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

Alexander Lochmann   Bertram Felgenhauer

May 26, 2024

Abstract

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

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

Contents

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

1 Introduction

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

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

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

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

1.1 Misc

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

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

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

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

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

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

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

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

lemma *sorted-append-bigger*:
sorted xs ⟹ ∀ *x* ∈ *set xs*. *x* ≤ *y* ⟹ *sorted* (*xs* @ [*y*])
⟨*proof*⟩

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 $\mathcal{R}$ $\Longrightarrow$ *finite* (*funas-trs* $\mathcal{R}$)
⟨*proof*⟩

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

lemma *finite-subterms-fun*: *finite* (*subterms s*)
⟨*proof*⟩

lemma *subterms-supteq-conv*: $t \in$ *subterms* $s \longleftrightarrow s \unrhd t$
⟨*proof*⟩

lemma *set-all-subteq-subterms*:
subterms $s = \{t.\ s \unrhd t\}$
⟨*proof*⟩

lemma *finite-subterms*: *finite* $\{t.\ s \unrhd t\}$
⟨*proof*⟩

lemma *finite-strict-subterms*: *finite* $\{t.\ s \rhd t\}$
⟨*proof*⟩

lemma *finite-UN-I2*:
finite A $\Longrightarrow$ ($\forall$ $B \in A$. *finite B*) $\Longrightarrow$ *finite* (⋃ *A*)
⟨*proof*⟩

lemma *root-substerms-funas-term*:
the ‘ (*root* ‘ (*subterms s*) − {*None*}) = *funas-term s* (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *root-substerms-funas-term-set*:
the ‘ (*root* ‘⋃ (*subterms* ‘ *R*) − {*None*}) = ⋃ (*funas-term* ‘ *R*)
⟨*proof*⟩

lemma *subst-merge*:
assumes *part*: *is-partition* (*map vars-term ts*)
shows $\exists\sigma$. $\forall$ *i*<*length ts*. $\forall$ *x*∈*vars-term* (*ts* ! *i*). $\sigma\ x = \tau\ i\ x$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *choice-nat*:
assumes $\forall\, i{<}n.\ \exists\, x.\ P\ x\ i$
shows $\exists f.\ \forall\, x{<}n.\ P\ (f\ x)\ x$ ⟨*proof*⟩

lemma *subseteq-set-conv-nth*:
$(\forall\ \ i < \mathit{length}\ \mathit{ss}.\ \mathit{ss}\ !\ i \in T) \longleftrightarrow \mathit{set}\ \mathit{ss} \subseteq T$
⟨*proof*⟩

lemma *singelton-trancl* [*simp*]: $\{a\}^+ = \{a\}$
⟨*proof*⟩

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

lemma *set-take-nth*:
assumes $x \in \mathit{set}\ (\mathit{take}\ i\ \mathit{xs})$
shows $\exists\ \ j < \mathit{length}\ \mathit{xs}.\ j < i \wedge \mathit{xs}\ !\ j = x$ ⟨*proof*⟩

lemma *nth-sum-listI*:
assumes $\mathit{length}\ \mathit{xs} = \mathit{length}\ \mathit{ys}$
and $\forall\ \ i < \mathit{length}\ \mathit{xs}.\ \mathit{xs}\ !\ i = \mathit{ys}\ !\ i$
shows $\mathit{sum\text{-}list}\ \mathit{xs} = \mathit{sum\text{-}list}\ \mathit{ys}$
⟨*proof*⟩

lemma *concat-nth-length*:
$i < \mathit{length}\ \mathit{uss} \Longrightarrow j < \mathit{length}\ (\mathit{uss}\ !\ i) \Longrightarrow$
$\mathit{sum\text{-}list}\ (\mathit{map}\ \mathit{length}\ (\mathit{take}\ i\ \mathit{uss})) + j < \mathit{length}\ (\mathit{concat}\ \mathit{uss})$
⟨*proof*⟩

lemma *sum-list-1-E* [*elim*]:
assumes $\mathit{sum\text{-}list}\ \mathit{xs} = \mathit{Suc}\ \mathit{0}$
obtains i **where** $i < \mathit{length}\ \mathit{xs}\ \mathit{xs}\ !\ i = \mathit{Suc}\ \mathit{0}\ \forall\ \ j < \mathit{length}\ \mathit{xs}.\ j \neq i \longrightarrow \mathit{xs}\ !\ j$
$= \mathit{0}$
⟨*proof*⟩

lemma *nth-equalityE*:
$\mathit{xs} = \mathit{ys} \Longrightarrow (\mathit{length}\ \mathit{xs} = \mathit{length}\ \mathit{ys} \Longrightarrow (\bigwedge i.\ i < \mathit{length}\ \mathit{xs} \Longrightarrow \mathit{xs}\ !\ i = \mathit{ys}\ !\ i)$
$\Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemma *map-cons-presv-distinct*:
distinct t $\Longrightarrow$ *distinct* (*map* ((#) *i*) *t*)
⟨*proof*⟩

lemma *concat-nth-nthI*:
assumes *length ss* = *length ts* ∀ *i* < *length ts*. *length* (*ss* ! *i*) = *length* (*ts* ! *i*)
and ∀ *i* < *length ts*. ∀ *j* < *length* (*ts* ! *i*). *P* (*ss* ! *i* ! *j*) (*ts* ! *i* ! *j*)
shows ∀ *i* < *length* (*concat ts*). *P* (*concat ss* ! *i*) (*concat ts* ! *i*)
⟨*proof*⟩

lemma *last-nthI*:
assumes *i* < *length ts* ¬ *i* < *length ts* − *Suc 0*
shows *ts* ! *i* = *last ts* ⟨*proof*⟩

lemma *trancl-list-appendI* [*simp*, *intro*]:
(*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$ $\Longrightarrow$ (*x*, *y*) ∈ $\mathcal{R}$ $\Longrightarrow$ (*x* # *xs*, *y* # *ys*) ∈ *trancl-list* $\mathcal{R}$
⟨*proof*⟩

lemma *trancl-list-append-tranclI* [*intro*]:
(*x*, *y*) ∈ $\mathcal{R}^+$ $\Longrightarrow$ (*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$ $\Longrightarrow$ (*x* # *xs*, *y* # *ys*) ∈ *trancl-list* $\mathcal{R}$
⟨*proof*⟩

lemma *trancl-list-conv*:
(*xs*, *ys*) ∈ *trancl-list* $\mathcal{R}$ $\longleftrightarrow$ *length xs* = *length ys* ∧ (∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}^+$) (**is** *?Ls* $\longleftrightarrow$ *?Rs*)
⟨*proof*⟩

lemma *trancl-list-induct* [*consumes 2*, *case-names base step*]:
assumes *length ss* = *length ts* ∀ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) ∈ $\mathcal{R}^+$
and ⋀*xs ys*. *length xs* = *length ys* $\Longrightarrow$ ∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}$ $\Longrightarrow$ *P xs ys*
and ⋀*xs ys i z*. *length xs* = *length ys* $\Longrightarrow$ ∀ *i* < *length ys*. (*xs* ! *i*, *ys* ! *i*) ∈ $\mathcal{R}^+$ $\Longrightarrow$ *P xs ys*
$\Longrightarrow$ *i* < *length ys* $\Longrightarrow$ (*ys* ! *i*, *z*) ∈ $\mathcal{R}$ $\Longrightarrow$ *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*:

$R \subseteq X \times X \Longrightarrow Restr\ (R^+)\ X = R^+$
$R \subseteq X \times X \Longrightarrow Restr\ Id\ X\ O\ R = R$
$R \subseteq X \times X \Longrightarrow R\ O\ Restr\ Id\ X = R$
$R \subseteq X \times X \Longrightarrow S \subseteq X \times X \Longrightarrow Restr\ (R\ O\ S)\ X = R\ O\ S$
$R \subseteq X \times X \Longrightarrow R^+ \subseteq X \times X$
⟨*proof*⟩

lemma *Restr-tracl-comp-simps*:
$\mathcal{R} \subseteq X \times X \Longrightarrow \mathcal{L} \subseteq X \times X \Longrightarrow \mathcal{L}^+\ O\ \mathcal{R} \subseteq X \times X$
$\mathcal{R} \subseteq X \times X \Longrightarrow \mathcal{L} \subseteq X \times X \Longrightarrow \mathcal{L}\ O\ \mathcal{R}^+ \subseteq X \times X$
$\mathcal{R} \subseteq X \times X \Longrightarrow \mathcal{L} \subseteq X \times X \Longrightarrow \mathcal{L}^+\ O\ \mathcal{R}\ O\ \mathcal{L}^+ \subseteq X \times X$
⟨*proof*⟩

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

lemma *concat-index-split-mono-first-arg*:
$i < length\ (concat\ xs) \Longrightarrow o\text{-}idx \leq fst\ (concat\text{-}index\text{-}split\ (o\text{-}idx,\ i)\ xs)$
⟨*proof*⟩

lemma *concat-index-split-sound-fst-arg-aux*:
$i < length\ (concat\ xs) \Longrightarrow fst\ (concat\text{-}index\text{-}split\ (o\text{-}idx,\ i)\ xs) < length\ xs + o\text{-}idx$
⟨*proof*⟩

lemma *concat-index-split-sound-fst-arg*:
$i < length\ (concat\ xs) \Longrightarrow fst\ (concat\text{-}index\text{-}split\ (0,\ i)\ xs) < length\ xs$
⟨*proof*⟩

lemma *concat-index-split-sound-snd-arg-aux*:
assumes $i < length\ (concat\ xs)$
shows $snd\ (concat\text{-}index\text{-}split\ (n,\ i)\ xs) < length\ (xs\ !\ (fst\ (concat\text{-}index\text{-}split\ (n,\ i)\ xs) - n))$ ⟨*proof*⟩

lemma *concat-index-split-sound-snd-arg*:
assumes $i < length\ (concat\ xs)$
shows $snd\ (concat\text{-}index\text{-}split\ (0,\ i)\ xs) < length\ (xs\ !\ fst\ (concat\text{-}index\text{-}split\ (0,\ i)\ xs))$
⟨*proof*⟩

lemma *reconstr-1d-concat-index-split*:
assumes $i < length\ (concat\ xs)$
shows $i = (\lambda\ (m, j).\ sum\text{-}list\ (map\ length\ (take\ (m - n)\ xs)) + j)\ (concat\text{-}index\text{-}split\ (n,\ i)\ xs)$ ⟨*proof*⟩

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

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

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

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

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

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

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

lemma *sum-list-split* [*dest*!, *consumes 1*]:
assumes $sum\text{-}list\ (map\ length\ (take\ i\ xs)) + j = sum\text{-}list\ (map\ length\ (take\ k\ xs)) + l$
and $i < length\ xs\ k < length\ xs$
and $j < length\ (xs\ !\ i)\ l < length\ (xs\ !\ k)$
shows $i = k \wedge j = l$ $\langle proof \rangle$

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

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

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

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

shows *p* ∈ *poss t*
⟨*proof*⟩

lemma *vaposs-list-fun*:
assumes *p* ∈ *set* (*varposs-list* (*Fun f ts*))
obtains *i ps* **where** *i* < *length ts p* = *i* # *ps*
⟨*proof*⟩

lemma *varposs-list-distinct*:
distinct (*varposs-list t*)
⟨*proof*⟩

lemma *varposs-append*:
varposs (*Fun f* (*ts* @ [*t*])) = *varposs* (*Fun f ts*) ∪ ((#) (*length ts*)) ' *varposs t*
⟨*proof*⟩

lemma *varposs-eq-varposs-list*:
set (*varposs-list t*) = *varposs t*
⟨*proof*⟩

lemma *varposs-list-var-terms-length*:
length (*varposs-list t*) = *length* (*vars-term-list t*)
⟨*proof*⟩

lemma *vars-term-list-nth*:
assumes *i* < *length* (*vars-term-list* (*Fun f ts*))
and *concat-index-split* (*0*, *i*) (*map vars-term-list ts*) = (*k*, *j*)
shows *k* < *length ts* ∧ *j* < *length* (*vars-term-list* (*ts* ! *k*)) ∧
vars-term-list (*Fun f ts*) ! *i* = *map vars-term-list ts* ! *k* ! *j* ∧
i = *sum-list* (*map length* (*map vars-term-list* (*take k ts*))) + *j*
⟨*proof*⟩

lemma *varposs-list-nth*:
assumes *i* < *length* (*varposs-list* (*Fun f ts*))
and *concat-index-split* (*0*, *i*) (*poss-args varposs-list ts*) = (*k*, *j*)
shows *k* < *length ts* ∧ *j* < *length* (*varposs-list* (*ts* ! *k*)) ∧
varposs-list (*Fun f ts*) ! *i* = *k* # (*map varposs-list ts*) ! *k* ! *j* ∧
i = *sum-list* (*map length* (*map varposs-list* (*take k ts*))) + *j*
⟨*proof*⟩

lemma *varposs-list-to-var-term-list*:
assumes *i* < *length* (*varposs-list t*)
shows *the-Var* (*t* |- (*varposs-list t* ! *i*)) = (*vars-term-list t*) ! *i* ⟨*proof*⟩

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

lemma *sum-list-take-eq*:
fixes $\mathit{xs} :: \mathit{nat}\ \mathit{list}$
shows $k < i \Longrightarrow i < \mathit{length}\ \mathit{xs} \Longrightarrow \mathit{sum\text{-}list}\ (\mathit{take}\ i\ \mathit{xs}) =$
$\mathit{sum\text{-}list}\ (\mathit{take}\ k\ \mathit{xs}) + \mathit{xs}\ !\ k + \mathit{sum\text{-}list}\ (\mathit{take}\ (i - \mathit{Suc}\ k)\ (\mathit{drop}\ (\mathit{Suc}\ k)\ \mathit{xs}))$
⟨*proof*⟩

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

lemma *partition-by-map0-append* [*simp*]:
$\mathit{partition\text{-}by}\ \mathit{xs}\ (\mathit{map}\ (\lambda x.\ 0)\ \mathit{ys}\ @\ \mathit{zs}) = \mathit{replicate}\ (\mathit{length}\ \mathit{ys})\ []\ @\ \mathit{partition\text{-}by}\ \mathit{xs}\ \mathit{zs}$
⟨*proof*⟩

lemma *concat-partition-by* [*simp*]:
$\mathit{sum\text{-}list}\ \mathit{ys} = \mathit{length}\ \mathit{xs} \Longrightarrow \mathit{concat}\ (\mathit{partition\text{-}by}\ \mathit{xs}\ \mathit{ys}) = \mathit{xs}$
⟨*proof*⟩

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

lemma *partition-by-nth-nth-old*:
assumes $i < \mathit{length}\ (\mathit{partition\text{-}by}\ \mathit{xs}\ \mathit{ys})$
and $j < \mathit{length}\ (\mathit{partition\text{-}by}\ \mathit{xs}\ \mathit{ys}\ !\ i)$
and $\mathit{sum\text{-}list}\ \mathit{ys} = \mathit{length}\ \mathit{xs}$
shows $\mathit{partition\text{-}by}\ \mathit{xs}\ \mathit{ys}\ !\ i\ !\ j = \mathit{xs}\ !\ (\mathit{sum\text{-}list}\ (\mathit{map}\ \mathit{length}\ (\mathit{take}\ i\ (\mathit{partition\text{-}by}\ \mathit{xs}\ \mathit{ys}))) + j)$
⟨*proof*⟩

lemma *map-map-partition-by*:

map (*map f*) (*partition-by xs ys*) = *partition-by* (*map f xs*) *ys*
⟨*proof*⟩

lemma *length-partition-by* [*simp*]:
length (*partition-by xs ys*) = *length ys*
⟨*proof*⟩

lemma *partition-by-Nil* [*simp*]:
partition-by [] *ys* = *replicate* (*length ys*) []
⟨*proof*⟩

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

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

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

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

lemma *length-partition-by-nth*:
sum-list ys = *length xs* $\Longrightarrow i <$ *length ys* $\Longrightarrow$ *length* (*partition-by xs ys* ! *i*) = *ys* ! *i*
⟨*proof*⟩

lemma *partition-by-nth-nth-elem*:
assumes *sum-list ys* = *length xs* $i <$ *length ys* $j <$ *ys* ! *i*
shows *partition-by xs ys* ! *i* ! *j* $\in$ *set xs*
⟨*proof*⟩

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

$partition\text{-}by\text{-}idx\ (length\ xs)\ ys\ i\ j < length\ xs$
⟨proof⟩

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

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

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

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

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

lemma *UN-set-partition-by*:
$sum\text{-}list\ ys = length\ xs \Longrightarrow (\bigcup zs \in set\ (partition\text{-}by\ xs\ ys).\ \bigcup x \in set\ zs.\ f\ x) = (\bigcup x \in set\ xs.\ f\ x)$
⟨proof⟩

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

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

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

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

concat (*map* (λ*i* . *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 (λ*i*. *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* (λ*i*. *unfill-holes* (*Cs* ! *i*) (*ts* ! *i*)) [*0*..<*length ts*]) *else undefined*)

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

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

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

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

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

2.1.4 Semilattice Structures

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

begin

fun *inf-mctxt* :: ('*a*, '*b*) *mctxt* ⇒ ('*a*, '*b*) *mctxt* ⇒ ('*a*, '*b*) *mctxt*
where

MHole ⊓ *D* = *MHole* |
C ⊓ *MHole* = *MHole* |
MVar x ⊓ *MVar y* = (*if x* = *y then MVar x else MHole*) |
MFun f Cs ⊓ *MFun g Ds* =
(*if f* = *g* ∧ *length Cs* = *length Ds then MFun f* (*map* (*case-prod* (⊓)) (*zip Cs Ds*))
else MHole) |
C ⊓ *D* = *MHole*

instance ⟨*proof*⟩

end

lemma *inf-mctxt-idem* [*simp*]:
fixes *C* :: (′*f*, ′*v*) *mctxt*
shows *C* ⊓ *C* = *C*
⟨*proof*⟩

lemma *inf-mctxt-MHole2* [*simp*]:
C ⊓ *MHole* = *MHole*
⟨*proof*⟩

lemma *inf-mctxt-comm* [*ac-simps*]:
(*C* :: (′*f*, ′*v*) *mctxt*) ⊓ *D* = *D* ⊓ *C*
⟨*proof*⟩

lemma *inf-mctxt-assoc* [*ac-simps*]:
fixes *C* :: (′*f*, ′*v*) *mctxt*
shows *C* ⊓ *D* ⊓ *E* = *C* ⊓ (*D* ⊓ *E*)
⟨*proof*⟩

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

definition (*C* :: (′*a*, ′*b*) *mctxt*) ≤ *D* ⟷ *C* ⊓ *D* = *C*
definition (*C* :: (′*a*, ′*b*) *mctxt*) < *D* ⟷ *C* ≤ *D* ∧ ¬ *D* ≤ *C*

instance
⟨*proof*⟩

end

inductive *less-eq-mctxt′* :: (′*f*, ′*v*) *mctxt* ⇒ (′*f*,′*v*) *mctxt* ⇒ *bool* **where**
less-eq-mctxt′ MHole u
| *less-eq-mctxt′* (*MVar v*) (*MVar v*)
| *length cs* = *length ds* ⟹ (⋀*i*. *i* < *length cs* ⟹ *less-eq-mctxt′* (*cs* ! *i*) (*ds* ! *i*))
⟹ *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* = *length ?xs* ⟹ *concat* (*partition-by ?xs ?ys*) = *?xs* simplification rule.

lemma *fill-holes-induct2*[*consumes 2*, *case-names MHole MVar MFun*]:
fixes *P* :: (*'f*,*'v*) *mctxt* ⇒ *'a list* ⇒ *'b list* ⇒ *bool*
assumes *len1*: *num-holes C* = *length xs* **and** *len2*: *num-holes C* = *length ys*
and *Hole*: ⋀*x y*. *P MHole* [*x*] [*y*]
and *Var*: ⋀*v*. *P* (*MVar v*) [] []
and *Fun*: ⋀*f Cs xs ys*. *sum-list* (*map num-holes Cs*) = *length xs* ⟹
sum-list (*map num-holes Cs*) = *length ys* ⟹
(⋀*i*. *i* < *length Cs* ⟹ *P* (*Cs* ! *i*) (*partition-holes xs Cs* ! *i*) (*partition-holes ys Cs* ! *i*)) ⟹
P (*MFun f Cs*) (*concat* (*partition-holes xs Cs*)) (*concat* (*partition-holes ys Cs*))
shows *P C xs ys*
⟨*proof*⟩

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

lemma *fill-holes-term-of-mctxt*:
num-holes C = *0* ⟹ *fill-holes C* [] = *term-of-mctxt C*
⟨*proof*⟩

lemma *fill-holes-MHole*:
length ts = *Suc 0* ⟹ *ts* ! *0* = *u* ⟹ *fill-holes MHole ts* = *u*
⟨*proof*⟩

lemma *fill-holes-arbitrary*:
assumes *lCs*: *length Cs* = *length ts*
and *lss*: *length ss* = *length ts*
and *rec*: ⋀ *i*. *i* < *length ts* ⟹ *num-holes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧ *f* (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *map* (λ*i*. *f* (*Cs* ! *i*) (*partition-holes* (*concat ss*) *Cs* ! *i*)) [*0* ..< *length Cs*] = *ts*
⟨*proof*⟩

lemma *fill-holes-MFun*:
assumes *lCs*: *length Cs* = *length ts*
and *lss*: *length ss* = *length ts*
and *rec*: ⋀ *i*. *i* < *length ts* ⟹ *num-holes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧ *fill-holes* (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *fill-holes* (*MFun f Cs*) (*concat ss*) = *Fun f ts*
⟨*proof*⟩

lemma *eqfE*:
assumes $t =_f$ (*D*, *ss*) **shows** *t* = *fill-holes D ss num-holes D* = *length ss*
⟨*proof*⟩

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

lemma *eqf-MFunI*:
assumes *length sss* = *length Cs*
and *length ts* = *length Cs*
and⋀ *i*. *i* < *length Cs* ⟹ *ts* ! $i =_f$ (*Cs* ! *i*, *sss* ! *i*)
shows *Fun f ts* $=_f$ (*MFun f Cs*, *concat sss*)
⟨*proof*⟩

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*: $t =_f (C,ss) \Longrightarrow$ *ground t* = (*ground-mctxt C* ∧ (∀ *s* ∈ *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* ∧ (∀ *s* ∈ *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* ≤ *D* ⟷ *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* ≤ *MHole*
  obtains (*MHole*) *C* = *MHole*
  ⟨*proof*⟩

lemma *less-eq-mctxt-MVarE2*:
  assumes *C* ≤ *MVar v*
  obtains (*MHole*) *C* = *MHole* | (*MVar*) *C* = *MVar v*
  ⟨*proof*⟩

lemma *less-eq-mctxt-MFunE2*:

assumes $C \leq$ *MFun f ds*
obtains (*MHole*) C = *MHole*
| (*MFun*) *cs* **where** C = *MFun f cs length cs* = *length ds* $\bigwedge i.\ i <$ *length cs* $\Longrightarrow$ *cs* ! $i \leq$ *ds* ! i
⟨*proof*⟩

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

lemma *less-eq-mctxt-MVarE1*:
assumes *MVar v* $\leq D$
obtains (*MVar*) D = *MVar v*
⟨*proof*⟩

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

lemma *less-eq-mctxt-MFunE1*:
assumes *MFun f cs* $\leq D$
obtains (*MFun*) *ds* **where** D = *MFun f ds length cs* = *length ds* $\bigwedge i.\ i <$ *length cs* $\Longrightarrow$ *cs* ! $i \leq$ *ds* ! i
⟨*proof*⟩

lemma *length-unfill-holes* [*simp*]:
assumes $C \leq$ *mctxt-of-term t*
shows *length* (*unfill-holes C t*) = *num-holes C*
⟨*proof*⟩

lemma *map-vars-mctxt-id* [*simp*]:
map-vars-mctxt (λ *x. x*) $C = C$
⟨*proof*⟩

lemma *split-vars-eqf-subst-map-vars-term*:
$t \cdot \sigma =_f$ (*map-vars-mctxt vw* (*fst* (*split-vars t*)), *map* σ (*snd* (*split-vars t*)))
⟨*proof*⟩

lemma *split-vars-eqf-subst*: $t \cdot \sigma =_f$ (*fst* (*split-vars t*), (*map* σ (*snd* (*split-vars t*))))
⟨*proof*⟩

lemma *split-vars-fill-holes*:
assumes C = *fst* (*split-vars s*) **and** *ss* = *map Var* (*snd* (*split-vars s*))
shows *fill-holes C ss* = *s* ⟨*proof*⟩

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

⟨*proof*⟩

lemma *hole-poss-list-length*:
length (*hole-poss-list D*) = *num-holes D*
⟨*proof*⟩

lemma *unfill-holles-hole-poss-list-length*:
assumes $C \leq$ *mctxt-of-term t*
shows *length* (*unfill-holes C t*) = *length* (*hole-poss-list C*) ⟨*proof*⟩

lemma *unfill-holes-to-subst-at-hole-poss*:
assumes $C \leq$ *mctxt-of-term t*
shows *unfill-holes C t* = *map* ((|-) *t*) (*hole-poss-list C*) ⟨*proof*⟩

lemma *hole-poss-split-varposs-list-length* [*simp*]:
length (*hole-poss-list* (*fst* (*split-vars t*))) = *length* (*varposs-list t*)
⟨*proof*⟩

lemma *hole-poss-split-vars-varposs-list*:
hole-poss-list (*fst* (*split-vars t*)) = *varposs-list t*
⟨*proof*⟩

lemma *funas-term-fill-holes-iff*: *num-holes C* = *length ts* $\Longrightarrow$
g ∈ *funas-term* (*fill-holes C ts*) ⟷ *g* ∈ *funas-mctxt C* ∨ (∃ *t* ∈ *set ts*. *g* ∈ *funas-term t*)
⟨*proof*⟩

lemma *vars-term-fill-holes* [*simp*]:
num-holes C = *length ts* $\Longrightarrow$ *ground-mctxt C* $\Longrightarrow$
vars-term (*fill-holes C ts*) = $\bigcup$(*vars-term* ‘ *set ts*)
⟨*proof*⟩

lemma *funas-mctxt-fill-holes* [*simp*]:
assumes *num-holes C* = *length ts*
shows *funas-term* (*fill-holes C ts*) = *funas-mctxt C* ∪ $\bigcup$(*set* (*map funas-term ts*))
⟨*proof*⟩

lemma *funas-mctxt-fill-holes-mctxt* [*simp*]:
assumes *num-holes C* = *length Ds*
shows *funas-mctxt* (*fill-holes-mctxt C Ds*) = *funas-mctxt C* ∪ $\bigcup$(*set* (*map funas-mctxt Ds*))
(**is** *?f C Ds* = *?g C Ds*)
⟨*proof*⟩

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

2.2 Ground multihole context

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

2.2.1 Basic function on ground mutlihole contexts

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

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

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

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

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

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

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

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

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

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

fun *fill-gholes* :: ′*f gmctxt* ⇒ ′*f gterm list* ⇒ ′*f gterm* **where**
fill-gholes GMHole [*t*] = *t*
| *fill-gholes* (*GMFun f cs*) *ts* = *GFun f* (*map* (λ *i*. *fill-gholes* (*cs* ! *i*)
(*partition-gholes ts cs* ! *i*)) [*0* ..< *length cs*])

fun *fill-gholes-gmctxt* :: ′*f gmctxt* ⇒ ′*f gmctxt list* ⇒ ′*f gmctxt* **where**
fill-gholes-gmctxt GMHole [] = *GMHole* |
fill-gholes-gmctxt GMHole [*t*] = *t* |
fill-gholes-gmctxt (*GMFun f cs*) *ts* = (*GMFun f* (*map* (λ *i*. *fill-gholes-gmctxt* (*cs* ! *i*)
(*partition-gholes ts cs* ! *i*)) [*0* ..< *length cs*]))

2.2.2 An inverse of *fill-gholes*

fun *unfill-gholes* :: ′*f gmctxt* ⇒ ′*f gterm* ⇒ ′*f gterm list* **where**
unfill-gholes GMHole t = [*t*]
| *unfill-gholes* (*GMFun g Cs*) (*GFun f ts*) = (*if f* = *g* ∧ *length ts* = *length Cs then*
concat (*map* (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* | *i p*. *i* < *length Cs* ∧ *p* ∈ *ghole-poss* (*Cs* ! *i*)}
poss-gmctxt (*GMFun f Cs*) = {[]} ∪ {*i* # *p* | *i p*. *i* < *length Cs* ∧ *p* ∈ *poss-gmctxt* (*Cs* ! *i*)}
⟨*proof*⟩

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

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

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

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

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

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

instance ⟨*proof*⟩
end

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

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

else undefined)

instance ⟨*proof*⟩
end

2.2.3 Orderings and compatibility of ground multihole contexts

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

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

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

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

2.2.4 Conversions from and to ground multihole contexts

lemma *num-gholes-o-gmctxt-of-gterm* [*simp*]:
num-gholes ∘ *gmctxt-of-gterm* = (λ*x*. *0*)
⟨*proof*⟩

lemma *mctxt-of-term-term-of-mctxt-id* [*simp*]:
num-gholes C = *0* ⟹ *gmctxt-of-gterm* (*gterm-of-gmctxt C*) = *C*
⟨*proof*⟩

lemma *num-holes-mctxt-of-term* [*simp*]:
num-gholes (*gmctxt-of-gterm t*) = *0*
⟨*proof*⟩

lemma *num-gholes-gmctxt-of-mctxt* [*simp*]:
ground-mctxt C ⟹ *num-gholes* (*gmctxt-of-mctxt C*) = *num-holes C*
⟨*proof*⟩

lemma *num-holes-mctxt-of-gmctxt* [*simp*]:
num-holes (*mctxt-of-gmctxt C*) = *num-gholes C*
⟨*proof*⟩

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

num-holes ∘ *mctxt-of-gmctxt* = *num-gholes*
⟨*proof*⟩

lemma *gmctxt-of-gctxt-num-gholes* [*simp*]:
num-gholes (*gmctxt-of-gctxt C*) = *Suc 0*
⟨*proof*⟩

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

lemma *num-gholes-map-gmctxt* [*simp*]:
num-gholes (*map-gmctxt f C*) = *num-gholes C*
⟨*proof*⟩

lemma *map-num-gholes-map-gmctxt* [*simp*]:
map (*num-gholes* ∘ *map-gmctxt f*) *Cs* = *map num-gholes Cs*
⟨*proof*⟩

lemma *gterm-of-gmctxt-gmctxt-of-gterm-id* [*simp*]:
gterm-of-gmctxt (*gmctxt-of-gterm t*) = *t*
⟨*proof*⟩

lemma *no-gholes-gmctxt-of-gterm-gterm-of-gmctxt-id* [*simp*]:
num-gholes C = *0* ⟹ *gmctxt-of-gterm* (*gterm-of-gmctxt C*) = *C*
⟨*proof*⟩

lemma *no-gholes-term-of-gterm-gterm-of-gmctxt* [*simp*]:
num-gholes C = *0* ⟹ *term-of-gterm* (*gterm-of-gmctxt C*) = *term-of-gmctxt C*
⟨*proof*⟩

lemma *no-gholes-term-of-mctxt-mctxt-of-gmctxt* [*simp*]:
num-gholes C = *0* ⟹ *term-of-mctxt* (*mctxt-of-gmctxt C*) = *term-of-gmctxt C*
⟨*proof*⟩

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

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

lemma *gctxt-of-gmctxt-gmctxt-of-gctxt* [*simp*]:
gctxt-of-gmctxt (*gmctxt-of-gctxt C*) = *C*

⟨*proof*⟩

lemma *gmctxt-of-gctxt-GMHole-Hole*:
gmctxt-of-gctxt C = *GMHole* $\Longrightarrow$ *C* = $\square_G$
⟨*proof*⟩

lemma *gmctxt-of-gctxt-gctxt-of-gmctxt*:
num-gholes C = *Suc 0* $\Longrightarrow$ *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* $\Longrightarrow$ *gctxt-of-gmctxt C* = $\square_G$ $\Longrightarrow$ *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* $\Longrightarrow$ *ground-mctxt* (*MFun f D*)
⟨*proof*⟩

lemma *gmctxt-of-mctxt-inv* [*simp*]:
ground-mctxt C $\Longrightarrow$ *mctxt-of-gmctxt* (*gmctxt-of-mctxt C*) = *C*
⟨*proof*⟩

lemma *ground-mctxt-of-gmctxtD*:
ground-mctxt C $\Longrightarrow$ ∃ *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* ∘ *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 ⟹ *map-gmctxt f* (*gmctxt-of-mctxt C*) = *gmctxt-of-mctxt* (*map-mctxt f C*)
⟨*proof*⟩

lemma *map-gmctxt-nempty* [*simp*]:
C ≠ *GMHole* ⟹ *map-gmctxt f C* ≠ *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*) ⊆ *Q* ⟷ *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*
⋀ *i*. *i* < *length Cs* ⟹ *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* ∈ *set* (*partition-holes xs Cs* ! *i*)
shows *x* ∈ *set xs*
⟨*proof*⟩

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

and *x* ∈ *set* (*partition-gholes xs Cs* ! *i*)
shows *x* ∈ *set xs*
⟨*proof*⟩

lemma *list-elem-to-partition-nth* [*elim*]:
assumes *sum-list* (*map num-gholes Cs*) = *length xs x* ∈ *set xs*
obtains *i* **where** *i* < *length Cs x* ∈ *set* (*partition-gholes xs Cs* ! *i*) ⟨*proof*⟩

lemma *partition-holes-fill-gholes-conv′*:
fill-gholes (*GMFun f Cs*) *ts* =
GFun f (*map* (*case-prod fill-gholes*) (*zip Cs* (*partition-gholes ts Cs*)))
⟨*proof*⟩

lemma *unfill-gholes-conv*:
assumes *length Cs* = *length ts*
shows *unfill-gholes* (*GMFun f Cs*) (*GFun f ts*) =
concat (*map* (*case-prod unfill-gholes*) (*zip Cs ts*)) ⟨*proof*⟩

lemma *partition-holes-fill-gholes-gmctxt-conv*:
fill-gholes-gmctxt (*GMFun f Cs*) *ts* =
GMFun f [*fill-gholes-gmctxt* (*Cs* ! *i*) (*partition-gholes ts Cs* ! *i*). *i* ← [*0* ..< *length Cs*]]
⟨*proof*⟩

lemma *partition-holes-fill-gholes-gmctxt-conv′*:
fill-gholes-gmctxt (*GMFun f Cs*) *ts* =
GMFun f (*map* (*case-prod fill-gholes-gmctxt*) (*zip Cs* (*partition-gholes ts Cs*)))
⟨*proof*⟩

lemma *fill-gholes-no-holes* [*simp*]:
num-gholes C = *0* ⟹ *fill-gholes C* [] = *gterm-of-gmctxt C*
⟨*proof*⟩

lemma *fill-gholes-gmctxt-no-holes* [*simp*]:
num-gholes C = *0* ⟹ *fill-gholes-gmctxt C* [] = *C*
⟨*proof*⟩

lemma *eqgf-GMFunI*:
assumes ⋀ *i*. *i* < *length Cs* ⟹ *ss* ! *i* $=_{Gf}$ (*Cs* ! *i*, *ts* ! *i*)
and *length Cs* = *length ss length ss* = *length ts*
shows *GFun f ss* $=_{Gf}$ (*GMFun f Cs*, *concat ts*) ⟨*proof*⟩

lemma *length-partition-gholes-nth*:
assumes *sum-list* (*map num-gholes cs*) = *length ts*
and *i* < *length cs*
shows *length* (*partition-gholes ts cs* ! *i*) = *num-gholes* (*cs* ! *i*)
⟨*proof*⟩

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

fixes P :: *'f gmctxt* ⇒ *'a list* ⇒ *'b list* ⇒ *bool*
assumes *len1*: *num-gholes C* = *length xs* **and** *len2*: *num-gholes C* = *length ys*
and *Hole*: ⋀*x y*. *P GMHole* [*x*] [*y*]
and *Fun*: ⋀*f Cs xs ys*. *sum-list* (*map num-gholes Cs*) = *length xs* ⟹
sum-list (*map num-gholes Cs*) = *length ys* ⟹
(⋀*i*. *i* < *length Cs* ⟹ *P* (*Cs* ! *i*) (*partition-gholes xs Cs* ! *i*) (*partition-gholes ys Cs* ! *i*)) ⟹
P (*GMFun f Cs*) (*concat* (*partition-gholes xs Cs*)) (*concat* (*partition-gholes ys Cs*))
shows *P C xs ys*
⟨*proof*⟩

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

lemma *eq-gfill-induct* [*consumes 1*, *case-names GMHole GMFun*]:
assumes $t =_{Gf}$ (*C*, *ts*)
and ⋀*t*. *P t GMHole* [*t*]
and ⋀*f ss Cs ts*. ⟦*length Cs* = *length ss*; *sum-list* (*map num-gholes Cs*) = *length ts*;
∀ *i* < *length ss*. *ss* ! $i =_{Gf}$ (*Cs* ! *i*, *partition-gholes ts Cs* ! *i*) ∧
P (*ss* ! *i*) (*Cs* ! *i*) (*partition-gholes ts Cs* ! *i*)⟧
⟹ *P* (*GFun f ss*) (*GMFun f Cs*) *ts*
shows *P t C ts* ⟨*proof*⟩

lemma *nempty-ground-mctxt-gmctxt* [*simp*]:
C ≠ *MHole* ⟹ *ground-mctxt C* ⟹ *gmctxt-of-mctxt C* ≠ *GMHole*
⟨*proof*⟩

lemma *mctxt-of-gmctxt-fill-holes* [*simp*]:
assumes *num-gholes C* = *length ss*
shows *gterm-of-term* (*fill-holes* (*mctxt-of-gmctxt C*) (*map term-of-gterm ss*)) = *fill-gholes C ss* ⟨*proof*⟩

lemma *mctxt-of-gmctxt-terms-fill-holes*:
assumes *num-gholes C* = *length ss*
shows *gterm-of-term* (*fill-holes* (*mctxt-of-gmctxt C*) *ss*) = *fill-gholes C* (*map gterm-of-term ss*) ⟨*proof*⟩

lemma *ground-gmctxt-of-mctxt-gterm-fill-holes*:
assumes *num-holes C* = *length ss* **and** *ground-mctxt C*
shows *term-of-gterm* (*fill-gholes* (*gmctxt-of-mctxt C*) *ss*) = *fill-holes C* (*map*

term-of-gterm ss) ⟨*proof*⟩

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* ⟨*proof*⟩

lemma *ground-gmctxt-of-mctxt-fill-holes* [*simp*]:
assumes *num-holes C = length ss* **and** *ground-mctxt C* ∀ *s* ∈ *set ss. ground s*
shows *term-of-gterm* (*fill-gholes* (*gmctxt-of-mctxt C*) (*map gterm-of-term ss*)) = *fill-holes C ss* ⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *fill-gholes-gmctxt-of-gterm* [*simp*]:
fill-gholes (*gmctxt-of-gterm s*) [] = *s*
⟨*proof*⟩

lemma *fill-gholes-GMHole* [*simp*]:
length ss = Suc 0 ⟹ *fill-gholes GMHole ss = ss* ! *0*
⟨*proof*⟩

lemma *apply-gctxt-fill-gholes*:
$C\langle s\rangle_G$ = *fill-gholes* (*gmctxt-of-gctxt C*) [*s*]
⟨*proof*⟩

lemma *fill-gholes-apply-gctxt*:
num-gholes C = Suc 0 ⟹ *fill-gholes C* [*s*] = (*gctxt-of-gmctxt C*)$\langle s\rangle_G$
⟨*proof*⟩

lemma *ctxt-of-gctxt-gctxt-of-gmctxt-apply*:
num-gholes C = Suc 0 ⟹ *fill-holes* (*mctxt-of-gmctxt C*) [*s*] = (*ctxt-of-gctxt* (*gctxt-of-gmctxt C*))⟨*s*⟩
⟨*proof*⟩

lemma *fill-gholes-replicate* [*simp*]:
n = length ss ⟹ *fill-gholes* (*GMFun f* (*replicate n GMHole*)) *ss = GFun f ss*
⟨*proof*⟩

lemma *fill-gholes-gmctxt-replicate-MHole* [*simp*]:
fill-gholes-gmctxt C (*replicate* (*num-gholes C*) *GMHole*) = *C*
⟨*proof*⟩

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*: ⋀ *i*. *i* < *length ts* ⟹ *num-gholes* (*Cs* ! *i*) = *length* (*ss* ! *i*) ∧
fill-gholes-gmctxt (*Cs* ! *i*) (*ss* ! *i*) = *ts* ! *i*
shows *fill-gholes-gmctxt* (*GMFun f Cs*) (*concat ss*) = *GMFun f ts*
⟨*proof*⟩

lemma *fill-gholes-gmctxt-nHole* [*simp*]:
C ≠ *GMHole* ⟹ *num-gholes C* = *length Ds* ⟹ *fill-gholes-gmctxt C Ds* ≠ *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 ∃ *D* ∈ *set Ds*. *0* < *num-gholes D*
shows *0* < *sum-list* (*map 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* ← [*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*: ⋀ *i*. *i* < *length Ds* ⟹ *ts* ! *i* $=_{Gf}$ (*Ds* ! *i*, *sss* ! *i*)
shows *fill-gholes C ts* $=_{Gf}$ (*fill-gholes-gmctxt C Ds*, *concat sss*)
⟨*proof*⟩

2.2.6 Semilattice Structures

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

lemma *inf-gmctxt-GMHole2* [*simp*]:
  $C \sqcap GMHole = GMHole$
  ⟨*proof*⟩

lemma *inf-gmctxt-comm* [*ac-simps*]:
  $(C :: 'f\ gmctxt) \sqcap D = D \sqcap C$
  ⟨*proof*⟩

lemma *inf-gmctxt-assoc* [*ac-simps*]:
  fixes $C :: 'f\ gmctxt$
  shows $C \sqcap D \sqcap E = C \sqcap (D \sqcap E)$
  ⟨*proof*⟩

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

definition $(C :: 'a\ gmctxt) \le D \longleftrightarrow C \sqcap D = C$
definition $(C :: 'a\ gmctxt) < D \longleftrightarrow C \le D \land \neg D \le C$

instance
  ⟨*proof*⟩

end

lemma *less-eq-gmctxt-prime*: $C \le D \longleftrightarrow$ *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*:
  less-eq-gmctxt C GMHole $\Longrightarrow C = GMHole$
  ⟨*proof*⟩

lemma *num-gholes-at-least1*:
  $0 <$ *num-gholes* $C \Longrightarrow 0 <$ *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 $C :: 'f\ gmctxt$
  shows $C \sqcup C = C$

⟨*proof*⟩

lemma *sup-gmctxt-MHole* [*simp*]: $C \sqcup GMHole = C$
⟨*proof*⟩

lemma *sup-gmctxt-comm* [*ac-simps*]:
fixes C :: *'f gmctxt*
shows $C \sqcup D = D \sqcup C$
⟨*proof*⟩

lemma *comp-gmctxt-refl*:
$(C, C) \in$ *comp-gmctxt*
⟨*proof*⟩

lemma *comp-gmctxt-sym*:
assumes $(C, D) \in$ *comp-gmctxt*
shows $(D, C) \in$ *comp-gmctxt*
⟨*proof*⟩

lemma *sup-gmctxt-assoc* [*ac-simps*]:
assumes $(C, D) \in$ *comp-gmctxt* **and** $(D, E) \in$ *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 $(C, D) \in$ *comp-gmctxt*
shows $C \leq C \sqcup D$
⟨*proof*⟩

lemma *sup-gmctxt-ge2* [*simp*]:
assumes $(C, D) \in$ *comp-gmctxt*
shows $D \leq C \sqcup D$
⟨*proof*⟩

lemma *sup-gmctxt-least*:
assumes $(D, E) \in$ *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 $(C, D) \in$ *comp-gmctxt*
shows *num-gholes C* = *length* (*sup-gmctxt-args C D*)
⟨*proof*⟩

lemma *sup-gmctxt-sup-gmctxt-args*:
assumes $(C, D) \in$ *comp-gmctxt*
shows *fill-gholes-gmctxt C* (*sup-gmctxt-args C D*) $= C \sqcup D$ ⟨*proof*⟩

lemma *eqgf-comp-gmctxt*:
assumes $s =_{Gf} (C, ss)$ **and** $s =_{Gf} (D, ts)$
shows $(C, D) \in$ *comp-gmctxt* ⟨*proof*⟩

lemma *eqgf-less-eq* [*simp*]:
assumes $s =_{Gf} (C, ss)$
shows $C \leq$ *gmctxt-of-gterm s* ⟨*proof*⟩

lemma *less-eq-comp-gmctxt* [*simp*]:
$C \leq D \Longrightarrow (C, D) \in$ *comp-gmctxt*
⟨*proof*⟩

lemma *gmctxt-less-eq-sup*:
(*C* :: *'f gmctxt*) $\leq D \Longrightarrow C \sqcup D = D$
⟨*proof*⟩

lemma *fill-gholes-gmctxt-less-eq*:
assumes *num-gholes C* = *length Ds*
shows $C \leq$ *fill-gholes-gmctxt C Ds* ⟨*proof*⟩

lemma *less-eq-to-sup-mctxt-args* [*elim*]:
assumes $C \leq D$
obtains *Ds* **where** *num-gholes C* = *length Ds D* = *fill-gholes-gmctxt C Ds*
⟨*proof*⟩

lemma *fill-gholes-gmctxt-sup-mctxt-args* [*simp*]:
assumes *num-gholes C* = *length Ds*
shows *sup-gmctxt-args C* (*fill-gholes-gmctxt C Ds*) = *Ds* ⟨*proof*⟩

lemma *map2-fill-gholes-gmctxt-id* [*simp*]:
assumes $\bigwedge$ *i*. *i* < *length Ds* $\Longrightarrow$ *num-gholes* (*Ds* ! *i*) = *0*
shows *map2 fill-gholes-gmctxt Ds* (*replicate* (*length Ds*) []) = *Ds*
⟨*proof*⟩

lemma *fill-gholes-gmctxt-GMFun-replicate-append* [*simp*]:
assumes *length Cs* = *n* **and** $\bigwedge$ *t*. *t* $\in$ *set Ds* $\Longrightarrow$ *num-gholes t* = *0*
shows *fill-gholes-gmctxt* (*GMFun f* ((*replicate n GMHole*) @ *Ds*)) *Cs* = *GMFun f* (*Cs* @ *Ds*) ⟨*proof*⟩

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-ghloes-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** $q \leq_p p$ *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 $\Longrightarrow$ *gmctxt-closing D E* $\Longrightarrow$ *gmctxt-closing C E*
⟨*proof*⟩

lemma *gmctxt-closing-sup-args-ghole-or-gterm*:
assumes *gmctxt-closing C D*
shows ∀ *E* ∈ *set* (*sup-gmctxt-args C D*). *E* = *GMHole* ∨ *num-gholes E* = *0*
⟨*proof*⟩

lemma *inv-imples-ghole-poss-subseteq*:
C ≤ *D* $\Longrightarrow$ ∀ *E* ∈ *set* (*sup-gmctxt-args C D*). *E* = *GMHole* ∨ *num-gholes E* = *0* $\Longrightarrow$ *ghole-poss D* ⊆ *ghole-poss C*
⟨*proof*⟩

lemma *fill-gholes-gmctxt-ghole-poss-subseteq*:
assumes *num-gholes C* = *length Ds* ∀ *i* < *length Ds*. *Ds* ! *i* = *GMHole* ∨ *num-gholes* (*Ds* ! *i*) = *0*
shows *ghole-poss* (*fill-gholes-gmctxt C Ds*) ⊆ *ghole-poss C* ⟨*proof*⟩

lemma *ghole-poss-not-in-poss-gmctxt*:
assumes *p* ∈ *ghole-poss C*
shows *p* ∉ *poss-gmctxt C* ⟨*proof*⟩

lemma *comp-gmctxt-inf-ghole-poss-cases*:
assumes (*C*, *D*) ∈ *comp-gmctxt p* ∈ *ghole-poss* (*C* ⊓ *D*)
shows *p* ∈ *ghole-poss C* ∧ *p* ∈ *ghole-poss D* ∨
p ∈ *ghole-poss C* ∧ *p* ∈ *poss-gmctxt D* ∨
p ∈ *ghole-poss D* ∧ *p* ∈ *poss-gmctxt C* ⟨*proof*⟩

lemma *length-ghole-poss-list-num-gholes*:
num-gholes C = *length* (*ghole-poss-list C*)
⟨*proof*⟩

lemma *ghole-poss-list-distict*:
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 ∈ *ghole-poss C* $\Longrightarrow$ *subgm-at C p* = *GMHole*
⟨*proof*⟩

lemma *subgm-at-mctxt-of-term*:
$p \in gposs\ t \Longrightarrow subgm\text{-}at\ (gmctxt\text{-}of\text{-}gterm\ t)\ p = gmctxt\text{-}of\text{-}gterm\ (gsubt\text{-}at\ t\ p)$
⟨*proof*⟩

lemma *num-gholes-subgm-at*:
assumes $p \in poss\text{-}gmctxt\ C$
shows $num\text{-}gholes\ (subgm\text{-}at\ C\ p) = ghole\text{-}num\text{-}at\text{-}pos\ p\ C$ ⟨*proof*⟩

lemma *gmctxt-subtgm-at-fill-args-empty-pos* [*simp*]:
assumes $num\text{-}gholes\ C = length\ ts$
shows $gmctxt\text{-}subtgm\text{-}at\text{-}fill\text{-}args\ []\ C\ ts = ts$
⟨*proof*⟩

lemma *ghole-num-bef-at-pos-num-gholes-less-eq*:
assumes $p \in poss\text{-}gmctxt\ C$
shows $ghole\text{-}num\text{-}bef\text{-}pos\ p\ C + ghole\text{-}num\text{-}at\text{-}pos\ p\ C \leq num\text{-}gholes\ C$ ⟨*proof*⟩

lemma *ghole-num-at-pos-fill-args-length*:
assumes $p \in poss\text{-}gmctxt\ C\ num\text{-}gholes\ C = length\ ts$
shows $ghole\text{-}num\text{-}at\text{-}pos\ p\ C = length\ (gmctxt\text{-}subtgm\text{-}at\text{-}fill\text{-}args\ p\ C\ ts)$
⟨*proof*⟩

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

lemma *poss-gmctxt-fill-gholes-split*:
assumes $t =_{Gf} (C,\ ts)$ **and** $p \in poss\text{-}gmctxt\ C$
shows $gsubt\text{-}at\ t\ p =_{Gf} (subgm\text{-}at\ C\ p\ ,\ gmctxt\text{-}subtgm\text{-}at\text{-}fill\text{-}args\ p\ C\ ts)$
⟨*proof*⟩

lemma *fill-gholes-ghole-poss*:
assumes $t =_{Gf} (C,\ ts)$ **and** $i < length\ ts$
shows $gsubt\text{-}at\ t\ (ghole\text{-}poss\text{-}list\ C\ !\ i) = ts\ !\ i$ ⟨*proof*⟩

lemma *length-unfill-gholes* [*simp*]:
assumes $C \leq gmctxt\text{-}of\text{-}gterm\ t$
shows $length\ (unfill\text{-}gholes\ C\ t) = num\text{-}gholes\ C$
⟨*proof*⟩

lemma *fill-gholes-arbitrary*:
assumes *lCs*: $length\ Cs = length\ ts$
and *lss*: $length\ ss = length\ ts$
and *rec*: $\bigwedge i.\ i < length\ ts \Longrightarrow num\text{-}gholes\ (Cs\ !\ i) = length\ (ss\ !\ i) \wedge f\ (Cs\ !\ i)\ (ss\ !\ i) = ts\ !\ i$
shows $map\ (\lambda i.\ f\ (Cs\ !\ i)\ (partition\text{-}gholes\ (concat\ ss)\ Cs\ !\ i))\ [0\ ..<\ length\ Cs] = ts$
⟨*proof*⟩

lemma *fill-unfill-gholes*:
assumes $C \leq$ *gmctxt-of-gterm t*
shows *fill-gholes C* (*unfill-gholes C t*) = *t*
⟨*proof*⟩

lemma *funas-gmctxt-of-mctxt* [*simp*]:
ground-mctxt C ⟹ *funas-gmctxt* (*gmctxt-of-mctxt C*) = *funas-mctxt C*
⟨*proof*⟩

lemma *funas-mctxt-of-gmctxt-conv*:
funas-mctxt (*mctxt-of-gmctxt C*) = *funas-gmctxt C*
⟨*proof*⟩

lemma *funas-gterm-ctxt-apply* [*simp*]:
assumes *num-gholes C* = *length ss*
shows *funas-gterm* (*fill-gholes C ss*) = *funas-gmctxt C* ∪ ⋃ (*set* (*map funas-gterm ss*)) ⟨*proof*⟩

lemma *funas-gmctxt-gmctxt-of-gterm* [*simp*]:
funas-gmctxt (*gmctxt-of-gterm s*) = *funas-gterm s*
⟨*proof*⟩

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

lemma *funas-gmctxt-gmctxt-of-gctxt* [*simp*]:
funas-gmctxt (*gmctxt-of-gctxt C*) = *funas-gctxt C*
⟨*proof*⟩

lemma *funas-gmctxt-fill-gholes-gmctxt* [*simp*]:
assumes *num-gholes C* = *length Ds*
shows *funas-gmctxt* (*fill-gholes-gmctxt C Ds*) = *funas-gmctxt C* ∪ ⋃(*set* (*map funas-gmctxt Ds*))
(**is** *?f C Ds* = *?g C Ds*) ⟨*proof*⟩

lemma *funas-supremum*:
$C \leq D$ ⟹ *funas-gmctxt D* = *funas-gmctxt C* ∪ ⋃ (*set* (*map funas-gmctxt* (*sup-gmctxt-args C D*)))
⟨*proof*⟩

lemma *funas-gctxt-gctxt-of-gmctxt* [*simp*]:
num-gholes D = *Suc 0* ⟹ *funas-gctxt* (*gctxt-of-gmctxt D*) = *funas-gmctxt D*
⟨*proof*⟩

lemma *funas-gterm-gterm-of-gmctxt* [*simp*]:
num-gholes C = *0* ⟹ *funas-gterm* (*gterm-of-gmctxt C*) = *funas-gmctxt C*
⟨*proof*⟩

lemma *less-sup-gmctxt-args-funas-gmctxt*:
$C \leq D \Longrightarrow$ *funas-gmctxt* $C \subseteq \mathcal{F} \Longrightarrow \forall$ *Ds* $\in$ *set* (*sup-gmctxt-args* C D). *funas-gmctxt* *Ds* $\subseteq \mathcal{F} \Longrightarrow$ *funas-gmctxt* $D \subseteq \mathcal{F}$
⟨*proof*⟩

lemma *funas-gmctxt-poss-gmctxt-subgm-at-funas*:
assumes *funas-gmctxt* $C \subseteq \mathcal{F}$ $p \in$ *poss-gmctxt* C
shows *funas-gmctxt* (*subgm-at* C p) $\subseteq \mathcal{F}$
⟨*proof*⟩

lemma *inf-funas-gmctxt-subset1*:
funas-gmctxt $(C \sqcap D) \subseteq$ *funas-gmctxt* C
⟨*proof*⟩

lemma *inf-funas-gmctxt-subset2*:
funas-gmctxt $(C \sqcap D) \subseteq$ *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* ⇒ *'f bot-term* $(\text{-}^{\perp}\ [80]\ 80)$ **where**
$(Var\ \text{-})^{\perp} = Bot$
| $(Fun\ f\ ts)^{\perp} =$ *BFun f* (*map term-to-bot-term ts*)

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

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

lemma *finite-funas-bot-term*:
finite (*funas-bot-term t*)
⟨*proof*⟩

lemma *funas-bot-term-funas-term*:
funas-bot-term $(t^{\perp})$ = *funas-term t*
⟨*proof*⟩

lemma *term-to-bot-term-root-bot* [*simp*]:
  root-bot ($t^\perp$) = *root t*
  ⟨*proof*⟩

lemma *term-to-bot-term-root-bot-comp* [*simp*]:
  root-bot ∘ *term-to-bot-term* = *root*
  ⟨*proof*⟩

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

lemma *merge-refl*:
  (*s*, *s*) ∈ *mergeP*
  ⟨*proof*⟩

lemma *merge-symmetric*:
  assumes (*s*, *t*) ∈ *mergeP*
  shows (*t*, *s*) ∈ *mergeP*
  ⟨*proof*⟩

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

lemma *merge-terms-bot-rhs*[*simp*]:
  s ↑ *Bot* = *s* ⟨*proof*⟩

lemma *merge-terms-idem*: *s* ↑ *s* = *s*
  ⟨*proof*⟩

lemma *merge-terms-assoc* [*ac-simps*]:
  assumes (*s*, *t*) ∈ *mergeP* **and** (*t*, *u*) ∈ *mergeP*
  shows (*s* ↑ *t*) ↑ *u* = *s* ↑ *t* ↑ *u*
  ⟨*proof*⟩

lemma *merge-terms-commutative* [*ac-simps*]:
  shows *s* ↑ *t* = *t* ↑ *s*
  ⟨*proof*⟩

lemma *merge-dist*:
  assumes (*s*, *t* ↑ *u*) ∈ *mergeP* **and** (*t*, *u*) ∈ *mergeP*
  shows (*s*, *t*) ∈ *mergeP* ⟨*proof*⟩

lemma *megeP-ass*:
$(s,\ t \uparrow u) \in mergeP \Longrightarrow (t,\ u) \in mergeP \Longrightarrow (s \uparrow t,\ u) \in mergeP$
⟨*proof*⟩

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

Infix syntax.

abbreviation *bless-eq-pred* $s\ t \equiv (s,\ t) \in bless\text{-}eq$
notation
bless-eq $(\{\leq_b\})$ **and**
bless-eq-pred $((\text{-}/ \leq_b \text{-})\ [56,\ 56]\ 55)$

lemma *BFun-leq-Bot-False* [*simp*]:
$BFun\ f\ ts \leq_b Bot \longleftrightarrow False$
⟨*proof*⟩

lemma *BFun-lesseqE* [*elim*]:
assumes $BFun\ f\ ts \leq_b t$
obtains *us* **where** $length\ ts = length\ us\ t = BFun\ f\ us$
⟨*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*:
$s \leq_b t \Longrightarrow t \leq_b s \Longrightarrow s = t$
⟨*proof*⟩

lemma *bless-eq-mergeP*:
$s \leq_b t \Longrightarrow (s,\ t) \in mergeP$
⟨*proof*⟩

lemma *merge-bot-args-bless-eq-merge*:
assumes $(s,\ t) \in mergeP$
shows $s \leq_b s \uparrow t$ ⟨*proof*⟩

lemma *bless-eq-closued-under-merge*:
assumes $(s,\ t) \in mergeP\ (u,\ v) \in mergeP\ s \leq_b u\ t \leq_b v$
shows $s \uparrow t \leq_b u \uparrow v$ ⟨*proof*⟩

lemma *bless-eq-closued-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 *length ss* = *length ts*
and $\bigwedge i.\ i < length\ ts \Longrightarrow ss\ !\ i \cdot \sigma\ i = ts\ !\ i$
shows $\exists\ \sigma.\ Fun\ f\ ss \cdot \sigma = Fun\ f\ ts$
⟨*proof*⟩

lemma *bless-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-bless-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*:
$(\forall\ i < length\ ss.\ ss\ !\ i \in S) \longleftrightarrow set\ ss \subseteq S$
⟨*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*:
fold f xs (*fold f ys y*) = *fold f ys* (*fold f xs y*)
⟨*proof*⟩

lemma (**in** *comp-fun-commute*) *foldr-list-swap*:
foldr f xs (*foldr f ys y*) = *foldr f ys* (*foldr f xs y*)
⟨*proof*⟩

lemma (**in** *comp-fun-commute*) *foldr-to-fold*:
foldr f xs = *fold f xs*
⟨*proof*⟩

lemma (**in** *comp-fun-commute*) *fold-commute-f*:
f x (*foldr f xs y*) = *foldr f xs* (*f x y*)

⟨*proof*⟩

lemma *closure-sound*:
assumes *cl*: $\bigwedge s\ t.\ s \in S \Longrightarrow t \in S \Longrightarrow f\ s\ t \in S$
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*) $\in S$ ⟨*proof*⟩

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

sublocale *set-closure-oprator* $\subseteq$ *comp-fun-idem*
⟨*proof*⟩

context *set-closure-oprator*
begin

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

lemma *closure-idem* [*simp*]:
closure (*closure S*) = *closure S* (**is** *?LS* = *?RS*)
⟨*proof*⟩

lemma *fold-dist*:
assumes *xs* $\neq []$
shows *f* (*fold f* (*tl xs*) (*hd xs*)) *t* = *fold f xs t* ⟨*proof*⟩

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* $\in S$) ⟨*proof*⟩

lemma *sound-fold*:
assumes *set ss* $\subseteq$ *closure S* **and** *ss* $\neq []$
shows *fold f* (*tl ss*) (*hd ss*) $\in$ *closure S* ⟨*proof*⟩

lemma *closure-empty* [*simp*]: *closure* {} = {}
⟨*proof*⟩

lemma *closure-mono*:
$S \subseteq T \Longrightarrow$ *closure S* $\subseteq$ *closure T*
⟨*proof*⟩

lemma *closure-insert*:

closure (*insert x S*) = {*x*} ∪ *closure S* ∪ {*f x s* | *s*. *s* ∈ *closure S*}
⟨*proof*⟩

lemma *finite-S-finite-closure* [*intro*]:
finite S ⟹ *finite* (*closure S*)
⟨*proof*⟩

end

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

lemma *unique-neut-elem* [*simp*]:
f x y = *e* ⟷ *x* = *e* ∧ *y* = *e*
⟨*proof*⟩

abbreviation *closure S* ≡ *cl.closure S*

lemma *closure-to-cons-listE*:
assumes *s* ∈ *closure S*
obtains *ss* **where** *ss* ≠ [] *fold f ss e* = *s set ss* ⊆ *S*
⟨*proof*⟩

lemma *sound-fold*:
assumes *set ss* ⊆ *closure S ss* ≠ []
shows *fold f ss e* ∈ *closure S*
⟨*proof*⟩

abbreviation *supremum S* ≡ *Finite-Set.fold f e S*
definition *smaller-subset x S* ≡ {*y*. *less-eq y x* ∧ *y* ∈ *S*}

lemma *smaller-subset-empty* [*simp*]:
smaller-subset x {} = {}
⟨*proof*⟩

lemma *finite-smaller-subset* [*simp*, *intro*]:
finite S ⟹ *finite* (*smaller-subset x S*)
⟨*proof*⟩

lemma *smaller-subset-mono*:
smaller-subset x S $\subseteq$ *S*
⟨*proof*⟩

lemma *sound-set-fold*:
assumes *set ss* $\subseteq$ *closure S* **and** *ss* $\neq$ []
shows *supremum* (*set ss*) $\in$ *closure S*
⟨*proof*⟩

lemma *supremum-neutral* [*simp*]:
assumes *finite S* **and** *supremum S* = *e*
shows *S* $\subseteq$ {*e*} ⟨*proof*⟩

lemma *supremum-in-closure*:
assumes *finite S* **and** *R* $\subseteq$ *closure S* **and** *R* $\neq$ {}
shows *supremum R* $\in$ *closure S*
⟨*proof*⟩

lemma *supremum-sound*:
assumes *finite S*
shows $\bigwedge$ *t*. *t* $\in$ *S* $\Longrightarrow$ *less-eq t* (*supremum S*)
⟨*proof*⟩

lemma *supremum-sound-list*:
$\forall$ *i* < *length ss*. *less-eq* (*ss* ! *i*) (*fold f ss e*)
⟨*proof*⟩

lemma *smaller-subset-insert* [*simp*]:
less-eq y x $\Longrightarrow$ *smaller-subset x* (*insert y S*) = *insert y* (*smaller-subset x S*)
$\neg$ *less-eq y x* $\Longrightarrow$ *smaller-subset x* (*insert y S*) = *smaller-subset x S*
⟨*proof*⟩

lemma *supremum-smaller-subset*:
assumes *finite S*
shows *less-eq* (*supremum* (*smaller-subset x S*)) *x* ⟨*proof*⟩

lemma *pre-subset-eq-pos-subset* [*simp*]:
shows *smaller-subset x* (*closure S*) = *closure* (*smaller-subset x S*) (**is** *?LS* = *?RS*)
⟨*proof*⟩

lemma *supremum-in-smaller-closure*:
assumes *finite S*
shows *supremum* (*smaller-subset x S*) $\in$ {*e*} $\cup$ (*closure S*)
⟨*proof*⟩

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-oprator* =
fixes *P f*
assumes *refl*: $\bigwedge x.\ P\ x\ x$
and *sym*: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow P\ y\ x$
and *dist*: $\bigwedge x\ y\ z.\ P\ y\ z \Longrightarrow P\ x\ (f\ y\ z) \Longrightarrow P\ x\ y$
and *assP*: $\bigwedge x\ y\ z.\ P\ x\ (f\ y\ z) \Longrightarrow P\ y\ z \Longrightarrow P\ (f\ x\ y)\ z$
and *com* [*ac-simps*]: $\bigwedge x\ y.\ P\ x\ y \Longrightarrow f\ x\ y = f\ y\ x$
and *ass* [*ac-simps*]: $\bigwedge x\ y\ z.\ P\ x\ y \Longrightarrow P\ y\ z \Longrightarrow f\ x\ (f\ y\ z) = f\ (f\ x\ y)\ z$
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-oprator*:
set-closure-oprator (*lift-f-total P f*)
⟨*proof*⟩

end

sublocale *set-closure-partial-oprator* ⊆ *lift-fun*: *set-closure-oprator lift-f-total P f*
⟨*proof*⟩

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

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

inductive-set *pred-closure* **for** *S* **where**
base [*simp*]: *s* ∈ *S* ⟹ *s* ∈ *pred-closure S*
| *step* [*intro*]: *s* ∈ *pred-closure S* ⟹ *t* ∈ *pred-closure S* ⟹ *P s t* ⟹ *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* ≡ *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 ⟹ *finite* (*pred-closure S*)
⟨*proof*⟩

lemma *closure-mono*:
assumes *S* ⊆ *T*
shows *pred-closure S* ⊆ *pred-closure T*
⟨*proof*⟩

lemma *pred-closure-empty* [*simp*]:
pred-closure {} = {}
⟨*proof*⟩
end

locale *semilattice-closure-partial-operator* =

cl: *set-closure-partial-oprator 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. less-eq e x*
and *pred-less*: $\bigwedge$ *x y z. less-eq x y* $\Longrightarrow$ *less-eq z y* $\Longrightarrow$ *P x z*
and *sup-l*: $\bigwedge$ *x y. P x y* $\Longrightarrow$ *less-eq x (f x y)*
and *sup-r*: $\bigwedge$ *x y. P x y* $\Longrightarrow$ *less-eq y (f x y)*
and *upper-bound*: $\bigwedge$ *x y z. less-eq x z* $\Longrightarrow$ *less-eq y z* $\Longrightarrow$ *less-eq (f x y) z*
and *trans*: $\bigwedge$ *x y z. less-eq x y* $\Longrightarrow$ *less-eq y z* $\Longrightarrow$ *less-eq x z*
and *anti-sym*: $\bigwedge$ *x y. less-eq x y* $\Longrightarrow$ *less-eq y x* $\Longrightarrow$ *x = y*
begin

abbreviation *lifted-less-eq* $\equiv$ *lift-less-eq-total less-eq*
abbreviation *lifted-fun* $\equiv$ *lift-f-total P f*

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

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

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

lemma *lifted-less-eq-trans* [*trans*]:
lifted-less-eq x y $\Longrightarrow$ *lifted-less-eq y z* $\Longrightarrow$ *lifted-less-eq x z*
⟨*proof*⟩

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

lemma *lifted-less-eq-upper*:
lifted-less-eq x z $\Longrightarrow$ *lifted-less-eq y z* $\Longrightarrow$ *lifted-less-eq (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* ⊆ *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* ≡ *lift-ord.supremum*
abbreviation *smaller-subset* ≡ *lift-ord.smaller-subset*

lemma *supremum-impl*:
assumes *supremum* (*set* (*map Some ss*)) = *Some t*
shows *foldr f ss e* = *t* ⟨*proof*⟩

lemma *supremum-smaller-exists-unique*:
assumes *finite S*
shows ∃! *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*))) ∈ {*e*} ∪ *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-oprator*) *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 $'f\ sig = ('f \times nat)\ set$
type-synonym $('f, 'v)\ rule = ('f, 'v)\ term \times ('f, 'v)\ term$
type-synonym $('f, 'v)\ trs\ = ('f, 'v)\ rule\ set$

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

inductive-set $rstep :: \text{-} \Rightarrow ('f, 'v)\ term\ rel$ **for** $R :: ('f, 'v)\ trs$
  where
    $rstep$: $\bigwedge C\ \sigma\ l\ r.\ (l, r) \in R \Longrightarrow s = C\langle l \cdot \sigma\rangle \Longrightarrow t = C\langle r \cdot \sigma\rangle \Longrightarrow (s, t) \in rstep\ R$

definition $rstep\text{-}r\text{-}p\text{-}s :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ rule \Rightarrow pos \Rightarrow ('f, 'v)\ subst \Rightarrow ('f, 'v)\ trs$ **where**
  $rstep\text{-}r\text{-}p\text{-}s\ R\ r\ p\ \sigma = \{(s, t).\ p \in poss\ s \wedge p \in poss\ t \wedge r \in R \wedge ctxt\text{-}at\text{-}pos\ s\ p = ctxt\text{-}at\text{-}pos\ t\ p \wedge$
    $s[p \leftarrow (fst\ r \cdot \sigma)] = s \wedge t[p \leftarrow (snd\ r \cdot \sigma)] = t\}$

Rewriting steps below the root position.

definition $nrrstep :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ trs$ **where**
  $nrrstep\ R = \{(s,t).\ \exists\, r\ i\ ps\ \sigma.\ (s,t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ r\ (i\#ps)\ \sigma\}$

Rewriting step at the root position.

definition $rrstep :: ('f, 'v)\ trs \Rightarrow ('f, 'v)\ trs$ **where**
  $rrstep\ R = \{(s,t).\ \exists\, r\ \sigma.\ (s,t) \in rstep\text{-}r\text{-}p\text{-}s\ R\ r\ []\ \sigma\}$

the parallel rewrite relation

inductive-set $par\text{-}rstep :: ('f,'v)trs \Rightarrow ('f,'v)trs$ **for** $R :: ('f,'v)trs$
  where $root\text{-}step[intro]$: $(s,t) \in R \Longrightarrow (s \cdot \sigma, t \cdot \sigma) \in par\text{-}rstep\ R$
    | $par\text{-}step\text{-}fun[intro]$: $[\![\bigwedge i.\ i < length\ ts \Longrightarrow (ss\ !\ i, ts\ !\ i) \in par\text{-}rstep\ R]\!] \Longrightarrow length\ ss = length\ ts$
      $\Longrightarrow (Fun\ f\ ss,\ Fun\ f\ ts) \in par\text{-}rstep\ R$
    | $par\text{-}step\text{-}var[intro]$: $(Var\ x,\ Var\ x) \in par\text{-}rstep\ R$

3.2 Ground variants connecting to FORT

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

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

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

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

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

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

lemma *rstep-induct* [*consumes 1*, *case-names rule subst ctxt*]:
assumes (s, t) ∈ *rstep* R
and *rule*: $\bigwedge l\ r.\ (l,\ r) \in R \Longrightarrow P\ l\ r$
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 $P\ s\ t$
⟨*proof*⟩

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

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

lemma *rstep-ctxt* [*intro*]: (s, t) ∈ *rstep* R ⟹ ($C\langle s\rangle$, $C\langle t\rangle$) ∈ *rstep* R
⟨*proof*⟩

lemma *rstep-rule* [*intro*]: (l, r) ∈ R ⟹ (l, r) ∈ *rstep* R
⟨*proof*⟩

lemma *rstep-subst* [*intro*]: (s, t) ∈ *rstep* R ⟹ ($s \cdot \sigma$, $t \cdot \sigma$) ∈ *rstep* R
⟨*proof*⟩

lemma *nrrstep-def'*:
nrrstep R = $\{(s,\ t).\ \exists\, l\ r\ C\ \sigma.\ (l,\ r) \in R \wedge C \neq \Box \wedge s = C\langle l{\cdot}\sigma\rangle \wedge t = C\langle r{\cdot}\sigma\rangle\}$
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

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

lemma *rstep-imp-C-s-r*:
assumes (s,t) ∈ *rstep* R
shows $\exists\, C\ \sigma\ l\ r.\ (l,r) \in R \wedge s = C\langle l{\cdot}\sigma\rangle \wedge t = C\langle r{\cdot}\sigma\rangle$ ⟨*proof*⟩

lemma *rhs-wf*:

assumes *R*: (*l*, *r*) ∈ *R* **and** *funas-trs R* ⊆ *F*
shows *funas-term r* ⊆ *F*
⟨*proof*⟩

abbreviation *linear-sys* $\mathcal{R}$ ≡ (∀ (*l*, *r*) ∈ $\mathcal{R}$. *linear-term l* ∧ *linear-term r*)
abbreviation *const-subt c* ≡ λ *x*. *Fun c* []

end

4 Primitive constructions

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

4.1 Recognizing subterms of linear terms

abbreviation *ffunas-terms* **where**
ffunas-terms R ≡ |⋃| (*ffunas-term* |'| *R*)

definition *states R* ≡ {$t^{\perp}$ | *s t*. *s* ∈ *R* ∧ *s* ⊵ *t*}

lemma *states-conv*:
states R = *term-to-bot-term* ' (⋃ *s* ∈ *R*. *subterms s*)
⟨*proof*⟩

lemma *finite-states*:
assumes *finite R* **shows** *finite* (*states R*)
⟨*proof*⟩

lemma *root-bot-diff*:
root-bot ' (*R* − {*Bot*}) = (*root-bot* ' *R*) − {*None*}
⟨*proof*⟩

lemma *root-bot-states-root-subterms*:
the ' (*root-bot* ' (*states R* − {*Bot*})) = *the* ' (*root* ' (⋃ *s* ∈ *R*. *subterms s*) − {*None*})
⟨*proof*⟩

context
includes *fset.lifting*
begin

lift-definition *fstates* :: (′*f*, ′*v*) *term fset* ⇒ ′*f bot-term fset* **is** *states*
⟨*proof*⟩

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

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*
⟨*proof*⟩

lemma *fstates-def'*:
t |∈| *fstates* $R \longleftrightarrow (\exists\ s\ u.\ s$ |∈| $R \wedge s \trianglerighteq u \wedge u^{\perp} = t)$
⟨*proof*⟩

lemma *fstates-fmemberE* [*elim!*]:
assumes t |∈| *fstates* R
obtains $s\ u$ **where** s |∈| $R \wedge s \trianglerighteq u \wedge u^{\perp} = t$
⟨*proof*⟩

lemma *fstates-fmemberI* [*intro*]:
s |∈| $R \Longrightarrow s \trianglerighteq u \Longrightarrow u^{\perp}$ |∈| *fstates* R
⟨*proof*⟩

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

lemma *fstates*[*code*]:
fstates R = *term-to-bot-term* |'| (|⋃| (*fsubterms* |'| R))
⟨*proof*⟩

end

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

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

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

lemma *ta-rule-sig-term-to-ta-rule-root-set*:
assumes *Bot* |∉| R
shows *ta-rule-sig* |'| (*term-to-ta-rule* |'| R) = *the* |'| (*root-bot* |'| R)
⟨*proof*⟩

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

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

lemma *pattern-automaton-rules-BotD*:
assumes *TA-rule f ss Bot* |∈| *pattern-automaton-rules* $\mathcal{F}$ *R*
shows *TA-rule f ss Bot* |∈| (λ (*f*, *n*). *TA-rule f* (*replicate n Bot*) *Bot*) |'| $\mathcal{F}$
⟨*proof*⟩

lemma *pattern-automaton-rules-FunD*:
assumes *TA-rule f ss* (*BFun g ts*) |∈| *pattern-automaton-rules* $\mathcal{F}$ *R*
shows *g* = *f* ∧ *ts* = *ss* ∧
TA-rule f ss (*BFun g ts*) |∈| *term-to-ta-rule* |'| ((*fstates R*) − {|*Bot*|}) ⟨*proof*⟩

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

lemma *ta-sig-pattern-automaton* [*simp*]:
ta-sig (*pattern-automaton* $\mathcal{F}$ *R*) = $\mathcal{F}$ |∪| *ffunas-terms R*
⟨*proof*⟩

lemma *terms-reach-Bot*:
assumes *ffunas-gterm t* |⊆| $\mathcal{F}$
shows *Bot* |∈| *ta-der* (*pattern-automaton* $\mathcal{F}$ *R*) (*term-of-gterm t*) ⟨*proof*⟩

lemma *pattern-automaton-reach-smaller-term*:
assumes *l* |∈| *R l* ⊵ *s* $s^{\perp} \leq_b$ (*term-of-gterm t*)$^{\perp}$ *ffunas-gterm t* |⊆| $\mathcal{F}$
shows $s^{\perp}$ |∈| *ta-der* (*pattern-automaton* $\mathcal{F}$ *R*) (*term-of-gterm t*) ⟨*proof*⟩

lemma *bot-term-of-gterm-conv*:
term-of-gterm $s^{\perp}$ = *term-of-gterm* $s^{\perp}$
⟨*proof*⟩

lemma *pattern-automaton-ground-instance-reach*:
assumes *l* |∈| *R l* · σ = (*term-of-gterm t*) *ffunas-gterm t* |⊆| $\mathcal{F}$
shows $l^{\perp}$ |∈| *ta-der* (*pattern-automaton* $\mathcal{F}$ *R*) (*term-of-gterm t*)
⟨*proof*⟩

lemma *pattern-automaton-reach-smallet-term*:
assumes $l^{\perp}$ |∈| *ta-der* (*pattern-automaton* $\mathcal{F}$ *R*) *t ground t*
shows $l^{\perp} \leq_b t^{\perp}$ ⟨*proof*⟩

4.2 Recognizing root step relation of LV-TRSs

definition *lv-trs* :: ('*f*, '*v*) *trs* ⇒ *bool* **where**
lv-trs R ≡ ∀ (*l*, *r*) ∈ *R*. *linear-term l* ∧ *linear-term r* ∧ (*vars-term l* ∩ *vars-term r* = {})

lemma *subst-unification*:
  assumes *vars-term* $s \cap$ *vars-term* $t = \{\}$
  obtains μ **where** $s \cdot \sigma = s \cdot \mu$ $t \cdot \tau = t \cdot \mu$
  ⟨*proof*⟩

lemma *lv-trs-subst-unification*:
  assumes *lv-trs* R $(l, r) \in R$ $s = l \cdot \sigma$ $t = r \cdot \tau$
  obtains μ **where** $s = l \cdot \mu \wedge t = r \cdot \mu$
  ⟨*proof*⟩

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

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

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

lemma *agtt-grrstep-eps-trancl* [*simp*]:
  (*eps* (*fst* (*agtt-grrstep* $\mathcal{R}$ $\mathcal{F}$)))$|^+|$ = *eps* (*fst* (*agtt-grrstep* $\mathcal{R}$ $\mathcal{F}$))
  (*eps* (*snd* (*agtt-grrstep* $\mathcal{R}$ $\mathcal{F}$))) = {||}
  ⟨*proof*⟩

lemma *root-pair-automaton-grrstep*:
  fixes R :: ('*f*, '*v*) *rule fset*
  assumes *lv-trs* (*fset* R) *ffunas-trs* R |⊆| $\mathcal{F}$
  shows *pair-at-lang* (*root-pair-automaton* $\mathcal{F}$ R) (Rel_f R) = *Restr* (*grrstep* (*fset* R)) ($\mathcal{T}_G$ (*fset* $\mathcal{F}$)) (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *agtt-grrstep*:
  fixes R :: ('*f*, '*v*) *rule fset*
  assumes *lv-trs* (*fset* R) *ffunas-trs* R |⊆| $\mathcal{F}$
  shows *agtt-lang* (*agtt-grrstep* R $\mathcal{F}$) = *Restr* (*grrstep* (*fset* R)) ($\mathcal{T}_G$ (*fset* $\mathcal{F}$))
  ⟨*proof*⟩

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

lemma *agtt-grrstep-set*:
  fixes R :: ('*f*, '*v*) *rule set*

assumes *finite R finite* $\mathcal{F}$ *lv-trs R funas-trs R* $\subseteq \mathcal{F}$
shows *agtt-lang* (*agtt-grrstep* (*Abs-fset R*) (*Abs-fset* $\mathcal{F}$)) = *Restr* (*grrstep R*) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*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* λ *x y*. (*x*, *y*) $\in$ *mergeP* (↑) λ *x y*. $x \leq_b y$ *Bot*
⟨*proof*⟩

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

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

lemma *psubt-mono*:
$R \subseteq S \Longrightarrow$ *psubt-lhs-bot R* $\subseteq$ *psubt-lhs-bot S* ⟨*proof*⟩

lemma *states-mono*:
$R \subseteq S \Longrightarrow$ *states R* $\subseteq$ *states S*
⟨*proof*⟩

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

lemma *states-ref-closure*:
states R $\subseteq$ *insert Bot* (*closure* (*psubt-lhs-bot R*))
⟨*proof*⟩

lemma *finite-R-finite-states* [*simp*, *intro*]:
finite R $\Longrightarrow$ *finite* (*states R*)
⟨*proof*⟩

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

lemma *bound-max-state-set*:
  assumes *finite R*
  shows *bound-max t* (*psubt-lhs-bot R*) ∈ *states R*
  ⟨*proof*⟩

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

lemma *bound-max-state-fset*:
  bound-max t (*psubt-lhs-bot* (*fset R*)) |∈| *fstates R*
  ⟨*proof*⟩

end

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

lemma *nf-rules-fmember*:
  TA-rule f qs q |∈| *nf-rules R* $\mathcal{F}$ ⟷ (*f*, *length qs*) |∈| $\mathcal{F}$ ∧ *fset-of-list qs* |⊆| *fstates R* ∧
  ¬(∃ *l* |∈| *R*. $l^\perp \leq_b$ *BFun f qs*) ∧ *q* = *bound-max* (*BFun f qs*) (*psubt-lhs-bot* (*fset R*))
⟨*proof*⟩

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

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

lemma *bound-max-sound*:
  assumes *finite R*
  shows *bound-max t* (*psubt-lhs-bot R*) $\leq_b$ *t*
  ⟨*proof*⟩

lemma *Bot-in-filter*:
  Bot ∈ *Set.filter* (λ*s*. $s \leq_b t$) (*states R*)
  ⟨*proof*⟩

lemma *bound-max-exists*:
  ∃ *p*. *p* = *bound-max t* (*psubt-lhs-bot R*)
  ⟨*proof*⟩

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

lemma *nf-rule-to-bound-max*:
$f\ qs \to q\ |{\in}|\ \textit{nf-rules}\ R\ \mathcal{F} \Longrightarrow q = \textit{bound-max}\ (\textit{BFun}\ f\ qs)\ (\textit{psubt-lhs-bot}\ (\textit{fset}\ R))$
⟨*proof*⟩

lemma *nf-rules-unique*:
assumes $f\ qs \to q\ |{\in}|\ \textit{nf-rules}\ R\ \mathcal{F}$ **and** $f\ qs \to q'\ |{\in}|\ \textit{nf-rules}\ R\ \mathcal{F}$
shows $q = q'$ ⟨*proof*⟩

lemma *nf-ta-det*:
shows $\textit{ta-det}\ (\textit{nf-ta}\ R\ \mathcal{F})$
⟨*proof*⟩

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

lemma [*simp*]: $i < \textit{length}\ ss \Longrightarrow l \rhd \textit{Fun}\ f\ ss \Longrightarrow l \rhd ss\ !\ i$
⟨*proof*⟩

lemma *subt-less-eq-res-less-eq*:
assumes *ground*: $\textit{ground}\ t$ **and** $l\ |{\in}|\ R$ **and** $l \rhd s$ **and** $s^{\perp} \leq_b t^{\perp}$
and $q\ |{\in}|\ \textit{ta-der}\ (\textit{nf-ta}\ R\ \mathcal{F})\ (\textit{adapt-vars}\ t)$
shows $s^{\perp} \leq_b q$ ⟨*proof*⟩

lemma *ta-nf-sound1*:
assumes *ground*: $\textit{ground}\ t$ **and** *lhs*: $l\ |{\in}|\ R$ **and** *inst*: $l^{\perp} \leq_b t^{\perp}$
shows $\textit{ta-der}\ (\textit{nf-ta}\ R\ \mathcal{F})\ (\textit{adapt-vars}\ t) = \{||\}$
⟨*proof*⟩

lemma *ta-nf-tr-to-state*:
assumes $\textit{ground}\ t$ **and** $q\ |{\in}|\ \textit{ta-der}\ (\textit{nf-ta}\ R\ \mathcal{F})\ (\textit{adapt-vars}\ t)$
shows $q\ |{\in}|\ \textit{fstates}\ R$ ⟨*proof*⟩

lemma *ta-nf-sound2*:
assumes *linear*: $\forall\ l\ |{\in}|\ R.\ \textit{linear-term}\ l$
and $\textit{ground}\ (t :: ('f, 'v)\ \textit{term})$ **and** $\textit{funas-term}\ t \subseteq \textit{fset}\ \mathcal{F}$
and *NF*: $\bigwedge l\ s.\ l\ |{\in}|\ R \Longrightarrow t \unrhd s \Longrightarrow \neg\ l^{\perp} \leq_b s^{\perp}$
shows $\exists\ q.\ q\ |{\in}|\ \textit{ta-der}\ (\textit{nf-ta}\ R\ \mathcal{F})\ (\textit{adapt-vars}\ t)$ ⟨*proof*⟩

lemma *ta-nf-lang-sound*:
assumes $l\ |{\in}|\ R$
shows $C\langle l \cdot \sigma\rangle \notin \textit{ta-lang}\ (\textit{fstates}\ R)\ (\textit{nf-ta}\ R\ \mathcal{F})$
⟨*proof*⟩

lemma *ta-nf-lang-complete*:
assumes *linear*: $\forall$ *l* |$\in$| *R*. *linear-term l*
and *ground*: *ground* (*t* :: ($'f$, $'v$) *term*) **and** *sig*: *funas-term t* $\subseteq$ *fset* $\mathcal{F}$
and *nf*: $\bigwedge C\ \sigma\ l.\ l$ |$\in$| $R \Longrightarrow C\langle l{\cdot}\sigma\rangle \neq t$
shows $t \in$ *ta-lang* (*fstates R*) (*nf-ta R* $\mathcal{F}$)
⟨*proof*⟩

lemma *ta-nf-*$\mathcal{L}$*-complete*:
assumes *linear*: $\forall$ *l* |$\in$| *R*. *linear-term l*
and *sig*: *funas-gterm t* $\subseteq$ *fset* $\mathcal{F}$
and *nf*: $\bigwedge C\ \sigma\ l.\ l$ |$\in$| $R \Longrightarrow C\langle l{\cdot}\sigma\rangle \neq$ (*term-of-gterm t*)
shows $t \in \mathcal{L}$ (*nf-reg R* $\mathcal{F}$)
⟨*proof*⟩

lemma *nf-ta-funas*:
assumes *ground t q* |$\in$| *ta-der* (*nf-ta R* $\mathcal{F}$) *t*
shows *funas-term t* $\subseteq$ *fset* $\mathcal{F}$ ⟨*proof*⟩

lemma *gta-lang-nf-ta-funas*:
assumes $t \in \mathcal{L}$ (*nf-reg R* $\mathcal{F}$)
shows *funas-gterm t* $\subseteq$ *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* |$\in$| *ta-der′* $\mathcal{A}$ *s*
shows *fst* (*split-vars t*) $\leq$ (*mctxt-of-term s*) ⟨*proof*⟩

lemma *ta-der′-poss-subt-at-ta-der′*:
assumes *t* |$\in$| *ta-der′* $\mathcal{A}$ *s* **and** $p \in$ *poss t*
shows *t* |- *p* |$\in$| *ta-der′* $\mathcal{A}$ (*s* |- *p*) ⟨*proof*⟩

lemma *ta-der′-varposs-to-ta-der*:
assumes *t* |$\in$| *ta-der′* $\mathcal{A}$ *s* **and** $p \in$ *varposs t*
shows *the-Var* (*t* |- *p*) |$\in$| *ta-der* $\mathcal{A}$ (*s* |- *p*) ⟨*proof*⟩

definition *ta-der′-target-mctxt t* $\equiv$ *fst* (*split-vars t*)
definition *ta-der′-target-args t* $\equiv$ *snd* (*split-vars t*)
definition *ta-der′-source-args t s* $\equiv$ *unfill-holes* (*fst* (*split-vars t*)) *s*

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

lemma *ta-der′-target-mctxt-funas* [*simp*]:
funas-mctxt (*ta-der′-target-mctxt u*) = *funas-term u*

⟨*proof*⟩

lemma *ta-der′-target-mctxt-ground* [*simp*]:
ground-mctxt (*ta-der′-target-mctxt t*)
⟨*proof*⟩

lemma *ta-der′-source-args-ground*:
t |∈| *ta-der′* $\mathcal{A}$ $s \Longrightarrow$ *ground* $s \Longrightarrow \forall\ u \in$ *set* (*ta-der′-source-args t s*). *ground u*
⟨*proof*⟩

lemma *ta-der′-source-args-term-of-gterm*:
t |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) $\Longrightarrow \forall\ u \in$ *set* (*ta-der′-source-args t* (*term-of-gterm s*)). *ground u*
⟨*proof*⟩

lemma *ta-der′-source-args-length*:
t |∈| *ta-der′* $\mathcal{A}$ $s \Longrightarrow$ *num-holes* (*ta-der′-target-mctxt t*) = *length* (*ta-der′-source-args t s*)
⟨*proof*⟩

lemma *ta-der′-target-args-length*:
num-holes (*ta-der′-target-mctxt t*) = *length* (*ta-der′-target-args t*)
⟨*proof*⟩

lemma *ta-der′-target-args-vars-term-conv*:
vars-term t = *set* (*ta-der′-target-args t*)
⟨*proof*⟩

lemma *ta-der′-target-args-vars-term-list-conv*:
ta-der′-target-args t = *vars-term-list t*
⟨*proof*⟩

lemma *mctxt-args-ta-der′*:
assumes *num-holes C* = *length qs* *num-holes C* = *length ss*
and $\forall\ i <$ *length ss*. *qs* ! *i* |∈| *ta-der* $\mathcal{A}$ (*ss* ! *i*)
shows (*fill-holes C* (*map Var qs*)) |∈| *ta-der′* $\mathcal{A}$ (*fill-holes C ss*) ⟨*proof*⟩
lemma *ta-der′-mctxt-structure*:
assumes *t* |∈| *ta-der′* $\mathcal{A}$ *s*
shows *t* = *fill-holes* (*ta-der′-target-mctxt t*) (*map Var* (*ta-der′-target-args t*)) (**is** *?G1*)
s = *fill-holes* (*ta-der′-target-mctxt t*) (*ta-der′-source-args t s*) (**is** *?G2*)
num-holes (*ta-der′-target-mctxt t*) = *length* (*ta-der′-source-args t s*) ∧
length (*ta-der′-source-args t s*) = *length* (*ta-der′-target-args t*) (**is** *?G3*)
$i <$ *length* (*ta-der′-source-args t s*) $\Longrightarrow$ *ta-der′-target-args t* ! *i* |∈| *ta-der* $\mathcal{A}$ (*ta-der′-source-args t s* ! *i*)
⟨*proof*⟩

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

assumes t |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm s*)
shows t = *fill-holes* (*ta-der′-target-mctxt t*) (*map Var* (*ta-der′-target-args t*))
term-of-gterm s = *fill-holes* (*ta-der′-target-mctxt t*) (*ta-der′-source-args t* (*term-of-gterm s*))
num-holes (*ta-der′-target-mctxt t*) = *length* (*ta-der′-source-args t* (*term-of-gterm s*)) ∧
length (*ta-der′-source-args t* (*term-of-gterm s*)) = *length* (*ta-der′-target-args t*)
i < *length* (*ta-der′-target-args t*) ⟹ *ta-der′-target-args t* ! *i* |∈| *ta-der* $\mathcal{A}$ (*ta-der′-source-args t* (*term-of-gterm s*) ! *i*)
⟨*proof*⟩

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

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

lemma *ta-der′-ctxt-structure*:
assumes t |∈| *ta-der′* $\mathcal{A}$ *s vars-term-list t* = [*q*]
shows t = (*ta-der′-ctxt t*)⟨*Var q*⟩ (**is** *?G1*)
s = (*ta-der′-ctxt t*)⟨*ta-der′-source-ctxt-arg t s*⟩ (**is** *?G2*)
ground-ctxt (*ta-der′-ctxt t*) (**is** *?G3*)
q |∈| *ta-der* $\mathcal{A}$ (*ta-der′-source-ctxt-arg t s*) (**is** *?G4*)
⟨*proof*⟩

lemma *ta-der′-ground-ctxt-structure*:
assumes t |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) *vars-term-list t* = [*q*]
shows t = (*ta-der′-ctxt t*)⟨*Var q*⟩
s = (*ta-der′-gctxt t*)⟨*ta-der′-source-gctxt-arg t s*⟩$_G$
ground (*ta-der′-source-ctxt-arg t* (*term-of-gterm s*))
q |∈| *ta-der* $\mathcal{A}$ (*ta-der′-source-ctxt-arg t* (*term-of-gterm s*))
⟨*proof*⟩

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

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

abbreviation Δ_A ≡ *rules* $\mathcal{A}$
abbreviation $\Delta_{\mathcal{E}A}$ ≡ *eps* $\mathcal{A}$
abbreviation Δ_B ≡ *rules* $\mathcal{B}$

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

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

lemma *states-split* [*simp*]: $\mathcal{Q}\ A = \mathcal{Q}\ \mathcal{A}\ |\cup|\ \mathcal{Q}\ \mathcal{B}$
⟨*proof*⟩

lemma *A-args-states-not-B*:
TA-rule f qs q $|\in|\ \Delta_A \Longrightarrow p\ |\in|$ *fset-of-list qs* $\Longrightarrow p\ |\notin|\ \mathcal{Q}_B$
⟨*proof*⟩

lemma *rule-statesD*:
$r\ |\in|\ \Delta_A \Longrightarrow$ *r-rhs* $r\ |\in|\ \mathcal{Q}_A$
$r\ |\in|\ \Delta_B \Longrightarrow$ *r-rhs* $r\ |\in|\ \mathcal{Q}_B$
$r\ |\in|\ \Delta_A \Longrightarrow p\ |\in|$ *fset-of-list* (*r-lhs-states* r) $\Longrightarrow p\ |\in|\ \mathcal{Q}_A$
TA-rule f qs q $|\in|\ \Delta_A \Longrightarrow q\ |\in|\ \mathcal{Q}_A$
TA-rule f qs q $|\in|\ \Delta_B \Longrightarrow q\ |\in|\ \mathcal{Q}_B$
TA-rule f qs q $|\in|\ \Delta_A \Longrightarrow p\ |\in|$ *fset-of-list qs* $\Longrightarrow p\ |\in|\ \mathcal{Q}_A$
⟨*proof*⟩

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

lemma *transcl-eps-simp*:
(*eps A*)$|^+|$ = $\Delta_{\mathcal{E}A}|^+|\ |\cup|\ \Delta_{\mathcal{E}B}|^+|\ |\cup|\ (\Delta_{\mathcal{E}A}|^+|\ |O|\ \Delta_{\mathcal{E}B}|^+|)$
⟨*proof*⟩

lemma *B-rule-eps-A-False*:
f qs $\rightarrow q\ |\in|\ \Delta_B \Longrightarrow (q,\ p)\ |\in|\ \Delta_{\mathcal{E}A}|^+| \Longrightarrow$ *False*
⟨*proof*⟩

lemma *to-A-rule-set*:
assumes *TA-rule f qs q* $|\in|$ *rules A* **and** $q = p \lor (q,\ p)\ |\in|$ (*eps A*)$|^+|$ **and** $p\ |\notin|$ $\mathcal{Q}_B$
shows *TA-rule f qs q* $|\in|\ \Delta_A\ q = p \lor (q,\ p)\ |\in|\ \Delta_{\mathcal{E}A}|^+|$ ⟨*proof*⟩

lemma *to-B-rule-set*:
assumes *TA-rule f qs q* $|\in|$ *rules A* **and** $q\ |\notin|\ \mathcal{Q}_A$
shows *TA-rule f qs q* $|\in|\ \Delta_B$ ⟨*proof*⟩

declare *fsubsetI*[*rule del*]
lemma *ta-der-monos*:
  ta-der $\mathcal{A}$ *t* |⊆| *ta-der A t ta-der* $\mathcal{B}$ *t* |⊆| *ta-der A t*
  ⟨*proof*⟩
declare *fsubsetI*[*intro*!]

lemma *ta-der-from-*Δ_A:
  assumes *q* |∈| *ta-der A* (*term-of-gterm t*) **and** *q* |∉| $\mathcal{Q}_B$
  shows *q* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm t*) ⟨*proof*⟩

lemma *ta-state*:
  assumes *q* |∈| *ta-der A* (*term-of-gterm s*)
  shows *q* |∈| $\mathcal{Q}_A$ ∨ *q* |∈| $\mathcal{Q}_B$ ⟨*proof*⟩

lemma *ta-der-split*:
  assumes *q* |∈| *ta-der A* (*term-of-gterm s*) **and** *q* |∈| $\mathcal{Q}_B$
  shows ∃ *t*. *t* |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) ∧ *q* |∈| *ta-der* $\mathcal{B}$ *t*
    (**is** ∃ *t* . *?P s q t*) ⟨*proof*⟩

lemma *ta-der′-split*:
  assumes *t* |∈| *ta-der′ A* (*term-of-gterm s*)
  shows ∃ *u*. *u* |∈| *ta-der′* $\mathcal{A}$ (*term-of-gterm s*) ∧ *t* |∈| *ta-der′* $\mathcal{B}$ *u*
    (**is** ∃ *u*. *?P s t u*) ⟨*proof*⟩

lemma *ta-der-to-mcxtx*:
  assumes *q* |∈| *ta-der A* (*term-of-gterm s*) **and** *q* |∈| $\mathcal{Q}_B$
  shows ∃ *C ss qs*. *length qs* = *length ss* ∧ *num-holes C* = *length ss* ∧
    (∀ *i* < *length ss*. *qs* ! *i* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm* (*ss* ! *i*))) ∧
    q |∈| *ta-der* $\mathcal{B}$ (*fill-holes C* (*map Var qs*)) ∧
    ground-mctxt C ∧ *fill-holes C* (*map term-of-gterm ss*) = *term-of-gterm s*
    (**is** ∃ *C ss qs*. *?P s q C ss qs*)
⟨*proof*⟩

lemma *ta-der-to-gmcxtx*:
  assumes *q* |∈| *ta-der A* (*term-of-gterm s*) **and** *q* |∈| $\mathcal{Q}_B$
  shows ∃ *C ss qs qs′*. *length qs′* = *length qs* ∧ *length qs* = *length ss* ∧ *num-gholes C* = *length ss* ∧
    (∀ *i* < *length ss*. *qs* ! *i* |∈| *ta-der* $\mathcal{A}$ (*term-of-gterm* (*ss* ! *i*))) ∧
    q |∈| *ta-der* $\mathcal{B}$ (*fill-holes* (*mctxt-of-gmctxt C*) (*map Var qs′*)) ∧
    fill-gholes C ss = *s*
  ⟨*proof*⟩

lemma *mctxt-const-to-ta-der*:
  assumes *num-holes* C = *length ss length ss* = *length qs*
    and ∀ i < *length qs*. *qs* ! i |∈| *ta-der* $\mathcal{A}$ (*ss* ! i)
    and q |∈| *ta-der* $\mathcal{B}$ (*fill-holes* C (*map Var qs*))
  shows q |∈| *ta-der* A (*fill-holes* C *ss*)
⟨*proof*⟩

lemma *ctxt-const-to-ta-der*:
  assumes q |∈| *ta-der* $\mathcal{A}$ s
    and p |∈| *ta-der* $\mathcal{B}$ C⟨*Var* q⟩
  shows p |∈| *ta-der* A C⟨s⟩ ⟨*proof*⟩

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

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

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

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

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

5.1.2 Multihole context closure over a given signature

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

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

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

5.1.3 Context closure of regular tree language

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

definition *reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f ≡
TA (*reflcl-rules* $\mathcal{F}$ q_c |∪| *semantic-path-rules* $\mathcal{F}$ q_c q_f q_f) ((λ p. (p, q_f)) |'| Q)

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

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

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

5.1.4 Not empty context closure of regular tree language

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

definition *reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q_c q_i q_f ≡
TA (*reflcl-rules* $\mathcal{F}$ q_c |∪| *semantic-path-rules* $\mathcal{F}$ q_c q_i q_f |∪| *semantic-path-rules* $\mathcal{F}$ q_c q_f q_f) ((λ p. (p, q_i)) |'| Q)

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

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

5.1.5 Non empty multihole context closure of regular tree language

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

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

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

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

5.1.6 Not empty multihole context closure of regular tree language

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

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

5.1.7 Multihole context closure of regular tree language

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

5.1.8 Lemmas about *ta-der'*

lemma *ta-det'-ground-id*:
$t\ |\in|$ *ta-der'* $\mathcal{A}\ s \Longrightarrow$ *ground* $t \Longrightarrow t = s$
⟨*proof*⟩

lemma *ta-det'-vars-term-id*:
$t\ |\in|$ *ta-der'* $\mathcal{A}\ s \Longrightarrow$ *vars-term* $t \cap$ *fset* ($\mathcal{Q}\ \mathcal{A}$) = {} $\Longrightarrow t = s$
⟨*proof*⟩

lemma *fresh-states-ta-der'-pres*:
assumes *st*: $q \in$ *vars-term* s $q\ |\notin|\ \mathcal{Q}\ \mathcal{A}$
and *reach*: $t\ |\in|$ *ta-der'* $\mathcal{A}\ s$
shows $q \in$ *vars-term* t ⟨*proof*⟩

lemma *ta-der'-states*:
$t\ |\in|$ *ta-der'* $\mathcal{A}\ s \Longrightarrow$ *vars-term* $t \subseteq$ *vars-term* $s\ \cup$ *fset* ($\mathcal{Q}\ \mathcal{A}$)
⟨*proof*⟩

lemma *ta-der'-gterm-states*:
t |∈| *ta-der'* $\mathcal{A}$ (*term-of-gterm* s) $\Longrightarrow$ *vars-term* $t \subseteq$ *fset* ($\mathcal{Q}$ $\mathcal{A}$)
⟨*proof*⟩

lemma *ta-der'-Var-funas*:
Var q |∈| *ta-der'* $\mathcal{A}$ $s \Longrightarrow$ *funas-term* $s \subseteq$ *fset* (*ta-sig* $\mathcal{A}$)
⟨*proof*⟩

lemma *ta-sig-fsubsetI*:
assumes $\bigwedge r.$ r |∈| *rules* $\mathcal{A} \Longrightarrow$ (*r-root* r, *length* (*r-lhs-states* r)) |∈| $\mathcal{F}$
shows *ta-sig* $\mathcal{A}$ |⊆| $\mathcal{F}$ ⟨*proof*⟩

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

lemma *refl-ta-sig* [*simp*]:
ta-sig (*refl-ta* $\mathcal{F}$ q) = $\mathcal{F}$
ta-sig (*refl-over-states-ta* Q $\mathcal{F}$ $\mathcal{A}$ q) = $\mathcal{F}$
⟨*proof*⟩

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

lemma *refl-ta-eps* [*simp*]: *eps* (*refl-ta* $\mathcal{F}$ q) = {||}
⟨*proof*⟩

lemma *refl-ta-sound*:
$s \in \mathcal{T}_G$ (*fset* $\mathcal{F}$) $\Longrightarrow$ q |∈| *ta-der* (*refl-ta* $\mathcal{F}$ q) (*term-of-gterm* s)
⟨*proof*⟩

lemma *reflcl-rules-args*:
length $ps = n \Longrightarrow f\ ps \rightarrow p$ |∈| *reflcl-rules* $\mathcal{F}$ $q \Longrightarrow ps =$ *replicate* n q
⟨*proof*⟩

lemma *Q-refl-ta*:
$\mathcal{Q}$ (*refl-ta* $\mathcal{F}$ q) |⊆| {|q|}
⟨*proof*⟩

lemma *refl-ta-complete1*:
Var p |∈| *ta-der'* (*refl-ta* $\mathcal{F}$ q) $s \Longrightarrow p \neq q \Longrightarrow s =$ *Var* p
⟨*proof*⟩

lemma *refl-ta-complete2*:
Var q |∈| *ta-der'* (*refl-ta* $\mathcal{F}$ q) $s \Longrightarrow$ *funas-term* $s \subseteq$ *fset* $\mathcal{F} \land$ *vars-term* $s \subseteq \{q\}$
⟨*proof*⟩

lemma *gen-reflcl-lang*:
assumes q |∉| $\mathcal{Q}$ $\mathcal{A}$
shows *gta-lang* (*finsert* q Q) (*gen-reflcl-automaton* $\mathcal{F}$ $\mathcal{A}$ q) = *gta-lang* Q $\mathcal{A}$ $\cup$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)

(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *reflcl-lang*:
gta-lang (*finsert None* (*Some* |'| *Q*)) (*reflcl-automaton* $\mathcal{F}$ $\mathcal{A}$) = *gta-lang* *Q* $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*fset* $\mathcal{F}$)
⟨*proof*⟩

lemma $\mathcal{L}$*-reflcl-reg*:
$\mathcal{L}$ (*reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$ ∪ $\mathcal{T}_G$ (*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 ⊆ *A* ⟹ *i* < *length xs* ⟹ *xs* ! *i* ∈ *A*
⟨*proof*⟩

lemma *ground-gmctxt-of-mctxt-fill-holes'*:
num-holes C = *length ss* ⟹ *ground-mctxt C* ⟹ ∀ *s*∈*set ss*. *ground s* ⟹
fill-gholes (*gmctxt-of-mctxt C*) (*map gterm-of-term ss*) = *gterm-of-term* (*fill-holes* *C ss*)
⟨*proof*⟩

lemma *refl-over-states-ta-eps-trancl* [*simp*]:
(*eps* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*))|$^+$| = *eps* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*)
⟨*proof*⟩

lemma *refl-over-states-ta-epsD*:
(*p*, *q*) |∈| (*eps* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*)) ⟹ *p* |∈| *Q*
⟨*proof*⟩

lemma *refl-over-states-ta-vars-term*:
q |∈| *ta-der* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*) *u* ⟹ *vars-term u* ⊆ *insert q* (*fset Q*)
⟨*proof*⟩

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

lemma *refl-over-states-ta-sound*:
funas-term u ⊆ *fset* $\mathcal{F}$ ⟹ *vars-term u* ⊆ *insert q* (*fset* (*Q* |∩| $\mathcal{Q}$ $\mathcal{A}$)) ⟹ *q* |∈| *ta-der* (*refl-over-states-ta Q* $\mathcal{F}$ $\mathcal{A}$ *q*) *u*
⟨*proof*⟩

lemma *gen-parallelcl-lang*:
fixes $\mathcal{A}$ :: (*'q*, *'f*) *ta*
assumes *q* |∉| $\mathcal{Q}$ $\mathcal{A}$

shows *gta-lang* $\{|q|\}$ (*gen-parallel-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q) =
{*fill-gholes* C *ss* | C *ss*. *num-gholes* C = *length ss* ∧ *funas-gmctxt* $C \subseteq$ (*fset* $\mathcal{F}$) ∧ (∀ $i <$ *length ss*. *ss* ! $i \in$ *gta-lang* Q $\mathcal{A}$)}
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *parallelcl-gmctxt-lang*:
fixes $\mathcal{A}$ :: ('*q*, '*f*) *reg*
shows $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{*fill-gholes* C *ss* |
C *ss*. *num-gholes* C = *length ss* ∧ *funas-gmctxt* $C \subseteq$ *fset* $\mathcal{F}$ ∧ (∀ $i <$ *length ss*. *ss* ! $i \in \mathcal{L}$ $\mathcal{A}$)}
⟨*proof*⟩

lemma *parallelcl-mctxt-lang*:
shows $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{(*gterm-of-term* :: ('*f*, '*q option*) *term* ⇒ '*f gterm*) (*fill-holes* C (*map term-of-gterm ss*)) |
C *ss*. *num-holes* C = *length ss* ∧ *ground-mctxt* C ∧ *funas-mctxt* $C \subseteq$ *fset* $\mathcal{F}$ ∧ (∀ $i <$ *length ss*. *ss* ! $i \in \mathcal{L}$ $\mathcal{A}$)}
⟨*proof*⟩

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

lemma *semantic-path-rules-rhs*:
r |∈| *semantic-path-rules* Q q_c q_i q_f ⟹ *r-rhs* r = q_f
⟨*proof*⟩

lemma *reflcl-over-single-ta-transl* [*simp*]:
(*eps* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f))$|^+|$ = *eps* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f)
⟨*proof*⟩

lemma *reflcl-over-single-ta-epsD*:
(p, q_f) |∈| *eps* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) ⟹ p |∈| Q
(p, q) |∈| *eps* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) ⟹ $q = q_f$
⟨*proof*⟩

lemma *reflcl-over-single-ta-rules-split*:
r |∈| *rules* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f) ⟹
r |∈| *reflcl-rules* $\mathcal{F}$ q_c ∨ r |∈| *semantic-path-rules* $\mathcal{F}$ q_c q_f q_f
⟨*proof*⟩

lemma *reflcl-over-single-ta-rules-semantic-path-rulesI*:
r |∈| *semantic-path-rules* $\mathcal{F}$ q_c q_f q_f ⟹ r |∈| *rules* (*reflcl-over-single-ta* Q $\mathcal{F}$ q_c q_f)
⟨*proof*⟩

lemma *semantic-path-rules-fmember* [*intro*]:
TA-rule f qs q |∈| *semantic-path-rules* $\mathcal{F}$ q_c q_i q_f ⟷ (∃ n i. (f, n) |∈| $\mathcal{F}$ ∧ $i <$

$n \wedge q = q_f \wedge$
$(qs = (replicate\ n\ q_c)[i := q_i]))$ (**is** *?Ls* $\longleftrightarrow$ *?Rs*)
⟨*proof*⟩

lemma *semantic-path-rules-fmemberD*:
$r\ |\in|\ semantic\text{-}path\text{-}rules\ \mathcal{F}\ q_c\ q_i\ q_f \Longrightarrow (\exists\ n\ i.\ (r\text{-}root\ r,\ n)\ |\in|\ \mathcal{F} \wedge i < n \wedge$
$r\text{-}rhs\ r = q_f \wedge$
$(r\text{-}lhs\text{-}states\ r = (replicate\ n\ q_c)[i := q_i]))$
⟨*proof*⟩

lemma *reflcl-over-single-ta-vars-term-*q_c:
$q_c \neq q_f \Longrightarrow q_c\ |\in|\ ta\text{-}der\ (reflcl\text{-}over\text{-}single\text{-}ta\ Q\ \mathcal{F}\ q_c\ q_f)\ u \Longrightarrow$
$vars\text{-}term\text{-}list\ u = replicate\ (length\ (vars\text{-}term\text{-}list\ u))\ q_c$
⟨*proof*⟩

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

lemma *refl-ta-reflcl-over-single-ta-mono*:
$q\ |\in|\ ta\text{-}der\ (refl\text{-}ta\ \mathcal{F}\ q)\ t \Longrightarrow q\ |\in|\ ta\text{-}der\ (reflcl\text{-}over\text{-}single\text{-}ta\ Q\ \mathcal{F}\ q\ q_f)\ t$
⟨*proof*⟩

lemma *reflcl-over-single-ta-sound*:
assumes $funas\text{-}gctxt\ C \subseteq fset\ \mathcal{F}\ q\ |\in|\ Q$
shows $q_f\ |\in|\ ta\text{-}der\ (reflcl\text{-}over\text{-}single\text{-}ta\ Q\ \mathcal{F}\ q_c\ q_f)\ (ctxt\text{-}of\text{-}gctxt\ C)\langle Var\ q\rangle$
⟨*proof*⟩

lemma *reflcl-over-single-ta-sig*: $ta\text{-}sig\ (reflcl\text{-}over\text{-}single\text{-}ta\ Q\ \mathcal{F}\ q_c\ q_f)\ |\subseteq|\ \mathcal{F}$
⟨*proof*⟩

lemma *gen-gctxtcl-lang*:
assumes $q_c\ |\notin|\ \mathcal{Q}\ \mathcal{A}$ **and** $q_f\ |\notin|\ \mathcal{Q}\ \mathcal{A}$ **and** $q_c\ |\notin|\ Q$ **and** $q_c \neq q_f$
shows $gta\text{-}lang\ \{|q_f|\}\ (gen\text{-}ctxt\text{-}closure\text{-}automaton\ Q\ \mathcal{F}\ \mathcal{A}\ q_c\ q_f) =$
$\{C\langle s\rangle_G \mid C\ s.\ funas\text{-}gctxt\ C \subseteq fset\ \mathcal{F} \wedge s \in gta\text{-}lang\ Q\ \mathcal{A}\}$
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *gen-gctxt-closure-sound*:
fixes $\mathcal{A}$ :: *('q, 'f) reg*
assumes $q_c\ |\notin|\ \mathcal{Q}_r\ \mathcal{A}$ **and** $q_f\ |\notin|\ \mathcal{Q}_r\ \mathcal{A}$ **and** $q_c\ |\notin|\ fin\ \mathcal{A}$ **and** $q_c \neq q_f$
shows $\mathcal{L}\ (gen\text{-}ctxt\text{-}closure\text{-}reg\ \mathcal{F}\ \mathcal{A}\ q_c\ q_f) = \{C\langle s\rangle_G \mid C\ s.\ funas\text{-}gctxt\ C \subseteq fset$
$\mathcal{F} \wedge s \in \mathcal{L}\ \mathcal{A}\}$
⟨*proof*⟩

lemma *gen-ctxt-closure-sound*:

fixes $\mathcal{A}$ :: ('q, 'f) *reg*
assumes q_c |∉| $\mathcal{Q}_r$ $\mathcal{A}$ **and** q_f |∉| $\mathcal{Q}_r$ $\mathcal{A}$ **and** q_c |∉| *fin* $\mathcal{A}$ **and** $q_c \neq q_f$
shows $\mathcal{L}$ (*gen-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ q_c q_f) =
{(*gterm-of-term* :: ('f, 'q) *term* ⇒ 'f *gterm*) C⟨*term-of-gterm s*⟩ | C s. *ground-ctxt* C ∧ *funas-ctxt* C ⊆ *fset* $\mathcal{F}$ ∧ s ∈ $\mathcal{L}$ $\mathcal{A}$}
⟨*proof*⟩

lemma *gctxt-closure-lang*:
shows $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{ $C\langle s\rangle_G$ | C s. *funas-gctxt* C ⊆ *fset* $\mathcal{F}$ ∧ s ∈ $\mathcal{L}$ $\mathcal{A}$}
⟨*proof*⟩

lemma *ctxt-closure-lang*:
shows $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{(*gterm-of-term* :: ('f, 'q + *bool*) *term* ⇒ 'f *gterm*) C⟨*term-of-gterm s*⟩ |
C s. *ground-ctxt* C ∧ *funas-ctxt* C ⊆ *fset* $\mathcal{F}$ ∧ s ∈ $\mathcal{L}$ $\mathcal{A}$}
⟨*proof*⟩

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

lemma *reflcl-over-nhole-ctxt-ta-vars-term-*q_c:
$q_c \neq q_f \Longrightarrow q_c \neq q_i \Longrightarrow q_c$ |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) u ⟹
vars-term-list u = *replicate* (*length* (*vars-term-list* u)) q_c
⟨*proof*⟩

lemma *reflcl-over-nhole-ctxt-ta-vars-term-Var*:
assumes *disj*: q_c |∉| Q q_f |∉| Q $q_c \neq q_f$ $q_i \neq q_f$ $q_c \neq q_i$
and *reach*: q_i |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) u
shows (∃ q. q |∈| *finsert* q_i Q ∧ u = *Var* q) ⟨*proof*⟩

lemma *reflcl-over-nhole-ctxt-ta-vars-term*:
assumes *disj*: q_c |∉| Q q_f |∉| Q $q_c \neq q_f$ $q_i \neq q_f$ $q_c \neq q_i$
and *reach*: q_f |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) u
shows (∃ i q. i < *length* (*vars-term-list* u) ∧ q |∈| {|q_i, q_f|} |∪| Q ∧ *vars-term-list* u = (*replicate* (*length* (*vars-term-list* u)) q_c)[i := q])
⟨*proof*⟩

lemma *reflcl-over-nhole-ctxt-ta-mono*:
q |∈| *ta-der* (*refl-ta* $\mathcal{F}$ q) t ⟹ q |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q q_i q_f) t
⟨*proof*⟩

lemma *reflcl-over-nhole-ctxt-ta-sound*:
assumes *funas-gctxt* C ⊆ *fset* $\mathcal{F}$ C ≠ *GHole* q |∈| Q
shows q_f |∈| *ta-der* (*reflcl-over-nhole-ctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) (*ctxt-of-gctxt* C)⟨*Var* q⟩ ⟨*proof*⟩

lemma *reflcl-over-nhole-ctxt-ta-sig*: $\textit{ta-sig}$ $(\textit{reflcl-over-nhole-ctxt-ta}\ Q\ \mathcal{F}\ q_c\ q_i\ q_f)$ $|\subseteq|\ \mathcal{F}$
⟨*proof*⟩

lemma *gen-nhole-gctxt-closure-lang*:
assumes $q_c\ |\notin|\ \mathcal{Q}\ \mathcal{A}\ q_i\ |\notin|\ \mathcal{Q}\ \mathcal{A}\ q_f\ |\notin|\ \mathcal{Q}\ \mathcal{A}$
and $q_c\ |\notin|\ Q\ q_f\ |\notin|\ Q$
and $q_c \neq q_i\ q_c \neq q_f\ q_i \neq q_f$
shows *gta-lang* $\{|q_f|\}$ (*gen-nhole-ctxt-closure-automaton* $Q\ \mathcal{F}\ \mathcal{A}\ q_c\ q_i\ q_f$) =
$\{C\langle s\rangle_G \mid C\ s.\ C \neq \textit{GHole} \wedge \textit{funas-gctxt}\ C \subseteq \textit{fset}\ \mathcal{F} \wedge s \in \textit{gta-lang}\ Q\ \mathcal{A}\}$
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *gen-nhole-gctxt-closure-sound*:
assumes $q_c\ |\notin|\ \mathcal{Q}_r\ \mathcal{A}\ q_i\ |\notin|\ \mathcal{Q}_r\ \mathcal{A}\ q_f\ |\notin|\ \mathcal{Q}_r\ \mathcal{A}$
and $q_c\ |\notin|\ (\textit{fin}\ \mathcal{A})\ q_f\ |\notin|\ (\textit{fin}\ \mathcal{A})$
and $q_c \neq q_i\ q_c \neq q_f\ q_i \neq q_f$
shows $\mathcal{L}$ (*gen-nhole-ctxt-closure-reg* $\mathcal{F}\ \mathcal{A}\ q_c\ q_i\ q_f$) =
$\{\ C\langle s\rangle_G \mid C\ s.\ C \neq \textit{GHole} \wedge \textit{funas-gctxt}\ C \subseteq \textit{fset}\ \mathcal{F} \wedge s \in \mathcal{L}\ \mathcal{A}\}$
⟨*proof*⟩

lemma *nhole-ctxtcl-lang*:
$\mathcal{L}$ (*nhole-ctxt-closure-reg* $\mathcal{F}\ \mathcal{A}$) =
$\{\ C\langle s\rangle_G \mid C\ s.\ C \neq \textit{GHole} \wedge \textit{funas-gctxt}\ C \subseteq \textit{fset}\ \mathcal{F} \wedge s \in \mathcal{L}\ \mathcal{A}\}$
⟨*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*:
$f\ ps \to q\ |\in|\ \textit{reflcl-rules}\ \mathcal{F}\ q_c \Longrightarrow q = q_c$
⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-vars-term*:
assumes $q\ |\in|\ \textit{ta-der}\ (\textit{reflcl-over-nhole-mctxt-ta}\ Q\ \mathcal{F}\ q_c\ q_i\ q_f)\ t$
and $q_c\ |\notin|\ Q\ q \neq q_c\ q_f \neq q_c\ q_i \neq q_c$
shows *vars-term* $t \neq \{\}$ ⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-Fun*:
assumes $q_f\ |\in|\ \textit{ta-der}\ (\textit{reflcl-over-nhole-mctxt-ta}\ Q\ \mathcal{F}\ q_c\ q_i\ q_f)\ t\ t \neq \textit{Var}\ q_f$
and $q_f \neq q_c\ q_f \neq q_i$
shows *is-Fun* t ⟨*proof*⟩

lemma *rule-states-reflcl-rulesD*:
$p\ |\in|\ \textit{rule-states}\ (\textit{reflcl-rules}\ \mathcal{F}\ q) \Longrightarrow p = q$
⟨*proof*⟩

lemma *rule-states-semantic-path-rulesD*:
p |∈| *rule-states* (*semantic-path-rules* $\mathcal{F}$ q_c q_i q_f) $\Longrightarrow p = q_c \lor p = q_i \lor p = q_f$
⟨*proof*⟩

lemma $\mathcal{Q}$*-reflcl-over-nhole-mctxt-ta*:
$\mathcal{Q}$ (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) |⊆| Q |∪| {|q_c, q_i, q_f|}
⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-vars-term-subset-eq*:
assumes q |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) t $q = q_f \lor q = q_i$
shows *vars-term* $t \subseteq \{q_c, q_i, q_f\} \cup$ *fset* Q
⟨*proof*⟩

lemma *sig-reflcl-over-nhole-mctxt-ta* [*simp*]:
ta-sig (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) $= \mathcal{F}$
⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-aux-sound*:
assumes *funas-term* $t \subseteq$ *fset* $\mathcal{F}$ *vars-term* $t \subseteq$ *fset* Q
shows q_c |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) t ⟨*proof*⟩

lemma *reflcl-over-nhole-mctxt-ta-sound*:
assumes *funas-term* $t \subseteq$ *fset* $\mathcal{F}$ *vars-term* $t \subseteq$ *fset* Q *vars-term* $t \neq \{\}$
shows (*is-Var* $t \longrightarrow q_i$ |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) t) $\land$
(*is-Fun* $t \longrightarrow q_f$ |∈| *ta-der* (*reflcl-over-nhole-mctxt-ta* Q $\mathcal{F}$ q_c q_i q_f) t) ⟨*proof*⟩

lemma *gen-nhole-gmctxt-closure-lang*:
assumes q_c |∉| $\mathcal{Q}$ $\mathcal{A}$ **and** q_i |∉| $\mathcal{Q}$ $\mathcal{A}$ q_f |∉| $\mathcal{Q}$ $\mathcal{A}$
and q_c |∉| Q $q_f \neq q_c$ $q_f \neq q_i$ $q_i \neq q_c$
shows *gta-lang* {|q_f|} (*gen-nhole-mctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f) =
{ *fill-gholes* C *ss* |
C *ss*. $0 <$ *num-gholes* $C \land$ *num-gholes* $C =$ *length ss* $\land C \neq$ *GMHole* $\land$
funas-gmctxt $C \subseteq$ *fset* $\mathcal{F} \land (\forall\ i <$ *length ss*. *ss* ! $i \in$ *gta-lang* Q $\mathcal{A}$)}
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *nhole-gmctxt-closure-lang*:
$\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{ *fill-gholes* C *ss* | C *ss*. *num-gholes* $C =$ *length ss* $\land 0 <$ *num-gholes* $C \land C \neq$ *GMHole* $\land$
funas-gmctxt $C \subseteq$ *fset* $\mathcal{F} \land (\forall\ i <$ *length ss*. *ss* ! $i \in \mathcal{L}$ $\mathcal{A}$)}
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

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

lemma *gen-gmctxt-closure-lang*:

assumes q_c |∉| $\mathcal{Q}$ $\mathcal{A}$ **and** q_i |∉| $\mathcal{Q}$ $\mathcal{A}$ q_f |∉| $\mathcal{Q}$ $\mathcal{A}$
and *disj*: q_c |∉| Q $q_f \neq q_c$ $q_f \neq q_i$ $q_i \neq q_c$
shows *gta-lang* {|q_f, q_i|} (*gen-nhole-mctxt-closure-automaton* Q $\mathcal{F}$ $\mathcal{A}$ q_c q_i q_f)
=
{ *fill-gholes C ss* |
C ss. 0 < *num-gholes C* ∧ *num-gholes C* = *length ss* ∧
funas-gmctxt C ⊆ *fset* $\mathcal{F}$ ∧ (∀ *i* < *length ss. ss* ! *i* ∈ *gta-lang Q* $\mathcal{A}$)}
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *gmctxt-closure-lang*:
$\mathcal{L}$ (*mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) =
{ *fill-gholes C ss* | *C ss. num-gholes C* = *length ss* ∧ *0* < *num-gholes C* ∧
funas-gmctxt C ⊆ *fset* $\mathcal{F}$ ∧ (∀ *i* < *length ss. ss* ! *i* ∈ $\mathcal{L}$ $\mathcal{A}$)}
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

5.1.16 Correctness of *nhole-mctxt-reflcl-reg*

lemma *nhole-mctxt-reflcl-lang*:
$\mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) ∪ $\mathcal{T}_G$ (*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* ≡ [*cl-state*, *tr-state*, *fin-state*, *fin-clstate*]
definition *cEnum-cl-states* :: (*cl-states list* × ((*cl-states* ⇒ *bool*) ⇒ *bool*) × ((*cl-states* ⇒ *bool*) ⇒ *bool*)) *option*
where *cEnum-cl-states* = *Some* (*cl-all-list*, (λ *P*. *list-all P cl-all-list*), (λ *P*. *list-ex P cl-all-list*))
instance
⟨*proof*⟩
end

lemma *infinite-bot-term-UNIV*[*simp*, *intro*]: *infinite* (*UNIV* :: ′*f bot-term set*)
⟨*proof*⟩

lemma *finite-cl-states*: (*UNIV* :: *cl-states set*) = {*cl-state*, *tr-state*, *fin-state*, *fin-clstate*}
⟨*proof*⟩

instantiation *cl-states* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*cl-states*) *True*
definition *card-UNIV* = *Phantom*(*cl-states*) *4*
instance
⟨*proof*⟩
end

instantiation *bot-term* :: (*type*) *finite-UNIV*
begin
definition *finite-UNIV* = *Phantom*(′*a bot-term*) *False*
instance
⟨*proof*⟩
end

instantiation *bot-term* :: (*compare*) *cproper-interval*
begin
definition *cproper-interval* = (λ (- :: ′*a bot-term option*) - . *False*)
instance ⟨*proof*⟩
end

instantiation *cl-states* :: *cproper-interval*
begin

definition *cproper-interval-cl-states* :: *cl-states option* ⇒ *cl-states option* ⇒ *bool*
where *cproper-interval-cl-states x y* =
(*case ID CCOMPARE*(*cl-states*) *of Some f* ⇒
(*case x of None* ⇒
(*case y of None* ⇒ *True* | *Some c* ⇒ *list-ex* (λ *x*. (*lt-of-comp f*) *x c*) *cl-all-list*)
| *Some c* ⇒
(*case y of None* ⇒ *list-ex* (λ *x*. (*lt-of-comp f*) *c x*) *cl-all-list*
| *Some d* ⇒ (*filter* (λ *x*. (*lt-of-comp f*) *x d* ∧ (*lt-of-comp f*) *c x*) *cl-all-list*) ≠

[])))

instance
⟨*proof*⟩
end

derive (*rbt*) *mapping-impl cl-states*
derive (*rbt*) *mapping-impl bot-term*

end
theory *NF-Impl*
  imports *NF*
    Type-Instances-Impl
begin

6.0.1 Implementation of normal form construction

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

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

lemma *supteq-list* [*simp*]:
  set (*supteq-list t*) = {*s*. *t* ⊵ *s*}
⟨*proof*⟩

lemma *supt-list-sound* [*simp*]:
  set (*supt-list t*) = {*s*. *t* ⊳ *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* ∧ *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* ⟷ (*s*, *t*) ∈ *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* ⟷ (*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-intance-lhs* **where**
check-intance-lhs qs f R ≡ *list-all* (λ *u*. ¬ *bless-eq-impl u* (*BFun f qs*)) *R*

definition *min-elem* **where**
min-elem s ss = (*let ts* = *filter* (λ *x*. *bless-eq-impl x s*) *ss in*
foldr (↑) *ts Bot*)

lemma *bound-impl* [*simp*, *code*]:
bound-max s (*set ss*) = *min-elem s ss*
⟨*proof*⟩

definition *nf-rule-impl* **where**
nf-rule-impl S R SR h = (*let* (*f*, *n*) = *h in*
let states = *List.n-lists n S in*
let nlhs-inst = *filter* (λ *qs*. *check-intance-lhs qs f R*) *states in*
map (λ *qs*. *TA-rule f qs* (*min-elem* (*BFun f qs*) *SR*)) *nlhs-inst*)

abbreviation *nf-rules-impl* **where**
nf-rules-impl R $\mathcal{F}$ ≡ *concat* (*map* (*nf-rule-impl* (*states-impl R*) (*map term-to-bot-term R*) (*psubt-bot-impl R*)) $\mathcal{F}$)

lemma *nf-rules-in-impl*:
assumes *TA-rule f qs q* |∈| *nf-rules* (*fset-of-list R*) (*fset-of-list* $\mathcal{F}$)
shows *TA-rule f qs q* |∈| *fset-of-list* (*nf-rules-impl R* $\mathcal{F}$)
⟨*proof*⟩

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**
  $gctxtex\text{-}onp\ P\ \mathcal{R} = \{(C\langle s\rangle_G,\ C\langle t\rangle_G)\ |\ C\ s\ t.\ P\ C \land (s,\ t) \in \mathcal{R}\}$

definition *gmctxtex-onp* **where**
  gmctxtex-onp P $\mathcal{R}$ = {(*fill-gholes C ss*, *fill-gholes C ts*) | *C ss ts*.
    num-gholes C = *length ss* ∧ *length ss* = *length ts* ∧ *P C* ∧ (∀ *i* < *length ts*. (*ss* ! *i* , *ts* ! *i*) ∈ $\mathcal{R}$)}

definition *compatible-p* **where**
  compatible-p P Q ≡ (∀ *C*. *P C* ⟶ *Q* (*gmctxt-of-gctxt C*))

7.1 Elimination and introduction rules for the extensions

lemma *gctxtex-onpE* [*elim*]:

assumes $(s, t) \in$ *gctxtex-onp* $P\ \mathcal{R}$
obtains $C\ u\ v$ **where** $s = C\langle u\rangle_G\ t = C\langle v\rangle_G\ P\ C\ (u, v) \in \mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-neq-rootE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gctxtex-onp* $P\ \mathcal{R}$ **and** $f \neq g$
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-neq-lengthE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gctxtex-onp* $P\ \mathcal{R}$ **and** *length ss* ≠ *length ts*
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
⟨*proof*⟩

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* ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
⟨*proof*⟩

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 ∀ *i* < *length vs*. (*us* ! *i*, *vs* ! *i*) ∈ $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-neq-rootE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gmctxtex-onp* $P\ \mathcal{R}$ **and** $f \neq g$
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-neq-lengthE* [*elim*]:
assumes (*GFun f ss*, *GFun g ts*) ∈ *gmctxtex-onp* $P\ \mathcal{R}$ **and** *length ss* ≠ *length ts*
shows (*GFun f ss*, *GFun g ts*) ∈ $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-listE*:
assumes ∀ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) ∈ *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*
∀ *i* < *length tss*. *length* (*sss* ! *i*) = *length* (*tss* ! *i*) ∀ *D* ∈ *set Ds*. *Q D*
∀ *i* < *length tss*. *ss* ! $i =_{Gf}$ (*Ds* ! *i*, *sss* ! *i*) ∀ *i* < *length tss*. *ts* ! $i =_{Gf}$ (*Ds* ! *i*, *tss* ! *i*)
∀ *i* < *length* (*concat tss*). (*concat sss* ! *i*, *concat tss* ! *i*) ∈ $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-doubleE* [*elim*]:

assumes $(s, t) \in$ *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$)
obtains *C Ds ss ts us vs* **where** $s =_{Gf} (C, ss)$ $t =_{Gf} (C, ts)$ *P C* $\forall\ D \in$ *set Ds. Q D*
num-gholes C = *length Ds length Ds* = *length ss length ss* = *length ts length ts* = *length us length us* = *length vs*
$\forall\ i <$ *length Ds. ss* ! $i =_{Gf}$ (*Ds* ! *i*, *us* ! *i*) $\wedge$ *ts* ! $i =_{Gf}$ (*Ds* ! *i*, *vs* ! *i*)
$\forall\ i <$ *length Ds.* $\forall\ j <$ *length* (*vs* ! *i*). (*us* ! *i* ! *j*, *vs* ! *i* ! *j*) $\in \mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onpI* [*intro*]:
assumes *P C* **and** $(s, t) \in \mathcal{R}$
shows $(C\langle s\rangle_G, C\langle t\rangle_G) \in$ *gctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onpI* [*intro*]:
assumes *P C* **and** *num-gholes C* = *length us* **and** *length us* = *length vs*
and $\forall\ i <$ *length vs.* (*us* ! *i*, *vs* ! *i*) $\in \mathcal{R}$
shows (*fill-gholes C us*, *fill-gholes C vs*) $\in$ *gmctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-arg-monoI*:
assumes *P GMHole*
shows $\mathcal{R} \subseteq$ *gmctxtex-onp P* $\mathcal{R}$ ⟨*proof*⟩

lemma *gmctxtex-onpI2* [*intro*]:
assumes *P C* **and** $s =_{Gf} (C, ss)$ $t =_{Gf} (C, ts)$
and $\forall\ i <$ *length ts.* (*ss* ! *i*, *ts* ! *i*) $\in \mathcal{R}$
shows $(s, t) \in$ *gmctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-hold-cond* [*simp*]:
$(s, t) \in$ *gctxtex-onp P* $\mathcal{R} \Longrightarrow$ *groot s* $\neq$ *groot t* $\Longrightarrow P\ \Box_G$
$(s, t) \in$ *gctxtex-onp P* $\mathcal{R} \Longrightarrow$ *length* (*gargs s*) $\neq$ *length* (*gargs t*) $\Longrightarrow P\ \Box_G$
⟨*proof*⟩

7.2 Monotonicity rules for the extensions

lemma *gctxtex-onp-rel-mono*:
$\mathcal{L} \subseteq \mathcal{R} \Longrightarrow$ *gctxtex-onp P* $\mathcal{L} \subseteq$ *gctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-rel-mono*:
$\mathcal{L} \subseteq \mathcal{R} \Longrightarrow$ *gmctxtex-onp P* $\mathcal{L} \subseteq$ *gmctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *compatible-p-gctxtex-gmctxtex-subseteq* [*dest*]:
compatible-p P Q $\Longrightarrow$ *gctxtex-onp P* $\mathcal{R} \subseteq$ *gmctxtex-onp Q* $\mathcal{R}$
⟨*proof*⟩

lemma *compatible-p-mono1*:
$P \leq R \Longrightarrow$ *compatible-p* R $Q \Longrightarrow$ *compatible-p* P Q
⟨*proof*⟩

lemma *compatible-p-mono2*:
$Q \leq R \Longrightarrow$ *compatible-p* P $Q \Longrightarrow$ *compatible-p* P R
⟨*proof*⟩

lemma *gctxtex-onp-mono* [*intro*]:
$P \leq Q \Longrightarrow$ *gctxtex-onp* P $\mathcal{R} \subseteq$ *gctxtex-onp* Q $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-mem*:
$P \leq Q \Longrightarrow (s, t) \in$ *gctxtex-onp* P $\mathcal{R} \Longrightarrow (s, t) \in$ *gctxtex-onp* Q $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-mono* [*intro*]:
$P \leq Q \Longrightarrow$ *gmctxtex-onp* P $\mathcal{R} \subseteq$ *gmctxtex-onp* Q $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-mem*:
$P \leq Q \Longrightarrow (s, t) \in$ *gmctxtex-onp* P $\mathcal{R} \Longrightarrow (s, t) \in$ *gmctxtex-onp* Q $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-eqI* [*intro*]:
$P = Q \Longrightarrow \mathcal{R} = \mathcal{L} \Longrightarrow$ *gctxtex-onp* P $\mathcal{R} =$ *gctxtex-onp* Q $\mathcal{L}$
⟨*proof*⟩

lemma *gmctxtex-eqI* [*intro*]:
$P = Q \Longrightarrow \mathcal{R} = \mathcal{L} \Longrightarrow$ *gmctxtex-onp* P $\mathcal{R} =$ *gmctxtex-onp* Q $\mathcal{L}$
⟨*proof*⟩

7.3 Relation swap and converse

lemma *swap-gctxtex-onp*:
gctxtex-onp P (*prod.swap* ‘ $\mathcal{R}$) = *prod.swap* ‘ *gctxtex-onp* P $\mathcal{R}$
⟨*proof*⟩

lemma *swap-gmctxtex-onp*:
gmctxtex-onp P (*prod.swap* ‘ $\mathcal{R}$) = *prod.swap* ‘ *gmctxtex-onp* P $\mathcal{R}$
⟨*proof*⟩

lemma *converse-gctxtex-onp*:
(*gctxtex-onp* P $\mathcal{R})^{-1}$ = *gctxtex-onp* P $(\mathcal{R}^{-1})$
⟨*proof*⟩

lemma *converse-gmctxtex-onp*:
(*gmctxtex-onp* P $\mathcal{R})^{-1}$ = *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* $\Longrightarrow$ $(s,\ t) \in \mathcal{R}$ $\Longrightarrow$ $(C\langle s\rangle_G,\ C\langle t\rangle_G) \in \mathcal{R}$
shows *gctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-closure-predI*:
assumes $\bigwedge$ *C ss ts*. *P C* $\Longrightarrow$ *num-gholes C* = *length ss* $\Longrightarrow$ *length ss* = *length ts* $\Longrightarrow$
($\forall$ *i* < *length ts*. (*ss* ! *i*, *ts* ! *i*) $\in \mathcal{R}$) $\Longrightarrow$ (*fill-gholes C ss*, *fill-gholes C ts*) $\in \mathcal{R}$
shows *gmctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-closure-predE*:
assumes *gctxtex-onp P* $\mathcal{R} \subseteq \mathcal{R}$
shows $\bigwedge$ *C s t*. *P C* $\Longrightarrow$ $(s,\ t) \in \mathcal{R}$ $\Longrightarrow$ $(C\langle s\rangle_G,\ C\langle t\rangle_G) \in \mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-closure* [*intro*]:
P $\Box_G$ $\Longrightarrow$ $\mathcal{R} \subseteq$ *gctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-closure* [*intro*]:
assumes *P GMHole*
shows $\mathcal{R} \subseteq$ (*gmctxtex-onp P* $\mathcal{R}$)
⟨*proof*⟩

lemma *gctxtex-pred-cmp-subseteq*:
assumes $\bigwedge$ *C D*. *P C* $\Longrightarrow$ *Q D* $\Longrightarrow$ *Q* $(C \circ_{Gc} D)$
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gctxtex-onp Q* $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-pred-cmp-subseteq2*:
assumes $\bigwedge$ *C D*. *P C* $\Longrightarrow$ *Q D* $\Longrightarrow$ *P* $(C \circ_{Gc} D)$
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-pred-cmp-subseteq*:
assumes $\bigwedge$ *C D*. $C \leq D$ $\Longrightarrow$ *P C* $\Longrightarrow$ ($\forall$ *Ds* $\in$ *set* (*sup-gmctxt-args C D*). *Q Ds*) $\Longrightarrow$ *Q D*
shows *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gmctxtex-onp Q* $\mathcal{R}$ (**is** *?Ls* $\subseteq$ *?Rs*)
⟨*proof*⟩

lemma *gmctxtex-pred-cmp-subseteq2*:
assumes $\bigwedge$ *C D*. $C \leq D$ $\Longrightarrow$ *P C* $\Longrightarrow$ ($\forall$ *Ds* $\in$ *set* (*sup-gmctxt-args C D*). *Q Ds*) $\Longrightarrow$ *P D*
shows *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$) $\subseteq$ *gmctxtex-onp P* $\mathcal{R}$ (**is** *?Ls* $\subseteq$ *?Rs*)
⟨*proof*⟩

lemma *gctxtex-onp-idem* [*simp*]:
assumes $P\ \Box_G$ **and** $\bigwedge\ C\ D.\ P\ C \Longrightarrow Q\ D \Longrightarrow Q\ (C \circ_{Gc} D)$
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) = *gctxtex-onp Q* $\mathcal{R}$ (**is** *?Ls = ?Rs*)
⟨*proof*⟩

lemma *gctxtex-onp-idem2* [*simp*]:
assumes $Q\ \Box_G$ **and** $\bigwedge\ C\ D.\ P\ C \Longrightarrow Q\ D \Longrightarrow P\ (C \circ_{Gc} D)$
shows *gctxtex-onp P* (*gctxtex-onp Q* $\mathcal{R}$) = *gctxtex-onp P* $\mathcal{R}$ (**is** *?Ls = ?Rs*)
⟨*proof*⟩

lemma *gmctxtex-onp-idem* [*simp*]:
assumes *P GMHole*
and $\bigwedge$ *C D*. $C \leq D \Longrightarrow P\ C \Longrightarrow (\forall\ Ds \in$ *set* (*sup-gmctxt-args C D*). *Q Ds*) $\Longrightarrow Q\ D$
shows *gmctxtex-onp P* (*gmctxtex-onp Q* $\mathcal{R}$) = *gmctxtex-onp Q* $\mathcal{R}$
⟨*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-closing C D* $\longrightarrow$ *num-gholes D* = *1* $\longrightarrow$ *funas-gmctxt D* $\subseteq \mathcal{F}$
$\longrightarrow$ *Q* (*gctxt-of-gmctxt D*))

lemma *gmctxt-p-invE*:
gmctxt-p-inv C $\mathcal{F}$ *Q* $\Longrightarrow C \leq D \Longrightarrow$ *ghole-poss D* $\subseteq$ *ghole-poss C* $\Longrightarrow$ *num-gholes D* = *1* $\Longrightarrow$
funas-gmctxt D $\subseteq \mathcal{F} \Longrightarrow$ *Q* (*gctxt-of-gmctxt D*)
⟨*proof*⟩

lemma *gmctxt-closing-gmctxt-p-inv-comp*:
gmctxt-closing C D $\Longrightarrow$ *gmctxt-p-inv C* $\mathcal{F}$ *Q* $\Longrightarrow$ *gmctxt-p-inv D* $\mathcal{F}$ *Q*
⟨*proof*⟩

lemma *GMHole-gmctxt-p-inv-GHole* [*simp*]:
gmctxt-p-inv GMHole $\mathcal{F}$ *Q* $\Longrightarrow Q\ \Box_G$
⟨*proof*⟩

lemma *gmctxtex-onp-gctxtex-onp-trancl*:
assumes *sig*: $\bigwedge$ *C*. $P\ C \Longrightarrow 0 <$ *num-gholes C* $\wedge$ *funas-gmctxt C* $\subseteq \mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
and $\bigwedge$ *C*. $P\ C \Longrightarrow$ *gmctxt-p-inv C* $\mathcal{F}$ *Q*
shows *gmctxtex-onp P* $\mathcal{R} \subseteq$ (*gctxtex-onp Q* $\mathcal{R}$)$^+$
⟨*proof*⟩

lemma *gmctxtex-onp-gctxtex-onp-rtrancl*:
assumes *sig*: $\bigwedge$ *C*. $P\ C \Longrightarrow$ *funas-gmctxt* $C \subseteq \mathcal{F}$ $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
and $\bigwedge$ *C D*. $P\ C \Longrightarrow$ *gmctxt-p-inv* $C\ \mathcal{F}\ Q$
shows *gmctxtex-onp* $P\ \mathcal{R} \subseteq$ (*gctxtex-onp* $Q\ \mathcal{R})^*$
⟨*proof*⟩

lemma *rtrancl-gmctxtex-onp-rtrancl-gctxtex-onp-eq*:
assumes *sig*: $\bigwedge$ *C*. $P\ C \Longrightarrow$ *funas-gmctxt* $C \subseteq \mathcal{F}$ $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
and $\bigwedge$ *C D*. $P\ C \Longrightarrow$ *gmctxt-p-inv* $C\ \mathcal{F}\ Q$
and *compatible-p* $Q\ P$
shows (*gmctxtex-onp* $P\ \mathcal{R})^*$ = (*gctxtex-onp* $Q\ \mathcal{R})^*$ (**is** *?Ls** = *?Rs**)
⟨*proof*⟩

7.6 Extensions to reflexive transitive closures

lemma *gctxtex-onp-substep-trancl*:
assumes *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$
shows *gctxtex-onp* $P\ (\mathcal{R}^+) \subseteq \mathcal{R}^+$
⟨*proof*⟩

lemma *gctxtex-onp-substep-rtrancl*:
assumes *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$
shows *gctxtex-onp* $P\ (\mathcal{R}^*) \subseteq \mathcal{R}^*$
⟨*proof*⟩

lemma *gctxtex-onp-substep-trancl-diff-pred* [*intro*]:
assumes $\bigwedge$ *C D*. $P\ C \Longrightarrow Q\ D \Longrightarrow Q\ (D \circ_{Gc} C)$
shows *gctxtex-onp* Q ((*gctxtex-onp* $P\ \mathcal{R})^+$) $\subseteq$ (*gctxtex-onp* $Q\ \mathcal{R})^+$
⟨*proof*⟩

lemma *gctxtcl-pres-trancl*:
assumes $(s, t) \in \mathcal{R}^+$ **and** *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$ **and** $P\ C$
shows $(C\langle s\rangle_G,\ C\langle t\rangle_G) \in \mathcal{R}^+$
⟨*proof*⟩

lemma *gctxtcl-pres-rtrancl*:
assumes $(s, t) \in \mathcal{R}^*$ **and** *gctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$ **and** $P\ C$
shows $(C\langle s\rangle_G,\ C\langle t\rangle_G) \in \mathcal{R}^*$
⟨*proof*⟩

lemma *gmctxtex-onp-substep-trancl*:
assumes *gmctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{R}$
and *Id-on* (*snd* ‘ $\mathcal{R}$) $\subseteq \mathcal{R}$
shows *gmctxtex-onp* $P\ (\mathcal{R}^+) \subseteq \mathcal{R}^+$
⟨*proof*⟩

lemma *gmctxtex-onp-substep-tranclE*:
assumes *trans* $\mathcal{R}$ **and** *gmctxtex-onp* $Q\ \mathcal{R}\ O\ \mathcal{R} \subseteq \mathcal{R}$ **and** $\mathcal{R}\ O$ *gmctxtex-onp* Q

$\mathcal{R} \subseteq \mathcal{R}$
and $\bigwedge$ *p C*. *P C* $\Longrightarrow$ *p* $\in$ *poss-gmctxt C* $\Longrightarrow$ *Q* (*subgm-at C p*)
and $\bigwedge$ *C D*. *P C* $\Longrightarrow$ *P D* $\Longrightarrow$ (*C*, *D*) $\in$ *comp-gmctxt* $\Longrightarrow$ *P* (*C* $\sqcap$ *D*)
shows (*gmctxtex-onp P* $\mathcal{R}$)$^+$ = *gmctxtex-onp P* $\mathcal{R}$ (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

7.7 Restr to set, union and predicate distribution

lemma *Restr-gctxtex-onp-dist* [*simp*]:
Restr (*gctxtex-onp P* $\mathcal{R}$) ($\mathcal{T}_G$ $\mathcal{F}$) =
gctxtex-onp (λ *C*. *funas-gctxt C* $\subseteq \mathcal{F}$ $\wedge$ *P C*) (*Restr* $\mathcal{R}$ ($\mathcal{T}_G$ $\mathcal{F}$))
⟨*proof*⟩

lemma *Restr-gmctxtex-onp-dist* [*simp*]:
Restr (*gmctxtex-onp P* $\mathcal{R}$) ($\mathcal{T}_G$ $\mathcal{F}$) =
gmctxtex-onp (λ *C*. *funas-gmctxt C* $\subseteq \mathcal{F}$ $\wedge$ *P C*) (*Restr* $\mathcal{R}$ ($\mathcal{T}_G$ $\mathcal{F}$))
⟨*proof*⟩

lemma *Restr-id-subset-gmctxtex-onp* [*intro*]:
assumes $\bigwedge$ *C*. *num-gholes C* = *0* $\wedge$ *funas-gmctxt C* $\subseteq \mathcal{F}$ $\Longrightarrow$ *P C*
shows *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) $\subseteq$ *gmctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *Restr-id-subset-gmctxtex-onp2* [*intro*]:
assumes $\bigwedge$ *f n*. (*f*, *n*) $\in \mathcal{F}$ $\Longrightarrow$ *P* (*GMFun f* (*replicate n GMHole*))
and $\bigwedge$ *C Ds*. *num-gholes C* = *length Ds* $\Longrightarrow$ *P C* $\Longrightarrow$ $\forall$ *D* $\in$ *set Ds*. *P D* $\Longrightarrow$
P (*fill-gholes-gmctxt C Ds*)
shows *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) $\subseteq$ *gmctxtex-onp P* $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-union* [*simp*]:
gctxtex-onp P ($\mathcal{R} \cup \mathcal{L}$) = *gctxtex-onp P* $\mathcal{R}$ $\cup$ *gctxtex-onp P* $\mathcal{L}$
⟨*proof*⟩

lemma *gctxtex-onp-pred-dist*:
assumes $\bigwedge$ *C*. *P C* $\longleftrightarrow$ *Q C* $\vee$ *R C*
shows *gctxtex-onp P* $\mathcal{R}$ = *gctxtex-onp Q* $\mathcal{R}$ $\cup$ *gctxtex-onp R* $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onp-pred-dist*:
assumes $\bigwedge$ *C*. *P C* $\longleftrightarrow$ *Q C* $\vee$ *R C*
shows *gmctxtex-onp P* $\mathcal{R}$ = *gmctxtex-onp Q* $\mathcal{R}$ $\cup$ *gmctxtex-onp R* $\mathcal{R}$
⟨*proof*⟩

lemma *trivial-gctxtex-onp* [*simp*]: *gctxtex-onp* (λ *C*. *C* = $\Box_G$) $\mathcal{R}$ = $\mathcal{R}$
⟨*proof*⟩

lemma *trivial-gmctxtex-onp* [*simp*]: *gmctxtex-onp* (λ *C*. *C* = *GMHole*) $\mathcal{R}$ = $\mathcal{R}$
⟨*proof*⟩

7.8 Distribution of context closures over relation composition

lemma *gctxtex-onp-relcomp-inner*:
gctxtex-onp P ($\mathcal{R}$ *O* $\mathcal{L}$) $\subseteq$ *gctxtex-onp P* $\mathcal{R}$ *O gctxtex-onp P* $\mathcal{L}$
⟨*proof*⟩

lemma *gmctxtex-onp-relcomp-inner*:
gmctxtex-onp P ($\mathcal{R}$ *O* $\mathcal{L}$) $\subseteq$ *gmctxtex-onp P* $\mathcal{R}$ *O gmctxtex-onp P* $\mathcal{L}$
⟨*proof*⟩

7.9 Signature preserving and signature closed

definition *function-closed* **where**
function-closed $\mathcal{F}$ $\mathcal{R}$ $\longleftrightarrow$ ($\forall$ *f ss ts*. (*f*, *length ts*) $\in \mathcal{F}$ $\longrightarrow$ *0* $\neq$ *length ts* $\longrightarrow$
length ss = *length ts* $\longrightarrow$ ($\forall$ *i*. *i* < *length ts* $\longrightarrow$ (*ss* ! *i*, *ts* ! *i*) $\in \mathcal{R}$) $\longrightarrow$
(*GFun f ss*, *GFun f ts*) $\in \mathcal{R}$)

lemma *function-closedD*: *function-closed* $\mathcal{F}$ $\mathcal{R}$ $\Longrightarrow$
(*f*,*length ts*) $\in \mathcal{F}$ $\Longrightarrow$ *0* $\neq$ *length ts* $\Longrightarrow$ *length ss* = *length ts* $\Longrightarrow$
$[\![\bigwedge$ *i*. *i* < *length ts* $\Longrightarrow$ (*ss* ! *i*, *ts* ! *i*) $\in \mathcal{R}]\!]$ $\Longrightarrow$
(*GFun f ss*, *GFun f ts*) $\in \mathcal{R}$
⟨*proof*⟩

lemma *all-ctxt-closed-imp-function-closed*:
all-ctxt-closed $\mathcal{F}$ $\mathcal{R}$ $\Longrightarrow$ *function-closed* $\mathcal{F}$ $\mathcal{R}$
⟨*proof*⟩

lemma *all-ctxt-closed-imp-reflx-on-sig*:
assumes *all-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-closed $\mathcal{F}$ $\mathcal{R}$ $\Longrightarrow$ *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) $\subseteq \mathcal{R}$ $\Longrightarrow$ *all-ctxt-closed* $\mathcal{F}$ $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-onp-in-signature* [*intro*]:
assumes $\bigwedge$ *C*. *P C* $\Longrightarrow$ *funas-gctxt C* $\subseteq \mathcal{F}$ $\bigwedge$ *C*. *P C* $\Longrightarrow$ *funas-gctxt C* $\subseteq \mathcal{G}$
and $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$
shows *gctxtex-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* $\Longrightarrow$ *funas-gmctxt C* $\subseteq \mathcal{F}$ $\bigwedge$ *C*. *P C* $\Longrightarrow$ *funas-gmctxt C* $\subseteq$
$\mathcal{G}$
and $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$

shows *gmctxtex-onp* $P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{G}$ ⟨*proof*⟩

lemma *gctxtex-onp-in-signature-tranc* [*intro*]:
gctxtex-onp $P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ (*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 $P\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ (*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 $\bigwedge f\ n.\ (f,\ n) \in \mathcal{F} \Longrightarrow n \neq 0 \Longrightarrow P$ (*GMFun f* (*replicate n GMHole*))
and $\bigwedge$ *C Ds. P C* $\Longrightarrow$ *num-gholes C* = *length Ds* $\Longrightarrow 0 <$ *num-gholes C* $\Longrightarrow$
$\forall\ D \in$ *set Ds. P D* $\Longrightarrow P$ (*fill-gholes-gmctxt C Ds*)
shows *function-closed* $\mathcal{F}$ (*gmctxtex-onp* $P\ \mathcal{R}$) ⟨*proof*⟩

declare *subsetI*[*rule del*]
lemma *gmctxtex-onp-sig-closed* [*intro*]:
assumes $\bigwedge f\ n.\ (f,\ n) \in \mathcal{F} \Longrightarrow P$ (*GMFun f* (*replicate n GMHole*))
and $\bigwedge$ *C Ds. num-gholes C* = *length Ds* $\Longrightarrow P\ C \Longrightarrow \forall\ D \in$ *set Ds. P D* $\Longrightarrow$
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* (λ *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* | *Bwd nat* — TRS id and direction

definition *map-ftrs* **where**
map-ftrs f = *case-ftrs* (*Fwd* ∘ *f*) (*Bwd* ∘ *f*)

8.1 GTT relations

datatype *'trs gtt-rel* — GTT relations
= *ARoot 'trs list* — root steps
| *GInv 'trs gtt-rel* — inverse of anchored or ordinary GTT relation
| *AUnion 'trs gtt-rel 'trs gtt-rel* — union of anchored GTT relation
| *ATrancl 'trs gtt-rel* — transitive closure of anchored GTT relation

| *GTrancl 'trs gtt-rel* — transitive closure of ordinary GTT relation
| *AComp 'trs gtt-rel 'trs gtt-rel* — composition of anchored GTT relations
| *GComp 'trs gtt-rel 'trs gtt-rel* — composition of ordinary GTT relations

definition *GSteps* **where** *GSteps trss* = *GTrancl* (*ARoot trss*)

8.2 RR1 and RR2 relations

datatype *pos-step* — position specification for lifting anchored GTT relation
= *PRoot* — allow only root steps
| *PNonRoot* — allow only non-root steps
| *PAny* — allow any position

datatype *ext-step* — kind of rewrite steps for lifting anchored GTT relation
= *ESingle* — single steps
| *EParallel* — parallel steps, allowing the empty step
| *EStrictParallel* — parallel steps, no allowing the empty step

datatype *'trs rr1-rel* — RR1 relations, aka regular tree languages
= *R1Terms* — all terms as RR1 relation (regular tree languages)
| *R1NF 'trs list* — direct normal form construction wrt. single steps
| *R1Inf 'trs rr2-rel* — infiniteness predicate
| *R1Proj nat 'trs rr2-rel* — projection of RR2 relation
| *R1Union 'trs rr1-rel 'trs rr1-rel* — union of RR1 relations
| *R1Inter 'trs rr1-rel 'trs rr1-rel* — intersection of RR1 relations
| *R1Diff 'trs rr1-rel 'trs rr1-rel* — difference of RR1 relations
and *'trs rr2-rel* — RR2 relations
= *R2GTT-Rel 'trs gtt-rel pos-step ext-step* — lifted GTT relations
| *R2Diag 'trs rr1-rel* — diagonal relation
| *R2Prod 'trs rr1-rel 'trs rr1-rel* — Cartesian product
| *R2Inv 'trs rr2-rel* — inverse of RR2 relation
| *R2Union 'trs rr2-rel 'trs rr2-rel* — union of RR2 relations
| *R2Inter 'trs rr2-rel 'trs rr2-rel* — intersection of RR2 relations
| *R2Diff 'trs rr2-rel 'trs rr2-rel* — difference of RR2 relations
| *R2Comp 'trs rr2-rel 'trs rr2-rel* — composition of RR2 relations

definition *R1Fin* **where** — finiteness predicate
R1Fin r = *R1Diff R1Terms* (*R1Inf r*)
definition *R2Eq* **where** — equality
R2Eq = *R2Diag R1Terms*
definition *R2Reflc* **where** — reflexive closure
R2Reflc r = *R2Union r R2Eq*
definition *R2Step* **where** — single step $\rightarrow$
R2Step trss = *R2GTT-Rel* (*ARoot trss*) *PAny ESingle*
definition *R2StepEq* **where** — at most one step $\rightarrow^=$

R2StepEq trss = *R2Reflc* (*R2Step trss*)
definition *R2Steps* **where** — at least one step $\to^+$
R2Steps trss = *R2GTT-Rel* (*GSteps trss*) *PAny EStrictParallel*
definition *R2StepsEq* **where** — many steps $\to^*$
R2StepsEq trss = *R2GTT-Rel* (*GSteps trss*) *PAny EParallel*
definition *R2StepsNF* **where** — rewrite to normal form $\to^!$
R2StepsNF trss = *R2Inter* (*R2StepsEq trss*) (*R2Prod R1Terms* (*R1NF trss*))
definition *R2ParStep* **where** — parallel step
R2ParStep trss = *R2GTT-Rel* (*ARoot trss*) *PAny EParallel*
definition *R2RootStep* **where** — root step $\to_\epsilon$
R2RootStep trss = *R2GTT-Rel* (*ARoot trss*) *PRoot ESingle*
definition *R2RootStepEq* **where** — at most one root step $\to_\epsilon^=$
R2RootStepEq trss = *R2Reflc* (*R2RootStep trss*)

definition *R2RootSteps* **where** — at least one root step $\to_\epsilon^+$
R2RootSteps trss = *R2GTT-Rel* (*ATrancl* (*ARoot trss*)) *PRoot ESingle*
definition *R2RootStepsEq* **where** — many root steps $\to_\epsilon^*$
R2RootStepsEq trss = *R2Reflc* (*R2RootSteps trss*)
definition *R2NonRootStep* **where** — non-root step $\to_{>\epsilon}$
R2NonRootStep trss = *R2GTT-Rel* (*ARoot trss*) *PNonRoot ESingle*
definition *R2NonRootStepEq* **where** — at most one non-root step $\to_{>\epsilon}^=$
R2NonRootStepEq trss = *R2Reflc* (*R2NonRootStep trss*)
definition *R2NonRootSteps* **where** — at least one non-root step $\to_{>\epsilon}^+$
R2NonRootSteps trss = *R2GTT-Rel* (*GSteps trss*) *PNonRoot EStrictParallel*
definition *R2NonRootStepsEq* **where** — many non-root steps $\to_{>\epsilon}^*$
R2NonRootStepsEq trss = *R2GTT-Rel* (*GSteps trss*) *PNonRoot EParallel*
definition *R2Meet* **where** — meet ↑
R2Meet trss = *R2GTT-Rel* (*GComp* (*GInv* (*GSteps trss*)) (*GSteps trss*)) *PAny EParallel*
definition *R2Join* **where** — join ↓
R2Join trss = *R2GTT-Rel* (*GComp* (*GSteps trss*) (*GInv* (*GSteps trss*))) *PAny EParallel*

8.3 Formulas

datatype *'trs formula* — formulas
= *FRR1 'trs rr1-rel fvar* — application of RR1 relation
| *FRR2 'trs rr2-rel fvar fvar* — application of RR2 relation
| *FAnd* (*'trs formula*) *list* — conjunction
| *FOr* (*'trs formula*) *list* — disjunction
| *FNot 'trs formula* — negation
| *FExists 'trs formula* — existential quantification
| *FForall 'trs formula* — universal quantification

definition *FTrue* **where** — true
FTrue ≡ *FAnd* []
definition *FFalse* **where** — false
FFalse ≡ *FOr* []

definition *FRestrict* **where** — reorder/rename/restrict TRSs for subformula
FRestrict f trss ≡ *map-formula* (*map-ftrs* (λ*n. if n* ≥ *length trss then 0 else trss* ! *n*)) *f*

8.4 Signatures and Problems

datatype ('*f*, '*v*, '*t*) *many-sorted-sig*
= *Many-Sorted-Sig* (*ms-functions*: ('*f* × '*t list* × '*t*) *list*) (*ms-variables*: ('*v* × '*t*) *list*)

datatype ('*f*, '*v*, '*t*) *problem*
= *Problem* (*p-signature*: ('*f*, '*v*, '*t*) *many-sorted-sig*)
(*p-trss*: ('*f*, '*v*) *trs list*)
(*p-formula*: *ftrs formula*)

8.5 Proofs

datatype *equivalence* — formula equivalences
= *EDistribAndOr* — distributivity: conjunction over disjunction
| *EDistribOrAnd* — distributivity: disjunction over conjunction

datatype '*trs inference* — inference rules for formula creation
= *IRR1* '*trs rr1-rel fvar* — formula from RR1 relation
| *IRR2* '*trs rr2-rel fvar fvar* — formula from RR2 relation
| *IAnd nat list* — conjunction
| *IOr nat list* — disjunction
| *INot nat* — negation
| *IExists nat* — existential quantification
| *IRename nat fvar list* — permute variables
| *INNFPlus nat* — equivalence modulo negation normal form plus ACIU0 for ∧ and ∨
| *IRepl equivalence nat list nat* — replacement according to given equivalence

datatype *claim* = *Empty* | *Nonempty*

datatype *info* = *Size nat nat nat*

datatype '*trs certificate*
= *Certificate* (*nat* × '*trs inference* × '*trs formula* × *info list*) *list claim nat*

8.6 Example

definition *no-normal-forms-cert* :: *ftrs certificate* **where**
no-normal-forms-cert = *Certificate*
[(*0*, (*IRR2* (*R2Step* [*Fwd 0*]) *1 0*),
(*FRR2* (*R2Step* [*Fwd 0*]) *1 0*), [])
, (*1*, (*IExists 0*),
(*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*)), [])
, (*2*, (*INot 1*),

    (*FNot* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*))), [])
  , (*3*, (*IExists 2*),
    (*FExists* (*FNot* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*)))), [])
  , (*4*, (*INot 3*),
    (*FNot* (*FExists* (*FNot* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*))))), [])
  , (*5*, (*INNFPlus 4*),
    (*FForall* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*))), [])
  ] *Nonempty 5*

definition *no-normal-forms-problem* :: (*string*, *string*, *unit*) *problem* **where**
  no-normal-forms-problem = *Problem*
    (*Many-Sorted-Sig* [(''*f*'',[()],()), (''*a*'',[],())] [(''*x*'',())])
    [{(*Fun* ''*f*'' [*Var* ''*x*''],*Fun* ''*a*'' [])}]
    (*FForall* (*FExists* (*FRR2* (*R2Step* [*Fwd 0*]) *1 0*)))

end

9 Lifting root steps to single/parallel root/non-root steps

theory *Lift-Root-Step*
  imports
    Rewriting
    FOR-Certificate
    Context-Extensions
    Multihole-Context
begin

Closure under all contexts

abbreviation *gctxtcl* $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. *True*) $\mathcal{R}$
abbreviation *gmctxtcl* $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. *True*) $\mathcal{R}$

Extension under all non empty contexts

abbreviation *gctxtex-nempty* $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. $C \neq \Box_G$) $\mathcal{R}$
abbreviation *gmctxtex-nempty* $\mathcal{R} \equiv$ *gmctxtex-onp* (λ *C*. $C \neq$ *GMHole*) $\mathcal{R}$

Closure under all contexts respecting the signature

abbreviation *gctxtcl-funas* $\mathcal{F}$ $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. *funas-gctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$
abbreviation *gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R} \equiv$ *gmctxtex-onp* (λ *C*. *funas-gmctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$

Closure under all multihole contexts with at least one hole respecting the signature

abbreviation *gmctxtcl-funas-strict* $\mathcal{F}$ $\mathcal{R} \equiv$ *gmctxtex-onp* (λ *C*. $0 <$ *num-gholes* *C* $\wedge$ *funas-gmctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$

Extension under all non empty contexts respecting the signature

abbreviation *gctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R} \equiv$ *gctxtex-onp* (λ *C*. *funas-gctxt* $C \subseteq \mathcal{F} \wedge C \neq \Box_G$) $\mathcal{R}$

abbreviation *gmctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$ ≡ *gmctxtex-onp* (λ *C*. *funas-gmctxt* *C* ⊆ $\mathcal{F}$ ∧ *C* ≠ *GMHole*) $\mathcal{R}$

Extension under all non empty contexts respecting the signature

abbreviation *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$ ≡
gmctxtex-onp (λ *C*. *0* < *num-gholes* *C* ∧ *funas-gmctxt* *C* ⊆ $\mathcal{F}$ ∧ *C* ≠ *GMHole*) $\mathcal{R}$

9.1 Rewrite steps equivalent definitions

definition *gsubst-cl* :: ('*f*, '*v*) *trs* ⇒ '*f* *gterm* *rel* **where**
gsubst-cl $\mathcal{R}$ = {(*gterm-of-term* (*l* · σ), *gterm-of-term* (*r* · σ)) |
l *r* (σ :: '*v* ⇒ ('*f*, '*v*) *Term.term*). (*l*, *r*) ∈ $\mathcal{R}$ ∧ *ground* (*l* · σ) ∧ *ground* (*r* · σ)}

definition *gnrrstepD* :: '*f* *sig* ⇒ '*f* *gterm* *rel* ⇒ '*f* *gterm* *rel* **where**
gnrrstepD $\mathcal{F}$ $\mathcal{R}$ = *gctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$

definition *grstepD* :: '*f* *sig* ⇒ '*f* *gterm* *rel* ⇒ '*f* *gterm* *rel* **where**
grstepD $\mathcal{F}$ $\mathcal{R}$ = *gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$

definition *gpar-rstepD* :: '*f* *sig* ⇒ '*f* *gterm* *rel* ⇒ '*f* *gterm* *rel* **where**
gpar-rstepD $\mathcal{F}$ $\mathcal{R}$ = *gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$

inductive-set *gpar-rstepD'* :: '*f* *sig* ⇒ '*f* *gterm* *rel* ⇒ '*f* *gterm* *rel* **for** $\mathcal{F}$ :: '*f* *sig* **and** $\mathcal{R}$ :: '*f* *gterm* *rel*
where *groot-step* [*intro*]: (*s*, *t*) ∈ $\mathcal{R}$ ⟹ (*s*, *t*) ∈ *gpar-rstepD'* $\mathcal{F}$ $\mathcal{R}$
| *gpar-step-fun* [*intro*]: ⟦⋀ *i*. *i* < *length* *ts* ⟹ (*ss* ! *i*, *ts* ! *i*) ∈ *gpar-rstepD'* $\mathcal{F}$ $\mathcal{R}$⟧ ⟹ *length* *ss* = *length* *ts*
⟹ (*f*, *length* *ts*) ∈ $\mathcal{F}$ ⟹ (*GFun* *f* *ss*, *GFun* *f* *ts*) ∈ *gpar-rstepD'* $\mathcal{F}$ $\mathcal{R}$

9.2 Interface between rewrite step definitions and sets

fun *lift-root-step* :: ('*f* × *nat*) *set* ⇒ *pos-step* ⇒ *ext-step* ⇒ '*f* *gterm* *rel* ⇒ '*f* *gterm* *rel* **where**
lift-root-step $\mathcal{F}$ *PAny* *ESingle* $\mathcal{R}$ = *gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PAny* *EStrictParallel* $\mathcal{R}$ = *gmctxtcl-funas-strict* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PAny* *EParallel* $\mathcal{R}$ = *gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PNonRoot* *ESingle* $\mathcal{R}$ = *gctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PNonRoot* *EStrictParallel* $\mathcal{R}$ = *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PNonRoot* *EParallel* $\mathcal{R}$ = *gmctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PRoot* *ESingle* $\mathcal{R}$ = $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PRoot* *EStrictParallel* $\mathcal{R}$ = $\mathcal{R}$
| *lift-root-step* $\mathcal{F}$ *PRoot* *EParallel* $\mathcal{R}$ = $\mathcal{R}$ ∪ *Restr* *Id* ($\mathcal{T}_G$ $\mathcal{F}$)

9.3 Compatibility of used predicate extensions and signature closure

lemma *compatible-p* [*simp*]:
compatible-p (λ *C*. *C* ≠ $\Box_G$) (λ *C*. *C* ≠ *GMHole*)
compatible-p (λ *C*. *funas-gctxt* *C* ⊆ $\mathcal{F}$) (λ *C*. *funas-gmctxt* *C* ⊆ $\mathcal{F}$)

compatible-p (λ *C*. *funas-gctxt* $C \subseteq \mathcal{F} \wedge C \neq \square_G$) ($\lambda$ *C*. *funas-gmctxt* $C \subseteq \mathcal{F} \wedge$ $C \neq$ *GMHole*)
⟨*proof*⟩

lemma *gmctxtcl-funas-sigcl*:
all-ctxt-closed $\mathcal{F}$ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)
⟨*proof*⟩

lemma *gctxtex-funas-nroot-sigcl*:
all-ctxt-closed $\mathcal{F}$ (*gmctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$)
⟨*proof*⟩

lemma *gmctxtcl-funas-strict-funcl*:
function-closed $\mathcal{F}$ (*gmctxtcl-funas-strict* $\mathcal{F}$ $\mathcal{R}$)
⟨*proof*⟩

lemma *gmctxtex-funas-nroot-strict-funcl*:
function-closed $\mathcal{F}$ (*gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R}$)
⟨*proof*⟩

lemma *gctxtcl-funas-dist*:
gctxtcl-funas $\mathcal{F}$ $\mathcal{R}$ = *gctxtex-onp* (λ *C*. $C = \square_G$) $\mathcal{R} \cup$ *gctxtex-funas-nroot* $\mathcal{F}$ $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-funas-nroot-dist*:
gmctxtex-funas-nroot $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R} \cup$
gmctxtex-onp (λ *C*. *num-gholes* $C = 0 \wedge$ *funas-gmctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtcl-funas-dist*:
gmctxtcl-funas $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-onp* (λ *C*. *num-gholes* $C = 0 \wedge$ *funas-gmctxt* C $\subseteq \mathcal{F}$) $\mathcal{R} \cup$
gmctxtex-onp (λ *C*. $0 <$ *num-gholes* $C \wedge$ *funas-gmctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtcl-funas-strict-dist*:
gmctxtcl-funas-strict $\mathcal{F}$ $\mathcal{R}$ = *gmctxtex-funas-nroot-strict* $\mathcal{F}$ $\mathcal{R} \cup$ *gmctxtex-onp* (λ *C*. $C =$ *GMHole*) $\mathcal{R}$
⟨*proof*⟩

lemma *gmctxtex-onpzero-num-gholes-id* [*simp*]:
gmctxtex-onp (λ *C*. *num-gholes* $C = 0 \wedge$ *funas-gmctxt* $C \subseteq \mathcal{F}$) $\mathcal{R}$ = *Restr Id* ($\mathcal{T}_G$ $\mathcal{F}$) (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *gctxtex-onp-sign-trans-fst*:
assumes $(s, t) \in$ *gctxtex-onp* P R **and** $s \in \mathcal{T}_G$ $\mathcal{F}$
shows $(s, t) \in$ *gctxtex-onp* (λ *C*. *funas-gctxt* $C \subseteq \mathcal{F} \wedge P$ C) R
⟨*proof*⟩

lemma *gctxtex-onp-sign-trans-snd*:
assumes $(s, t) \in$ *gctxtex-onp* P R **and** $t \in \mathcal{T}_G\ \mathcal{F}$
shows $(s, t) \in$ *gctxtex-onp* $(\lambda\ C.$ *funas-gctxt* $C \subseteq \mathcal{F} \land P\ C)\ R$
⟨*proof*⟩

lemma *gmctxtex-onp-sign-trans-fst*:
assumes $(s, t) \in$ *gmctxtex-onp* P R **and** $s \in \mathcal{T}_G\ \mathcal{F}$
shows $(s, t) \in$ *gmctxtex-onp* $(\lambda\ C.\ P\ C \land$ *funas-gmctxt* $C \subseteq \mathcal{F})\ R$
⟨*proof*⟩

lemma *gmctxtex-onp-sign-trans-snd*:
assumes $(s, t) \in$ *gmctxtex-onp* P R **and** $t \in \mathcal{T}_G\ \mathcal{F}$
shows $(s, t) \in$ *gmctxtex-onp* $(\lambda\ C.\ P\ C \land$ *funas-gmctxt* $C \subseteq \mathcal{F})\ R$
⟨*proof*⟩

9.4 Basic lemmas

lemma *gsubst-cl*:
fixes $\mathcal{R}$ :: $('f, 'v)$ *trs* **and** σ :: $'v \Rightarrow ('f, 'v)$ *term*
assumes $(l, r) \in \mathcal{R}$ **and** *ground* $(l \cdot \sigma)$ *ground* $(r \cdot \sigma)$
shows (*gterm-of-term* $(l \cdot \sigma)$, *gterm-of-term* $(r \cdot \sigma)) \in$ *gsubst-cl* $\mathcal{R}$
⟨*proof*⟩

lemma *grstepD* [*simp*]:
$(s, t) \in \mathcal{R} \Longrightarrow (s, t) \in$ *grstepD* $\mathcal{F}$ $\mathcal{R}$
⟨*proof*⟩

lemma *grstepD-ctxtI* [*intro*]:
$(l, r) \in \mathcal{R} \Longrightarrow$ *funas-gctxt* $C \subseteq \mathcal{F} \Longrightarrow (C\langle l\rangle_G,\ C\langle r\rangle_G) \in$ *grstepD* $\mathcal{F}$ $\mathcal{R}$
⟨*proof*⟩

lemma *gctxtex-funas-nroot-gctxtcl-funas-subseteq*:
gctxtex-funas-nroot $\mathcal{F}$ (*grstepD* $\mathcal{F}$ $\mathcal{R}$) $\subseteq$ *grstepD* $\mathcal{F}$ $\mathcal{R}$
⟨*proof*⟩

lemma *Restr-gnrrstepD-dist* [*simp*]:
Restr (*gnrrstepD* $\mathcal{F}$ $\mathcal{R}$) $(\mathcal{T}_G\ \mathcal{G})$ = *gnrrstepD* $(\mathcal{F} \cap \mathcal{G})$ (*Restr* $\mathcal{R}$ $(\mathcal{T}_G\ \mathcal{G})$)
⟨*proof*⟩

lemma *Restr-grstepD-dist* [*simp*]:
Restr (*grstepD* $\mathcal{F}$ $\mathcal{R}$) $(\mathcal{T}_G\ \mathcal{G})$ = *grstepD* $(\mathcal{F} \cap \mathcal{G})$ (*Restr* $\mathcal{R}$ $(\mathcal{T}_G\ \mathcal{G})$)
⟨*proof*⟩

lemma *Restr-gpar-rstepD-dist* [*simp*]:
Restr (*gpar-rstepD* $\mathcal{F}$ $\mathcal{R}$) $(\mathcal{T}_G\ \mathcal{G})$ = *gpar-rstepD* $(\mathcal{F} \cap \mathcal{G})$ (*Restr* $\mathcal{R}$ $(\mathcal{T}_G\ \mathcal{G})$) (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

9.5 Equivalence lemmas

lemma *grrstep-subst-cl-conv*:
  $grrstep\ \mathcal{R} = gsubst\text{-}cl\ \mathcal{R}$
  ⟨*proof*⟩

lemma *gnrrstepD-gnrrstep-conv*:
  $gnrrstep\ \mathcal{R} = gnrrstepD\ UNIV\ (gsubst\text{-}cl\ \mathcal{R})$ (**is** *?Ls = ?Rs*)
⟨*proof*⟩

lemma *grstepD-grstep-conv*:
  $grstep\ \mathcal{R} = grstepD\ UNIV\ (gsubst\text{-}cl\ \mathcal{R})$ (**is** *?Ls = ?Rs*)
⟨*proof*⟩

lemma *gpar-rstep-gpar-rstepD-conv*:
  $gpar\text{-}rstep\ \mathcal{R} = gpar\text{-}rstepD'\ UNIV\ (gsubst\text{-}cl\ \mathcal{R})$ (**is** *?Ls = ?Rs*)
⟨*proof*⟩

lemma *gmctxtcl-funas-idem*:
  $gmctxtcl\text{-}funas\ \mathcal{F}\ (gmctxtcl\text{-}funas\ \mathcal{F}\ \mathcal{R}) \subseteq gmctxtcl\text{-}funas\ \mathcal{F}\ \mathcal{R}$
  ⟨*proof*⟩

lemma *gpar-rstepD-gpar-rstepD′-conv*:
  $gpar\text{-}rstepD\ \mathcal{F}\ \mathcal{R} = gpar\text{-}rstepD'\ \mathcal{F}\ \mathcal{R}$ (**is** *?Ls = ?Rs*)
⟨*proof*⟩

9.6 Signature preserving lemmas

lemma $\mathcal{T}_G$*-trans-closure-id* [*simp*]:
  $(\mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F})^+ = \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
  ⟨*proof*⟩

lemma *signature-pres-funas-cl* [*simp*]:
  $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow gctxtcl\text{-}funas\ \mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
  $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow gmctxtcl\text{-}funas\ \mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
  ⟨*proof*⟩

lemma *relf-on-gmctxtcl-funas*:
  assumes $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
  shows $refl\text{-}on\ (\mathcal{T}_G\ \mathcal{F})\ (gmctxtcl\text{-}funas\ \mathcal{F}\ \mathcal{R})$
⟨*proof*⟩

lemma *gtrancl-rel-sound*:
  $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow gtrancl\text{-}rel\ \mathcal{F}\ \mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
  ⟨*proof*⟩

9.7 *gcomp-rel* and *gtrancl-rel* lemmas

lemma *gcomp-rel*:
  $lift\text{-}root\text{-}step\ \mathcal{F}\ PAny\ EParallel\ (gcomp\text{-}rel\ \mathcal{F}\ \mathcal{R}\ \mathcal{S}) = lift\text{-}root\text{-}step\ \mathcal{F}\ PAny$

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* $\mathcal{F}$ $\mathcal{R}$ ⊆ (*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 $\mathcal{R}$ ⊆ *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* $\mathcal{F}$ $\mathcal{R}$) ⊆ (*gmctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$
⟨*proof*⟩

lemma *gtrancl-rel-aux*:
  $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *gmctxtcl-funas* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) *O gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$ ⊆ *gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$
  $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F} \Longrightarrow$ *gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$ *O gmctxtcl-funas* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) ⊆ *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* $\Longrightarrow$ *funas-gmctxt C* ⊆ $\mathcal{F}$
    and $\bigwedge$ *C D*. *P C* $\Longrightarrow$ *P D* $\Longrightarrow$ (*C*, *D*) ∈ *comp-gmctxt* $\Longrightarrow$ *P* (*C* ⊓ *D*)
  shows (*gctxtex-onp Q* $\mathcal{R}$)$^+$ ⊆ *gmctxtex-onp P* (*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* $\mathcal{F}$ $\mathcal{R}$ ⊆ (*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* $\Longrightarrow$ *funas-gctxt D* ⊆ $\mathcal{F}$ $\Longrightarrow$ *Q* (*C* $\circ_{Gc}$ *D*)
    and $\bigwedge$*C*. *P C* $\Longrightarrow$ *0* < *num-gholes C* ∧ *funas-gmctxt C* ⊆ $\mathcal{F}$
    and $\bigwedge$*C*. *P C* $\Longrightarrow$ *gmctxt-p-inv C* $\mathcal{F}$ *Q*
  shows *gmctxtex-onp P* (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) ⊆ (*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 *gmctxtcl-funas-strict* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) = (*gctxtcl-funas* $\mathcal{F}$ $\mathcal{R}$)$^+$ (**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 *gmctxtex-funas-nroot-strict* $\mathcal{F}$ (*gtrancl-rel* $\mathcal{F}$ $\mathcal{R}$) = (*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 *lift-root-step* $\mathcal{F}$ *W X* $\mathcal{R} \subseteq \mathcal{T}_G\ \mathcal{G} \times \mathcal{T}_G\ \mathcal{H}$
⟨*proof*⟩

lemmas *lift-root-step-sig* = *lift-root-step-sig'*[*OF - subset-refl subset-refl*]

lemma *lift-root-step-incr*:
$\mathcal{R} \subseteq \mathcal{S} \Longrightarrow$ *lift-root-step* $\mathcal{F}$ *W X* $\mathcal{R} \subseteq$ *lift-root-step* $\mathcal{F}$ *W X* $\mathcal{S}$
⟨*proof*⟩

lemma *Restr-id-mono*:
$\mathcal{F} \subseteq \mathcal{G} \Longrightarrow$ *Restr Id* ($\mathcal{T}_G\ \mathcal{F}$) $\subseteq$ *Restr Id* ($\mathcal{T}_G\ \mathcal{G}$)
⟨*proof*⟩

lemma *lift-root-step-mono*:
$\mathcal{F} \subseteq \mathcal{G} \Longrightarrow$ *lift-root-step* $\mathcal{F}$ *W X* $\mathcal{R} \subseteq$ *lift-root-step* $\mathcal{G}$ *W X* $\mathcal{R}$
⟨*proof*⟩

lemma *grstep-lift-root-step*:
lift-root-step $\mathcal{F}$ *PAny ESingle* (*Restr* (*grrstep* $\mathcal{R}$) ($\mathcal{T}_G\ \mathcal{F}$)) = *Restr* (*grstep* $\mathcal{R}$) ($\mathcal{T}_G\ \mathcal{F}$)
⟨*proof*⟩

lemma *prod-swap-id-on-refl* [*simp*]:
Restr Id ($\mathcal{T}_G\ \mathcal{F}$) $\subseteq$ *prod.swap* ‘ ($\mathcal{R} \cup$ *Restr Id* ($\mathcal{T}_G\ \mathcal{F}$))
⟨*proof*⟩

lemma *swap-lift-root-step*:
lift-root-step $\mathcal{F}$ *W X* (*prod.swap* ‘ $\mathcal{R}$) = *prod.swap* ‘ *lift-root-step* $\mathcal{F}$ *W X* $\mathcal{R}$
⟨*proof*⟩

lemma *converse-lift-root-step*:
(*lift-root-step* $\mathcal{F}$ *W X R*)$^{-1}$ = *lift-root-step* $\mathcal{F}$ *W X* (R^{-1})

⟨*proof*⟩

lemma *lift-root-step-sig-transfer*:
assumes $p \in$ *lift-root-step* $\mathcal{F}$ *W X R snd* ' $R \subseteq \mathcal{T}_G\ \mathcal{F}$ *funas-gterm* (*fst p*) $\subseteq \mathcal{G}$
shows $p \in$ *lift-root-step* $\mathcal{G}$ *W X R* ⟨*proof*⟩

lemma *lift-root-step-sig-transfer2*:
assumes $p \in$ *lift-root-step* $\mathcal{F}$ *W X R snd* ' $R \subseteq \mathcal{T}_G\ \mathcal{G}$ *funas-gterm* (*fst p*) $\subseteq \mathcal{G}$
shows $p \in$ *lift-root-step* $\mathcal{G}$ *W X R*
⟨*proof*⟩

lemma *lift-root-steps-sig-transfer*:
assumes $(s, t) \in ($*lift-root-step* $\mathcal{F}$ *W X R*$)^+$ *snd* ' $R \subseteq \mathcal{T}_G\ \mathcal{G}$ *funas-gterm s* $\subseteq \mathcal{G}$
shows $(s, t) \in ($*lift-root-step* $\mathcal{G}$ *W X R*$)^+$
⟨*proof*⟩

lemma *lift-root-stepseq-sig-transfer*:
assumes $(s, t) \in ($*lift-root-step* $\mathcal{F}$ *W X R*$)^*$ *snd* ' $R \subseteq \mathcal{T}_G\ \mathcal{G}$ *funas-gterm s* $\subseteq \mathcal{G}$
shows $(s, t) \in ($*lift-root-step* $\mathcal{G}$ *W X R*$)^*$
⟨*proof*⟩

lemmas *lift-root-step-sig-transfer'* = *lift-root-step-sig-transfer*[*of prod.swap p* $\mathcal{F}$ *W X prod.swap* ' *R* $\mathcal{G}$ **for** *p* $\mathcal{F}$ *W X* $\mathcal{G}$ *R*,
unfolded swap-lift-root-step, *OF imageI*, *THEN imageI* [*of - - prod.swap*],
unfolded image-comp comp-def fst-swap snd-swap swap-swap image-ident]

lemmas *lift-root-steps-sig-transfer'* = *lift-root-steps-sig-transfer*[*of t s* $\mathcal{F}$ *W X prod.swap* ' *R* $\mathcal{G}$ **for** *t s* $\mathcal{F}$ *W X* $\mathcal{G}$ *R*,
THEN imageI [*of - - prod.swap*], *unfolded swap-lift-root-step swap-trancl pair-in-swap-image*
image-comp comp-def snd-swap swap-swap swap-simp image-ident]

lemmas *lift-root-stepseq-sig-transfer'* = *lift-root-stepseq-sig-transfer*[*of t s* $\mathcal{F}$ *W X prod.swap* ' *R* $\mathcal{G}$ **for** *t s* $\mathcal{F}$ *W X* $\mathcal{G}$ *R*,
THEN imageI [*of - - prod.swap*], *unfolded swap-lift-root-step swap-rtrancl pair-in-swap-image*
image-comp comp-def snd-swap swap-swap swap-simp image-ident]

lemma *lift-root-step-PRoot-ESingle* [*simp*]:
lift-root-step $\mathcal{F}$ *PRoot ESingle* $\mathcal{R} = \mathcal{R}$
⟨*proof*⟩

lemma *lift-root-step-PRoot-EStrictParallel* [*simp*]:
lift-root-step $\mathcal{F}$ *PRoot EStrictParallel* $\mathcal{R} = \mathcal{R}$
⟨*proof*⟩

lemma *lift-root-step-Parallel-conv*:
shows *lift-root-step* $\mathcal{F}$ *W EParallel* $\mathcal{R}$ = *lift-root-step* $\mathcal{F}$ *W EStrictParallel* $\mathcal{R} \cup$ *Restr Id* $(\mathcal{T}_G\ \mathcal{F})$

⟨proof⟩

lemma *relax-pos-lift-root-step*:
lift-root-step $\mathcal{F}$ *W X R* ⊆ *lift-root-step* $\mathcal{F}$ *PAny X R*
⟨proof⟩

lemma *relax-pos-lift-root-steps*:
(*lift-root-step* $\mathcal{F}$ *W X R*)$^+$ ⊆ (*lift-root-step* $\mathcal{F}$ *PAny X R*)$^+$
⟨proof⟩

lemma *relax-ext-lift-root-step*:
lift-root-step $\mathcal{F}$ *W X R* ⊆ *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* ⊆ (*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* ⊆ (*lift-root-step* $\mathcal{F}$ *PAny ESingle R*)$^+$ ∪ *Restr Id* ($\mathcal{T}_G\ \mathcal{F}$)
⟨proof⟩

lemma *lift-root-step-Single-to-Parallel*:
shows *lift-root-step* $\mathcal{F}$ *PAny ESingle R* ⊆ *lift-root-step* $\mathcal{F}$ *PAny EParallel R*
⟨proof⟩

lemma *trancl-partial-reflcl*:
(*X* ∪ *Restr Id Y*)$^+$ = X^+ ∪ *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 (*lift-root-step* $\mathcal{F}$ *PAny EParallel R*)$^+$ = (*lift-root-step* $\mathcal{F}$ *PAny ESingle R*)$^+$ ∪ *Restr Id* ($\mathcal{T}_G\ \mathcal{F}$)
⟨proof⟩

lemma *lift-root-Any-Single-eq*:
shows *lift-root-step* $\mathcal{F}$ *PAny ESingle R* = *R* ∪ *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* ∪ *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* $\mathcal{F}$ $\mathcal{R}$ ⊆ (*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* (λ *f* .(*Some f*, *Some f*)) *C*)⟨*u*⟩$_G$ = *gpair C*⟨*s*⟩$_G$ *C*⟨*t*⟩$_G$ ⟨*proof*⟩

lemma *gpair-gctxt′*:
assumes *gpair C*⟨*v*⟩$_G$ *C*⟨*w*⟩$_G$ = *u*
shows *u* = (*map-gctxt* (λ *f* .(*Some f*, *Some f*)) *C*)⟨*gpair v w*⟩$_G$
⟨*proof*⟩

lemma *gpair-gmctxt*:
assumes ∀ *i* < *length us*. *gpair* (*ss* ! *i*) (*ts* ! *i*) = *us* ! *i*
and *num-gholes C* = *length ss length ss* = *length ts length ts* = *length us*
shows *fill-gholes* (*map-gmctxt* (λ*f* . (*Some f*, *Some f*)) *C*) *us* = *gpair* (*fill-gholes C ss*) (*fill-gholes C ts*)
⟨*proof*⟩

lemma *gctxtex-onp-gpair-set-conv*:
{*gpair t u* |*t u*. (*t*, *u*) ∈ *gctxtex-onp P* $\mathcal{R}$} =

{(*map-gctxt* (λ *f* .(*Some f*, *Some f*)) *C*)$\langle s\rangle_G$ | *C s*. *P C* ∧ *s* ∈ {*gpair t u* |*t u*. (*t*, *u*) ∈ $\mathcal{R}$}} (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *gmctxtex-onp-gpair-set-conv*:
{*gpair t u* |*t u*. (*t*, *u*) ∈ *gmctxtex-onp P* $\mathcal{R}$} =
{*fill-gholes* (*map-gmctxt* (λ *f* .(*Some f*, *Some f*)) *C*) *ss* | *C ss*. *num-gholes C* = *length ss* ∧ *P C* ∧
(∀ *i* < *length ss*. *ss* ! *i* ∈ {*gpair t u* |*t u*. (*t*, *u*) ∈ $\mathcal{R}$})} (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

abbreviation *lift-sig-RR2* ≡ λ (*f*, *n*). ((*Some f*, *Some f*), *n*)
abbreviation *lift-fun* ≡ (λ *f*. (*Some f*, *Some f*))
abbreviation *unlift-fst* ≡ (λ *f*. *the* (*fst f*))
abbreviation *unlift-snd* ≡ (λ *f*. *the* (*snd f*))

lemma *RR2-gterm-unlift-lift-id* [*simp*]:
funas-gterm t ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *map-gterm* (*lift-fun* ∘ *unlift-fst*) *t* = *t*
⟨*proof*⟩

lemma *RR2-gterm-unlift-funas* [*simp*]:
funas-gterm t ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *funas-gterm* (*map-gterm unlift-fst t*) ⊆ $\mathcal{F}$
⟨*proof*⟩

lemma *gterm-funas-lift-RR2-funas* [*simp*]:
funas-gterm t ⊆ $\mathcal{F}$ ⟹ *funas-gterm* (*map-gterm lift-fun t*) ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
⟨*proof*⟩

lemma *RR2-gctxt-unlift-lift-id* [*simp*, *intro*]:
funas-gctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ (*map-gctxt* (*lift-fun* ∘ *unlift-fst*) *C*) = *C*
⟨*proof*⟩

lemma *RR2-gctxt-unlift-funas* [*simp*, *intro*]:
funas-gctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *funas-gctxt* (*map-gctxt unlift-fst C*) ⊆ $\mathcal{F}$
⟨*proof*⟩

lemma *gctxt-funas-lift-RR2-funas* [*simp*, *intro*]:
funas-gctxt C ⊆ $\mathcal{F}$ ⟹ *funas-gctxt* (*map-gctxt lift-fun C*) ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$
⟨*proof*⟩

lemma *RR2-gmctxt-unlift-lift-id* [*simp*, *intro*]:
funas-gmctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ (*map-gmctxt* (*lift-fun* ∘ *unlift-fst*) *C*) = *C*
⟨*proof*⟩

lemma *RR2-gmctxt-unlift-funas* [*simp*, *intro*]:
funas-gmctxt C ⊆ *lift-sig-RR2* ‘ $\mathcal{F}$ ⟹ *funas-gmctxt* (*map-gmctxt unlift-fst C*) ⊆

$\mathcal{F}$
⟨*proof*⟩

lemma *gmctxt-funas-lift-RR2-funas* [*simp*, *intro*]:
funas-gmctxt $C \subseteq \mathcal{F} \Longrightarrow$ *funas-gmctxt* (*map-gmctxt lift-fun* C) $\subseteq$ *lift-sig-RR2* ' $\mathcal{F}$
⟨*proof*⟩

lemma *RR2-gctxt-cl-to-gctxt*:
assumes $\bigwedge C.\ P\ C \Longrightarrow$ *funas-gctxt* $C \subseteq$ *lift-sig-RR2* ' $\mathcal{F}$
and $\bigwedge C.\ P\ C \Longrightarrow R$ (*map-gctxt unlift-fst* C)
and $\bigwedge C.\ R\ C \Longrightarrow P$ (*map-gctxt lift-fun* C)
shows $\{C\langle s\rangle_G \mid C\ s.\ P\ C \wedge Q\ s\} = \{($*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 $\bigwedge C.\ P\ C \Longrightarrow$ *funas-gmctxt* $C \subseteq$ *lift-sig-RR2* ' $\mathcal{F}$
and $\bigwedge C.\ P\ C \Longrightarrow R$ (*map-gmctxt* (λ *f*. *the* (*fst f*)) C)
and $\bigwedge C.\ R\ C \Longrightarrow P$ (*map-gmctxt* (λ *f*. (*Some f*, *Some f*)) C)
shows {*fill-gholes* C *ss* |C *ss*. *num-gholes* C = *length ss* $\wedge$ $P\ C$ $\wedge$ ($\forall$ $i <$ *length ss*. Q (*ss* ! i))} =
{*fill-gholes* (*map-gmctxt* (λ*f*. (*Some f*, *Some f*)) C) *ss* |C *ss*. *num-gholes* C = *length ss* $\wedge$
$R\ C$ $\wedge$ ($\forall$ $i <$ *length ss*. Q (*ss* ! i))} (**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *RR2-id-terms-gpair-set* [*simp*]:
$\mathcal{T}_G$ (*lift-sig-RR2* ' $\mathcal{F}$) = {*gpair t u* |*t u*. (*t*, *u*) $\in$ *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**
ggtt-lang F G $\equiv$ *map-both gterm-of-term* ' (*Restr* (*gtt-lang-terms G*) {*t*. *funas-term t* $\subseteq$ *fset F*})

lemma *ground-mctxt-map-vars-mctxt* [*simp*]:
ground-mctxt (*map-vars-mctxt f C*) = *ground-mctxt C*

⟨proof⟩

lemma *root-single-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* $\mathcal{A}$ (*lift-root-step* $\mathcal{F}$ *PRoot ESingle R*)
⟨proof⟩

lemma *root-strictparallel-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* $\mathcal{A}$ (*lift-root-step* $\mathcal{F}$ *PRoot EStrictParallel R*)
⟨proof⟩

lemma *reflcl-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*reflcl-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PRoot EParallel R*)
⟨proof⟩

lemma *parallel-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*parallel-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PAny EParallel R*)
⟨proof⟩

lemma *ctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*ctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PAny ESingle R*)
⟨proof⟩

lemma *mctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*mctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PAny EStrictParallel R*)
⟨proof⟩

lemma *nhole-ctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*nhole-ctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PNonRoot ESingle R*)
⟨proof⟩

lemma *nhole-mctxt-closure-automaton*:
assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*nhole-mctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PNonRoot EStrictParallel R*)
⟨proof⟩

lemma *nhole-mctxt-reflcl-automaton*:

assumes *RR2-spec* $\mathcal{A}$ *R*
shows *RR2-spec* (*nhole-mctxt-reflcl-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) $\mathcal{A}$) (*lift-root-step* (*fset* $\mathcal{F}$) *PNonRoot EParallel R*)
⟨*proof*⟩

definition *GTT-to-RR2-root* :: ('*q*, '*f*) *gtt* ⇒ (-, '*f option* × '*f option*) *ta* **where**
GTT-to-RR2-root $\mathcal{G}$ = *pair-automaton* (*fst* $\mathcal{G}$) (*snd* $\mathcal{G}$)

definition *GTT-to-RR2-root-reg* **where**
GTT-to-RR2-root-reg $\mathcal{G}$ = *Reg* (*map-both Some* |'| *fId-on* (*gtt-states* $\mathcal{G}$)) (*GTT-to-RR2-root* $\mathcal{G}$)

lemma *GTT-to-RR2-root*:
RR2-spec (*GTT-to-RR2-root-reg* $\mathcal{G}$) (*agtt-lang* $\mathcal{G}$)
⟨*proof*⟩

lemma *swap-GTT-to-RR2-root*:
gpair s t ∈ $\mathcal{L}$ (*GTT-to-RR2-root-reg* (*prod.swap* $\mathcal{G}$)) ⟷
gpair t s ∈ $\mathcal{L}$ (*GTT-to-RR2-root-reg* $\mathcal{G}$)
⟨*proof*⟩

lemma *funas-mctxt-map-vars-mctxt* [*simp*]:
funas-mctxt (*map-vars-mctxt f C*) = *funas-mctxt C*
⟨*proof*⟩

definition *GTT-to-RR2-reg* :: ('*f* × *nat*) *fset* ⇒ ('*q*, '*f*) *gtt* ⇒ (-, '*f option* × '*f option*) *reg* **where**
GTT-to-RR2-reg F G = *parallel-closure-reg* (*lift-sig-RR2* |'| *F*) (*GTT-to-RR2-root-reg G*)

lemma *agtt-lang-syms*:
gtt-syms $\mathcal{G}$ |⊆| $\mathcal{F}$ ⟹ *agtt-lang* $\mathcal{G}$ ⊆ {*t*. *funas-gterm t* ⊆ *fset* $\mathcal{F}$} × {*t*. *funas-gterm t* ⊆ *fset* $\mathcal{F}$}
⟨*proof*⟩

lemma *gtt-lang-from-agtt-lang*:
gtt-lang $\mathcal{G}$ = *lift-root-step UNIV PAny EParallel* (*agtt-lang* $\mathcal{G}$)
⟨*proof*⟩

lemma *GTT-to-RR2*:
assumes *gtt-syms* $\mathcal{G}$ |⊆| $\mathcal{F}$
shows *RR2-spec* (*GTT-to-RR2-reg* $\mathcal{F}$ $\mathcal{G}$) (*ggtt-lang* $\mathcal{F}$ $\mathcal{G}$)
⟨*proof*⟩

end
theory *FOL-Extra*
imports

Type-Instances-Impl
FOL−Fitting.FOL-Fitting
HOL−Library.FSet
begin

11 Additional support for FOL-Fitting

11.1 Iff

definition *Iff* **where**
Iff p q = And (Impl p q) (Impl q p)

lemma *eval-Iff*:
eval e f g (Iff p q) ⟷ (eval e f g p ⟷ eval e f g q)
⟨*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* ⇒ (′*a*, ′*b*) *form* ⇒ (′*a*, ′*b*) *form* **where**
apply-ctxt Hole p = p
| *apply-ctxt (And1 c v) p = And (apply-ctxt c p) v*
| *apply-ctxt (And2 u c) p = And u (apply-ctxt c p)*
| *apply-ctxt (Or1 c v) p = Or (apply-ctxt c p) v*
| *apply-ctxt (Or2 u c) p = Or u (apply-ctxt c p)*
| *apply-ctxt (Impl1 c v) p = Impl (apply-ctxt c p) v*
| *apply-ctxt (Impl2 u c) p = Impl u (apply-ctxt c p)*
| *apply-ctxt (Neg1 c) p = Neg (apply-ctxt c p)*
| *apply-ctxt (Forall1 c) p = Forall (apply-ctxt c p)*
| *apply-ctxt (Exists1 c) p = Exists (apply-ctxt c p)*

lemma *replace-subformula*:
assumes ⋀*e. eval e f g (Iff p q)*
shows *eval e f g (Iff (apply-ctxt c p) (apply-ctxt c q))*
⟨*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)))

⟨*proof*⟩

11.4 de Bruijn index manipulation for formulas; cf. *liftt*

primrec *liftti* :: *nat* ⇒ *'a term* ⇒ *'a term* **where**
liftti i (Var j) = (if i > j then Var j else Var (Suc j))
| *liftti i (App f ts) = App f (map (liftti i) ts)*

lemma *liftts-def'*:
liftts ts = map liftt ts
⟨*proof*⟩

liftt is a special case of *liftti*

lemma *lifttti-0*:
liftti 0 t = liftt t
⟨*proof*⟩

primrec *lifti* :: *nat* ⇒ (*'a*, *'b*) *form* ⇒ (*'a*, *'b*) *form* **where**
lifti i FF = FF
| *lifti i TT = TT*
| *lifti i (Pred b ts) = Pred b (map (liftti i) ts)*
| *lifti i (And p q) = And (lifti i p) (lifti i q)*
| *lifti i (Or p q) = Or (lifti i p) (lifti i q)*
| *lifti i (Impl p q) = Impl (lifti i p) (lifti i q)*
| *lifti i (Neg p) = Neg (lifti i p)*
| *lifti i (Forall p) = Forall (lifti (Suc i) p)*
| *lifti i (Exists p) = Exists (lifti (Suc i) p)*

abbreviation *lift* **where**
lift ≡ lifti 0

interaction of *lifti* and *eval*

lemma *evalts-def'*:
evalts e f ts = map (evalt e f) ts
⟨*proof*⟩

lemma *evalt-liftti*:
evalt (e⟨i:z⟩) f (liftti i t) = evalt e f t
⟨*proof*⟩

lemma *eval-lifti* [*simp*]:
eval ($e\langle i{:}z\rangle$) *f g* (*lifti i p*) = *eval e f g p*
⟨*proof*⟩

11.5 Quantifier Identities

lemma *quant-ids*:
eval e f g (*Iff* (*Neg* (*Exists p*)) (*Forall* (*Neg p*)))
eval e f g (*Iff* (*Neg* (*Forall p*)) (*Exists* (*Neg p*)))
eval e f g (*Iff* (*And p* (*Forall q*)) (*Forall* (*And* (*lift p*) *q*)))
eval e f g (*Iff* (*And p* (*Exists q*)) (*Exists* (*And* (*lift p*) *q*)))
eval e f g (*Iff* (*Or p* (*Forall q*)) (*Forall* (*Or* (*lift p*) *q*)))
eval e f g (*Iff* (*Or p* (*Exists q*)) (*Exists* (*Or* (*lift p*) *q*)))

⟨*proof*⟩

11.6 Function symbols and predicates, with arities.

primrec *predas-form* :: (*'a*, *'b*) *form* ⇒ (*'b* × *nat*) *set* **where**
predas-form FF = {}
| *predas-form TT* = {}
| *predas-form* (*Pred b ts*) = {(*b*, *length ts*)}
| *predas-form* (*And p q*) = *predas-form p* ∪ *predas-form q*
| *predas-form* (*Or p q*) = *predas-form p* ∪ *predas-form q*
| *predas-form* (*Impl p q*) = *predas-form p* ∪ *predas-form q*
| *predas-form* (*Neg p*) = *predas-form p*
| *predas-form* (*Forall p*) = *predas-form p*
| *predas-form* (*Exists p*) = *predas-form p*

primrec *funas-term* :: *'a term* ⇒ (*'a* × *nat*) *set* **where**
funas-term (*Var x*) = {}
| *funas-term* (*App f ts*) = {(*f*, *length ts*)} ∪ ⋃(*set* (*map funas-term ts*))

primrec *terms-form* :: (*'a*, *'b*) *form* ⇒ *'a term set* **where**
terms-form FF = {}
| *terms-form TT* = {}
| *terms-form* (*Pred b ts*) = *set ts*
| *terms-form* (*And p q*) = *terms-form p* ∪ *terms-form q*
| *terms-form* (*Or p q*) = *terms-form p* ∪ *terms-form q*
| *terms-form* (*Impl p q*) = *terms-form p* ∪ *terms-form q*
| *terms-form* (*Neg p*) = *terms-form p*
| *terms-form* (*Forall p*) = *terms-form p*
| *terms-form* (*Exists p*) = *terms-form p*

definition *funas-form* :: (*'a*, *'b*) *form* ⇒ (*'a* × *nat*) *set* **where**
funas-form f ≡ ⋃(*funas-term* ‘ *terms-form f*)

11.7 Negation Normal Form

inductive *is-nnf* :: (*'a*, *'b*) *form* ⇒ *bool* **where**

  is-nnf TT
| *is-nnf FF*
| *is-nnf (Pred p ts)*
| *is-nnf (Neg (Pred p ts))*
| *is-nnf p ⟹ is-nnf q ⟹ is-nnf (And p q)*
| *is-nnf p ⟹ is-nnf q ⟹ is-nnf (Or p q)*
| *is-nnf p ⟹ is-nnf (Forall p)*
| *is-nnf p ⟹ is-nnf (Exists p)*

primrec *nnf′* :: *bool* ⇒ (*′a*, *′b*) *form* ⇒ (*′a*, *′b*) *form* **where**
  nnf′ b TT = (*if b then TT else FF*)
| *nnf′ b FF* = (*if b then FF else TT*)
| *nnf′ b (Pred p ts)* = (*if b then id else Neg*) (*Pred p ts*)
| *nnf′ b (And p q)* = (*if b then And else Or*) (*nnf′ b p*) (*nnf′ b q*)
| *nnf′ b (Or p q)* = (*if b then Or else And*) (*nnf′ b p*) (*nnf′ b q*)
| *nnf′ b (Impl p q)* = (*if b then Or else And*) (*nnf′* (¬ *b*) *p*) (*nnf′ b q*)
| *nnf′ b (Neg p)* = *nnf′* (¬ *b*) *p*
| *nnf′ b (Forall p)* = (*if b then Forall else Exists*) (*nnf′ b p*)
| *nnf′ b (Exists p)* = (*if b then Exists else Forall*) (*nnf′ b p*)

lemma *eval-nnf′*:
  eval e f g (*nnf′ b p*) ⟷ (*eval e f g p* ⟷ *b*)
  ⟨*proof*⟩

lemma *is-nnf-nnf′*:
  is-nnf (*nnf′ b p*)
  ⟨*proof*⟩

abbreviation *nnf* **where**
  nnf ≡ *nnf′ True*

lemmas *nnf-simpls* [*simp*] = *eval-nnf′*[**where** *b* = *True*, *unfolded eq-True*] *is-nnf-nnf′*[**where** *b* = *True*]

11.8 Reasoning modulo ACI01

datatype (*′a*, *′b*) *form-aci*
  = *TT-aci*
  | *FF-aci*
  | *Pred-aci bool ′b ′a term list*
  | *And-aci* (*′a*, *′b*) *form-aci fset*
  | *Or-aci* (*′a*, *′b*) *form-aci fset*
  | *Forall-aci* (*′a*, *′b*) *form-aci*
  | *Exists-aci* (*′a*, *′b*) *form-aci*

evaluation, see *eval*

primrec *eval-aci* :: ‹(*nat* ⇒ *′c*) ⇒ (*′a* ⇒ *′c list* ⇒ *′c*) ⇒
  (*′b* ⇒ *′c list* ⇒ *bool*) ⇒ (*′a*, *′b*) *form-aci* ⇒ *bool*› **where**
  eval-aci e f g FF-aci ⟷ *False*
| *eval-aci e f g TT-aci* ⟷ *True*

| *eval-aci e f g* (*Pred-aci b a ts*) ⟷ (*g a* (*evalts e f ts*) ⟷ *b*)
| *eval-aci e f g* (*And-aci ps*) ⟷ *fBall* (*fimage* (*eval-aci e f g*) *ps*) *id*
| *eval-aci e f g* (*Or-aci ps*) ⟷ *fBex* (*fimage* (*eval-aci e f g*) *ps*) *id*
| *eval-aci e f g* (*Forall-aci p*) ⟷ (∀ *z*. *eval-aci* (*e*⟨*0*:*z*⟩) *f g p*)
| *eval-aci e f g* (*Exists-aci p*) ⟷ (∃ *z*. *eval-aci* (*e*⟨*0*:*z*⟩) *f g p*)

smart constructor: conjunction

fun *and-aci* **where**
and-aci FF-aci - = FF-aci
| *and-aci - FF-aci = FF-aci*
| *and-aci TT-aci q = q*
| *and-aci p TT-aci = p*
| *and-aci* (*And-aci ps*) (*And-aci qs*) = *And-aci* (*ps* |∪| *qs*)
| *and-aci* (*And-aci ps*) *q* = *And-aci* (*ps* |∪| {|*q*|})
| *and-aci p* (*And-aci qs*) = *And-aci* ({|*p*|} |∪| *qs*)
| *and-aci p q* = (*if p* = *q then p else And-aci* {|*p*,*q*|})

lemma *eval-and-aci* [*simp*]:
eval-aci e f g (*and-aci p q*) ⟷ *eval-aci e f g p* ∧ *eval-aci e f g q*
⟨*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* |∪| *qs*)
| *or-aci* (*Or-aci ps*) *q* = *Or-aci* (*ps* |∪| {|*q*|})
| *or-aci p* (*Or-aci qs*) = *Or-aci* ({|*p*|} |∪| *qs*)
| *or-aci p q* = (*if p* = *q then p else Or-aci* {|*p*,*q*|})

lemma *eval-or-aci* [*simp*]:
eval-aci e f g (*or-aci p q*) ⟷ *eval-aci e f g p* ∨ *eval-aci e f g q*
⟨*proof*⟩

declare *or-aci.simps* [*simp del*]

convert negation normal form to ACIU01 normal form

fun *nnf-to-aci* :: ('*a*, '*b*) *form* ⇒ ('*a*, '*b*) *form-aci* **where**
nnf-to-aci FF = FF-aci
| *nnf-to-aci TT = TT-aci*
| *nnf-to-aci* (*Pred b ts*) = *Pred-aci True b ts*
| *nnf-to-aci* (*Neg* (*Pred b ts*)) = *Pred-aci False b ts*
| *nnf-to-aci* (*And p q*) = *and-aci* (*nnf-to-aci p*) (*nnf-to-aci q*)
| *nnf-to-aci* (*Or p q*) = *or-aci* (*nnf-to-aci p*) (*nnf-to-aci q*)
| *nnf-to-aci* (*Forall p*) = *Forall-aci* (*nnf-to-aci p*)

| *nnf-to-aci* (*Exists p*) = *Exists-aci* (*nnf-to-aci p*)
| *nnf-to-aci* - = *undefined*

lemma *eval-nnf-to-aci*:
is-nnf p $\Longrightarrow$ *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*) $\Longrightarrow$ *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* :: ‹(*nat* ⇒ *'c*) ⇒ (*'a* ⇒ *'c list* ⇒ *'c*) ⇒ (*'b* ⇒ *'c list* ⇒ *bool*) ⇒ (*'a*, *'b*) *form-list-aci* ⇒ *bool*› **where**
eval-list-aci e f g FF-aci $\longleftrightarrow$ *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* (λ *fm. eval-list-aci e f g fm*) *ps*
| *eval-list-aci e f g* (*Or-aci ps*) $\longleftrightarrow$ *list-ex* (λ *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* ⇒ (*'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*) ⟷ *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*)) ⟹ *eval e f g p* ⟷ *eval e f g q*
  ⟨*proof*⟩

hide-type (**open**) *term*
hide-const (**open**) *Var*
hide-type (**open**) *ctxt*

end
theory *FOR-Semantics*
  imports *FOR-Certificate*
    Lift-Root-Step
    FOL−Fitting.FOL-Fitting
begin

12 Semantics of Relations

definition *is-to-trs* :: ('*f*, '*v*) *trs list* ⇒ *ftrs list* ⇒ ('*f*, '*v*) *trs* **where**
  is-to-trs Rs is = ⋃(*set* (*map* (*case-ftrs* ((!) *Rs*) ((') *prod.swap* ∘ (!) *Rs*)) *is*))

primrec *eval-gtt-rel* :: ('*f* × *nat*) *set* ⇒ ('*f*, '*v*) *trs list* ⇒ *ftrs gtt-rel* ⇒ '*f gterm rel* **where**
  eval-gtt-rel $\mathcal{F}$ *Rs* (*ARoot is*) = *Restr* (*grrstep* (*is-to-trs Rs is*)) ($\mathcal{T}_G$ $\mathcal{F}$)

| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*GInv g*) = *prod.swap* ‘ (*eval-gtt-rel* $\mathcal{F}$ *Rs g*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*AUnion g1 g2*) = (*eval-gtt-rel* $\mathcal{F}$ *Rs g1*) ∪ (*eval-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*ATrancl g*) = (*eval-gtt-rel* $\mathcal{F}$ *Rs g*)$^+$
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*AComp g1 g2*) = (*eval-gtt-rel* $\mathcal{F}$ *Rs g1*) *O* (*eval-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*GTrancl g*) = *gtrancl-rel* $\mathcal{F}$ (*eval-gtt-rel* $\mathcal{F}$ *Rs g*)
| *eval-gtt-rel* $\mathcal{F}$ *Rs* (*GComp g1 g2*) = *gcomp-rel* $\mathcal{F}$ (*eval-gtt-rel* $\mathcal{F}$ *Rs g1*) (*eval-gtt-rel* $\mathcal{F}$ *Rs g2*)

primrec *eval-rr1-rel* :: (*'f* × *nat*) *set* ⇒ (*'f*, *'v*) *trs list* ⇒ *ftrs rr1-rel* ⇒ *'f gterm set*
and *eval-rr2-rel* :: (*'f* × *nat*) *set* ⇒ (*'f*, *'v*) *trs list* ⇒ *ftrs rr2-rel* ⇒ *'f gterm rel*
where
eval-rr1-rel $\mathcal{F}$ *Rs R1Terms* = ($\mathcal{T}_G$ $\mathcal{F}$)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Union R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) ∪ (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Inter R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) ∩ (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Diff R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) − (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Proj n R*) = (*case n of 0* ⇒ *fst* ‘ (*eval-rr2-rel* $\mathcal{F}$ *Rs R*)
| - ⇒ *snd* ‘ (*eval-rr2-rel* $\mathcal{F}$ *Rs R*))
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1NF is*) = *NF* (*Restr* (*grstep* (*is-to-trs Rs is*)) ($\mathcal{T}_G$ $\mathcal{F}$)) ∩ ($\mathcal{T}_G$ $\mathcal{F}$)
| *eval-rr1-rel* $\mathcal{F}$ *Rs* (*R1Inf R*) = {*s. infinite* (*eval-rr2-rel* $\mathcal{F}$ *Rs R* “ {*s*})}
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2GTT-Rel A W X*) = *lift-root-step* $\mathcal{F}$ *W X* (*eval-gtt-rel* $\mathcal{F}$ *Rs A*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Inv R*) = *prod.swap* ‘ (*eval-rr2-rel* $\mathcal{F}$ *Rs R*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Union R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) ∪ (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Inter R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) ∩ (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Diff R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) − (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Comp R S*) = (*eval-rr2-rel* $\mathcal{F}$ *Rs R*) *O* (*eval-rr2-rel* $\mathcal{F}$ *Rs S*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Diag R*) = *Id-on* (*eval-rr1-rel* $\mathcal{F}$ *Rs R*)
| *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Prod R S*) = (*eval-rr1-rel* $\mathcal{F}$ *Rs R*) × (*eval-rr1-rel* $\mathcal{F}$ *Rs S*)

12.1 Semantics of Formulas

fun *eval-formula* :: (*'f* × *nat*) *set* ⇒ (*'f*, *'v*) *trs list* ⇒ (*nat* ⇒ *'f gterm*) ⇒
ftrs formula ⇒ *bool* **where**
eval-formula $\mathcal{F}$ *Rs* α (*FRR1 r1 x*) ⟷ α *x* ∈ *eval-rr1-rel* $\mathcal{F}$ *Rs r1*
| *eval-formula* $\mathcal{F}$ *Rs* α (*FRR2 r2 x y*) ⟷ (α *x*, α *y*) ∈ *eval-rr2-rel* $\mathcal{F}$ *Rs r2*
| *eval-formula* $\mathcal{F}$ *Rs* α (*FAnd fs*) ⟷ (∀ *f* ∈ *set fs*. *eval-formula* $\mathcal{F}$ *Rs* α *f*)
| *eval-formula* $\mathcal{F}$ *Rs* α (*FOr fs*) ⟷ (∃ *f* ∈ *set fs*. *eval-formula* $\mathcal{F}$ *Rs* α *f*)
| *eval-formula* $\mathcal{F}$ *Rs* α (*FNot f*) ⟷ ¬ *eval-formula* $\mathcal{F}$ *Rs* α *f*
| *eval-formula* $\mathcal{F}$ *Rs* α (*FExists f*) ⟷ (∃ *z* ∈ $\mathcal{T}_G$ $\mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* (α⟨*0* : *z*⟩) *f*)
| *eval-formula* $\mathcal{F}$ *Rs* α (*FForall f*) ⟷ (∀ *z* ∈ $\mathcal{T}_G$ $\mathcal{F}$. *eval-formula* $\mathcal{F}$ *Rs* (α⟨*0* : *z*⟩) *f*)

fun *formula-arity* :: *'trs formula* ⇒ *nat* **where**
formula-arity (*FRR1 r1 x*) = *Suc x*
| *formula-arity* (*FRR2 r2 x y*) = *max* (*Suc x*) (*Suc y*)
| *formula-arity* (*FAnd fs*) = *max-list* (*map formula-arity fs*)
| *formula-arity* (*FOr fs*) = *max-list* (*map formula-arity fs*)
| *formula-arity* (*FNot f*) = *formula-arity f*
| *formula-arity* (*FExists f*) = *formula-arity f* − *1*
| *formula-arity* (*FForall f*) = *formula-arity f* − *1*

lemma *R1NF-reps*:
assumes *funas-trs* $R \subseteq \mathcal{F}$ ∀ *t*. (*term-of-gterm s*, *term-of-gterm t*) ∈ *rstep R* ⟶ ¬*funas-gterm t* $\subseteq \mathcal{F}$
and *funas-gterm s* $\subseteq \mathcal{F}$ (*l*, *r*) ∈ *R* *term-of-gterm s* = *C*⟨*l* · (*σ* :: *'b* ⇒ (*'a*, *'b*) *Term.term*)⟩
shows *False*
⟨*proof*⟩

The central property we are interested in is satisfiability

definition *formula-satisfiable* **where**
formula-satisfiable $\mathcal{F}$ *Rs f* ⟷ (∃ *α*. *range α* ⊆ $\mathcal{T}_G$ $\mathcal{F}$ ∧ *eval-formula* $\mathcal{F}$ *Rs α f*)

12.2 Validation

12.3 Defining properties of *gcomp-rel* and *gtrancl-rel*

lemma *gcomp-rel-sig*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$ **and** $S \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *gcomp-rel* $\mathcal{F}$ *R S* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
⟨*proof*⟩

lemma *gtrancl-rel-sig*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *gtrancl-rel* $\mathcal{F}$ $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
⟨*proof*⟩

lemma *gtrancl-rel*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *lift-root-step* $\mathcal{F}$ *PAny EStrictParallel* (*gtrancl-rel* $\mathcal{F}$ *R*) = (*lift-root-step* $\mathcal{F}$ *PAny ESingle R*)$^+$
⟨*proof*⟩

lemma *gtrancl-rel'*:
assumes $R \subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
shows *lift-root-step* $\mathcal{F}$ *PAny EParallel* (*gtrancl-rel* $\mathcal{F}$ *R*) = *Restr* ((*lift-root-step* $\mathcal{F}$ *PAny ESingle R*)*) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

GTT relation semantics respects the signature

lemma *eval-gtt-rel-sig*:
eval-gtt-rel $\mathcal{F}$ *Rs g* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
⟨*proof*⟩

RR1 and RR2 relation semantics respect the signature

lemma *eval-rr12-rel-sig*:
eval-rr1-rel $\mathcal{F}$ *Rs r1* $\subseteq \mathcal{T}_G\ \mathcal{F}$
eval-rr2-rel $\mathcal{F}$ *Rs r2* $\subseteq \mathcal{T}_G\ \mathcal{F} \times \mathcal{T}_G\ \mathcal{F}$
⟨*proof*⟩

12.4 Correctness of derived constructions

lemma *R1Fin*:
eval-rr1-rel $\mathcal{F}$ *Rs* (*R1Fin r*) = {*t* ∈ $\mathcal{T}_G\ \mathcal{F}$. *finite* {*s*. (*t*, *s*) ∈ *eval-rr2-rel* $\mathcal{F}$ *Rs r*}}
⟨*proof*⟩

lemma *R2Eq*:
eval-rr2-rel $\mathcal{F}$ *Rs R2Eq* = *Id-on* ($\mathcal{T}_G\ \mathcal{F}$)
⟨*proof*⟩

lemma *R2Reflc*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2Reflc r*) = *eval-rr2-rel* $\mathcal{F}$ *Rs r* ∪ *Id-on* ($\mathcal{T}_G\ \mathcal{F}$)
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2Reflc r*) = *Restr* ((*eval-rr2-rel* $\mathcal{F}$ *Rs r*)$^=$) ($\mathcal{T}_G\ \mathcal{F}$)
⟨*proof*⟩

lemma *R2Step*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2Step ts*) = *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G\ \mathcal{F}$)
⟨*proof*⟩

lemma *R2StepEq*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2StepEq ts*) = *Restr* ((*grstep* (*is-to-trs Rs ts*))$^=$) ($\mathcal{T}_G\ \mathcal{F}$)
⟨*proof*⟩

lemma *R2Steps*:
fixes $\mathcal{F}$ *Rs ts* **defines** *R* ≡ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G\ \mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Steps ts*) = R^+
⟨*proof*⟩

lemma *R2StepsEq*:
fixes $\mathcal{F}$ *Rs ts* **defines** *R* ≡ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G\ \mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2StepsEq ts*) = *Restr* (R^*) ($\mathcal{T}_G\ \mathcal{F}$)
⟨*proof*⟩

lemma *R2StepsNF*:
fixes $\mathcal{F}$ *Rs ts* **defines** *R* ≡ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G\ \mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2StepsNF ts*) = *Restr* (R^* ∩ *UNIV* × *NF R*) ($\mathcal{T}_G\ \mathcal{F}$)

⟨*proof*⟩

lemma *R2ParStep*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2ParStep ts*) = *Restr* (*gpar-rstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *R2RootStep*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2RootStep ts*) = *Restr* (*grrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *R2RootStepEq*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2RootStepEq ts*) = *Restr* ((*grrstep* (*is-to-trs Rs ts*))$^=$) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *R2RootSteps*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2RootSteps ts*) = R^+
⟨*proof*⟩

lemma *R2RootStepsEq*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2RootStepsEq ts*) = *Restr* (R^*) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *R2NonRootStep*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2NonRootStep ts*) = *Restr* (*gnrrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *R2NonRootStepEq*:
eval-rr2-rel $\mathcal{F}$ *Rs* (*R2NonRootStepEq ts*) = *Restr* ((*gnrrstep* (*is-to-trs Rs ts*))$^=$) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *R2NonRootSteps*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*gnrrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2NonRootSteps ts*) = R^+
⟨*proof*⟩

lemma *R2NonRootStepsEq*:
fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*gnrrstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2NonRootStepsEq ts*) = *Restr* (R^*) ($\mathcal{T}_G$ $\mathcal{F}$)
⟨*proof*⟩

lemma *converse-to-prod-swap*:
R^{-1} = *prod.swap* ‘ *R*
⟨*proof*⟩

lemma *R2Meet*:
  fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
  shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Meet ts*) = *Restr* $((R^{-1})^*\ O\ R^*)$ ($\mathcal{T}_G$ $\mathcal{F}$)
  ⟨*proof*⟩

lemma *R2Join*:
  fixes $\mathcal{F}$ *Rs ts* **defines** $R \equiv$ *Restr* (*grstep* (*is-to-trs Rs ts*)) ($\mathcal{T}_G$ $\mathcal{F}$)
  shows *eval-rr2-rel* $\mathcal{F}$ *Rs* (*R2Join ts*) = *Restr* $(R^*\ O\ (R^{-1})^*)$ ($\mathcal{T}_G$ $\mathcal{F}$)
  ⟨*proof*⟩

end
theory *FOR-Check*
  imports
    FOR-Semantics
    FOL-Extra
    GTT-RRn
    First-Order-Terms.Option-Monad
    LV-to-GTT
    NF
    Regular-Tree-Relations.GTT-Transitive-Closure
    Regular-Tree-Relations.AGTT
    Regular-Tree-Relations.RR2-Infinite-Q-infinity
    Regular-Tree-Relations.RRn-Automata
begin

13 Check inference steps

type-synonym ('*f*, '*v*) *fin-trs* = ('*f*, '*v*) *rule fset*

lemma *tl-drop-conv*:
  tl xs = *drop 1 xs*
  ⟨*proof*⟩

definition *rrn-drop-fst* **where**
  rrn-drop-fst $\mathcal{A}$ = *relabel-reg* (*trim-reg* (*collapse-automaton-reg* (*fmap-funs-reg* (*drop-none-rule 1*) (*trim-reg* $\mathcal{A}$))))

lemma *rrn-drop-fst-lang*:
  assumes *RRn-spec n A T* $1 < n$
  shows *RRn-spec* $(n - 1)$ (*rrn-drop-fst A*) (*drop 1* ‘ *T*)
  ⟨*proof*⟩

definition *liftO1* :: ('*a* ⇒ '*b*) ⇒ '*a option* ⇒ '*b option* **where**
  liftO1 = *map-option*

definition *liftO2* :: ('*a* ⇒ '*b* ⇒ '*c*) ⇒ '*a option* ⇒ '*b option* ⇒ '*c option* **where**
  liftO2 f a b = *case-option None* (λ*a*'. *liftO1* (*f a*') *b*) *a*

lemma *liftO1-Some* [*simp*]:
$liftO1\ f\ x = Some\ y \longleftrightarrow (\exists x'.\ x = Some\ x') \wedge y = f\ (the\ x)$
⟨*proof*⟩

lemma *liftO2-Some* [*simp*]:
$liftO2\ f\ x\ y = Some\ z \longleftrightarrow (\exists x'\ y'.\ x = Some\ x' \wedge y = Some\ y') \wedge z = f\ (the\ x)\ (the\ y)$
⟨*proof*⟩

13.1 Computing TRSs

lemma *is-to-trs-props*:
assumes $\forall\ R \in set\ Rs.\ finite\ R \wedge lv\text{-}trs\ R \wedge funas\text{-}trs\ R \subseteq \mathcal{F}$ $\forall i \in set\ is.\ case\text{-}ftrs\ id\ id\ i < length\ Rs$
shows $funas\text{-}trs\ (is\text{-}to\text{-}trs\ Rs\ is) \subseteq \mathcal{F}$ $lv\text{-}trs\ (is\text{-}to\text{-}trs\ Rs\ is)$ $finite\ (is\text{-}to\text{-}trs\ Rs\ is)$
⟨*proof*⟩

definition *is-to-fin-trs* :: $('f, 'v)\ fin\text{-}trs\ list \Rightarrow ftrs\ list \Rightarrow ('f, 'v)\ fin\text{-}trs$ **where**
$is\text{-}to\text{-}fin\text{-}trs\ Rs\ is = |\bigcup|\ (fset\text{-}of\text{-}list\ (map\ (case\text{-}ftrs\ ((!)\ Rs)\ ((|^{-1}|)\ prod.swap \circ (!)\ Rs))\ is))$

lemma *is-to-fin-trs-conv*:
assumes $\forall i \in set\ is.\ case\text{-}ftrs\ id\ id\ i < length\ Rs$
shows $is\text{-}to\text{-}trs\ (map\ fset\ Rs)\ is = fset\ (is\text{-}to\text{-}fin\text{-}trs\ Rs\ is)$
⟨*proof*⟩

definition *is-to-trs'* :: $('f, 'v)\ fin\text{-}trs\ list \Rightarrow ftrs\ list \Rightarrow ('f, 'v)\ fin\text{-}trs\ option$ **where**
$is\text{-}to\text{-}trs'\ Rs\ is = do\ \{$
$guard\ (\forall i \in set\ is.\ case\text{-}ftrs\ id\ id\ i < length\ Rs);$
$Some\ (is\text{-}to\text{-}fin\text{-}trs\ Rs\ is)$
$\}$

lemma *is-to-trs-conv*:
$is\text{-}to\text{-}trs'\ Rs\ is = Some\ S \Longrightarrow is\text{-}to\text{-}trs\ (map\ fset\ Rs)\ is = fset\ S$
⟨*proof*⟩

lemma *is-to-trs'-props*:
assumes $\forall\ R \in set\ Rs.\ lv\text{-}trs\ (fset\ R) \wedge ffunas\text{-}trs\ R\ |\subseteq|\ \mathcal{F}$ **and** $is\text{-}to\text{-}trs'\ Rs\ is = Some\ S$
shows $ffunas\text{-}trs\ S\ |\subseteq|\ \mathcal{F}$ $lv\text{-}trs\ (fset\ S)$
⟨*proof*⟩

13.2 Computing GTTs

fun *gtt-of-gtt-rel* :: $('f \times nat)\ fset \Rightarrow ('f :: linorder, 'v)\ fin\text{-}trs\ list \Rightarrow ftrs\ gtt\text{-}rel \Rightarrow (nat, 'f)\ gtt\ option$ **where**

gtt-of-gtt-rel $\mathcal{F}$ *Rs* (*ARoot is*) = *liftO1* (λR. *relabel-gtt* (*agtt-grrstep R* $\mathcal{F}$)) (*is-to-trs′ Rs is*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*GInv g*) = *liftO1 prod.swap* (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*AUnion g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-union′ g1 g2*)) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*ATrancl g*) = *liftO1* (*relabel-gtt* ∘ *AGTT-trancl*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*GTrancl g*) = *liftO1 GTT-trancl* (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*AComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-comp′ g1 g2*)) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel* $\mathcal{F}$ *Rs* (*GComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*GTT-comp′ g1 g2*)) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g2*)

lemma *gtt-of-gtt-rel-correct*:
assumes ∀ *R* ∈ *set Rs*. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| $\mathcal{F}$
shows *gtt-of-gtt-rel* $\mathcal{F}$ *Rs g* = *Some g′* ⟹ *agtt-lang g′* = *eval-gtt-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *g*
⟨*proof*⟩

13.3 Computing RR1 and RR2 relations

definition *simplify-reg* $\mathcal{A}$ = (*relabel-reg* (*trim-reg* $\mathcal{A}$))

lemma $\mathcal{L}$*-simplify-reg* [*simp*]: $\mathcal{L}$ (*simplify-reg* $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$
⟨*proof*⟩

lemma *RR1-spec-simplify-reg*[*simp*]:
RR1-spec (*simplify-reg* $\mathcal{A}$) *R* = *RR1-spec* $\mathcal{A}$ *R*
⟨*proof*⟩
lemma *RR2-spec-simplify-reg*[*simp*]:
RR2-spec (*simplify-reg* $\mathcal{A}$) *R* = *RR2-spec* $\mathcal{A}$ *R*
⟨*proof*⟩
lemma *RRn-spec-simplify-reg*[*simp*]:
RRn-spec n (*simplify-reg* $\mathcal{A}$) *R* = *RRn-spec n* $\mathcal{A}$ *R*
⟨*proof*⟩

lemma *RR1-spec-eps-free-reg*[*simp*]:
RR1-spec (*eps-free-reg* $\mathcal{A}$) *R* = *RR1-spec* $\mathcal{A}$ *R*
⟨*proof*⟩
lemma *RR2-spec-eps-free-reg*[*simp*]:
RR2-spec (*eps-free-reg* $\mathcal{A}$) *R* = *RR2-spec* $\mathcal{A}$ *R*
⟨*proof*⟩
lemma *RRn-spec-eps-free-reg*[*simp*]:
RRn-spec n (*eps-free-reg* $\mathcal{A}$) *R* = *RRn-spec n* $\mathcal{A}$ *R*
⟨*proof*⟩

fun *rr1-of-rr1-rel* :: (′*f* × *nat*) *fset* ⇒ (′*f* :: *linorder*, ′*v*) *fin-trs list* ⇒ *ftrs rr1-rel* ⇒ (*nat*, ′*f*) *reg option*

and *rr2-of-rr2-rel* :: ('*f* × *nat*) *fset* ⇒ ('*f*, '*v*) *fin-trs list* ⇒ *ftrs rr2-rel* ⇒ (*nat*, '*f* *option* × '*f option*) *reg option* **where**
rr1-of-rr1-rel $\mathcal{F}$ *Rs R1Terms* = *Some* (*relabel-reg* (*term-reg* $\mathcal{F}$))
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1NF is*) = *liftO1* (λ*R*. (*simplify-reg* (*nf-reg* (*fst* |'| *R*) $\mathcal{F}$)))
(*is-to-trs' Rs is*)
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1Inf r*) = *liftO1* (λ*R*.
let $\mathcal{A}$ = *trim-reg R in*
simplify-reg (*proj-1-reg* (*Inf-reg-impl* $\mathcal{A}$))
) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r*)
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1Proj i r*) = (*case i of 0* ⇒
liftO1 (*trim-reg* ∘ *proj-1-reg*) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r*)
| - ⇒ *liftO1* (*trim-reg* ∘ *proj-2-reg*) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r*))
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1Union s1 s2*) =
liftO2 (λ *x y*. *relabel-reg* (*reg-union x y*)) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s2*)
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1Inter s1 s2*) =
liftO2 (λ *x y*. *simplify-reg* (*reg-intersect x y*)) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s2*)
| *rr1-of-rr1-rel* $\mathcal{F}$ *Rs* (*R1Diff s1 s2*) = *liftO2* (λ *x y*. *relabel-reg* (*trim-reg* (*difference-reg x y*))) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s2*)

| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2GTT-Rel g w x*) =
(*case w of PRoot* ⇒
(*case x of ESingle* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *EParallel* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *reflcl-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *EStrictParallel* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*))
| *PNonRoot* ⇒
(*case x of ESingle* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *nhole-ctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *EParallel* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *nhole-mctxt-reflcl-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *EStrictParallel* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *nhole-mctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*))
| *PAny* ⇒
(*case x of ESingle* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *ctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *EParallel* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *parallel-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)
| *EStrictParallel* ⇒ *liftO1* (*simplify-reg* ∘ *eps-free-reg* ∘ *mctxt-closure-reg* (*lift-sig-RR2* |'| $\mathcal{F}$) ∘ *GTT-to-RR2-root-reg*) (*gtt-of-gtt-rel* $\mathcal{F}$ *Rs g*)))
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Diag s*) =
liftO1 (λ *x*. *fmap-funs-reg* (λ*f*. (*Some f*, *Some f*)) *x*) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s*)
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Prod s1 s2*) =
liftO2 (λ *x y*. *simplify-reg* (*pair-automaton-reg x y*)) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s1*) (*rr1-of-rr1-rel* $\mathcal{F}$ *Rs s2*)
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Inv r*) = *liftO1* (*fmap-funs-reg prod.swap*) (*rr2-of-rr2-rel*

$\mathcal{F}$ *Rs r*)
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Union r1 r2*) =
liftO2 (λ *x y. relabel-reg* (*reg-union x y*)) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2*)
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Inter r1 r2*) =
liftO2 (λ *x y. simplify-reg* (*reg-intersect x y*)) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2*)
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Diff r1 r2*) = *liftO2* (λ *x y. simplify-reg* (*difference-reg x y*)) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2*)
| *rr2-of-rr2-rel* $\mathcal{F}$ *Rs* (*R2Comp r1 r2*) = *liftO2* (λ *x y. simplify-reg* (*rr2-compositon* $\mathcal{F}$ *x y*))
(*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2*)

abbreviation *lhss* **where**
lhss R $\equiv$ *fst* |'| *R*

lemma *rr12-of-rr12-rel-correct*:
fixes *Rs* :: (('*f* :: *linorder*, '*v*) *Term.term* $\times$ ('*f*, '*v*) *Term.term*) *fset list*
assumes $\forall$ *R* $\in$ *set Rs. lv-trs* (*fset R*) $\wedge$ *ffunas-trs R* |$\subseteq$| $\mathcal{F}$
shows $\forall$ *ta1. rr1-of-rr1-rel* $\mathcal{F}$ *Rs r1* = *Some ta1* $\longrightarrow$ *RR1-spec ta1* (*eval-rr1-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r1*)
$\forall$ *ta2. rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2* = *Some ta2* $\longrightarrow$ *RR2-spec ta2* (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r2*)
⟨*proof*⟩

13.4 Misc

lemma *eval-formula-arity-cong*:
assumes $\bigwedge$*i. i* < *formula-arity f* $\Longrightarrow$ α' *i* = α *i*
shows *eval-formula* $\mathcal{F}$ *Rs* α' *f* = *eval-formula* $\mathcal{F}$ *Rs* α *f*
⟨*proof*⟩

13.5 Connect semantics to FOL-Fitting

primrec *form-of-formula* :: '*trs formula* $\Rightarrow$ (*unit*, '*trs rr1-rel* + '*trs rr2-rel*) *form*
where
form-of-formula (*FRR1 r1 x*) = *Pred* (*Inl r1*) [*Var x*]
| *form-of-formula* (*FRR2 r2 x y*) = *Pred* (*Inr r2*) [*Var x*, *Var y*]
| *form-of-formula* (*FAnd fs*) = *foldr And* (*map form-of-formula fs*) *TT*
| *form-of-formula* (*FOr fs*) = *foldr Or* (*map form-of-formula fs*) *FF*
| *form-of-formula* (*FNot f*) = *Neg* (*form-of-formula f*)
| *form-of-formula* (*FExists f*) = *Exists* (*And* (*Pred* (*Inl R1Terms*) [*Var 0*]) (*form-of-formula f*))
| *form-of-formula* (*FForall f*) = *Forall* (*Impl* (*Pred* (*Inl R1Terms*) [*Var 0*]) (*form-of-formula f*))

fun *for-eval-rel* :: ('*f* $\times$ *nat*) *set* $\Rightarrow$ ('*f*, '*v*) *trs list* $\Rightarrow$ *ftrs rr1-rel* + *ftrs rr2-rel* $\Rightarrow$ '*f gterm list* $\Rightarrow$ *bool* **where**

$\mathit{for\text{-}eval\text{-}rel}\ \mathcal{F}\ \mathit{Rs}\ (\mathit{Inl}\ \mathit{r1})\ [t] \longleftrightarrow t \in \mathit{eval\text{-}rr1\text{-}rel}\ \mathcal{F}\ \mathit{Rs}\ \mathit{r1}$
$\mid \mathit{for\text{-}eval\text{-}rel}\ \mathcal{F}\ \mathit{Rs}\ (\mathit{Inr}\ \mathit{r2})\ [t,\ u] \longleftrightarrow (t,\ u) \in \mathit{eval\text{-}rr2\text{-}rel}\ \mathcal{F}\ \mathit{Rs}\ \mathit{r2}$

lemma *eval-formula-conv*:
$\mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha\ f = \mathit{eval}\ \alpha\ \mathit{undefined}\ (\mathit{for\text{-}eval\text{-}rel}\ \mathcal{F}\ \mathit{Rs})\ (\mathit{form\text{-}of\text{-}formula}\ f)$
⟨*proof*⟩

13.6 RRn relations and formulas

lemma *shift-rangeI* [*intro*!]:
$\mathit{range}\ \alpha \subseteq T \Longrightarrow x \in T \Longrightarrow \mathit{range}\ (\mathit{shift}\ \alpha\ i\ x) \subseteq T$
⟨*proof*⟩

definition *formula-relevant* **where**
$\mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ \mathit{fm} \longleftrightarrow$
$(\forall \alpha\ \alpha'.\ \mathit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \longrightarrow \mathit{range}\ \alpha' \subseteq \mathcal{T}_G\ \mathcal{F} \longrightarrow \mathit{map}\ \alpha\ \mathit{vs} = \mathit{map}\ \alpha'\ \mathit{vs} \longrightarrow \mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha\ \mathit{fm} \longrightarrow \mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha'\ \mathit{fm})$

lemma *formula-relevant-mono*:
$\mathit{set}\ \mathit{vs} \subseteq \mathit{set}\ \mathit{ws} \Longrightarrow \mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ \mathit{fm} \Longrightarrow \mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ \mathit{ws}\ \mathit{fm}$
⟨*proof*⟩

lemma *formula-relevantD*:
$\mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ \mathit{fm} \Longrightarrow$
$\mathit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \Longrightarrow \mathit{range}\ \alpha' \subseteq \mathcal{T}_G\ \mathcal{F} \Longrightarrow \mathit{map}\ \alpha\ \mathit{vs} = \mathit{map}\ \alpha'\ \mathit{vs} \Longrightarrow$
$\mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha\ \mathit{fm} \Longrightarrow \mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha'\ \mathit{fm}$
⟨*proof*⟩

lemma *trivial-formula-relevant*:
assumes $\bigwedge\alpha.\ \mathit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \Longrightarrow \neg\ \mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha\ \mathit{fm}$
shows $\mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ \mathit{fm}$
⟨*proof*⟩

lemma *formula-relevant-0-FExists*:
assumes $\mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ [0]\ \mathit{fm}$
shows $\mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ []\ (\mathit{FExists}\ \mathit{fm})$
⟨*proof*⟩

definition *formula-spec* **where**
$\mathit{formula\text{-}spec}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ A\ \mathit{fm} \longleftrightarrow \mathit{sorted}\ \mathit{vs} \wedge \mathit{distinct}\ \mathit{vs} \wedge$
$\mathit{formula\text{-}relevant}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ \mathit{fm} \wedge$
$\mathit{RRn\text{-}spec}\ (\mathit{length}\ \mathit{vs})\ A\ \{\mathit{map}\ \alpha\ \mathit{vs}\ |\alpha.\ \mathit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge \mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha\ \mathit{fm}\}$

lemma *formula-spec-RRn-spec*:
$\mathit{formula\text{-}spec}\ \mathcal{F}\ \mathit{Rs}\ \mathit{vs}\ A\ \mathit{fm} \Longrightarrow \mathit{RRn\text{-}spec}\ (\mathit{length}\ \mathit{vs})\ A\ \{\mathit{map}\ \alpha\ \mathit{vs}\ |\alpha.\ \mathit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge \mathit{eval\text{-}formula}\ \mathcal{F}\ \mathit{Rs}\ \alpha\ \mathit{fm}\}$

⟨*proof*⟩

lemma *formula-spec-nt-empty-form-sat*:
$\neg$ *reg-empty A* $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ $\exists$ α. *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\wedge$ *eval-formula* $\mathcal{F}$ *Rs* α *fm*
⟨*proof*⟩

lemma *formula-spec-empty*:
reg-empty A $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ *range* $\alpha \subseteq \mathcal{T}_G$ $\mathcal{F}$ $\Longrightarrow$ *eval-formula* $\mathcal{F}$ *Rs* α *fm* $\longleftrightarrow$ *False*
⟨*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*:
sorted vs $\Longrightarrow$ *distinct vs* $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs empty-reg FFalse*
⟨*proof*⟩

lemma *true-formula-spec*:
assumes *vs* $\neq$ [] $\vee$ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) $\neq$ {} *sorted vs distinct vs*
shows *formula-spec* (*fset* $\mathcal{F}$) *Rs vs* (*true-RRn* $\mathcal{F}$ (*length vs*)) *FTrue*
⟨*proof*⟩

lemma *relabel-formula-spec*:
formula-spec $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs* (*relabel-reg A*) *fm*
⟨*proof*⟩

lemma *trim-formula-spec*:
formula-spec $\mathcal{F}$ *Rs vs A fm* $\Longrightarrow$ *formula-spec* $\mathcal{F}$ *Rs vs* (*trim-reg A*) *fm*
⟨*proof*⟩

definition *fit-permute* :: *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* **where**
fit-permute vs vs′ vs″ = *map* (λ*v*. *if v* $\in$ *set vs then the* (*mem-idx v vs*) *else length vs* + *the* (*mem-idx v vs″*)) *vs′*

definition *fit-rrn* :: (*′f* $\times$ *nat*) *fset* $\Rightarrow$ *nat list* $\Rightarrow$ *nat list* $\Rightarrow$ (*nat*, *′f option list*) *reg* $\Rightarrow$ (-, *′f option list*) *reg* **where**
fit-rrn $\mathcal{F}$ *vs vs′ A* = (*let vs″* = *subtract-list-sorted vs′ vs in*
fmap-funs-reg (λ*fs*. *map* ((!) *fs*) (*fit-permute vs vs′ vs″*))
(*fmap-funs-reg* (*pad-with-Nones* (*length vs*) (*length vs″*)) (*pair-automaton-reg A* (*true-RRn* $\mathcal{F}$ (*length vs″*)))))

lemma *the-mem-idx-simp* [*simp*]:
distinct xs $\Longrightarrow$ *i* < *length xs* $\Longrightarrow$ *the* (*mem-idx* (*xs* ! *i*) *xs*) = *i*
⟨*proof*⟩

lemma *fit-rrn*:

assumes *spec*: *formula-spec* (*fset* $\mathcal{F}$) *Rs vs A fm* **and** *vs*: *sorted vs′ distinct vs′ set vs* ⊆ *set vs′*
shows *formula-spec* (*fset* $\mathcal{F}$) *Rs vs′* (*fit-rrn* $\mathcal{F}$ *vs vs′ A*) *fm*
⟨*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 ⟹ *sorted ys* ⟹ *sorted* (*union-list-sorted xs ys*)
⟨*proof*⟩

lemma *distinct-union-list-sortedI* [*simp*]:
sorted xs ⟹ *sorted ys* ⟹ *distinct xs* ⟹ *distinct ys* ⟹ *distinct* (*union-list-sorted xs ys*)
⟨*proof*⟩

lemma *fit-rrns*:
assumes *infs*: ⋀*fvA*. *fvA* ∈ *set rrns* ⟹ *formula-spec* (*fset* $\mathcal{F}$) *Rs* (*fst* (*snd fvA*)) (*snd* (*snd fvA*)) (*fst fvA*)
assumes (*vs′*, *tas′*) = *fit-rrns* $\mathcal{F}$ *rrns*
shows *length tas′* = *length rrns* ⋀*i*. *i* < *length rrns* ⟹ *formula-spec* (*fset* $\mathcal{F}$) *Rs vs′* (*tas′* ! *i*) (*fst* (*rrns* ! *i*))
distinct vs′ sorted vs′
⟨*proof*⟩

13.7 Building blocks

definition *for-rrn* **where**
for-rrn tas = *fold* (λ*A B*. *relabel-reg* (*reg-union A B*)) *tas* (*Reg* {||} (*TA* {||} {||}))

lemma *for-rrn*:
assumes *length tas* = *length fs* ⋀*i*. *i* < *length fs* ⟹ *formula-spec* $\mathcal{F}$ *Rs vs* (*tas* ! *i*) (*fs* ! *i*)
and *vs*: *sorted vs distinct vs*
shows *formula-spec* $\mathcal{F}$ *Rs 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* $\mathcal{F}$) ≠ {} *length tas* = *length fs* ⋀*i*. *i* < *length fs* ⟹ *formula-spec* (*fset* $\mathcal{F}$) *Rs vs* (*tas* ! *i*) (*fs* ! *i*)
and *vs*: *sorted vs distinct vs*

shows *formula-spec* (*fset* $\mathcal{F}$) *Rs vs* (*fand-rrn* $\mathcal{F}$ (*length vs*) *tas*) (*FAnd fs*)
⟨*proof*⟩

13.7.1 IExists inference rule

lemma *lift-fun-gpairD*:
map-gterm lift-fun s = *gpair t u* $\Longrightarrow$ *t* = *s*
map-gterm lift-fun s = *gpair t u* $\Longrightarrow$ *u* = *s*
⟨*proof*⟩

definition *upd-bruijn* :: *nat list* $\Rightarrow$ *nat list* **where**
upd-bruijn vs = *tl* (*map* (λ *x*. *x* − *1*) *vs*)

lemma *upd-bruijn-length*[*simp*]:
length (*upd-bruijn vs*) = *length vs* − *1*
⟨*proof*⟩

lemma *pres-sorted-dec*:
sorted xs $\Longrightarrow$ *sorted* (*map* (λ*x*. *x* − *Suc 0*) *xs*)
⟨*proof*⟩

lemma *upd-bruijn-pres-sorted*:
sorted xs $\Longrightarrow$ *sorted* (*upd-bruijn xs*)
⟨*proof*⟩

lemma *pres-distinct-not-0-list-dec*:
distinct xs $\Longrightarrow$ *0* $\notin$ *set xs* $\Longrightarrow$ *distinct* (*map* (λ*x*. *x* − *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* $\in$ *set vs*
and $\bigwedge$ *x*. *x* $\in$ *set* (*upd-bruijn vs*) $\Longrightarrow$ α *x* = α' *x*
shows $\bigwedge$ *x*. *x* $\in$ *set vs* $\Longrightarrow$ (*shift* α *0 z*) *x* = (*shift* α' *0 z*) *x*
⟨*proof*⟩

lemma *ExistsI-upd-brujin-0*:
assumes *sorted vs distinct vs 0* $\in$ *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* $\notin$ *set vs* **and** *formula-relevant* $\mathcal{F}$ *Rs vs fm*
shows *formula-relevant* $\mathcal{F}$ *Rs* (*map* (λ*x*. *x* − *Suc 0*) *vs*) (*FExists fm*)

⟨*proof*⟩

definition *shift-right* **where**
shift-right $\alpha \equiv \lambda\ i.\ \alpha\ (i + 1)$

lemma *shift-right-nt-0*:
$i \neq 0 \Longrightarrow \alpha\ i = \textit{shift-right}\ \alpha\ (i - \textit{Suc}\ 0)$
⟨*proof*⟩

lemma *shift-shift-right-id* [*simp*]:
$\textit{shift}\ (\textit{shift-right}\ \alpha)\ 0\ (\alpha\ 0) = \alpha$
⟨*proof*⟩

lemma *shift-right-rangeI* [*intro*]:
$\textit{range}\ \alpha \subseteq T \Longrightarrow \textit{range}\ (\textit{shift-right}\ \alpha) \subseteq T$
⟨*proof*⟩

lemma *eval-formula-shift-right-eval*:
$\textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ \alpha\ \textit{fm} \Longrightarrow \textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ (\textit{shift}\ (\textit{shift-right}\ \alpha)\ 0\ (\alpha\ 0))\ \textit{fm}$
$\textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ (\textit{shift}\ (\textit{shift-right}\ \alpha)\ 0\ (\alpha\ 0))\ \textit{fm} \Longrightarrow \textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ \alpha\ \textit{fm}$
⟨*proof*⟩
declare *subsetI*[*intro*!]

lemma *nt-rel-0-trivial-shift*:
assumes $0 \notin \textit{set}\ \textit{vs}$
shows $\{\textit{map}\ \alpha\ \textit{vs}\ |\alpha.\ \textit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge \textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ \alpha\ \textit{fm}\} =$
$\{\textit{map}\ (\lambda x.\ \alpha\ (x - \textit{Suc}\ 0))\ \textit{vs}\ |\alpha.\ \textit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z \in \mathcal{T}_G\ \mathcal{F}.$
$\textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ (\alpha\langle 0{:}z\rangle)\ \textit{fm})\}$
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *relevant-vars-upd-bruijn-tl*:
assumes *sorted vs distinct vs*
shows $\textit{map}\ (\textit{shift-right}\ \alpha)\ (\textit{upd-bruijn}\ \textit{vs}) = \textit{tl}\ (\textit{map}\ \alpha\ \textit{vs})$ ⟨*proof*⟩

lemma *drop-upd-bruijn-set*:
assumes *sorted vs distinct vs*
shows $\textit{drop}\ 1\ \text{`}\ \{\textit{map}\ \alpha\ \textit{vs}\ |\alpha.\ \textit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge \textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ \alpha\ \textit{fm}\} =$
$\{\textit{map}\ \alpha\ (\textit{upd-bruijn}\ \textit{vs})\ |\alpha.\ \textit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists z{\in}\mathcal{T}_G\ \mathcal{F}.\ \textit{eval-formula}\ \mathcal{F}$
$\textit{Rs}\ (\alpha\langle 0{:}z\rangle)\ \textit{fm})\}$
(**is** *?Ls* = *?Rs*)
⟨*proof*⟩

lemma *closed-sat-form-env-dom*:
assumes $\textit{formula-relevant}\ \mathcal{F}\ \textit{Rs}\ []\ (\textit{FExists}\ \textit{fm})\ \textit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F}\ \textit{eval-formula}$
$\mathcal{F}\ \textit{Rs}\ \alpha\ \textit{fm}$
shows $\{[\alpha\ 0]\ |\alpha.\ \textit{range}\ \alpha \subseteq \mathcal{T}_G\ \mathcal{F} \wedge (\exists\ z \in \mathcal{T}_G\ \mathcal{F}.\ \textit{eval-formula}\ \mathcal{F}\ \textit{Rs}\ (\alpha\langle 0{:}z\rangle)$
$\textit{fm})\} = \{[t]\ |\ t.\ t \in \mathcal{T}_G\ \mathcal{F}\}$

⟨*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 $\mathcal{F}$ Rs infs (l, step, fm, is) = do {
guard (l = length infs);
case step of
IRR1 s x ⇒ do {
guard (fm = FRR1 s x);
liftO1 (λta. (FRR1 s x, [x], fmap-funs-reg (λf. [Some f]) ta)) (rr1c $\mathcal{F}$ Rs s)
}
| IRR2 r x y ⇒ do {
guard (fm = FRR2 r x y);
case compare x y of
Lt ⇒ liftO1 (λta. (FRR2 r x y, [x, y], fmap-funs-reg (λ(f, g). [f, g]) ta)) (rr2c $\mathcal{F}$ 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 $\mathcal{F}$ Rs r) (rr2c $\mathcal{F}$ Rs (R2Diag R1Terms))))
| Gt ⇒ liftO1 (λta. (FRR2 r x y, [y, x], fmap-funs-reg (λ(f, g). [g, f]) ta)) (rr2c $\mathcal{F}$ 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 $\mathcal{F}$ (map ((!) infs) ls) in
Some (fm, vs', fand-rrn $\mathcal{F}$ (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 $\mathcal{F}$ (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*: ⋀*fvA*. *fvA* ∈ *set infs* ⟹ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd fvA*)) (*snd* (*snd fvA*)) (*fst fvA*)
assumes *inf*: *check-inference rr1c rr2c* $\mathcal{F}$ *Rs infs* (*l*, *step*, *fm*, *is*) = *Some* (*fm*′, *vs*, *A*′)
assumes *rr1*: ⋀*r1*. ∀ *ta1*. *rr1c* $\mathcal{F}$ *Rs r1* = *Some ta1* ⟶ *RR1-spec ta1* (*eval-rr1-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r1*)
assumes *rr2*: ⋀*r2*. ∀ *ta2*. *rr2c* $\mathcal{F}$ *Rs r2* = *Some ta2* ⟶ *RR2-spec ta2* (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*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* $\mathcal{A}$ = *eps-free-automata* (*eps* $\mathcal{A}$) $\mathcal{A}$
abbreviation *ftrancl-eps-free-reg* $\mathcal{A}$ ≡ *Reg* (*fin* $\mathcal{A}$) (*ftrancl-eps-free-closures* (*ta* $\mathcal{A}$))

lemma *ftrancl-eps-free-ta-derI*:
(*eps* $\mathcal{A}$)$|^+|$ = *eps* $\mathcal{A}$ ⟹ *ta-der* (*ftrancl-eps-free-closures* $\mathcal{A}$) (*term-of-gterm t*) = *ta-der* $\mathcal{A}$ (*term-of-gterm t*)
⟨*proof*⟩

lemma $\mathcal{L}$*-ftrancl-eps-free-closuresI*:
(*eps* (*ta* $\mathcal{A}$))$|^+|$ = *eps* (*ta* $\mathcal{A}$) ⟹ $\mathcal{L}$ (*ftrancl-eps-free-reg* $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$
⟨*proof*⟩

definition *root-step R* $\mathcal{F}$ ≡ (*let* (*TA1*, *TA2*) = *agtt-grrstep R* $\mathcal{F}$ *in*
(*ftrancl-eps-free-closures TA1*, *TA2*))

definition *AGTT-trancl-eps-free* :: ('*q*, '*f*) *gtt* ⇒ ('*q* + '*q*, '*f*) *gtt* **where**
AGTT-trancl-eps-free $\mathcal{G}$ = (*let* ($\mathcal{A}$, $\mathcal{B}$) = *AGTT-trancl* $\mathcal{G}$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$, $\mathcal{B}$))

definition *GTT-trancl-eps-free* **where**
GTT-trancl-eps-free $\mathcal{G}$ = (*let* ($\mathcal{A}$, $\mathcal{B}$) = *GTT-trancl* $\mathcal{G}$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$,
ftrancl-eps-free-closures $\mathcal{B}$))

definition *AGTT-comp-eps-free* **where**
AGTT-comp-eps-free $\mathcal{G}_1$ $\mathcal{G}_2$ = (*let* ($\mathcal{A}$, $\mathcal{B}$) = *AGTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$, $\mathcal{B}$))

definition *GTT-comp-eps-free* **where**
GTT-comp-eps-free $\mathcal{G}_1$ $\mathcal{G}_2$ =(*let* ($\mathcal{A}$, $\mathcal{B}$) = *GTT-comp'* $\mathcal{G}_1$ $\mathcal{G}_2$ *in*
(*ftrancl-eps-free-closures* $\mathcal{A}$, *ftrancl-eps-free-closures* $\mathcal{B}$))

lemma *eps-free-relable* [*simp*]:
is-gtt-eps-free (*relabel-gtt* $\mathcal{G}$) = *is-gtt-eps-free* $\mathcal{G}$
⟨*proof*⟩

lemma *eps-free-prod-swap*:
is-gtt-eps-free ($\mathcal{A}$, $\mathcal{B}$) ⟹ *is-gtt-eps-free* ($\mathcal{B}$, $\mathcal{A}$)

⟨*proof*⟩

lemma *eps-free-root-step*:
is-gtt-eps-free (*root-step R* $\mathcal{F}$)
⟨*proof*⟩

lemma *eps-free-AGTT-trancl-eps-free*:
is-gtt-eps-free $\mathcal{G}$ $\Longrightarrow$ *is-gtt-eps-free* (*AGTT-trancl-eps-free* $\mathcal{G}$)
⟨*proof*⟩

lemma *eps-free-GTT-trancl-eps-free*:
is-gtt-eps-free $\mathcal{G}$ $\Longrightarrow$ *is-gtt-eps-free* (*GTT-trancl-eps-free* $\mathcal{G}$)
⟨*proof*⟩

lemma *eps-free-AGTT-comp-eps-free*:
is-gtt-eps-free $\mathcal{G}_2$ $\Longrightarrow$ *is-gtt-eps-free* (*AGTT-comp-eps-free* $\mathcal{G}_1$ $\mathcal{G}_2$)
⟨*proof*⟩

lemma *eps-free-GTT-comp-eps-free*:
is-gtt-eps-free (*GTT-comp-eps-free* $\mathcal{G}_1$ $\mathcal{G}_2$)
⟨*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-der* (*fst* $\mathcal{A}$) (*term-of-gterm t*) = *ta-der* (*fst* $\mathcal{B}$) (*term-of-gterm t*)
and $\bigwedge$ *t. ta-der* (*snd* $\mathcal{A}$) (*term-of-gterm t*) = *ta-der* (*snd* $\mathcal{B}$) (*term-of-gterm t*)
shows *agtt-lang* $\mathcal{A}$ = *agtt-lang* $\mathcal{B}$ ⟨*proof*⟩

lemma *agtt-lang-root-step-conv*:
agtt-lang (*root-step R* $\mathcal{F}$) = *agtt-lang* (*agtt-grrstep R* $\mathcal{F}$)
⟨*proof*⟩

lemma *agtt-lang-AGTT-trancl-eps-free-conv*:
assumes *is-gtt-eps-free* $\mathcal{G}$
shows *agtt-lang* (*AGTT-trancl-eps-free* $\mathcal{G}$) = *agtt-lang* (*AGTT-trancl* $\mathcal{G}$)
⟨*proof*⟩

lemma *agtt-lang-GTT-trancl-eps-free-conv*:
assumes *is-gtt-eps-free* $\mathcal{G}$
shows *agtt-lang* (*GTT-trancl-eps-free* $\mathcal{G}$) = *agtt-lang* (*GTT-trancl* $\mathcal{G}$)

⟨*proof*⟩

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* (*ARoot is*) = *liftO1* (λ*R*. *relabel-gtt* (*root-step R* $\mathcal{F}$)) (*is-to-trs′ Rs is*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*GInv g*) = *liftO1 prod.swap* (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*AUnion g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-union′ g1 g2*)) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*ATrancl g*) = *liftO1* (*relabel-gtt* ∘ *AGTT-trancl-eps-free*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*GTrancl g*) = *liftO1 GTT-trancl-eps-free* (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*AComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*AGTT-comp-eps-free g1 g2*)) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)
| *gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs* (*GComp g1 g2*) = *liftO2* (λ*g1 g2*. *relabel-gtt* (*GTT-comp-eps-free g1 g2*)) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g1*) (*gtt-of-gtt-rel-impl* $\mathcal{F}$ *Rs g2*)

lemma *gtt-of-gtt-rel-impl-is-gtt-eps-free*:
gtt-of-gtt-rel-impl $\mathcal{F}$ *Rs g* = *Some g′* ⟹ *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 $\mathcal{L}$*-eps-free-nhole-ctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*ftrancl-eps-free-reg* (*nhole-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = $\mathcal{L}$ (*nhole-ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)
⟨*proof*⟩

lemma $\mathcal{L}$*-eps-free-ctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*ftrancl-eps-free-reg* (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = $\mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$)
⟨*proof*⟩

lemma $\mathcal{L}$*-eps-free-parallel-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*ftrancl-eps-free-reg* (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$)) = $\mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}$ $\mathcal{A}$)
⟨*proof*⟩

abbreviation *eps-free-reg′ S R* ≡ *Reg* (*fin R*) (*eps-free-automata S* (*ta R*))

definition *eps-free-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ *in*
eps-free-reg′ ((λ *p*. (*fst p*, *Inr cl-state*)) |'| (*eps* (*ta* $\mathcal{B}$)) |∪| *eps* (*ta* $\mathcal{B}$)) $\mathcal{B}$)

definition *eps-free-nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$ *in*
eps-free-reg′ ((λ *p*. (*fst p*, *Inl* (*Inr cl-state*))) |'| (*eps* (*ta* $\mathcal{B}$)) |∪| *eps* (*ta* $\mathcal{B}$)) $\mathcal{B}$)

definition *eps-free-nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ =
(*let* $\mathcal{B}$ = *nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$ *in*
eps-free-reg′ ((λ *p*. (*fst p*, (*Inr cl-state*))) |'| (*eps* (*ta* $\mathcal{B}$)) |∪| *eps* (*ta* $\mathcal{B}$)) $\mathcal{B}$)

lemma $\mathcal{L}$*-eps-free-reg′I*:
(*eps* (*ta* $\mathcal{A}$))|$^+$| = *S* ⟹ $\mathcal{L}$ (*eps-free-reg′ S* $\mathcal{A}$) = $\mathcal{L}$ $\mathcal{A}$
⟨*proof*⟩

lemma $\mathcal{L}$*-eps-free-mctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*eps-free-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) ⟨*proof*⟩

lemma $\mathcal{L}$*-eps-free-nhole-mctxt-reflcl-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*eps-free-nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{F}$ $\mathcal{A}$)
⟨*proof*⟩

lemma $\mathcal{L}$*-eps-free-nhole-mctxt-closure-reg*:
assumes *is-ta-eps-free* (*ta* $\mathcal{A}$)
shows $\mathcal{L}$ (*eps-free-nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) = $\mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}$ $\mathcal{A}$) ⟨*proof*⟩

fun *rr1-of-rr1-rel-impl* :: (*′f* × *nat*) *fset* ⇒ (*′f* :: *linorder*, *′v*) *fin-trs list* ⇒ *ftrs rr1-rel* ⇒ (*nat*, *′f*) *reg option*
and *rr2-of-rr2-rel-impl* :: (*′f* × *nat*) *fset* ⇒ (*′f*, *′v*) *fin-trs list* ⇒ *ftrs rr2-rel* ⇒ (*nat*, *′f option* × *′f option*) *reg option* **where**
rr1-of-rr1-rel-impl $\mathcal{F}$ *Rs R1Terms* = *Some* (*relabel-reg* (*term-reg* $\mathcal{F}$))
| *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs* (*R1NF is*) = *liftO1* (λ*R*. (*simplify-reg* (*nf-reg* (*fst* |'| *R*)

𝓕))) (is-to-trs′ Rs is)
| rr1-of-rr1-rel-impl 𝓕 Rs (R1Inf r) = liftO1 (λR.
let 𝒜 = trim-reg R in
simplify-reg (proj-1-reg (Inf-reg-impl 𝒜))
) (rr2-of-rr2-rel-impl 𝓕 Rs r)
| rr1-of-rr1-rel-impl 𝓕 Rs (R1Proj i r) = (case i of 0 ⇒
liftO1 (trim-reg ∘ proj-1-reg) (rr2-of-rr2-rel-impl 𝓕 Rs r)
| - ⇒ liftO1 (trim-reg ∘ proj-2-reg) (rr2-of-rr2-rel-impl 𝓕 Rs r))
| rr1-of-rr1-rel-impl 𝓕 Rs (R1Union s1 s2) =
liftO2 (λ x y. relabel-reg (reg-union x y)) (rr1-of-rr1-rel-impl 𝓕 Rs s1) (rr1-of-rr1-rel-impl 𝓕 Rs s2)
| rr1-of-rr1-rel-impl 𝓕 Rs (R1Inter s1 s2) =
liftO2 (λ x y. simplify-reg (reg-intersect x y)) (rr1-of-rr1-rel-impl 𝓕 Rs s1) (rr1-of-rr1-rel-impl 𝓕 Rs s2)
| rr1-of-rr1-rel-impl 𝓕 Rs (R1Diff s1 s2) = liftO2 (λ x y. relabel-reg (trim-reg (difference-reg x y))) (rr1-of-rr1-rel-impl 𝓕 Rs s1) (rr1-of-rr1-rel-impl 𝓕 Rs s2)

| rr2-of-rr2-rel-impl 𝓕 Rs (R2GTT-Rel g w x) =
(case w of PRoot ⇒
(case x of ESingle ⇒ liftO1 (simplify-reg ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)
| EParallel ⇒ liftO1 (simplify-reg ∘ reflcl-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)
| EStrictParallel ⇒ liftO1 (simplify-reg ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g))
| PNonRoot ⇒
(case x of ESingle ⇒ liftO1 (simplify-reg ∘ ftrancl-eps-free-reg ∘ nhole-ctxt-closure-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)
| EParallel ⇒ liftO1 (simplify-reg ∘ eps-free-nhole-mctxt-reflcl-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)
| EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-nhole-mctxt-closure-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g))
| PAny ⇒
(case x of ESingle ⇒ liftO1 (simplify-reg ∘ ftrancl-eps-free-reg ∘ ctxt-closure-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)
| EParallel ⇒ liftO1 (simplify-reg ∘ ftrancl-eps-free-reg ∘ parallel-closure-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)
| EStrictParallel ⇒ liftO1 (simplify-reg ∘ eps-free-mctxt-closure-reg (lift-sig-RR2 |‘| 𝓕) ∘ GTT-to-RR2-root-reg) (gtt-of-gtt-rel-impl 𝓕 Rs g)))
| rr2-of-rr2-rel-impl 𝓕 Rs (R2Diag s) =
liftO1 (λ x. fmap-funs-reg (λf. (Some f, Some f)) x) (rr1-of-rr1-rel-impl 𝓕 Rs s)
| rr2-of-rr2-rel-impl 𝓕 Rs (R2Prod s1 s2) =
liftO2 (λ x y. simplify-reg (pair-automaton-reg x y)) (rr1-of-rr1-rel-impl 𝓕 Rs s1) (rr1-of-rr1-rel-impl 𝓕 Rs s2)
| rr2-of-rr2-rel-impl 𝓕 Rs (R2Inv r) = liftO1 (fmap-funs-reg prod.swap) (rr2-of-rr2-rel-impl 𝓕 Rs r)
| rr2-of-rr2-rel-impl 𝓕 Rs (R2Union r1 r2) =
liftO2 (λ x y. relabel-reg (reg-union x y)) (rr2-of-rr2-rel-impl 𝓕 Rs r1) (rr2-of-rr2-rel-impl

$\mathcal{F}$ *Rs r2*)
| *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs* (*R2Inter r1 r2*) =
liftO2 (λ *x y. simplify-reg* (*reg-intersect x y*)) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2*)
| *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs* (*R2Diff r1 r2*) = *liftO2* (λ *x y. simplify-reg* (*difference-reg x y*)) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2*)
| *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs* (*R2Comp r1 r2*) = *liftO2* (λ *x y. simplify-reg* (*rr2-compositon* $\mathcal{F}$ *x y*))
(*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r1*) (*rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2*)

lemmas *ta-simp-unfold* = *simplify-reg-def relabel-reg-def trim-reg-def relabel-ta-def term-reg-def*
lemma *is-ta-eps-free-trim-reg* [*intro*!]:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*trim-reg R*))
⟨*proof*⟩

lemma *is-ta-eps-free-relabel-reg* [*intro*!]:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*relabel-reg R*))
⟨*proof*⟩

lemma *is-ta-eps-free-simplify-reg* [*intro*!]:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*simplify-reg R*))
⟨*proof*⟩

lemma *is-ta-emptyI* [*simp*]:
is-ta-eps-free (*TA R* {||}) $\longleftrightarrow$ *True*
⟨*proof*⟩

lemma *is-ta-empty-trim-reg*:
is-ta-eps-free (*ta A*) $\Longrightarrow$ *eps* (*ta* (*trim-reg A*)) = {||}
⟨*proof*⟩

lemma *is-proj-ta-eps-empty*:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*proj-1-reg R*))
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*proj-2-reg R*))
⟨*proof*⟩

lemma *is-pod-ta-eps-empty*:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta L*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*reg-intersect R L*))
⟨*proof*⟩

lemma *is-fmap-funs-reg-eps-empty*:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*fmap-funs-reg f R*))
⟨*proof*⟩

lemma *is-collapse-automaton-reg-eps-empty*:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*collapse-automaton-reg R*))
⟨*proof*⟩

lemma *is-pair-automaton-reg-eps-empty*:
is-ta-eps-free (*ta R*) $\Longrightarrow$ *is-ta-eps-free* (*ta L*) $\Longrightarrow$ *is-ta-eps-free* (*ta* (*pair-automaton-reg R L*))
⟨*proof*⟩

lemma *is-reflcl-automaton-eps-free*:
is-ta-eps-free A $\Longrightarrow$ *is-ta-eps-free* (*reflcl-automaton* (*lift-sig-RR2* |'| $\mathcal{F}$) *A*)
⟨*proof*⟩

lemma *is-GTT-to-RR2-root-eps-empty*:
is-gtt-eps-free $\mathcal{G}$ $\Longrightarrow$ *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*]:
$\mathcal{Q}_r$ (*fmap-funs-reg f* $\mathcal{A}$) = $\mathcal{Q}_r$ $\mathcal{A}$
⟨*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*:
$\mathcal{Q}_r$ $\mathcal{A}$ |⊆| *ta-reachable* (*ta* $\mathcal{A}$) $\Longrightarrow$ $\mathcal{Q}_r$ $\mathcal{B}$ |⊆| *ta-reachable* (*ta* $\mathcal{B}$) $\Longrightarrow$ $\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ $\Longrightarrow$ $\mathcal{L}$ (*collapse-automaton-reg* $\mathcal{A}$) = $\mathcal{L}$ (*collapse-automaton-reg* $\mathcal{B}$)
⟨*proof*⟩

lemma *L-fmap-funs-reg-cong*:
$\mathcal{L}$ $\mathcal{A}$ = $\mathcal{L}$ $\mathcal{B}$ $\Longrightarrow$ $\mathcal{L}$ (*fmap-funs-reg h* $\mathcal{A}$) = $\mathcal{L}$ (*fmap-funs-reg h* $\mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-pair-automaton-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{L}\ \mathcal{C} = \mathcal{L}\ \mathcal{D} \Longrightarrow \mathcal{L}$ (*pair-automaton-reg* $\mathcal{A}\ \mathcal{C}$) $= \mathcal{L}$ (*pair-automaton-reg* $\mathcal{B}\ \mathcal{D}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-nhole-ctxt-closure-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{F} = \mathcal{G} \Longrightarrow \mathcal{L}$ (*nhole-ctxt-closure-reg* $\mathcal{F}\ \mathcal{A}$) $= \mathcal{L}$ (*nhole-ctxt-closure-reg* $\mathcal{G}\ \mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-nhole-mctxt-closure-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{F} = \mathcal{G} \Longrightarrow \mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{F}\ \mathcal{A}$) $= \mathcal{L}$ (*nhole-mctxt-closure-reg* $\mathcal{G}\ \mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-ctxt-closure-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{F} = \mathcal{G} \Longrightarrow \mathcal{L}$ (*ctxt-closure-reg* $\mathcal{F}\ \mathcal{A}$) $= \mathcal{L}$ (*ctxt-closure-reg* $\mathcal{G}\ \mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-parallel-closure-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{F} = \mathcal{G} \Longrightarrow \mathcal{L}$ (*parallel-closure-reg* $\mathcal{F}\ \mathcal{A}$) $= \mathcal{L}$ (*parallel-closure-reg* $\mathcal{G}\ \mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-mctxt-closure-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{F} = \mathcal{G} \Longrightarrow \mathcal{L}$ (*mctxt-closure-reg* $\mathcal{F}\ \mathcal{A}$) $= \mathcal{L}$ (*mctxt-closure-reg* $\mathcal{G}$ $\mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-nhole-mctxt-reflcl-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{F} = \mathcal{G} \Longrightarrow \mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{F}\ \mathcal{A}$) $= \mathcal{L}$ (*nhole-mctxt-reflcl-reg* $\mathcal{G}\ \mathcal{B}$)
⟨*proof*⟩

declare *equalityI*[*rule del*]
declare *fsubsetI*[*rule del*]
lemma *$\mathcal{L}$-proj-1-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{L}$ (*proj-1-reg* $\mathcal{A}$) $= \mathcal{L}$ (*proj-1-reg* $\mathcal{B}$)
⟨*proof*⟩

lemma *$\mathcal{L}$-proj-2-reg-cong*:
$\mathcal{L}\ \mathcal{A} = \mathcal{L}\ \mathcal{B} \Longrightarrow \mathcal{L}$ (*proj-2-reg* $\mathcal{A}$) $= \mathcal{L}$ (*proj-2-reg* $\mathcal{B}$)
⟨*proof*⟩

lemma *rr2-of-rr2-rel-impl-sound*:
assumes $\forall R \in$ *set Rs. lv-trs* (*fset R*) $\wedge$ *ffunas-trs R* $|\subseteq|$ $\mathcal{F}$
shows $\bigwedge$ *A B. rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs r1* = *Some A* $\Longrightarrow$ *rr1-of-rr1-rel* $\mathcal{F}$ *Rs r1* = *Some B* $\Longrightarrow \mathcal{L}\ A = \mathcal{L}\ B$
$\bigwedge$ *A B. rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2* = *Some A* $\Longrightarrow$ *rr2-of-rr2-rel* $\mathcal{F}$ *Rs r2* =

Some B $\Longrightarrow$ $\mathcal{L}$ *A* = $\mathcal{L}$ *B*
⟨*proof*⟩
declare *equalityI*[*intro*!]
declare *fsubsetI*[*intro*!]

lemma *rr12-of-rr12-rel-impl-correct*:
assumes ∀ *R* ∈ *set Rs*. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| $\mathcal{F}$
shows ∀ *ta1*. *rr1-of-rr1-rel-impl* $\mathcal{F}$ *Rs r1* = *Some ta1* ⟶ *RR1-spec ta1* (*eval-rr1-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r1*)
∀ *ta2*. *rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs r2* = *Some ta2* ⟶ *RR2-spec ta2* (*eval-rr2-rel* (*fset* $\mathcal{F}$) (*map fset Rs*) *r2*)
⟨*proof*⟩

lemma *check-inference-rrn-impl-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*: ⋀*fvA*. *fvA* ∈ *set infs* $\Longrightarrow$ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd fvA*)) (*snd* (*snd fvA*)) (*fst fvA*)
assumes *inf*: *check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs infs* (*l*, *step*, *fm*, *is*) = *Some* (*fm'*, *vs*, *A'*)
shows *l* = *length infs* ∧ *fm* = *fm'* ∧ *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) *vs A'* *fm'*
⟨*proof*⟩

definition *check-sig-nempty* **where**
check-sig-nempty $\mathcal{F}$ = (*0* |∈| *snd* |'| $\mathcal{F}$)

definition *check-trss* **where**
check-trss $\mathcal{R}$ $\mathcal{F}$ = *list-all* (λ *R*. *lv-trs* (*fset R*) ∧ *funas-trs* (*fset R*) ⊆ *fset* $\mathcal{F}$) $\mathcal{R}$

lemma *check-sig-nempty*:
check-sig-nempty $\mathcal{F}$ ⟷ $\mathcal{T}_G$ (*fset* $\mathcal{F}$) ≠ {} (**is** *?Ls* ⟷ *?Rs*)
⟨*proof*⟩

lemma *check-trss*:
check-trss $\mathcal{R}$ $\mathcal{F}$ ⟷ (∀ *R* ∈ *set* $\mathcal{R}$. *lv-trs* (*fset R*) ∧ *ffunas-trs R* |⊆| $\mathcal{F}$)
⟨*proof*⟩

fun *check-inference-list* :: (*'f* × *nat*) *fset* ⇒ (*'f* :: {*compare*,*linorder*}, *'v*) *fin-trs list*
⇒ (*nat* × *ftrs inference* × *ftrs formula* × *info list*) *list*
⇒ (*ftrs formula* × *nat list* × (*nat*, *'f option list*) *reg*) *list option* **where**
check-inference-list $\mathcal{F}$ *Rs infs* = *do* {
  guard (*check-sig-nempty* $\mathcal{F}$);
  guard (*check-trss Rs* $\mathcal{F}$);
  foldl (λ *tas inf*. *do* {
    tas' ← *tas*;
    r ← *check-inference rr1-of-rr1-rel-impl rr2-of-rr2-rel-impl* $\mathcal{F}$ *Rs tas' inf*;
    Some (*tas'* @ [*r*])

})
    (*Some* []) *infs*
  }

lemma *check-inference-list-correct*:
  assumes *check-inference-list* $\mathcal{F}$ *Rs infs* = *Some fvAs*
  shows *length infs* = *length fvAs* ∧ (∀ *i* < *length fvAs*. *fst* (*snd* (*snd* (*infs* ! *i*))) = *fst* (*fvAs* ! *i*)) ∧
    (∀ *i* < *length fvAs*. *formula-spec* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd* (*fvAs* ! *i*))) (*snd* (*snd* (*fvAs* ! *i*))) (*fst* (*fvAs* ! *i*)))
  ⟨*proof*⟩

fun *check-certificate* **where**
  check-certificate $\mathcal{F}$ *Rs A fm* (*Certificate infs claim n*) = *do* {
    guard (*n* < *length infs*);
    guard (*A* ⟷ *claim* = *Nonempty*);
    guard (*fm* = *fst* (*snd* (*snd* (*infs* ! *n*))));
    fvA ← *check-inference-list* $\mathcal{F}$ *Rs* (*take* (*Suc n*) *infs*);
    (*let E* = *reg-empty* (*snd* (*snd* (*last fvA*))) *in*
     case claim of Empty ⇒ *Some E*
       | - ⇒ *Some* (¬ *E*))
  }

definition *formula-unsatisfiable* **where**
  formula-unsatisfiable $\mathcal{F}$ *Rs fm* ⟷ (*formula-satisfiable* $\mathcal{F}$ *Rs fm* = *False*)

definition *correct-certificate* **where**
  correct-certificate $\mathcal{F}$ *Rs claim infs n* ≡
     (*claim* = *Empty* ⟷ (*formula-unsatisfiable* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd* (*snd* (*infs* ! *n*))))) ∧
      claim = *Nonempty* ⟷ *formula-satisfiable* (*fset* $\mathcal{F}$) (*map fset Rs*) (*fst* (*snd* (*snd* (*infs* ! *n*)))))

lemma *check-certificate-sound*:
  assumes *check-certificate* $\mathcal{F}$ *Rs A fm* (*Certificate infs claim n*) = *Some B*
  shows *fm* = *fst* (*snd* (*snd* (*infs* ! *n*))) *A* ⟷ *claim* = *Nonempty*
  ⟨*proof*⟩

lemma *check-certificate-correct*:
  assumes *check-certificate* $\mathcal{F}$ *Rs A fm* (*Certificate infs claim n*) = *Some B*
  shows (*B* = *True* ⟶ *correct-certificate* $\mathcal{F}$ *Rs claim infs n*) ∧
    (*B* = *False* ⟶ *correct-certificate* $\mathcal{F}$ *Rs* (*case-claim Nonempty Empty claim*) *infs n*)
⟨*proof*⟩

definition *check-certificate-string* ::
  (*integer list* × *fvar*) *fset* ⇒
    ((*integer list*, *integer list*) *Term.term* × (*integer list*, *integer list*) *Term.term*)

fset list ⇒
  bool ⇒ *ftrs formula* ⇒ *ftrs certificate* ⇒ *bool option*
  where *check-certificate-string* = *check-certificate*

export-code *check-certificate-string Var Fun fset-of-list nat-of-integer Certificate*
  R2GTT-Rel R2Eq R2Reflc R2Step R2StepEq R2Steps R2StepsEq R2StepsNF
R2ParStep R2RootStep
  R2RootStepEq R2RootSteps R2RootStepsEq R2NonRootStep R2NonRootStepEq
R2NonRootSteps
  R2NonRootStepsEq R2Meet R2Join
  ARoot GSteps PRoot ESingle Empty Size EDistribAndOr
  R1Terms R1Fin
  FRR1 FRestrict FTrue FFalse
  IRR1 Fwd **in** *Haskell* **module-name** *FOR*

end

References

[1] S. Berghofer. First-order logic according to fitting. *Archive of Formal Proofs*, Aug. 2007. https://isa-afp.org/entries/FOL-Fitting.html, Formal proof development.

[2] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990.

[3] A. Lochmann, A. Middeldorp, F. Mitterwallner, and B. Felgenhauer. A verified decision procedure for the first-order theory of rewriting for linear variable-separated rewrite systems variable-separated rewrite systems in Isabelle/HOL. In C. Hriţcu 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.