

# A Formalization of Weighted Path Orders and Recursive Path Orders\*

Christian Sternagel    René Thiemann    Akihisa Yamada

November 24, 2021

## Abstract

We define the weighted path order (WPO) and formalize several properties such as strong normalization, the subterm property, and closure properties under substitutions and contexts. Our definition of WPO extends the original definition by also permitting multiset comparisons of arguments instead of just lexicographic extensions. Therefore, our WPO not only subsumes lexicographic path orders (LPO), but also recursive path orders (RPO). We formally prove these subsumptions and therefore all of the mentioned properties of WPO are automatically transferable to LPO and RPO as well. Such a transformation is not required for Knuth–Bendix orders (KBO), since they have already been formalized. Nevertheless, we still provide a proof that WPO subsumes KBO and thereby underline the generality of WPO.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Preliminaries</b>	<b>3</b>
2.1	Status functions . . . . .	3
2.2	Precedence . . . . .	4
2.3	Local versions of relations . . . . .	5
2.4	Interface for extending an order pair on lists . . . . .	10
<b>3</b>	<b>Multiset extension of an order pair</b>	<b>10</b>
3.1	Pointwise multiset order . . . . .	11
3.2	Multiset extension for order pairs via the pointwise order and <i>mult</i> . . . . .	13
3.3	One-step versions of the multiset extensions . . . . .	14
3.4	Cancellation . . . . .	15

---

\*Supported by FWF (Austrian Science Fund) projects P27502 and Y757.

3.5	Implementation friendly versions of <i>mult2-s</i> and <i>mult2-ns</i> . . .	16
3.6	Local well-foundedness: restriction to downward closure of a set . . . . .	17
3.7	Trivial cases . . . . .	18
3.8	Executable version . . . . .	20
<b>4</b>	<b>Multiset extension of order pairs in the other direction</b>	<b>22</b>
4.1	List based characterization of <i>multpw</i> . . . . .	22
4.2	Definition of the multiset extension of $>$ -orders . . . . .	22
4.3	Basic properties . . . . .	23
4.4	Multisets as order on lists . . . . .	29
4.5	Special case: non-strict order is equality . . . . .	30
4.6	Executable version . . . . .	31
<b>5</b>	<b>The Weighted Path Order</b>	<b>34</b>
<b>6</b>	<b>The Recursive Path Order as an instance of WPO</b>	<b>39</b>
<b>7</b>	<b>The Lexicographic Path Order as an instance of WPO</b>	<b>41</b>
<b>8</b>	<b>The Knuth–Bendix Order as an instance of WPO</b>	<b>43</b>
8.1	Aligning least elements . . . . .	43
8.2	A restricted equality between KBO and WPO . . . . .	44
<b>9</b>	<b>Executability of the orders</b>	<b>47</b>

## 1 Introduction

Path orders are well-founded orders on terms that are useful for automated deduction, e.g., for termination proving of term rewrite systems or for completion-based theorem provers. Well-known path orders are the lexicographic path order (LPO) [3], the recursive path order (RPO) [2], and the Knuth–Bendix order (KBO) [4], and all of these orders are presented in a standard textbook on term rewriting [1, Chapter 5].

Whereas the mentioned path orders date back to the last century, the weighted path order (WPO) has only recently been presented [9, 10]. It has two nice properties. First, the search for suitable parameters is feasible and tools like NaTT and TTT2 implement it. Second, WPO is quite powerful and versatile: in fact, KBO and LPO are just instances of WPO. Moreover, with a slight extension of WPO (adding multiset-comparisons) also RPO is covered.

This AFP-entry provides a full formalization of WPO and also the connection to KBO, LPO, and RPO. Here, for the existing formal version of KBO [5, 6] it is just proven that WPO can simulate it by choosing suitable

parameters, whereas LPO and RPO are defined from scratch and many properties of LPO and RPO—such as strong normalization, closure under contexts and substitutions, transitivity, etc.—are derived from the corresponding WPO properties.

Note that most of the WPO formalization is described in [8]. The formal version deviates from the paper version only by the additional possibility to perform multiset-comparisons instead of lexicographic comparisons within WPO. The formal version of LPO and RPO extend their original definitions as well: the RPO definition is taken from [7], and LPO is defined as this extended RPO where always lexicographic comparisons are performed when comparing lists of terms. The formalization of multiset-comparisons (w.r.t. two orders) is described in more detail in [7].

## 2 Preliminaries

### 2.1 Status functions

A status function assigns to each n-ary symbol a list of indices between 0 and n-1. These functions are encapsulated into a separate type, so that recursion on the i-th subterm does not have to perform out-of-bounds checks (e.g., to ensure termination).

**theory** *Status*

**imports**

*First-Order-Terms.Term*

**begin**

**typedef** *'f status* = { ( $\sigma :: 'f \times \text{nat} \Rightarrow \text{nat list}$ ). ( $\forall f k. \text{set } (\sigma (f, k)) \subseteq \{0 ..< k\}$ )}

**morphisms** *status Abs-status*

*<proof>*

**setup-lifting** *type-definition-status*

**lemma** *status*:  $\text{set } (\text{status } \sigma (f, n)) \subseteq \{0 ..< n\}$

*<proof>*

**lemma** *status-aux[termination-simp]*:  $i \in \text{set } (\text{status } \sigma (f, \text{length } ss)) \implies ss ! i \in \text{set } ss$

*<proof>*

**lemma** *status-termination-simps[termination-simp]*:

**assumes** *i1*:  $i < \text{length } (\text{status } \sigma (f, \text{length } xs))$

**shows**  $\text{size } (xs ! (\text{status } \sigma (f, \text{length } xs) ! i)) < \text{Suc } (\text{size-list size } xs)$  (**is**  $?a < ?c$ )

*<proof>*

**lemma** *status-ne*:

$\text{status } \sigma (f, n) \neq [] \implies \exists i < n. i \in \text{set } (\text{status } \sigma (f, n))$

*<proof>*

**lemma** *set-status-nth*:

$length\ xs = n \implies i \in set\ (status\ \sigma\ (f,\ n)) \implies i < length\ xs \wedge xs\ !\ i \in set\ xs$   
*<proof>*

**lift-definition** *full-status* :: '*f* status is  $\lambda\ (f,\ n).$   $[0\ ..<\ n]$  *<proof>*

**lemma** *full-status[simp]*:  $status\ full\ status\ (f,\ n) = [0\ ..<\ n]$

*<proof>*

An argument position *i* is simple wrt. some term relation, if the *i*-th subterm is in relation to the full term.

**definition** *simple-arg-pos* :: ('*f*, '*v*) term rel  $\Rightarrow$  '*f*  $\times$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool **where**

$simple\ arg\ pos\ rel\ f\ i \equiv \forall\ ts.\ i < snd\ f \longrightarrow length\ ts = snd\ f \longrightarrow (Fun\ (fst\ f)\ ts,\ ts\ !\ i) \in rel$

**lemma** *simple-arg-posI*:  $\llbracket \bigwedge\ ts.\ length\ ts = n \implies i < n \implies (Fun\ f\ ts,\ ts\ !\ i) \in rel \rrbracket \implies simple\ arg\ pos\ rel\ (f,\ n)\ i$

*<proof>*

**end**

## 2.2 Precedence

A precedence consists of two compatible relations (strict and non-strict) on symbols such that the strict relation is strongly normalizing. In the formalization we model this via a function "prc" (precedence-compare) which returns two Booleans, indicating whether the one symbol is strictly or weakly bigger than the other symbol. Moreover, there also is a function "prl" (precedence-least) which gives quick access to whether a symbol is least in precedence, i.e., without comparing it to all other symbols explicitly.

**theory** *Precedence*

**imports**

*Abstract-Rewriting.Abstract-Rewriting*

**begin**

**locale** *precedence* =

**fixes** *prc* :: '*f*  $\Rightarrow$  '*f*  $\Rightarrow$  bool  $\times$  bool

**and** *prl* :: '*f*  $\Rightarrow$  bool

**assumes** *prc-refl*:  $prc\ f\ f = (False,\ True)$

**and** *prc-SN*:  $SN\ \{(f,\ g).\ fst\ (prc\ f\ g)\}$

**and** *prl*:  $prl\ g \implies snd\ (prc\ f\ g) = True$

**and** *prl3*:  $prl\ f \implies snd\ (prc\ f\ g) \implies prl\ g$

**and** *prc-compat*:  $prc\ f\ g = (s1,\ ns1) \implies prc\ g\ h = (s2,\ ns2) \implies prc\ f\ h = (s,\ ns) \implies$

$(ns1 \wedge ns2 \longrightarrow ns) \wedge (ns1 \wedge s2 \longrightarrow s) \wedge (s1 \wedge ns2 \longrightarrow s)$

**and** *prc-stri-imp-nstri*:  $fst\ (prc\ f\ g) \implies snd\ (prc\ f\ g)$

**begin**

**lemma** *prl2*:

**assumes** *g*: *prl g* **shows** *fst (prc g f) = False*  
*<proof>*

**abbreviation** *pr*  $\equiv$  (*prc*, *prl*)

**end**

**end**

## 2.3 Local versions of relations

**theory** *Relations*

**imports**

*HOL-Library.Multiset*

*Abstract-Rewriting.Abstract-Rewriting*

**begin**

Common predicates on relations

**definition** *compatible-l* :: '*a* rel  $\Rightarrow$  '*a* rel  $\Rightarrow$  bool **where**

*compatible-l* *R1 R2*  $\equiv$  *R1 O R2*  $\subseteq$  *R2*

**definition** *compatible-r* :: '*a* rel  $\Rightarrow$  '*a* rel  $\Rightarrow$  bool **where**

*compatible-r* *R1 R2*  $\equiv$  *R2 O R1*  $\subseteq$  *R2*

Local reflexivity

**definition** *locally-refl* :: '*a* rel  $\Rightarrow$  '*a* multiset  $\Rightarrow$  bool **where**

*locally-refl* *R A*  $\equiv$  ( $\forall$  *a*. *a*  $\in\#$  *A*  $\longrightarrow$  (*a,a*)  $\in$  *R*)

**definition** *locally-irrefl* :: '*a* rel  $\Rightarrow$  '*a* multiset  $\Rightarrow$  bool **where**

*locally-irrefl* *R A*  $\equiv$  ( $\forall$  *t*. *t*  $\in\#$  *A*  $\longrightarrow$  (*t,t*)  $\notin$  *R*)

Local symmetry

**definition** *locally-sym* :: '*a* rel  $\Rightarrow$  '*a* multiset  $\Rightarrow$  bool **where**

*locally-sym* *R A*  $\equiv$  ( $\forall$  *t u*. *t*  $\in\#$  *A*  $\longrightarrow$  *u*  $\in\#$  *A*  $\longrightarrow$   
(*t,u*)  $\in$  *R*  $\longrightarrow$  (*u,t*)  $\in$  *R*)

**definition** *locally-antisym* :: '*a* rel  $\Rightarrow$  '*a* multiset  $\Rightarrow$  bool **where**

*locally-antisym* *R A*  $\equiv$  ( $\forall$  *t u*. *t*  $\in\#$  *A*  $\longrightarrow$  *u*  $\in\#$  *A*  $\longrightarrow$   
(*t,u*)  $\in$  *R*  $\longrightarrow$  (*u,t*)  $\in$  *R*  $\longrightarrow$  *t = u*)

Local transitivity

**definition** *locally-trans* :: '*a* rel  $\Rightarrow$  '*a* multiset  $\Rightarrow$  '*a* multiset  $\Rightarrow$  '*a* multiset  $\Rightarrow$  bool  
**where**

*locally-trans* *R A B C*  $\equiv$  ( $\forall$  *t u v*.  
*t*  $\in\#$  *A*  $\longrightarrow$  *u*  $\in\#$  *B*  $\longrightarrow$  *v*  $\in\#$  *C*  $\longrightarrow$   
(*t,u*)  $\in$  *R*  $\longrightarrow$  (*u,v*)  $\in$  *R*  $\longrightarrow$  (*t,v*)  $\in$  *R*)

### Local inclusion

**definition** *locally-included* :: 'a rel  $\Rightarrow$  'a rel  $\Rightarrow$  'a multiset  $\Rightarrow$  'a multiset  $\Rightarrow$  bool  
**where**

$$\text{locally-included } R1 \ R2 \ A \ B \equiv (\forall t \ u. t \in\# \ A \longrightarrow u \in\# \ B \longrightarrow \\ (t,u) \in R1 \longrightarrow (t,u) \in R2)$$

### Local transitivity compatibility

**definition** *locally-compatible-l* :: 'a rel  $\Rightarrow$  'a rel  $\Rightarrow$  'a multiset  $\Rightarrow$  'a multiset  $\Rightarrow$  'a multiset  $\Rightarrow$  bool  
**where**

$$\text{locally-compatible-l } R1 \ R2 \ A \ B \ C \equiv (\forall t \ u \ v. t \in\# \ A \longrightarrow u \in\# \ B \longrightarrow v \in\# \ C \\ \longrightarrow \\ (t,u) \in R1 \longrightarrow (u,v) \in R2 \longrightarrow (t,v) \in R2)$$

**definition** *locally-compatible-r* :: 'a rel  $\Rightarrow$  'a rel  $\Rightarrow$  'a multiset  $\Rightarrow$  'a multiset  $\Rightarrow$  'a multiset  $\Rightarrow$  bool  
**where**

$$\text{locally-compatible-r } R1 \ R2 \ A \ B \ C \equiv (\forall t \ u \ v. t \in\# \ A \longrightarrow u \in\# \ B \longrightarrow v \in\# \ C \\ \longrightarrow \\ (t,u) \in R2 \longrightarrow (u,v) \in R1 \longrightarrow (t,v) \in R2)$$

included + compatible  $\longrightarrow$  transitive

**lemma** *in-cl-tr*:

**assumes**  $R1 \subseteq R2$

**and** *compatible-l*  $R2 \ R1$

**shows** *trans*  $R1$

*<proof>*

**lemma** *in-cr-tr*:

**assumes**  $R1 \subseteq R2$

**and** *compatible-r*  $R2 \ R1$

**shows** *trans*  $R1$

*<proof>*

If a property holds globally, it also holds locally. Obviously.

**lemma** *r-lr*:

**assumes** *refl*  $R$

**shows** *locally-refl*  $R \ A$

*<proof>*

**lemma** *tr-ltr*:

**assumes** *trans*  $R$

**shows** *locally-trans*  $R \ A \ B \ C$

*<proof>*

**lemma** *in-lin*:

**assumes**  $R1 \subseteq R2$

**shows** *locally-included*  $R1 \ R2 \ A \ B$

*<proof>*

**lemma** *cl-lcl*:

**assumes** *compatible-l*  $R1\ R2$   
**shows** *locally-compatible-l*  $R1\ R2\ A\ B\ C$   
*<proof>*

**lemma** *cr-lcr*:

**assumes** *compatible-r*  $R1\ R2$   
**shows** *locally-compatible-r*  $R1\ R2\ A\ B\ C$   
*<proof>*

If a predicate holds on a set then it holds on all the subsets:

**lemma** *lr-trans-l*:

**assumes** *locally-refl*  $R\ (A + B)$   
**shows** *locally-refl*  $R\ A$   
*<proof>*

**lemma** *li-trans-l*:

**assumes** *locally-irrefl*  $R\ (A + B)$   
**shows** *locally-irrefl*  $R\ A$   
*<proof>*

**lemma** *ls-trans-l*:

**assumes** *locally-sym*  $R\ (A + B)$   
**shows** *locally-sym*  $R\ A$   
*<proof>*

**lemma** *las-trans-l*:

**assumes** *locally-antisym*  $R\ (A + B)$   
**shows** *locally-antisym*  $R\ A$   
*<proof>*

**lemma** *lt-trans-l*:

**assumes** *locally-trans*  $R\ (A + B)\ (C + D)\ (E + F)$   
**shows** *locally-trans*  $R\ A\ C\ E$   
*<proof>*

**lemma** *lin-trans-l*:

**assumes** *locally-included*  $R1\ R2\ (A + B)\ (C + D)$   
**shows** *locally-included*  $R1\ R2\ A\ C$   
*<proof>*

**lemma** *lcl-trans-l*:

**assumes** *locally-compatible-l*  $R1\ R2\ (A + B)\ (C + D)\ (E + F)$   
**shows** *locally-compatible-l*  $R1\ R2\ A\ C\ E$   
*<proof>*

**lemma** *lcr-trans-l*:

**assumes** *locally-compatible-r*  $R1\ R2\ (A + B)\ (C + D)\ (E + F)$   
**shows** *locally-compatible-r*  $R1\ R2\ A\ C\ E$   
*<proof>*

**lemma** *lr-trans-r*:

**assumes** *locally-refl*  $R (A + B)$

**shows** *locally-refl*  $R B$

*<proof>*

**lemma** *li-trans-r*:

**assumes** *locally-irrefl*  $R (A + B)$

**shows** *locally-irrefl*  $R B$

*<proof>*

**lemma** *ls-trans-r*:

**assumes** *locally-sym*  $R (A + B)$

**shows** *locally-sym*  $R B$

*<proof>*

**lemma** *las-trans-r*:

**assumes** *locally-antisym*  $R (A + B)$

**shows** *locally-antisym*  $R B$

*<proof>*

**lemma** *lt-trans-r*:

**assumes** *locally-trans*  $R (A + B) (C + D) (E + F)$

**shows** *locally-trans*  $R B D F$

*<proof>*

**lemma** *lin-trans-r*:

**assumes** *locally-included*  $R1 R2 (A + B) (C + D)$

**shows** *locally-included*  $R1 R2 B D$

*<proof>*

**lemma** *lcl-trans-r*:

**assumes** *locally-compatible-l*  $R1 R2 (A + B) (C + D) (E + F)$

**shows** *locally-compatible-l*  $R1 R2 B D F$

*<proof>*

**lemma** *lcr-trans-r*:

**assumes** *locally-compatible-r*  $R1 R2 (A + B) (C + D) (E + F)$

**shows** *locally-compatible-r*  $R1 R2 B D F$

*<proof>*

**lemma** *lr-minus*:

**assumes** *locally-refl*  $R A$

**shows** *locally-refl*  $R (A - B)$

*<proof>*

**lemma** *li-minus*:

**assumes** *locally-irrefl*  $R A$

**shows** *locally-irrefl*  $R (A - B)$



*<proof>*

**lemma** *ls-minus*:

**assumes** *locally-sym*  $R$   $A$

**shows** *locally-sym*  $R$   $(A - B)$

*<proof>*

**lemma** *las-minus*:

**assumes** *locally-antisym*  $R$   $A$

**shows** *locally-antisym*  $R$   $(A - B)$

*<proof>*

**lemma** *lt-minus*:

**assumes** *locally-trans*  $R$   $A$   $C$   $E$

**shows** *locally-trans*  $R$   $(A - B)$   $(C - D)$   $(E - F)$

*<proof>*

**lemma** *lin-minus*:

**assumes** *locally-included*  $R1$   $R2$   $A$   $C$

**shows** *locally-included*  $R1$   $R2$   $(A - B)$   $(C - D)$

*<proof>*

**lemma** *lcl-minus*:

**assumes** *locally-compatible-l*  $R1$   $R2$   $A$   $C$   $E$

**shows** *locally-compatible-l*  $R1$   $R2$   $(A - B)$   $(C - D)$   $(E - F)$

*<proof>*

**lemma** *lcr-minus*:

**assumes** *locally-compatible-r*  $R1$   $R2$   $A$   $C$   $E$

**shows** *locally-compatible-r*  $R1$   $R2$   $(A - B)$   $(C - D)$   $(E - F)$

*<proof>*

Notations

**notation** *restrict* (**infixl**  $\uparrow$  80)

**lemma** *mem-restrictI*[*intro!*]: **assumes**  $x \in X$   $y \in X$   $(x,y) \in R$  **shows**  $(x,y) \in R$

$\uparrow$   $X$

*<proof>*

**lemma** *mem-restrictD*[*dest*]: **assumes**  $(x,y) \in R$   $\uparrow$   $X$  **shows**  $x \in X$   $y \in X$   $(x,y) \in R$

*<proof>*

**end**

## 2.4 Interface for extending an order pair on lists

**theory** *List-Order*

**imports**

*Knuth-Bendix-Order.Order-Pair*

**begin**

**type-synonym** *'a list-ext* = *'a rel*  $\Rightarrow$  *'a rel*  $\Rightarrow$  *'a list rel*

**locale** *list-order-extension* =

**fixes** *s-list* :: *'a list-ext*

**and** *ns-list* :: *'a list-ext*

**assumes** *extension*: *SN-order-pair* *S NS*  $\Longrightarrow$  *SN-order-pair* (*s-list S NS*) (*ns-list S NS*)

**and** *s-map*:  $\llbracket \bigwedge a b. (a,b) \in S \Longrightarrow (f a, f b) \in S; \bigwedge a b. (a,b) \in NS \Longrightarrow (f a, f b) \in NS \rrbracket \Longrightarrow (as, bs) \in s\text{-list } S NS \Longrightarrow (map f as, map f bs) \in s\text{-list } S NS$

**and** *ns-map*:  $\llbracket \bigwedge a b. (a,b) \in S \Longrightarrow (f a, f b) \in S; \bigwedge a b. (a,b) \in NS \Longrightarrow (f a, f b) \in NS \rrbracket \Longrightarrow (as, bs) \in ns\text{-list } S NS \Longrightarrow (map f as, map f bs) \in ns\text{-list } S NS$

**and** *all-ns-imp-ns*: *length as* = *length bs*  $\Longrightarrow \llbracket \bigwedge i. i < \text{length } bs \Longrightarrow (as ! i, bs ! i) \in NS \rrbracket \Longrightarrow (as, bs) \in ns\text{-list } S NS$

**type-synonym** *'a list-ext-impl* = (*'a*  $\Rightarrow$  *'a*  $\Rightarrow$  *bool*  $\times$  *bool*)  $\Rightarrow$  *'a list*  $\Rightarrow$  *'a list*  $\Rightarrow$  *bool*  $\times$  *bool*

**locale** *list-order-extension-impl* = *list-order-extension s-list ns-list for*

*s-list ns-list* :: *'a list-ext* +

**fixes** *list-ext-impl*

**assumes** *list-ext-s*:  $\bigwedge s ns. s\text{-list } \{(a,b). s a b\} \{(a,b). ns a b\} = \{(as,bs). fst (list\text{-ext } (\lambda a b. (s a b, ns a b)) as bs)\}$

**and** *list-ext-ns*:  $\bigwedge s ns. ns\text{-list } \{(a,b). s a b\} \{(a,b). ns a b\} = \{(as,bs). snd (list\text{-ext } (\lambda a b. (s a b, ns a b)) as bs)\}$

**and** *s-ext-local-mono*:  $\bigwedge s ns s' ns' as bs. (set as \times set bs) \cap ns \subseteq ns' \Longrightarrow (set as \times set bs) \cap s \subseteq s' \Longrightarrow (as,bs) \in s\text{-list } ns s \Longrightarrow (as,bs) \in s\text{-list } ns' s'$

**and** *ns-ext-local-mono*:  $\bigwedge s ns s' ns' as bs. (set as \times set bs) \cap ns \subseteq ns' \Longrightarrow (set as \times set bs) \cap s \subseteq s' \Longrightarrow (as,bs) \in ns\text{-list } ns s \Longrightarrow (as,bs) \in ns\text{-list } ns' s'$

**end**

## 3 Multiset extension of an order pair

Given a well-founded order  $\prec$  and a compatible non-strict order  $\succsim$ , we define the corresponding multiset-extension of these orders.

**theory** *Multiset-Extension-Pair*

**imports**

*HOL-Library.Multiset*

*Regular-Sets.Regexp-Method*

*Abstract-Rewriting.Abstract-Rewriting*

*Relations*

**begin**

**lemma** *mult-locally-cancel*:

**assumes** *trans s* **and** *locally-irrefl s (X + Z)* **and** *locally-irrefl s (Y + Z)*  
**shows**  $(X + Z, Y + Z) \in \text{mult } s \longleftrightarrow (X, Y) \in \text{mult } s$  (**is**  $?L \longleftrightarrow ?R$ )  
*<proof>*

**lemma** *mult-locally-cancelL*:

**assumes** *trans s* *locally-irrefl s (X + Z)* *locally-irrefl s (Y + Z)*  
**shows**  $(Z + X, Z + Y) \in \text{mult } s \longleftrightarrow (X, Y) \in \text{mult } s$   
*<proof>*

**lemma** *mult-cancelL*:

**assumes** *trans s* *irrefl s* **shows**  $(Z + X, Z + Y) \in \text{mult } s \longleftrightarrow (X, Y) \in \text{mult } s$   
*<proof>*

**lemma** *wf-trancl-conv*:

**shows**  $\text{wf } (r^+) \longleftrightarrow \text{wf } r$   
*<proof>*

### 3.1 Pointwise multiset order

**inductive-set** *multpw* :: 'a rel  $\Rightarrow$  'a multiset rel **for** *ns* :: 'a rel **where**

*empty*:  $(\{\#\}, \{\#\}) \in \text{multpw } ns$   
*| add*:  $(x, y) \in ns \Longrightarrow (X, Y) \in \text{multpw } ns \Longrightarrow (\text{add-mset } x \ X, \text{add-mset } y \ Y) \in \text{multpw } ns$

**lemma** *multpw-emptyL* [*simp*]:

$(\{\#\}, X) \in \text{multpw } ns \longleftrightarrow X = \{\#\}$   
*<proof>*

**lemma** *multpw-emptyR* [*simp*]:

$(X, \{\#\}) \in \text{multpw } ns \longleftrightarrow X = \{\#\}$   
*<proof>*

**lemma** *refl-multpw*:

**assumes** *refl ns* **shows** *refl (multpw ns)*  
*<proof>*

**lemma** *multpw-Id-Id* [*simp*]:

*multpw Id = Id*  
*<proof>*

**lemma** *mono-multpw*:

**assumes**  $ns \subseteq ns'$  **shows**  $\text{multpw } ns \subseteq \text{multpw } ns'$   
*<proof>*

**lemma** *multpw-converse*:

$multpw (ns^{-1}) = (multpw ns)^{-1}$   
*<proof>*

**lemma** *multpw-local*:

$(X, Y) \in multpw ns \implies (X, Y) \in multpw (ns \cap set-mset X \times set-mset Y)$   
*<proof>*

**lemma** *multpw-split1R*:

**assumes**  $(add-mset x X, Y) \in multpw ns$   
**obtains**  $z Z$  **where**  $Y = add-mset z Z$  **and**  $(x, z) \in ns$  **and**  $(X, Z) \in multpw ns$   
*<proof>*

**lemma** *multpw-splitR*:

**assumes**  $(X1 + X2, Y) \in multpw ns$   
**obtains**  $Y1 Y2$  **where**  $Y = Y1 + Y2$  **and**  $(X1, Y1) \in multpw ns$  **and**  $(X2, Y2) \in multpw ns$   
*<proof>*

**lemma** *multpw-split1L*:

**assumes**  $(X, add-mset y Y) \in multpw ns$   
**obtains**  $z Z$  **where**  $X = add-mset z Z$  **and**  $(z, y) \in ns$  **and**  $(Z, Y) \in multpw ns$   
*<proof>*

**lemma** *multpw-splitL*:

**assumes**  $(X, Y1 + Y2) \in multpw ns$   
**obtains**  $X1 X2$  **where**  $X = X1 + X2$  **and**  $(X1, Y1) \in multpw ns$  **and**  $(X2, Y2) \in multpw ns$   
*<proof>*

**lemma** *locally-trans-multpw*:

**assumes** *locally-trans ns S T U*  
**and**  $(S, T) \in multpw ns$   
**and**  $(T, U) \in multpw ns$   
**shows**  $(S, U) \in multpw ns$   
*<proof>*

**lemma** *trans-multpw*:

**assumes** *trans ns* **shows** *trans (multpw ns)*  
*<proof>*

**lemma** *multpw-add*:

**assumes**  $(X1, Y1) \in multpw ns$   $(X2, Y2) \in multpw ns$  **shows**  $(X1 + X2, Y1 + Y2) \in multpw ns$   
*<proof>*

**lemma** *multpw-single*:

$(x, y) \in ns \implies (\{x\}, \{y\}) \in multpw ns$   
*<proof>*

**lemma** *multpw-mult1-commute*:

**assumes** *compat*:  $s \ O \ ns \subseteq s$  **and** *refl ns*:  $refl \ ns$

**shows**  $mult1 \ s \ O \ multpw \ ns \subseteq multpw \ ns \ O \ mult1 \ s$

*<proof>*

**lemma** *multpw-mult-commute*:

**assumes**  $s \ O \ ns \subseteq s$  **and** *refl ns*:  $refl \ ns$  **shows**  $mult \ s \ O \ multpw \ ns \subseteq multpw \ ns \ O \ mult \ s$

*<proof>*

**lemma** *wf-mult-rel-multpw*:

**assumes** *wf s s*:  $wf \ s \ s$  **and** *refl ns*:  $refl \ ns$  **shows**  $wf \ ((multpw \ ns)^* \ O \ mult \ s \ O \ (multpw \ ns)^*)$

*<proof>*

**lemma** *multpw-cancel1*:

**assumes** *trans ns*:  $(y, x) \in ns$

**shows**  $(add-mset \ x \ X, add-mset \ y \ Y) \in multpw \ ns \implies (X, Y) \in multpw \ ns$  (**is**  $?L \implies ?R$ )

*<proof>*

**lemma** *multpw-cancel*:

**assumes** *refl ns*:  $refl \ ns$  **and** *trans ns*:  $trans \ ns$

**shows**  $(X + Z, Y + Z) \in multpw \ ns \longleftrightarrow (X, Y) \in multpw \ ns$  (**is**  $?L \longleftrightarrow ?R$ )

*<proof>*

**lemma** *multpw-cancelL*:

**assumes** *refl ns*:  $refl \ ns$  **and** *trans ns*:  $trans \ ns$  **shows**  $(Z + X, Z + Y) \in multpw \ ns \longleftrightarrow (X, Y) \in multpw \ ns$

*<proof>*

### 3.2 Multiset extension for order pairs via the pointwise order and *mult*

**definition** *mult2-s ns s*:  $mult2-s \ ns \ s \equiv multpw \ ns \ O \ mult \ s$

**definition** *mult2-ns ns s*:  $mult2-ns \ ns \ s \equiv multpw \ ns \ O \ (mult \ s)^=$

**lemma** *mult2-ns-conv*:

**shows**  $mult2-ns \ ns \ s = mult2-s \ ns \ s \cup multpw \ ns$

*<proof>*

**lemma** *mono-mult2-s*:

**assumes**  $ns \subseteq ns'$  **and**  $s \subseteq s'$  **shows**  $mult2-s \ ns \ s \subseteq mult2-s \ ns' \ s'$

*<proof>*

**lemma** *mono-mult2-ns*:

**assumes**  $ns \subseteq ns'$  **and**  $s \subseteq s'$  **shows**  $mult2-ns \ ns \ s \subseteq mult2-ns \ ns' \ s'$

*<proof>*

**lemma** *wf-mult2-s*:

**assumes**  $wf\ s\ s\ O\ ns \subseteq s\ refl\ ns$   
**shows**  $wf\ (mult2-s\ ns\ s)$   
 $\langle proof \rangle$

**lemma** *refl-mult2-ns*:  
**assumes**  $refl\ ns$  **shows**  $refl\ (mult2-ns\ ns\ s)$   
 $\langle proof \rangle$

**lemma** *trans-mult2-s*:  
**assumes**  $s\ O\ ns \subseteq s\ refl\ ns\ trans\ ns$   
**shows**  $trans\ (mult2-s\ ns\ s)$   
 $\langle proof \rangle$

**lemma** *trans-mult2-ns*:  
**assumes**  $s\ O\ ns \subseteq s\ refl\ ns\ trans\ ns$   
**shows**  $trans\ (mult2-ns\ ns\ s)$   
 $\langle proof \rangle$

**lemma** *compat-mult2*:  
**assumes**  $s\ O\ ns \subseteq s\ refl\ ns\ trans\ ns$   
**shows**  $mult2-ns\ ns\ s\ O\ mult2-s\ ns\ s \subseteq mult2-s\ ns\ s\ mult2-s\ ns\ s\ O\ mult2-ns\ ns$   
 $s \subseteq mult2-s\ ns\ s$   
 $\langle proof \rangle$

Trivial inclusions

**lemma** *mult-implies-mult2-s*:  
**assumes**  $refl\ ns\ (X, Y) \in mult\ s$   
**shows**  $(X, Y) \in mult2-s\ ns\ s$   
 $\langle proof \rangle$

**lemma** *mult-implies-mult2-ns*:  
**assumes**  $refl\ ns\ (X, Y) \in (mult\ s)^=$   
**shows**  $(X, Y) \in mult2-ns\ ns\ s$   
 $\langle proof \rangle$

**lemma** *multpw-implies-mult2-ns*:  
**assumes**  $(X, Y) \in multpw\ ns$   
**shows**  $(X, Y) \in mult2-ns\ ns\ s$   
 $\langle proof \rangle$

### 3.3 One-step versions of the multiset extensions

**lemma** *mult2-s-one-step*:  
**assumes**  $ns\ O\ s \subseteq s\ refl\ ns\ trans\ s$   
**shows**  $(X, Y) \in mult2-s\ ns\ s \longleftrightarrow (\exists X1\ X2\ Y1\ Y2. X = X1 + X2 \wedge Y = Y1 + Y2 \wedge (X1, Y1) \in multpw\ ns \wedge Y2 \neq \{\#\} \wedge (\forall x. x \in\# X2 \longrightarrow (\exists y. y \in\# Y2 \wedge (x, y) \in s)))$  (is ?L  $\longleftrightarrow$  ?R)  
 $\langle proof \rangle$

**lemma** *mult2-ns-one-step*:

**assumes**  $ns \ O \ s \subseteq s \ refl \ ns \ trans \ s$   
**shows**  $(X, Y) \in mult2-ns \ ns \ s \longleftrightarrow (\exists X1 \ X2 \ Y1 \ Y2. X = X1 + X2 \wedge Y = Y1 + Y2 \wedge (X1, Y1) \in multpw \ ns \wedge (\forall x. x \in\# \ X2 \longrightarrow (\exists y. y \in\# \ Y2 \wedge (x, y) \in s)))$  (**is**  $?L \longleftrightarrow ?R$ )  
*<proof>*

**lemma** *mult2-s-locally-one-step'*:

**assumes**  $ns \ O \ s \subseteq s \ refl \ ns \ locally-irrefl \ s \ X \ locally-irrefl \ s \ Y \ trans \ s$   
**shows**  $(X, Y) \in mult2-s \ ns \ s \longleftrightarrow (\exists X1 \ X2 \ Y1 \ Y2. X = X1 + X2 \wedge Y = Y1 + Y2 \wedge (X1, Y1) \in multpw \ ns \wedge (X2, Y2) \in mult \ s)$  (**is**  $?L \longleftrightarrow ?R$ )  
*<proof>*

**lemma** *mult2-s-one-step'*:

**assumes**  $ns \ O \ s \subseteq s \ refl \ ns \ irrefl \ s \ trans \ s$   
**shows**  $(X, Y) \in mult2-s \ ns \ s \longleftrightarrow (\exists X1 \ X2 \ Y1 \ Y2. X = X1 + X2 \wedge Y = Y1 + Y2 \wedge (X1, Y1) \in multpw \ ns \wedge (X2, Y2) \in mult \ s)$  (**is**  $?L \longleftrightarrow ?R$ )  
*<proof>*

**lemma** *mult2-ns-one-step'*:

**assumes**  $ns \ O \ s \subseteq s \ refl \ ns \ irrefl \ s \ trans \ s$   
**shows**  $(X, Y) \in mult2-ns \ ns \ s \longleftrightarrow (\exists X1 \ X2 \ Y1 \ Y2. X = X1 + X2 \wedge Y = Y1 + Y2 \wedge (X1, Y1) \in multpw \ ns \wedge (X2, Y2) \in (mult \ s)^=)$  (**is**  $?L \longleftrightarrow ?R$ )  
*<proof>*

### 3.4 Cancellation

**lemma** *mult2-s-locally-cancel1*:

**assumes**  $s \ O \ ns \subseteq s \ ns \ O \ s \subseteq s \ refl \ ns \ trans \ ns \ locally-irrefl \ s \ (add-mset \ z \ X) \ locally-irrefl \ s \ (add-mset \ z \ Y) \ trans \ s$   
 $(add-mset \ z \ X, add-mset \ z \ Y) \in mult2-s \ ns \ s$   
**shows**  $(X, Y) \in mult2-s \ ns \ s$   
*<proof>*

**lemma** *mult2-s-cancel1*:

**assumes**  $s \ O \ ns \subseteq s \ ns \ O \ s \subseteq s \ refl \ ns \ trans \ ns \ irrefl \ s \ trans \ s \ (add-mset \ z \ X, add-mset \ z \ Y) \in mult2-s \ ns \ s$   
**shows**  $(X, Y) \in mult2-s \ ns \ s$   
*<proof>*

**lemma** *mult2-s-locally-cancel*:

**assumes**  $s \ O \ ns \subseteq s \ ns \ O \ s \subseteq s \ refl \ ns \ trans \ ns \ locally-irrefl \ s \ (X + Z) \ locally-irrefl \ s \ (Y + Z) \ trans \ s$   
**shows**  $(X + Z, Y + Z) \in mult2-s \ ns \ s \implies (X, Y) \in mult2-s \ ns \ s$   
*<proof>*

**lemma** *mult2-s-cancel*:

**assumes**  $s \ O \ ns \subseteq s \ ns \ O \ s \subseteq s \ refl \ ns \ trans \ ns \ irrefl \ s \ trans \ s$   
**shows**  $(X + Z, Y + Z) \in mult2-s \ ns \ s \implies (X, Y) \in mult2-s \ ns \ s$   
*<proof>*

**lemma** *mult2-ns-cancel*:

**assumes**  $s \ O \ ns \subseteq s \ ns \ O \ s \subseteq s \ refl \ ns \ trans \ s \ irrefl \ s \ trans \ ns$   
**shows**  $(X + Z, Y + Z) \in mult2-s \ ns \ s \implies (X, Y) \in mult2-ns \ ns \ s$   
*<proof>*

### 3.5 Implementation friendly versions of *mult2-s* and *mult2-ns*

**definition** *mult2-alt* :: *bool*  $\Rightarrow$  '*a rel*  $\Rightarrow$  '*a multiset rel* **where**

*mult2-alt*  $b \ ns \ s = \{(X, Y). (\exists X1 \ X2 \ Y1 \ Y2. X = X1 + X2 \wedge Y = Y1 + Y2 \wedge$   
 $(X1, Y1) \in multpw \ ns \wedge (b \vee Y2 \neq \{\#\}) \wedge (\forall x. x \in\# \ X2 \longrightarrow (\exists y. y \in\# \ Y2$   
 $\wedge (x, y) \in s))\}$

**lemma** *mult2-altI*:

**assumes**  $X = X1 + X2 \ Y = Y1 + Y2 \ (X1, Y1) \in multpw \ ns$   
 $b \vee Y2 \neq \{\#\} \ \forall x. x \in\# \ X2 \longrightarrow (\exists y. y \in\# \ Y2 \wedge (x, y) \in s)$   
**shows**  $(X, Y) \in mult2-alt \ b \ ns \ s$   
*<proof>*

**lemma** *mult2-altE*:

**assumes**  $(X, Y) \in mult2-alt \ b \ ns \ s$   
**obtains**  $X1 \ X2 \ Y1 \ Y2$  **where**  $X = X1 + X2 \ Y = Y1 + Y2 \ (X1, Y1) \in multpw$   
 $ns$   
 $b \vee Y2 \neq \{\#\} \ \forall x. x \in\# \ X2 \longrightarrow (\exists y. y \in\# \ Y2 \wedge (x, y) \in s)$   
*<proof>*

**lemma** *mono-mult2-alt*:

**assumes**  $ns \subseteq ns' \ s \subseteq s'$  **shows**  $mult2-alt \ b \ ns \ s \subseteq mult2-alt \ b \ ns' \ s'$   
*<proof>*

**abbreviation** *mult2-alt-s*  $\equiv mult2-alt \ False$

**abbreviation** *mult2-alt-ns*  $\equiv mult2-alt \ True$

**lemmas** *mult2-alt-s-def* = *mult2-alt-def*[**where**  $b = False$ , *unfolded simp-thms*]

**lemmas** *mult2-alt-ns-def* = *mult2-alt-def*[**where**  $b = True$ , *unfolded simp-thms*]

**lemmas** *mult2-alt-sI* = *mult2-altI*[**where**  $b = False$ , *unfolded simp-thms*]

**lemmas** *mult2-alt-nsI* = *mult2-altI*[**where**  $b = True$ , *unfolded simp-thms True-implies-equals*]

**lemmas** *mult2-alt-sE* = *mult2-altE*[**where**  $b = False$ , *unfolded simp-thms*]

**lemmas** *mult2-alt-nsE* = *mult2-altE*[**where**  $b = True$ , *unfolded simp-thms True-implies-equals*]

**Equivalence to *mult2-s* and *mult2-ns*** **lemma** *mult2-s-eq-mult2-s-alt*:

**assumes**  $ns \ O \ s \subseteq s \ refl \ ns \ trans \ s$   
**shows**  $mult2-alt-s \ ns \ s = mult2-s \ ns \ s$   
*<proof>*



**lemma** *mult2-ns-eq-mult2-ns-alt*:  
**assumes**  $ns \ O \ s \subseteq \ s \ refl \ ns \ trans \ s$   
**shows**  $mult2-alt-ns \ ns \ s = mult2-ns \ ns \ s$   
 $\langle proof \rangle$

**lemma** *mult2-alt-local*:  
**assumes**  $(X, Y) \in mult2-alt \ b \ ns \ s$   
**shows**  $(X, Y) \in mult2-alt \ b \ (ns \cap \ set-mset \ X \times \ set-mset \ Y) \ (s \cap \ set-mset \ X \times \ set-mset \ Y)$   
 $\langle proof \rangle$

### 3.6 Local well-foundedness: restriction to downward closure of a set

**definition** *wf-below* ::  $'a \ rel \Rightarrow 'a \ set \Rightarrow bool$  **where**  
 $wf-below \ r \ A = wf \ (Restr \ r \ ((r^*)^{-1} \ \ \ A))$

**lemma** *wf-below-UNIV[simp]*:  
**shows**  $wf-below \ r \ UNIV \longleftrightarrow wf \ r$   
 $\langle proof \rangle$

**lemma** *wf-below-mono1*:  
**assumes**  $r \subseteq r'$  **shows**  $wf-below \ r' \ A$   
 $\langle proof \rangle$

**lemma** *wf-below-mono2*:  
**assumes**  $A \subseteq A'$  **shows**  $wf-below \ r \ A'$   
 $\langle proof \rangle$

**lemma** *wf-below-pointwise*:  
 $wf-below \ r \ A \longleftrightarrow (\forall a. a \in A \longrightarrow wf-below \ r \ \{a\})$  **(is ?L  $\longleftrightarrow$  ?R)**  
 $\langle proof \rangle$

**lemma** *SN-on-Image-rtrancl-conv*:  
 $SN-on \ r \ A \longleftrightarrow SN-on \ r \ (r^* \ \ \ A)$  **(is ?L  $\longleftrightarrow$  ?R)**  
 $\langle proof \rangle$

**lemma** *SN-on-iff-wf-below*:  
 $SN-on \ r \ A \longleftrightarrow wf-below \ (r^{-1}) \ A$   
 $\langle proof \rangle$

**lemma** *restr-trancl-under*:  
**shows**  $Restr \ (r^+) \ ((r^*)^{-1} \ \ \ A) = (Restr \ r \ ((r^*)^{-1} \ \ \ A))^+$   
 $\langle proof \rangle$

**lemma** *wf-below-trancl*:  
**shows**  $wf-below \ (r^+) \ A \longleftrightarrow wf-below \ r \ A$

*<proof>*

**lemma** *wf-below-mult-local*:

**assumes** *wf-below r (set-mset X)* **shows** *wf-below (mult r) {X}*

*<proof>*

**lemma** *qc-wf-below*:

**assumes**  $s \ O \ ns \subseteq (s \cup ns)^* \ O \ s$  *wf-below s A*

**shows** *wf-below (ns\* O s O ns\*) A*

*<proof>*

**lemma** *wf-below-mult2-s-local*:

**assumes** *wf-below s (set-mset X)*  $s \ O \ ns \subseteq s \ refl \ ns \ trans \ ns$

**shows** *wf-below (mult2-s ns s) {X}*

*<proof>*

### 3.7 Trivial cases

**lemma** *mult2-alt-emptyL*:

$(\{\#\}, Y) \in \text{mult2-alt } b \ ns \ s \iff b \vee Y \neq \{\#\}$

*<proof>*

**lemma** *mult2-alt-emptyR*:

$(X, \{\#\}) \in \text{mult2-alt } b \ ns \ s \iff b \wedge X = \{\#\}$

*<proof>*

**lemma** *mult2-alt-s-single*:

$(a, b) \in s \implies (\{\#a\#\}, \{\#b\#\}) \in \text{mult2-alt-s } ns \ s$

*<proof>*

**lemma** *multpw-implies-mult2-alt-ns*:

**assumes**  $(X, Y) \in \text{multpw } ns$

**shows**  $(X, Y) \in \text{mult2-alt-ns } ns \ s$

*<proof>*

**lemma** *mult2-alt-ns-conv*:

$\text{mult2-alt-ns } ns \ s = \text{mult2-alt-s } ns \ s \cup \text{multpw } ns \ (\text{is } ?l = ?r)$

*<proof>*

**lemma** *mult2-alt-s-implies-mult2-alt-ns*:

**assumes**  $(X, Y) \in \text{mult2-alt-s } ns \ s$

**shows**  $(X, Y) \in \text{mult2-alt-ns } ns \ s$

*<proof>*

**lemma** *mult2-alt-add*:

**assumes**  $(X1, Y1) \in \text{mult2-alt } b1 \ ns \ s$  **and**  $(X2, Y2) \in \text{mult2-alt } b2 \ ns \ s$

**shows**  $(X1 + X2, Y1 + Y2) \in \text{mult2-alt } (b1 \wedge b2) \ ns \ s$

*<proof>*

**lemmas** *mult2-alt-s-s-add* = *mult2-alt-add*[of - - *False* - - - *False*, *unfolded simp-thms*]  
**lemmas** *mult2-alt-ns-s-add* = *mult2-alt-add*[of - - *True* - - - *False*, *unfolded simp-thms*]  
**lemmas** *mult2-alt-s-ns-add* = *mult2-alt-add*[of - - *False* - - - *True*, *unfolded simp-thms*]  
**lemmas** *mult2-alt-ns-ns-add* = *mult2-alt-add*[of - - *True* - - - *True*, *unfolded simp-thms*]

**lemma** *multpw-map*:

**assumes**  $\bigwedge x y. x \in\# X \implies y \in\# Y \implies (x, y) \in ns \implies (f x, g y) \in ns'$   
**and**  $(X, Y) \in \text{multpw } ns$   
**shows**  $(\text{image-mset } f X, \text{image-mset } g Y) \in \text{multpw } ns'$   
*<proof>*

**lemma** *mult2-alt-map*:

**assumes**  $\bigwedge x y. x \in\# X \implies y \in\# Y \implies (x, y) \in ns \implies (f x, g y) \in ns'$   
**and**  $\bigwedge x y. x \in\# X \implies y \in\# Y \implies (x, y) \in s \implies (f x, g y) \in s'$   
**and**  $(X, Y) \in \text{mult2-alt } b \ ns \ s$   
**shows**  $(\text{image-mset } f X, \text{image-mset } g Y) \in \text{mult2-alt } b \ ns' \ s'$   
*<proof>*

Local transitivity of *mult2-alt*

**lemma** *trans-mult2-alt-local*:

**assumes** *ss*:  $\bigwedge x y z. x \in\# X \implies y \in\# Y \implies z \in\# Z \implies (x, y) \in s \implies (y, z) \in s \implies (x, z) \in s$   
**and** *ns*:  $\bigwedge x y z. x \in\# X \implies y \in\# Y \implies z \in\# Z \implies (x, y) \in ns \implies (y, z) \in ns \implies (x, z) \in ns$   
**and** *sn*:  $\bigwedge x y z. x \in\# X \implies y \in\# Y \implies z \in\# Z \implies (x, y) \in s \implies (y, z) \in ns \implies (x, z) \in s$   
**and** *nn*:  $\bigwedge x y z. x \in\# X \implies y \in\# Y \implies z \in\# Z \implies (x, y) \in ns \implies (y, z) \in ns \implies (x, z) \in ns$   
**and** *xyz*:  $(X, Y) \in \text{mult2-alt } b1 \ ns \ s \ (Y, Z) \in \text{mult2-alt } b2 \ ns \ s$   
**shows**  $(X, Z) \in \text{mult2-alt } (b1 \wedge b2) \ ns \ s$   
*<proof>*

**lemmas** *trans-mult2-alt-s-s-local* = *trans-mult2-alt-local*[of - - - - *False False*, *unfolded simp-thms*]

**lemmas** *trans-mult2-alt-ns-s-local* = *trans-mult2-alt-local*[of - - - - *True False*, *unfolded simp-thms*]

**lemmas** *trans-mult2-alt-s-ns-local* = *trans-mult2-alt-local*[of - - - - *False True*, *unfolded simp-thms*]

**lemmas** *trans-mult2-alt-ns-ns-local* = *trans-mult2-alt-local*[of - - - - *True True*, *unfolded simp-thms*]

**end**

### 3.8 Executable version

**theory** *Multiset-Extension-Pair-Impl*

**imports**

*Multiset-Extension-Pair*

**begin**

**lemma** *subset-mult2-alt*:

**assumes**  $X \subseteq\# Y$   $(Y, Z) \in \text{mult2-alt } b \text{ ns } s \text{ b} \implies b'$

**shows**  $(X, Z) \in \text{mult2-alt } b' \text{ ns } s$

*<proof>*

Case distinction for recursion on left argument

**lemma** *mem-multiset-diff*:  $x \in\# A \implies x \neq y \implies x \in\# (A - \{\#y\#})$

*<proof>*

**lemma** *mult2-alt-addL*:  $(\text{add-mset } x \ X, Y) \in \text{mult2-alt } b \text{ ns } s \longleftrightarrow$

$(\exists y. y \in\# Y \wedge (x, y) \in s \wedge (\{\#x \in\# X. (x, y) \notin s \# \}, Y - \{\#y\#}) \in \text{mult2-alt-ns } ns \ s) \vee$

$(\exists y. y \in\# Y \wedge (x, y) \in ns \wedge (x, y) \notin s \wedge (X, Y - \{\#y\#}) \in \text{mult2-alt } b \text{ ns } s)$

**(is ?L  $\longleftrightarrow$  ?R1  $\vee$  ?R2)**

*<proof>*

Auxiliary version with an extra *bool* argument for distinguishing between the non-strict and the strict orders

**context fixes**  $nss :: 'a \Rightarrow 'a \Rightarrow \text{bool} \Rightarrow \text{bool}$

**begin**

**fun** *mult2-impl0* ::  $'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow \text{bool} \Rightarrow \text{bool}$

**and** *mult2-ex-dom0* ::  $'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow \text{bool} \Rightarrow \text{bool}$

**where**

$\text{mult2-impl0 } [] \quad [] \text{ b} \longleftrightarrow \text{b}$   
 $| \text{mult2-impl0 } xs \quad [] \text{ b} \longleftrightarrow \text{False}$   
 $| \text{mult2-impl0 } [] \quad ys \text{ b} \longleftrightarrow \text{True}$   
 $| \text{mult2-impl0 } (x \# xs) \text{ ys } b \longleftrightarrow \text{mult2-ex-dom0 } x \ xs \ ys \ [] \ b$

$| \text{mult2-ex-dom0 } x \ xs \ [] \quad ys' \text{ b} \longleftrightarrow \text{False}$

$| \text{mult2-ex-dom0 } x \ xs \ (y \# ys) \text{ ys}' \text{ b} \longleftrightarrow$

$nss \ x \ y \ \text{False} \wedge \text{mult2-impl0 } (\text{filter } (\lambda x. \neg nss \ x \ y \ \text{False}) \ xs) \ (ys \ @ \ ys') \ \text{True} \vee$

$nss \ x \ y \ \text{True} \wedge \neg nss \ x \ y \ \text{False} \wedge \text{mult2-impl0 } xs \ (ys \ @ \ ys') \ b \vee$

$\text{mult2-ex-dom0 } x \ xs \ ys \ (y \# ys') \ b$

**end**

**lemma** *mult2-impl0-sound*:

**fixes**  $nss$

**defines**  $ns \equiv \{(x, y). nss \ x \ y \ \text{True}\}$  **and**  $s \equiv \{(x, y). nss \ x \ y \ \text{False}\}$

**shows**  $\text{mult2-impl0 } nss \ xs \ ys \ b \longleftrightarrow (\text{mset } xs, \text{mset } ys) \in \text{mult2-alt } b \text{ ns } s$

$\text{mult2-ex-dom0 } nss \ x \ xs \ ys \ ys' \ b \longleftrightarrow$

$(\exists y. y \in \# \text{ mset } ys \wedge (x, y) \in s \wedge (\text{mset } (\text{filter } (\lambda x. (x, y) \notin s) xs), \text{mset } (ys @ ys') - \{\#y\#}) \in \text{mult2-alt True ns } s) \vee$   
 $(\exists y. y \in \# \text{ mset } ys \wedge (x, y) \in ns \wedge (x, y) \notin s \wedge (\text{mset } xs, \text{mset } (ys @ ys') - \{\#y\#}) \in \text{mult2-alt b ns } s)$   
 <proof>

Now, instead of functions of type  $bool \Rightarrow bool$ , use pairs of type  $bool \times bool$

**definition** [*simp*]:  $or2 a b = (fst a \vee fst b, snd a \vee snd b)$

**context fixes**  $sns :: 'a \Rightarrow 'a \Rightarrow bool \times bool$   
**begin**

**fun**  $mult2-impl :: 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool \times bool$

**and**  $mult2-ex-dom :: 'a \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool \times bool$

**where**

$mult2-impl [] \quad [] = (False, True)$

|  $mult2-impl xs \quad [] = (False, False)$

|  $mult2-impl [] \quad ys = (True, True)$

|  $mult2-impl (x \# xs) ys = mult2-ex-dom x xs ys []$

|  $mult2-ex-dom x xs [] \quad ys' = (False, False)$

|  $mult2-ex-dom x xs (y \# ys) ys' =$

( $case \text{ sns } x \text{ y of}$

( $True, -$ )  $\Rightarrow$  if  $snd (mult2-impl (\text{filter } (\lambda x. \neg fst (sns x y)) xs) (ys @ ys'))$  then  
 $(True, True)$

else  $mult2-ex-dom x xs ys (y \# ys')$

| ( $False, True$ )  $\Rightarrow or2 (mult2-impl xs (ys @ ys')) (mult2-ex-dom x xs ys (y \# ys'))$

|  $- \Rightarrow mult2-ex-dom x xs ys (y \# ys')$

**end**

**lemma**  $mult2-impl-sound0$ :

**defines**  $pair \equiv \lambda f. (f False, f True)$  **and**  $fun \equiv \lambda p b. \text{if } b \text{ then } snd p \text{ else } fst p$

**shows**  $mult2-impl sns xs ys = pair (mult2-impl0 (\lambda x y. fun (sns x y)) xs ys)$  (**is**  $?P$ )

$mult2-ex-dom sns x xs ys ys' = pair (mult2-ex-dom0 (\lambda x y. fun (sns x y)) x xs ys ys')$  (**is**  $?Q$ )

<proof>

**lemmas**  $mult2-impl-sound = mult2-impl-sound0(1)[unfolded mult2-impl0-sound \text{ if-True if-False}]$

**end**

## 4 Multiset extension of order pairs in the other direction

Many term orders are formulated in the other direction, i.e., they use strong normalization of  $>$  instead of well-foundedness of  $<$ . Here, we flip the direction of the multiset extension of two orders, connect it to existing interfaces, and prove some further properties of the multiset extension.

```

theory Multiset-Extension2
  imports
    List-Order
    Multiset-Extension-Pair
begin

```

### 4.1 List based characterization of *multpw*

```

lemma multpw-listI:
  assumes  $\text{length } xs = \text{length } ys \ X = \text{mset } xs \ Y = \text{mset } ys$ 
     $\forall i. i < \text{length } ys \longrightarrow (xs ! i, ys ! i) \in ns$ 
  shows  $(X, Y) \in \text{multpw } ns$ 
  <proof>

```

```

lemma multpw-listE:
  assumes  $(X, Y) \in \text{multpw } ns$ 
  obtains  $xs \ ys$  where  $\text{length } xs = \text{length } ys \ X = \text{mset } xs \ Y = \text{mset } ys$ 
     $\forall i. i < \text{length } ys \longrightarrow (xs ! i, ys ! i) \in ns$ 
  <proof>

```

### 4.2 Definition of the multiset extension of $>$ -orders

We define here the non-strict extension of the order pair  $(\geq, >)$  – usually written as  $(ns, s)$  in the sources – by just flipping the directions twice.

```

definition ns-mul-ext :: 'a rel  $\Rightarrow$  'a rel  $\Rightarrow$  'a multiset rel
  where  $ns\text{-mul-ext } ns \ s \equiv (\text{mult2-alt-ns } (ns^{-1}) \ (s^{-1}))^{-1}$ 

```

```

lemma ns-mul-extI:
  assumes  $A = A1 + A2$  and  $B = B1 + B2$ 
    and  $(A1, B1) \in \text{multpw } ns$ 
    and  $\bigwedge b. b \in \# B2 \implies \exists a. a \in \# A2 \wedge (a, b) \in s$ 
  shows  $(A, B) \in ns\text{-mul-ext } ns \ s$ 
  <proof>

```

```

lemma ns-mul-extE:
  assumes  $(A, B) \in ns\text{-mul-ext } ns \ s$ 
  obtains  $A1 \ A2 \ B1 \ B2$  where  $A = A1 + A2$  and  $B = B1 + B2$ 
    and  $(A1, B1) \in \text{multpw } ns$ 
    and  $\bigwedge b. b \in \# B2 \implies \exists a. a \in \# A2 \wedge (a, b) \in s$ 
  <proof>

```

**lemmas**  $ns\text{-mul-extI-old} = ns\text{-mul-extI}[OF - - \text{multpw-listI}[OF - refl refl], \text{rule-format}]$

Same for the "greater than" order on multisets.

**definition**  $s\text{-mul-ext} :: 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ multiset rel}$   
**where**  $s\text{-mul-ext } ns \ s \equiv (\text{mult2-alt-s } (ns^{-1}) \ (s^{-1}))^{-1}$

**lemma**  $s\text{-mul-extI}$ :

**assumes**  $A = A1 + A2$  **and**  $B = B1 + B2$

**and**  $(A1, B1) \in \text{multpw } ns$

**and**  $A2 \neq \{\#\}$  **and**  $\bigwedge b. b \in \# B2 \implies \exists a. a \in \# A2 \wedge (a, b) \in s$

**shows**  $(A, B) \in s\text{-mul-ext } ns \ s$

$\langle \text{proof} \rangle$

**lemma**  $s\text{-mul-extE}$ :

**assumes**  $(A, B) \in s\text{-mul-ext } ns \ s$

**obtains**  $A1 \ A2 \ B1 \ B2$  **where**  $A = A1 + A2$  **and**  $B = B1 + B2$

**and**  $(A1, B1) \in \text{multpw } ns$

**and**  $A2 \neq \{\#\}$  **and**  $\bigwedge b. b \in \# B2 \implies \exists a. a \in \# A2 \wedge (a, b) \in s$

$\langle \text{proof} \rangle$

**lemmas**  $s\text{-mul-extI-old} = s\text{-mul-extI}[OF - - \text{multpw-listI}[OF - refl refl], \text{rule-format}]$

### 4.3 Basic properties

**lemma**  $s\text{-mul-ext-mono}$ :

**assumes**  $ns \subseteq ns' \ s \subseteq s'$  **shows**  $s\text{-mul-ext } ns \ s \subseteq s\text{-mul-ext } ns' \ s'$

$\langle \text{proof} \rangle$

**lemma**  $ns\text{-mul-ext-mono}$ :

**assumes**  $ns \subseteq ns' \ s \subseteq s'$  **shows**  $ns\text{-mul-ext } ns \ s \subseteq ns\text{-mul-ext } ns' \ s'$

$\langle \text{proof} \rangle$

**lemma**  $s\text{-mul-ext-local-mono}$ :

**assumes**  $\text{sub}: (\text{set-mset } xs \times \text{set-mset } ys) \cap ns \subseteq ns' (\text{set-mset } xs \times \text{set-mset } ys)$

$\cap s \subseteq s'$

**and**  $\text{rel}: (xs, ys) \in s\text{-mul-ext } ns \ s$

**shows**  $(xs, ys) \in s\text{-mul-ext } ns' \ s'$

$\langle \text{proof} \rangle$

**lemma**  $ns\text{-mul-ext-local-mono}$ :

**assumes**  $\text{sub}: (\text{set-mset } xs \times \text{set-mset } ys) \cap ns \subseteq ns' (\text{set-mset } xs \times \text{set-mset } ys)$

$\cap s \subseteq s'$

**and**  $\text{rel}: (xs, ys) \in ns\text{-mul-ext } ns \ s$

**shows**  $(xs, ys) \in ns\text{-mul-ext } ns' \ s'$

$\langle \text{proof} \rangle$

The empty multiset is the minimal element for these orders

**lemma**  $ns\text{-mul-ext-bottom}$ :  $(A, \{\#\}) \in ns\text{-mul-ext } ns \ s$

$\langle \text{proof} \rangle$

**lemma** *ns-mul-ext-bottom-uniqueness*:  
**assumes**  $(\{\#\}, A) \in ns\text{-mul-ext } ns \ s$   
**shows**  $A = \{\#\}$   
 $\langle proof \rangle$

**lemma** *ns-mul-ext-bottom2*:  
**assumes**  $(A, B) \in ns\text{-mul-ext } ns \ s$   
**and**  $B \neq \{\#\}$   
**shows**  $A \neq \{\#\}$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-bottom*:  
**assumes**  $A \neq \{\#\}$   
**shows**  $(A, \{\#\}) \in s\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-bottom-strict*:  
 $(\{\#\}, A) \notin s\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

Obvious introduction rules.

**lemma** *all-ns-ns-mul-ext*:  
**assumes**  $length \ as = length \ bs$   
**and**  $\forall i. i < length \ bs \longrightarrow (as \ ! \ i, bs \ ! \ i) \in ns$   
**shows**  $(mset \ as, mset \ bs) \in ns\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

**lemma** *all-s-s-mul-ext*:  
**assumes**  $A \neq \{\#\}$   
**and**  $\forall b. b \in \# \ B \longrightarrow (\exists a. a \in \# \ A \wedge (a, b) \in s)$   
**shows**  $(A, B) \in s\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

Being strictly lesser than implies being lesser than

**lemma** *s-ns-mul-ext*:  
**assumes**  $(A, B) \in s\text{-mul-ext } ns \ s$   
**shows**  $(A, B) \in ns\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

The non-strict order is reflexive.

**lemma** *multpw-refl'*:  
**assumes** *locally-refl ns A*  
**shows**  $(A, A) \in multpw \ ns$   
 $\langle proof \rangle$

**lemma** *ns-mul-ext-refl-local*:  
**assumes** *locally-refl ns A*  
**shows**  $(A, A) \in ns\text{-mul-ext } ns \ s$



*<proof>*

**lemma** *ns-mul-ext-refl*:

**assumes** *refl ns*

**shows**  $(A, A) \in ns\text{-mul-ext } ns \ s$

*<proof>*

The orders are union-compatible

**lemma** *ns-s-mul-ext-union-multiset-l*:

**assumes**  $(A, B) \in ns\text{-mul-ext } ns \ s$

**and**  $C \neq \{\#\}$

**and**  $\forall d. d \in\# D \longrightarrow (\exists c. c \in\# C \wedge (c,d) \in s)$

**shows**  $(A + C, B + D) \in s\text{-mul-ext } ns \ s$

*<proof>*

**lemma** *s-mul-ext-union-compat*:

**assumes**  $(A, B) \in s\text{-mul-ext } ns \ s$

**and** *locally-refl ns C*

**shows**  $(A + C, B + C) \in s\text{-mul-ext } ns \ s$

*<proof>*

**lemma** *ns-mul-ext-union-compat*:

**assumes**  $(A, B) \in ns\text{-mul-ext } ns \ s$

**and** *locally-refl ns C*

**shows**  $(A + C, B + C) \in ns\text{-mul-ext } ns \ s$

*<proof>*

**context**

**fixes** *NS :: 'a rel*

**assumes** *NS: refl NS*

**begin**

**lemma** *refl-imp-locally-refl*: *locally-refl NS A* *<proof>*

**lemma** *supseteq-imp-ns-mul-ext*:

**assumes**  $A \supseteq\# B$

**shows**  $(A, B) \in ns\text{-mul-ext } NS \ S$

*<proof>*

**lemma** *supset-imp-s-mul-ext*:

**assumes**  $A \supset\# B$

**shows**  $(A, B) \in s\text{-mul-ext } NS \ S$

*<proof>*

**end**

**definition** *mul-ext* ::  $('a \Rightarrow 'a \Rightarrow bool \times bool) \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool \times bool$

**where** *mul-ext f xs ys*  $\equiv$  *let*  $s = \{(x,y). fst (f x y)\}$ ;  $ns = \{(x,y). snd (f x y)\}$

*in*  $((mset xs, mset ys) \in s\text{-mul-ext } ns \ s, (mset xs, mset ys) \in ns\text{-mul-ext } ns \ s)$

**definition**  $smulextp\ f\ m\ n \longleftrightarrow (m, n) \in s\text{-mul-ext}\ \{(x, y).\ snd\ (f\ x\ y)\}\ \{(x, y).\ fst\ (f\ x\ y)\}$   
**definition**  $nsmulextp\ f\ m\ n \longleftrightarrow (m, n) \in ns\text{-mul-ext}\ \{(x, y).\ snd\ (f\ x\ y)\}\ \{(x, y).\ fst\ (f\ x\ y)\}$

**lemma**  $smulextp\text{-cong}[fundef\text{-cong}]$ :  
**assumes**  $xs1 = ys1$   
**and**  $xs2 = ys2$   
**and**  $\bigwedge x\ x'.\ x \in\# ys1 \implies x' \in\# ys2 \implies f\ x\ x' = g\ x\ x'$   
**shows**  $smulextp\ f\ xs1\ xs2 = smulextp\ g\ ys1\ ys2$   
 $\langle proof \rangle$

**lemma**  $nsmulextp\text{-cong}[fundef\text{-cong}]$ :  
**assumes**  $xs1 = ys1$   
**and**  $xs2 = ys2$   
**and**  $\bigwedge x\ x'.\ x \in\# ys1 \implies x' \in\# ys2 \implies f\ x\ x' = g\ x\ x'$   
**shows**  $nsmulextp\ f\ xs1\ xs2 = nsmulextp\ g\ ys1\ ys2$   
 $\langle proof \rangle$

**definition**  $mulextp\ f\ m\ n = (smulextp\ f\ m\ n,\ nsmulextp\ f\ m\ n)$

**lemma**  $mulextp\text{-cong}[fundef\text{-cong}]$ :  
**assumes**  $xs1 = ys1$   
**and**  $xs2 = ys2$   
**and**  $\bigwedge x\ x'.\ x \in\# ys1 \implies x' \in\# ys2 \implies f\ x\ x' = g\ x\ x'$   
**shows**  $mulextp\ f\ xs1\ xs2 = mulextp\ g\ ys1\ ys2$   
 $\langle proof \rangle$

**lemma**  $mset\text{-s-mul-ext}$ :  
 $(mset\ xs,\ mset\ ys) \in s\text{-mul-ext}\ \{(x, y).\ snd\ (f\ x\ y)\}\ \{(x, y).\ fst\ (f\ x\ y)\} \longleftrightarrow$   
 $\quad fst\ (mul\text{-ext}\ f\ xs\ ys)$   
 $\langle proof \rangle$

**lemma**  $mset\text{-ns-mul-ext}$ :  
 $(mset\ xs,\ mset\ ys) \in ns\text{-mul-ext}\ \{(x, y).\ snd\ (f\ x\ y)\}\ \{(x, y).\ fst\ (f\ x\ y)\} \longleftrightarrow$   
 $\quad snd\ (mul\text{-ext}\ f\ xs\ ys)$   
 $\langle proof \rangle$

**lemma**  $smulextp\text{-mset-code}$ :  
 $smulextp\ f\ (mset\ xs)\ (mset\ ys) \longleftrightarrow fst\ (mul\text{-ext}\ f\ xs\ ys)$   
 $\langle proof \rangle$

**lemma**  $nsmulextp\text{-mset-code}$ :  
 $nsmulextp\ f\ (mset\ xs)\ (mset\ ys) \longleftrightarrow snd\ (mul\text{-ext}\ f\ xs\ ys)$   
 $\langle proof \rangle$

**lemma** *nstri-mul-ext-map*:

**assumes**  $\bigwedge s t. s \in \text{set } ss \implies t \in \text{set } ts \implies \text{fst } (\text{order } s t) \implies \text{fst } (\text{order}' (f s) (f t))$   
**and**  $\bigwedge s t. s \in \text{set } ss \implies t \in \text{set } ts \implies \text{snd } (\text{order } s t) \implies \text{snd } (\text{order}' (f s) (f t))$   
**and**  $\text{snd } (\text{mul-ext } \text{order } ss ts)$   
**shows**  $\text{snd } (\text{mul-ext } \text{order}' (map f ss) (map f ts))$   
*<proof>*

**lemma** *stri-mul-ext-map*:

**assumes**  $\bigwedge s t. s \in \text{set } ss \implies t \in \text{set } ts \implies \text{fst } (\text{order } s t) \implies \text{fst } (\text{order}' (f s) (f t))$   
**and**  $\bigwedge s t. s \in \text{set } ss \implies t \in \text{set } ts \implies \text{snd } (\text{order } s t) \implies \text{snd } (\text{order}' (f s) (f t))$   
**and**  $\text{fst } (\text{mul-ext } \text{order } ss ts)$   
**shows**  $\text{fst } (\text{mul-ext } \text{order}' (map f ss) (map f ts))$   
*<proof>*

The non-strict order is transitive.

**lemma** *ns-mul-ext-trans*:

**assumes**  $\text{trans } s \text{ trans } ns \text{ compatible-l } ns s \text{ compatible-r } ns s \text{ refl } ns$   
**and**  $(A, B) \in ns\text{-mul-ext } ns s$   
**and**  $(B, C) \in ns\text{-mul-ext } ns s$   
**shows**  $(A, C) \in ns\text{-mul-ext } ns s$   
*<proof>*

The strict order is trans.

**lemma** *s-mul-ext-trans*:

**assumes**  $\text{trans } s \text{ trans } ns \text{ compatible-l } ns s \text{ compatible-r } ns s \text{ refl } ns$   
**and**  $(A, B) \in s\text{-mul-ext } ns s$   
**and**  $(B, C) \in s\text{-mul-ext } ns s$   
**shows**  $(A, C) \in s\text{-mul-ext } ns s$   
*<proof>*

The strict order is compatible on the left with the non strict one

**lemma** *s-ns-mul-ext-trans*:

**assumes**  $\text{trans } s \text{ trans } ns \text{ compatible-l } ns s \text{ compatible-r } ns s \text{ refl } ns$   
**and**  $(A, B) \in s\text{-mul-ext } ns s$   
**and**  $(B, C) \in ns\text{-mul-ext } ns s$   
**shows**  $(A, C) \in s\text{-mul-ext } ns s$   
*<proof>*

The strict order is compatible on the right with the non-strict one.

**lemma** *ns-s-mul-ext-trans*:

**assumes**  $\text{trans } s \text{ trans } ns \text{ compatible-l } ns s \text{ compatible-r } ns s \text{ refl } ns$   
**and**  $(A, B) \in ns\text{-mul-ext } ns s$   
**and**  $(B, C) \in s\text{-mul-ext } ns s$   
**shows**  $(A, C) \in ns\text{-mul-ext } ns s$   
*<proof>*

*s-mul-ext* is strongly normalizing

**lemma** *SN-s-mul-ext-strong*:

**assumes** *order-pair s ns*  
**and**  $\forall y. y \in \# M \longrightarrow \text{SN-on } s \{y\}$   
**shows** *SN-on (s-mul-ext ns s) {M}*  
 $\langle \text{proof} \rangle$

**lemma** *SN-s-mul-ext*:

**assumes** *order-pair s ns SN s*  
**shows** *SN (s-mul-ext ns s)*  
 $\langle \text{proof} \rangle$

**lemma** (**in** *order-pair*) *mul-ext-order-pair*:

*order-pair (s-mul-ext NS S) (ns-mul-ext NS S) (is order-pair ?S ?NS)*  
 $\langle \text{proof} \rangle$

**lemma** (**in** *SN-order-pair*) *mul-ext-SN-order-pair*: *SN-order-pair (s-mul-ext NS S) (ns-mul-ext NS S)*

(**is** *SN-order-pair ?S ?NS*)  
 $\langle \text{proof} \rangle$

**lemma** *mul-ext-compat*:

**assumes** *compat*:  $\bigwedge s t u. \llbracket s \in \text{set } ss; t \in \text{set } ts; u \in \text{set } us \rrbracket \implies$   
 $(\text{snd } (f s t) \wedge \text{fst } (f t u) \longrightarrow \text{fst } (f s u)) \wedge$   
 $(\text{fst } (f s t) \wedge \text{snd } (f t u) \longrightarrow \text{fst } (f s u)) \wedge$   
 $(\text{snd } (f s t) \wedge \text{snd } (f t u) \longrightarrow \text{snd } (f s u)) \wedge$   
 $(\text{fst } (f s t) \wedge \text{fst } (f t u) \longrightarrow \text{fst } (f s u))$

**shows**

$(\text{snd } (\text{mul-ext } f ss ts) \wedge \text{fst } (\text{mul-ext } f ts us) \longrightarrow \text{fst } (\text{mul-ext } f ss us)) \wedge$   
 $(\text{fst } (\text{mul-ext } f ss ts) \wedge \text{snd } (\text{mul-ext } f ts us) \longrightarrow \text{fst } (\text{mul-ext } f ss us)) \wedge$   
 $(\text{snd } (\text{mul-ext } f ss ts) \wedge \text{snd } (\text{mul-ext } f ts us) \longrightarrow \text{snd } (\text{mul-ext } f ss us)) \wedge$   
 $(\text{fst } (\text{mul-ext } f ss ts) \wedge \text{fst } (\text{mul-ext } f ts us) \longrightarrow \text{fst } (\text{mul-ext } f ss us))$

$\langle \text{proof} \rangle$

**lemma** *mul-ext-cong[fundef-cong]*:

**assumes** *mset xs1 = mset ys1*  
**and** *mset xs2 = mset ys2*  
**and**  $\bigwedge x x'. x \in \text{set } ys1 \implies x' \in \text{set } ys2 \implies f x x' = g x x'$   
**shows** *mul-ext f xs1 xs2 = mul-ext g ys1 ys2*  
 $\langle \text{proof} \rangle$

**lemma** *all-nstri-imp-mul-nstri*:

**assumes**  $\forall i < \text{length } ys. \text{snd } (f (xs ! i) (ys ! i))$   
**and** *length xs = length ys*  
**shows** *snd (mul-ext f xs ys)*  
 $\langle \text{proof} \rangle$

**lemma** *relation-inter*:

**shows**  $\{(x,y). P x y\} \cap \{(x,y). Q x y\} = \{(x,y). P x y \wedge Q x y\}$

*<proof>*

**lemma** *mul-ext-unfold*:

$(x,y) \in \{(a,b).fst (mul-ext\ g\ a\ b)\} \longleftrightarrow (mset\ x, mset\ y) \in (s-mul-ext\ \{(a,b).snd (g\ a\ b)\} \{(a,b).fst (g\ a\ b)\})$   
*<proof>*

The next lemma is a local version of strong-normalization of the multi-set extension, where the base-order only has to be strongly normalizing on elements of the multisets. This will be crucial for orders that are defined recursively on terms, such as RPO or WPO.

**lemma** *mul-ext-SN*:

**assumes**  $\forall x. snd (g\ x\ x)$   
**and**  $\forall x\ y\ z. fst (g\ x\ y) \longrightarrow snd (g\ y\ z) \longrightarrow fst (g\ x\ z)$   
**and**  $\forall x\ y\ z. snd (g\ x\ y) \longrightarrow fst (g\ y\ z) \longrightarrow fst (g\ x\ z)$   
**and**  $\forall x\ y\ z. snd (g\ x\ y) \longrightarrow snd (g\ y\ z) \longrightarrow snd (g\ x\ z)$   
**and**  $\forall x\ y\ z. fst (g\ x\ y) \longrightarrow fst (g\ y\ z) \longrightarrow fst (g\ x\ z)$   
**shows**  $SN\ \{(ys, xs).$   
 $(\forall y \in set\ ys. SN-on\ \{(s, t). fst (g\ s\ t)\}\ \{y\}) \wedge$   
 $fst (mul-ext\ g\ ys\ xs)\}$   
*<proof>*

**lemma** *mul-ext-stri-imp-nstri*:

**assumes**  $fst (mul-ext\ f\ as\ bs)$   
**shows**  $snd (mul-ext\ f\ as\ bs)$   
*<proof>*

**lemma** *ns-ns-mul-ext-union-compat*:

**assumes**  $(A,B) \in ns-mul-ext\ ns\ s$   
**and**  $(C,D) \in ns-mul-ext\ ns\ s$   
**shows**  $(A + C, B + D) \in ns-mul-ext\ ns\ s$   
*<proof>*

**lemma** *s-ns-mul-ext-union-compat*:

**assumes**  $(A,B) \in s-mul-ext\ ns\ s$   
**and**  $(C,D) \in ns-mul-ext\ ns\ s$   
**shows**  $(A + C, B + D) \in s-mul-ext\ ns\ s$   
*<proof>*

**lemma** *ns-ns-mul-ext-union-compat-rtrancl*: **assumes** *refl*:  $refl\ ns$

**and**  $AB: (A, B) \in (ns-mul-ext\ ns\ s)^*$   
**and**  $CD: (C, D) \in (ns-mul-ext\ ns\ s)^*$   
**shows**  $(A + C, B + D) \in (ns-mul-ext\ ns\ s)^*$   
*<proof>*

#### 4.4 Multisets as order on lists

**interpretation** *mul-ext-list*: *list-order-extension*

$\lambda s\ ns. \{(as, bs). (mset\ as, mset\ bs) \in s-mul-ext\ ns\ s\}$

$\lambda s ns. \{(as, bs). (mset as, mset bs) \in ns\text{-mul-ext } ns \ s\}$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-singleton* [*simp, intro*]:  
**assumes**  $(a, b) \in s$   
**shows**  $(\{ \#a \}, \{ \#b \}) \in s\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

**lemma** *ns-mul-ext-singleton* [*simp, intro*]:  
 $(a, b) \in ns \implies (\{ \#a \}, \{ \#b \}) \in ns\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

**lemma** *ns-mul-ext-singleton2*:  
 $(a, b) \in s \implies (\{ \#a \}, \{ \#b \}) \in ns\text{-mul-ext } ns \ s$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-self-extend-left*:  
**assumes**  $A \neq \{ \# \}$  **and** *locally-refl*  $W \ B$   
**shows**  $(A + B, B) \in s\text{-mul-ext } W \ S$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-ne-extend-left*:  
**assumes**  $A \neq \{ \# \}$  **and**  $(B, C) \in ns\text{-mul-ext } W \ S$   
**shows**  $(A + B, C) \in s\text{-mul-ext } W \ S$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-extend-left*:  
**assumes**  $(B, C) \in s\text{-mul-ext } W \ S$   
**shows**  $(A + B, C) \in s\text{-mul-ext } W \ S$   
 $\langle proof \rangle$

**lemma** *mul-ext-mono*:  
**assumes**  $\bigwedge x y. \llbracket x \in set \ xs; y \in set \ ys; fst \ (P \ x \ y) \rrbracket \implies fst \ (P' \ x \ y)$   
**and**  $\bigwedge x y. \llbracket x \in set \ xs; y \in set \ ys; snd \ (P \ x \ y) \rrbracket \implies snd \ (P' \ x \ y)$   
**shows**  
 $fst \ (mul\text{-ext } P \ xs \ ys) \implies fst \ (mul\text{-ext } P' \ xs \ ys)$   
 $snd \ (mul\text{-ext } P \ xs \ ys) \implies snd \ (mul\text{-ext } P' \ xs \ ys)$   
 $\langle proof \rangle$

## 4.5 Special case: non-strict order is equality

**lemma** *ns-mul-ext-IdE*:  
**assumes**  $(M, N) \in ns\text{-mul-ext } Id \ R$   
**obtains**  $X$  **and**  $Y$  **and**  $Z$  **where**  $M = X + Z$  **and**  $N = Y + Z$   
**and**  $\forall y \in set\text{-mset } Y. \exists x \in set\text{-mset } X. (x, y) \in R$   
 $\langle proof \rangle$

**lemma** *ns-mul-ext-IdI*:  
**assumes**  $M = X + Z$  **and**  $N = Y + Z$  **and**  $\forall y \in set\text{-mset } Y. \exists x \in set\text{-mset}$

*X.*  $(x, y) \in R$   
**shows**  $(M, N) \in ns\text{-mul-ext Id } R$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-IdE*:  
**assumes**  $(M, N) \in s\text{-mul-ext Id } R$   
**obtains**  $X$  **and**  $Y$  **and**  $Z$  **where**  $X \neq \{\#\}$  **and**  $M = X + Z$  **and**  $N = Y + Z$   
**and**  $\forall y \in \text{set-mset } Y. \exists x \in \text{set-mset } X. (x, y) \in R$   
 $\langle proof \rangle$

**lemma** *s-mul-ext-IdI*:  
**assumes**  $X \neq \{\#\}$  **and**  $M = X + Z$  **and**  $N = Y + Z$   
**and**  $\forall y \in \text{set-mset } Y. \exists x \in \text{set-mset } X. (x, y) \in R$   
**shows**  $(M, N) \in s\text{-mul-ext Id } R$   
 $\langle proof \rangle$

**lemma** *mult-s-mul-ext-conv*:  
**assumes** *trans*  $R$   
**shows**  $(\text{mult } (R^{-1}))^{-1} = s\text{-mul-ext Id } R$   
 $\langle proof \rangle$

**lemma** *ns-mul-ext-Id-eq*:  
 $ns\text{-mul-ext Id } R = (s\text{-mul-ext Id } R)^{=}$   
 $\langle proof \rangle$

**lemma** *subsetq-mset-imp-ns-mul-ext-Id*:  
**assumes**  $A \subseteq\# B$   
**shows**  $(B, A) \in ns\text{-mul-ext Id } R$   
 $\langle proof \rangle$

**lemma** *subset-mset-imp-s-mul-ext-Id*:  
**assumes**  $A \subset\# B$   
**shows**  $(B, A) \in s\text{-mul-ext Id } R$   
 $\langle proof \rangle$

**end**

## 4.6 Executable version

**theory** *Multiset-Extension2-Impl*  
**imports**  
*HOL-Library.DAList-Multiset*  
*List-Order*  
*Multiset-Extension2*  
*Multiset-Extension-Pair-Impl*  
**begin**

**lemma** *mul-ext-list-ext*:  $\exists s \text{ ns. list-order-extension-impl } s \text{ ns } \text{mul-ext}$   
 ⟨*proof*⟩

**context fixes** *sns* :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool  $\times$  bool  
**begin**

**fun** *mul-ext-impl* :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  bool  $\times$  bool  
**and** *mul-ex-dom* :: 'a list  $\Rightarrow$  'a list  $\Rightarrow$  'a  $\Rightarrow$  'a list  $\Rightarrow$  bool  $\times$  bool

**where**

*mul-ext-impl* [] [] = (False, True)  
 | *mul-ext-impl* [] ys = (False, False)  
 | *mul-ext-impl* xs [] = (True, True)  
 | *mul-ext-impl* xs (y # ys) = *mul-ex-dom* xs [] y ys  
  
 | *mul-ex-dom* [] xs' y ys = (False, False)  
 | *mul-ex-dom* (x # xs) xs' y ys =  
   (case *sns* x y of  
     (True, -)  $\Rightarrow$  if snd (*mul-ext-impl* (xs @ xs') (filter ( $\lambda y. \neg \text{fst } (\text{sns } x \text{ } y)$ ) ys))  
     then (True, True)  
     else *mul-ex-dom* xs (x # xs') y ys  
   | (False, True)  $\Rightarrow$  or2 (*mul-ext-impl* (xs @ xs') ys) (*mul-ex-dom* xs (x # xs') y  
 ys)  
 | -  $\Rightarrow$  *mul-ex-dom* xs (x # xs') y ys)

**end**

**context**

**begin**

**lemma** *mul-ext-impl-sound0*:

*mul-ext-impl* *sns* *xs* *ys* = *mult2-impl* ( $\lambda x \ y. \text{sns } y \ x$ ) *ys* *xs*  
*mul-ex-dom* *sns* *xs* *xs'* *y* *ys* = *mult2-ex-dom* ( $\lambda x \ y. \text{sns } y \ x$ ) *y* *ys* *xs* *xs'*

⟨*proof*⟩ **definition** *cond1* **where**

*cond1* *f* *bs* *y* *xs* *ys*  $\equiv$   
 (( $\exists b. b \in \text{set } bs \wedge \text{fst } (f \ b \ y) \wedge \text{snd } (\text{mul-ext } f \ (\text{remove1 } b \ xs) \ [y \leftarrow ys \ . \ \neg \text{fst } (f \ b \ y)])$ ))  
 $\vee (\exists b. b \in \text{set } bs \wedge \text{snd } (f \ b \ y) \wedge \text{fst } (\text{mul-ext } f \ (\text{remove1 } b \ xs) \ ys))$ )

**private lemma** *cond1-propagate*:

**assumes** *cond1* *f* *bs* *y* *xs* *ys*

**shows** *cond1* *f* (b # *bs*) *y* *xs* *ys*

⟨*proof*⟩ **definition** *cond2* **where**

*cond2* *f* *bs* *y* *xs* *ys*  $\equiv$  (*cond1* *f* *bs* *y* *xs* *ys*)

$\vee (\exists b. b \in \text{set } bs \wedge \text{snd } (f \ b \ y) \wedge \text{snd } (\text{mul-ext } f \ (\text{remove1 } b \ xs) \ ys))$ )

**private lemma** *cond2-propagate*:

**assumes** *cond2* *f* *bs* *y* *xs* *ys*

**shows** *cond2* *f* (b # *bs*) *y* *xs* *ys*

⟨*proof*⟩ **lemma** *cond1-cond2*:



**assumes**  $cond1\ f\ bs\ y\ xs\ ys$   
**shows**  $cond2\ f\ bs\ y\ xs\ ys$   
 $\langle proof \rangle$

**lemma** *mul-ext-impl-sound*:  
**shows**  $mul-ext-impl\ f\ xs\ ys = mul-ext\ f\ xs\ ys$   
 $\langle proof \rangle$

**lemma** *mul-ext-code* [code]:  $mul-ext = mul-ext-impl$   
 $\langle proof \rangle$

**lemma** *mul-ext-impl-cong*[*fundef-cong*]:  
**assumes**  $\bigwedge x\ x'. x \in set\ xs \implies x' \in set\ ys \implies f\ x\ x' = g\ x\ x'$   
**shows**  $mul-ext-impl\ f\ xs\ ys = mul-ext-impl\ g\ xs\ ys$   
 $\langle proof \rangle$   
**end**

**fun** *ass-list-to-single-list* ::  $('a \times nat)\ list \Rightarrow 'a\ list$   
**where**  
 $ass-list-to-single-list\ [] = []$   
 $| ass-list-to-single-list\ ((x, n) \# xs) = replicate\ n\ x @ ass-list-to-single-list\ xs$

**lemma** *set-ass-list-to-single-list* [*simp*]:  
 $set\ (ass-list-to-single-list\ xs) = \{x. \exists n. (x, n) \in set\ xs \wedge n > 0\}$   
 $\langle proof \rangle$

**lemma** *count-mset-replicate* [*simp*]:  
 $count\ (mset\ (replicate\ n\ x))\ x = n$   
 $\langle proof \rangle$

**lemma** *count-mset-lal-ge*:  
 $(x, n) \in set\ xs \implies count\ (mset\ (ass-list-to-single-list\ xs))\ x \geq n$   
 $\langle proof \rangle$

**lemma** *count-of-count-mset-lal* [*simp*]:  
 $distinct\ (map\ fst\ y) \implies count-of\ y\ x = count\ (mset\ (ass-list-to-single-list\ y))\ x$   
 $\langle proof \rangle$

**lemma** *Bag-mset*:  $Bag\ xs = mset\ (ass-list-to-single-list\ (DAList.impl-of\ xs))$   
 $\langle proof \rangle$

**lemma** *Bag-Alist-Cons*:  
 $x \notin fst\ `set\ xs \implies distinct\ (map\ fst\ xs) \implies$   
 $Bag\ (Alist\ ((x, n) \# xs)) = mset\ (replicate\ n\ x) + Bag\ (Alist\ xs)$   
 $\langle proof \rangle$

**lemma** *mset-lal* [*simp*]:  
 $distinct\ (map\ fst\ xs) \implies mset\ (ass-list-to-single-list\ xs) = Bag\ (Alist\ xs)$   
 $\langle proof \rangle$

**lemma** *Bag-s-mul-ext*:

$(Bag\ xs, Bag\ ys) \in s\text{-mul-ext}\ \{(x, y). snd\ (f\ x\ y)\}\ \{(x, y). fst\ (f\ x\ y)\} \longleftrightarrow$   
 $\text{fst}\ (mul\text{-ext}\ f\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}\ xs))\ (ass\text{-list-to-single-list}$   
 $(DAList.impl\text{-of}\ ys)))$   
*<proof>*

**lemma** *Bag-ns-mul-ext*:

$(Bag\ xs, Bag\ ys) \in ns\text{-mul-ext}\ \{(x, y). snd\ (f\ x\ y)\}\ \{(x, y). fst\ (f\ x\ y)\} \longleftrightarrow$   
 $snd\ (mul\text{-ext}\ f\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}\ xs))\ (ass\text{-list-to-single-list}$   
 $(DAList.impl\text{-of}\ ys)))$   
*<proof>*

**lemma** *smulextp-code*[code]:

$smulextp\ f\ (Bag\ xs)\ (Bag\ ys) \longleftrightarrow \text{fst}\ (mul\text{-ext}\ f\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}$   
 $xs))\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}\ ys)))$   
*<proof>*

**lemma** *nsmulextp-code*[code]:

$nsmulextp\ f\ (Bag\ xs)\ (Bag\ ys) \longleftrightarrow \text{snd}\ (mul\text{-ext}\ f\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}$   
 $xs))\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}\ ys)))$   
*<proof>*

**lemma** *mulextp-code*[code]:

$mulextp\ f\ (Bag\ xs)\ (Bag\ ys) = mul\text{-ext}\ f\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}$   
 $xs))\ (ass\text{-list-to-single-list}\ (DAList.impl\text{-of}\ ys))$   
*<proof>*

**end**

## 5 The Weighted Path Order

This is a version of WPO that also permits multiset comparisons of lists of terms. It therefore generalizes RPO.

**theory** *WPO*

**imports**

*Knuth-Bendix-Order.Lexicographic-Extension*

*Knuth-Bendix-Order.Term-Aux*

*Knuth-Bendix-Order.Order-Pair*

*Polynomial-Factorization.Missing-List*

*Status*

*Precedence*

*Multiset-Extension2*

**begin**

**datatype** *order-tag* = *Lex* | *Mul*

**locale** *wpo* =

```

fixes  $n :: \text{nat}$ 
  and  $S\ NS :: ('f, 'v) \text{ term rel}$ 
  and  $\text{prc} :: ('f \times \text{nat} \Rightarrow 'f \times \text{nat} \Rightarrow \text{bool} \times \text{bool})$ 
  and  $\text{prl} :: 'f \times \text{nat} \Rightarrow \text{bool}$ 
  and  $\sigma\sigma :: 'f \text{ status}$ 
  and  $c :: 'f \times \text{nat} \Rightarrow \text{order-tag}$ 
  and  $\text{ssimple} :: \text{bool}$ 
  and  $\text{large} :: 'f \times \text{nat} \Rightarrow \text{bool}$ 
begin

fun  $\text{wpo} :: ('f, 'v) \text{ term} \Rightarrow ('f, 'v) \text{ term} \Rightarrow \text{bool} \times \text{bool}$ 
  where
     $\text{wpo } s\ t =$ 
    (case  $s$  of
       $\text{Var } x \Rightarrow (\text{False},$ 
        (case  $t$  of
           $\text{Var } y \Rightarrow x = y$ 
          |  $\text{Fun } g\ ts \Rightarrow (s, t) \in NS \wedge \text{status } \sigma\sigma (g, \text{length } ts) = [] \wedge \text{prl} (g, \text{length } ts)))$ 
      |  $\text{Fun } f\ ss \Rightarrow$ 
        if  $(s, t) \in S$  then  $(\text{True}, \text{True})$ 
        else if  $(s, t) \in NS$  then
          if  $\exists i \in \text{set } (\text{status } \sigma\sigma (f, \text{length } ss)). \text{snd } (\text{wpo } (ss ! i) t)$  then  $(\text{True}, \text{True})$ 
          else
            (case  $t$  of
               $\text{Var } - \Rightarrow (\text{False}, \text{ssimple} \wedge \text{large } (f, \text{length } ss))$ 
              |  $\text{Fun } g\ ts \Rightarrow$ 
                (case  $\text{prc } (f, \text{length } ss) (g, \text{length } ts)$  of  $(\text{pr}_s, \text{pr}_n) \Rightarrow$ 
                  if  $\text{pr}_n \wedge (\forall j \in \text{set } (\text{status } \sigma\sigma (g, \text{length } ts)). \text{fst } (\text{wpo } s (ts ! j)))$  then
                    if  $\text{pr}_s$  then  $(\text{True}, \text{True})$ 
                    else let  $ss' = \text{map } (\lambda i. ss ! i) (\text{status } \sigma\sigma (f, \text{length } ss));$ 
                       $ts' = \text{map } (\lambda i. ts ! i) (\text{status } \sigma\sigma (g, \text{length } ts));$ 
                       $cf = c (f, \text{length } ss);$ 
                       $cg = c (g, \text{length } ts)$ 
                      in if  $cf = \text{Lex} \wedge cg = \text{Lex}$ 
                        then  $\text{lex-ext wpo } n\ ss'\ ts'$ 
                        else if  $cf = \text{Mul} \wedge cg = \text{Mul}$ 
                          then  $\text{mul-ext wpo } ss'\ ts'$ 
                          else  $(\text{length } ss' \neq 0 \wedge \text{length } ts' = 0, \text{length } ts' = 0)$ 
                        else  $(\text{False}, \text{False}))$ 
                else  $(\text{False}, \text{False}))$ 
    )
  )

```

**declare**  $\text{wpo.simps}$  [*simp del*]

**abbreviation**  $\text{wpo-s}$  (**infix**  $\succ$  50) **where**  $s \succ t \equiv \text{fst } (\text{wpo } s\ t)$

**abbreviation**  $\text{wpo-ns}$  (**infix**  $\succeq$  50) **where**  $s \succeq t \equiv \text{snd } (\text{wpo } s\ t)$

**abbreviation**  $\text{WPO-S} \equiv \{(s, t). s \succ t\}$

**abbreviation**  $\text{WPO-NS} \equiv \{(s, t). s \succeq t\}$

**lemma** *wpo-s-imp-ns*:  $s \succ t \implies s \succeq t$   
 ⟨*proof*⟩

**end**

**declare** *wpo.wpo.simps*[*code*]

**definition** *strictly-simple-status* :: '*f status*  $\implies$  ('*f,v*)*term rel*  $\implies$  bool' **where**  
*strictly-simple-status*  $\sigma$  *rel* =  
 ( $\forall f$  *ts* *i*.  $i \in \text{set}(\text{status } \sigma (f, \text{length } ts)) \longrightarrow (\text{Fun } f \text{ } ts, ts ! i) \in \text{rel}$ )

**locale** *wpo-with-assms* = *wpo* +  
*SN-order-pair* + *precedence* +

**constrains** *S* :: ('*f, 'v*) *term rel* **and** *NS* :: -

**and** *prc* :: '*f*  $\times$  *nat*  $\implies$  '*f*  $\times$  *nat*  $\implies$  bool  $\times$  bool

**and** *prl* :: '*f*  $\times$  *nat*  $\implies$  bool

**and** *ssimple* :: bool

**and** *large* :: '*f*  $\times$  *nat*  $\implies$  bool

**and** *c* :: '*f*  $\times$  *nat*  $\implies$  *order-tag*

**and** *n* :: *nat*

**and**  $\sigma\sigma$  :: '*f status*

**assumes** *subst-S*:  $(s,t) \in S \implies (s \cdot \sigma, t \cdot \sigma) \in S$

**and** *subst-NS*:  $(s,t) \in NS \implies (s \cdot \sigma, t \cdot \sigma) \in NS$

**and** *ctxt-NS*:  $(s,t) \in NS \implies (\text{Fun } f (\text{bef } @ s \# \text{aft}), \text{Fun } f (\text{bef } @ t \# \text{aft})) \in$

*NS*

**and** *S-imp-NS*:  $S \subseteq NS$

**and** *ws-status*:  $i \in \text{set}(\text{status } \sigma\sigma \text{ } fn) \implies \text{simple-arg-pos } NS \text{ } fn \text{ } i$

**and** *large*:  $ssimple \implies \text{large } fn \implies \text{fst}(\text{prc } fn \text{ } gm) \vee \text{snd}(\text{prc } fn \text{ } gm) \wedge \text{status}$   
 $\sigma\sigma \text{ } gm = []$

**and** *large-trans*:  $ssimple \implies \text{large } fn \implies \text{snd}(\text{prc } gm \text{ } fn) \implies \text{large } gm$

**and** *ss-status*:  $ssimple \implies i \in \text{set}(\text{status } \sigma\sigma \text{ } fn) \implies \text{simple-arg-pos } S \text{ } fn \text{ } i$

**and** *ss-S-non-empty*:  $ssimple \implies S \neq \{\}$

**begin**

**lemma** *ss-NS-not-UNIV*:  $ssimple \implies NS \neq UNIV$

⟨*proof*⟩

**abbreviation**  $\sigma \equiv \text{status } \sigma\sigma$

**lemmas**  $\sigma = \text{status}$ [*of*  $\sigma\sigma$ ]

**lemma** *NS-arg*: **assumes** *i*:  $i \in \text{set}(\sigma (f, \text{length } ts))$

**shows**  $(\text{Fun } f \text{ } ts, ts ! i) \in NS$

⟨*proof*⟩

**lemma** *NS-subterm*: **assumes** *all*:  $\bigwedge f k. \text{set}(\sigma (f, k)) = \{0 ..< k\}$

**shows**  $s \succeq t \implies (s, t) \in NS$

$\langle proof \rangle$

**lemma**  $\sigma E$ :  $i \in set (\sigma (f, length\ ss)) \implies ss ! i \in set\ ss$   $\langle proof \rangle$

**lemma** *wpo-ns-reft*:

**shows**  $s \succeq s$

$\langle proof \rangle$

**lemma** *wpo-ns-imp-NS*:  $s \succeq t \implies (s,t) \in NS$

$\langle proof \rangle$

**lemma** *wpo-s-imp-NS*:  $s \succ t \implies (s,t) \in NS$

$\langle proof \rangle$

**lemma** *S-imp-wpo-s*: **assumes**  $st$ :  $(s,t) \in S$

**shows**  $s \succ t$

$\langle proof \rangle$

**lemma** *wpo-least-1*: **assumes**  $prl (f, length\ ss)$

**and**  $(t, Fun\ f\ ss) \in NS$

**and**  $\sigma (f, length\ ss) = []$

**shows**  $t \succeq Fun\ f\ ss$

$\langle proof \rangle$

**lemma** *wpo-least-2*: **assumes**  $prl (f, length\ ss)$  (**is**  $prl\ ?f$ )

**and**  $(Fun\ f\ ss, t) \notin S$

**and**  $\sigma (f, length\ ss) = []$

**shows**  $\neg Fun\ f\ ss \succ t$

$\langle proof \rangle$

**lemma** *wpo-least-3*: **assumes**  $prl (f, length\ ss)$  (**is**  $prl\ ?f$ )

**and**  $ns$ :  $Fun\ f\ ss \succeq t$

**and**  $NS$ :  $(u, Fun\ f\ ss) \in NS$

**and**  $ss$ :  $\sigma (f, length\ ss) = []$

**and**  $S$ :  $\bigwedge x. (Fun\ f\ ss, x) \notin S$

**and**  $u$ :  $u = Var\ x$

**shows**  $u \succeq t$

$\langle proof \rangle$

**lemma** *wpo-compat*:

$(s \succeq t \wedge t \succ u \implies s \succ u) \wedge$

$(s \succ t \wedge t \succeq u \implies s \succ u) \wedge$

$(s \succeq t \wedge t \succeq u \implies s \succeq u)$  (**is**  $?tran\ s\ t\ u$ )

$\langle proof \rangle$

**lemma** *subterm-wpo-s-arg*: **assumes**  $i$ :  $i \in set (\sigma (f, length\ ss))$

**shows**  $Fun\ f\ ss \succ ss ! i$   
 ⟨proof⟩

**lemma** *subterm-wpo-ns-arg*: **assumes**  $i: i \in set\ (\sigma\ (f, length\ ss))$   
**shows**  $Fun\ f\ ss \succeq ss ! i$   
 ⟨proof⟩

**lemma** *wpo-ns-pre-mono*: **fixes**  $f$  **and**  $bef\ aft :: ('f, 'v)term\ list$   
**defines**  $\sigma f \equiv \sigma\ (f, Suc\ (length\ bef + length\ aft))$   
**assumes**  $rel: (wpo-ns\ s\ t)$   
**shows**  $(\forall j \in set\ \sigma f. Fun\ f\ (bef\ @\ s\ \#\ aft) \succ (bef\ @\ t\ \#\ aft) ! j)$   
 $\wedge (Fun\ f\ (bef\ @\ s\ \#\ aft), (Fun\ f\ (bef\ @\ t\ \#\ aft))) \in NS$   
 $\wedge (\forall i < length\ \sigma f. ((map\ (!)\ (bef\ @\ s\ \#\ aft))\ \sigma f) ! i) \succeq ((map\ (!)\ (bef\ @\ t\ \#\ aft))\ \sigma f) ! i)$   
**(is -  $\wedge$  -  $\wedge$  ?three)**  
 ⟨proof⟩

**lemma** *wpo-ns-mono*:  
**assumes**  $rel: s \succeq t$   
**shows**  $Fun\ f\ (bef\ @\ s\ \#\ aft) \succeq Fun\ f\ (bef\ @\ t\ \#\ aft)$   
 ⟨proof⟩

**lemma** *wpo-stable*: **fixes**  $\delta :: ('f, 'v)subst$   
**shows**  $(s \succ t \longrightarrow s \cdot \delta \succ t \cdot \delta) \wedge (s \succeq t \longrightarrow s \cdot \delta \succeq t \cdot \delta)$   
**(is ?p s t)**  
 ⟨proof⟩

**lemma** *WPO-S-SN*:  $SN\ WPO-S$   
 ⟨proof⟩

**theorem** *WPO-SN-order-pair*:  $SN\ order-pair\ WPO-S\ WPO-NS$   
 ⟨proof⟩

**theorem** *WPO-S-subst*:  $(s, t) \in WPO-S \implies (s \cdot \sigma, t \cdot \sigma) \in WPO-S$  **for**  $\sigma$   
 ⟨proof⟩

**theorem** *WPO-NS-subst*:  $(s, t) \in WPO-NS \implies (s \cdot \sigma, t \cdot \sigma) \in WPO-NS$  **for**  $\sigma$   
 ⟨proof⟩

**theorem** *WPO-NS-ctxt*:  $(s, t) \in WPO-NS \implies (Fun\ f\ (bef\ @\ s\ \#\ aft), Fun\ f\ (bef\ @\ t\ \#\ aft)) \in WPO-NS$   
 ⟨proof⟩

**theorem** *WPO-S-subset-WPO-NS*:  $WPO-S \subseteq WPO-NS$   
 ⟨proof⟩

**context**  
**assumes**  $\sigma\ full: \bigwedge f\ k. set\ (\sigma\ (f, k)) = \{0 ..< k\}$

**begin**

**lemma** *subterm-wpo-s*:  $s \triangleright t \implies s \succ t$   
*<proof>*

**lemma** *subterm-wpo-ns*: **assumes** *supteq*:  $s \trianglerighteq t$  **shows**  $s \succeq t$   
*<proof>*

**lemma** *wpo-s-mono*: **assumes** *rels*:  $s \succ t$   
**shows**  $\text{Fun } f \text{ (bef @ } s \# \text{ aft)} \succ \text{Fun } f \text{ (bef @ } t \# \text{ aft)}$   
*<proof>*

**theorem** *WPO-S-ctxt*:  $(s, t) \in \text{WPO-S} \implies (\text{Fun } f \text{ (bef @ } s \# \text{ aft)}, \text{Fun } f \text{ (bef @ } t \# \text{ aft)}) \in \text{WPO-S}$   
*<proof>*

**theorem** *supt-subset-WPO-S*:  $\{\triangleright\} \subseteq \text{WPO-S}$   
*<proof>*

**theorem** *supteq-subset-WPO-NS*:  $\{\trianglerighteq\} \subseteq \text{WPO-NS}$   
*<proof>*

**end**

**end**

**end**

## 6 The Recursive Path Order as an instance of WPO

This theory defines the recursive path order (RPO) that given two terms provides two Booleans, whether the terms can be strictly or non-strictly oriented. It is proven that RPO is an instance of WPO, and hence, carries over all the nice properties of WPO immediately.

**theory** *RPO*

**imports**

*WPO*

**begin**

**context**

**fixes** *pr* ::  $'f \times \text{nat} \Rightarrow 'f \times \text{nat} \Rightarrow \text{bool} \times \text{bool}$

**and** *prl* ::  $'f \times \text{nat} \Rightarrow \text{bool}$

**and** *c* ::  $'f \times \text{nat} \Rightarrow \text{order-tag}$

**and** *n* :: *nat*

**begin**

**fun** *rpo* ::  $(f, v) \text{ term} \Rightarrow (f, v) \text{ term} \Rightarrow \text{bool} \times \text{bool}$

**where**  
 $rpo (Var x) (Var y) = (False, x = y) \mid$   
 $rpo (Var x) (Fun g ts) = (False, ts = [] \wedge prl (g,0)) \mid$   
 $rpo (Fun f ss) (Var y) = (let con = (\exists s \in set ss. snd (rpo s (Var y))) in$   
 $(con,con)) \mid$   
 $rpo (Fun f ss) (Fun g ts) = ($   
 $if (\exists s \in set ss. snd (rpo s (Fun g ts)))$   
 $then (True,True)$   
 $else (let (prs,prns) = pr (f,length ss) (g,length ts) in$   
 $if prns \wedge (\forall t \in set ts. fst (rpo (Fun f ss) t))$   
 $then if prs$   
 $then (True,True)$   
 $else if c (f,length ss) = Lex \wedge c (g,length ts) = Lex$   
 $then lex-ext rpo n ss ts$   
 $else if c (f,length ss) = Mul \wedge c (g,length ts) = Mul$   
 $then mul-ext rpo ss ts$   
 $else (length ss \neq 0 \wedge length ts = 0, length ts = 0)$   
 $else (False,False)))$   
**end**

**locale** *rpo-with-assms* = precedence prc prl  
**for**  $prc :: 'f \times nat \Rightarrow 'f \times nat \Rightarrow bool \times bool$   
**and**  $prl :: 'f \times nat \Rightarrow bool$   
**and**  $c :: 'f \times nat \Rightarrow order-tag$   
**and**  $n :: nat$   
**begin**

**sublocale** *wpo-with-assms*  $n \{ \}$  UNIV prc prl full-status c False  $\lambda -.$  False  
 $\langle proof \rangle$

**abbreviation**  $rpo-pr \equiv rpo prc prl c n$   
**abbreviation**  $rpo-s \equiv \lambda s t. fst (rpo-pr s t)$   
**abbreviation**  $rpo-ns \equiv \lambda s t. snd (rpo-pr s t)$

**lemma** *rpo-eq-wpo*:  $rpo-pr s t = wpo s t$   
 $\langle proof \rangle$

**abbreviation**  $RPO-S \equiv \{(s,t). rpo-s s t\}$   
**abbreviation**  $RPO-NS \equiv \{(s,t). rpo-ns s t\}$

**theorem** *RPO-SN-order-pair*: SN-order-pair  $RPO-S RPO-NS$   
 $\langle proof \rangle$

**theorem** *RPO-S-subst*:  $(s,t) \in RPO-S \implies (s \cdot \sigma, t \cdot \sigma) \in RPO-S$  **for**  $\sigma ::$   
 $(f, 'a)subst$   
 $\langle proof \rangle$

**theorem** *RPO-NS-subst*:  $(s,t) \in RPO-NS \implies (s \cdot \sigma, t \cdot \sigma) \in RPO-NS$  **for**  $\sigma ::$



*(f, 'a)subst*  
 ⟨proof⟩

**theorem** *RPO-NS-ctxt*:  $(s,t) \in RPO-NS \implies (Fun\ f\ (bef\ @\ s\ \#\ aft), Fun\ f\ (bef\ @\ t\ \#\ aft)) \in RPO-NS$   
 ⟨proof⟩

**theorem** *RPO-S-ctxt*:  $(s,t) \in RPO-S \implies (Fun\ f\ (bef\ @\ s\ \#\ aft), Fun\ f\ (bef\ @\ t\ \#\ aft)) \in RPO-S$   
 ⟨proof⟩

**theorem** *RPO-S-subset-RPO-NS*:  $RPO-S \subseteq RPO-NS$   
 ⟨proof⟩

**theorem** *supt-subset-RPO-S*:  $\{\triangleright\} \subseteq RPO-S$   
 ⟨proof⟩

**theorem** *supteq-subset-RPO-NS*:  $\{\triangleright\} \subseteq RPO-NS$   
 ⟨proof⟩

**end**  
**end**

## 7 The Lexicographic Path Order as an instance of WPO

We first directly define the strict- and non-strict lexicographic path orders (LPO) w.r.t. some precedence, and then show that it is an instance of WPO. For this instance we use the trivial reduction pair in WPO ( $\emptyset$ , UNIV) and the status is the full one, i.e., taking parameters  $[0, \dots, n-1]$  for each n-ary symbol.

**theory** *LPO*

**imports**  
*WPO*

**begin**

**context**

**fixes** *pr* ::  $(f \times nat \Rightarrow f \times nat \Rightarrow bool \times bool)$   
**and** *prl* ::  $f \times nat \Rightarrow bool$   
**and** *n* :: *nat*

**begin**

**fun** *lpo* ::  $(f, 'v)\ term \Rightarrow (f, 'v)\ term \Rightarrow bool \times bool$

**where**

*lpo* (*Var* *x*) (*Var* *y*) = (*False*,  $x = y$ ) |  
*lpo* (*Var* *x*) (*Fun* *g* *ts*) = (*False*,  $ts = [] \wedge prl\ (g, 0)$ ) |  
*lpo* (*Fun* *f* *ss*) (*Var* *y*) = (*let* *con* =  $(\exists\ s \in set\ ss.\ snd\ (lpo\ s\ (Var\ y)))$ ) *in*  
 (*con*, *con*) |

```

lpo (Fun f ss) (Fun g ts) = (
  if (∃ s ∈ set ss. snd (lpo s (Fun g ts)))
  then (True, True)
  else (let (prs, prns) = pr (f, length ss) (g, length ts) in
    if prns ∧ (∀ t ∈ set ts. fst (lpo (Fun f ss) t))
    then if prs
      then (True, True)
      else lex-ext lpo n ss ts
    else (False, False)))

```

**end**

```

locale lpo-with-assms = precedence prc prl
  for prc :: 'f × nat ⇒ 'f × nat ⇒ bool × bool
  and prl :: 'f × nat ⇒ bool
  and n :: nat
begin

```

```

sublocale wpo-with-assms n {} UNIV prc prl full-status λ -. Lex False λ -. False
  ⟨proof⟩

```

```

abbreviation lpo-pr ≡ lpo prc prl n
abbreviation lpo-s ≡ λ s t. fst (lpo-pr s t)
abbreviation lpo-ns ≡ λ s t. snd (lpo-pr s t)

```

```

lemma lpo-eq-wpo: lpo-pr s t = wpo s t
  ⟨proof⟩

```

```

abbreviation LPO-S ≡ {(s,t). lpo-s s t}
abbreviation LPO-NS ≡ {(s,t). lpo-ns s t}

```

```

theorem LPO-SN-order-pair: SN-order-pair LPO-S LPO-NS
  ⟨proof⟩

```

```

theorem LPO-S-subst: (s,t) ∈ LPO-S ⇒ (s · σ, t · σ) ∈ LPO-S for σ ::
  ('f, 'a)subst
  ⟨proof⟩

```

```

theorem LPO-NS-subst: (s,t) ∈ LPO-NS ⇒ (s · σ, t · σ) ∈ LPO-NS for σ ::
  ('f, 'a)subst
  ⟨proof⟩

```

```

theorem LPO-NS-ctxt: (s,t) ∈ LPO-NS ⇒ (Fun f (bef @ s # aft), Fun f (bef
  @ t # aft)) ∈ LPO-NS
  ⟨proof⟩

```

```

theorem LPO-S-ctxt: (s,t) ∈ LPO-S ⇒ (Fun f (bef @ s # aft), Fun f (bef @ t
  # aft)) ∈ LPO-S

```

*<proof>*

**theorem** *LPO-S-subset-LPO-NS: LPO-S  $\subseteq$  LPO-NS*  
*<proof>*

**theorem** *supt-subset-LPO-S:  $\{\triangleright\} \subseteq$  LPO-S*  
*<proof>*

**theorem** *supteq-subset-LPO-NS:  $\{\triangleright\} \subseteq$  LPO-NS*  
*<proof>*

**end**

**end**

## 8 The Knuth–Bendix Order as an instance of WPO

Making the Knuth–Bendix an instance of WPO is more complicated than in the case of RPO and LPO, because of syntactic and semantic differences. We face the two main challenges in two different theories and sub-sections.

### 8.1 Aligning least elements

In all of RPO, LPO and WPO there is the concept of a minimal term, e.g., a constant term  $c$  where  $c$  is least in precedence among *all function symbols*. By contrast, in KBO a constant  $c$  is minimal if it has minimal weight and has least precedence *among all constants of minimal weight*.

In this theory we prove that for any KBO one can modify the precedence in a way that least constants  $c$  also have least precedence among *all function symbols*, without changing the defined order. Hence, afterwards it will be simpler to relate such a KBO to WPO.

**theory** *KBO-Transformation*  
**imports** *WPO Knuth-Bendix-Order.KBO*  
**begin**

**context** *admissible-kbo*  
**begin**

**lemma** *weight-w0-unary:*  
**assumes**  $*$ : *weight*  $t = w0$   $t = Fun$   $f$   $ts$   $ts = t1$   $\#$   $ts'$   
**shows**  $ts' = []$   $w(f,1) = 0$   
*<proof>*

**definition** *lConsts* ::  $(f \times nat) set$  **where**  $lConsts = \{ (f,0) \mid f. \text{least } f \}$

**definition** *pr-strict'* **where**  $pr\text{-strict}' f g = (f \notin lConsts \wedge (pr\text{-strict } f g \vee g \in lConsts))$

**definition** *pr-weak'* **where**  $pr\text{-weak}' f g = ((f \notin lConsts \wedge pr\text{-weak} f g) \vee g \in lConsts)$

**lemma** *admissible-kbo'*: *admissible-kbo w w0 pr-strict' pr-weak' least scf*  
 ⟨*proof*⟩

**lemma** *least-pr-weak'*:  $least f \implies pr\text{-weak}' g (f,0)$  ⟨*proof*⟩

**lemma** *least-pr-weak'-trans*:  $least f \implies pr\text{-weak}' (f,0) g \implies least (fst g) \wedge snd g = 0$   
 ⟨*proof*⟩

**context**

**begin**

**interpretation** *kbo'*: *admissible-kbo w w0 pr-strict' pr-weak' least scf*  
 ⟨*proof*⟩

**lemma** *kbo'-eq-kbo*:  $kbo'.kbo s t = kbo s t$   
 ⟨*proof*⟩

**end**

**end**

**end**

## 8.2 A restricted equality between KBO and WPO

The remaining difficulty to make KBO an instance of WPO is the different treatment of lexicographic comparisons, which is unrestricted in KBO, but there is a length-restriction in WPO. Therefore we will only show that KBO is an instance of WPO if we compare terms with bounded arity.

This restriction does however not prohibit us from lifting properties of WPO to KBO. For instance, for several properties one can choose a large-enough bound restriction of WPO, since there are only finitely many arities occurring in a property.

**theory** *KBO-as-WPO*

**imports**

*WPO*

*KBO-Transformation*

**begin**

**definition** *bounded-arity* ::  $nat \Rightarrow (f \times nat) set \Rightarrow bool$  **where**  
*bounded-arity* b F =  $(\forall (f,n) \in F. n \leq b)$

**lemma** *finite-funas-term[simp,intro]*: *finite (funas-term t)*  
 ⟨*proof*⟩

**context** *weight-fun* **begin**

**definition** *weight-le* s t  $\equiv$

$(\text{vars-term-ms } (SCF\ s) \subseteq\# \text{ vars-term-ms } (SCF\ t) \wedge \text{weight } s \leq \text{weight } t)$

**definition**  $\text{weight-less } s\ t \equiv$

$(\text{vars-term-ms } (SCF\ s) \subseteq\# \text{ vars-term-ms } (SCF\ t) \wedge \text{weight } s < \text{weight } t)$

**lemma**  $\text{weight-le-less-iff}$ :  $\text{weight-le } s\ t \implies \text{weight-less } s\ t \iff \text{weight } s < \text{weight } t$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{weight-less-iff}$ :  $\text{weight-less } s\ t \implies \text{weight-le } s\ t \wedge \text{weight } s < \text{weight } t$   
 $\langle \text{proof} \rangle$

**abbreviation**  $\text{weight-NS} \equiv \{(t,s). \text{weight-le } s\ t\}$

**abbreviation**  $\text{weight-S} \equiv \{(t,s). \text{weight-less } s\ t\}$

**lemma**  $\text{weight-le-mono-one}$ :

**assumes**  $S$ :  $\text{weight-le } s\ t$

**shows**  $\text{weight-le } (\text{Fun } f\ (ss1\ @\ s\ \# \ ss2))\ (\text{Fun } f\ (ss1\ @\ t\ \# \ ss2))$  (**is**  $\text{weight-le } ?s\ ?t$ )  
 $\langle \text{proof} \rangle$

**lemma**  $\text{weight-le-ctxt}$ :  $\text{weight-le } s\ t \implies \text{weight-le } (C\langle s \rangle)\ (C\langle t \rangle)$   
 $\langle \text{proof} \rangle$

**lemma**  $SCF\text{-stable}$ :

**assumes**  $\text{vars-term-ms } (SCF\ s) \subseteq\# \text{ vars-term-ms } (SCF\ t)$

**shows**  $\text{vars-term-ms } (SCF\ (s \cdot \sigma)) \subseteq\# \text{ vars-term-ms } (SCF\ (t \cdot \sigma))$

$\langle \text{proof} \rangle$

**lemma**  $SN\text{-weight-S}$ :  $SN\ \text{weight-S}$

$\langle \text{proof} \rangle$

**lemma**  $\text{weight-less-imp-le}$ :  $\text{weight-less } s\ t \implies \text{weight-le } s\ t$   $\langle \text{proof} \rangle$

**lemma**  $\text{weight-le-Var-Var}$ :  $\text{weight-le } (Var\ x)\ (Var\ y) \iff x = y$   
 $\langle \text{proof} \rangle$

**end**

**context**  $kbo$  **begin**

**lemma**  $kbo\text{-altdef}$ :

$kbo\ s\ t =$  (*if*  $\text{weight-le } t\ s$

*then* *if*  $\text{weight-less } t\ s$

*then*  $(True, True)$

*else* (*case*  $s$  *of*

$Var\ y \Rightarrow (False, (\text{case } t \text{ of } Var\ x \Rightarrow x = y \mid Fun\ g\ ts \Rightarrow ts = [] \wedge \text{least } g))$ )

$| Fun\ f\ ss \Rightarrow (\text{case } t \text{ of}$

$Var\ x \Rightarrow (True, True)$ )

```

| Fun g ts  $\Rightarrow$  if pr-strict (f, length ss) (g, length ts)
  then (True, True)
  else if pr-weak (f, length ss) (g, length ts)
  then lex-ext-unbounded kbo ss ts
  else (False, False)))
else (False, False))
⟨proof⟩

end

context admissible-kbo begin

lemma weight-le-stable:
  assumes weight-le s t
  shows weight-le (s ·  $\sigma$ ) (t ·  $\sigma$ )
  ⟨proof⟩

lemma weight-less-stable:
  assumes weight-less s t
  shows weight-less (s ·  $\sigma$ ) (t ·  $\sigma$ )
  ⟨proof⟩

lemma simple-arg-pos-weight: simple-arg-pos weight-NS (f,n) i
  ⟨proof⟩

lemma weight-lemmas:
  shows refl weight-NS and trans weight-NS and trans weight-S
  and weight-NS O weight-S  $\subseteq$  weight-S and weight-S O weight-NS  $\subseteq$  weight-S
  ⟨proof⟩

interpretation kbo': admissible-kbo w w0 pr-strict' pr-weak' least scf
  ⟨proof⟩

context
  assumes least-global:  $\bigwedge$  f g. least f  $\Longrightarrow$  pr-weak g (f,0)
  and least-trans:  $\bigwedge$  f g. least f  $\Longrightarrow$  pr-weak (f,0) g  $\Longrightarrow$  least (fst g)  $\wedge$  snd g = 0
  fixes n :: nat
begin

lemma kbo-instance-of-wpo-with-assms: wpo-with-assms
  weight-S weight-NS ( $\lambda$ f g. (pr-strict f g, pr-weak f g))
  ( $\lambda$ (f, n). n = 0  $\wedge$  least f) full-status False ( $\lambda$ f. False)
  ⟨proof⟩

interpretation wpo: wpo-with-assms
  where S = weight-S and NS = weight-NS
  and prc =  $\lambda$ f g. (pr-strict f g, pr-weak f g) and prl =  $\lambda$ (f,n). n = 0  $\wedge$  least f
  and c =  $\lambda$ -. Lex
  and ssimple = False and large =  $\lambda$ f. False and  $\sigma\sigma$  = full-status

```

**and**  $n = n$   
 $\langle proof \rangle$

**lemma** *kbo-as-wpo-with-assms*: **assumes** *bounded-arity*  $n$  (*funas-term*  $t$ )  
**shows**  $kbo\ s\ t = wpo.wpo\ s\ t$   
 $\langle proof \rangle$   
**end**

This is the main theorem. It tells us that KBO can be seen as an instance of WPO, under mild preconditions: the parameter  $n$  for the lexicographic extension has to be chosen high enough to cover the arities of all terms that should be compared.

**lemma defines**  $prec \equiv ((\lambda f\ g.\ (pr\text{-}strict'\ f\ g,\ pr\text{-}weak'\ f\ g)))$   
**and**  $prl \equiv (\lambda(f,\ n).\ n = 0 \wedge least\ f)$   
**shows**  
*kbo-encoding-is-valid-wpo*: *wpo-with-assms weight-S weight-NS prec prl full-status*  
 $False\ (\lambda f.\ False)$   
**and**  
*kbo-as-wpo*: *bounded-arity*  $n$  (*funas-term*  $t$ )  $\implies kbo\ s\ t = wpo.wpo\ n\ weight-S$   
*weight-NS prec prl full-status*  $(\lambda.\ Lex)\ False\ (\lambda f.\ False)\ s\ t$   
 $\langle proof \rangle$

As a proof-of-concept we show that now properties of WPO can be used to prove these properties for KBO. Here, as example we consider closure under substitutions and strong normalization, but the following idea can be applied for several more properties: if the property involves only terms where the arities are bounded, then just choose the parameter  $n$  large enough. This even works for strong normalization, since in an infinite chain of KBO-decreases  $t_1 > t_2 > t_3 > \dots$  all terms have a weight of at most the weight of  $t_1$ , and this weight is also a bound on the arities.

**lemma** *KBO-stable-via-WPO*:  $S\ s\ t \implies S\ (s \cdot (\sigma :: ('f, 'a)\ subst))\ (t \cdot \sigma)$   
 $\langle proof \rangle$

**lemma** *weight-is-arity-bound*:  $weight\ t \leq b \implies bounded-arity\ b$  (*funas-term*  $t$ )  
 $\langle proof \rangle$

**lemma** *KBO-SN-via-WPO*:  $SN\ \{(s, t).\ S\ s\ t\}$   
 $\langle proof \rangle$

**end**

**end**

## 9 Executability of the orders

**theory** *Executable-Orders*  
**imports**

```
WPO
RPO
LPO
Multiset-Extension2-Impl
begin
```

If one loads the implementation of multiset orders (in particular for *mul-ext*), then all orders defined in this AFP-entry (WPO, RPO, LPO, multiset extension of order pairs) are executable.

```
export-code
lpo
rpo
wpo.wpo
mul-ext
mult2-impl
in Haskell
```

```
end
```

## References

- [1] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
- [2] N. Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1/2):69–116, 1987.
- [3] S. Kamin and J. J. Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, IL, USA, 1980.
- [4] D. E. Knuth and P. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. 1970.
- [5] C. Sternagel and R. Thiemann. Formalizing Knuth–Bendix orders and Knuth–Bendix completion. In *Rewriting Techniques and Applications, RTA ’13*, volume 2 of *Leibniz International Proceedings in Informatics*, pages 287–302, 2013.
- [6] C. Sternagel and R. Thiemann. A formalization of Knuth–Bendix orders. *Archive of Formal Proofs*, May 2020. [https://isa-afp.org/entries/Knuth\\_Bendix\\_Order.html](https://isa-afp.org/entries/Knuth_Bendix_Order.html), Formal proof development.
- [7] R. Thiemann, G. Allais, and J. Nagele. On the formalization of termination techniques based on multiset orderings. In A. Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA ’12)*, *RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPICs*, pages 339–354. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.



- [8] R. Thiemann, J. Schöpf, C. Sternagel, and A. Yamada. Certifying the weighted path order (invited talk). In Z. M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPICs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [9] A. Yamada, K. Kusakari, and T. Sakabe. Unifying the Knuth-Bendix, recursive path and polynomial orders. In R. Peña and T. Schrijvers, editors, *15th International Symposium on Principles and Practice of Declarative Programming, PPDP '13, Madrid, Spain, September 16-18, 2013*, pages 181–192. ACM, 2013.
- [10] A. Yamada, K. Kusakari, and T. Sakabe. A unified ordering for termination proving. *Sci. Comput. Program.*, 111:110–134, 2015.