

# Sorted Rewriting, Conditional Rewriting, and Logically Constrained Rewriting

Akihisa Yamada

National Institute of Advanced Industrial Science and Technology,  
Japan

March 22, 2026

## Abstract

This entry provides various materials for sorted term rewrite systems (sorted TRSs), (sorted) conditional TRSs (CTRSs), and logically constrained TRSs (LCTRSs). For (C)TRSs we formalize the fundamental result that the rewrite steps induced by a (C)TRS is the least rewrite relation that models the (C)TRS. For LCTRSs we simply formulate logics as sorted algebras with the bool sort and logical symbols which are interpreted as expected. This allows us to define rewrite steps of LCTRSs as rewrite steps of an (infinite) TRS.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>More About Binary Relations</b>	<b>3</b>
2.1	Missing Lemmas . . . . .	4
2.1.1	Extending Binary Relations AFP Entry . . . . .	6
2.1.2	Relation defined precisely on a set . . . . .	9
2.1.3	Closed Relations . . . . .	9
2.1.4	Function Space on Specific Input . . . . .	10
2.2	Lists . . . . .	11
2.2.1	Reflexive Closure Restricted on Set . . . . .	12
2.2.2	Dual . . . . .	15
<b>3</b>	<b>More About Sorted Terms</b>	<b>16</b>
<b>4</b>	<b>Sorted Relations</b>	<b>19</b>
4.1	Subject Reduction Property . . . . .	19

<b>5</b>	<b>Sorted Rules</b>	<b>22</b>
5.1	Equation . . . . .	22
5.2	Axioms – Rewrite Rules . . . . .	23
5.3	Inference Rules – Conditional Rewrite Rules . . . . .	23
5.4	Sortedness of Rules . . . . .	24
<b>6</b>	<b>Models</b>	<b>26</b>
6.1	Satisfaction of Equations . . . . .	26
6.2	Validity of Axioms . . . . .	27
6.2.1	Models . . . . .	28
<b>7</b>	<b>Ordered Algebras</b>	<b>29</b>
<b>8</b>	<b>Monotone Algebras</b>	<b>31</b>
8.1	Ordered Monotone Algebras . . . . .	35
<b>9</b>	<b>Sorted Rewrite Relations</b>	<b>36</b>
9.1	Closure Under Substitutions . . . . .	37
9.2	Rewrite Relations . . . . .	39
<b>10</b>	<b>Sorted Rewriting</b>	<b>41</b>
10.1	Root Rewrite Steps . . . . .	41
10.2	Rewrite Steps . . . . .	43
10.3	Closures . . . . .	46
10.4	Sorted Rewrite Systems . . . . .	49
10.5	Models of Rewrite Systems . . . . .	52
<b>11</b>	<b>Conditional Rewriting</b>	<b>55</b>
<b>12</b>	<b>Logic</b>	<b>62</b>
12.1	Syntax . . . . .	62
12.2	Semantics . . . . .	66
12.3	Propositional Logic . . . . .	71
<b>13</b>	<b>Logically Constrained Rewriting</b>	<b>72</b>
13.1	Sorted Injections . . . . .	75
<b>14</b>	<b>Extension of Algebras</b>	<b>77</b>
14.1	Disjoint Sum of Sorted Sets . . . . .	78
14.2	Extending Signature and Interpretations . . . . .	78
<b>15</b>	<b>Extending Algebra into Logic</b>	<b>80</b>

<b>16 Concrete Logics</b>	<b>83</b>
16.1 Bool Logic . . . . .	83
16.2 Natural Arithmetic . . . . .	83
16.3 Integer Arithmetic . . . . .	88
<b>17 Examples</b>	<b>90</b>
17.1 Less-Than TRS . . . . .	91
17.2 Addition TRS . . . . .	91
17.3 Even TRS . . . . .	92
17.4 Even CTRS . . . . .	92
17.5 LCTRS . . . . .	93

## 1 Introduction

This entry provides various materials needed for *logically constrained term rewrite systems* (LCTRS) [2], a term rewrite system (TRS) where rules come with constraints which are interpreted in a background logic.

Building upon the IsaFoR (Isabelle Formalization of Rewriting [4]) formalizations of unsorted terms [3] and sorted terms [6], we start with formalizing sorted TRSs. Here we formalize fundamental results such as that the rewrite relation induced by a sorted TRS is the least rewrite relation which models the TRS. We also formalize (sorted) conditional TRSs (CTRSs) and the fundamental result that the many-step reduction is the least rewrite preorder which models the CTRS.

As LCTRSs require *logics*, we provide a light-weight and rewriting-oriented formalization of logics. While it is standard to treat logical symbols and function symbols differently—cf. the IsaFoL (Isabelle Formalization of Logics) entry [1]—in this development we consider a logic just as a sorted algebra where the signature contains logic symbols and their interpretations are as expected. This became possible thanks to the formalization of sorted terms, and is crucial for LCTRSs since the formalism allows logical symbols to appear inside terms.

We also provide means to extend an algebra with another algebra, and to extend an algebra with logic. This allows us to build concrete logics modularly: we define algebras for natural numbers and integers, and obtain corresponding logics by extending these algebras with a pure Bool logic.

## 2 More About Binary Relations

**theory** *Binary-Relations-More*

**imports** *Complete-Non-Orders.Well-Relations Main HOL-Library.FuncSet  
Abstract-Rewriting.Abstract-Rewriting*

**begin**

## 2.1 Missing Lemmas

**lemma** *relpowp-Suc'*:  $r \overset{\sim}{\sim} \text{Suc } n = r \text{ OO } r \overset{\sim}{\sim} n$   
(proof)

**lemma** *tranclp-greater*:  $r \leq r^{++}$  (proof)

**lemma** *relation-ofI*:

**assumes**  $r \ x \ y$  **and**  $x \in X$  **and**  $y \in X$

**shows**  $(x,y) \in \text{relation-of } r \ X$

(proof)

**lemma** *relation-ofE*:

**assumes**  $(x,y) \in \text{relation-of } r \ X$   $r \ x \ y \implies x \in X \implies y \in X \implies \text{thesis}$

**shows** *thesis*

(proof)

**lemma** *in-relation-of-UNIV[simp]*:  $(x,y) \in \text{relation-of } r \ \text{UNIV} \longleftrightarrow r \ x \ y$

(proof)

**lemma** *relation-of-mono*:

**assumes**  $r \leq s$   $X \subseteq Y$  **shows**  $\text{relation-of } r \ X \subseteq \text{relation-of } s \ Y$

(proof)

**lemmas** *relation-of-subrel = relation-of-mono*[OF - subset-refl]

**lemmas** *relation-of-subset = relation-of-mono*[OF order.refl]

**lemma** *relation-of-sup*:  $\text{relation-of } (r \sqcup s) \ X = \text{relation-of } r \ X \cup \text{relation-of } s \ X$

(proof)

**lemma** *relation-of-UNIV-OO*:

$\text{relation-of } (r \text{ OO } s) \ \text{UNIV} = \text{relation-of } r \ \text{UNIV} \ O \ \text{relation-of } s \ \text{UNIV}$

(proof)

**lemma** *relation-of-UNIV-trancl*:

$\text{relation-of } (\text{tranclp } r) \ \text{UNIV} = \text{trancl } (\text{relation-of } r \ \text{UNIV})$

(proof)

**lemma** *relation-of-UNIV-rtrancl*:

$\text{relation-of } (\text{rtranclp } r) \ \text{UNIV} = \text{rtrancl } (\text{relation-of } r \ \text{UNIV})$

(proof)

**lemma** *rtrancl-relation-of*:  $(\text{relation-of } r \ \text{UNIV})^* = \text{relation-of } (\text{rtranclp } r) \ \text{UNIV}$

(proof)

**lemma** *in-rel-un*:  $\text{in-rel } (r \cup s) = \text{in-rel } r \sqcup \text{in-rel } s$

(proof)

**lemma** *in-rel-relcomp*:  $\text{in-rel } (r \ O \ s) = \text{in-rel } r \ \text{OO } \text{in-rel } s$

(proof)

**lemma** *in-rel-trancl*:  $in-rel (S^+) = (in-rel S)^{++}$

*<proof>*

**lemma** *in-rel-rtrancl*:  $in-rel (S^*) = (in-rel S)^{**}$

*<proof>*

**lemma** *tranclpD2*:  $tranclp r x y \implies \exists z. rtranclp r x z \wedge r z y$

*<proof>*

**lemma** *tranclp-mono*:

**assumes** *rs*:  $r \leq s$  **shows**  $tranclp r \leq tranclp s$

*<proof>*

**lemma** *mono-tranclp[mono]*:

$(\bigwedge x y. r x y \longrightarrow s x y) \implies tranclp r x y \longrightarrow tranclp s x y$

*<proof>*

**lemmas** *tranclp-subrel* = *tranclp-mono*[*THEN le-funD*, *THEN le-funD*, *THEN le-boolD*, *rule-format*]

**lemmas** *rtranclp-subrel* = *rtranclp-mono*[*THEN le-funD*, *THEN le-funD*, *THEN le-boolD*, *rule-format*]

**lemma** [*simp*]:

**shows**  $tranclp\text{-}tranclp: tranclp (tranclp r) = tranclp r$

*<proof>*

**lemma** *rtranclp-iff-tranclp*:  $rtranclp r s t \longleftrightarrow s = t \vee tranclp r s t$

*<proof>*

**lemma** *tranclp-imp-relpow*:  $r^{++} x y \implies \exists n > 0. (r \overset{\sim}{\sim} n) x y$  **for**  $x y$

*<proof>*

**lemma** *relpow-imp-tranclp*:

**assumes**  $xy: (r \overset{\sim}{\sim} n) x y$  **shows**  $n = 0 \wedge x = y \vee r^{++} x y$

*<proof>*

**lemma** *tranclp-is-Sup-relpow*:  $r^{++} = (\bigsqcup n \in \{0 < ..\}. r \overset{\sim}{\sim} n)$

*<proof>*

**lemma** *relpow-dual*: **fixes**  $r :: 'a \Rightarrow 'a \Rightarrow bool$  **shows**  $r^- \overset{\sim}{\sim} n = (r \overset{\sim}{\sim} n)^-$

*<proof>*

**lemma** *symclp-mono*:  $r \leq s \implies symclp r \leq symclp s$

*<proof>*

**lemma** *wf-iff-wfP*:  $wf R \longleftrightarrow wfP (in-rel R)$

*<proof>*

**lemma** *wfP-trancl*:  $wfP\ r^{++} \longleftrightarrow wfP\ r$   
*<proof>*

**lemma** *rtranclp-relcomp-rtranclp-le*:  
 $(rtranclp\ r\ OO\ rtranclp\ r) \leq rtranclp\ r$   
*<proof>*

**lemma** *wfP-iff-nonempty-minimal*:  $wfP\ r \longleftrightarrow (\forall X. X \neq \{\} \longrightarrow (\exists x \in X. \forall y \in X. \neg r\ y\ x))$   
*<proof>*

**lemmas** *wfP-imp-nonempty-minimal = wfP-iff-nonempty-minimal*[*THEN iffD1, rule-format*]

**lemmas** *nonempty-minimal-imp-wfP = wfP-iff-nonempty-minimal*[*THEN iffD2, rule-format*]

**lemma** *quasi-commute-imp-SN-Un-iff*:  $quasi-commute\ r\ s \implies SN\ (r \cup s) \longleftrightarrow SN\ r \wedge SN\ s$   
*<proof>*

**lemma** *symclp-symmetric*: *symmetric* *A* (*symclp* *r*) *<proof>*

**context** *reflexive* **begin**

**interpretation** *tranclp*: *quasi-ordered-set* *A* *tranclp* ( $\sqsubseteq$ )  
*<proof>*

**lemmas** *tranclp-quasi-order = tranclp.quasi-ordered-set-axioms*  
**and** *tranclp-reflexive = tranclp.reflexive-axioms*

**interpretation** *symclp*: *tolerance* *A* *symclp* ( $\sqsubseteq$ )  
*<proof>*

**lemmas** *symclp-tolerance = symclp.tolerance-axioms*  
**and** *symclp-reflexive = symclp.reflexive-axioms*

**end**

### 2.1.1 Extending Binary Relations AFP Entry

**lemma** *sup-Restrp*:  $r \sqcup s \upharpoonright A = (r \upharpoonright A) \sqcup (s \upharpoonright A)$   
*<proof>*

**lemma** *tranclp-Restrp-mem*[*simp*]:  
**assumes** *tranclp* (*s*  $\upharpoonright$  *A*) *x* *y*

**shows**  $x \in A \ y \in A$   
*<proof>*

**lemma** *Restrp-dual*:  $(r \upharpoonright A)^- = r^- \upharpoonright A$   
*<proof>*

**lemma** *Restrp-mono*: **assumes** *rs*:  $r \leq s$  **and** *AB*:  $A \subseteq B$  **shows**  $r \upharpoonright A \leq s \upharpoonright B$   
*<proof>*

**lemmas** *Restrp-subrel* = *Restrp-mono*[*OF - subset-refl*]  
**lemmas** *Restrp-subset* = *Restrp-mono*[*OF order.refl*]

**interpretation** *reflclp*: *reflexive UNIV reflclp r*  
**rewrites**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$   
**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
*<proof>*

**lemmas** *symclp-cases*[*consumes 1, case-names fw bw, elim!*] = *symclp-def*[*unfolded*  
*atomize-eq, THEN iffD1, THEN disjE*]

**interpretation** *symclp*: *symmetric UNIV symclp r*  
**rewrites**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$   
**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
*<proof>*

**interpretation** *tranclp*: *transitive UNIV tranclp r*  
**rewrites**  $(sympartp (tranclp r))^- \equiv sympartp (tranclp r)$   
**and**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$   
**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
*<proof>*

**interpretation** *rtranclp*: *quasi-ordered-set UNIV rtranclp r*  
**rewrites**  $(sympartp (rtranclp r))^- \equiv sympartp (rtranclp r)$   
**and**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$   
**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
*<proof>*

**interpretation** *transymclp*: *partial-equivalence UNIV tranclp (symclp r)*  
**rewrites**  $(sympartp (rtranclp r))^- \equiv sympartp (rtranclp r)$   
**and**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$

**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
 <proof>

**interpretation** *rtransymclp: equivalence UNIV rtranclp (symclp r)*  
**rewrites**  $(sympartp (rtranclp (symclp r)))^- \equiv sympartp (rtranclp (symclp r))$   
**and**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$   
**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
 <proof>

**lemma** (in *symmetric*) *reflclp-symmetric: symmetric A (reflclp (~))*  
 <proof>

**locale** *compatible* =  
*related-set + less-syntax +*  
**assumes** *strict-implies-weak*:  $x \sqsubset y \implies x \in A \implies y \in A \implies x \sqsubseteq y$   
**assumes** *weak-strict-trans[trans]*:  $x \sqsubseteq y \implies y \sqsubset z \implies x \in A \implies y \in A \implies z \in A \implies x \sqsubset z$   
**assumes** *strict-weak-trans[trans]*:  $x \sqsubset y \implies y \sqsubseteq z \implies x \in A \implies y \in A \implies z \in A \implies x \sqsubset z$   
**begin**

**sublocale** *strict: transitive A (⊆)*  
 <proof>

**end**

**interpretation** *rtranclp: compatible UNIV rtranclp r tranclp r*  
**rewrites**  $(sympartp (tranclp r))^- \equiv sympartp (tranclp r)$   
**and**  $\bigwedge x. x \in UNIV \equiv True$   
**and**  $\bigwedge X. X \subseteq UNIV \equiv True$   
**and**  $\bigwedge P1. (True \implies PROP P1) \equiv PROP P1$   
**and**  $\bigwedge P1. (True \implies P1) \equiv Trueprop P1$   
 <proof>

**context** *compatible-ordering begin*

**sublocale** *compatible*  
 <proof>

**end**

**lemma** *trans-relation-of: trans (relation-of r A)  $\longleftrightarrow$  transitive A r*  
 <proof>

### 2.1.2 Relation defined precisely on a set

**locale** *relation-on* = *related-set* +  
**assumes** *mem-left*:  $a \sqsubseteq b \implies a \in A$   
**and** *mem-right*:  $a \sqsubseteq b \implies b \in A$   
**begin**

**lemma** *Restrp-eq[simp]*:  $(\sqsubseteq) \upharpoonright A = (\sqsubseteq)$   
 $\langle proof \rangle$

**lemma** *dual-relation-on*: *relation-on*  $A$   $(\sqsubseteq)^-$   
 $\langle proof \rangle$

**lemma** *symmetric-imp-symclp-eq*:  
**assumes** *symmetric*  $A$   $(\sqsubseteq)$   
**shows** *symclp*  $(\sqsubseteq) = (\sqsubseteq)$   
 $\langle proof \rangle$

**lemma** *tranclp-relation-on*: *relation-on*  $A$   $(\sqsubseteq)^{++}$   
 $\langle proof \rangle$

**lemma** *symclp-relation-on*: *relation-on*  $A$   $(\text{symclp } (\sqsubseteq))$   
 $\langle proof \rangle$

**lemma** *symmetric-UNIV*: *symmetric*  $A$   $(\sqsubseteq) \longleftrightarrow \text{symmetric UNIV } (\sqsubseteq)$   
 $\langle proof \rangle$

**end**

**interpretation** *Restrp*: *relation-on*  $A$   $r \upharpoonright A$   
 $\langle proof \rangle$

**interpretation** *Restrp.tranclp*: *relation-on*  $A$  *tranclp*  $(r \upharpoonright A)$   
 $\langle proof \rangle$

**lemma** *relation-on-sup*:  
*relation-on*  $A$   $(r \sqcup s) \longleftrightarrow \text{relation-on } A$   $r \wedge \text{relation-on } A$   $s$   
 $\langle proof \rangle$

### 2.1.3 Closed Relations

**locale** *relation-closed* = *related-set* +  
**assumes** *closed*:  $a \sqsubseteq b \implies a \in A \implies b \in A$   
**begin**

**interpretation** *tranclp*: *relation-closed*  $A$   $(\sqsubseteq)^{++}$   
 $\langle proof \rangle$

**lemmas** *tranclp-closed* = *tranclp.relation-closed-axioms*

**lemma** *reflclp-closed: relation-closed*  $A (\sqsubseteq)^{==}$   
*<proof>*

**lemma** *rtranclp-closed: relation-closed*  $A (\sqsubseteq)^{**}$   
*<proof>*

**lemma** *tranclp-partial-equivalence:*  
**assumes** *symmetric*  $A (\sqsubseteq)$  **shows** *partial-equivalence*  $A (\sqsubseteq)^{++}$   
*<proof>*

**lemma** *tranclp-equivalence:*  
**assumes** *tolerance*  $A (\sqsubseteq)$   
**shows** *equivalence*  $A (\text{tranclp } (\sqsubseteq))$   
*<proof>*

**end**

#### 2.1.4 Function Space on Specific Input

**definition** *fun-ord-on where fun-ord-on*  $I (\sqsubseteq) f g \equiv \forall i \in I. f i \sqsubseteq g i$   
**for**  $le$  (**infix**  $\sqsubseteq 50$ )

**lemmas** *fun-ord-onI = fun-ord-on-def*[*unfolded atomize-eq, THEN iffD2, rule-format*]  
**lemmas** *fun-ord-onD = fun-ord-on-def*[*unfolded atomize-eq, THEN iffD1, rule-format*]

**lemma** *fun-ord-on-empty*[*simp*]: *fun-ord-on*  $\{\} = \top$   
*<proof>*

**lemma** *fun-ord-on-UNIV*[*simp*]: *fun-ord-on*  $UNIV = \text{fun-ord}$   
*<proof>*

**lemma** *dual-fun-ord-on: (fun-ord-on I r)<sup>-</sup> = fun-ord-on I r<sup>-</sup>*  
*<proof>*

**lemma** (**in reflexive**) *fun-reflexive:*  
*reflexive*  $(I \rightarrow A)$  (*fun-ord-on*  $I (\sqsubseteq)$ )  
*<proof>*

**lemma** (**in irreflexive**) *fun-irreflexive:*  
**assumes**  $I0: I \neq \{\}$   
**shows** *irreflexive*  $(I \rightarrow A)$  (*fun-ord-on*  $I (\sqsubseteq)$ )  
*<proof>*

**lemma** (**in semiattractive**) *fun-semiattractive:*  
*semiattractive*  $(I \rightarrow A)$  (*fun-ord-on*  $I (\sqsubseteq)$ )  
*<proof>*

**lemma** (**in attractive**) *fun-attractive:*  
*attractive*  $(I \rightarrow A)$  (*fun-ord-on*  $I (\sqsubseteq)$ )

*<proof>*

**lemma** (*in transitive*) *fun-transitive*:  
*transitive* ( $I \rightarrow A$ ) (*fun-ord-on*  $I$  ( $\sqsubseteq$ ))  
*<proof>*

**lemma** (*in symmetric*) *fun-symmetric*:  
*symmetric* ( $I \rightarrow A$ ) (*fun-ord-on*  $I$  ( $\sim$ ))  
*<proof>*

**lemma** (*in quasi-ordered-set*) *fun-quasi-order*:  
*quasi-ordered-set* ( $I \rightarrow A$ ) (*fun-ord-on*  $I$  ( $\sqsubseteq$ ))  
*<proof>*

**lemma** (*in tolerance*) *fun-tolerance*:  
*tolerance* ( $I \rightarrow A$ ) (*fun-ord-on*  $I$  ( $\sim$ ))  
*<proof>*

**lemma** (*in equivalence*) *fun-equivalence*:  
*equivalence* ( $I \rightarrow A$ ) (*fun-ord-on*  $I$  ( $\sim$ ))  
*<proof>*

## 2.2 Lists

**lemma** *dual-list-all2*:  $(\text{list-all2 } r)^- = \text{list-all2 } r^-$  *<proof>*

**lemma** (*in reflexive*) *lists-reflexive*: *reflexive* (*lists*  $A$ ) (*list-all2* ( $\sqsubseteq$ ))  
*<proof>*

**lemma** (*in transitive*) *lists-transitive*: *transitive* (*lists*  $A$ ) (*list-all2* ( $\sqsubseteq$ )) (*is transitive*  $?A$   $?r$ )  
*<proof>*

**lemma** (*in symmetric*) *lists-symmetric*: *symmetric* (*lists*  $A$ ) (*list-all2* ( $\sim$ ))  
*<proof>*

**lemma** (*in antisymmetric*) *lists-antisymmetric*: *antisymmetric* (*lists*  $A$ ) (*list-all2* ( $\sqsubseteq$ ))  
*<proof>*

**lemma** (*in semiattractive*) *lists-semiattractive*: *semiattractive* (*lists*  $A$ ) (*list-all2* ( $\sqsubseteq$ ))  
*<proof>*

**lemma** (*in attractive*) *lists-attractive*: *attractive* (*lists*  $A$ ) (*list-all2* ( $\sqsubseteq$ ))  
*<proof>*

**lemma** (*in quasi-ordered-set*) *lists-quasi-ordered-set*: *quasi-ordered-set* (*lists*  $A$ ) (*list-all2* ( $\sqsubseteq$ ))

*<proof>*

**lemma** (in *near-ordered-set*) *lists-near-ordered-set: near-ordered-set (lists A) (list-all2 (⊆))*  
*<proof>*

**lemma** (in *pseudo-ordered-set*) *lists-pseudo-ordered-set: pseudo-ordered-set (lists A) (list-all2 (⊆))*  
*<proof>*

**lemma** (in *partially-ordered-set*) *lists-partially-ordered-set: partially-ordered-set (lists A) (list-all2 (⊆))*  
*<proof>*

**lemma** (in *tolerance*) *lists-tolerance: tolerance (lists A) (list-all2 (∼))*  
*<proof>*

**lemma** (in *partial-equivalence*) *lists-partial-equivalence: partial-equivalence (lists A) (list-all2 (∼))*  
*<proof>*

**lemma** (in *equivalence*) *lists-equivalence: equivalence (lists A) (list-all2 (∼))*  
*<proof>*

### 2.2.1 Reflexive Closure Restricted on Set

**definition** *id-on where id-on A x y ≡ x = y ∧ y ∈ A*

**lemmas** *id-onI[intro!] = id-on-def[unfolded atomize-eq, THEN iffD2, rule-format]*  
**lemmas** *id-onE[elim!] = id-on-def[unfolded atomize-eq, THEN iffD1, elim-format, unfolded conj-imp-eq-imp-imp]*

**lemma** *id-on-relcompp[simp]: (id-on A OO r) x y ⟷ x ∈ A ∧ r x y*  
**and** *relcompp-id-on[simp]: (r OO id-on A) x y ⟷ r x y ∧ y ∈ A*  
*<proof>*

**lemma** *id-on-relcompp-id-on-simps[simp]:*  
*id-on A OO id-on B = id-on (A ∩ B)*  
*r OO id-on A OO id-on B = r OO id-on (A ∩ B)*  
*id-on A OO id-on B OO s = id-on (A ∩ B) OO s*  
*r OO id-on A OO id-on B OO s = r OO id-on (A ∩ B) OO s*  
*<proof>*

**lemma** *Restrp-eq-relcomp: r|A = id-on A OO r OO id-on A* *<proof>*

**definition** *reflclp-on A r ≡ r ⊔ id-on A*

**lemma** *reflclp-onI: r x y ∨ x = y ∧ x ∈ A ⟹ reflclp-on A r x y*  
*<proof>*

**lemma** *reflclp-onE*[consumes 1, case-names *refl base*]:  
 $\text{reflclp-on } A \ r \ x \ y \implies (x = y \implies x \in A \implies \text{thesis}) \implies (r \ x \ y \implies \text{thesis}) \implies$   
*thesis*  
 ⟨proof⟩

**lemma** *reflclp-on-refl*[simp]:  $x \in A \implies \text{reflclp-on } A \ r \ x \ x$   
**and** *reflclp-on-base*[simp]:  $r \ x \ y \implies \text{reflclp-on } A \ r \ x \ y$   
 ⟨proof⟩

**lemma** *reflclp-on-mono*:  
 $r \leq s \implies \text{reflclp-on } A \ r \leq \text{reflclp-on } A \ s$   
 ⟨proof⟩

**lemmas** *reflclp-on-subrel = reflclp-on-mono*[THEN *le-funD*, THEN *le-funD*, THEN *le-boolD*, *rule-format*]

**lemma** *mono-reflclp-on*[mono]:  
 $(\bigwedge x \ y. r \ x \ y \longrightarrow s \ x \ y) \implies \text{reflclp-on } A \ r \ x \ y \longrightarrow \text{reflclp-on } A \ s \ x \ y$   
 ⟨proof⟩

**interpretation** *reflclp-on*: *reflexive*  $A$  ⟨*reflclp-on*  $A$   $r$ ⟩  
 ⟨proof⟩

**lemma** (in *reflexive*) *reflclp-on-id*[simp]:  $\text{reflclp-on } A \ (\sqsubseteq) = (\sqsubseteq)$   
 ⟨proof⟩

**lemma** *reflclp-on-UNIV*:  $\text{reflclp-on } UNIV = \text{reflclp}$   
 ⟨proof⟩

**lemma** *reflclp-on-Restr*:  $\text{reflclp-on } A \ r \ \upharpoonright \ B = \text{reflclp-on } (A \cap B) \ (r \ \upharpoonright \ B)$   
 ⟨proof⟩

**lemma** (in *transitive*) *reflclp-on-transitive*: *transitive*  $A$  ( $\text{reflclp-on } B \ (\sqsubseteq)$ )  
 ⟨proof⟩

**lemma** (in *transitive*) *reflclp-on-quasi-order*: *quasi-ordered-set*  $A$  ( $\text{reflclp-on } A \ (\sqsubseteq)$ )  
 ⟨proof⟩

**lemma** (in *symmetric*) *reflclp-on-symmetric*: *symmetric*  $A$  ( $\text{reflclp-on } B \ (\sim)$ )  
 ⟨proof⟩

**lemma** (in *symmetric*) *reflclp-on-tolerance*: *tolerance*  $A$  ( $\text{reflclp-on } A \ (\sim)$ )  
 ⟨proof⟩

**lemma** (in *partial-equivalence*) *reflclp-equivalence*: *equivalence*  $A$  ( $\text{reflclp } (\sim)$ )  
 ⟨proof⟩

**lemma** (in *partial-equivalence*) *reflclp-on-equivalence*: *equivalence*  $A$  ( $\text{reflclp-on } A$ )

( $\sim$ )  
<proof>

**lemma** *sympartp-dual*:  $(\text{sympartp } r)^- = \text{sympartp } r$   
<proof>

Restricted reflexive transitive closures should better be defined as the (unrestricted) transitive closure of restricted reflexive closure, since then facts of transitive closure can be reused.

**abbreviation** *rtranclp-on*  $A r \equiv \text{tranclp } (\text{reflclp-on } A r)$

**thm** *reflclp-on.tranclp-quasi-order*

**interpretation** *rtranclp-on*: *reflexive*  $A$  *rtranclp-on*  $A r$   
<proof>

**lemma** *rtranclp-on-eq-reflclp-on*:  $\text{rtranclp-on } A r = \text{reflclp-on } A (\text{tranclp } r)$   
<proof>

**lemma** *rtranclp-on-induct*[*consumes 1, case-names refl base step*]:

**assumes** \*: *rtranclp-on*  $A r x y$

**and** 0:  $x \in A \implies P x$

**and** 1:  $\bigwedge y. r x y \implies P y$

**and** *step*:  $\bigwedge y z. \text{tranclp } r x y \implies r y z \implies P y \implies P z$

**shows**  $P y$

<proof>

**abbreviation** *rsymclp-on*  $A r \equiv \text{symclp } (\text{reflclp-on } A r)$

**interpretation** *rsymclp-on*: *reflexive*  $A$  *rsymclp-on*  $A r$   
<proof>

**lemma** *rsymclp-on-eq-reflclp-on*:  $\text{rsymclp-on } A r = \text{reflclp-on } A (\text{symclp } r)$   
<proof>

**abbreviation** *rtransymclp-on*  $A r \equiv \text{tranclp } (\text{rsymclp-on } A r)$

**interpretation** *rtransymclp-on*: *reflexive*  $A$  *rtransymclp-on*  $A r$   
<proof>

**lemma** *rtransymclp-eq-reflclp*:  $\text{rtransymclp-on } A r = \text{reflclp-on } A (\text{tranclp } (\text{symclp } r))$   
<proof>

**lemma** *rtransymclp-on-equivalence*: *equivalence*  $A$  (*rtransymclp-on*  $A r$ )  
<proof>

**context** *relation-on* **begin**

**interpretation** *reflclp-on: relation-on A reflclp-on A* ( $\sqsubseteq$ )  
*<proof>*

**lemmas** *reflclp-relation-on = reflclp-on.relation-on-axioms*

**lemma** *relcompp-id-on[simp]:* ( $\sqsubseteq$ ) *OO id-on A =* ( $\sqsubseteq$ )  
*<proof>*

**lemma** *relcompp-id-on-ext[simp]:* ( $\sqsubseteq$ ) *OO id-on A OO r =* ( $\sqsubseteq$ ) *OO r*  
*<proof>*

**lemma** *id-on-relcompp[simp]:* *id-on A OO* ( $\sqsubseteq$ ) *=* ( $\sqsubseteq$ )  
*<proof>*

**lemma** *id-on-relcompp-ext[simp]:* *id-on A OO* ( $\sqsubseteq$ ) *OO r =* ( $\sqsubseteq$ ) *OO r*  
*<proof>*

**lemma** *reflclp-on-eq-reflclp-Restrp:*  
*reflclp-on A* ( $\sqsubseteq$ ) *= reflclp* ( $\sqsubseteq$ ) *↑ A*  
*<proof>*

**end**

## 2.2.2 Dual

**lemma** *dual-le-symclp:*  $r^- \leq \text{symclp } r$   
*<proof>*

**lemma** *list-all2-dual:* *list-all2*  $r^-$  *xs ys*  $\implies$  *list-all2*  $r$  *ys xs*  
*<proof>*

**lemma** *reflclp-dual:*  $(\text{reflclp } r)^- = \text{reflclp } r^-$  *<proof>*

**lemma** *rtranclp-dual:*  $(\text{rtranclp } r)^- = \text{rtranclp } r^-$   
*<proof>*

**lemma** *tranclp-dual:*  $(\text{tranclp } r)^- = \text{tranclp } r^-$   
*<proof>*

**lemma** *reflclp-on-dual:*  $(\text{reflclp-on } A \ r)^- = \text{reflclp-on } A \ r^-$   
*<proof>*

**lemma** *symclp-dual:*  $(\text{symclp } r)^- = \text{symclp } r$   
*<proof>*

**lemma** *tranclp-le-transymclp:*  $\text{tranclp } r \leq \text{tranclp } (\text{symclp } r)$   
*<proof>*

**lemmas** *tranclp-imp-transymclp =*

*tranclp-le-transymclp*[*THEN le-funD, THEN le-funD, THEN le-boolD, rule-format*]

**lemma** *dual-tranclp-le-transymclp*:  $(\text{tranclp } r)^- \leq \text{tranclp } (\text{symclp } r)$   
 ⟨*proof*⟩

**lemmas** *dual-tranclp-imp-transymclp* =  
*dual-tranclp-le-transymclp*[*THEN le-funD, THEN le-funD, THEN le-boolD, rule-format*]

**end**

### 3 More About Sorted Terms

**theory** *Sorted-Terms-More*  
**imports** *Sorted-Terms.Sorted-Contexts*  
**begin**

**declare** *domIff*[*iff del*]

**lemma** *grounding-subst*:  
**assumes**  $\vartheta: \vartheta :_s X \mid' XG \rightarrow \mathcal{T}(F, \emptyset)$  **and** *var*:  $\forall x \in \text{dom } X - XG. \vartheta x = \text{Var } x$   
**shows**  $\vartheta :_s X \rightarrow \mathcal{T}(F, X \mid' (-XG))$   
 ⟨*proof*⟩

Below defines an algebra where every element has a representation as a ground term. To avoid complication in the type system, the type of variables in the ground representations must be *unit*.

**locale** *sorted-algebra-represented* = *sorted-algebra* +  
*term-of-unit*: *sorted-map*  $\eta$  *A*  $\mathcal{T}(F)$  **for**  $\eta$  +  
**assumes** *eval-term-of-unit*:  $\bigwedge a \sigma \alpha. a : \sigma \text{ in } A \implies I[\eta a]\alpha = a$   
**begin**

Because locales do not support polymorphic parameters, we need to extend  $\eta$  in polymorphic to the type of variables.

**abbreviation** *term-of where* *term-of*  $a \equiv \eta a \cdot \text{undefined}$

**sublocale** *term-of*: *sorted-map* *term-of* *A*  $\mathcal{T}(F, V)$  **for** *V*  
 ⟨*proof*⟩

**lemma** *eval-term-of*: **assumes**  $a : \sigma \text{ in } A$  **shows**  $I[\text{term-of } a]\alpha = a$   
 ⟨*proof*⟩

**lemma** *map-eval-term-of*:  $ds :_i \sigma s \text{ in } A \implies [I[\text{term-of } d]\alpha. d \leftarrow ds] = ds$   
 ⟨*proof*⟩

**lemma** *eval-subst-term-of*:  
**assumes**  $s : s : \sigma \text{ in } \mathcal{T}(F, V)$  **and**  $\alpha : \alpha :_s V \rightarrow A$   
**shows**  $I[s \cdot (\text{term-of } \circ \alpha)]\beta = I[s]\alpha$   
 ⟨*proof*⟩

**lemma** *eval-upd-as-subst*:

**assumes**  $s: s : \sigma$  in  $\mathcal{T}(F, V)$

**and**  $\alpha: \alpha :_s V \rightarrow A$  **and**  $a: a : \tau$  in  $A$

**shows**  $I[s]\alpha(v:=a) = I[s \cdot \text{Var}(v:=\text{term-of } a)]\alpha$

*<proof>*

**lemma** *term-of-actxt*:

**assumes**  $C : \sigma \rightarrow \tau$  in *aContext*  $F A$

**shows**  $I\langle I[\text{map-args-actxt term-of } C]_c \alpha; a \rangle = I\langle C; a \rangle$

*<proof>*

**lemma** *term-of-subst*:  $a : \sigma$  in  $A \implies \text{term-of } a \cdot \vartheta = \text{term-of } a$

*<proof>*

**lemma** *term-of-unit-subst*:  $a : \sigma$  in  $A \implies \eta a \cdot \vartheta = \text{term-of } a$

*<proof>*

**lemma** *term-of-unit*:  $a : \sigma$  in  $A \implies \text{term-of } a = \eta a$

*<proof>*

**lemma** *map-term-of-subst*:  $ds ;_i \sigma s$  in  $A \implies [\text{term-of } d \cdot \vartheta. d \leftarrow ds] = \text{map term-of } ds$

*<proof>*

**end**

**locale** *sorted-algebra-const* = *sorted-algebra* +

*const: subsignature-algebra*  $C F$  +

*const: sorted-algebra-represented*  $C$  **for**  $C$

**begin**

**thm** *const.Term-subset*

**end**

Restricting range of a partial map.

**definition** *restrict-ran* (**infix**  $\uparrow^r$  100) **where**

$(A \uparrow^r S) a \equiv \text{if } A a \in \text{Some } 'S \text{ then } A a \text{ else None}$

**lemma** *ran-restrict-ran*:  $\text{ran } (A \uparrow^r S) = \text{ran } A \cap S$

*<proof>*

**lemma** *restrict-ran-le*:  $A \uparrow^r S \subseteq_m A$  *<proof>*

**lemma** *hastype-in-restrict-ran*:  $a : \sigma$  in  $A \uparrow^r S \iff a : \sigma$  in  $A \wedge \sigma \in S$

*<proof>*

**lemma** *hastype-in-restrict-ranI[intro!]*:  $a : \sigma$  in  $A \implies \sigma \in S \implies a : \sigma$  in  $A \uparrow^r S$

*<proof>*

Sorts of Signature

**definition** *sorts-ssig*  $F \equiv \bigcup \{ \text{set } \sigma s \cup \{ \tau \} \mid \sigma s \ \tau \ f. \ f : \sigma s \rightarrow \tau \text{ in } F \}$

**lemma** *assumes*  $f : \sigma s \rightarrow \tau \text{ in } F$

**shows** *sorts-ssig-arg*:  $\sigma \in \text{set } \sigma s \implies \sigma \in \text{sorts-ssig } F$

**and** *sorts-ssig-ret*:  $\tau \in \text{sorts-ssig } F$

*<proof>*

**lemma** *nth-arg-in-sorts-ssig*:

**assumes**  $f : \sigma s \rightarrow \tau \text{ in } F$

**shows**  $n < \text{length } \sigma s \implies \sigma s!n \in \text{sorts-ssig } F$

*<proof>*

**lemma** *hastype-in-sorts-ssig-ret*:

**assumes**  $f : \sigma s \rightarrow \tau \text{ in } F$

**shows**  $a : \tau \text{ in } A \uparrow^r \text{sorts-ssig } F \longleftrightarrow a : \tau \text{ in } A$

*<proof>*

**lemma** *hastype-in-sorts-ssig-args*:

**assumes**  $f : \sigma s \rightarrow \tau \text{ in } F$

**shows**  $as : \iota \sigma s \text{ in } A \uparrow^r \text{sorts-ssig } F \longleftrightarrow as : \iota \sigma s \text{ in } A$

*<proof>*

**lemma** *sorted-algebra-sorts-ssig*:

*sorted-algebra*  $F (A \uparrow^r \text{sorts-ssig } F) I \longleftrightarrow \text{sorted-algebra } F A I$

*<proof>*

**lemma** *sorted-algebra-carrier-cong*:

**assumes**  $A \uparrow^r \text{sorts-ssig } F = B \uparrow^r \text{sorts-ssig } F$

**shows** *sorted-algebra*  $F A I \longleftrightarrow \text{sorted-algebra } F B I$

*<proof>*

**lemma** (*in sorted-algebra*) *sorted-algebra-carrier*:

**assumes**  $A \uparrow^r \text{sorts-ssig } F = B \uparrow^r \text{sorts-ssig } F$

**shows** *sorted-algebra*  $F B I$

*<proof>*

**lemma** (*in sort-preserving*) *sorted-image-restrict-ran*:  $(f \text{ } ^{\text{as}} A) \uparrow^r S = f \text{ } ^{\text{as}} A \uparrow^r S$

*<proof>*

**end**

**theory** *Sorted-Relations*

**imports** *Sorted-Terms.Sorted-Sets Binary-Relations-More*

**begin**

**declare** *domIff*[*iff del*]

## 4 Sorted Relations

This part contains notions about binary relations over sorted sets.

### 4.1 Subject Reduction Property

**locale** *subject-reduction* =  
  **fixes**  $A$  **and** *less-eq* (**infix**  $\sqsubseteq$  50)  
  **assumes** *subject-reduction*:  $a \sqsubseteq b \implies a : \sigma \text{ in } A \implies b : \sigma \text{ in } A$

**lemma** *eq-subject-reduction*: *subject-reduction*  $A$  (=)  
   $\langle$ *proof* $\rangle$

**lemma** *Sup-subject-reduction*:  
  **assumes**  $\forall r \in R. \text{subject-reduction } A \ r$   
  **shows** *subject-reduction*  $A$  ( $\bigsqcup R$ )  
   $\langle$ *proof* $\rangle$

**context** *subject-reduction* **begin**

**sublocale** *relation-closed dom*  $A$   
   $\langle$ *proof* $\rangle$

**lemma** *closed-in-dom*:  $a \sqsubseteq b \implies a \in \text{dom } A \implies b \in \text{dom } A$   
   $\langle$ *proof* $\rangle$

**lemma** *list-all2-subject-reduction*: *list-all2* ( $\sqsubseteq$ )  $as \ bs \implies as :_i \ \sigma s \text{ in } A \implies bs :_i \ \sigma s \text{ in } A$   
   $\langle$ *proof* $\rangle$

**lemma** *relcompp-subject-reduction*:  
  **assumes** *subject-reduction*  $A \ r$   
  **shows** *subject-reduction*  $A$  ( $(\sqsubseteq) \circ r$ )  
   $\langle$ *proof* $\rangle$

**lemma** *relpowp-subject-reduction*: *subject-reduction*  $A$  ( $(\sqsubseteq) \overset{\sim}{\sim} n$ )  
   $\langle$ *proof* $\rangle$

**interpretation** *less-eq-symmetrize* $\langle$ *proof* $\rangle$

**lemma** *reflclp-subject-reduction*: *subject-reduction*  $A$  (*reflclp* ( $\sqsubseteq$ ))  
   $\langle$ *proof* $\rangle$

**lemma** *sympartp-subject-reduction*: *subject-reduction*  $A$  ( $\sim$ )  
   $\langle$ *proof* $\rangle$

**lemma** *equiv-subject-reduction*: *subject-reduction*  $A$  ( $\simeq$ )  
   $\langle$ *proof* $\rangle$

**lemma** *tranclp-subject-reduction*: *subject-reduction*  $A$  ( $\text{tranclp } (\sqsubseteq)$ )  
⟨*proof*⟩

**lemma** *rtranclp-subject-reduction*: *subject-reduction*  $A$  ( $\text{rtranclp } (\sqsubseteq)$ )  
⟨*proof*⟩

**lemma** *reflclp-on-subject-reduction*: *subject-reduction*  $A$  ( $\text{reflclp-on } (\text{dom } A) (\sqsubseteq)$ )  
⟨*proof*⟩

**lemma** *tranclp-reflclp-on-subject-reduction*: *subject-reduction*  $A$  ( $\text{reflclp-on } (\text{dom } A) (\text{tranclp } (\sqsubseteq))$ )  
⟨*proof*⟩

**end**

**lemma** *subject-reduction-cong*:  
 **assumes**  $r: \bigwedge a b. a \in \text{dom } A \implies r a b \longleftrightarrow r' a b$   
 **shows** *subject-reduction*  $A r \longleftrightarrow \text{subject-reduction } A r'$   
⟨*proof*⟩

**locale** *sorted-relation = subject-reduction + dual*: *subject-reduction*  $A$  ( $\sqsubseteq$ )<sup>-</sup>

**lemma** *Sup-sorted-relation*:  
 **assumes**  $\forall r \in R. \text{sorted-relation } A r$   
 **shows** *sorted-relation*  $A (\bigsqcup R)$   
⟨*proof*⟩

**lemma** *sup-sorted-relation*:  
 **assumes** *sorted-relation*  $A r$  *sorted-relation*  $A s$   
 **shows** *sorted-relation*  $A (r \sqcup s)$   
⟨*proof*⟩

**context** *sorted-relation* **begin**

**lemma** *relpowp-sorted-relation*: *sorted-relation*  $A ((\sqsubseteq) \sim^n)$   
⟨*proof*⟩

**lemma** *related-imp-hastype-iff*:  $a \sqsubseteq b \implies a : \sigma \text{ in } A \longleftrightarrow b : \sigma \text{ in } A$   
⟨*proof*⟩

**lemma** *related-imp-in-dom-iff*: **assumes**  $ab: a \sqsubseteq b$  **shows**  $a \in \text{dom } A \longleftrightarrow b \in \text{dom } A$   
⟨*proof*⟩

**lemma** *list-all2-sorted-relation*: *list-all2* ( $\sqsubseteq$ )  $as\ bs \implies as ;_l \sigma s \text{ in } A \longleftrightarrow bs ;_l \sigma s \text{ in } A$   
⟨*proof*⟩

**lemma** *dual-sorted-relation*: *sorted-relation*  $A (\sqsubseteq)$ <sup>-</sup> ⟨*proof*⟩

**lemma** *reflclp-sorted-relation: sorted-relation A (reflclp (□))*  
 ⟨proof⟩

**lemma** *tranclp-sorted-relation: sorted-relation A (tranclp (□))*  
 ⟨proof⟩

**lemma** *rtranclp-sorted-relation: sorted-relation A (rtranclp (□))*  
 ⟨proof⟩

**lemma** *reflclp-on-sorted-relation: sorted-relation A (reflclp-on (dom A) (□))*  
 ⟨proof⟩

**lemma** *tranclp-reflclp-on-sorted-relation: sorted-relation A (reflclp-on (dom A) (tranclp (□)))*  
 ⟨proof⟩

**lemma** *symclp-sorted-relation: sorted-relation A (symclp (□))*  
 ⟨proof⟩

**end**

**lemma** *sorted-relation-cong:*  
**assumes**  $r1: \bigwedge a b. a \in \text{dom } A \implies r a b \longleftrightarrow r' a b$   
**assumes**  $r2: \bigwedge a b. b \in \text{dom } A \implies r a b \longleftrightarrow r' a b$   
**shows** *sorted-relation A r*  $\longleftrightarrow$  *sorted-relation A r'*  
 ⟨proof⟩

**lemma** *sorted-relation-iff: sorted-relation A r*  $\longleftrightarrow$   $(\forall a b \sigma. r a b \longrightarrow a : \sigma \text{ in } A \longleftrightarrow b : \sigma \text{ in } A)$   
 ⟨proof⟩

**interpretation** *eq: sorted-relation A (=) for A*  
 ⟨proof⟩

**context** *subject-reduction* **begin**

**interpretation** *less-eq-symmetrize*⟨proof⟩

**lemma** *equiv-sorted-relation: sorted-relation A ( $\simeq$ )*  
 ⟨proof⟩

**end**

**end**

**theory** *Sorted-Rules*  
**imports** *Sorted-Terms.Sorted-Terms*  
**begin**

## 5 Sorted Rules

This part defines datatypes for sorted rewrite rules and inference rules.

**declare** *Ball-Pair-conv*[*simp*]

### 5.1 Equation

An *equation* is a pair of terms.

**type-synonym** *(f,v) term-pair* = *(f,v) term* × *(f,v) term*

**abbreviation** *Equation* :: *(f,v) term* ⇒ *(f,v) term* ⇒ - (- ~> - [51,51]24)  
**where**  $s \rightsquigarrow t \equiv (s,t)$

Equation  $s \rightsquigarrow t$  represents that the two terms are in relation, where the relation is specified later.

**syntax**

*-ball-term-pair* :: *pttrn* ⇒ *pttrn* ⇒ *(f,v) term-pair set* ⇒ *bool* ⇒ *bool*  
 $((\exists \forall ((- \rightsquigarrow -') / \in -) / -) [0, 0, 0, 10] 10)$   
*-bex-term-pair* :: *pttrn* ⇒ *pttrn* ⇒ *(f,v) term-pair set* ⇒ *bool* ⇒ *bool*  
 $((\exists \exists ((- \rightsquigarrow -') / \in -) / -) [0, 0, 0, 10] 10)$

**translations**

$\forall (x \rightsquigarrow y) \in R. e \rightarrow \forall (x,y) \in R. e$   
 $\exists (x \rightsquigarrow y) \in R. e \rightarrow \exists (x,y) \in R. e$

**primrec** *vars-term-pair* **where** *vars-term-pair* ( $l \rightsquigarrow r$ ) = *vars*  $l \cup \text{vars } r$

**adhoc-overloading** *vars* ⇌ *vars-term-pair*

**abbreviation** *vars-term-pair-set* :: *(f,v) term-pair set* ⇒ *'v set* **where**  
*vars-term-pair-set*  $C \equiv \bigcup (\text{vars } 'C)$

**adhoc-overloading** *vars* ⇌ *vars-term-pair-set*

**abbreviation** *vars-term-pair-list* :: *(f,v) term-pair list* ⇒ *'v set* **where**  
*vars-term-pair-list*  $cs \equiv \text{vars } (\text{set } cs)$

**adhoc-overloading** *vars* ⇌ *vars-term-pair-list*

**primrec** *map-vars-term-pair* **where**

*map-vars-term-pair*  $f (l \rightsquigarrow r) = (\text{map-vars } f l \rightsquigarrow \text{map-vars } f r)$

**adhoc-overloading** *map-vars* ⇌ *map-vars-term-pair*

**adhoc-overloading** *map-vars* ⇌  $\text{map} \circ \text{map-vars-term-pair}$

## 5.2 Axioms – Rewrite Rules

An axiom is an equation whose variables are considered universally quantified. So the following datatype additionally specifies the sorted set of variables which it can contain. An axiom is also seen as a rewrite rule.

**datatype** (*dead 'f, dead 'v, dead 's*) *axiom* =  
*Axiom* (*vars: 'v → 's*) (*lhs: ('f,'v) term*) (*rhs: ('f,'v) term*)  
 (*-. - ~> - [100,51,51]21*)

**hide-const** (**open**) *axiom.vars axiom.lhs axiom.rhs*

**declare** *axiom.split[split] axiom.split-asm[split]*

**abbreviation** *ball-axiom* **where** *ball-axiom R P*  $\equiv \forall X l r. (X. l \rightsquigarrow r) \in R \longrightarrow P X l r$

**abbreviation** *bex-axiom* **where** *bex-axiom R P*  $\equiv \exists X l r. (X. l \rightsquigarrow r) \in R \wedge P X l r$

**syntax** *-ball-axiom* :: *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  (*'f,'v,'s*) *axiom set*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool*  
 (( $\exists \forall (('(-. - \rightsquigarrow -)/ \in -). / -)$  [0, 0, 0, 0, 10] 10)  
*-bex-axiom* :: *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  (*'f,'v,'s*) *axiom set*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool*  
 (( $\exists \exists (('(-. - \rightsquigarrow -)/ \in -). / -)$  [0, 0, 0, 0, 10] 10)

**translations**

$\forall (X. l \rightsquigarrow r) \in R. e \equiv \text{CONST } \textit{ball-axiom } R (\lambda X l r. e)$   
 $\exists (X. l \rightsquigarrow r) \in R. e \equiv \text{CONST } \textit{bex-axiom } R (\lambda X l r. e)$

## 5.3 Inference Rules – Conditional Rewrite Rules

An inference rule is an axiom extended with a list of premises. An inference rule is also seen as a conditional rewrite rule.

**datatype** (*dead 'f, dead 'v, dead 's*) *rule* =  
*Rule* (*vars: 'v → 's*) (*lhs: ('f,'v) term*) (*rhs: ('f,'v) term*) (*prems: (('f,'v) term-pair)*  
*list*)  
 (*-. - ~> -  $\Leftarrow$  - [100,51,51,51]21*)

**hide-const** (**open**) *rule.vars rule.lhs rule.rhs rule.prems*

**declare** *rule.split[split] rule.split-asm[split]*

**lemma** *rule-split-all*:  $(\forall \rho. P \rho) \longleftrightarrow (\forall V l r cs V. P (V. l \rightsquigarrow r \Leftarrow cs))$   
 (*proof*)

**abbreviation** *ball-rule* **where** *ball-rule R P*  $\equiv \forall X l r cs. (X. l \rightsquigarrow r \Leftarrow cs) \in R \longrightarrow P X l r cs$

**abbreviation** *bex-rule* **where** *bex-rule R P*  $\equiv \exists X l r cs. (X. l \rightsquigarrow r \Leftarrow cs) \in R \wedge P X l r cs$

### syntax

-ball-rule :: pttrn ⇒ pttrn ⇒ pttrn ⇒ pttrn ⇒ ('f,'v,'s) rule set ⇒ bool ⇒ bool  
((∃∀('(-. - ∼ - ← -')/ ∈ -)/ -) [0, 0, 0, 0, 0, 10] 10)  
-bex-rule :: pttrn ⇒ pttrn ⇒ pttrn ⇒ pttrn ⇒ ('f,'v,'s) rule set ⇒ bool ⇒ bool  
((∃∃('(-. - ∼ - ← -')/ ∈ -)/ -) [0, 0, 0, 0, 0, 10] 10)

### translations

∀(X. l ∼ r ← cs) ∈ R. e ⇒ CONST ball-rule R (λX l r cs. e)  
∃(X. l ∼ r ← cs) ∈ R. e ⇒ CONST bex-rule R (λX l r cs. e)

### primrec vars-rule where

vars-rule (V. l ∼ r ← cs) = vars l ∪ vars r ∪ vars cs

### adhoc-overloading vars ⇒ vars-rule

### primrec map-vars-rule where

map-vars-rule f (V. l ∼ r ← cs) = ((f <sup>cs</sup> V). map-vars f l ∼ map-vars f r ← map-vars f cs)

### adhoc-overloading map-vars ⇒ map-vars-rule

**lemma** vars-rule-lhs: vars (rule.lhs ρ) ⊆ vars ρ  
**and** vars-rule-rhs: vars (rule.rhs ρ) ⊆ vars ρ  
⟨proof⟩

### lemma

**assumes** (s ∼ t) ∈ set (rule.prem s ρ)  
**shows** vars-rule-prems-left: vars s ⊆ vars ρ  
**and** vars-rule-prems-right: vars t ⊆ vars ρ  
⟨proof⟩

One can see axioms as unconditional inference rules.

**definition** unconditional **where** unconditional ≡ λ(V. l ∼ r) ⇒ V. l ∼ r ← []

**lemma** unconditional[simp]: unconditional (V. l ∼ r) = (V. l ∼ r ← []) ⟨proof⟩

**lemma** in-unconditional[simp]: (V. l ∼ r ← cs) ∈ unconditional ' R ↔ (V. l ∼ r) ∈ R ∧ cs = []  
⟨proof⟩

## 5.4 Sortedness of Rules

When rules are well-typed, then the derivation is a sorted relation. Generally, we allow relating terms of different sorts.

**definition** sorted-rule F ≡ λ(V. l ∼ r ← cs) ⇒  
l ∈ dom T(F, V) ∧ r ∈ dom T(F, V) ∧  
(∀(s ∼ t) ∈ set cs. s ∈ dom T(F, V) ∧ t ∈ dom T(F, V))

**lemma** *sorted-rule*:

*sorted-rule*  $F (V. l \rightsquigarrow r \Leftarrow cs) \longleftrightarrow l \in \text{dom } \mathcal{T}(F, V) \wedge r \in \text{dom } \mathcal{T}(F, V) \wedge$   
 $(\forall (s \rightsquigarrow t) \in \text{set } cs. s \in \text{dom } \mathcal{T}(F, V) \wedge t \in \text{dom } \mathcal{T}(F, V))$   
*<proof>*

**lemma** *sorted-ruleI*:

**assumes**  $\varrho = (V. l \rightsquigarrow r \Leftarrow cs)$   
**and**  $l : \sigma$  *in*  $\mathcal{T}(F, V)$  **and**  $r : \sigma'$  *in*  $\mathcal{T}(F, V)$   
**and**  $\bigwedge s t. (s \rightsquigarrow t) \in \text{set } cs \implies \exists \tau \tau'. s : \tau$  *in*  $\mathcal{T}(F, V) \wedge t : \tau'$  *in*  $\mathcal{T}(F, V)$   
**shows** *sorted-rule*  $F \varrho$   
*<proof>*

**lemma** **assumes** *sorted-rule*  $F (V. l \rightsquigarrow r \Leftarrow cs)$

**shows** *sorted-rule-cond-domD*:  
 $\bigwedge s t. (s \rightsquigarrow t) \in \text{set } cs \implies s \in \text{dom } \mathcal{T}(F, V) \wedge t \in \text{dom } \mathcal{T}(F, V)$   
**and** *sorted-rule-domD*:  $l \in \text{dom } \mathcal{T}(F, V) \wedge r \in \text{dom } \mathcal{T}(F, V)$   
*<proof>*

Often we consider relating a term to a term of the same sort. We say a rule is *sort safe* if the conclusion relates terms of the same sort, when all conditions do so.

**definition** *sort-safe* **where** *sort-safe*  $F \varrho \equiv$

*sorted-rule*  $F \varrho \wedge$  (  
*case*  $\varrho$  *of*  $(V. l \rightsquigarrow r \Leftarrow cs) \implies$   
 $((\forall (s \rightsquigarrow t) \in \text{set } cs. \exists \tau. s : \tau$  *in*  $\mathcal{T}(F, V) \wedge t : \tau$  *in*  $\mathcal{T}(F, V)) \longrightarrow$   
 $(\exists \sigma. l : \sigma$  *in*  $\mathcal{T}(F, V) \wedge r : \sigma$  *in*  $\mathcal{T}(F, V)))$ )

**lemma** *sort-safeI*:

**assumes**  $\varrho = (V. l \rightsquigarrow r \Leftarrow cs)$   
**and**  $l : \sigma$  *in*  $\mathcal{T}(F, V)$  **and**  $r : \sigma'$  *in*  $\mathcal{T}(F, V)$   
**and**  $\bigwedge s t. (s \rightsquigarrow t) \in \text{set } cs \implies \exists \tau \tau'. s : \tau$  *in*  $\mathcal{T}(F, V) \wedge t : \tau'$  *in*  $\mathcal{T}(F, V)$   
**and**  $\forall (s \rightsquigarrow t) \in \text{set } cs. \exists \tau. s : \tau$  *in*  $\mathcal{T}(F, V) \wedge t : \tau$  *in*  $\mathcal{T}(F, V) \implies \exists \sigma. l : \sigma$   
*in*  $\mathcal{T}(F, V) \wedge r : \sigma$  *in*  $\mathcal{T}(F, V)$   
**shows** *sort-safe*  $F \varrho$   
*<proof>*

**lemma** *sort-safe-imp-sorted*: *sort-safe*  $F \varrho \implies$  *sorted-rule*  $F \varrho$

*<proof>*

**lemma**

**assumes** *sort-safe*  $F (V. l \rightsquigarrow r \Leftarrow cs)$   
**shows** *sort-safeD*:  
 $\forall (s \rightsquigarrow t) \in \text{set } cs. \exists \tau. s : \tau$  *in*  $\mathcal{T}(F, V) \wedge t : \tau$  *in*  $\mathcal{T}(F, V) \implies \exists \sigma. l : \sigma$  *in*  
 $\mathcal{T}(F, V) \wedge r : \sigma$  *in*  $\mathcal{T}(F, V)$   
*<proof>*

**definition** *sort-safe-rules*  $F R \equiv \forall \varrho \in R. \text{sort-safe } F \varrho$

**lemma** *sort-safe-rules-Un*: *sort-safe-rules*  $F (R \cup S) \longleftrightarrow \text{sort-safe-rules } F R \wedge$

*sort-safe-rules*  $F S$   
 ⟨*proof*⟩

**lemmas** *sort-safe-rules-UnI*[*intro!*] = *sort-safe-rules-Un*[*THEN iffD2, unfolded conj-imp-eq-imp-imp*]

**abbreviation** *sorted-rules*  $F R \equiv \forall \rho \in R. \text{sorted-rule } F \rho$

**lemma** *sort-safe-rules-imp-sorted*: *sort-safe-rules*  $F R \implies \text{sorted-rules } F R$   
 ⟨*proof*⟩

**end**

## 6 Models

**theory** *Models*  
**imports** *Sorted-Rules*  
**begin**

Here we formalize *relational models* [5] of sets of rules. Relational models assert two objects are in specified relation where standard models assert equality.

### 6.1 Satisfaction of Equations

Given an interpretation of function symbols and a relation, a variable assignment *satisfies* an equation if the evaluation of the terms are in relation:

**definition** *satisfies*  $((\mathcal{L}(-;/-/ \models)/ (- \rightsquigarrow/ -)) [51,51,3,51,51]4)$  **where**  
 $I:(\sqsubseteq);\alpha \models s \rightsquigarrow t \equiv I[s]\alpha \sqsubseteq I[t]\alpha$  **for** *le* (**infix**  $\sqsubseteq$  50)

Particularly important case is where the relation is the equality.

**abbreviation** *eq-satisfies*  $((\mathcal{L}(-;/- \models)/ (- =/ -)) [51,3,51,51]4)$  **where**  
 $I;\alpha \models s = t \equiv I:(=);\alpha \models s \rightsquigarrow t$

**lemmas** *satisfiesI* = *satisfies-def*[*unfolded atomize-eq, THEN iffD2*]  
**lemmas** *satisfiesD* = *satisfies-def*[*unfolded atomize-eq, THEN iffD1*]  
**lemmas** *satisfiesE* = *satisfiesD*[*elim-format*]

**lemma** *satisfies-mono*:

**assumes**  $r \leq r'$  **shows** *satisfies*  $I r \leq \text{satisfies } I r'$   
 ⟨*proof*⟩

**lemma** *dual-satisfies[simp]*:  $(\text{satisfies } I r \alpha)^- = \text{satisfies } I r^- \alpha$   
 ⟨*proof*⟩

**lemma** *satisfies-subst*:  $(I:r;\alpha \models s \cdot \vartheta \rightsquigarrow t \cdot \vartheta) \longleftrightarrow (I:r;I[\vartheta]_s \alpha \models s \rightsquigarrow t)$   
 ⟨*proof*⟩

**lemma** *Term-Var-satisfies[simp]*:  
 $(Fun:(\sqsubseteq); Var \models s \rightsquigarrow t) \longleftrightarrow s \sqsubseteq t$  for *less-eq* (**infix**  $\sqsubseteq$  50)  
 $\langle proof \rangle$

**lemma** *satisfies-same-vars*:  
**assumes**  $\forall x \in vars\ s \cup vars\ t. \alpha\ x = \beta\ x$   
**shows**  $(I:r;\alpha \models s \rightsquigarrow t) \longleftrightarrow (I:r;\beta \models s \rightsquigarrow t)$   
 $\langle proof \rangle$

**interpretation** *eq-satisfies: equivalence UNIV eq-satisfies I alpha*  
 $\langle proof \rangle$

**lemma** (**in sorted-algebra**) *eq-satisfies-has-same-type*:  
**assumes**  $\alpha: \alpha :_s V \rightarrow A$  **and**  $st: I;\alpha \models s = t$   
**and**  $s: s : \sigma$  in  $\mathcal{T}(F, V)$  **and**  $t: t : \tau$  in  $\mathcal{T}(F, V)$   
**shows**  $\sigma = \tau$   
 $\langle proof \rangle$

## 6.2 Validity of Axioms

An axiom is *valid* if any assignment satisfies it.

**definition** *valid*  $((2(-:/- \models)/ -/ (- \rightsquigarrow/ -)) [51,51,51,100,51,51]_4)$  **where**  
 $A:I:(\sqsubseteq) \models V. s \rightsquigarrow t \equiv \forall \alpha :_s V \rightarrow A. (I:(\sqsubseteq);\alpha \models s \rightsquigarrow t)$  for *le* (**infix**  $\sqsubseteq$  50)

**abbreviation** *valid-eq*  $((2(-:/- \models)/ -/ (- =/ -)) [51,51,100,51,51]_4)$  **where**  
 $A:I \models V. s = t \equiv A:I:(=) \models V. s \rightsquigarrow t$

**lemmas** *validI[intro?]* = *valid-def[unfolded atomize-eq, THEN iffD2, rule-format]*  
**lemmas** *validD* = *valid-def[unfolded atomize-eq, THEN iffD1, rule-format, simp]*  
**lemmas** *validE* = *valid-def[unfolded atomize-eq, THEN iffD1, elim-format, rule-format]*

**interpretation** *valid-eq: equivalence UNIV valid-eq A I V*  
 $\langle proof \rangle$

**lemma** *valid-mono*:  
**assumes**  $r \leq r'$  **shows** *valid A I r*  $\leq$  *valid A I r'*  
 $\langle proof \rangle$

**lemma** *dual-valid[simp]*:  $(\text{valid } A\ I\ r\ V)^{\neg} = \text{valid } A\ I\ r^{\neg}\ V$   
 $\langle proof \rangle$

**lemma** (**in sorted-algebra**) *valid-subst-closed*:  
**assumes**  $st: A:I:r \models V. s \rightsquigarrow t$  **and**  $\vartheta: \vartheta :_s V \rightarrow \mathcal{T}(F, W)$   
**shows**  $A:I:r \models W. s \cdot \vartheta \rightsquigarrow t \cdot \vartheta$   
 $\langle proof \rangle$

Validity in the term algebra with respect to a relation implies the relation (and vice versa if the relation is closed under substitution).

**lemma** *Term-valid-relates*:  $\mathcal{T}(F, V):Fun:r \models V. s \rightsquigarrow t \implies r\ s\ t$

$\langle proof \rangle$

### 6.2.1 Models

An algebra coupled with a binary relation *models* an inference rule if all assignments that satisfy the premise satisfy the conclusion.

**definition** *models-rule*  $((\mathcal{L}:-/- \models - / (\mathcal{L} \rightsquigarrow / - \Leftarrow / -)) [51,51,51,100,51,51,51]_4)$  **where**

$$A:I:e \models V. l \rightsquigarrow r \Leftarrow cs \equiv \\ \forall \alpha :_s V \rightarrow A. (\forall (s \rightsquigarrow t) \in \text{set } cs. (I:e;\alpha \models s \rightsquigarrow t)) \longrightarrow (I:e;\alpha \models l \rightsquigarrow r)$$

**abbreviation** *models-rule-eq*  $((\mathcal{L}:-/- \models - / (\mathcal{L} = / - \Leftarrow / -)) [51,51,100,51,51,51]_4)$  **where**

$$A:I \models V. l = r \Leftarrow cs \equiv A:I:(=) \models V. l \rightsquigarrow r \Leftarrow cs$$

**lemma** *models-ruleI*:

**assumes**  $\bigwedge \alpha. \alpha :_s V \rightarrow A \implies \forall (s \rightsquigarrow t) \in \text{set } cs. (I:e;\alpha \models s \rightsquigarrow t) \implies I:e;\alpha \models l \rightsquigarrow r$

**shows**  $A:I:e \models V. l \rightsquigarrow r \Leftarrow cs$

$\langle proof \rangle$

**lemma** *models-ruleD*:

$$A:I:e \models V. l \rightsquigarrow r \Leftarrow cs \implies$$

$$\alpha :_s V \rightarrow A \implies (\bigwedge s t. (s \rightsquigarrow t) \in \text{set } cs \implies (I:e;\alpha \models s \rightsquigarrow t)) \implies I:e;\alpha \models l \rightsquigarrow r$$

$\langle proof \rangle$

An algebra coupled with a binary relation *models* a set of rules, if it models all the rules.

**definition** *models*  $((\mathcal{L}:-/- \models -) [51,51,51,51]_4)$  **where**

$$A:I:e \models R \equiv \forall (V. l \rightsquigarrow r \Leftarrow cs) \in R. (A:I:e \models V. l \rightsquigarrow r \Leftarrow cs)$$

**abbreviation** *models-eq*  $((\mathcal{L}:-/- \models -) [51,51,51]_4)$  **where**

$$A:I \models R \equiv A:I:(=) \models R$$

**lemmas** *modelsI* = *models-def*[*unfolded atomize-eq*, *THEN iffD2*, *rule-format*]

**lemmas** *modelsD* = *models-def*[*unfolded atomize-eq*, *THEN iffD1*, *rule-format*]

**lemmas** *modelsE* = *models-def*[*unfolded atomize-eq*, *THEN iffD1*, *elim-format*, *rule-format*]

**lemma** *models-empty[simp]*:  $A:I:e \models \{\}$   $\langle proof \rangle$

**lemma** *models-Un[simp]*:  $(A:I:e \models R \cup R') \longleftrightarrow (A:I:e \models R) \wedge (A:I:e \models R')$   $\langle proof \rangle$

**lemma** *models-cmono*:  $R \subseteq R' \implies A:I:e \models R' \implies A:I:e \models R$   $\langle proof \rangle$

**lemma** *models-axiom[simp]*:  $(A:I:e \models V. l \rightsquigarrow r \Leftarrow []) \longleftrightarrow (A:I:e \models V. l \rightsquigarrow r)$

*⟨proof⟩*

**lemma** *models-unconditional-iff:*

$(A:I:(\sqsubseteq) \models \text{unconditional } \text{' } R) \iff (\forall (X. l \rightsquigarrow r) \in R. \forall \alpha :_s X \rightarrow A. I[\![l]\!] \alpha \sqsubseteq I[\![r]\!] \alpha)$

**for** *rel* (**infix**  $\sqsubseteq$  50)

*⟨proof⟩*

**lemmas** *models-unconditionalI =*

*models-unconditional-iff [THEN iffD2, rule-format]*

**lemmas** *models-unconditionalD =*

*models-unconditional-iff [THEN iffD1, rule-format]*

**lemma** *models-unconditional-mono:*

**assumes**  $rel \leq rel'$

**shows**  $(A:I:rel \models \text{unconditional } \text{' } R) \implies (A:I:rel' \models \text{unconditional } \text{' } R)$

*⟨proof⟩*

**end**

## 7 Ordered Algebras

We formalize algebras associated with a binary relation, especially those in which properties of the binary relation is preserved in terms. Order properties are assumed only over well-typed elements ( $\text{dom } A$ ), as we do not want to specify how ill-typed elements are related.

**locale** *reflexive-algebra = sorted-algebra + reflexive dom A*

**begin**

**lemma** *satisfies-reflexive:*

**assumes**  $\alpha :_s V \rightarrow A$

**shows** *reflexive (dom  $\mathcal{T}(F, V)$ ) (satisfies  $I (\sqsubseteq) \alpha$ )*

*⟨proof⟩*

**sublocale** *valid: reflexive dom  $\mathcal{T}(F, V)$  valid A I ( $\sqsubseteq$ ) V*

*⟨proof⟩*

**sublocale** *lists: reflexive ⟨lists (dom A)⟩ ⟨list-all2 ( $\sqsubseteq$ )⟩ ⟨proof⟩*

**end**

**locale** *attractive-algebra = sorted-algebra + attractive dom A*

**begin**

**lemma** *satisfies-attractive:*

**assumes**  $\alpha :_s V \rightarrow A$

**shows** *attractive (dom  $\mathcal{T}(F, V)$ ) (satisfies  $I (\sqsubseteq) \alpha$ )*

$\langle proof \rangle$   
**sublocale** *valid: attractive dom  $\mathcal{T}(F, V)$  valid  $A I (\sqsubseteq) V$*   
 $\langle proof \rangle$   
**sublocale** *lists: attractive  $\langle lists (dom A) \rangle \langle list-all2 (\sqsubseteq) \rangle \langle proof \rangle$*   
**end**  
**locale** *transitive-algebra = sorted-algebra + transitive dom  $A$*   
**begin**  
**sublocale** *attractive-algebra* $\langle proof \rangle$   
**lemma** *satisfies-transitive:*  
**assumes**  $\alpha: \alpha :_s V \rightarrow A$   
**shows** *transitive (dom  $\mathcal{T}(F, V)$ ) (satisfies  $I (\sqsubseteq) \alpha$ )*  
 $\langle proof \rangle$   
**sublocale** *valid: transitive  $\langle dom \mathcal{T}(F, V) \rangle \langle valid A I (\sqsubseteq) V \rangle$  for  $V$*   
 $\langle proof \rangle$   
**sublocale** *lists: transitive  $\langle lists (dom A) \rangle \langle list-all2 (\sqsubseteq) \rangle \langle proof \rangle$*   
**end**  
**locale** *quasi-ordered-algebra = sorted-algebra + quasi-ordered-set dom  $A$*   
**begin**  
**sublocale** *transitive-algebra + reflexive-algebra* $\langle proof \rangle$   
**lemma** *satisfies-quasi-order:  $\alpha :_s V \rightarrow A \implies quasi-ordered-set (dom \mathcal{T}(F, V))$*   
*(satisfies  $I (\sqsubseteq) \alpha$ )*  
 $\langle proof \rangle$   
**sublocale** *valid: quasi-ordered-set dom  $\mathcal{T}(F, V)$  valid  $A I (\sqsubseteq) V$*  $\langle proof \rangle$   
**sublocale** *lists: quasi-ordered-set  $\langle lists (dom A) \rangle \langle list-all2 (\sqsubseteq) \rangle \langle proof \rangle$*   
**end**  
We do not consider antisymmetry alone here: it is not preserved in the term algebra, because different terms may have the same evaluation. So we just define ordered algebra and derive that terms are quasi-ordered.  
**locale** *ordered-algebra = sorted-algebra + partially-ordered-set dom  $A$*   
**begin**  
**sublocale** *quasi-ordered-algebra* $\langle proof \rangle$

**end**

Irreflexivity is carried over to terms, if the algebra is inhabited.

**context** *inhabited* **begin**

**lemma** *Term-inhabited: inhabited*  $\mathcal{T}(F,A)$   
*<proof>*

**end**

**locale** *irreflexive-algebra = sorted-algebra + irreflexive dom A + inhabited*  
**begin**

**lemma** *satisfies-irreflexive:*  
**assumes**  $\alpha: \alpha :_s V \rightarrow A$   
**shows** *irreflexive (dom  $\mathcal{T}(F,V)$ ) (satisfies I ( $\sqsubset$ )  $\alpha$ )*  
*<proof>*

**sublocale** *valid: irreflexive dom  $\mathcal{T}(F,V)$  valid A I ( $\sqsubset$ ) V*  
*<proof>*

**end**

Hence, strict order is carried over to terms.

**locale** *strict-ordered-algebra = sorted-algebra + strict-ordered-set dom A + inhabited*  
**begin**

**sublocale** *transitive-algebra F A I ( $\sqsubset$ ) + irreflexive-algebra**<proof>*

**sublocale** *valid: strict-ordered-set dom  $\mathcal{T}(F,V)$  valid A I ( $\sqsubset$ ) V**<proof>*

**end**

**end**

## 8 Monotone Algebras

For monotonicity, we only consider elements of the same type. It is possible to consider elements of different types and overloaded function symbols, but then one cannot have the nice correspondence of monotonicity and congruence under quasi-order.

**locale** *monotone = fixes F A I and less-eq (infix  $\sqsubseteq$  50)*  
**assumes** *comp-arg:*  
 $a \sqsubseteq b \implies$   
 $f : \pi s @ \sigma \# \rho s \rightarrow \tau \text{ in } F \implies$   
 $ls :_l \pi s \text{ in } A \implies a : \sigma \text{ in } A \implies b : \sigma \text{ in } A \implies rs :_l \rho s \text{ in } A \implies$

$I f (ls @ a \# rs) \sqsubseteq I f (ls @ b \# rs)$

**lemma** *eq-monotone*: *monotone F A I (=)*  
 ⟨*proof*⟩

**locale** *monotone-algebra = sorted-algebra + monotone*  
**begin**

The compatibility of the interpretation of function symbols carries over to contexts.

**lemma** *ctxt-closed*:  
**assumes** *ab*:  $a \sqsubseteq b$  **and** *C*:  $C : \sigma \rightarrow \tau$  *in aContext F A*  
**and** *a*:  $a : \sigma$  *in A* **and** *b*:  $b : \sigma$  *in A*  
**shows**  $I\langle C;a \rangle \sqsubseteq I\langle C;b \rangle$   
 ⟨*proof*⟩

**lemma** *ctxtI*:  
**assumes** *ab*:  $a \sqsubseteq b$  **and** *C*:  $C : \sigma \rightarrow \tau$  *in aContext F A*  
**and** *a*:  $a : \sigma$  *in A* **and** *b*:  $b : \sigma$  *in A*  
**and** *s*:  $s = I\langle C;a \rangle$  **and** *t*:  $t = I\langle C;b \rangle$   
**shows**  $s \sqsubseteq t$   
 ⟨*proof*⟩

**interpretation** *less-eq-dualize*⟨*proof*⟩

**interpretation** *dual*: *monotone-algebra F A I ( $\sqsupseteq$ )*  
 ⟨*proof*⟩

**lemmas** *dual-monotone = dual.monotone-axioms*

**sublocale** *symclp*: *monotone F A I symclp ( $\sqsubseteq$ )*  
 ⟨*proof*⟩

**sublocale** *reflclp*: *monotone F A I reflclp ( $\sqsubseteq$ )*  
 ⟨*proof*⟩

**sublocale** *reflclp-on*: *monotone F A I reflclp-on (dom A) ( $\sqsubseteq$ )*  
 ⟨*proof*⟩

**lemma** *monotone-subalgebra*:  
**assumes** *sub*:  $A' \subseteq_m A$  **and** *alg*: *sorted-algebra F A' I*  
**shows** *monotone-algebra F A' I ( $\sqsubseteq$ )*  
 ⟨*proof*⟩

**context** *fixes*  $\alpha V$  **assumes**  $\alpha : \alpha :_s V \rightarrow A$   
**begin**

**interpretation** *satisfies*: *monotone-algebra F  $\mathcal{T}(F, V)$  Fun satisfies I ( $\sqsubseteq$ )  $\alpha$*   
 ⟨*proof*⟩

**lemmas** *satisfies-ctxt-closed* = *satisfies ctxt-closed*

**lemmas** *satisfies-monotone* = *satisfies.monotone-axioms*

**end**

**interpretation** *valid: monotone-algebra F T(F,V) Fun valid A I ( $\sqsubseteq$ ) V*  
*<proof>*

**sublocale** *valid: monotone F T(F,V) Fun valid A I ( $\sqsubseteq$ ) V<proof>*

**lemmas** *valid-ctxt-closed* = *valid ctxt-closed*

**lemma** *dual-monotone-algebra: monotone-algebra F A I ( $\sqsubseteq$ )<sup>-</sup>*  
*<proof>*

**end**

**lemma** *monotone-algebra-cong:*

**assumes** *r:  $\bigwedge a b. a \in \text{dom } A \implies b \in \text{dom } A \implies r a b \longleftrightarrow r' a b$*

**shows** *monotone-algebra F A I r  $\longleftrightarrow$  monotone-algebra F A I r'*

*<proof>*

Monotonicity can be derived from closure under contexts.

**lemma** *(in sorted-algebra) monotone-iff-ctxt-closed:*

**fixes** *less-eq (infix  $\sqsubseteq$  50)*

**shows** *monotone F A I ( $\sqsubseteq$ )  $\longleftrightarrow$  ( $\forall C \sigma \tau a b.$*

*C :  $\sigma \rightarrow \tau$  in aContext F A  $\longrightarrow$  a :  $\sigma$  in A  $\longrightarrow$  b :  $\sigma$  in A  $\longrightarrow$  a  $\sqsubseteq$  b  $\longrightarrow$  I(C;a)*

*$\sqsubseteq$  I(C;b))*

*(is ?l  $\longleftrightarrow$  ?r)*

*<proof>*

**lemma** *(in sorted-algebra) Sup-monotone:*

**assumes**  *$\forall r \in R. \text{monotone } F A I r$*

**shows** *monotone F A I ( $\bigsqcup R$ )*

*<proof>*

**lemma** *(in sorted-algebra) sup-monotone:*

**assumes** *monotone F A I r monotone F A I s*

**shows** *monotone F A I (r  $\sqcup$  s)*

*<proof>*

Monotonicity is preserved by the transitive closure, if the relation satisfies the subject reduction property.

**locale** *monotone-algebra-subject-reduction* = *sorted-algebra + monotone + subject-reduction*

**begin**

**sublocale** *monotone-algebra*⟨*proof*⟩

**lemma** *relcompp-monotone-algebra*:  
  **assumes** *monotone-algebra F A I r*  
  **shows** *monotone F A I (( $\sqsubseteq$ ) OO r)*  
  ⟨*proof*⟩

**interpretation** *relpowp: monotone-algebra-subject-reduction F A I ( $\sqsubseteq$ )  $\sim^n$*   
  ⟨*proof*⟩

**sublocale** *relpowp: monotone-algebra F A I ( $\sqsubseteq$ )  $\sim^n$*   
  + *relpowp: subject-reduction A ( $\sqsubseteq$ )  $\sim^n$* ⟨*proof*⟩

**interpretation** *reflclp: monotone-algebra-subject-reduction F A I reflclp ( $\sqsubseteq$ )*  
  ⟨*proof*⟩

**sublocale** *reflclp: monotone-algebra F A I reflclp ( $\sqsubseteq$ )*  
  + *reflclp: subject-reduction A reflclp ( $\sqsubseteq$ )*⟨*proof*⟩

**interpretation** *tranclp: monotone-algebra-subject-reduction F A I tranclp ( $\sqsubseteq$ )*  
  ⟨*proof*⟩

**sublocale** *tranclp: monotone-algebra F A I tranclp ( $\sqsubseteq$ )*  
  + *tranclp: subject-reduction A tranclp ( $\sqsubseteq$ )*⟨*proof*⟩

**interpretation** *rtranclp: monotone-algebra-subject-reduction F A I (rtranclp ( $\sqsubseteq$ ))*  
  ⟨*proof*⟩

**sublocale** *rtranclp: monotone-algebra F A I (rtranclp ( $\sqsubseteq$ ))*  
  + *rtranclp: subject-reduction A rtranclp ( $\sqsubseteq$ )*⟨*proof*⟩

**interpretation** *reflclp-on: monotone-algebra-subject-reduction F A I reflclp-on (dom A) ( $\sqsubseteq$ )*  
  ⟨*proof*⟩

**sublocale** *reflclp-on: monotone-algebra F A I reflclp-on (dom A) ( $\sqsubseteq$ )*  
  + *reflclp-on: subject-reduction A reflclp-on (dom A) ( $\sqsubseteq$ )*⟨*proof*⟩

**interpretation** *rtranclp-on: monotone-algebra-subject-reduction F A I (rtranclp-on (dom A) ( $\sqsubseteq$ ))*⟨*proof*⟩

**sublocale** *rtranclp-on: monotone-algebra F A I (rtranclp-on (dom A) ( $\sqsubseteq$ ))*  
  + *rtranclp-on: subject-reduction A rtranclp-on (dom A) ( $\sqsubseteq$ )*⟨*proof*⟩

**end**

**lemma** (**in** *sorted-algebra-represented*) *monotone-iff-eval-ctx*:  
  **fixes** *less-eq (infix  $\sqsubseteq$  50)*  
  **shows** *monotone-algebra F A I ( $\sqsubseteq$ )  $\longleftrightarrow (\forall C \sigma \tau a b \alpha.$*

$C : \sigma \rightarrow \tau$  in  $\mathcal{C}(F, \emptyset :: \text{unit} \rightarrow -)$   $\rightarrow a : \sigma$  in  $A \rightarrow b : \sigma$  in  $A \rightarrow a \sqsubseteq b \rightarrow$   
 $I\langle I[[C]]_c \alpha; a \rangle \sqsubseteq I\langle I[[C]]_c \alpha; b \rangle$   
 (is ?l  $\leftrightarrow$  ?r)  
 <proof>

## 8.1 Ordered Monotone Algebras

**locale** *reflexive-monotone-algebra* = *reflexive-algebra* + *monotone*  
**begin**

**sublocale** *monotone-algebra* <proof>

**interpretation** *valid: reflexive-monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun valid*  $A I (\sqsubseteq)$   
 $V$  <proof>

**lemmas** *valid-reflexive-monotone-algebra* = *valid.reflexive-monotone-algebra-axioms*

**end**

**locale** *transitive-monotone-algebra* = *transitive-algebra* + *monotone*  
**begin**

**sublocale** *monotone-algebra* <proof>

**interpretation** *valid: transitive-monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun valid*  $A I (\sqsubseteq)$   
 $V$  <proof>

**lemmas** *valid-transitive-monotone-algebra* = *valid.transitive-monotone-algebra-axioms*

**end**

**locale** *quasi-ordered-monotone-algebra* = *quasi-ordered-algebra* + *monotone*  
**begin**

**sublocale** *monotone-algebra* + *reflexive-monotone-algebra* + *transitive-monotone-algebra* <proof>

**interpretation** *valid: quasi-ordered-monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun valid*  $A I (\sqsubseteq)$   
 $V$  <proof>

**lemmas** *valid-quasi-ordered-monotone-algebra* = *valid.quasi-ordered-monotone-algebra-axioms*

**interpretation** *less-eq-dualize* <proof>

**lemma** *dual-quasi-ordered-monotone-algebra:*  
*quasi-ordered-monotone-algebra*  $F A I (\sqsupseteq)$   
 <proof>

**end**

```

locale strict-ordered-monotone-algebra = strict-ordered-algebra + monotone F A I
( $\sqsubseteq$ )
begin

sublocale transitive-monotone-algebra F A I ( $\sqsubseteq$ ) $\langle$ proof $\rangle$ 

end

end

```

## 9 Sorted Rewrite Relations

```

theory Sorted-Rewrite-Relations
  imports Monotone-Algebras
begin

```

```

named-theorems rewriting-simps

```

```

declare relpowp-Suc-I2[trans]

```

An ARS is just a set equipped with a binary relation. We introduce a locale to allow for convenient reasoning on many-step rewriting.

```

locale ars = less-eq-syntax
begin

```

Declaring some trans rules for relpowp:

```

lemma relpowp2[trans]:  $a \sqsubseteq b \implies b \sqsubseteq c \implies ((\sqsubseteq) \overset{\sim}{\sim} \text{Suc} (\text{Suc } 0)) a c$ 
 $\langle$ proof $\rangle$ 

```

```

lemmas relpowp-Suc-I[trans] = relpowp-Suc-I[of - ( $\sqsubseteq$ )]
lemmas relpowp-Suc-I2[trans] = relpowp-Suc-I2[of - ( $\sqsubseteq$ )]

```

```

lemmas relpowp-imp-rtranclp[simp] = relpowp-imp-rtranclp[of - ( $\sqsubseteq$ )]

```

```

lemma relpowp-imp-trancl[simp]:  $((\sqsubseteq) \overset{\sim}{\sim} \text{Suc } n) s t \implies (\sqsubseteq)^{++} s t$ 
 $\langle$ proof $\rangle$ 

```

```

declare relpowp.simps(2)[simp del]

```

```

lemma relpowp-tranclp-trans[trans]:  $((\sqsubseteq) \overset{\sim}{\sim} n) s t \implies (\sqsubseteq)^{++} t u \implies (\sqsubseteq)^{++} s u$ 
 $\langle$ proof $\rangle$ 

```

```

lemma tranclp-relpowp-trans[trans]:  $(\sqsubseteq)^{++} s t \implies ((\sqsubseteq) \overset{\sim}{\sim} n) t u \implies (\sqsubseteq)^{++} s u$ 
 $\langle$ proof $\rangle$ 

```

```

end

```

## 9.1 Closure Under Substitutions

We would like to define closure under substitution so that the source and target variables can have different types, but we cannot make polymorphic assumptions in a locale. Hence, we fix the two types of variables and specify two relations. This leads to a more general notion for a term relation that implies validity in an algebra. When the algebra is the term algebra, this property is closure under substitutions.

**locale** *stable* =  
**fixes**  $V$  **and**  $A$  **and**  $I$  **and** *less-eqT* (**infix**  $\preceq$  50) **and** *less-eq* (**infix**  $\sqsubseteq$  50)  
**assumes** *stable*:  $s \preceq t \implies \alpha :_s V \rightarrow A \implies I[s]\alpha \sqsubseteq I[t]\alpha$

**lemma** *eq-stable*: *stable*  $V A I$  (=) (=)  
 $\langle$ *proof* $\rangle$

**lemma** *Sup-stable*:  
**assumes**  $\forall r \in R. \text{stable } V A I r \text{ le}$   
**shows** *stable*  $V A I (\bigsqcup R) \text{ le}$   
 $\langle$ *proof* $\rangle$

**context** *stable* **begin**

**lemma** *super-stable*:  
**assumes**  $(\sqsubseteq) \leq \text{le}$  **shows** *stable*  $V A I (\preceq) \text{ le}$   
 $\langle$ *proof* $\rangle$

**lemma** *sub-stable*:  
**assumes**  $\text{le} \leq (\preceq)$  **shows** *stable*  $V A I \text{ le} (\sqsubseteq)$   
 $\langle$ *proof* $\rangle$

**lemma** *dual-stable*: *stable*  $V A I (\preceq)^- (\sqsubseteq)^-$   
**and** *symclp-stable*: *stable*  $V A I (\text{symclp } (\preceq)) (\text{symclp } (\sqsubseteq))$   
 $\langle$ *proof* $\rangle$

**lemma** *relcompp-stable*:  
**fixes**  $\text{le}A'$  (**infix**  $\sqsubseteq''$  50) **and**  $\text{le}T'$  (**infix**  $\preceq''$  50)  
**assumes** *stable*  $V A I (\preceq') (\sqsubseteq')$   
**shows** *stable*  $V A I ((\preceq) \text{ OO } (\preceq')) ((\sqsubseteq) \text{ OO } (\sqsubseteq'))$   
 $\langle$ *proof* $\rangle$

**lemma** *relpowp-stable*: *stable*  $V A I ((\preceq) \sim^n) ((\sqsubseteq) \sim^n)$   
 $\langle$ *proof* $\rangle$

**lemma** *rtranclp-stable*: *stable*  $V A I (\text{rtranclp } (\preceq)) (\text{rtranclp } (\sqsubseteq))$   
 $\langle$ *proof* $\rangle$

**lemma** *tranclp-stable*: *stable*  $V A I (\text{tranclp } (\preceq)) (\text{tranclp } (\sqsubseteq))$   
 $\langle$ *proof* $\rangle$

**lemma** *related-imp-valid*:

**assumes**  $s \preceq t$  **shows**  $A:I:(\sqsubseteq) \models V. s \rightsquigarrow t$   
*<proof>*

**end**

**locale** *stable-algebra* = *sorted-algebra* + *stable*  
**begin**

**lemma** *reflclp-on-stable*:

*stable*  $V A I$  (*reflclp-on* (*dom*  $\mathcal{T}(F, V)$ ) ( $\preceq$ )) (*reflclp-on* (*dom*  $A$ ) ( $\sqsubseteq$ ))  
*<proof>*

**end**

**lemma** *stable-mono*:

**assumes**  $le \leq le'$  **shows** *stable*  $V A I leT le \leq$  *stable*  $V A I leT le'$   
*<proof>*

**lemma** *stable-cmono*:

**assumes**  $leT \leq leT'$  **shows** *stable*  $V A I leT' \leq$  *stable*  $V A I leT$   
*<proof>*

Validity is stable under evaluation.

**interpretation** *valid*: *stable*  $V A I$  *valid*  $A I r V r$   
*<proof>*

Closure under substitution is a special instance, where the target is a term algebra.

**locale** *subst-closed-general* = *stable*  $V \mathcal{T}(F, W)$  *Fun* ( $\preceq$ ) ( $\sqsubseteq$ )

**for**  $F$  **and**  $V$  **and**  $W$  **and** *less-eqT* (**infix**  $\preceq$  50) **and** *less-eq* (**infix**  $\sqsubseteq$  50)  
**begin**

**sublocale** *term-target*: *sorted-algebra*  $F \mathcal{T}(F, W)$  *Fun**<proof>*

**sublocale** *stable-algebra*  $F \mathcal{T}(F, W)$  *Fun* *<proof>*

**sublocale** *reflclp-on*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *reflclp-on* (*dom*  $\mathcal{T}(F, V)$ ) ( $\preceq$ ) *reflclp-on* (*dom*  $\mathcal{T}(F, W)$ ) ( $\sqsubseteq$ )  
*<proof>*

**sublocale** *symclp*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *symclp* ( $\preceq$ ) *symclp* ( $\sqsubseteq$ )  
*<proof>*

**sublocale** *rsymclp-on*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *rsymclp-on* (*dom*  $\mathcal{T}(F, V)$ ) ( $\preceq$ ) *rsymclp-on* (*dom*  $\mathcal{T}(F, W)$ ) ( $\sqsubseteq$ )  
*<proof>*

**sublocale** *relpowp*: *stable*  $V \mathcal{T}(F, W)$  *Fun*  $(\preceq) \sim^n (\sqsubseteq) \sim^n$   
 ⟨*proof*⟩

**sublocale** *rtranclp*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *rtranclp*  $(\preceq)$  *rtranclp*  $(\sqsubseteq)$   
 ⟨*proof*⟩

**sublocale** *tranclp*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *tranclp*  $(\preceq)$  *tranclp*  $(\sqsubseteq)$   
 ⟨*proof*⟩

**sublocale** *rtranclp-on*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *rtranclp-on*  $(\text{dom } \mathcal{T}(F, V)) (\preceq)$   
*rtranclp-on*  $(\text{dom } \mathcal{T}(F, W)) (\sqsubseteq)$   
 ⟨*proof*⟩

**sublocale** *rtransymclp-on*: *stable*  $V \mathcal{T}(F, W)$  *Fun* *rtransymclp-on*  $(\text{dom } \mathcal{T}(F, V))$   
 $(\preceq)$  *rtransymclp-on*  $(\text{dom } \mathcal{T}(F, W)) (\sqsubseteq)$   
 ⟨*proof*⟩

**end**

**lemma** *subst-closed-general-iff*:  
*subst-closed-general*  $F V W (\preceq) (\sqsubseteq) \longleftrightarrow$   
 $(\forall s t \vartheta. s \preceq t \longrightarrow \vartheta :_s V \rightarrow \mathcal{T}(F, W) \longrightarrow s \cdot \vartheta \sqsubseteq t \cdot \vartheta)$   
**for** *le1* (**infix**  $\preceq$  50) **and** *le2* (**infix**  $\sqsubseteq$  50)  
 ⟨*proof*⟩

**locale** *subst-closed* = *subst-closed-general* **where** *less-eq* =  $(\preceq)$  **and**  $W = V$   
**begin**

**lemma** *valid-eq-relate*: *valid*  $\mathcal{T}(F, V)$  *Fun*  $(\preceq)$   $V = (\preceq)$   
 ⟨*proof*⟩

**lemma** *reflclp-on-subst-closed*:  
*subst-closed*  $F V (\text{reflclp-on } (\text{dom } \mathcal{T}(F, V)) (\preceq))$   
 ⟨*proof*⟩

**end**

## 9.2 Rewrite Relations

A relation closed under context is where the term algebra is monotone.

**locale** *ctxt-closed* = *monotone*  $F \mathcal{T}(F, V)$  *Fun*  $(\preceq)$  **for**  $F V$  **and** *less-eqT* (**infix**  
 $\preceq$  50)  
**begin**

**sublocale** *term'*: *sorted-algebra*  $F \mathcal{T}(F, V)$  *Fun*⟨*proof*⟩

**sublocale** *monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun*  $(\preceq)$ ⟨*proof*⟩

**sublocale** *reflclp-on: monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun reflclp-on*  $(\text{dom } \mathcal{T}(F, V))$   
 $(\preceq)$ *<proof>*

**sublocale** *symclp: monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun symclp*  $(\preceq)$ *<proof>*

**thm** *ctxt-closed*

**end**

**lemma** *ctxt-closed*  $F V (\preceq) \longleftrightarrow (\forall f \pi s \sigma \rho s \tau \text{ ls } s \text{ t } rs.$   
 $f : \pi s @ \sigma \# \rho s \rightarrow \tau \text{ in } F \rightarrow$   
 $\text{ls} :_l \pi s \text{ in } \mathcal{T}(F, V) \rightarrow s : \sigma \text{ in } \mathcal{T}(F, V) \rightarrow t : \sigma \text{ in } \mathcal{T}(F, V) \rightarrow rs :_l \rho s \text{ in}$   
 $\mathcal{T}(F, V) \rightarrow$   
 $s \preceq t \rightarrow \text{Fun } f (\text{ls} @ s \# rs) \preceq \text{Fun } f (\text{ls} @ t \# rs))$   
**for** *le* (**infix**  $\preceq 50$ )  
*<proof>*

Especially, context-closed relation with subject reduction property maintains monotonicity over transitive closures.

**locale** *ctxt-closed-subject-reduction* =  
 $\text{ctxt-closed} + \text{subject-reduction } \mathcal{T}(F, V) (\preceq)$

**begin**

**sublocale** *monotone-algebra-subject-reduction*  $F \mathcal{T}(F, V)$  *Fun*  $(\preceq)$ *<proof>*

**thm** *tranclp ctxt-closed*

**end**

**locale** *rewrite-relation* =  $\text{ctxt-closed} + \text{subst-closed}$   
**begin**

**lemma** *reflclp-on-rewrite-relation:*  
 $\text{rewrite-relation } F V (\text{reflclp-on } (\text{dom } \mathcal{T}(F, V)) (\preceq))$   
*<proof>*

**end**

**locale** *rewrite-preorder* =  $\text{rewrite-relation} + \text{quasi-ordered-set } \text{dom } \mathcal{T}(F, V) (\preceq)$   
**begin**

**sublocale** *quasi-ordered-monotone-algebra*  $F \mathcal{T}(F, V)$  *Fun*  $(\preceq)$ *<proof>*

**end**

**locale** *sorted-rewrite-relation* =  $\text{ctxt-closed} + \text{subst-closed} + \text{sorted-relation } \mathcal{T}(F, V)$   
 $(\preceq)$   
**begin**

**sublocale** *ctxt-closed-subject-reduction*⟨*proof*⟩

**interpretation** *symclp*: *ctxt-closed-subject-reduction*  $F$   $V$  *symclp*  $(\preceq)$   
⟨*proof*⟩

**sublocale** *symclp*: *monotone*  $F$   $\mathcal{T}(F, V)$  *Fun* *symclp*  $(\preceq)$ ⟨*proof*⟩

**sublocale** *transymclp*: *monotone*  $F$   $\mathcal{T}(F, V)$  *Fun* *tranclp* (*symclp*  $(\preceq)$ )⟨*proof*⟩

**sublocale** *reflclp-on*: *monotone*  $F$   $\mathcal{T}(F, V)$  *Fun* *reflclp-on* (*dom*  $\mathcal{T}(F, V)$ )  $(\preceq)$   
+ *rsymclp-on*: *monotone*  $F$   $\mathcal{T}(F, V)$  *Fun* *rsymclp-on* (*dom*  $\mathcal{T}(F, V)$ )  $(\preceq)$   
+ *rtransymclp-on*: *monotone*  $F$   $\mathcal{T}(F, V)$  *Fun* *rtransymclp-on* (*dom*  $\mathcal{T}(F, V)$ )  $(\preceq)$   
⟨*proof*⟩

**sublocale** *rtranclp-on*: *rewrite-preorder*  $F$   $V$  *rtranclp-on* (*dom*  $\mathcal{T}(F, V)$ )  $(\preceq)$   
⟨*proof*⟩

**end**

**end**

## 10 Sorted Rewriting

**theory** *Sorted-Rewriting*

**imports** *Sorted-Rewrite-Relations*

**begin**

We define a *sorted term rewrite system* as a set of sorted axioms over a signature. To define the rewrite relation over terms, we also specify the sorted set of variables which the rewritten terms may contain.

### 10.1 Root Rewrite Steps

The *root rewrite steps* are the instances of rewrite rules.

**definition** *rootstepp* ( $\{(-\cdot:-\cdot\rightarrow^\varepsilon)\}$  [51,51,51]1000) **where**  
 $(-F:V:R\rightarrow^\varepsilon) s t \equiv \exists(X. l \rightsquigarrow r) \in R. \exists \vartheta :_s X \rightarrow \mathcal{T}(F, V). s = l \cdot \vartheta \wedge t = r \cdot \vartheta$

**interpretation** *rootstepp*: *ars*  $(-F:V:R\rightarrow^\varepsilon)$  **for**  $F$   $V$   $R$ ⟨*proof*⟩

**abbreviation** *rootstep-op* ( $\{((-) / -\cdot:-\cdot\rightarrow^\varepsilon / (-)\}$  [51,51,51,51,51]50)  
**where**  $s -F:V:R\rightarrow^\varepsilon t \equiv (-F:V:R\rightarrow^\varepsilon) s t$

**lemma**

$s -F:V:R\rightarrow^\varepsilon t \equiv \exists(X. l \rightsquigarrow r) \in R. \exists \vartheta :_s X \rightarrow \mathcal{T}(F, V). s = l \cdot \vartheta \wedge t = r \cdot \vartheta$   
⟨*proof*⟩

**abbreviation** *rootstep* ( $\{(-\cdot:-\cdot\rightarrow^\varepsilon)\}$  [51,51,51]1000)

**where**  $\{-F:V:R \rightarrow^\varepsilon\} \equiv \{(s,t). s -F:V:R \rightarrow^\varepsilon t\}$

**lemma** *rootsteppI*:  $(X. l \rightsquigarrow r) \in R \implies \vartheta :_s X \rightarrow \mathcal{T}(F,V) \implies s = l.\vartheta \implies t = r.\vartheta \implies s -F:V:R \rightarrow^\varepsilon t$   
*<proof>*

**lemma** *rootstepp-root*:  $(X. l \rightsquigarrow r) \in R \implies \vartheta :_s X \rightarrow \mathcal{T}(F,V) \implies l.\vartheta -F:V:R \rightarrow^\varepsilon r.\vartheta$   
*<proof>*

**lemma** *rootsteppE*[*consumes 1, case-names root*]:

**assumes**  $s -F:V:R \rightarrow^\varepsilon t$   
**and**  $\bigwedge X l r \vartheta. (X. l \rightsquigarrow r) \in R \implies \vartheta :_s X \rightarrow \mathcal{T}(F,V) \implies s = l.\vartheta \implies t = r.\vartheta \implies \textit{thesis}$   
**shows** *thesis*  
*<proof>*

**lemma** *rootstepp-un*:  $(-F:V:R \cup S \rightarrow^\varepsilon) = (-F:V:R \rightarrow^\varepsilon) \sqcup (-F:V:S \rightarrow^\varepsilon)$   
*<proof>*

**lemma** *rootstep-un*:  $\{-F:V:R \cup S \rightarrow^\varepsilon\} = \{-F:V:R \rightarrow^\varepsilon\} \cup \{-F:V:S \rightarrow^\varepsilon\}$   
*<proof>*

**lemma** *rootstepp-le-fun*:

**assumes**  $FG: F \subseteq_m G$  **and**  $st: s -F:V:R \rightarrow^\varepsilon t$  **shows**  $s -G:V:R \rightarrow^\varepsilon t$   
*<proof>*

**lemma** *rootstepp-mono-fun*:

**assumes**  $FG: F \subseteq_m G$  **shows**  $(-F:V:R \rightarrow^\varepsilon) \leq (-G:V:R \rightarrow^\varepsilon)$   
*<proof>*

**lemma** *rootstepp-le-var*:

**assumes**  $VW: V \subseteq_m W$  **and**  $st: s -F:V:R \rightarrow^\varepsilon t$  **shows**  $s -F:W:R \rightarrow^\varepsilon t$   
*<proof>*

**lemma** *rootstepp-mono-var*:

**assumes**  $VW: V \subseteq_m W$  **shows**  $(-F:V:R \rightarrow^\varepsilon) \leq (-F:W:R \rightarrow^\varepsilon)$   
*<proof>*

Rewrite steps are closed under substitution. Note that here the types of variables are polymorphic.

**interpretation** *rootstepp*: *subst-closed-general*  $F V W (-F:V:R \rightarrow^\varepsilon) (-F:W:R \rightarrow^\varepsilon)$   
*<proof>*

**thm** *rootstepp.stable*

**thm** *rootstepp.tranclp.stable*

**interpretation** *rootstepp*: *subst-closed*  $F V (-F:V:R \rightarrow^\varepsilon)$ *<proof>*

## 10.2 Rewrite Steps

The *rewrite steps* are their closure under sorted contexts.

**inductive** *stepp* ('(-:-:->') [51,51,51]1000) for  $F V R$  where

*root*:  $(-F:V:R\rightarrow) s t$  if  $s -F:V:R\rightarrow^\varepsilon t$   
| *comp*:  $(-F:V:R\rightarrow) (Fun f (ls @ s \# rs)) (Fun f (ls @ t \# rs))$   
if  $f : \pi s @ \sigma \# \rho s \rightarrow \tau$  in  $F$   
**and**  $ls :_l \pi s$  in  $\mathcal{T}(F, V)$  **and**  $s : \sigma$  in  $\mathcal{T}(F, V)$  **and**  $t : \sigma$  in  $\mathcal{T}(F, V)$  **and**  $rs :_l \rho s$   
in  $\mathcal{T}(F, V)$   
**and**  $(-F:V:R\rightarrow) s t$   
**for**  $f \pi s \sigma \rho s \tau ls rs s t$

**hide-fact(open)** *stepp.root stepp.comp*

**declare** *stepp.root[simp]*

**abbreviation** *step-op* (((2-) / -:-:-> / (2-)) [51,51,51,51,51]50)  
**where**  $s -F:V:R\rightarrow t \equiv (-F:V:R\rightarrow) s t$

**abbreviation** *step* ({-:-:->} [51,51,51]1000)  
**where**  $\{-F:V:R\rightarrow\} \equiv \{(s,t). s -F:V:R\rightarrow t\}$

**abbreviation(input)** *dual-stepp* ('(←-:-:-) [51,51,51]1000)  
**where**  $(\leftarrow F:V:R-) \equiv (-F:V:R\rightarrow)^-$

**abbreviation** *dual-step* ({←-:-:-} [51,51,51]1000)  
**where**  $\{\leftarrow F:V:R-\} \equiv \{-F:V:R\rightarrow\}^{-1}$

For reflexive closure, it is convenient to stay within well-sorted terms.

**definition** *step-reflclp* ('(-:-:->=) [51,51,51]1000)  
**where**  $(-F:V:R\rightarrow^=) \equiv \text{reflclp-on } (\text{dom } \mathcal{T}(F, V)) (-F:V:R\rightarrow)$

**abbreviation** *step-reflcl-op* (((2-) / -:-:->= / (2-)) [51,51,51,51,51]50)  
**where**  $s -F:V:R\rightarrow^= t \equiv (-F:V:R\rightarrow^=) s t$

**abbreviation** *step-reflcl* ({-:-:->=} [51,51,51]1000)  
**where**  $\{-F:V:R\rightarrow^=\} \equiv \{(s,t). s -F:V:R\rightarrow^= t\}$

**abbreviation(input)** *dual-step-reflclp* ('(←-:-:-=) [51,51,51]1000)  
**where**  $(\leftarrow F:V:R-^=) \equiv (-F:V:R\rightarrow^=)^-$

**abbreviation(input)** *dual-step-reflcl* ({←-:-:-= } [51,51,51]1000)  
**where**  $\{\leftarrow F:V:R-^=\} \equiv \{-F:V:R\rightarrow^=\}^{-1}$

**abbreviation** *stepp-fold* (((2-) / -:-:->^ / (2-)) [51,51,51,51,1000,51]50)  
**where**  $s -F:V:R\rightarrow^{\wedge n} t \equiv ((-F:V:R\rightarrow)^{\wedge n}) s t$

**abbreviation** *step-trancl-op* (((2-) / -:-:->+ / (2-)) [51,51,51,51,51]50)  
**where**  $s -F:V:R\rightarrow^+ t \equiv (-F:V:R\rightarrow)^{++} s t$

**abbreviation** *step-rtranclp* ( $'(-\cdot\cdot\cdot\rightarrow^*)'$ ) [51,51,51]1000)

where  $(-F:V:R\rightarrow^*) \equiv (-F:V:R\rightarrow^=)^{++}$

**abbreviation** *step-rtrancl-op* ( $((2-)/-\cdot\cdot\cdot\rightarrow^*/(2-))$ ) [51,51,51,51,51]50)

where  $s -F:V:R\rightarrow^* t \equiv (-F:V:R\rightarrow^*) s t$

**abbreviation** *step-rtrancl* ( $\{-\cdot\cdot\cdot\rightarrow^*\}$ ) [51,51,51]1000)

where  $\{-F:V:R\rightarrow^*\} \equiv \{(s,t). s -F:V:R\rightarrow^* t\}$

**abbreviation** *dual-step-rtrancl* ( $\{\leftarrow\cdot\cdot\cdot\rightarrow^*\}$ ) [51,51,51]1000)

where  $\{\leftarrow F:V:R\rightarrow^*\} \equiv \{\leftarrow F:V:R\rightarrow^*\}^{-1}$

**abbreviation** *step-symclp* ( $'(\leftarrow\cdot\cdot\cdot\rightarrow)'$ ) [51,51,51]1000)

where  $(\leftarrow F:V:R\rightarrow) \equiv \text{symclp}(-F:V:R\rightarrow)$

**abbreviation** *step-symcl-op* ( $((2-)/\leftarrow\cdot\cdot\cdot\rightarrow/(2-))$ ) [51,51,51,51,51]50)

where  $s \leftarrow F:V:R\rightarrow t \equiv (\leftarrow F:V:R\rightarrow) s t$

**abbreviation** *step-symcl* ( $\{\leftarrow\cdot\cdot\cdot\rightarrow\}$ ) [51,51,51]1000)

where  $\{\leftarrow F:V:R\rightarrow\} \equiv \{(s,t). s \leftarrow F:V:R\rightarrow t\}$

**abbreviation** *step-rsymclp* ( $'(\leftarrow\cdot\cdot\cdot\rightarrow^=)'$ ) [51,51,51]1000)

where  $(\leftarrow F:V:R\rightarrow^=) \equiv \text{symclp}(-F:V:R\rightarrow^=)$

**abbreviation** *step-rsymcl-op* ( $((2-)/\leftarrow\cdot\cdot\cdot\rightarrow^=/(2-))$ ) [51,51,51,51,51]50)

where  $s \leftarrow F:V:R\rightarrow^= t \equiv (\leftarrow F:V:R\rightarrow^=) s t$

**abbreviation** *step-rtransymclp* ( $'(\leftarrow\cdot\cdot\cdot\rightarrow^*)'$ ) [51,51,51]1000)

where  $(\leftarrow F:V:R\rightarrow^*) \equiv (\leftarrow F:V:R\rightarrow^=)^{++}$

**abbreviation** *step-rtransymcl-op* ( $((2-)/\leftarrow\cdot\cdot\cdot\rightarrow^*/(2-))$ ) [51,51,51,51,51]50)

where  $s \leftarrow F:V:R\rightarrow^* t \equiv (\leftarrow F:V:R\rightarrow^*) s t$

**abbreviation** *step-rtransymcl* ( $\{\leftarrow\cdot\cdot\cdot\rightarrow^*\}$ ) [51,51,51]1000)

where  $\{\leftarrow F:V:R\rightarrow^*\} \equiv \{(s,t). s \leftarrow F:V:R\rightarrow^* t\}$

**abbreviation** *step-rsymcl* ( $\{\leftarrow\cdot\cdot\cdot\rightarrow^=\}$ ) [51,51,51]1000)

where  $\{\leftarrow F:V:R\rightarrow^=\} \equiv \{(s,t). s \leftarrow F:V:R\rightarrow^= t\}$

**interpretation** *stepp: ars*  $(-F:V:R\rightarrow)$  for  $F V R\langle\text{proof}\rangle$

**lemma** *rootstepp-le-step*[rewriting-simps]:  $(-F:V:R\rightarrow^\varepsilon) \leq (-F:V:R\rightarrow) \langle\text{proof}\rangle$

We first state that rewrite step is closed under substitution with heterogeneous variables.

**interpretation** *stepp: subst-closed-general*  $F V W (-F:V:R\rightarrow) (-F:W:R\rightarrow)$

rewrites  $\bigwedge X. \text{reflclp-on}(\text{dom } \mathcal{T}(F,X)) (-F:X:R\rightarrow) \equiv (-F:X:R\rightarrow^=)$

$\langle\text{proof}\rangle$

This gives, e.g., that transitive closures are closed under substitutions.

**thm** *stepp.stable*[of  $F V R s t \vartheta W$ ]

**thm** *stepp.rtranclp-on.stable*

**thm** *stepp.rtransymclp-on.stable*

**interpretation** *stepp*: subst-closed  $F V (-F:V:R\rightarrow)$

rewrites  $\bigwedge X. \text{reflclp-on} (\text{dom } \mathcal{T}(F,X)) (-F:X:R\rightarrow) \equiv (-F:X:R\rightarrow^=)$

*<proof>*

Then we state that the rewrite step is a rewrite relation.

**interpretation** *stepp*: rewrite-relation  $F V (-F:V:R\rightarrow)$

rewrites  $\bigwedge X. \text{reflclp-on} (\text{dom } \mathcal{T}(F,X)) (-F:X:R\rightarrow) \equiv (-F:X:R\rightarrow^=)$

*<proof>*

**thm** *stepp.ctxt-closed*

**thm** *stepp.reflclp-on.ctxt-closed*

**thm** *stepp.symclp.ctxt-closed*

**lemma** *steppI-ctxt-rootstepp*:

assumes  $st: s -F:V:R\rightarrow^\varepsilon t$  and  $C: C : \sigma \rightarrow \tau$  in  $\mathcal{C}(F,V)$

and  $s: s : \sigma$  in  $\mathcal{T}(F,V)$  and  $t: t : \sigma$  in  $\mathcal{T}(F,V)$

and  $u: u = C\langle s \rangle$  and  $v: v = C\langle t \rangle$

shows  $u -F:V:R\rightarrow v$

*<proof>*

**lemmas** *stepp-ctxt-rootstepp = steppI-ctxt-rootstepp*[OF - - - refl refl]

The rewrite step is either the root step or a typed root step under nonempty typed context. Distinguishing the first case is necessary only when the rewrite rules are not well-typed.

**lemma** *stepp-iff-rootstep-or*:

$s -F:V:R\rightarrow t \iff$

$s -F:V:R\rightarrow^\varepsilon t \vee$

$(\exists C \sigma \tau u v. C : \sigma \rightarrow \tau \text{ in } \mathcal{C}(F,V) \wedge u : \sigma \text{ in } \mathcal{T}(F,V) \wedge v : \sigma \text{ in } \mathcal{T}(F,V) \wedge C$

$\neq \square \wedge$

$u -F:V:R\rightarrow^\varepsilon v \wedge s = C\langle u \rangle \wedge t = C\langle v \rangle)$

(is  $?l \iff ?m \vee ?r s t$ )

*<proof>*

**lemma** *steppE-rootstep-or*[consumes 1, case-names root ctxt]:

assumes  $s -F:V:R\rightarrow t$

and  $s -F:V:R\rightarrow^\varepsilon t \implies \text{thesis}$

and  $\bigwedge C \sigma \tau u v. C : \sigma \rightarrow \tau \text{ in } \mathcal{C}(F,V) \implies$

$u : \sigma \text{ in } \mathcal{T}(F,V) \implies v : \sigma \text{ in } \mathcal{T}(F,V) \implies C \neq \square \implies$

$u -F:V:R\rightarrow^\varepsilon v \implies s = C\langle u \rangle \implies t = C\langle v \rangle \implies \text{thesis}$

shows *thesis*

*<proof>*

**lemma** *stepp-un*:  $(-F:V:R \cup S \rightarrow) = (-F:V:R \rightarrow) \sqcup (-F:V:S \rightarrow)$  (is ?l = ?r)  
 ⟨proof⟩

**lemma** *step-un*:  $\{-F:V:R \cup S \rightarrow\} = \{-F:V:R \rightarrow\} \cup \{-F:V:S \rightarrow\}$  ⟨proof⟩

**lemma** *Context-le-fun*:

**assumes**  $FG: F \subseteq_m G$  **and**  $C: C : \sigma \rightarrow \tau$  **in**  $\mathcal{C}(F, V)$  **shows**  $C : \sigma \rightarrow \tau$  **in**  $\mathcal{C}(G, V)$   
 ⟨proof⟩

**lemma** *Context-mono-fun*: **assumes**  $FG: F \subseteq_m G$  **shows**  $\mathcal{C}(F, V) \subseteq_m \mathcal{C}(G, V)$   
 ⟨proof⟩

**lemma** *Context-le-var*:

**assumes**  $VW: V \subseteq_m W$  **and**  $C: C : \sigma \rightarrow \tau$  **in**  $\mathcal{C}(F, V)$  **shows**  $C : \sigma \rightarrow \tau$  **in**  $\mathcal{C}(F, W)$   
 ⟨proof⟩

**lemma** *Context-mono-var*: **assumes**  $VW: V \subseteq_m W$  **shows**  $\mathcal{C}(F, V) \subseteq_m \mathcal{C}(F, W)$   
 ⟨proof⟩

**lemma** *step-le-fun*: **assumes**  $FG: F \subseteq_m G$  **and**  $st: s -F:V:R \rightarrow t$  **shows**  $s -G:V:R \rightarrow t$   
 ⟨proof⟩

**lemma** *step-mono-fun*: **assumes**  $FG: F \subseteq_m G$  **shows**  $(-F:V:R \rightarrow) \leq (-G:V:R \rightarrow)$   
 ⟨proof⟩

**lemma** *step-le-var*: **assumes**  $VW: V \subseteq_m W$  **and**  $st: s -F:V:R \rightarrow t$  **shows**  $s -F:W:R \rightarrow t$   
 ⟨proof⟩

**lemma** *step-mono-var*: **assumes**  $VW: V \subseteq_m W$  **shows**  $(-F:V:R \rightarrow) \leq (-F:W:R \rightarrow)$   
 ⟨proof⟩

### 10.3 Closures

**interpretation** *step-reflclp*: *reflexive dom*  $\mathcal{T}(F, V)$   $(-F:V:R \rightarrow^=)$   
 ⟨proof⟩

**lemma** *step-reflclp-iff*:  $s -F:V:R \rightarrow^= t \iff s = t \wedge s \in \text{dom } \mathcal{T}(F, V) \vee s -F:V:R \rightarrow t$   
 ⟨proof⟩

**lemma** *step-reflclpE*[*consumes 1, case-names refl step*]:

**assumes**  $s -F:V:R \rightarrow^= t$   
**and**  $\bigwedge \sigma. s : \sigma$  **in**  $\mathcal{T}(F, V) \implies s = t \implies \text{thesis}$   
**and**  $s -F:V:R \rightarrow t \implies \text{thesis}$   
**shows** *thesis*

$\langle proof \rangle$

**lemma** *step-reflclp-refl[simp]*:  $s : \sigma$  in  $\mathcal{T}(F, V) \implies s -F:V:R \rightarrow^= s$   
**and** *step-reflclp-step[simp]*:  $s -F:V:R \rightarrow t \implies s -F:V:R \rightarrow^= t$   
 $\langle proof \rangle$

**lemma** *stepp-le-reflcl[rewriting-simps]*:  $(-F:V:R \rightarrow) \leq (-F:V:R \rightarrow^=)$   $\langle proof \rangle$

**interpretation** *step-reflclp*: reflexive-monotone-algebra  $F \mathcal{T}(F, V) Fun (-F:V:R \rightarrow^=)$   
 $\langle proof \rangle$

**interpretation** *step-tranclp*: transitive-algebra  $F \mathcal{T}(F, V) Fun (-F:V:R \rightarrow)^{++}$   
 $\langle proof \rangle$

**lemma** *step-trancl[simp]*:  $(s, t) \in \{-F:V:R \rightarrow\}^+ \iff s -F:V:R \rightarrow^+ t$   
 $\langle proof \rangle$

**lemma** *step-rtranclp-eq*:  $(-F:V:R \rightarrow^*) = rtranclp-on (dom \mathcal{T}(F, V)) (-F:V:R \rightarrow)$   
 $\langle proof \rangle$

**interpretation** *step-rtranclp*: quasi-ordered-algebra  $F \mathcal{T}(F, V) Fun (-F:V:R \rightarrow^*)$   
 $\langle proof \rangle$

**lemma** *step-rtranclp-trancl[rewriting-simps]*:  $s -F:V:R \rightarrow^+ t \implies s -F:V:R \rightarrow^* t$   
 $\langle proof \rangle$

**lemma** *step-tranclp-step[rewriting-simps]*:  $s -F:V:R \rightarrow t \implies s -F:V:R \rightarrow^+ t$   $\langle proof \rangle$

**lemma** *[rewriting-simps]*:  
**shows** *step-tranclp-le-rtrancl*:  $(-F:V:R \rightarrow)^{++} \leq (-F:V:R \rightarrow^*)$   
**and** *stepp-le-rtrancl*:  $(-F:V:R \rightarrow) \leq (-F:V:R \rightarrow^*)$   
 $\langle proof \rangle$

**lemma** *step-rtranclp-eq-reflclp*:  
 $(-F:V:R \rightarrow^*) = reflclp-on (dom \mathcal{T}(F, V)) (-F:V:R \rightarrow)^{++}$   
 $\langle proof \rangle$

**lemma** *step-rtranclpE[consumes 1, case-names refl trancl]*:  
**assumes**  $s -F:V:R \rightarrow^* t$   
**and**  $\bigwedge \sigma. s : \sigma$  in  $\mathcal{T}(F, V) \implies s = t \implies thesis$   
**and**  $s -F:V:R \rightarrow^+ t \implies thesis$   
**shows** *thesis*  $\langle proof \rangle$

**lemma** *stepp-transs[trans]*:  
 $s -F:V:R \rightarrow^= t \implies t -F:V:R \rightarrow u \implies s -F:V:R \rightarrow^+ u$   
 $s -F:V:R \rightarrow t \implies t -F:V:R \rightarrow^= u \implies s -F:V:R \rightarrow^+ u$   
 $s -F:V:R \rightarrow^= t \implies t -F:V:R \rightarrow^= u \implies s -F:V:R \rightarrow^* u$   
 $s -F:V:R \rightarrow^* t \implies t -F:V:R \rightarrow u \implies s -F:V:R \rightarrow^+ u$

$$\begin{aligned}
s -F:V:R \rightarrow t &\Longrightarrow t -F:V:R \rightarrow^* u \Longrightarrow s -F:V:R \rightarrow^+ u \\
s -F:V:R \rightarrow^+ t &\Longrightarrow t -F:V:R \rightarrow^= u \Longrightarrow s -F:V:R \rightarrow^+ u \\
s -F:V:R \rightarrow^= t &\Longrightarrow t -F:V:R \rightarrow^+ u \Longrightarrow s -F:V:R \rightarrow^+ u \\
s -F:V:R \rightarrow^+ t &\Longrightarrow t -F:V:R \rightarrow^* u \Longrightarrow s -F:V:R \rightarrow^+ u \\
s -F:V:R \rightarrow^* t &\Longrightarrow t -F:V:R \rightarrow^+ u \Longrightarrow s -F:V:R \rightarrow^+ u \\
\langle \text{proof} \rangle
\end{aligned}$$

**lemma** *step-rsymclp-iff*:  $s \leftarrow F:V:R \rightarrow^= t \longleftrightarrow (s = t \wedge s \in \text{dom } \mathcal{T}(F,V)) \vee s \leftarrow F:V:R \rightarrow t$   
 $\langle \text{proof} \rangle$

**lemma** *step-rsymclpE*[*consumes 1, case-names refl sym*]:  
**assumes**  $s \leftarrow F:V:R \rightarrow^= t$   
**and**  $\bigwedge \sigma. s : \sigma \text{ in } \mathcal{T}(F,V) \Longrightarrow s = t \Longrightarrow \text{thesis}$   
**and**  $s \leftarrow F:V:R \rightarrow t \Longrightarrow \text{thesis}$   
**shows** *thesis*  
 $\langle \text{proof} \rangle$

**lemma** [*rewriting-simps*]:  
**shows** *step-rsymclp-refl*:  $s : \sigma \text{ in } \mathcal{T}(F,V) \Longrightarrow s \leftarrow F:V:R \rightarrow^= s$   
**and** *step-rsymclp-symcl*:  $s \leftarrow F:V:R \rightarrow t \Longrightarrow s \leftarrow F:V:R \rightarrow^= t$   
 $\langle \text{proof} \rangle$

**lemma** [*rewriting-simps*]:  
**shows** *step-symclp-le-rsymcl*:  $(\leftarrow F:V:R \rightarrow) \leq (\leftarrow F:V:R \rightarrow^=)$   
**and** *stepp-le-rsymcl*:  $(-F:V:R \rightarrow) \leq (\leftarrow F:V:R \rightarrow^=)$   
**and** *stepp-dual-le-rsymcl*:  $(\leftarrow F:V:R -) \leq (\leftarrow F:V:R \rightarrow^=)$   
 $\langle \text{proof} \rangle$

**interpretation** *step-rsymclp-on*: *reflexive dom*  $\mathcal{T}(F,V)$   $(\leftarrow F:V:R \rightarrow^=)$   
 $\langle \text{proof} \rangle$

**lemma** *step-tranclp-le-rtransymcl*[*rewriting-simps*]:  $(-F:V:R \rightarrow)^{++} \leq (\leftarrow F:V:R \rightarrow^*)$   
 $\langle \text{proof} \rangle$

**lemma** *step-rtransymclp-trancl*[*rewriting-simps*]:  $s -F:V:R \rightarrow^+ t \Longrightarrow s \leftarrow F:V:R \rightarrow^* t$   
 $\langle \text{proof} \rangle$

**lemma** *step-dual-rtransymclp-trancl*[*rewriting-simps*]:  $t -F:V:R \rightarrow^+ s \Longrightarrow s \leftarrow F:V:R \rightarrow^* t$   
 $\langle \text{proof} \rangle$

**lemma** *step-rtransymclp-rtrancl*[*rewriting-simps*]:  $s -F:V:R \rightarrow^* t \Longrightarrow s \leftarrow F:V:R \rightarrow^* t$   
 $\langle \text{proof} \rangle$

**lemma** *step-dual-rtransymclp-rtrancl*[*rewriting-simps*]:  $t -F:V:R \rightarrow^* s \Longrightarrow s \leftarrow F:V:R \rightarrow^* t$   
 $\langle \text{proof} \rangle$

*<proof>*

**lemma** *step-rtransymclp-transs*[*trans*]:

$s \leftarrow F:V:R \rightarrow^* t \implies t \leftarrow F:V:R \rightarrow u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^* t \implies u \leftarrow F:V:R \rightarrow t \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow t \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $t \leftarrow F:V:R \rightarrow s \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^* t \implies t \leftarrow F:V:R \rightarrow^= u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^* t \implies u \leftarrow F:V:R \rightarrow^= t \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^= t \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $t \leftarrow F:V:R \rightarrow^= s \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^* t \implies t \leftarrow F:V:R \rightarrow^+ u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^* t \implies u \leftarrow F:V:R \rightarrow^+ t \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^+ t \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $t \leftarrow F:V:R \rightarrow^+ s \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
 $s \leftarrow F:V:R \rightarrow^* t \implies t \leftarrow F:V:R \rightarrow^* u \implies s \leftarrow F:V:R \rightarrow^* u$   
*<proof>*

## 10.4 Sorted Rewrite Systems

Now we demand that all rewrite rules respect sorts.

**locale** *sorted-trs* =

**fixes**  $F :: ('f, 's) \text{ sig}$  **and**  $R :: ('f, 'x, 's) \text{ axiom set}$

**assumes** *axiom-typed*:

$(X. l \rightsquigarrow r) \in R \implies \exists \sigma. l : \sigma \text{ in } \mathcal{T}(F, X) \wedge r : \sigma \text{ in } \mathcal{T}(F, X)$

**begin**

**lemmas** *axiom-hastypeE* = *axiom-typed*[*THEN exE, unfolded conj-imp-eq-imp-imp*]

**lemma** *axiom-hastype-iff*:  $(X. l \rightsquigarrow r) \in R \implies l : \sigma \text{ in } \mathcal{T}(F, X) \longleftrightarrow r : \sigma \text{ in } \mathcal{T}(F, X)$

*<proof>*

**lemma**

**assumes**  $(X. l \rightsquigarrow r) \in R$

**shows** *lhs-in-dom*:  $l \in \text{dom } \mathcal{T}(F, X)$  **and** *rhs-in-dom*:  $r \in \text{dom } \mathcal{T}(F, X)$

*<proof>*

**lemma** *rootstepp-typed*:  $s \leftarrow F:V:R \rightarrow^\varepsilon t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$

*<proof>*

**sublocale** *rootstepp*: *relation-on dom*  $\mathcal{T}(F, V)$   $(\leftarrow F:V:R \rightarrow^\varepsilon)$

*<proof>*

**lemma** *stepp-typed*:  $s \leftarrow F:V:R \rightarrow t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$

*<proof>*

**sublocale** *stepp*: *relation-on dom*  $\mathcal{T}(F, V)$   $(\leftarrow F:V:R \rightarrow)$

*<proof>*

**sublocale** *step-reflclp*: relation-on dom  $\mathcal{T}(F, V)$  ( $-F:V:R \rightarrow^=$ )  
 ⟨proof⟩

**sublocale** *step-tranclp*: relation-on dom  $\mathcal{T}(F, V)$  *tranclp* ( $-F:V:R \rightarrow$ )  
 ⟨proof⟩

**sublocale** *step-rtranclp*: relation-on dom  $\mathcal{T}(F, V)$  ( $-F:V:R \rightarrow^*$ )  
 ⟨proof⟩

**sublocale** *step-symclp*: relation-on dom  $\mathcal{T}(F, V)$  ( $\leftarrow F:V:R \rightarrow$ )  
 ⟨proof⟩

**sublocale** *step-rsymclp*: relation-on dom  $\mathcal{T}(F, V)$  ( $\leftarrow F:V:R \rightarrow^=$ )  
 ⟨proof⟩

**sublocale** *step-rtransymclp*: relation-on dom  $\mathcal{T}(F, V)$  ( $\leftarrow F:V:R \rightarrow^*$ )  
 ⟨proof⟩

**lemma** *step-reflclp-typed*:  $s -F:V:R \rightarrow^= t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma** *step-tranclp-typed*:  $s -F:V:R \rightarrow^+ t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma** *step-rtranclp-typed*:  $s -F:V:R \rightarrow^* t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma** *step-symclp-typed*:  $s \leftarrow F:V:R \rightarrow t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma** *step-rsymclp-typed*:  $s \leftarrow F:V:R \rightarrow^= t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma** *step-rtransymclp-typed*:  $s \leftarrow F:V:R \rightarrow^* t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma** *stepp-iff-ctxt-rule*:

$s -F:V:R \rightarrow t \iff$   
 $(\exists C \sigma \tau X l r \vartheta.$   
 $C : \sigma \rightarrow \tau \text{ in } \mathcal{C}(F, V) \wedge (X. l \rightsquigarrow r) \in R \wedge \vartheta :_s X \rightarrow \mathcal{T}(F, V) \wedge l : \sigma \text{ in } \mathcal{T}(F, X)$   
 $\wedge$   
 $s = C(l \cdot \vartheta) \wedge t = C(r \cdot \vartheta))$  (**is**  $?l \iff (\exists C \sigma. ?r C \sigma)$ )

*<proof>*

**lemma** *stepp-iff-ctxt-rootstepp*:

$s -F:V:R \rightarrow t \iff$   
 $(\exists C \sigma \tau u v.$   
 $C : \sigma \rightarrow \tau \text{ in } \mathcal{C}(F, V) \wedge u : \sigma \text{ in } \mathcal{T}(F, V) \wedge s = C\langle u \rangle \wedge t = C\langle v \rangle \wedge u -F:V:R \rightarrow^\varepsilon$   
 $v)$   
*<proof>*

**lemma** *steppE-ctxt-rule*:

**assumes**  $s -F:V:R \rightarrow t$   
**and**  $\bigwedge C \sigma \tau X l r \vartheta.$   
 $C : \sigma \rightarrow \tau \text{ in } \mathcal{C}(F, V) \implies (X. l \rightsquigarrow r) \in R \wedge \vartheta :_s X \rightarrow \mathcal{T}(F, V) \implies l : \sigma \text{ in}$   
 $\mathcal{T}(F, X) \implies$   
 $s = C\langle l \cdot \vartheta \rangle \implies t = C\langle r \cdot \vartheta \rangle \implies \textit{thesis}$   
**shows** *thesis*  
*<proof>*

**sublocale** *rootstepp: sorted-relation*  $\mathcal{T}(F, V) (-F:V:R \rightarrow^\varepsilon)$   
*<proof>*

**sublocale** *stepp: sorted-rewrite-relation*  $F V (-F:V:R \rightarrow)$   
**rewrites**  $\bigwedge X. \text{reflclp-on}(\text{dom } \mathcal{T}(F, X)) (-F:X:R \rightarrow) \equiv (-F:X:R \rightarrow^=)$   
*<proof>*

**thm** *stepp.rtranclp-on ctxt-closed*

**sublocale** *step-symclp: sorted-relation*  $\mathcal{T}(F, V) (\leftarrow F:V:R \rightarrow)$   
*<proof>*

**sublocale** *step-reflclp: sorted-relation*  $\mathcal{T}(F, V) (-F:V:R \rightarrow^=)$   
*<proof>*

**sublocale** *step-rsymclp: sorted-relation*  $\mathcal{T}(F, V) (\leftarrow F:V:R \rightarrow^=)$   
*<proof>*

**lemma** *arg-stepp-imp-ex-stepp*:

**assumes** *fss*:  $\text{Fun } f \text{ ss} \in \text{dom } \mathcal{T}(F, V)$  **and**  $s : s \in \text{set } \text{ss}$   
**and**  $st$ :  $s -F:V:R \rightarrow t$  **shows**  $\exists t'. \text{Fun } f \text{ ss} -F:V:R \rightarrow t'$   
*<proof>*

**lemma** *step-rtranclp-induct[consumes 1, case-names refl step]*:

**assumes**  $st$ :  $s -F:V:R \rightarrow^* t$   
**and** *Refl*:  $\bigwedge \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \implies P s$   
**and** *Step*:  $\bigwedge t u. s -F:V:R \rightarrow^* t \implies t -F:V:R \rightarrow u \implies P t \implies P u$   
**shows**  $P t$   
*<proof>*

**lemma** *step-rtranclp-induct-dual[consumes 1, case-names refl step]*:

**assumes**  $st: s \leftarrow F:V:R \rightarrow^* t$   
**and**  $Refl: \bigwedge \sigma. t : \sigma \text{ in } \mathcal{T}(F, V) \implies P t$   
**and**  $Step: \bigwedge s u. s \leftarrow F:V:R \rightarrow u \implies u \leftarrow F:V:R \rightarrow^* t \implies P u \implies P s$   
**shows**  $P s$   
 $\langle proof \rangle$

**lemma** *step-rtransymclp-induct*[*consumes 1, case-names refl step*]:

**assumes**  $st: s \leftarrow F:V:R \rightarrow^* t$   
**and**  $Refl: \bigwedge \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \implies P s$   
**and**  $Step: \bigwedge t u. s \leftarrow F:V:R \rightarrow^* t \implies t \leftarrow F:V:R \rightarrow u \implies P t \implies P u$   
**shows**  $P t$   
 $\langle proof \rangle$

**sublocale** *step-tranclp*: *transitive-monotone-algebra*  $F \mathcal{T}(F, V) Fun (-F:V:R \rightarrow)^{++} \langle proof \rangle$

**end**

## 10.5 Models of Rewrite Systems

The term algebra coupled with rewrite steps is a model of the TRS.

**lemma** *rootstepp-models*:  $\mathcal{T}(F, V):Fun:(-F:V:R \rightarrow^\varepsilon) \models unconditional \text{ ' } R$   
 $\langle proof \rangle$

**lemma** *stepp-models*:  $\mathcal{T}(F, V):Fun:(-F:V:R \rightarrow) \models unconditional \text{ ' } R$   
 $\langle proof \rangle$

**lemma** *step-tranclp-models*:  $\mathcal{T}(F, V):Fun:(-F:V:R \rightarrow)^{++} \models unconditional \text{ ' } R$   
**and** *step-reflclp-models*:  $\mathcal{T}(F, V):Fun:(-F:V:R \rightarrow^\varepsilon) \models unconditional \text{ ' } R$   
**and** *step-rtranclp-models*:  $\mathcal{T}(F, V):Fun:(-F:V:R \rightarrow^*) \models unconditional \text{ ' } R$   
**and** *step-symclp-models*:  $\mathcal{T}(F, V):Fun:(\leftarrow F:V:R \rightarrow) \models unconditional \text{ ' } R$   
**and** *step-rsymclp-models*:  $\mathcal{T}(F, V):Fun:(\leftarrow F:V:R \rightarrow^\varepsilon) \models unconditional \text{ ' } R$   
**and** *step-rtransymclp-models*:  $\mathcal{T}(F, V):Fun:(\leftarrow F:V:R \rightarrow^*) \models unconditional \text{ ' } R$   
 $\langle proof \rangle$

Moreover, root steps are valid in any models.

**context** *sorted-algebra* **begin**

**context**

**fixes** *less-eq* (**infix**  $\sqsubseteq$  50) **and**  $R$   
**assumes**  $models: A:I:(\sqsubseteq) \models unconditional \text{ ' } R$   
**begin**

**lemma** *rootstepp-imp-valid*:

**assumes**  $st: s \leftarrow F:V:R \rightarrow^\varepsilon t$   
**shows**  $A:I:(\sqsubseteq) \models V. s \rightsquigarrow t$   
 $\langle proof \rangle$

**lemma** *rootstepp-le-valid*:  $(-F:V:R \rightarrow^\varepsilon) \leq valid A I (\sqsubseteq) V$   
 $\langle proof \rangle$

**end**

**end**

Therefore, a root step implies validity in the term algebra.

**lemmas** *rootstepp-imp-valid-term = term.rootstepp-imp-valid*[OF *rootstepp-models*]

Any relation closed under substitutions contains the root steps if it models the TRS.

**lemma** (in *subst-closed*) *rootstepp-le*:

$\mathcal{T}(F, V):Fun:(\preceq) \models \text{unconditional } \ulcorner R \implies (-F:V:R \rightarrow^\varepsilon) \leq (\preceq)$   
*<proof>*

Therefore, the root steps is the least term model which is closed under substitution.

**theorem** *rootstepp-eq-least*:

$(-F:V:R \rightarrow^\varepsilon) = (LEAST\ r.\ \text{subst-closed } F\ V\ r \wedge (\mathcal{T}(F, V):Fun:r \models \text{unconditional } \ulcorner R))$   
*<proof>*

The rewrite steps are valid in all monotone models.

**context** *monotone-algebra* **begin**

**context** **fixes** *R*

**assumes** *models*:  $A:I:(\sqsubseteq) \models \text{unconditional } \ulcorner R$

**begin**

**interpretation** *valid*: *monotone-algebra* *F*  $\mathcal{T}(F, V)$  *Fun* *valid* *A* *I*  $(\sqsubseteq)$  *V**<proof>*

**lemma** *stepp-imp-valid*:  $s -F:V:R \rightarrow t \implies A:I:(\sqsubseteq) \models V.\ s \rightsquigarrow t$   
*<proof>*

**lemma** *stepp-le-valid*:  $(-F:V:R \rightarrow) \leq \text{valid } A\ I\ (\sqsubseteq)\ V$   
*<proof>*

**end**

**end**

In particular, any rewrite relation that models a TRS contains the rewrite step.

**lemma** (in *rewrite-relation*) *stepp-le*:

$\mathcal{T}(F, V):Fun:(\preceq) \models \text{unconditional } \ulcorner R \implies (-F:V:R \rightarrow) \leq (\preceq)$   
*<proof>*

Therefore, the rewrite step is the least rewrite relation that models the TRS.

**theorem** *stepp-eq-least*:

$(-F:V:R\rightarrow) = (LEAST\ r.\ rewrite\ relation\ F\ V\ r \wedge (\mathcal{T}(F,V):Fun:r \models unconditional\ 'R))$   
 $\langle proof \rangle$

Similar results hold for rewrite preorders.

**context** *reflexive-monotone-algebra* **begin**

**context** **fixes**  $R$

**assumes** *models*:  $A:I:(\sqsubseteq) \models unconditional\ 'R$   
**begin**

**lemma** *step-reflclp-imp-valid*:

$s\ -F:V:R\rightarrow^= t \implies A:I:(\sqsubseteq) \models V.\ s \rightsquigarrow t$   
 $\langle proof \rangle$

**lemma** *step-reflclp-le-valid*:

$(-F:V:R\rightarrow^=) \leq valid\ A\ I\ (\sqsubseteq)\ V$   
 $\langle proof \rangle$

**end**

**end**

For transitivity, rewrite rule must relate a sorted term to a sorted term. Sorts need not be the same but for simplicity we assume *sorted-trs* assumption.

**context** *transitive-monotone-algebra* **begin**

**context** **fixes**  $R$

**assumes**  $R$ : *sorted-trs*  $F\ R$  **and** *models*:  $A:I:(\sqsubseteq) \models unconditional\ 'R$   
**begin**

**interpretation**  $R$ : *sorted-trs*  $\langle proof \rangle$

**lemma** *step-tranclp-imp-valid*:

**assumes** *st*:  $s\ -F:V:R\rightarrow^+ t$   
**shows**  $A:I:(\sqsubseteq) \models V.\ s \rightsquigarrow t$   
 $\langle proof \rangle$

**lemma** *step-tranclp-le-valid*:

$(-F:V:R\rightarrow)^{++} \leq valid\ A\ I\ (\sqsubseteq)\ V$   
 $\langle proof \rangle$

**end**

**end**

**context** *quasi-ordered-monotone-algebra* **begin**

**context** **fixes**  $R$

**assumes**  $R$ : sorted-trs  $F R$  **and** models:  $A:I:(\square) \models$  unconditional ‘  $R$   
**begin**

**lemma** *step-rtranclp-imp-valid*:  
 $s -F:V:R \rightarrow^* t \implies A:I:(\square) \models V. s \rightsquigarrow t$   
*<proof>*

**lemma** *step-rtranclp-le-valid*:  $(-F:V:R \rightarrow^*) \leq$  valid  $A I (\square) V$   
*<proof>*

**end**

**end**

**lemma** (in *rewrite-preorder*) *step-rtranclp-le*:  
 $\text{sorted-trs } F R \implies \mathcal{T}(F, V):\text{Fun}:(\preceq) \models$  unconditional ‘  $R \implies (-F:V:R \rightarrow^*) \leq$   
 $(\preceq)$   
*<proof>*

Finally, the rewrite relation is the least rewrite preorder.

**theorem** (in *sorted-trs*) *step-rtranclp-eq-least*:  
 $(-F:V:R \rightarrow^*) = (\text{LEAST } r. \text{rewrite-preorder } F V r \wedge (\mathcal{T}(F, V):\text{Fun}:r \models \text{unconditional ' } R))$   
*<proof>*

**end**

## 11 Conditional Rewriting

**theory** *Conditional-Rewriting*  
**imports** *Sorted-Rewrite-Relations*  
**begin**

Here we define sorted conditional rewriting. As conditional rewrite rules we just use inference rules.

It is more convenient to define the full rewrite step inductively, as satisfaction of conditions involves full rewrite steps. Afterwards we define the root rewrite step as a subset of the full rewrite step.

**inductive** *cstepp* ( $'(=::\rightarrow)'$  [51,51,51]1000) **for**  $F V R$  **where**  
*root*:  $(=F:V:R \Rightarrow) (l.\vartheta) (r.\vartheta)$   
**if**  $(X. l \rightsquigarrow r \Leftarrow cs) \in R$  **and**  $\vartheta :_s X \rightarrow \mathcal{T}(F, V)$   
**and**  $\forall (s \rightsquigarrow t) \in \text{set } cs. (=F:V:R \Rightarrow)^{**} (s.\vartheta) (t.\vartheta)$   
**for**  $X l r cs \vartheta$   
| *comp*:  $(=F:V:R \Rightarrow) (\text{Fun } f (ls @ s \# rs)) (\text{Fun } f (ls @ t \# rs))$   
**if**  $f : \pi s @ \sigma \# \varrho s \rightarrow \tau$  in  $F$   
**and**  $ls :_l \pi s$  in  $\mathcal{T}(F, V)$   
**and**  $s : \sigma$  in  $\mathcal{T}(F, V)$   
**and**  $t : \sigma$  in  $\mathcal{T}(F, V)$

**and**  $rs :_l qs$  in  $\mathcal{T}(F, V)$   
**and**  $(=F:V:R\Rightarrow) s t$   
**for**  $f \pi s \sigma qs \tau ls rs s t$

**hide-fact**(open) *cstepp.root cstepp.comp*

**abbreviation** *cstepp-op* (((2-)/ =-:.-:→ / (2-)) [51,51,51,51,51]50) **where**  
 $s =F:V:R\Rightarrow t \equiv (=F:V:R\Rightarrow) s t$

**abbreviation** *cstep* ({=-:.-:→}[51,51,51]1000) **where**  
 $\{=F:V:R\Rightarrow\} \equiv \{(s,t). s =F:V:R\Rightarrow t\}$

**definition** *cstep-reflclp* ('(=-:.-:→=') [51,51,51]1000) **where**  
 $(=F:V:R\Rightarrow^=) \equiv \text{reflclp-on } (\text{dom } \mathcal{T}(F, V)) (=F:V:R\Rightarrow)$

**abbreviation** *cstep-reflcl-op* (((2-)/ =-:.-:→= / (2-)) [51,51,51,51,51]50) **where**  
 $s =F:V:R\Rightarrow^= t \equiv (=F:V:R\Rightarrow^=) s t$

**abbreviation** *cstep-reflcl* ({=-:.-:→=} [51,51,51]1000)  
**where**  $\{=F:V:R\Rightarrow^=\} \equiv \{(s,t). s =F:V:R\Rightarrow^= t\}$

**abbreviation** (*input*) *dual-cstepp* ('(←-:.-:→=') [51,51,51]1000)  
**where**  $(\leftarrow F:V:R\Rightarrow) \equiv (=F:V:R\Rightarrow)^-$

**abbreviation** *cstep-tranclp* ('(=-:.-:→+)' [51,51,51]1000) **where**  
 $(=F:V:R\Rightarrow^+) \equiv \text{tranclp } (=F:V:R\Rightarrow)$

**abbreviation** *cstep-trancl-op* (((2-)/ =-:.-:→+ / (2-)) [51,51,51,51,51]50)  
**where**  $s =F:V:R\Rightarrow^+ t \equiv (=F:V:R\Rightarrow^+) s t$

**abbreviation** *cstep-rtranclp* ('(=-:.-:→\*)' [51,51,51]1000) **where**  
 $(=F:V:R\Rightarrow^*) \equiv \text{tranclp } (=F:V:R\Rightarrow^=)$

**abbreviation** *cstep-rtrancl-op* (((2-)/ =-:.-:→\* / (2-)) [51,51,51,51,51]50)  
**where**  $s =F:V:R\Rightarrow^* t \equiv (=F:V:R\Rightarrow^*) s t$

**abbreviation** *cstep-rtrancl* ({=-:.-:→\*}[51,51,51]1000)  
**where**  $\{=F:V:R\Rightarrow^*\} \equiv \{(s,t). s =F:V:R\Rightarrow^* t\}$

**abbreviation** *cstep-symclp* ('(←-:.-:→)' [51,51,51]1000)  
**where**  $(\leftarrow F:V:R\Rightarrow) \equiv \text{symclp } (=F:V:R\Rightarrow)$

**abbreviation** *cstep-symcl-op* (((2-)/ ←-:.-:→ / (2-)) [51,51,51,51,51]50)  
**where**  $s \leftarrow F:V:R\Rightarrow t \equiv (\leftarrow F:V:R\Rightarrow) s t$

**abbreviation** *cstep-symcl* ({←-:.-:→}[51,51,51]1000)  
**where**  $\{\leftarrow F:V:R\Rightarrow\} \equiv \{(s,t). s \leftarrow F:V:R\Rightarrow t\}$

**abbreviation** *cstep-rsymclp* ('(←-:.-:→=') [51,51,51]1000) **where**

$(\Leftarrow F:V:R \Rightarrow^=) \equiv \text{symclp } (=F:V:R \Rightarrow^=)$

**abbreviation** *cstep-rsymcl-op*  $((2-)/ \Leftarrow -: \Rightarrow^= / (2-)) [51,51,51,51,51]50$  **where**  
 $s \Leftarrow F:V:R \Rightarrow^= t \equiv (\Leftarrow F:V:R \Rightarrow^=) s t$

**abbreviation** *cstep-rsymcl*  $(\{\Leftarrow -: \Rightarrow^=\}) [51,51,51]1000$   
**where**  $\{\Leftarrow F:V:R \Rightarrow^=\} \equiv \{(s,t). s \Leftarrow F:V:R \Rightarrow^= t\}$

**abbreviation** *cstep-rtransymclp*  $(\prime(\Leftarrow -: \Rightarrow^*) [51,51,51]1000)$   
**where**  $(\Leftarrow F:V:R \Rightarrow^*) \equiv (\Leftarrow F:V:R \Rightarrow^=)^{++}$

**abbreviation** *cstep-rtransymcl-op*  $((2-)/ \Leftarrow -: \Rightarrow^* / (2-)) [51,51,51,51,51]50$   
**where**  $s \Leftarrow F:V:R \Rightarrow^* t \equiv (\Leftarrow F:V:R \Rightarrow^*) s t$

**interpretation** *ars*  $(=F:V:R \Rightarrow)$  **for**  $F V R(\text{proof})$

**lemma** *cstepp-induct*[*consumes 1, case-names root comp*]:

**fixes**  $P$  (**infix**  $\sqsubseteq$  50)

**assumes**  $st: s =F:V:R \Rightarrow t$

**and** *root*:  $\bigwedge X l r cs \vartheta.$

$(X. l \rightsquigarrow r \Leftarrow cs) \in R \implies$

$\vartheta :_s X \rightarrow \mathcal{T}(F, V) \implies$

$\forall s t. (s \rightsquigarrow t) \in \text{set } cs \longrightarrow ((=F:V:R \Rightarrow) \sqcap (\sqsubseteq))^* (s \cdot \vartheta) (t \cdot \vartheta) \implies$

$l \cdot \vartheta \sqsubseteq r \cdot \vartheta$

**and** *comp*:  $\bigwedge f \pi s \sigma \varrho s \tau ls rs s t.$

$f : \pi s @ \sigma \# \varrho s \rightarrow \tau$  **in**  $F \implies$

$ls :_l \pi s$  **in**  $\mathcal{T}(F, V) \implies$

$s : \sigma$  **in**  $\mathcal{T}(F, V) \implies$

$t : \sigma$  **in**  $\mathcal{T}(F, V) \implies$

$rs :_l \varrho s$  **in**  $\mathcal{T}(F, V) \implies$

$s =F:V:R \Rightarrow t \implies$

$s \sqsubseteq t \implies$

$\text{Fun } f (ls @ s \# rs) \sqsubseteq \text{Fun } f (ls @ t \# rs)$

**shows**  $s \sqsubseteq t$

*<proof>*

Conditional rewrite step is a rewrite relation.

**interpretation** *cstepp*: *subst-closed-general*  $F V W (=F:V:R \Rightarrow) (=F:W:R \Rightarrow)$

**rewrites**  $\bigwedge X. \text{reflclp-on } (\text{dom } \mathcal{T}(F, X)) (=F:X:R \Rightarrow) \equiv (=F:X:R \Rightarrow^=)$

*<proof>*

**thm** *cstepp.rtranclp-on.stable*

**interpretation** *cstepp*: *rewrite-relation*  $F V (=F:V:R \Rightarrow)$

**rewrites**  $\bigwedge X. \text{reflclp-on } (\text{dom } \mathcal{T}(F, X)) (=F:X:R \Rightarrow) \equiv (=F:X:R \Rightarrow^=)$

*<proof>*

**thm** *cstepp.reflclp-on.ctxt-closed*

**lemma** *cstep-reflclpE*:

**assumes**  $s = F:V:R \Rightarrow^= t$  **and**  $\bigwedge \sigma. s : \sigma$  in  $\mathcal{T}(F, V) \implies s = t \implies$  *thesis*  
**and**  $s = F:V:R \Rightarrow t \implies$  *thesis*  
**shows** *thesis*  
 $\langle$ *proof* $\rangle$

**interpretation** *cstep-reflclp*: reflexive dom  $\mathcal{T}(F, V)$  ( $=F:V:R \Rightarrow^=$ )  
 $\langle$ *proof* $\rangle$

**interpretation** *cstep-rsymclp*: reflexive dom  $\mathcal{T}(F, V)$  ( $\Leftarrow F:V:R \Rightarrow^=$ )  
 $\langle$ *proof* $\rangle$

**lemma** *cstep-rtranclp-iff*:  $s = F:V:R \Rightarrow^* t \iff (s = t \wedge s \in \text{dom } \mathcal{T}(F, V)) \vee s = F:V:R \Rightarrow^+ t$   
 $\langle$ *proof* $\rangle$

**lemma** [*rewriting-simps*]:  
**shows** *cstep-rtranclp-refl*:  $s : \sigma$  in  $\mathcal{T}(F, V) \implies s = F:V:R \Rightarrow^* s$   
**and** *cstep-rtranclp-trancl*:  $s = F:V:R \Rightarrow^+ t \implies s = F:V:R \Rightarrow^* t$   
**and** *cstep-rtranclp-step*:  $s = F:V:R \Rightarrow t \implies s = F:V:R \Rightarrow^* t$   
 $\langle$ *proof* $\rangle$

**lemma** *cstep-rtranclp-eq-reflclp*:  
 $(=F:V:R \Rightarrow^*) = \text{reflclp-on } (\text{dom } \mathcal{T}(F, V)) (=F:V:R \Rightarrow)^{++}$   
 $\langle$ *proof* $\rangle$

**lemma** *cstep-rtranclpE*[*consumes 1, case-names refl trancl*]:  
**assumes**  $s = F:V:R \Rightarrow^* t$   
**and**  $\bigwedge \sigma. s : \sigma$  in  $\mathcal{T}(F, V) \implies s = t \implies$  *thesis*  
**and**  $s = F:V:R \Rightarrow^+ t \implies$  *thesis*  
**shows** *thesis*  $\langle$ *proof* $\rangle$

Now we define the conditional root rewrite step.

**definition** *crootstepp* ( $'(=:-:.-\Rightarrow^\varepsilon)$  [51,51,51]1000)  
**where**  $(=F:V:R \Rightarrow^\varepsilon) s t \equiv$   
 $\exists (X. l \rightsquigarrow r \Leftarrow cs) \in R. \exists \vartheta :_s X \rightarrow \mathcal{T}(F, V). (\forall (s \rightsquigarrow t) \in \text{set } cs. s \cdot \vartheta = F:V:R \Rightarrow^* t \cdot \vartheta) \wedge$   
 $s = l \cdot \vartheta \wedge t = r \cdot \vartheta$

**abbreviation** *crootstep-op* ( $((2-)/ =:-:.-\Rightarrow^\varepsilon /(2-))$  [51,51,51,51,51]50)  
**where**  $s = F:V:R \Rightarrow^\varepsilon t \equiv (=F:V:R \Rightarrow^\varepsilon) s t$

**lemma** *crootsteppI*:  
**assumes**  $(X. l \rightsquigarrow r \Leftarrow cs) \in R$  **and**  $\vartheta :_s X \rightarrow \mathcal{T}(F, V)$   
**and**  $\bigwedge u v. (u \rightsquigarrow v) \in \text{set } cs \implies u \cdot \vartheta = F:V:R \Rightarrow^* v \cdot \vartheta$   
**and**  $s = l \cdot \vartheta$  **and**  $t = r \cdot \vartheta$   
**shows**  $s = F:V:R \Rightarrow^\varepsilon t$   
 $\langle$ *proof* $\rangle$

**lemma** *crootstepp-rule*:

**assumes**  $(X. l \rightsquigarrow r \leftarrow cs) \in R$  **and**  $\vartheta :_s X \rightarrow \mathcal{T}(F, V)$   
**and**  $\bigwedge u v. (u \rightsquigarrow v) \in \text{set } cs \implies u \cdot \vartheta =_{F:V:R \Rightarrow^*} v \cdot \vartheta$   
**shows**  $l \cdot \vartheta =_{F:V:R \Rightarrow^\varepsilon} r \cdot \vartheta$   
 $\langle \text{proof} \rangle$

**lemma** *crooststeppE*:

**assumes**  $s =_{F:V:R \Rightarrow^\varepsilon} t$   
**and**  $\bigwedge X l r cs \vartheta. (X. l \rightsquigarrow r \leftarrow cs) \in R \implies \vartheta :_s X \rightarrow \mathcal{T}(F, V) \implies$   
 $\forall u v. (u \rightsquigarrow v) \in \text{set } cs \longrightarrow u \cdot \vartheta =_{F:V:R \Rightarrow^*} v \cdot \vartheta \implies s = l \cdot \vartheta \implies t = r \cdot \vartheta \implies$

*thesis*

**shows** *thesis*  
 $\langle \text{proof} \rangle$

**lemma** *cstepp-root*: **assumes**  $s =_{F:V:R \Rightarrow^\varepsilon} t$  **shows**  $s =_{F:V:R \Rightarrow} t$   
 $\langle \text{proof} \rangle$

**interpretation** *crooststepp*: *subst-closed-general*  $F V W (=_{F:V:R \Rightarrow^\varepsilon}) (=_{F:W:R \Rightarrow^\varepsilon})$   
 $\langle \text{proof} \rangle$

**thm** *crooststepp.symclp.stable*

The conditional reduction is a model of the CTRS.

**lemma** *cstep-rtranclp-models*:  $\mathcal{T}(F, V):Fun:(=_{F:V:R \Rightarrow^*}) \models R$   
 $\langle \text{proof} \rangle$

**locale** *sorted-ctrs* =

**fixes**  $F :: ('f, 's)$  *ssig* **and**  $R :: ('f, 'x, 's)$  *rule set*  
**assumes** *sort-safe*: *sort-safe-rules*  $F R$

**begin**

**lemma** *cstepp-typed*:  $s =_{F:V:R \Rightarrow} t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
 $\langle \text{proof} \rangle$

**sublocale** *cstepp*: *sorted-rewrite-relation*  $F V (=_{F:V:R \Rightarrow})$   
**rewrites**  $\bigwedge X. \text{reflclp-on } (\text{dom } \mathcal{T}(F, X)) (=_{F:X:R \Rightarrow}) \equiv (=_{F:X:R \Rightarrow=})$   
 $\langle \text{proof} \rangle$

**sublocale** *cstep-reflclp*: *sorted-relation*  $\mathcal{T}(F, V) (=_{F:V:R \Rightarrow=})$   
 $\langle \text{proof} \rangle$

**sublocale** *cstep-tranclp*: *sorted-relation*  $\mathcal{T}(F, V) (=_{F:V:R \Rightarrow})^{++}$   
 $\langle \text{proof} \rangle$

**sublocale** *cstep-rtranclp*: *sorted-relation*  $\mathcal{T}(F, V) (=_{F:V:R \Rightarrow^*})$   
 $\langle \text{proof} \rangle$

**sublocale** *cstep-symclp*: *sorted-relation*  $\mathcal{T}(F, V) (\leftarrow_{F:V:R \Rightarrow})$   
 $\langle \text{proof} \rangle$

**sublocale** *cstep-rsymclp*: sorted-relation  $\mathcal{T}(F, V)$  ( $\Leftarrow F:V:R \Rightarrow^=$ )  
*<proof>*

**sublocale** *cstep-rtransymclp*: sorted-relation  $\mathcal{T}(F, V)$  ( $\Leftarrow F:V:R \Rightarrow^*$ )  
*<proof>*

**thm** *cstepp.rtranclp-on.ctx-closed*

**lemma** *cstep-symclp-typed*:  $s \Leftarrow F:V:R \Rightarrow t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
*<proof>*

**lemma** *cstep-tranclp-typed*:  $s = F:V:R \Rightarrow^+ t t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
*<proof>*

**lemma** *cstep-rtransymclp-typed*:  $s \Leftarrow F:V:R \Rightarrow^* t \implies \exists \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \wedge t : \sigma \text{ in } \mathcal{T}(F, V)$   
*<proof>*

**lemma** *cstep-rtranclp-induct*[consumes 1, case-names refl step]:

**assumes** *st*:  $s = F:V:R \Rightarrow^* t$

**and Refl**:  $\bigwedge \sigma. s : \sigma \text{ in } \mathcal{T}(F, V) \implies P s$

**and Step**:  $\bigwedge t u. s = F:V:R \Rightarrow^* t \implies t = F:V:R \Rightarrow u \implies P t \implies P u$

**shows**  $P t$

*<proof>*

**end**

**interpretation** *cstep-reflclp*: reflexive-algebra  $F \mathcal{T}(F, V) \text{ Fun } (=F:V:R \Rightarrow^=)$  *<proof>*

**interpretation** *cstep-tranclp*: transitive-algebra  $F \mathcal{T}(F, V) \text{ Fun } (=F:V:R \Rightarrow)^{++}$   
*<proof>*

**interpretation** *cstep-rtranclp*: quasi-ordered-algebra  $F \mathcal{T}(F, V) \text{ Fun } (=F:V:R \Rightarrow^*)$   
*<proof>*

**interpretation** *cstep-reflclp*: reflexive-monotone-algebra  $F \mathcal{T}(F, V) \text{ Fun } (=F:V:R \Rightarrow^=)$   
*<proof>*

**sublocale** *sorted-ctrs*  $\subseteq$  *cstep-tranclp*: transitive-monotone-algebra  $F \mathcal{T}(F, V) \text{ Fun } (=F:V:R \Rightarrow)^{++}$  *<proof>*

Because conditions are evaluated by full many-step reductions, models of CTRS make sense only if they are quasi-ordered monotone.

**context** *quasi-ordered-monotone-algebra* **begin**

**context** fixes  $R$

**assumes** *ctrs*: *sorted-ctrs*  $F R$  **and** *models*:  $A:I:(\sqsubseteq) \models R$   
**begin**

**interpretation** *sorted-ctrs*  $F R$   $\langle proof \rangle$

**lemma** *cstepp-imp-valid*: **assumes** *st*:  $s =F:V:R \Rightarrow t$  **shows**  $A:I:(\sqsubseteq) \models V. s \rightsquigarrow t$   
 $\langle proof \rangle$

**lemma** *cstepp-le-valid*:  $(=F:V:R \Rightarrow) \leq valid A I (\sqsubseteq) V$   $\langle proof \rangle$

**lemma** *cstep-tranclp-imp-valid*:  $s =F:V:R \Rightarrow^+ t t \Longrightarrow A:I:(\sqsubseteq) \models V. s \rightsquigarrow t$   
 $\langle proof \rangle$

**lemma** *cstep-tranclp-le-valid*:  $(=F:V:R \Rightarrow^+) \leq valid A I (\sqsubseteq) V$   
 $\langle proof \rangle$

**lemma** *cstep-rtranclp-imp-valid*:  $s =F:V:R \Rightarrow^* t \Longrightarrow A:I:(\sqsubseteq) \models V. s \rightsquigarrow t$   
 $\langle proof \rangle$

**lemma** *cstep-rtranclp-le-valid*:  $(=F:V:R \Rightarrow^*) \leq valid A I (\sqsubseteq) V$   
 $\langle proof \rangle$

**end**

**end**

**lemma** (**in** *rewrite-preorder*) *cstep-rtranclp-le*:  
 $sorted-ctrs F R \Longrightarrow \mathcal{T}(F, V):Fun:(\preceq) \models R \Longrightarrow (=F:V:R \Rightarrow^*) \leq (\preceq)$   
 $\langle proof \rangle$

The many-step conditional rewriting is the least rewrite preorder.

**lemma** (**in** *sorted-ctrs*) *cstep-rtranclp-eq-least*:  
 $(=F:V:R \Rightarrow^*) = (LEAST r. rewrite-preorder F V r \wedge (\mathcal{T}(F, V):Fun:r \models R))$   
 $\langle proof \rangle$

**end**

**theory** *Variadic-Signature*  
**imports** *Sorted-Terms.Sorted-Terms*  
**begin**

This theory introduces a convenient notation for variadic signatures.

**definition** *variadic-upd*  $F f \sigma \tau r \equiv$   
*if*  $fst r = f \wedge (\forall \sigma' \in set (snd r). \sigma' = \sigma)$  *then*  $Some \tau$  *else*  $F r$

**syntax**

*-variadic-upd*  $:: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow maplet (('-, -..') / \mapsto / -)$

**translations**

*-Update*  $F (-variadic-upd f \sigma \tau) \equiv CONST variadic-upd F f \sigma \tau$

**lemma** *hastype-in-variadic*:

$f : \sigma s \rightarrow \tau$  in  $F((f', \sigma..) \mapsto \tau') \longleftrightarrow$   
(if  $f = f' \wedge (\forall \sigma' \in \text{set } \sigma s. \sigma' = \sigma)$  then  $\tau = \tau'$  else  $f : \sigma s \rightarrow \tau$  in  $F$ )  
*<proof>*

**lemma** *dom-variadic-upd[simp]*:  $\text{dom } (F((f, \sigma..) \mapsto \tau)) = \text{dom } F \cup \{f\} \times \text{lists } \{\sigma\}$   
*<proof>*

**end**

## 12 Logic

We define propositional logic as a special case of equational logic, where special sort “bool” and logical operators exist.

**theory** *Logic*

**imports** *Models Variadic-Signature Sorted-Terms-More*

**begin**

**lemma** *all-set-iff-replicate*:  $(\forall x \in \text{set } xs. x = y) \longleftrightarrow (\exists n. xs = \text{replicate } n \ y)$   
*<proof>*

**lemma** *all-set-image-conv-ex-map*:

$(\forall fa \in \text{set } fas. fa \in f \text{ ' } A) \longleftrightarrow (\exists as \in \text{lists } A. fas = \text{map } f \ as)$   
*<proof>*

**lemma** (in *subsignature*) *has-same-type-subsig*:  $f : \sigma s \rightarrow \tau$  in  $F \implies f : \sigma s \rightarrow \tau'$   
in  $G \longleftrightarrow \tau = \tau'$   
*<proof>*

**lemma** *all-set-hastype*:

**assumes**  $\forall a \in \text{set } as. a : \sigma$  in  $A$

**shows**  $as :_l \sigma s$  in  $A \longleftrightarrow \sigma s = \text{replicate } (\text{length } as) \ \sigma$

*<proof>*

### 12.1 Syntax

The following locales introduce notations for propositional logic.

**locale** *truth-syntax* = **fixes** *conjF* :: 'f

**begin**

**abbreviation** *trueT*  $\equiv \text{Fun } \text{conjF } []$

**abbreviation** *satisfies-formula*  $((-;- \models -) [51,51,51,25]_4)$  **where**

$I;r;\alpha \models \varphi \equiv I;r;\alpha \models \varphi \rightsquigarrow \text{trueT}$

**abbreviation** *satisfies-formula-eq*  $((-;- \models -) [51,51,25]_4)$  **where**

$I;\alpha \models \varphi \equiv I:(=);\alpha \models \varphi$

**lemma** *satisfies-formula-subst*:  $(I:r;\alpha \models \varphi.\vartheta) \longleftrightarrow (I:r;I[\vartheta]_s \alpha \models \varphi)$   
 $\langle proof \rangle$

**lemma** *satisfies-formula-same-vars*:  
**assumes**  $\forall x \in vars-term \varphi. \alpha x = \beta x$   
**shows**  $(I:r;\alpha \models \varphi) \longleftrightarrow (I:r;\beta \models \varphi)$   
 $\langle proof \rangle$

**abbreviation** *valid-formula*  $((:-:- \models -. -) [51,51,51,100,25]_4)$  **where**  
 $A:I:r \models X. \varphi \equiv A:I:r \models X. \varphi \rightsquigarrow trueT$

**abbreviation** *valid-formula-eq*  $((:- \models -. -) [51,51,100,25]_4)$  **where**  
 $A:I \models X. \varphi \equiv A:I:(=) \models X. \varphi$

**end**

**locale** *logic-symbols-syntax* = **fixes** *conjF disjF negF* :: 'f  
**begin**

**sublocale** *truth-syntax*  $\langle proof \rangle$

**abbreviation** *falseT*  $\equiv Fun\ disjF\ []$

**abbreviation** *andT* (**infixl**  $\wedge_t$  35) **where**  $s \wedge_t t \equiv Fun\ conjF\ [s,t]$

**abbreviation** *orT* (**infixr**  $\vee_t$  30) **where**  $s \vee_t t \equiv Fun\ disjF\ [s,t]$

**abbreviation** *notT* ( $\neg_t$  - [40]40) **where**  $\neg_t s \equiv Fun\ negF\ [s]$

**definition** *impT* (**infixr**  $\longrightarrow_t$  25) **where**  $s \longrightarrow_t t \equiv \neg_t s \vee_t t$

**abbreviation** *list-exT*  $f\ ss \equiv Fun\ disjF\ (map\ f\ ss)$

**abbreviation** *list-allT*  $f\ ss \equiv Fun\ conjF\ (map\ f\ ss)$

**lemma** *logic-cases*[*case-names Not Conj Disj Other*]:  
**assumes**  $f = negF \implies thesis$   
**and**  $f \neq negF \implies f = conjF \implies thesis$   
**and**  $f \neq negF \implies f \neq conjF \implies f = disjF \implies thesis$   
**and**  $f \neq negF \implies f \neq conjF \implies f \neq disjF \implies thesis$   
**shows** *thesis*  
 $\langle proof \rangle$

**abbreviation** *const-of-bool*  $b \equiv if\ b\ then\ trueT\ else\ falseT$

**end**

The following locale defines the signature of logic symbols.

**locale** *logic-signature-syntax* =  
**fixes** *boolS* :: 's **and** *conjF disjF negF* :: 'f  
**begin**

**interpretation** *logic-symbols-syntax*⟨*proof*⟩

We define the (constructor) signature of logic symbols.

**definition** *C* **where**

$C \equiv [(conjF, []) \mapsto boolS, (disjF, []) \mapsto boolS]$

**lemma** *hastype-in-C*:  $f : \sigma s \rightarrow \tau$  *in* *C*  $\longleftrightarrow$   
 $(f = conjF \vee f = disjF) \wedge \sigma s = [] \wedge \tau = boolS$   
⟨*proof*⟩

**lemma** *hastype-in-C-E[elim!]*:

**assumes**  $f : \sigma s \rightarrow \tau$  *in* *C*

**and**  $f = conjF \implies \sigma s = [] \implies \tau = boolS \implies thesis$

**and**  $f = disjF \implies \sigma s = [] \implies \tau = boolS \implies thesis$

**shows** *thesis*

⟨*proof*⟩

**lemma** *conj-hastype-in-C-iff[simp]*:  $conjF : \sigma s \rightarrow \tau$  *in* *C*  $\longleftrightarrow \sigma s = [] \wedge \tau = boolS$   
**and** *disj-hastype-in-C-iff[simp]*:  $disjF : \sigma s \rightarrow \tau$  *in* *C*  $\longleftrightarrow \sigma s = [] \wedge \tau = boolS$   
⟨*proof*⟩

**lemma** *dom-C[simp]*:  $dom\ C = \{(conjF, []), (disjF, [])\}$   
⟨*proof*⟩

**definition** *F* **where**

$F \equiv [(negF, [boolS]) \mapsto boolS, (conjF, boolS..) \mapsto boolS, (disjF, boolS..) \mapsto boolS ]$

**lemma** *dom-F*:  $dom\ F = \{(negF, [boolS])\} \cup \{conjF, disjF\} \times lists\ \{boolS\}$   
⟨*proof*⟩

**lemma** *hastype-in-F*:  $f : \sigma s \rightarrow \tau$  *in* *F*  $\longleftrightarrow$

$\tau = boolS \wedge ($

$(f = conjF \vee f = disjF) \wedge (\forall \sigma \in set\ \sigma s. \sigma = boolS) \vee$

$f = negF \wedge \sigma s = [boolS])$

⟨*proof*⟩

**lemma** *hastype-in-F-E*:

**assumes**  $f : \sigma s \rightarrow \tau$  *in* *F*

**and**  $\bigwedge n. \tau = boolS \implies f = conjF \implies \sigma s = replicate\ n\ boolS \implies thesis$

**and**  $\bigwedge n. \tau = boolS \implies f = disjF \implies \sigma s = replicate\ n\ boolS \implies thesis$

**and**  $f = negF \implies \tau = boolS \implies \sigma s = [boolS] \implies thesis$

**shows** *thesis*

⟨*proof*⟩

**lemma** *conj-hastype-iff[simp]*:  $conjF : \sigma s \rightarrow \tau$  *in* *F*  $\longleftrightarrow (\forall \sigma \in set\ \sigma s. \sigma = boolS)$

$\wedge \tau = \text{boolS}$   
**and** *disj-hastype-iff*[simp]:  $\text{disj}F : \sigma s \rightarrow \tau \text{ in } F \longleftrightarrow (\forall \sigma \in \text{set } \sigma s. \sigma = \text{boolS}) \wedge \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *Fun-conj-hastype-iff*:  $\text{Fun conj}F \varphi s : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow (\forall \varphi \in \text{set } \varphi s. \varphi : \text{boolS} \text{ in } \mathcal{T}(F, V)) \wedge \tau = \text{boolS}$   
**and** *Fun-disj-hastype-iff*:  $\text{Fun disj}F \varphi s : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow (\forall \varphi \in \text{set } \varphi s. \varphi : \text{boolS} \text{ in } \mathcal{T}(F, V)) \wedge \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *and-hastype-iff*[simp]:  $(\varphi \wedge_t \psi) : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow \varphi : \text{boolS} \text{ in } \mathcal{T}(F, V) \wedge \psi : \text{boolS} \text{ in } \mathcal{T}(F, V) \wedge \tau = \text{boolS}$   
**and** *or-hastype-iff*[simp]:  $(\varphi \vee_t \psi) : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow \varphi : \text{boolS} \text{ in } \mathcal{T}(F, V) \wedge \psi : \text{boolS} \text{ in } \mathcal{T}(F, V) \wedge \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *const-of-bool-hastype-in-C*[simp]:  
 $\text{const-of-bool } b : \sigma \text{ in } \mathcal{T}(C, V) \longleftrightarrow \sigma = \text{boolS}$   
 ⟨proof⟩

**lemma** *const-of-bool-hastype-in-F*[simp]:  
 $\text{const-of-bool } b : \sigma \text{ in } \mathcal{T}(F, V) \longleftrightarrow \sigma = \text{boolS}$   
 ⟨proof⟩

**lemma** *sorts-ssig-F*[simp]:  $\text{sorts-ssig } F = \{\text{boolS}\}$   
 ⟨proof⟩

**lemma** *sorts-ssig-C*[simp]:  $\text{sorts-ssig } C = \{\text{boolS}\}$   
 ⟨proof⟩

**end**

**locale** *logic-signature* = *logic-signature-syntax* +

**assumes** *neg-neq-conj*[simp]:  $\text{neg}F \neq \text{conj}F$

**and** *neg-neq-disj*[simp]:  $\text{neg}F \neq \text{disj}F$

**and** *conj-neq-disj*[simp]:  $\text{conj}F \neq \text{disj}F$

**begin**

**lemmas** *conj-neq-neg*[simp] = *neg-neq-conj*[symmetric]

**lemmas** *disj-neq-neg*[simp] = *neg-neq-disj*[symmetric]

**lemmas** *disj-neq-conj*[simp] = *conj-neq-disj*[symmetric]

**interpretation** *logic-symbols-syntax*⟨proof⟩

**lemma** *neg-hastype-iff*[simp]:  $\text{neg}F : \sigma s \rightarrow \tau \text{ in } F \longleftrightarrow \sigma s = [\text{boolS}] \wedge \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *Fun-neg-hastype-iff*:  $\text{Fun neg}F \varphi s : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow (\exists \varphi. \varphi s = [\varphi] \wedge \varphi$

: *boolS* in  $\mathcal{T}(F, V) \wedge \tau = \text{boolS}$   
 ⟨*proof*⟩

**lemma** *not-hastype-iff[simp]*:  $(\neg_t \varphi) : \tau$  in  $\mathcal{T}(F, V) \longleftrightarrow \varphi : \text{boolS}$  in  $\mathcal{T}(F, V) \wedge \tau = \text{boolS}$   
 ⟨*proof*⟩

**lemma** *imp-hastype-iff[simp]*:  $(\varphi \longrightarrow_t \psi) : \text{boolS}$  in  $\mathcal{T}(F, V) \longleftrightarrow \varphi : \text{boolS}$  in  $\mathcal{T}(F, V) \wedge \psi : \text{boolS}$  in  $\mathcal{T}(F, V)$   
 ⟨*proof*⟩

**end**

## 12.2 Semantics

A logic is an algebra over the logic signature, which interprets logical operators in the expected manner.

**locale** *truth-interpretation-syntax* =  
 fixes  $I :: 'f \Rightarrow 'a \text{ list} \Rightarrow 'a$  and  $\text{conj}F :: 'f$   
**begin**

**sublocale** *truth-syntax*⟨*proof*⟩

**abbreviation** *true* **where**  $\text{true} \equiv I \text{ conj}F$  []

**lemma** *satisfies-formula-iff*:  $(I; (\sim); \alpha \models \varphi) \longleftrightarrow I[\varphi]\alpha \sim \text{true}$   
**for equiv** (**infix**  $\sim$  50) ⟨*proof*⟩

**lemmas** *satisfies-formulaI* = *satisfies-formula-iff*[*THEN iffD2*]  
**lemmas** *satisfies-formulaD* = *satisfies-formula-iff*[*THEN iffD1*]

**end**

**locale** *logic-syntax* =  
 fixes  $F :: ('f, 's) \text{ ssig}$  and  $A :: 'a \rightarrow 's$  and  $I :: 'f \Rightarrow 'a \text{ list} \Rightarrow 'a$   
 and  $\text{boolS} :: 's$  and  $\text{conj}F$   $\text{disj}F$   $\text{neg}F :: 'f$   
**begin**

**sublocale** *logic-symbols-syntax*⟨*proof*⟩

**sublocale** *truth-interpretation-syntax*⟨*proof*⟩

**abbreviation** *false* **where**  $\text{false} \equiv I \text{ disj}F$  []

**sublocale** *logic*: *logic-signature-syntax*⟨*proof*⟩

**end**

We introduce *quasi-logic*, where the interpretations of formulas can be multi-valued, but relates to either true or false.

**locale** *quasi-truth-algebra* = *sorted-algebra* + *equivalence-syntax* + *truth-interpretation-syntax*  
 +  
   **constrains** *equiv* :: -  
   **assumes** *true-is-true*:  $true \sim true$   
**begin**  
  
**sublocale** *truth-interpretation-syntax*⟨*proof*⟩  
  
**lemma** *valid-formula-subst*:  
  **assumes** *val*:  $A:I:(\sim) \models X. \varphi$  **and**  $\vartheta: \vartheta :_s X \rightarrow \mathcal{T}(F, V)$   
  **shows**  $A:I:(\sim) \models V. \varphi \cdot \vartheta$   
  ⟨*proof*⟩  
  
**lemma** *satisfies-true[simp]*:  $I:(\sim); \alpha \models trueT$  ⟨*proof*⟩  
  
**lemma** *valid-true[simp]*:  $A:I:(\sim) \models X. trueT$  ⟨*proof*⟩  
  
**end**  
  
**locale** *quasi-logic* = *sorted-algebra* +  
  *equivalence-syntax* + *logic-syntax* +  
  *logic: subsignature logic.F F* +  
  **constrains** *equiv* :: -  
  **assumes** *neg-is-true*:  $a : boolS \text{ in } A \implies$   
     $I \text{ neg}F [a] \sim true \iff \neg (a \sim true)$   
  **assumes** *conj-is-true*:  $\forall a \in \text{set } as. a : boolS \text{ in } A \implies$   
     $I \text{ conj}F as \sim true \iff (\forall a \in \text{set } as. a \sim true)$   
  **assumes** *disj-is-true*:  $\forall a \in \text{set } as. a : boolS \text{ in } A \implies$   
     $I \text{ disj}F as \sim true \iff (\exists a \in \text{set } as. a \sim true)$   
**begin**  
  
**sublocale** *quasi-truth-algebra*  
  ⟨*proof*⟩  
  
**sublocale** *logic-part: subsignature-algebra logic.F F*⟨*proof*⟩  
  
**lemma** *false-is-not-true*:  $\neg false \sim true$   
  ⟨*proof*⟩  
  
**lemma** **assumes**  $\forall \sigma \in \text{set } \sigma s. \sigma = boolS$   
  **shows** *conj-hastype-iff-bool[simp]*:  $\text{conj}F : \sigma s \rightarrow \tau \text{ in } F \iff \tau = boolS$   
  **and** *disj-hastype-iff-bool[simp]*:  $\text{disj}F : \sigma s \rightarrow \tau \text{ in } F \iff \tau = boolS$   
  ⟨*proof*⟩  
  
**lemma** *conj-hastype-replicate*:  $\text{conj}F : \text{replicate } n \text{ bool}S \rightarrow boolS \text{ in } F$  ⟨*proof*⟩  
  
**lemma** *disj-hastype-replicate*:  $\text{disj}F : \text{replicate } n \text{ bool}S \rightarrow boolS \text{ in } F$  ⟨*proof*⟩  
  
**lemma** **assumes**  $\forall \varphi \in \text{set } \varphi s. \varphi : boolS \text{ in } \mathcal{T}(F, V)$

**shows** *Fun-conj-hastype*:  $\text{Fun conjF } \varphi s : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow \tau = \text{boolS}$   
**and** *Fun-disj-hastype*:  $\text{Fun disjF } \varphi s : \tau \text{ in } \mathcal{T}(F, V) \longleftrightarrow \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *true-hastype-in-Term-iff[simp]*:  $\text{trueT} : \sigma \text{ in } \mathcal{T}(F, V) \longleftrightarrow \sigma = \text{boolS}$   
**and** *false-hastype-in-Term-iff[simp]*:  $\text{falseT} : \sigma \text{ in } \mathcal{T}(F, V) \longleftrightarrow \sigma = \text{boolS}$   
 ⟨proof⟩

**lemma** *true-hastype-iff[simp]*:  $\text{true} : \sigma \text{ in } A \longleftrightarrow \sigma = \text{boolS}$   
**and** *false-hastype-iff[simp]*:  $\text{false} : \sigma \text{ in } A \longleftrightarrow \sigma = \text{boolS}$   
 ⟨proof⟩

**sublocale** *logic*: *logic-signature*  
 ⟨proof⟩

**lemma** *neg-hastype*:  $\text{negF} : [\text{boolS}] \rightarrow \text{boolS} \text{ in } F$   
 ⟨proof⟩

**lemma** *not-hastype-in-ssig[simp]*:  $\text{negF} : [\text{boolS}] \rightarrow \tau \text{ in } F \longleftrightarrow \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *not-hastype-in-Term[simp]*:  $\varphi : \text{boolS} \text{ in } \mathcal{T}(F, V) \implies (\neg_t \varphi) : \text{boolS} \text{ in } \mathcal{T}(F, V)$   
 ⟨proof⟩

**lemma**  
**assumes** *as*:  $\forall a \in \text{set as. } a : \text{boolS} \text{ in } A$   
**shows** *intp-conj-hastype[simp]*:  $I \text{ conjF } as : \tau \text{ in } A \longleftrightarrow \tau = \text{boolS}$   
**and** *intp-disj-hastype[simp]*:  $I \text{ disjF } as : \tau \text{ in } A \longleftrightarrow \tau = \text{boolS}$   
 ⟨proof⟩

**lemma** *intp-neg-hastype[simp]*:  $a : \text{boolS} \text{ in } A \implies I \text{ negF } [a] : \tau \text{ in } A \longleftrightarrow \tau = \text{boolS}$   
 ⟨proof⟩

**lemma**  
**assumes** *a*:  $a : \text{boolS} \text{ in } A$  **and** *bs*:  $\forall b \in \text{set bs. } b : \text{boolS} \text{ in } A$   
**shows** *conj-Cons*:  $I \text{ conjF } (a \# bs) \sim \text{true} \longleftrightarrow a \sim \text{true} \wedge I \text{ conjF } bs \sim \text{true}$   
**and** *disj-Cons*:  $I \text{ disjF } (a \# bs) \sim \text{true} \longleftrightarrow a \sim \text{true} \vee I \text{ disjF } bs \sim \text{true}$   
 ⟨proof⟩

**lemma**  
**assumes**  $I[\varphi]\alpha : \text{boolS} \text{ in } A$  **and**  $\forall \psi \in \text{set } \varphi s. I[\psi]\alpha : \text{boolS} \text{ in } A$   
**shows** *satisfies-conj-Cons'*:  $(I:(\sim);\alpha \models \text{Fun conjF } (\varphi \# \varphi s)) \longleftrightarrow (I:(\sim);\alpha \models \varphi) \wedge (I:(\sim);\alpha \models \text{Fun conjF } \varphi s)$   
**and** *satisfies-disj-Cons'*:  $(I:(\sim);\alpha \models \text{Fun disjF } (\varphi \# \varphi s)) \longleftrightarrow (I:(\sim);\alpha \models \varphi) \vee (I:(\sim);\alpha \models \text{Fun disjF } \varphi s)$   
 ⟨proof⟩

**lemma**

**assumes**  $\forall \varphi \in \text{set } \varphi s. I[\varphi]\alpha : \text{boolS in } A$

**shows** *satisfies-conj*:  $(I:(\sim);\alpha \models \text{Fun conjF } \varphi s) \longleftrightarrow (\forall \varphi \in \text{set } \varphi s. (I:(\sim);\alpha \models \varphi))$

**and** *satisfies-disj*:  $(I:(\sim);\alpha \models \text{Fun disjF } \varphi s) \longleftrightarrow (\exists \varphi \in \text{set } \varphi s. (I:(\sim);\alpha \models \varphi))$

*<proof>*

**lemmas** *satisfies-conjD* = *satisfies-conj*[*THEN iffD1, rotated -1*]

**lemmas** *satisfies-disjD* = *satisfies-disj*[*THEN iffD1, rotated -1*]

**lemmas** *satisfies-conjI* = *satisfies-conj*[*THEN iffD2, unfolded conj-imp-eq-imp-imp*]

**lemmas** *satisfies-disjI* = *satisfies-disj*[*THEN iffD2*]

**lemma** *satisfies-and*:

**assumes**  $I[\varphi]\alpha : \text{boolS in } A$  **and**  $\psi: I[\psi]\alpha : \text{boolS in } A$

**shows**  $(I:(\sim);\alpha \models \varphi \wedge_t \psi) \longleftrightarrow (I:(\sim);\alpha \models \varphi) \wedge (I:(\sim);\alpha \models \psi)$

*<proof>*

**lemma** *satisfies-or*:

**assumes**  $I[\varphi]\alpha : \text{boolS in } A$  **and**  $\psi: I[\psi]\alpha : \text{boolS in } A$

**shows**  $(I:(\sim);\alpha \models \varphi \vee_t \psi) \longleftrightarrow (I:(\sim);\alpha \models \varphi) \vee (I:(\sim);\alpha \models \psi)$

*<proof>*

**lemmas** *satisfies-andD* = *satisfies-and*[*THEN iffD1, rotated -1*]

**lemmas** *satisfies-orD* = *satisfies-or*[*THEN iffD1, rotated -1*]

**lemmas** *satisfies-andI* = *satisfies-and*[*THEN iffD2, unfolded conj-imp-eq-imp-imp*]

**lemmas** *satisfies-orI* = *satisfies-or*[*THEN iffD2*]

**lemma** *satisfies-not*:

**assumes**  $I[\varphi]\alpha : \text{boolS in } A$

**shows**  $(I:(\sim);\alpha \models \neg_t \varphi) \longleftrightarrow \neg(I:(\sim);\alpha \models \varphi)$

*<proof>*

**lemmas** *satisfies-notD* = *satisfies-not*[*THEN iffD1, rotated -1*]

**lemmas** *satisfies-notI* = *satisfies-not*[*THEN iffD2, unfolded not-def, rule-format*]

**lemma** *satisfies-imp*:

**assumes**  $I[\varphi]\alpha : \text{boolS in } A$  **and**  $I[\psi]\alpha : \text{boolS in } A$

**shows**  $(I:(\sim);\alpha \models \varphi \longrightarrow_t \psi) \longleftrightarrow ((I:(\sim);\alpha \models \varphi) \longrightarrow (I:(\sim);\alpha \models \psi))$

*<proof>*

**lemmas** *satisfies-impD* = *satisfies-imp*[*THEN iffD1, rule-format, rotated -2*]

**lemmas** *satisfies-impI* = *satisfies-imp*[*THEN iffD2, rule-format*]

**lemma**

**assumes**  $\varphi : \text{boolS in } \mathcal{T}(F, V)$  **and**  $\forall \psi \in \text{set } \varphi s. \psi : \text{boolS in } \mathcal{T}(F, V)$  **and**  $\alpha :_s V \rightarrow A$

**shows** *satisfies-conj-Cons*:  $(I:(\sim);\alpha \models \text{Fun conjF } (\varphi \# \varphi s)) \longleftrightarrow (I:(\sim);\alpha \models \varphi) \wedge (I:(\sim);\alpha \models \text{Fun conjF } \varphi s)$

**and satisfies-disj-Cons:**  $(I:(\sim);\alpha \models \text{Fun disjF } (\varphi \# \varphi s)) \longleftrightarrow (I:(\sim);\alpha \models \varphi)$   
 $\vee (I:(\sim);\alpha \models \text{Fun disjF } \varphi s)$   
 $\langle \text{proof} \rangle$

**lemma valid-and:**

**assumes**  $\varphi : \text{boolS}$  in  $\mathcal{T}(F, V)$  **and**  $\psi : \text{boolS}$  in  $\mathcal{T}(F, V)$   
**shows**  $(A:I:(\sim) \models V. \varphi \wedge_t \psi) \longleftrightarrow (A:I:(\sim) \models V. \varphi) \wedge (A:I:(\sim) \models V. \psi)$   
 $\langle \text{proof} \rangle$

**lemma valid-conj:**

**assumes**  $\forall \varphi \in \text{set } \varphi s. \varphi : \text{boolS}$  in  $\mathcal{T}(F, V)$   
**shows**  $(A:I:(\sim) \models V. \text{Fun conjF } \varphi s) \longleftrightarrow (\forall \varphi \in \text{set } \varphi s. (A:I:(\sim) \models V. \varphi))$   
 $\langle \text{proof} \rangle$

**lemma valid-impD:**

**assumes**  $\text{imp}: A:I:(\sim) \models V. \varphi \longrightarrow_t \psi$  **and**  $\text{prem}: I:(\sim);\alpha \models \varphi$   
**and**  $\alpha : \alpha :_s V \rightarrow A$  **and**  $\varphi : \varphi : \text{boolS}$  in  $\mathcal{T}(F, V)$  **and**  $\psi : \psi : \text{boolS}$  in  $\mathcal{T}(F, V)$   
**shows**  $I:(\sim);\alpha \models \psi$   
 $\langle \text{proof} \rangle$

**lemma valid-impI:**

**assumes**  $*$ :  $\bigwedge \alpha. \alpha :_s V \rightarrow A \implies I:(\sim);\alpha \models \varphi \implies I:(\sim);\alpha \models \psi$   
**and**  $\varphi : \varphi : \text{boolS}$  in  $\mathcal{T}(F, V)$  **and**  $\psi : \psi : \text{boolS}$  in  $\mathcal{T}(F, V)$   
**shows**  $A:I:(\sim) \models V. \varphi \longrightarrow_t \psi$   
 $\langle \text{proof} \rangle$

**lemma bool-in-sorts-ssig:**  $\text{boolS} \in \text{sorts-ssig } F$

$\langle \text{proof} \rangle$

**end**

**lemma quasi-logic-cong:**

**assumes**  $F: F = F'$   
**and**  $A: A = A'$   
**and**  $I: \bigwedge f \sigma s \tau \text{ as}. f : \sigma s \rightarrow \tau$  in  $F' \implies \text{as} :_l \sigma s$  in  $A' \implies I f \text{ as} = I' f \text{ as}$   
**and**  $\text{le}: \bigwedge a. a : \text{boolS}$  in  $A' \implies \text{le } a (I' \text{ conjF } \square) \longleftrightarrow \text{le}' a (I' \text{ conjF } \square)$   
**and**  $\text{boolS}: \text{boolS} = \text{boolS}'$   
**and**  $\text{conjF}: \text{conjF} = \text{conjF}'$   
**shows**  $\text{quasi-logic } F A I \text{ le } \text{boolS } \text{conjF } \text{disjF } \text{negF} \longleftrightarrow$   
 $\text{quasi-logic } F' A' I' \text{ le}' \text{ boolS}' \text{ conjF}' \text{ disjF } \text{negF}$   
 $\langle \text{proof} \rangle$

A logic is a quasi-logic with equality.

**locale**  $\text{logic} = \text{quasi-logic}$  **where**  $\text{equiv} = (=)$

**begin**

**lemma**  $A:I \models X. \varphi \equiv \forall \alpha :_s X \rightarrow A. I[\varphi]\alpha = \text{true}$

$\langle \text{proof} \rangle$

**end**

**locale** *logic-const* = *sorted-algebra-const* + *logic*

### 12.3 Propositional Logic

Here we define the propositional logic.

**context** *logic-signature-syntax* **begin**

**definition**  $A :: \text{bool} \rightarrow 's$  **where**

$A\ b \equiv \text{Some } \text{boolS}$

**lemma** *hastype-in-A[simp]*:  $b : \sigma$  in  $A \longleftrightarrow \sigma = \text{boolS}$  *<proof>*

**lemma** *dom-A[simp]*:  $\text{dom } A = \text{UNIV}$  *<proof>*

**lemma** *A-restrict-ran*:  $A \upharpoonright^r S = (\text{if } \text{boolS} \in S \text{ then } A \text{ else } \emptyset)$   
*<proof>*

**definition**  $I$  **where**  $I\ f\ bs \equiv$

*if*  $f = \text{conjF}$  *then*  $\forall b \in \text{set } bs. b$

*else if*  $f = \text{disjF}$  *then*  $\exists b \in \text{set } bs. b$

*else*  $\neg bs!0$

**sublocale** *truth-interpretation-syntax*  $I$  *<proof>*

**sublocale** *sorted-algebra*  $F\ A\ I$   
*<proof>*

**sublocale** *subsignature-algebra*  $F\ F\ A\ I$   
*<proof>*

**end**

**context** *logic-signature* **begin**

**lemma** *I-simps[simp]*:

$I\ \text{conjF}\ bs \longleftrightarrow (\forall b \in \text{set } bs. b)$

$I\ \text{disjF}\ bs \longleftrightarrow (\exists b \in \text{set } bs. b)$

$I\ \text{negF}\ bs \longleftrightarrow \neg bs!0$

*<proof>*

**sublocale** *logic*  $F\ A\ I$   
*<proof>*

**sublocale** *logic-const*  $F\ A\ I\ \text{const-of-bool } C$   
*<proof>*

**lemma** *true-hastype-in-C*:  $\text{trueT} : \text{boolS}$  in  $\mathcal{T}(C, V)$

**and** *false-hastype-in-C*: *falseT* : *boolS* in  $\mathcal{T}(C, V)$   
 ⟨*proof*⟩

**end**

**end**

## 13 Logically Constrained Rewriting

**theory** *Constrained-Rewriting*  
**imports** *Logic Sorted-Rewriting*  
**begin**

Constrained rules extend rewrite rules with an extra term which represents the constraint.

**datatype** (*dead 'f*, *dead 'v*, *dead 's*) *lcrule* =  
*LCRule* (*vars*: *'v*  $\rightarrow$  *'s*) (*lhs*: (*'f*, *'v*) *term*) (*rhs*: (*'f*, *'v*) *term*) (*constraint*: (*'f*, *'v*)  
*term*)  
 ( $\cdot$ .  $\rightsquigarrow$   $\cdot$  |  $\cdot$  - [100,51,51,30]21)

**hide-const**(**open**) *lcrule.vars lcrule.lhs lcrule.rhs lcrule.constraint*

**abbreviation** *ball-lcrule* **where** *ball-lcrule*  $R P \equiv \forall X l r c. (X. l \rightsquigarrow r \mid c) \in R \rightarrow P X l r c$

**abbreviation** *bex-lcrule* **where** *bex-lcrule*  $R P \equiv \exists X l r c. (X. l \rightsquigarrow r \mid c) \in R \wedge P X l r c$

**syntax**

*-ball-lcrule* :: *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  (*'f*, *'v*, *'s*) *lcrule set*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool*  
 (( $\exists \forall$  (( $\cdot$ .  $\rightsquigarrow$   $\cdot$  |  $\cdot$ )/  $\in$   $\cdot$ )/  $\cdot$ ) [0, 0, 0, 0, 0, 10] 10)  
*-bex-lcrule* :: *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  *pttrn*  $\Rightarrow$  (*'f*, *'v*, *'s*) *lcrule set*  $\Rightarrow$  *bool*  $\Rightarrow$  *bool*  
 (( $\exists \exists$  (( $\cdot$ .  $\rightsquigarrow$   $\cdot$  |  $\cdot$ )/  $\in$   $\cdot$ )/  $\cdot$ ) [0, 0, 0, 0, 0, 10] 10)

**translations**

$\forall (X. l \rightsquigarrow r \mid c) \in R. e \Rightarrow \text{CONST } \textit{ball-lcrule } R (\lambda X l r c. e)$   
 $\exists (X. l \rightsquigarrow r \mid c) \in R. e \Rightarrow \text{CONST } \textit{bex-lcrule } R (\lambda X l r c. e)$

To be a well-typed constrained rewrite rule, the left- and right-hand sides must have the same type and the constraint must be of bool type.

**locale** *lcrule-syntax* =

**fixes** *F' F* :: (*'f*, *'s*) *ssig* **and** *boolS* :: *'s*

**begin**

**definition** *lcrule*  $\equiv \lambda (X. l \rightsquigarrow r \mid \varphi) \Rightarrow$

( $\exists \sigma. l : \sigma$  in  $\mathcal{T}(F', X) \wedge r : \sigma$  in  $\mathcal{T}(F', X) \wedge \varphi : \textit{boolS}$  in  $\mathcal{T}(F, X)$ )

**lemma** *lcrule-simp*:

$lcrule (X. l \rightsquigarrow r \mid \varphi) \longleftrightarrow (\exists \sigma. l : \sigma \text{ in } \mathcal{T}(F', X) \wedge r : \sigma \text{ in } \mathcal{T}(F', X) \wedge \varphi : boolS \text{ in } \mathcal{T}(F, X))$   
 ⟨proof⟩

**lemma** *lcruleI*:

$\varrho = (X. l \rightsquigarrow r \mid \varphi) \implies l : \sigma \text{ in } \mathcal{T}(F', X) \implies r : \sigma \text{ in } \mathcal{T}(F', X) \implies \varphi : boolS \text{ in } \mathcal{T}(F, X) \implies lcrule \varrho$   
 ⟨proof⟩

**lemma** *lcrule-has-same-type*:  $lcrule (X. l \rightsquigarrow r \mid \varphi) \implies l : \sigma \text{ in } \mathcal{T}(F', X) \longleftrightarrow r : \sigma \text{ in } \mathcal{T}(F', X)$   
 ⟨proof⟩

**end**

A well-typed LCTRS requires that all rules are well-typed and signatures satisfy expected inclusion.

**locale** *lctrs* = *lcrule-syntax*  $F' F boolS$  **for**  $F' R F boolS +$   
**assumes** *tsig-le-sig*:  $F \subseteq_m F'$   
**and** *typed*:  $\varrho \in R \implies lcrule \varrho$   
**begin**

**lemmas** *tTerm-le* = *Term-mono-left*[*OF tsig-le-sig*]

**end**

The LCTRS semantics demands that instantiations of rules are restricted to those which map variables in constraints to *values*. Therefore we need to specify what are values: here, rather than assuming a *constant symbol* for each semantic values [2], we only require every value  $v : \sigma \text{ in } A$  is represented by a ground constructor term  $\eta v : \sigma \text{ in } \mathcal{T}(C)$ .

**locale** *lctrs-semantics-syntax* =  
**fixes**  $F' :: ('f, 's) ssig$   
**and**  $R :: ('f, 'v, 's) lcrule \text{ set}$   
**and**  $F :: ('f, 's) ssig$   
**and**  $A :: 'a \rightarrow 's$   
**and**  $I :: 'f \Rightarrow 'a \text{ list} \Rightarrow 'a$   
**and**  $\eta :: 'a \Rightarrow ('f, unit) \text{ term}$   
**and**  $C :: ('f, 's) ssig$   
**and**  $boolS :: 's$   
**and**  $conjF :: 'f$   
**begin**

**sublocale** *truth-syntax*⟨proof⟩

We do not define constrained rewrite steps from scratch. We just define the (sorted) TRS which the LCTRS represents. That is, the instances of

constrained rules with respect to ground constructor substitutions over the variables in the constraints that satisfy the constraint.

**definition** *as-trs*  $\equiv$

$$\begin{aligned} & \{(X. l \cdot \vartheta \rightsquigarrow r \cdot \vartheta) \mid X l r \varphi \vartheta. \\ & (X. l \rightsquigarrow r \mid \varphi) \in R \wedge \\ & \vartheta :_s X \mid \text{vars } \varphi \rightarrow \mathcal{T}(C, \emptyset) \wedge (\forall x \in \text{dom } X - \text{vars } \varphi. \vartheta x = \text{Var } x) \wedge \\ & (A:I \models \emptyset. \varphi \cdot \vartheta)\} \end{aligned}$$

**lemma** *as-trs-iff*:  $(X. l' \rightsquigarrow r') \in \text{as-trs} \iff (\exists l r \varphi \vartheta.$

$$\begin{aligned} & (X. l \rightsquigarrow r \mid \varphi) \in R \wedge \\ & \vartheta :_s X \mid \text{vars } \varphi \rightarrow \mathcal{T}(C, \emptyset) \wedge (\forall x \in \text{dom } X - \text{vars } \varphi. \vartheta x = \text{Var } x) \wedge (A:I \models \emptyset. \\ & \varphi \cdot \vartheta) \wedge \\ & l' = l \cdot \vartheta \wedge r' = r \cdot \vartheta) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *as-trsI*:  $(X. l \rightsquigarrow r \mid \varphi) \in R \implies A:I \models \emptyset. \varphi \cdot \vartheta \implies$

$$\begin{aligned} & \vartheta :_s X \mid \text{vars } \varphi \rightarrow \mathcal{T}(C, \emptyset) \implies \forall x \in \text{dom } X - \text{vars } \varphi. \vartheta x = \text{Var } x \implies (X. l \cdot \vartheta \\ & \rightsquigarrow r \cdot \vartheta) \in \text{as-trs} \\ & \langle \text{proof} \rangle \end{aligned}$$

**end**

Rewriting in LCTRS requires extra rewrite steps of theory terms called “calculation step”. Below we define a one that reduces a ground basic theory term in one step to the representation of its value. We cannot just add definitions in existing locales, because the new definitions will not be imported to existing interpretations (in Isabelle 2025).

**locale** *sorted-algebra-calculation* = *sorted-algebra-const* +

**constrains**  $A :: 'a \rightarrow 's$  **and**  $I :: 'f \Rightarrow -$  **and**  $\eta :: -$

**begin**

To avoid complication in types, this TRS contains ground rules whose type of variables is *unit*.

**definition** *calculation-trs* ::  $('f, \text{unit}, 's)$  *axiom set* **where**

$$\begin{aligned} & \text{calculation-trs} \equiv \\ & \{(\emptyset. \text{Fun } f \text{ } l s \rightsquigarrow \eta (I f [I[l]] \text{undefined}. l \leftarrow l s)) \mid f \text{ } l s \sigma s \tau. \\ & f : \sigma s \rightarrow \tau \text{ in } F \wedge \neg f : \sigma s \rightarrow \tau \text{ in } C \wedge l s :_l \sigma s \text{ in } \mathcal{T}(C)\} \end{aligned}$$

A root step of the calculation TRS is polymorphic, as expected:

**lemma** *calculation-rootstepp-iff*:

$$\begin{aligned} & s - F':V:\text{calculation-trs} \rightarrow^\varepsilon t \iff (\exists f \text{ } l s \sigma s \tau. \\ & f : \sigma s \rightarrow \tau \text{ in } F \wedge \neg f : \sigma s \rightarrow \tau \text{ in } C \wedge l s :_l \sigma s \text{ in } \mathcal{T}(C, \emptyset) \wedge \\ & s = \text{Fun } f \text{ } l s \wedge t = \text{const.term-of } (I[s]\alpha)) \text{ (is } ?l \iff ?r) \\ & \langle \text{proof} \rangle \end{aligned}$$

**sublocale** *calculation-trs*: *sorted-trs*  $F$  *calculation-trs*

$\langle \text{proof} \rangle$

**end**

**locale** *lctrs-semantic* = *lctrs-semantic-syntax* +  
*lctrs* + *sorted-algebra-calculation*  
**begin**

**sublocale** *quasi-truth-algebra* *F A I* (=)  $\langle$ *proof* $\rangle$

**sublocale** *as-trs: sorted-trs* *F' as-trs*  
 $\langle$ *proof* $\rangle$

**lemma** *lctrs-rootstepp-iff*:  $s -F':V:as-trs \rightarrow^\varepsilon t \longleftrightarrow$  (  
 $\exists (X. l \rightsquigarrow r \mid \varphi) \in R. \exists \vartheta :_s X \rightarrow \mathcal{T}(F', V).$   
 $\vartheta :_s X \mid \text{vars } \varphi \rightarrow \mathcal{T}(C, \emptyset) \wedge (A:I \models \emptyset. \varphi \cdot \vartheta) \wedge s = l \cdot \vartheta \wedge t = r \cdot \vartheta)$   
(**is** ?*L*  $\longleftrightarrow$  ?*R*)  
 $\langle$ *proof* $\rangle$

**lemma** *lctrs-rootsteppI*:  $(X. l \rightsquigarrow r \mid \varphi) \in R \implies \vartheta :_s X \rightarrow \mathcal{T}(F', V) \implies$   
 $\vartheta :_s X \mid \text{vars } \varphi \rightarrow \mathcal{T}(C, \emptyset) \implies (A:I \models \emptyset. \varphi \cdot \vartheta) \implies s = l \cdot \vartheta \implies t = r \cdot \vartheta \implies$   
 $s -F':V:as-trs \rightarrow^\varepsilon t$   
 $\langle$ *proof* $\rangle$

**end**

**end**

### 13.1 Sorted Injections

**theory** *Sorted-Injections*  
**imports** *Sorted-Terms-More*  
**begin**

One can define the image of an algebra with respect to an injection.

**definition** *image-intp*  $\eta A I f as \equiv \eta (I f (map (inv-into (dom A) \eta) as))$

**lemma** *image-intp-Nil[simp]*:  
 $image-intp \eta A I g [] = \eta (I g [])$   
 $\langle$ *proof* $\rangle$

**locale** *sorted-injection* =  
**fixes**  $\eta A$   
**assumes** *inj-on*: *inj-on*  $\eta (dom A)$   
**begin**

**lemma** *inv-app[simp]*:  
**shows**  $a : \sigma \text{ in } A \implies inv-into (dom A) \eta (\eta a) = a$   
 $\langle$ *proof* $\rangle$

**lemma** *map-inv-app*:  $as : \iota \ \sigma s \text{ in } A \implies \text{map } (\text{inv-into } (\text{dom } A) \ \eta \circ \eta) \ as = as$   
 ⟨proof⟩

**lemma** *in-dom-inj*:  $a \in \text{dom } A \implies a' \in \text{dom } A \implies \eta \ a = \eta \ a' \longleftrightarrow a = a'$   
 ⟨proof⟩

**lemma** *hastype-inj*:  $a : \sigma \text{ in } A \implies a' : \sigma' \text{ in } A \implies \eta \ a = \eta \ a' \longleftrightarrow a = a'$   
 ⟨proof⟩

**sublocale** *sort-preserving*  $\eta$   
 ⟨proof⟩

**lemma** *image-intp-map*:  
 assumes  $as : \iota \ \sigma s \text{ in } A$  shows  $\text{image-intp } \eta \ A \ I \ g \ (\text{map } \eta \ as) = \eta \ (I \ g \ as)$   
 ⟨proof⟩

**lemma** *sorted-injection-subset*:  
 assumes  $A' \subseteq_m A$   
 shows  $\text{sorted-injection } \eta \ A'$   
 ⟨proof⟩

**end**

**lemma** *sorted-injection-cong*:  
 $(\bigwedge a. a \in \text{dom } A \implies f \ a = f' \ a) \implies \text{sorted-injection } f \ A = \text{sorted-injection } f' \ A$   
 ⟨proof⟩

**lemma** *sorted-injection-o*:  
 assumes  $\text{sorted-injection } f \ A$  and  $\text{sorted-injection } g \ (f \ ^s \ A)$   
 shows  $\text{sorted-injection } (g \circ f) \ A$   
 ⟨proof⟩

**lemma** *unsorted-injection[simp]*:  $\text{sorted-injection } f \ (\text{unsorted } A) \longleftrightarrow \text{inj-on } f \ A$   
 ⟨proof⟩

**interpretation** *Inl*:  $\text{sorted-injection } \text{Inl}$   
 ⟨proof⟩

**interpretation** *Inr*:  $\text{sorted-injection } \text{Inr}$   
 ⟨proof⟩

One can define the image of an algebra with respect to an injection.

**locale** *sorted-algebra-injection = source*:  $\text{sorted-algebra} + \text{sorted-injection}$   
**begin**

**sublocale** *sorted-distributive*  $F \ \eta \ A \ I \ \langle \text{image-intp } \eta \ A \ I \rangle$   
 ⟨proof⟩

**lemmas**  $\text{image-sorted-algebra} = \text{image.sorted-algebra-axioms}$

Therefore, one can embed an algebra in a larger carrier if the relevant sorts are preserved.

**lemma** *embed-sorted-algebra*:

**assumes**  $eq: B \uparrow^r \text{ sorts-ssig } F = \eta \text{ }^s A \uparrow^r \text{ sorts-ssig } F$

**shows** *sorted-algebra*  $F B$  (*image-intp*  $\eta A I$ )

*<proof>*

**end**

**end**

## 14 Extension of Algebras

**theory** *Algebra-Extensions*

**imports** *Sorted-Injections*

**begin**

**lemma** *all-set-isl*:  $(\forall a \in \text{set } as. \text{isl } a) \longleftrightarrow (\exists ls. as = \text{map } \text{Inl } ls)$

*<proof>*

**lemma** *not-isl*:  $\neg \text{isl } x \longleftrightarrow (\exists a. x = \text{Inr } a)$

*<proof>*

**lemma** *all-set-not-isl*:  $(\forall a \in \text{set } as. \neg \text{isl } a) \longleftrightarrow (\exists ls. as = \text{map } \text{Inr } ls)$

*<proof>*

**lemma** *map-Inl-eq-map-Inr*:  $\text{map } \text{Inl } xs = \text{map } \text{Inr } ys \longleftrightarrow xs = [] \wedge ys = []$

*<proof>*

**lemma** *map-Inr-eq-map-Inl*:  $\text{map } \text{Inr } ys = \text{map } \text{Inl } xs \longleftrightarrow xs = [] \wedge ys = []$

*<proof>*

**lemmas**  $\text{islE} = \text{isl-def}[\text{THEN } \text{iffD1}, \text{THEN } \text{exE}]$

**lemma** *map-le-map-add2*:

**assumes**  $\text{dom } F \cap \text{dom } G = \{\}$  **shows**  $F \subseteq_m F ++ G$

*<proof>*

**lemma** *hastype-in-add*:

$a : \sigma \text{ in } A ++ B \longleftrightarrow a : \sigma \text{ in } B \vee a \notin \text{dom } B \wedge a : \sigma \text{ in } A$

*<proof>*

**lemma** *fun-hastype-in-add*:

$f : \sigma s \rightarrow \tau \text{ in } F ++ G \longleftrightarrow f : \sigma s \rightarrow \tau \text{ in } G \vee (f, \sigma s) \notin \text{dom } G \wedge f : \sigma s \rightarrow \tau \text{ in } F$

*<proof>*

**lemma** *fun-hastype-in-add-disj*:

**assumes**  $\text{dom } F \cap \text{dom } G = \{\}$   
**shows**  $f : \sigma s \rightarrow \tau \text{ in } F \text{ ++ } G \longleftrightarrow f : \sigma s \rightarrow \tau \text{ in } G \vee f : \sigma s \rightarrow \tau \text{ in } F$   
 $\langle \text{proof} \rangle$

**lemma** *hastype-in-imageD*:  
**assumes**  $fa : \sigma \text{ in } f \text{ }^{\text{cs}} A$   
**shows**  $\exists a : \sigma \text{ in } A. fa = f a$   
 $\langle \text{proof} \rangle$

**lemma** *hastype-list-in-imageD*:  
**assumes**  $fas :_{\text{l}} \sigma s \text{ in } f \text{ }^{\text{cs}} A$   
**shows**  $\exists as. as :_{\text{l}} \sigma s \text{ in } A \wedge fas = \text{map } f as$   
 $\langle \text{proof} \rangle$

**lemmas** *hastype-list-in-imageE* =  
*hastype-list-in-imageD*[*THEN* *exE*, *unfolded conj-imp-eq-imp-imp*]

**lemma** *all-set-hastype-in-imageD*:  
**assumes**  $\forall fa \in \text{set } fas. fa : \sigma \text{ in } f \text{ }^{\text{cs}} A$   
**shows**  $\exists as. (\forall a \in \text{set } as. a : \sigma \text{ in } A) \wedge fas = \text{map } f as$   
 $\langle \text{proof} \rangle$

**lemmas** *all-set-hastype-in-imageE* =  
*all-set-hastype-in-imageD*[*THEN* *exE*, *unfolded conj-imp-eq-imp-imp*]

## 14.1 Disjoint Sum of Sorted Sets

The disjoint sum of two sorted sets  $A$  and  $B$  are just *case-sum*  $A B$ .

**lemma** *Inl-hastype[simp]*:  $\text{Inl } a : \sigma \text{ in case-sum } A B \longleftrightarrow a : \sigma \text{ in } A$   
**and** *Inr-hastype[simp]*:  $\text{Inr } b : \sigma \text{ in case-sum } A B \longleftrightarrow b : \sigma \text{ in } B$   
 $\langle \text{proof} \rangle$

**lemma** *hastype-in-case-sum*:  $x : \sigma \text{ in case-sum } A B \longleftrightarrow$   
 $(\exists a. x = \text{Inl } a \wedge a : \sigma \text{ in } A) \vee (\exists b. x = \text{Inr } b \wedge b : \sigma \text{ in } B)$   
 $\langle \text{proof} \rangle$

## 14.2 Extending Signature and Interpretations

We merge an  $F$ -interpretation  $I$  with a  $G$ -interpretation  $J$  into an  $(F + G)$ -interpretation. To this end, we test if  $I$  is applicable to given function symbol and arguments, in which case apply  $I$  and otherwise apply  $J$ .

**definition** *interpretable*  $F A f as \equiv$   
*case those* ( $\text{map } A as$ ) of *Some*  $\sigma s \Rightarrow (f, \sigma s) \in \text{dom } F \mid \text{None} \Rightarrow \text{False}$

**lemma** *interpretableI*:  
**assumes**  $f : \sigma s \rightarrow \tau \text{ in } F$  **and**  $as :_{\text{l}} \sigma s \text{ in } A$  **shows** *interpretable*  $F A f as$   
 $\langle \text{proof} \rangle$

**lemma** *interpretableE*:

**assumes** *interpretable F A f as*

**and**  $\bigwedge \sigma s \tau. f : \sigma s \rightarrow \tau \text{ in } F \implies as :_l \sigma s \text{ in } A \implies thesis$

**shows** *thesis*

*<proof>*

**lemma** *interpretable-iff*:

**assumes**  $as :_l \sigma s \text{ in } A$

**shows**  $interpretable F A f as \iff (f, \sigma s) \in dom F$

*<proof>*

**definition** *extend-intp F A I J f as*  $\equiv$

*if interpretable F A f as then I f as else J f as*

**lemma** *extend-intp1*:

**assumes**  $\neg interpretable F A f as$

**shows**  $extend-intp F A I J f as = J f as$

*<proof>*

**lemma** *extend-intp2*:

**assumes**  $f : \sigma s \rightarrow \tau \text{ in } F$  **and**  $as :_l \sigma s \text{ in } A$

**shows**  $extend-intp F A I J f as = I f as$

*<proof>*

**lemma** *interpretable-sorts-ssig*:  $interpretable F (A \uparrow^r \text{sorts-ssig } F) = interpretable F A$

*<proof>*

**lemma** *extend-intp-sorts-ssig*:  $extend-intp F (A \uparrow^r \text{sorts-ssig } F) = extend-intp F A$

*<proof>*

**lemma** *extend-intp-cong*:  $F = F' \implies A \uparrow^r \text{sorts-ssig } F' = A' \uparrow^r \text{sorts-ssig } F' \implies extend-intp F A = extend-intp F' A'$

*<proof>*

**definition** *extend-image-intp  $\eta$  F A I*  $\equiv extend-intp F (\eta \text{ } ^{as} A)$  (*image-intp  $\eta$  A I*)

**lemma** *extend-image-intp1*:

**assumes**  $\neg interpretable F (\eta \text{ } ^{as} A) f as$

**shows**  $extend-image-intp \eta F A I J f as = J f as$

*<proof>*

**lemma**(*in sorted-injection*) *extend-image-intp2*:

**assumes**  $f : \sigma s \rightarrow \tau \text{ in } F$  **and**  $as :_l \sigma s \text{ in } A$

**shows**  $extend-image-intp \eta F A I J f (map \eta as) = \eta (I f as)$

*<proof>*

**context** *sorted-algebra* **begin**

**lemma** *extend-intp-eval1*:

**assumes** *disj*:  $\text{dom } F \cap \text{dom } G = \{\}$   
**and**  $s : s : \sigma \text{ in } \mathcal{T}(F, V)$   
**and**  $\alpha : \alpha :_s V \rightarrow A$   
**shows** *extend-intp*  $G A J I \llbracket s \rrbracket \alpha = I \llbracket s \rrbracket \alpha$   
 $\langle \text{proof} \rangle$

**lemma** *extend-intp-eval2*:

**assumes**  $s : s : \sigma \text{ in } \mathcal{T}(F, V)$   
**and**  $\alpha : \alpha :_s V \rightarrow A$   
**shows** *extend-intp*  $F A I J \llbracket s \rrbracket \alpha = I \llbracket s \rrbracket \alpha$   
 $\langle \text{proof} \rangle$

One can extend the signature by merging interpretations.

**lemma** *extend-sorted-algebra*:

**assumes** *GAJ*: *sorted-algebra*  $G A J$   
**shows** *sorted-algebra*  $(G++F) A$  (*extend-intp*  $F A I J$ )  
 $\langle \text{proof} \rangle$

**end**

**context** *sorted-algebra-injection* **begin**

**lemma** *extend-image-intp-eval2*:

**assumes**  $s : s : \sigma \text{ in } \mathcal{T}(F, V)$  **and**  $\alpha : \alpha :_s V \rightarrow A$   
**shows** *extend-image-intp*  $\eta F A I J \llbracket s \rrbracket (\eta \circ \alpha) = \eta (I \llbracket s \rrbracket \alpha)$   
 $\langle \text{proof} \rangle$

In combination, one can extend algebra after injecting over a larger carrier.

**lemma** *extend-image-sorted-algebra*:

**assumes** *GBJ*: *sorted-algebra*  $G B J$   
**and**  $BA : B \uparrow^r \text{ sorts-ssig } F = \eta \text{ }^{\text{as}} A \uparrow^r \text{ sorts-ssig } F$   
**shows** *sorted-algebra*  $(G++F) B$  (*extend-image-intp*  $\eta F A I J$ )  
 $\langle \text{proof} \rangle$

**end**

**end**

## 15 Extending Algebra into Logic

**theory** *Logic-Extensions*

**imports** *Logic Algebra-Extensions*

**begin**

Let us define the image under  $f$  of a relation  $R$  by  $\{(f x, f y) \mid (x, y) \in R\}$ .

**definition**  $image-rel\ f\ R \equiv map-prod\ f\ f\ 'R$

**definition**  $image-relp\ f\ r\ a\ b \equiv \exists x\ y. r\ x\ y \wedge a = f\ x \wedge b = f\ y$

**lemma**  $image-rel: (a,b) \in image-rel\ f\ R \longleftrightarrow (\exists (x,y) \in R. a = f\ x \wedge b = f\ y)$   
*<proof>*

**lemma**  $image-relp: image-relp\ f\ r\ a\ b \longleftrightarrow (\exists x\ y. r\ x\ y \wedge a = f\ x \wedge b = f\ y)$   
*<proof>*

**lemma**  $image-relp-eq-range: image-relp\ f\ (=) = (=) \upharpoonright range\ f$   
*<proof>*

**lemma**  $in-rel-image: in-rel\ (image-rel\ f\ R) = image-relp\ f\ (in-rel\ R)$   
*<proof>*

**context** *quasi-logic begin*

The injective image of a quasi logic is a quasi logic with respect to the image relation.

**lemma** *image-quasi-logic:*

**assumes**  $inj-on\ \eta\ (dom\ A)$

**shows**  $quasi-logic\ F\ (\eta\ ^{as}\ A)\ (image-intp\ \eta\ A\ I)\ (image-relp\ \eta\ ((\sim) \upharpoonright dom\ A))$   
 $boolS\ conjF\ disjF\ negF$

*<proof>*

One can extend the carrier of logic if relevant sorts are preserved.

**lemma** *quasi-logic-carrier:*

**assumes**  $B \upharpoonright^r\ sorts-ssig\ F = A \upharpoonright^r\ sorts-ssig\ F$

**shows**  $quasi-logic\ F\ B\ I\ (\sim)\ boolS\ conjF\ disjF\ negF$

*<proof>*

Therefore, one can embed logic by injection.

**lemma** *embed-quasi-logic:*

**assumes**  $\eta: inj-on\ \eta\ (dom\ A)$

**and**  $BA: B \upharpoonright^r\ sorts-ssig\ F = \eta\ ^{as}\ A \upharpoonright^r\ sorts-ssig\ F$

**shows**  $quasi-logic\ F\ B\ (image-intp\ \eta\ A\ I)\ (image-relp\ \eta\ ((\sim) \upharpoonright dom\ A))\ boolS$   
 $conjF\ disjF\ negF$

*<proof>*

One can extend the signature and interpretation just as algebras.

**lemma**  $extend-true[simp]: extend-intp\ F\ A\ I\ J\ conjF\ [] = true$   
*<proof>*

**lemma**  $extend-false[simp]: extend-intp\ F\ A\ I\ J\ disjF\ [] = false$   
*<proof>*

**lemma**  $extend-image-intp-true[simp]: extend-image-intp\ f\ F\ A\ I\ J\ conjF\ [] = f\ true$

*<proof>*

**lemma** *extend-image-intp-false*[simp]: *extend-image-intp f F A I J disjF [] = f false*  
*<proof>*

**lemma** *extend-quasi-logic*:

**assumes** *J: sorted-algebra G A J*

**shows** *quasi-logic (G++F) A (extend-intp F A I J) (~) boolS conjF disjF negF*  
*<proof>*

Finally, one can extend the signature, interpretation and carrier after embedding.

**lemma** *extend-image-quasi-logic*:

**assumes** *J: sorted-algebra G B J*

**and** *η: inj-on η (dom A)*

**and** *BA: B  $\uparrow^r$  sorts-ssig F = η <sup>as</sup> A  $\uparrow^r$  sorts-ssig F*

**shows** *quasi-logic (G++F) B (extend-image-intp η F A I J) (image-relp η ((~)  $\uparrow$  dom A)) boolS conjF disjF negF*  
*<proof>*

**end**

**context** *logic begin*

**lemmas** *logic-carrier = quasi-logic-carrier*[folded *logic-def*]

**lemmas** *extend-logic = extend-quasi-logic*[folded *logic-def*]

**lemma** *image-logic*:

**assumes** *inj-on η (dom A)*

**shows** *logic F (η <sup>as</sup> A) (image-intp η A I) boolS conjF disjF negF*

*<proof>*

**lemma** *extend-image-logic*:

**assumes** *J: sorted-algebra G B J*

**and** *η: inj-on η (dom A)*

**and** *BA: B  $\uparrow^r$  sorts-ssig F = η <sup>as</sup> A  $\uparrow^r$  sorts-ssig F*

**shows** *logic (G++F) B (extend-image-intp η F A I J) boolS conjF disjF negF*

*<proof>*

**end**

**end**

## 16 Concrete Logics

### 16.1 Bool Logic

Here we define the core logic, that can be used to “logicalize” algebras. To be able to extend easily, we fix types of symbols to strings.

**theory** *Core-Logic*

**imports** *Logic-Extensions*

**begin**

**definition** *bool-s* (*bool<sub>s</sub>*) **where** *bool<sub>s</sub>*  $\equiv$  "Bool"

**definition** *conj-f* (*conj<sub>f</sub>*) **where** *conj<sub>f</sub>*  $\equiv$  "and"

**definition** *disj-f* (*disj<sub>f</sub>*) **where** *disj<sub>f</sub>*  $\equiv$  "or"

**definition** *neg-f* (*neg<sub>f</sub>*) **where** *neg<sub>f</sub>*  $\equiv$  "not"

**interpretation** *Bool*: *logic-signature-syntax bool<sub>s</sub> conj<sub>f</sub> disj<sub>f</sub> neg<sub>f</sub>*⟨*proof*⟩

**interpretation** *Bool*: *truth-syntax conj<sub>f</sub>*⟨*proof*⟩

**notation** *Bool.trueT* (*True<sub>t</sub>*)

**interpretation** *Bool*: *logic-symbols-syntax conj<sub>f</sub> disj<sub>f</sub> neg<sub>f</sub>*⟨*proof*⟩

**notation** *Bool.falseT* (*False<sub>t</sub>*)

**interpretation** *Bool*: *logic-signature bool<sub>s</sub> conj<sub>f</sub> disj<sub>f</sub> neg<sub>f</sub>*  
⟨*proof*⟩

**thm** *Bool.sorts-ssig-F*

Any algebra over strings can be “logicalized”, if precisely truth values have bool sort.

**abbreviation** *logicalize-intp*  $\equiv$  *extend-image-intp Inl Bool.F Bool.A Bool.I*

**thm** *Bool.extend-image-logic*[*OF - inj-Inl*]

**lemma** *logicalize*:

**assumes** *FAI*: *sorted-algebra F A I* **and** *b*:  $\{a. a : \text{bool}_s \text{ in } A\} = \text{range Inl}$

**shows** *logic* (*F++Bool.F*) *A* (*logicalize-intp I*) *bool<sub>s</sub> conj<sub>f</sub> disj<sub>f</sub> neg<sub>f</sub>*  
⟨*proof*⟩

**end**

### 16.2 Natural Arithmetic

**theory** *Nats-Logic*

**imports** *Core-Logic*

**begin**

The natural number algebra consists of zero and the successor operator as

constructors, and some defined operators and relations.

**definition** *nat-s* ( $nat_s$ ) **where**  $nat_s \equiv "nat"$

**definition** *zero-f* ( $0_f$ ) **where**  $0_f \equiv "0"$

**definition** *Suc-f* ( $Suc_f$ ) **where**  $Suc_f \equiv "Suc"$

**definition** *add-f* ( $+_f$ ) **where**  $+_f \equiv "+"$

**definition** *mult-f* ( $*_f$ ) **where**  $*_f \equiv "*"$

**definition** *less-f* ( $<_f$ ) **where**  $<_f \equiv "<"$

**lemma** *bool-neq-nat[simp]*:  $bool_s \neq nat_s$

*<proof>*

**lemma** *Nat-syms-neq[simp]*:

$0_f \neq Suc_f$   $0_f \neq +_f$   $0_f \neq *_f$   $0_f \neq <_f$   $0_f \neq conj_f$   $0_f \neq disj_f$   $0_f \neq neg_f$

$Suc_f \neq +_f$   $Suc_f \neq *_f$   $Suc_f \neq <_f$   $Suc_f \neq conj_f$   $Suc_f \neq disj_f$   $Suc_f \neq neg_f$

$+_f \neq <_f$   $+_f \neq *_f$   $+_f \neq conj_f$   $+_f \neq disj_f$   $+_f \neq neg_f$

$*_f \neq <_f$   $*_f \neq conj_f$   $*_f \neq disj_f$   $*_f \neq neg_f$

$<_f \neq conj_f$   $<_f \neq disj_f$   $<_f \neq neg_f$

*<proof>*

**lemmas** [*simp*] = *Nat-syms-neq[symmetric]*

**abbreviation** *zero-t* ( $0_t$ ) **where**  $0_t \equiv Fun\ 0_f\ []$

**abbreviation** *suc-t* ( $Suc_t$ ) **where**  $Suc_t\ s \equiv Fun\ Suc_f\ [s]$

**abbreviation** *add-t* (**infix**  $+_t$  70) **where**  $s +_t t \equiv Fun\ +_f\ [s,t]$

**abbreviation** *mult-t* (**infix**  $*_t$  70) **where**  $s *_t t \equiv Fun\ *_f\ [s,t]$

**abbreviation** *less-t* (**infix**  $<_t$  50) **where**  $s <_t t \equiv Fun\ <_f\ [s,t]$

**definition** *NatC*  $\equiv [(0_f, []) \mapsto nat_s, (Suc_f, [nat_s]) \mapsto nat_s]$

**abbreviation** *NatBoolC*  $\equiv NatC\ ++\ Bool.C$

**definition** *NatD*  $\equiv [(+_f, nat_s..) \mapsto nat_s, (*_f, nat_s..) \mapsto nat_s, (<_f, [nat_s, nat_s]) \mapsto bool_s]$

**abbreviation** *NatF*  $\equiv NatC\ ++\ NatD$

**abbreviation** *NatBoolF*  $\equiv NatF\ ++\ Bool.F$

**lemma** *hastype-in-NatC*:  $f : \sigma s \rightarrow \tau$  in *NatC*  $\longleftrightarrow$

$f = 0_f \wedge \sigma s = [] \wedge \tau = nat_s \vee$

$f = Suc_f \wedge \sigma s = [nat_s] \wedge \tau = nat_s$  *<proof>*

**lemma** *hastype-in-NatC-E*:

**assumes**  $f : \sigma s \rightarrow \tau$  in *NatC*

**and**  $f = 0_f \implies \sigma s = [] \implies \tau = nat_s \implies thesis$

**and**  $f = Suc_f \implies \sigma s = [nat_s] \implies \tau = nat_s \implies thesis$

**shows** *thesis*

*<proof>*

**lemma** *hastype-in-NatC-simps*[simp]:

$$0_f : \sigma s \rightarrow \tau \text{ in } \text{NatC} \longleftrightarrow \sigma s = [] \wedge \tau = \text{nat}_s$$

$$\text{Suc}_f : \sigma s \rightarrow \tau \text{ in } \text{NatC} \longleftrightarrow \sigma s = [\text{nat}_s] \wedge \tau = \text{nat}_s$$

*<proof>*

**lemma** *Fun-hastype-in-NatC*:

$$\text{shows [simp]: } \text{Fun } 0_f \text{ } ss : \tau \text{ in } \mathcal{T}(\text{NatC}, V) \longleftrightarrow ss = [] \wedge \tau = \text{nat}_s$$

$$\text{and Fun Suc}_f \text{ } ss : \tau \text{ in } \mathcal{T}(\text{NatC}, V) \longleftrightarrow$$

$$\tau = \text{nat}_s \wedge (\exists s. ss = [s] \wedge s : \text{nat}_s \text{ in } \mathcal{T}(\text{NatC}, V))$$

*<proof>*

**lemma** *Suc-t-hastype-in-NatC*[simp]:

$$\text{Suc}_t \text{ } s : \sigma \text{ in } \mathcal{T}(\text{NatC}, V) \longleftrightarrow \sigma = \text{nat}_s \wedge s : \text{nat}_s \text{ in } \mathcal{T}(\text{NatC}, V)$$

*<proof>*

**lemma** *dom-NatC*[simp]:  $\text{dom NatC} = \{(0_f, []), (\text{Suc}_f, [\text{nat}_s])\}$

*<proof>*

**lemma** *NatC-le-extended*:  $\text{NatC} \subseteq_m \text{NatC} ++ \text{Bool.C}$

*<proof>*

**lemma** *hastype-in-NatD*:  $f : \sigma s \rightarrow \tau \text{ in } \text{NatD} \longleftrightarrow$

$$(f = +_f \vee f = *_f) \wedge (\forall \sigma \in \text{set } \sigma s. \sigma = \text{nat}_s) \wedge \tau = \text{nat}_s \vee$$

$$f = <_f \wedge \sigma s = [\text{nat}_s, \text{nat}_s] \wedge \tau = \text{bool}_s$$

*<proof>*

**lemma** *hastype-in-NatD-E*:

**assumes**  $f : \sigma s \rightarrow \tau \text{ in } \text{NatD}$

$$\text{and } \bigwedge n. f = +_f \implies \sigma s = \text{replicate } n \text{ } \text{nat}_s \implies \tau = \text{nat}_s \implies \text{thesis}$$

$$\text{and } \bigwedge n. f = *_f \implies \sigma s = \text{replicate } n \text{ } \text{nat}_s \implies \tau = \text{nat}_s \implies \text{thesis}$$

$$\text{and } \bigwedge n. f = <_f \implies \sigma s = [\text{nat}_s, \text{nat}_s] \implies \tau = \text{bool}_s \implies \text{thesis}$$

**shows** *thesis*

*<proof>*

**lemma** *hastype-in-NatD-simps*[simp]:

$$+_f : \sigma s \rightarrow \tau \text{ in } \text{NatD} \longleftrightarrow (\forall \sigma \in \text{set } \sigma s. \sigma = \text{nat}_s) \wedge \tau = \text{nat}_s$$

$$*_f : \sigma s \rightarrow \tau \text{ in } \text{NatD} \longleftrightarrow (\forall \sigma \in \text{set } \sigma s. \sigma = \text{nat}_s) \wedge \tau = \text{nat}_s$$

$$<_f : \sigma s \rightarrow \tau \text{ in } \text{NatD} \longleftrightarrow \sigma s = [\text{nat}_s, \text{nat}_s] \wedge \tau = \text{bool}_s$$

*<proof>*

**lemma** *NatC-NatD*:  $\text{dom NatC} \cap \text{dom NatD} = \{\}$

*<proof>*

**lemmas** *hastype-in-NatF = fun-hastype-in-add-disj*[OF *NatC-NatD*]

**lemma** *hastype-in-NatF-simps*[simp]:

$$0_f : \sigma s \rightarrow \tau \text{ in } \text{NatF} \longleftrightarrow \sigma s = [] \wedge \tau = \text{nat}_s$$

$$\text{Suc}_f : \sigma s \rightarrow \tau \text{ in } \text{NatF} \longleftrightarrow \sigma s = [\text{nat}_s] \wedge \tau = \text{nat}_s$$

$$+_f : \sigma s \rightarrow \tau \text{ in } \text{NatF} \longleftrightarrow (\forall \sigma \in \text{set } \sigma s. \sigma = \text{nat}_s) \wedge \tau = \text{nat}_s$$

$*_f : \sigma s \rightarrow \tau \text{ in } \text{Nat}F \iff (\forall \sigma \in \text{set } \sigma s. \sigma = \text{nat}_s) \wedge \tau = \text{nat}_s$   
 $<_f : \sigma s \rightarrow \tau \text{ in } \text{Nat}F \iff \sigma s = [\text{nat}_s, \text{nat}_s] \wedge \tau = \text{bool}_s$   
 ⟨proof⟩

**lemma** *zero-t-hastype-in-NatF[simp]*:  $0_t : \sigma \text{ in } \mathcal{T}(\text{Nat}F, V) \iff \sigma = \text{nat}_s$   
 ⟨proof⟩

**lemma** *Suc-t-hastype-in-NatF[simp]*:  
 $\text{Suc}_t s : \sigma \text{ in } \mathcal{T}(\text{Nat}F, V) \iff \sigma = \text{nat}_s \wedge s : \text{nat}_s \text{ in } \mathcal{T}(\text{Nat}F, V)$   
 ⟨proof⟩

**lemma** *less-t-hastype-in-NatF[simp]*:  
 $(s <_t t) : \sigma \text{ in } \mathcal{T}(\text{Nat}F, V) \iff$   
 $\sigma = \text{bool}_s \wedge s : \text{nat}_s \text{ in } \mathcal{T}(\text{Nat}F, V) \wedge t : \text{nat}_s \text{ in } \mathcal{T}(\text{Nat}F, V)$   
 ⟨proof⟩

**lemma** *Fun-add-hastype-in-NatF[simp]*:  
 $\text{Fun } +_f ss : \sigma \text{ in } \mathcal{T}(\text{Nat}F, V) \iff$   
 $\sigma = \text{nat}_s \wedge (\forall s \in \text{set } ss. s : \text{nat}_s \text{ in } \mathcal{T}(\text{Nat}F, V))$   
 ⟨proof⟩

**lemma** *Fun-mult-hastype-in-NatF[simp]*:  
 $\text{Fun } *_f ss : \sigma \text{ in } \mathcal{T}(\text{Nat}F, V) \iff$   
 $\sigma = \text{nat}_s \wedge (\forall s \in \text{set } ss. s : \text{nat}_s \text{ in } \mathcal{T}(\text{Nat}F, V))$   
 ⟨proof⟩

**lemma** *NatC-BoolC*:  $\text{dom } \text{Nat}C \cap \text{dom } \text{Bool}.C = \{\}$   
 ⟨proof⟩

**lemmas** *hastype-in-NatBoolC* = *fun-hastype-in-add-disj[OF NatC-BoolC]*

**lemma** *NatF-BoolF*:  $\text{dom } \text{Nat}F \cap \text{dom } \text{Bool}.F = \{\}$   
 ⟨proof⟩

**lemmas** *hastype-in-NatBoolF* = *fun-hastype-in-add-disj[OF NatF-BoolF]*

**fun** *const-of-nat* **where**  
 $\text{const-of-nat } 0 = 0_t$   
 $| \text{const-of-nat } (\text{Suc } n) = \text{Suc}_t (\text{const-of-nat } n)$

**lemma** *const-of-nat-hastype*:  $\text{const-of-nat } n : \text{nat}_s \text{ in } \mathcal{T}(\text{Nat}C, V)$   
 ⟨proof⟩

**abbreviation** *NatBool-const*  $\equiv \text{case-sum } \text{Bool}. \text{const-of-bool } \text{const-of-nat}$

**definition** *NatA*  $(a::\text{nat}) \equiv \text{Some } \text{nat}_s$

**abbreviation** *NatBoolA*  $\equiv \text{case-sum } \text{Bool}.A \text{NatA}$

**lemma** *hastype-in-NatA[simp]*:  $a : s \text{ in } \text{NatA} \longleftrightarrow s = \text{nat}_s$   
 ⟨proof⟩

**lemma** *hastype-in-NatBoolA*:  
 $a : \sigma \text{ in } \text{NatBoolA} \longleftrightarrow (\sigma = \text{bool}_s \wedge a \in \text{range Inl} \vee \sigma = \text{nat}_s \wedge a \in \text{range Inr})$   
 ⟨proof⟩

**definition** *NatI f as*  $\equiv$   
 if  $f = \text{Suc}_f$  then  $\text{Inr} (\text{Suc} (\text{projr} (as!0)))$   
 else if  $f = +_f$  then  $\text{Inr} (\sum a \leftarrow as. \text{projr} a)$   
 else if  $f = *_f$  then  $\text{Inr} (\prod a \leftarrow as. \text{projr} a)$   
 else if  $f = <_f$  then  $\text{Inl} (\text{projr} (as!0) < \text{projr} (as!1))$   
 else  $\text{Inr } 0$

**abbreviation** *NatBoolI*  $\equiv \text{logicalize-intp } \text{NatI}$

**lemma** *NatI-simps*:  
 $\text{NatI } 0_f \text{ as} = \text{Inr } 0$   
 $\text{NatI } \text{Suc}_f \text{ as} = \text{Inr} (\text{Suc} (\text{projr} (as!0)))$   
 $\text{NatI } +_f \text{ as} = \text{Inr} (\sum a \leftarrow as. \text{projr} a)$   
 $\text{NatI } *_f \text{ as} = \text{Inr} (\prod a \leftarrow as. \text{projr} a)$   
 $\text{NatI } <_f \text{ as} = \text{Inl} (\text{projr} (as!0) < \text{projr} (as!1))$   
 ⟨proof⟩

**interpretation** *Nat*: *sorted-algebra* *NatF* *NatBoolA* *NatI*  
 ⟨proof⟩

**interpretation** *Nat-const*: *sorted-algebra* *NatC* *NatBoolA* *NatI*  
 ⟨proof⟩

**lemma** *Nat-eval-const-of-nat[simp]*:  $\text{NatI} [\text{const-of-nat } n] \alpha = \text{Inr } n$   
 ⟨proof⟩

**lemma** *logicalize-intp-Nat[simp]*:  
 $\text{logicalize-intp } I \ 0_f \text{ as} = I \ 0_f \text{ as}$   
 $\text{logicalize-intp } I \ \text{Suc}_f \text{ as} = I \ \text{Suc}_f \text{ as}$   
 $\text{logicalize-intp } I \ +_f \text{ as} = I \ +_f \text{ as}$   
 $\text{logicalize-intp } I \ *_f \text{ as} = I \ *_f \text{ as}$   
 $\text{logicalize-intp } I \ <_f \text{ as} = I \ <_f \text{ as}$   
 ⟨proof⟩

**lemma** *NatBool-eval-const-of-nat*:  $\text{NatBoolI} [\text{const-of-nat } b] \alpha = \text{Inr } b$  ⟨proof⟩

**interpretation** *NatBool*: *logic* *NatBoolF* *NatBoolA* *NatBoolI* *bool\_s* *conj\_f* *disj\_f* *neg\_f*  
 ⟨proof⟩

**interpretation** *NatBool*: *logic-const* *NatBoolF* *NatBoolA* *NatBoolI* *NatBool-const*

```

NatBoolC
  bools conjf disjf negf
  ⟨proof⟩

```

**end**

### 16.3 Integer Arithmetic

```

theory Ints-Logic
  imports Nats-Logic Show.Number-Parser
begin

```

For integer arithmetic, we allow string representations of integers to be constants, i.e., those strings that can be parsed as integers.

```

lemma isl-int-of-string-imp-show-neq:
  assumes isl (int-of-string s) shows show (i::int) ≠ s
  ⟨proof⟩

```

```

definition int-s (ints) where ints ≡ "Int"

```

```

lemma bool-neq-int[simp]: bools ≠ ints
  ⟨proof⟩

```

```

declare bool-neq-int[symmetric,simp]

```

```

lemma isl-int-of-string-less: isl (int-of-string <f)
  and isl-int-of-string-add: isl (int-of-string +f)
  and isl-int-of-string-mult: isl (int-of-string *f)
  and isl-int-of-string-conj: isl (int-of-string conjf)
  and isl-int-of-string-disj: isl (int-of-string disjf)
  ⟨proof⟩

```

```

lemma show-int-neq-less[simp]: show (i::int) ≠ <f
  and show-int-neq-add[simp]: show (i::int) ≠ +f
  and show-int-neq-mult[simp]: show (i::int) ≠ *f
  and show-int-neq-conj[simp]: show (i::int) ≠ conjf
  and show-int-neq-disj[simp]: show (i::int) ≠ disjf
  ⟨proof⟩

```

```

definition IntC ≡ λ(s,σs).
  if σs = [] ∧ int-of-string s ∈ range Inr then Some ints else None

```

```

abbreviation IntBoolC ≡ IntC ++ Bool.C

```

```

definition IntD ≡ [(+f,ints..) ↦ ints, (*f,ints..) ↦ ints, (<f,[ints,ints]) ↦ bools]

```

```

abbreviation IntF ≡ IntC ++ IntD

```

```

abbreviation IntBoolF ≡ IntF ++ Bool.F

```

**lemma** *hastype-in-IntC*:

$s : \sigma s \rightarrow \tau$  in *IntC*  $\longleftrightarrow \sigma s = [] \wedge \tau = int_s \wedge int\text{-of-string } s \in range\ Inr$   
 ⟨proof⟩

**lemma** *IntC-BoolC*:  $dom\ IntC \cap dom\ Bool.C = \{\}$

⟨proof⟩

**lemmas** *hastype-in-IntBoolC* = *fun-hastype-in-add-disj*[*OF IntC-BoolC*]

**lemma** *hastype-in-IntD*:  $f : \sigma s \rightarrow \tau$  in *IntD*  $\longleftrightarrow$

$(f = +_f \vee f = *_f) \wedge (\forall \sigma \in set\ \sigma s. \sigma = int_s) \wedge \tau = int_s \vee$

$f = <_f \wedge \sigma s = [int_s, int_s] \wedge \tau = bool_s$

⟨proof⟩

**lemma** *IntC-IntD*:  $dom\ IntC \cap dom\ IntD = \{\}$

⟨proof⟩

**lemmas** *hastype-in-IntF* = *fun-hastype-in-add-disj*[*OF IntC-IntD*]

**lemma** *IntF-BoolF*:  $dom\ IntF \cap dom\ Bool.F = \{\}$

⟨proof⟩

**lemmas** *hastype-in-IntBoolF* = *fun-hastype-in-add-disj*[*OF IntF-BoolF*]

**definition** *IntA* ( $i::int$ )  $\equiv Some\ int_s$

**abbreviation** *IntBoolA*  $\equiv case\text{-sum}\ Bool.A\ IntA$

**lemma** *hastype-in-IntA*[*simp*]:  $a : \sigma$  in *IntA*  $\longleftrightarrow \sigma = int_s$

⟨proof⟩

**definition** *IntI*  $:: string \Rightarrow (bool+int)\ list \Rightarrow bool+int$  **where** *IntI*  $f\ as \equiv$

*if*  $f = +_f$  *then*  $Inr\ (\sum a \leftarrow as.\ projr\ a)$

*else if*  $f = *_f$  *then*  $Inr\ (\prod a \leftarrow as.\ projr\ a)$

*else if*  $f = <_f$  *then*  $Inl\ (projr\ (as!0) < projr\ (as!1))$

*else case* *int-of-string*  $f$  *of*  $Inr\ i \Rightarrow Inr\ i$

**abbreviation** *IntBoolI*  $\equiv logicalize\text{-intp}\ IntI$

**lemma** *IntI-simps*:

*int-of-string*  $s = Inr\ i \implies IntI\ s\ as = Inr\ i$

$IntI\ +_f\ as = Inr\ (\sum a \leftarrow as.\ projr\ a)$

$IntI\ *_f\ as = Inr\ (\prod a \leftarrow as.\ projr\ a)$

$IntI\ <_f\ as = Inl\ (projr\ (as!0) < projr\ (as!1))$

⟨proof⟩

**interpretation** *Int*: *sorted-algebra* *IntF* *IntBoolA* *IntI*

⟨proof⟩

**interpretation** *Int.const*: sorted-algebra *IntC IntBoolA IntI*  
<proof>

**abbreviation** *const-of-int i*  $\equiv$  *Fun (show i)* []

**abbreviation** *IntBool-const*  $\equiv$  case-sum *Bool.const-of-bool const-of-int*

**lemma** *IntBoolI-int-of-string*:  
assumes *int-of-string f = Inr i*  
shows *IntBoolI f* [] = *Inr i*  
<proof>

**lemma** *IntBoolI-show[simp]*:  
*IntBoolI (show i)* [] = *Inr i*  
<proof>

**interpretation** *IntBool*: logic *IntBoolF IntBoolA IntBoolI bool<sub>s</sub> conj<sub>f</sub> disj<sub>f</sub> neg<sub>f</sub>*  
<proof>

**interpretation** *IntBool*:  
logic-const *IntBoolF IntBoolA IntBoolI IntBool-const IntBoolC bool<sub>s</sub> conj<sub>f</sub> disj<sub>f</sub> neg<sub>f</sub>*  
<proof>

end

## 17 Examples

**theory** *Test-Sorted-Rewriting*  
imports *Conditional-Rewriting Constrained-Rewriting Ints-Logic*  
begin

**definition** *min-f (min<sub>f</sub>)* **where** *min<sub>f</sub>*  $\equiv$  "min"

**definition** *even-f (even<sub>f</sub>)* **where** *even<sub>f</sub>*  $\equiv$  "even"

**lemma** *syms-neq[simp]*: *0<sub>f</sub>  $\neq$  min<sub>f</sub> 0<sub>f</sub>  $\neq$  even<sub>f</sub>*  
*Suc<sub>f</sub>  $\neq$  min<sub>f</sub> Suc<sub>f</sub>  $\neq$  even<sub>f</sub>*  
*+<sub>f</sub>  $\neq$  min<sub>f</sub> +<sub>f</sub>  $\neq$  even<sub>f</sub>*  
*\*<sub>f</sub>  $\neq$  min<sub>f</sub> \*<sub>f</sub>  $\neq$  even<sub>f</sub>*  
*<<sub>f</sub>  $\neq$  min<sub>f</sub> <<sub>f</sub>  $\neq$  even<sub>f</sub>*  
*min<sub>f</sub>  $\neq$  even<sub>f</sub> min<sub>f</sub>  $\neq$  conj<sub>f</sub> min<sub>f</sub>  $\neq$  disj<sub>f</sub> min<sub>f</sub>  $\neq$  neg<sub>f</sub>*  
*even<sub>f</sub>  $\neq$  conj<sub>f</sub> even<sub>f</sub>  $\neq$  disj<sub>f</sub> even<sub>f</sub>  $\neq$  neg<sub>f</sub>*  
<proof>

**lemmas** [*simp*] = *syms-neq[symmetric]*

## 17.1 Less-Than TRS

**lemma**

$(s +_t t) : nat_s \text{ in } \mathcal{T}(\text{NatF}, V) \longleftrightarrow$   
 $s : nat_s \text{ in } \mathcal{T}(\text{NatF}, V) \wedge t : nat_s \text{ in } \mathcal{T}(\text{NatF}, V)$   
 $(s <_t t) : bool_s \text{ in } \mathcal{T}(\text{NatF}, V) \longleftrightarrow$   
 $s : nat_s \text{ in } \mathcal{T}(\text{NatF}, V) \wedge t : nat_s \text{ in } \mathcal{T}(\text{NatF}, V)$   
 ⟨proof⟩

**abbreviation**  $lt0 \equiv [x'' \mapsto nat_s]. (Var\ x'' <_t 0_t) \rightsquigarrow False_t$

**abbreviation**  $ltSuc \equiv [y'' \mapsto nat_s]. (0_t <_t Suc_t (Var\ y'')) \rightsquigarrow True_t$

**abbreviation**  $ltSucSuc \equiv [x'' \mapsto nat_s, y'' \mapsto nat_s].$   
 $(Suc_t (Var\ x'') <_t Suc_t (Var\ y'')) \rightsquigarrow (Var\ x'' <_t Var\ y'')$

**definition**  $Rlt (\mathcal{R}_<)$  where  $\mathcal{R}_< \equiv \{lt0, ltSuc, ltSucSuc\}$

**lemma**  $Rlt$ :  $lt0 \in \mathcal{R}_< \quad ltSuc \in \mathcal{R}_< \quad ltSucSuc \in \mathcal{R}_<$   
 ⟨proof⟩

**interpretation**  $Rlt$ : *sorted-trs*  $\text{NatBoolF } \mathcal{R}_<$   
 ⟨proof⟩

**lemma**  $(0_t <_t 0_t) - \text{NatBoolF}: V: \mathcal{R}_< \rightarrow^\varepsilon False_t$   
 ⟨proof⟩

**lemma**  $(0_t <_t Suc_t 0_t) - \text{NatBoolF}: V: \mathcal{R}_< \rightarrow^\varepsilon True_t$   
 ⟨proof⟩

## 17.2 Addition TRS

**abbreviation**  $add0 \equiv [x'' \mapsto nat_s]. 0_t +_t Var\ x'' \rightsquigarrow Var\ x''$

**abbreviation**  $addSuc \equiv$   
 $[x'' \mapsto nat_s, y'' \mapsto nat_s].$   
 $Suc_t (Var\ x'') +_t Var\ y'' \rightsquigarrow Suc_t (Var\ x'' +_t Var\ y'')$

**definition**  $Radd (\mathcal{R}_+)$  where  $\mathcal{R}_+ \equiv \{add0, addSuc\}$

**lemma**  $Radd$ :  $add0 \in \mathcal{R}_+ \quad addSuc \in \mathcal{R}_+$   
 ⟨proof⟩

**interpretation**  $Radd$ : *sorted-trs*  $\text{NatF } \mathcal{R}_+$   
 ⟨proof⟩

**lemma**  $Suc0\text{-add}$ :  
**assumes**  $s : s : nat_s \text{ in } \mathcal{T}(\text{NatF}, V)$   
**shows**  $Suc_t 0_t +_t s - \text{NatF}: V: \mathcal{R}_+ \rightarrow^{\hat{2}} Suc_t s$   
 ⟨proof⟩

**lemma** *SucSuc0-add*:

**assumes**  $s : nat_s$  in  $\mathcal{T}(NatF, V)$

**shows**  $Suc_t (Suc_t 0_t) +_t s - NatF : V : \mathcal{R}_+ \rightarrow \hat{\sim} 3 Suc_t (Suc_t s)$

$\langle proof \rangle$

### 17.3 Even TRS

**definition** *Even*  $\equiv [$

$(conj_f, []) \mapsto bool_s,$

$(disj_f, []) \mapsto bool_s,$

$(0_f, []) \mapsto nat_s,$

$(Suc_f, [nat_s]) \mapsto nat_s,$

$(even_f, [nat_s]) \mapsto bool_s]$

**lemma** *Even[simp]*:

$conj_f : \sigma s \rightarrow \tau$  in *Even*  $\longleftrightarrow \sigma s = [] \wedge \tau = bool_s$

$disj_f : \sigma s \rightarrow \tau$  in *Even*  $\longleftrightarrow \sigma s = [] \wedge \tau = bool_s$

$0_f : \sigma s \rightarrow \tau$  in *Even*  $\longleftrightarrow \sigma s = [] \wedge \tau = nat_s$

$Suc_f : \sigma s \rightarrow \tau$  in *Even*  $\longleftrightarrow \sigma s = [nat_s] \wedge \tau = nat_s$

$even_f : \sigma s \rightarrow \tau$  in *Even*  $\longleftrightarrow \sigma s = [nat_s] \wedge \tau = bool_s$

$\langle proof \rangle$

**abbreviation** *even-t* ( $even_t$ ) **where**  $even_t s \equiv Fun\ even_f [s]$

**abbreviation** *axiom-even0*  $\equiv \emptyset. even_t 0_t \rightsquigarrow True_t$

**abbreviation** *axiom-even1*  $\equiv \emptyset. even_t (Suc_t 0_t) \rightsquigarrow False_t$

**abbreviation** *axiom-evenSucSuc*  $\equiv [x'' \mapsto nat_s].$

$even_t (Suc_t (Suc_t (Var\ ''x''))) \rightsquigarrow even_t (Var\ ''x'')$

**definition** *Reven*  $\equiv \{ axiom-even0, axiom-even1, axiom-evenSucSuc \}$

**lemma** *Reven*:  $axiom-even0 \in Reven\ axiom-even1 \in Reven\ axiom-evenSucSuc \in Reven$

$\langle proof \rangle$

**lemma** *evenSucSucSuc*:  $even_t (Suc_t (Suc_t (Suc_t 0_t))) - Feven : V : Reven \rightarrow \hat{\sim} 2 False_t$

$\langle proof \rangle$

### 17.4 Even CTRS

**abbreviation** *rule-evenSucFalse*  $\equiv [x'' \mapsto nat_s].$

$even_t (Suc_t (Var\ ''x'')) \rightsquigarrow False_t \Leftarrow [even_t (Var\ ''x'') \rightsquigarrow True_t]$

**abbreviation** *rule-evenSucTrue*  $\equiv [x'' \mapsto nat_s].$

$even_t (Suc_t (Var\ ''x'')) \rightsquigarrow True_t \Leftarrow [even_t (Var\ ''x'') \rightsquigarrow False_t]$

**definition**  $RevenC \equiv \{\text{unconditional axiom-even}0, \text{rule-evenSucFalse}, \text{rule-evenSucTrue}\}$

**lemma**  $RevenC$ :

$\text{unconditional axiom-even}0 \in RevenC$   
 $\text{rule-evenSucFalse} \in RevenC$   
 $\text{rule-evenSucTrue} \in RevenC$   
 $\langle \text{proof} \rangle$

**interpretation**  $RevenC$ : *sorted-ctrs*  $Feven$   $RevenC$

$\langle \text{proof} \rangle$

**lemma**  $\text{even}_t (\text{Suc}_t (\text{Suc}_t (\text{Suc}_t 0_t))) = \text{Feven} : V : RevenC \Rightarrow^\varepsilon \text{False}_t$

$\langle \text{proof} \rangle$

## 17.5 LCTRS

**definition**  $Fmin \equiv \text{NatBoolF} ++ [(min_f, [nat_s, nat_s]) \mapsto nat_s]$

**lemma**  $Fmin\text{-disj}$ :  $\text{dom} (\text{NatF} ++ \text{Bool.F}) \cap \text{dom} [(min_f, [nat_s, nat_s]) \mapsto nat_s] = \{\}$

$\langle \text{proof} \rangle$

**lemmas**  $\text{hastype-in-Fmin} = \text{fun-hastype-in-add-disj}[\text{OF } Fmin\text{-disj}, \text{folded } Fmin\text{-def}]$

**lemma**  $\text{min-hastype-in-Fmin}[\text{simp}]$ :  $\text{min}_f : \sigma s \rightarrow \tau$  in  $Fmin \longleftrightarrow \sigma s = [nat_s, nat_s]$

$\wedge \tau = nat_s$

$\langle \text{proof} \rangle$

**abbreviation**  $\text{lcrule-min1} \equiv [x'' \mapsto nat_s, y'' \mapsto nat_s]$ .

$\text{Fun } \text{min}_f [\text{Var } x'', \text{Var } y''] \rightsquigarrow \text{Var } x'' \mid \text{Var } x'' <_t \text{Var } y''$

**abbreviation**  $\text{lcrule-min2} \equiv [x'' \mapsto nat_s, y'' \mapsto nat_s]$ .

$\text{Fun } \text{min}_f [\text{Var } x'', \text{Var } y''] \rightsquigarrow \text{Var } y'' \mid \text{Var } y'' <_t \text{Var } x''$

**definition**  $\text{lctrs-min} \equiv \{\text{lcrule-min1}, \text{lcrule-min2}\}$

**lemma**  $\text{lctrs-min}$ :  $\text{lcrule-min1} \in \text{lctrs-min}$   $\text{lcrule-min2} \in \text{lctrs-min}$

$\langle \text{proof} \rangle$

Deriving calculation TRS.

**interpretation**  $\text{NatBool}$ : *sorted-algebra-calculation*  $\text{NatBoolF}$   $\text{NatBoolA}$   $\text{NatBoolI}$   $\text{NatBool-const}$   $\text{NatBoolC}$   $\langle \text{proof} \rangle$

**interpretation**  $Fmin$ : *lcrule-syntax*  $Fmin$   $\text{NatBoolF}$   $\text{bool}_s$   $\langle \text{proof} \rangle$

**interpretation**  $\text{lctrs-min}$ : *lctrs-semantics*  $Fmin$   $\text{lctrs-min}$   $\text{NatBoolF}$   $\text{NatBoolA}$   $\text{NatBoolI}$   $\text{NatBool-const}$   $\text{NatBoolC}$   $\text{bool}_s$   $\text{conj}_f$

$\langle \text{proof} \rangle$

**lemma**  $\text{Fun } \text{min}_f \text{ [Suc}_t \ 0_t, \ 0_t] - \text{Fmin}: V:\text{ltrs-min.as-trs} \rightarrow 0_t$   
*<proof>*

Demonstrating a calculation step:

**lemma**  $\text{Suc}_t \ 0_t +_t \ 0_t - \text{Fmin}: V:\text{NatBool.calculation-trs} \rightarrow \text{Suc}_t \ 0_t$   
*<proof>*

**end**

## References

- [1] J. C. Blanchette, A. Popescu, and D. Traytel. Abstract completeness. *Archive of Formal Proofs*, April 2014. [https://isa-afp.org/entries/Abstract\\_Completeness.html](https://isa-afp.org/entries/Abstract_Completeness.html), Formal proof development.
- [2] C. Kop and N. Nishida. Term rewriting with logical constraints. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2013.
- [3] C. Sternagel and R. Thiemann. First-order terms. *Archive of Formal Proofs*, February 2018. [https://isa-afp.org/entries/First\\_Order\\_Terms.html](https://isa-afp.org/entries/First_Order_Terms.html), Formal proof development.
- [4] R. Thiemann and C. Sternagel. Certification of Termination Proofs Using CeTA. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.
- [5] V. van Oostrom. Sub-birkhoff. In Y. Kameyama and P. J. Stuckey, editors, *7th International Symposium on Functional and Logic Programming, FLOPS 2004*, volume 2998 of *LNCS*, pages 180–195. Springer, 2004.
- [6] A. Yamada and R. Thiemann. Sorted terms. *Archive of Formal Proofs*, May 2024. [https://isa-afp.org/entries/Sorted\\_Terms.html](https://isa-afp.org/entries/Sorted_Terms.html), Formal proof development.