{list}$

$\Rightarrow (\text{ftrs formula} \times \text{nat list} \times (\text{nat}, 'f \text{option list}) \text{reg}) \text{list option where}$

check-inference-list $\mathcal{F} Rs infs = \text{do } \{$

guard (*check-sig-nempty* \mathcal{F});

guard (*check-trss* $Rs \mathcal{F}$);

foldl $(\lambda \text{tas } inf. \text{do } \{$

tas' $\leftarrow \text{tas};$

r $\leftarrow \text{check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl } \mathcal{F} Rs \text{tas}' inf;$

Some (*tas'* @ [*r*])

```

    })
  (Some []) infs
}

```

lemma *check-inference-list-correct*:

```

assumes check-inference-list  $\mathcal{F}$   $R_s$   $infs = \text{Some } fvAs$ 
shows  $\text{length } infs = \text{length } fvAs \wedge (\forall i < \text{length } fvAs. \text{fst } (snd (snd (infs ! i)))$ 
 $= \text{fst } (fvAs ! i)) \wedge$ 
 $(\forall i < \text{length } fvAs. \text{formula-spec } (fset \mathcal{F}) (map fset R_s) (\text{fst } (snd (fvAs ! i)))$ 
 $(snd (snd (fvAs ! i))) (\text{fst } (fvAs ! i)))$ 
 $\langle \text{proof} \rangle$ 

```

fun *check-certificate where*

```

check-certificate  $\mathcal{F}$   $R_s$   $A$   $fm$  (Certificate  $infs$   $claim$   $n$ ) = do {
  guard ( $n < \text{length } infs$ );
  guard ( $A \longleftrightarrow \text{claim} = \text{Nonempty}$ );
  guard ( $fm = \text{fst } (snd (snd (infs ! n)))$ );
   $fvA \leftarrow \text{check-inference-list } \mathcal{F} R_s (\text{take } (Suc\ n) infs)$ ;
  (let  $E = \text{reg-empty } (snd (snd (last\ fvA)))$ ) in
  case  $claim$  of Empty  $\Rightarrow$  Some  $E$ 
    | -  $\Rightarrow$  Some ( $\neg E$ )
}

```

definition *formula-unsatisfiable where*

```

formula-unsatisfiable  $\mathcal{F}$   $R_s$   $fm \longleftrightarrow (\text{formula-satisfiable } \mathcal{F} R_s fm = \text{False})$ 

```

definition *correct-certificate where*

```

correct-certificate  $\mathcal{F}$   $R_s$   $claim$   $infs$   $n \equiv$ 
 $(\text{claim} = \text{Empty} \longleftrightarrow (\text{formula-unsatisfiable } (fset \mathcal{F}) (map fset R_s) (\text{fst } (snd$ 
 $(snd (infs ! n)))))) \wedge$ 
 $\text{claim} = \text{Nonempty} \longleftrightarrow \text{formula-satisfiable } (fset \mathcal{F}) (map fset R_s) (\text{fst } (snd$ 
 $(snd (infs ! n))))$ 

```

lemma *check-certificate-sound*:

```

assumes check-certificate  $\mathcal{F}$   $R_s$   $A$   $fm$  (Certificate  $infs$   $claim$   $n$ ) = Some  $B$ 
shows  $fm = \text{fst } (snd (snd (infs ! n))) \wedge A \longleftrightarrow \text{claim} = \text{Nonempty}$ 
 $\langle \text{proof} \rangle$ 

```

lemma *check-certificate-correct*:

```

assumes check-certificate  $\mathcal{F}$   $R_s$   $A$   $fm$  (Certificate  $infs$   $claim$   $n$ ) = Some  $B$ 
shows ( $B = \text{True} \longrightarrow \text{correct-certificate } \mathcal{F} R_s \text{ claim } infs\ n$ )  $\wedge$ 
 $(B = \text{False} \longrightarrow \text{correct-certificate } \mathcal{F} R_s (\text{case-claim } \text{Nonempty } \text{Empty } \text{claim})$ 
 $infs\ n)$ 
 $\langle \text{proof} \rangle$ 

```

definition *check-certificate-string ::*

```

(integer list  $\times$  fvar) fset  $\Rightarrow$ 
((integer list, integer list) Term.term  $\times$  (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. Hrițcu 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.