A Formalization of Weighted Path Orders and Recursive Path Orders*

Christian Sternagel René Thiemann Akihisa Yamada

March 17, 2025

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

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

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 chosing 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* $= \{ (\sigma :: 'f \times nat \Rightarrow nat\ list).\ (\forall\ f\ k.\ set\ (\sigma\ (f,\ k)) \subseteq \{0\ ..< k\})\}$
  morphisms *status Abs-status*
  ⟨*proof*⟩

setup-lifting *type-definition-status*

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

lemma *status-aux*[*termination-simp*]: $i \in set\ (status\ \sigma\ (f,\ length\ ss)) \Longrightarrow ss\ !\ i \in set\ ss$
  ⟨*proof*⟩

lemma *status-termination-simps*[*termination-simp*]:
  assumes *i1*: $i < length\ (status\ \sigma\ (f,\ length\ xs))$
  shows $size\ (xs\ !\ (status\ \sigma\ (f,\ length\ xs)\ !\ i)) < Suc\ (size\text{-}list\ size\ xs)$ (**is** $?a < ?c$)
⟨*proof*⟩

lemma *status-ne*:

status σ $(f,\ n) \neq [] \Longrightarrow \exists\, i < n.\ i \in$ *set* (*status* σ $(f,\ n)$)
⟨*proof*⟩

lemma *set-status-nth*:
length xs $= n \Longrightarrow i \in$ *set* (*status* σ $(f,\ n)$) $\Longrightarrow i <$ *length xs* $\wedge$ *xs* ! $i \in$ *set xs*
⟨*proof*⟩

lift-definition *full-status* :: *'f status* **is** λ $(f,\ n)$. $[0\ ..{<}\ n]$ ⟨*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*: $[\![\bigwedge$ *ts*. *length ts* $= n \Longrightarrow i < n \Longrightarrow$ (*Fun f ts*, *ts* ! *i*) $\in$ *rel*$]\!] \Longrightarrow$ *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 *irrefl-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-stri-imp-nstri*: *fst* (*prc f g*) $\Longrightarrow$ *snd* (*prc f g*)
and *prl*: *prl g* $\Longrightarrow$ *snd* (*prc f g*) = *True*
and *prl3*: *prl f* $\Longrightarrow$ *snd* (*prc f g*) $\Longrightarrow$ *prl g*
and *prc-compat*: *prc f g* = (*s1*, *ns1*) $\Longrightarrow$ *prc g h* = (*s2*, *ns2*) $\Longrightarrow$ *prc f h* = (*s*, *ns*) $\Longrightarrow$
(*ns1* $\wedge$ *ns2* $\longrightarrow$ *ns*) $\wedge$ (*ns1* $\wedge$ *s2* $\longrightarrow$ *s*) $\wedge$ (*s1* $\wedge$ *ns2* $\longrightarrow$ *s*)

begin
lemma *prl2*:
  assumes *g*: *prl g* **shows** *fst* (*prc g f*) = *False*
⟨*proof*⟩

abbreviation *pr* ≡ (*prc*, *prl*)

end

locale *precedence* = *irrefl-precedence* +
  constrains *prc* :: *'f* ⇒ *'f* ⇒ *bool* × *bool*
    and *prl* :: *'f* ⇒ *bool*
  assumes *prc-SN*: *SN* {(*f*, *g*). *fst* (*prc f g*)}

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* ⇒ *'a rel* ⇒ *bool* **where**
  compatible-l R1 R2 ≡ *R1 O R2* ⊆ *R2*

definition *compatible-r* :: *'a rel* ⇒ *'a rel* ⇒ *bool* **where**
  compatible-r R1 R2 ≡ *R2 O R1* ⊆ *R2*

Local reflexivity

definition *locally-refl* :: *'a rel* ⇒ *'a multiset* ⇒ *bool* **where**
  locally-refl R A ≡ (∀ *a*. *a* ∈# *A* ⟶ (*a*,*a*) ∈ *R*)

definition *locally-irrefl* :: *'a rel* ⇒ *'a multiset* ⇒ *bool* **where**
  locally-irrefl R A ≡ (∀ *t*. *t* ∈# *A* ⟶ (*t*,*t*) ∉ *R*)

Local symmetry

definition *locally-sym* :: *'a rel* ⇒ *'a multiset* ⇒ *bool* **where**
  locally-sym R A ≡ (∀ *t u*. *t* ∈# *A* ⟶ *u* ∈# *A* ⟶
            (*t*,*u*) ∈ *R* ⟶ (*u*,*t*) ∈ *R*)

definition *locally-antisym* :: *'a rel* ⇒ *'a multiset* ⇒ *bool* **where**
  locally-antisym R A ≡ (∀ *t u*. *t* ∈# *A* ⟶ *u* ∈# *A* ⟶
            (*t*,*u*) ∈ *R* ⟶ (*u*,*t*) ∈ *R* ⟶ *t* = *u*)

Local transitivity

definition *locally-trans* :: *'a rel* ⇒ *'a multiset* ⇒ *'a multiset* ⇒ *'a multiset* ⇒ *bool*
where

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

Local inclusion

definition $locally\text{-}included :: 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$
where
$locally\text{-}included\ R1\ R2\ A\ B \equiv (\forall\, t\ u.\ t \in\#\ A \longrightarrow u \in\#\ B \longrightarrow$
$(t,u) \in R1 \longrightarrow (t,u) \in R2)$

Local transitivity compatibility

definition $locally\text{-}compatible\text{-}l :: 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ **where**
$locally\text{-}compatible\text{-}l\ R1\ R2\ A\ B\ C \equiv (\forall\, t\ u\ v.\ t \in\#\ A \longrightarrow u \in\#\ B \longrightarrow v \in\#\ C \longrightarrow$
$(t,u) \in R1 \longrightarrow (u,v) \in R2 \longrightarrow (t,v) \in R2)$

definition $locally\text{-}compatible\text{-}r :: 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$ **where**
$locally\text{-}compatible\text{-}r\ R1\ R2\ A\ B\ C \equiv (\forall\, t\ u\ v.\ t \in\#\ A \longrightarrow u \in\#\ B \longrightarrow v \in\#\ C \longrightarrow$
$(t,u) \in R2 \longrightarrow (u,v) \in R1 \longrightarrow (t,v) \in R2)$

included + compatible $\longrightarrow$ transitive

lemma *in-cl-tr*:
assumes $R1 \subseteq R2$
and $compatible\text{-}l\ R2\ R1$
shows $trans\ R1$
$\langle proof \rangle$

lemma *in-cr-tr*:
assumes $R1 \subseteq R2$
and $compatible\text{-}r\ R2\ R1$
shows $trans\ R1$
$\langle proof \rangle$

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

lemma *r-lr*:
assumes $refl\ R$
shows $locally\text{-}refl\ R\ A$
$\langle proof \rangle$

lemma *tr-ltr*:
assumes $trans\ R$
shows $locally\text{-}trans\ R\ A\ B\ C$
$\langle proof \rangle$

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** ‹↾› *80*)

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

lemma *mem-restrictD*[*dest*]: **assumes** $(x,y) \in R \upharpoonright X$ **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*: $[\![\bigwedge a\ b.\ (a,b) \in S \Longrightarrow (f\ a,f\ b) \in S; \bigwedge a\ b.\ (a,b) \in NS \Longrightarrow (f\ a,f\ b) \in NS]\!] \Longrightarrow (as,bs) \in$ *s-list S NS* $\Longrightarrow$ (*map f as*, *map f bs*) $\in$ *s-list S NS*
    and *ns-map*: $[\![\bigwedge a\ b.\ (a,b) \in S \Longrightarrow (f\ a,f\ b) \in S; \bigwedge a\ b.\ (a,b) \in NS \Longrightarrow (f\ a,f\ b) \in NS]\!] \Longrightarrow (as,bs) \in$ *ns-list S NS* $\Longrightarrow$ (*map f as*, *map f bs*) $\in$ *ns-list S NS*
    and *all-ns-imp-ns*: *length as* = *length bs* $\Longrightarrow [\![\bigwedge i.\ i <$ *length bs* $\Longrightarrow (as\ !\ i, bs\ !\ i) \in NS]\!] \Longrightarrow (as,bs) \in$ *ns-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* :: $'a$ *list-ext-impl*
  assumes *list-ext-s*: $\bigwedge s\ ns.$ *s-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-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-list ns s* $\Longrightarrow (as,bs) \in$ *s-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-list ns s* $\Longrightarrow (as,bs) \in$ *ns-list* $ns'\ s'$

end

3 Multiset extension of an order pair

Given a well-founded order $\prec$ and a compatible non-strict order $\precsim$, 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 $(X + Z, Y + Z) \in$ *mult s* $\longleftrightarrow (X, Y) \in$ *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 $(Z + X, Z + Y) \in$ *mult s* $\longleftrightarrow (X, Y) \in$ *mult s*
⟨*proof*⟩

lemma *mult-cancelL*:
assumes *trans s irrefl s* **shows** $(Z + X, Z + Y) \in$ *mult s* $\longleftrightarrow (X, Y) \in$ *mult s*
⟨*proof*⟩

lemma *wf-trancl-conv*:
shows *wf* $(r^+) \longleftrightarrow$ *wf r*
⟨*proof*⟩

3.1 Pointwise multiset order

inductive-set *multpw* :: *'a rel* ⇒ *'a multiset rel* **for** *ns* :: *'a rel* **where**
empty: $(\{\#\}, \{\#\}) \in$ *multpw ns*
| *add*: $(x, y) \in$ *ns* $\Longrightarrow (X, Y) \in$ *multpw ns* $\Longrightarrow$ (*add-mset x X*, *add-mset y Y*) $\in$ *multpw ns*

lemma *multpw-emptyL* [*simp*]:
$(\{\#\}, X) \in$ *multpw ns* $\longleftrightarrow X = \{\#\}$
⟨*proof*⟩

lemma *multpw-emptyR* [*simp*]:
$(X, \{\#\}) \in$ *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** $multpw\ ns \subseteq multpw\ ns'$
⟨*proof*⟩

lemma *multpw-converse*:
$multpw\ (ns^{-1}) = (multpw\ ns)^{-1}$
⟨*proof*⟩

lemma *multpw-local*:
$(X,\ Y) \in multpw\ ns \Longrightarrow (X,\ Y) \in multpw\ (ns \cap set\text{-}mset\ X \times set\text{-}mset\ Y)$
⟨*proof*⟩

lemma *multpw-split1R*:
assumes $(add\text{-}mset\ x\ X,\ Y) \in multpw\ ns$
obtains $z\ Z$ **where** $Y = add\text{-}mset\ z\ Z$ **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 $(X,\ add\text{-}mset\ y\ Y) \in multpw\ ns$
obtains $z\ Z$ **where** $X = add\text{-}mset\ z\ Z$ **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 $locally\text{-}trans\ ns\ S\ T\ U$
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** $(X1 + X2,\ Y1 + Y2) \in multpw\ ns$

$\langle proof \rangle$

lemma *multpw-single*:
$(x, y) \in ns \Longrightarrow (\{\#x\#\}, \{\#y\#\}) \in multpw\ ns$
$\langle proof \rangle$

lemma *multpw-mult1-commute*:
assumes *compat*: $s\ O\ ns \subseteq s$ **and** *reflns*: *refl ns*
shows $mult1\ s\ O\ multpw\ ns \subseteq multpw\ ns\ O\ mult1\ s$
$\langle proof \rangle$

lemma *multpw-mult-commute*:
assumes $s\ O\ ns \subseteq s$ *refl ns* **shows** $mult\ s\ O\ multpw\ ns \subseteq multpw\ ns\ O\ mult\ s$
$\langle proof \rangle$

lemma *wf-mult-rel-multpw*:
assumes *wf s* $s\ O\ ns \subseteq s$ *refl ns* **shows** $wf\ ((multpw\ ns)^*\ O\ mult\ s\ O\ (multpw\ ns)^*)$
$\langle proof \rangle$

lemma *multpw-cancel1*:
assumes *trans ns* $(y, x) \in ns$
shows $(add\text{-}mset\ x\ X, add\text{-}mset\ y\ Y) \in multpw\ ns \Longrightarrow (X, Y) \in multpw\ ns$ (**is** $?L \Longrightarrow ?R$)
$\langle proof \rangle$

lemma *multpw-cancel*:
assumes *refl ns trans ns*
shows $(X + Z, Y + Z) \in multpw\ ns \longleftrightarrow (X, Y) \in multpw\ ns$ (**is** $?L \longleftrightarrow ?R$)
$\langle proof \rangle$

lemma *multpw-cancelL*:
assumes *refl ns trans ns* **shows** $(Z + X, Z + Y) \in multpw\ ns \longleftrightarrow (X, Y) \in multpw\ ns$
$\langle proof \rangle$

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

definition $mult2\text{-}s\ ns\ s \equiv multpw\ ns\ O\ mult\ s$
definition $mult2\text{-}ns\ ns\ s \equiv multpw\ ns\ O\ (mult\ s)^=$

lemma *mult2-ns-conv*:
shows $mult2\text{-}ns\ ns\ s = mult2\text{-}s\ ns\ s \cup multpw\ ns$
$\langle proof \rangle$

lemma *mono-mult2-s*:
assumes $ns \subseteq ns'$ $s \subseteq s'$ **shows** $mult2\text{-}s\ ns\ s \subseteq mult2\text{-}s\ ns'\ s'$
$\langle proof \rangle$

lemma *mono-mult2-ns*:
  assumes $ns \subseteq ns'$ $s \subseteq s'$ **shows** $mult2\text{-}ns\ ns\ s \subseteq mult2\text{-}ns\ ns'\ s'$
  ⟨*proof*⟩

lemma *wf-mult2-s*:
  assumes $wf\ s$ $s\ O\ ns \subseteq s$ $refl\ ns$
  shows $wf\ (mult2\text{-}s\ ns\ s)$
  ⟨*proof*⟩

lemma *refl-mult2-ns*:
  assumes $refl\ ns$ **shows** $refl\ (mult2\text{-}ns\ ns\ s)$
  ⟨*proof*⟩

lemma *trans-mult2-s*:
  assumes $s\ O\ ns \subseteq s$ $refl\ ns$ $trans\ ns$
  shows $trans\ (mult2\text{-}s\ ns\ s)$
  ⟨*proof*⟩

lemma *trans-mult2-ns*:
  assumes $s\ O\ ns \subseteq s$ $refl\ ns$ $trans\ ns$
  shows $trans\ (mult2\text{-}ns\ ns\ s)$
  ⟨*proof*⟩

lemma *compat-mult2*:
  assumes $s\ O\ ns \subseteq s$ $refl\ ns$ $trans\ ns$
  shows $mult2\text{-}ns\ ns\ s\ O\ mult2\text{-}s\ ns\ s \subseteq mult2\text{-}s\ ns\ s$ $mult2\text{-}s\ ns\ s\ O\ mult2\text{-}ns\ ns\ s \subseteq mult2\text{-}s\ ns\ s$
  ⟨*proof*⟩

Trivial inclusions

lemma *mult-implies-mult2-s*:
  assumes $refl\ ns$ $(X,\ Y) \in mult\ s$
  shows $(X,\ Y) \in mult2\text{-}s\ ns\ s$
  ⟨*proof*⟩

lemma *mult-implies-mult2-ns*:
  assumes $refl\ ns$ $(X,\ Y) \in (mult\ s)^=$
  shows $(X,\ Y) \in mult2\text{-}ns\ ns\ s$
  ⟨*proof*⟩

lemma *multpw-implies-mult2-ns*:
  assumes $(X,\ Y) \in multpw\ ns$
  shows $(X,\ Y) \in mult2\text{-}ns\ ns\ s$
  ⟨*proof*⟩

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** $?L \longleftrightarrow ?R$)
⟨*proof*⟩

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* $\wedge\ (X2, Y2) \in (\textit{mult } s)^{=})$ (**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* $\Longrightarrow (X,\ Y) \in$ *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* $\Longrightarrow (X,\ Y) \in$ *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* $\Longrightarrow (X,\ Y) \in$ *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 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* ⊆ *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* ⊆ *s refl ns trans s*
shows *mult2-alt-ns ns s* = *mult2-ns ns s*
⟨*proof*⟩

lemma *mult2-alt-local*:
assumes (X, Y) ∈ *mult2-alt b ns s*
shows (X, Y) ∈ *mult2-alt b* (*ns* ∩ *set-mset X* × *set-mset Y*) (*s* ∩ *set-mset X* × *set-mset Y*)
⟨*proof*⟩

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

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

lemma *wf-below-UNIV*[*simp*]:
shows *wf-below r UNIV* ⟷ *wf r*
⟨*proof*⟩

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

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

lemma *wf-below-pointwise*:
wf-below r A ⟷ (∀ *a*. *a* ∈ *A* ⟶ *wf-below r* {*a*}) (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

lemma *SN-on-Image-rtrancl-conv*:
SN-on r A ⟷ *SN-on r* (r^* `` *A*) (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

lemma *SN-on-iff-wf-below*:
SN-on r A ⟷ *wf-below* (r^{-1}) *A*
⟨*proof*⟩

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

lemma *wf-below-trancl*:
shows $wf\text{-}below\ (r^+)\ A \longleftrightarrow wf\text{-}below\ r\ A$
$\langle proof \rangle$

lemma *wf-below-mult-local*:
assumes $wf\text{-}below\ r\ (set\text{-}mset\ X)$ **shows** $wf\text{-}below\ (mult\ r)\ \{X\}$

$\langle proof \rangle$

lemma *qc-wf-below*:
assumes $s\ O\ ns \subseteq (s \cup ns)^*\ O\ s$ $wf\text{-}below\ s\ A$
shows $wf\text{-}below\ (ns^*\ O\ s\ O\ ns^*)\ A$
$\langle proof \rangle$

lemma *wf-below-mult2-s-local*:
assumes $wf\text{-}below\ s\ (set\text{-}mset\ X)$ $s\ O\ ns \subseteq s$ $refl\ ns$ $trans\ ns$
shows $wf\text{-}below\ (mult2\text{-}s\ ns\ s)\ \{X\}$
$\langle proof \rangle$

3.7 Trivial cases

lemma *mult2-alt-emptyL*:
$(\{\#\},\ Y) \in mult2\text{-}alt\ b\ ns\ s \longleftrightarrow b \lor Y \neq \{\#\}$
$\langle proof \rangle$

lemma *mult2-alt-emptyR*:
$(X,\ \{\#\}) \in mult2\text{-}alt\ b\ ns\ s \longleftrightarrow b \land X = \{\#\}$
$\langle proof \rangle$

lemma *mult2-alt-s-single*:
$(a,\ b) \in s \Longrightarrow (\{\#a\#\},\ \{\#b\#\}) \in mult2\text{-}alt\text{-}s\ ns\ s$
$\langle proof \rangle$

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

lemma *mult2-alt-ns-conv*:
$mult2\text{-}alt\text{-}ns\ ns\ s = mult2\text{-}alt\text{-}s\ ns\ s \cup multpw\ ns$ (**is** $?l = ?r$)
$\langle proof \rangle$

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

shows $(X,\ Y) \in$ *mult2-alt-ns ns s*
⟨*proof*⟩

lemma *mult2-alt-add*:
assumes $(X1,\ Y1) \in$ *mult2-alt b1 ns s* **and** $(X2,\ Y2) \in$ *mult2-alt b2 ns s*
shows $(X1 + X2,\ Y1 + Y2) \in$ *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 $\bigwedge x\ y.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow (x,\ y) \in ns \Longrightarrow (f\ x,\ g\ y) \in ns'$
and $(X,\ Y) \in$ *multpw ns*
shows (*image-mset f X*, *image-mset g Y*) $\in$ *multpw ns'*
⟨*proof*⟩

lemma *mult2-alt-map*:
assumes $\bigwedge x\ y.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow (x,\ y) \in ns \Longrightarrow (f\ x,\ g\ y) \in ns'$
and $\bigwedge x\ y.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow (x,\ y) \in s \Longrightarrow (f\ x,\ g\ y) \in s'$
and $(X,\ Y) \in$ *mult2-alt b ns s*
shows (*image-mset f X*, *image-mset g Y*) $\in$ *mult2-alt b ns' s'*
⟨*proof*⟩

Local transitivity of *mult2-alt*

lemma *trans-mult2-alt-local*:
assumes *ss*: $\bigwedge x\ y\ z.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow z \in\#\ Z \Longrightarrow (x,\ y) \in s \Longrightarrow (y,\ z) \in s \Longrightarrow (x,\ z) \in s$
and *ns*: $\bigwedge x\ y\ z.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow z \in\#\ Z \Longrightarrow (x,\ y) \in ns \Longrightarrow (y,\ z) \in s \Longrightarrow (x,\ z) \in s$
and *sn*: $\bigwedge x\ y\ z.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow z \in\#\ Z \Longrightarrow (x,\ y) \in s \Longrightarrow (y,\ z) \in ns \Longrightarrow (x,\ z) \in s$
and *nn*: $\bigwedge x\ y\ z.\ x \in\#\ X \Longrightarrow y \in\#\ Y \Longrightarrow z \in\#\ Z \Longrightarrow (x,\ y) \in ns \Longrightarrow (y,\ z) \in ns \Longrightarrow (x,\ z) \in ns$
and *xyz*: $(X,\ Y) \in$ *mult2-alt b1 ns s* $(Y,\ Z) \in$ *mult2-alt b2 ns s*
shows $(X,\ Z) \in$ *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 $X \subseteq\# \ Y \ (Y, Z) \in$ *mult2-alt b ns s* $b \Longrightarrow b'$
shows $(X, Z) \in$ *mult2-alt* b' *ns s*
⟨*proof*⟩

Case distinction for recursion on left argument

lemma *mem-multiset-diff*: $x \in\# \ A \Longrightarrow x \neq y \Longrightarrow x \in\# \ (A - \{\#y\#\})$
⟨*proof*⟩

lemma *mult2-alt-addL*: (*add-mset x X*, Y) $\in$ *mult2-alt b ns s* $\longleftrightarrow$
($\exists y.\ y \in\# \ Y \wedge (x, y) \in s \wedge (\{\# \ x \in\# \ X.\ (x, y) \notin s \ \#\}, Y - \{\#y\#\}) \in$ *mult2-alt-ns ns s*) $\vee$
($\exists y.\ y \in\# \ Y \wedge (x, y) \in ns \wedge (x, y) \notin s \wedge (X, Y - \{\#y\#\}) \in$ *mult2-alt b 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* :: $'a \Rightarrow 'a \Rightarrow$ *bool* $\Rightarrow$ *bool*
begin

fun *mult2-impl0* :: $'a$ *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *bool* $\Rightarrow$ *bool*
and *mult2-ex-dom0* :: $'a \Rightarrow 'a$ *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *bool* $\Rightarrow$ *bool*
where
mult2-impl0 [] [] *b* $\longleftrightarrow$ *b*
| *mult2-impl0 xs* [] *b* $\longleftrightarrow$ *False*
| *mult2-impl0* [] *ys b* $\longleftrightarrow$ *True*
| *mult2-impl0* (*x* # *xs*) *ys b* $\longleftrightarrow$ *mult2-ex-dom0 x xs ys* [] *b*

| *mult2-ex-dom0 x xs* [] *ys′ b* $\longleftrightarrow$ *False*
| *mult2-ex-dom0 x xs* (*y* # *ys*) *ys′ b* $\longleftrightarrow$
nss x y False $\wedge$ *mult2-impl0* (*filter* ($\lambda x.\ \neg$ *nss x y False*) *xs*) (*ys* @ *ys′*) *True* $\vee$
nss x y True $\wedge$ $\neg$ *nss x y False* $\wedge$ *mult2-impl0 xs* (*ys* @ *ys′*) *b* $\vee$
mult2-ex-dom0 x xs ys (*y* # *ys′*) *b*

end

lemma *mult2-impl0-sound*:
  fixes *nss*
  defines $ns \equiv \{(x,\ y).\ nss\ x\ y\ True\}$ **and** $s \equiv \{(x,\ y).\ nss\ x\ y\ False\}$
  shows $mult2\text{-}impl0\ nss\ xs\ ys\ b \longleftrightarrow (mset\ xs,\ mset\ ys) \in mult2\text{-}alt\ b\ ns\ s$
    $mult2\text{-}ex\text{-}dom0\ nss\ x\ xs\ ys\ ys'\ b \longleftrightarrow$
      $(\exists\, y.\ y \in\#\ mset\ ys \wedge (x,\ y) \in s \wedge (mset\ (filter\ (\lambda x.\ (x,\ y) \notin s)\ xs),\ mset\ (ys\ @\ ys') - \{\#y\#\}) \in mult2\text{-}alt\ True\ ns\ s) \vee$
      $(\exists\, y.\ y \in\#\ mset\ ys \wedge (x,\ y) \in ns \wedge (x,\ y) \notin s \wedge (mset\ xs,\ mset\ (ys\ @\ ys') - \{\#y\#\}) \in mult2\text{-}alt\ b\ ns\ s)$
$\langle proof \rangle$

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 $mult2\text{-}impl :: 'a\ list \Rightarrow 'a\ list \Rightarrow bool \times bool$
  and $mult2\text{-}ex\text{-}dom :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool \times bool$
  where
    $mult2\text{-}impl\ []\ \ \ \ \ [] = (False,\ True)$
  $|\ mult2\text{-}impl\ xs\ \ \ \ \ [] = (False,\ False)$
  $|\ mult2\text{-}impl\ []\ \ \ \ \ ys = (True,\ True)$
  $|\ mult2\text{-}impl\ (x\ \#\ xs)\ ys = mult2\text{-}ex\text{-}dom\ x\ xs\ ys\ []$

$|\ mult2\text{-}ex\text{-}dom\ x\ xs\ []\ \ \ \ \ ys' = (False,\ False)$
$|\ mult2\text{-}ex\text{-}dom\ x\ xs\ (y\ \#\ ys)\ ys' =$
  $(case\ sns\ x\ y\ of$
    $(True,\ \text{-}) \Rightarrow if\ snd\ (mult2\text{-}impl\ (filter\ (\lambda x.\ \neg\ fst\ (sns\ x\ y))\ xs)\ (ys\ @\ ys'))\ then\ (True,\ True)$
          $else\ mult2\text{-}ex\text{-}dom\ x\ xs\ ys\ (y\ \#\ ys')$
  $|\ (False,\ True) \Rightarrow or2\ (mult2\text{-}impl\ xs\ (ys\ @\ ys'))\ (mult2\text{-}ex\text{-}dom\ x\ xs\ ys\ (y\ \#\ ys'))$
  $|\ \text{-} \Rightarrow mult2\text{-}ex\text{-}dom\ x\ xs\ ys\ (y\ \#\ ys'))$
end

lemma *mult2-impl-sound0*:
  defines $pair \equiv \lambda f.\ (f\ False,\ f\ True)$ **and** $fun \equiv \lambda p\ b.\ if\ b\ then\ snd\ p\ else\ fst\ p$
  shows $mult2\text{-}impl\ sns\ xs\ ys = pair\ (mult2\text{-}impl0\ (\lambda x\ y.\ fun\ (sns\ x\ y))\ xs\ ys)$ (**is** *?P*)
    $mult2\text{-}ex\text{-}dom\ sns\ x\ xs\ ys\ ys' = pair\ (mult2\text{-}ex\text{-}dom0\ (\lambda x\ y.\ fun\ (sns\ x\ y))\ x\ xs\ ys\ ys')$ (**is** *?Q*)
$\langle proof \rangle$

lemmas *mult2-impl-sound* = *mult2-impl-sound0(1)*[*unfolded mult2-impl0-sound 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 $length\ xs = length\ ys\ X = mset\ xs\ Y = mset\ ys$
    $\forall i.\ i < length\ ys \longrightarrow (xs\ !\ i,\ ys\ !\ i) \in ns$
  shows $(X,\ Y) \in multpw\ ns$
  $\langle proof \rangle$

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

4.2 Definition of the multiset extension of >-orders

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

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

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

lemma *ns-mul-extE*:
  assumes $(A,\ B) \in ns\text{-}mul\text{-}ext\ ns\ s$
  obtains $A1\ A2\ B1\ B2$ **where** $A = A1 + A2$ **and** $B = B1 + B2$
    and $(A1,\ B1) \in multpw\ ns$

and $\bigwedge b.\ b \in\# \ B2 \Longrightarrow \exists\, a.\ a \in\# \ A2 \wedge (a,\ b) \in s$
⟨*proof*⟩

lemmas *ns-mul-extI-old* = *ns-mul-extI*[*OF* - - *multpw-listI*[*OF* - *refl refl*], *rule-format*]

Same for the "greater than" order on multisets.

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

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

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

lemmas *s-mul-extI-old* = *s-mul-extI*[*OF* - - *multpw-listI*[*OF* - *refl refl*], *rule-format*]

4.3 Basic properties

lemma *s-mul-ext-mono*:
assumes $ns \subseteq ns'\ s \subseteq s'$ **shows** $s\text{-}mul\text{-}ext\ ns\ s \subseteq s\text{-}mul\text{-}ext\ ns'\ s'$
⟨*proof*⟩

lemma *ns-mul-ext-mono*:
assumes $ns \subseteq ns'\ s \subseteq s'$ **shows** $ns\text{-}mul\text{-}ext\ ns\ s \subseteq ns\text{-}mul\text{-}ext\ ns'\ s'$
⟨*proof*⟩

lemma *s-mul-ext-local-mono*:
assumes *sub*: $(set\text{-}mset\ xs \times set\text{-}mset\ ys) \cap ns \subseteq ns'\ (set\text{-}mset\ xs \times set\text{-}mset\ ys) \cap s \subseteq s'$
and *rel*: $(xs,ys) \in s\text{-}mul\text{-}ext\ ns\ s$
shows $(xs,ys) \in s\text{-}mul\text{-}ext\ ns'\ s'$
⟨*proof*⟩

lemma *ns-mul-ext-local-mono*:
assumes *sub*: $(set\text{-}mset\ xs \times set\text{-}mset\ ys) \cap ns \subseteq ns'\ (set\text{-}mset\ xs \times set\text{-}mset\ ys) \cap s \subseteq s'$
and *rel*: $(xs,ys) \in ns\text{-}mul\text{-}ext\ ns\ s$
shows $(xs,ys) \in ns\text{-}mul\text{-}ext\ ns'\ s'$
⟨*proof*⟩

lemma *s-mul-ext-ord-s* [*mono*]:

assumes $\bigwedge s\ t.\ ord\ s\ t \longrightarrow ord'\ s\ t$
shows $(s,\ t) \in s\text{-}mul\text{-}ext\ ns\ \{(s,t).\ ord\ s\ t\} \longrightarrow (s,\ t) \in s\text{-}mul\text{-}ext\ ns\ \{(s,t).\ ord'\ s\ t\}$
$\langle proof \rangle$

lemma *ns-mul-ext-ord-s* [*mono*]:
assumes $\bigwedge s\ t.\ ord\ s\ t \longrightarrow ord'\ s\ t$
shows $(s,\ t) \in ns\text{-}mul\text{-}ext\ ns\ \{(s,t).\ ord\ s\ t\} \longrightarrow (s,\ t) \in ns\text{-}mul\text{-}ext\ ns\ \{(s,t).\ ord'\ s\ t\}$
$\langle proof \rangle$

The empty multiset is the minimal element for these orders

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

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

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

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

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

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

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 $(A,\ B) \in s\text{-}mul\text{-}ext\ ns\ s$
$\langle proof \rangle$

Being stricly lesser than implies being lesser than

lemma *s-ns-mul-ext*:

assumes $(A,\ B) \in$ *s-mul-ext ns s*
shows $(A,\ B) \in$ *ns-mul-ext ns s*
⟨*proof*⟩

The non-strict order is reflexive.

lemma *multpw-refl′*:
assumes *locally-refl ns A*
shows $(A,\ A) \in$ *multpw ns*
⟨*proof*⟩

lemma *ns-mul-ext-refl-local*:
assumes *locally-refl ns A*
shows $(A,\ A) \in$ *ns-mul-ext ns s*
⟨*proof*⟩

lemma *ns-mul-ext-refl*:
assumes *refl ns*
shows $(A,\ A) \in$ *ns-mul-ext ns s*
⟨*proof*⟩

The orders are union-compatible

lemma *ns-s-mul-ext-union-multiset-l*:
assumes $(A,\ B) \in$ *ns-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-mul-ext ns s*
⟨*proof*⟩

lemma *s-mul-ext-union-compat*:
assumes $(A,\ B) \in$ *s-mul-ext ns s*
and *locally-refl ns C*
shows $(A + C,\ B + C) \in$ *s-mul-ext ns s*
⟨*proof*⟩

lemma *ns-mul-ext-union-compat*:
assumes $(A,\ B) \in$ *ns-mul-ext ns s*
and *locally-refl ns C*
shows $(A + C,\ B + C) \in$ *ns-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-mul-ext NS S*
⟨*proof*⟩

lemma *supset-imp-s-mul-ext*:
assumes $A \supset\# B$
shows $(A,\ B) \in$ *s-mul-ext NS S*
⟨*proof*⟩

end

definition *mul-ext* :: $('a \Rightarrow 'a \Rightarrow bool \times bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool \times bool$
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-mul-ext ns s*, (*mset xs*, *mset ys*) $\in$ *ns-mul-ext ns s*)

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

lemma *smulextp-cong*[*fundef-cong*]:
assumes $xs1 = ys1$
and $xs2 = ys2$
and $\bigwedge x\ x'.\ x \in\# ys1 \Longrightarrow x' \in\# ys2 \Longrightarrow f\ x\ x' = g\ x\ x'$
shows *smulextp f xs1 xs2* = *smulextp g ys1 ys2*
⟨*proof*⟩

lemma *nsmulextp-cong*[*fundef-cong*]:
assumes $xs1 = ys1$
and $xs2 = ys2$
and $\bigwedge x\ x'.\ x \in\# ys1 \Longrightarrow x' \in\# ys2 \Longrightarrow f\ x\ x' = g\ x\ x'$
shows *nsmulextp f xs1 xs2* = *nsmulextp g ys1 ys2*
⟨*proof*⟩

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

lemma *mulextp-cong*[*fundef-cong*]:
assumes $xs1 = ys1$
and $xs2 = ys2$
and $\bigwedge x\ x'.\ x \in\# ys1 \Longrightarrow x' \in\# ys2 \Longrightarrow f\ x\ x' = g\ x\ x'$
shows *mulextp f xs1 xs2* = *mulextp g ys1 ys2*
⟨*proof*⟩

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

lemma *mset-ns-mul-ext*:

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

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

lemma *nsmulextp-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 $\bigwedge s\ t.\ s \in set\ ss \Longrightarrow t \in set\ ts \Longrightarrow fst\ (order\ s\ t) \Longrightarrow fst\ (order'\ (f\ s)\ (f\ t))$
and $\bigwedge s\ t.\ s \in set\ ss \Longrightarrow t \in set\ ts \Longrightarrow snd\ (order\ s\ t) \Longrightarrow snd\ (order'\ (f\ s)\ (f\ t))$
and $snd\ (mul\text{-}ext\ order\ ss\ ts)$
shows $snd\ (mul\text{-}ext\ order'\ (map\ f\ ss)\ (map\ f\ ts))$
$\langle proof \rangle$

lemma *stri-mul-ext-map*:
assumes $\bigwedge s\ t.\ s \in set\ ss \Longrightarrow t \in set\ ts \Longrightarrow fst\ (order\ s\ t) \Longrightarrow fst\ (order'\ (f\ s)\ (f\ t))$
and $\bigwedge s\ t.\ s \in set\ ss \Longrightarrow t \in set\ ts \Longrightarrow snd\ (order\ s\ t) \Longrightarrow snd\ (order'\ (f\ s)\ (f\ t))$
and $fst\ (mul\text{-}ext\ order\ ss\ ts)$
shows $fst\ (mul\text{-}ext\ order'\ (map\ f\ ss)\ (map\ f\ ts))$
$\langle proof \rangle$

lemma *mul-ext-arg-empty*: $snd\ (mul\text{-}ext\ f\ []\ xs) \Longrightarrow xs = []$
$\langle proof \rangle$

The non-strict order is irreflexive

lemma *s-mul-ext-irrefl*: **assumes** *irr*: $irrefl\text{-}on\ (set\text{-}mset\ A)\ S$
and *S-NS*: $S \subseteq NS$
and *compat*: $S\ O\ NS \subseteq S$
shows $(A,A) \notin s\text{-}mul\text{-}ext\ NS\ S$ $\langle proof \rangle$

lemma *mul-ext-irrefl*: **assumes** $\bigwedge\ x.\ x \in set\ xs \Longrightarrow \neg\ fst\ (rel\ x\ x)$
and $\bigwedge\ x\ y\ z.\ fst\ (rel\ x\ y) \Longrightarrow snd\ (rel\ y\ z) \Longrightarrow fst\ (rel\ x\ z)$
and $\bigwedge\ x\ y.\ fst\ (rel\ x\ y) \Longrightarrow snd\ (rel\ x\ y)$
shows $\neg\ fst\ (mul\text{-}ext\ rel\ xs\ xs)$
$\langle proof \rangle$

The non-strict order is transitive.

lemma *ns-mul-ext-trans*:
assumes *trans s trans ns compatible-l ns s compatible-r ns s refl ns*

and $(A,\ B) \in$ *ns-mul-ext ns s*
and $(B,\ C) \in$ *ns-mul-ext ns s*
shows $(A,\ C) \in$ *ns-mul-ext ns s*
⟨*proof*⟩

The strict order is trans.

lemma *s-mul-ext-trans*:
assumes *trans s trans ns compatible-l ns s compatible-r ns s refl ns*
and $(A,\ B) \in$ *s-mul-ext ns s*
and $(B,\ C) \in$ *s-mul-ext ns s*
shows $(A,\ C) \in$ *s-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 *trans s trans ns compatible-l ns s compatible-r ns s refl ns*
and $(A,\ B) \in$ *s-mul-ext ns s*
and $(B,\ C) \in$ *ns-mul-ext ns s*
shows $(A,\ C) \in$ *s-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 *trans s trans ns compatible-l ns s compatible-r ns s refl ns*
and $(A,\ B) \in$ *ns-mul-ext ns s*
and $(B,\ C) \in$ *s-mul-ext ns s*
shows $(A,\ C) \in$ *s-mul-ext ns s*
⟨*proof*⟩

s-mul-ext is strongly normalizing

lemma *SN-s-mul-ext-strong*:
assumes *order-pair s ns*
and $\forall\, y.\ y \in\#\ M \longrightarrow$ *SN-on s* $\{y\}$
shows *SN-on* (*s-mul-ext ns s*) $\{M\}$
⟨*proof*⟩

lemma *SN-s-mul-ext*:
assumes *order-pair s ns SN s*
shows *SN* (*s-mul-ext ns s*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *mul-ext-compat*:
  assumes *compat*: $\bigwedge$ *s t u*. $[\![s \in set\ ss;\ t \in set\ ts;\ u \in set\ us]\!] \Longrightarrow$
    $(snd\ (f\ s\ t) \wedge fst\ (f\ t\ u) \longrightarrow fst\ (f\ s\ u)) \wedge$
    $(fst\ (f\ s\ t) \wedge snd\ (f\ t\ u) \longrightarrow fst\ (f\ s\ u)) \wedge$
    $(snd\ (f\ s\ t) \wedge snd\ (f\ t\ u) \longrightarrow snd\ (f\ s\ u)) \wedge$
    $(fst\ (f\ s\ t) \wedge fst\ (f\ t\ u) \longrightarrow fst\ (f\ s\ u))$
  shows
    $(snd\ (mul\text{-}ext\ f\ ss\ ts) \wedge fst\ (mul\text{-}ext\ f\ ts\ us) \longrightarrow fst\ (mul\text{-}ext\ f\ ss\ us)) \wedge$
    $(fst\ (mul\text{-}ext\ f\ ss\ ts) \wedge snd\ (mul\text{-}ext\ f\ ts\ us) \longrightarrow fst\ (mul\text{-}ext\ f\ ss\ us)) \wedge$
    $(snd\ (mul\text{-}ext\ f\ ss\ ts) \wedge snd\ (mul\text{-}ext\ f\ ts\ us) \longrightarrow snd\ (mul\text{-}ext\ f\ ss\ us)) \wedge$
    $(fst\ (mul\text{-}ext\ f\ ss\ ts) \wedge fst\ (mul\text{-}ext\ f\ ts\ us) \longrightarrow fst\ (mul\text{-}ext\ f\ ss\ us))$
⟨*proof*⟩

lemma *mul-ext-cong*[*fundef-cong*]:
  assumes *mset xs1* = *mset ys1*
    and *mset xs2* = *mset ys2*
    and $\bigwedge$ $x\ x'$. $x \in set\ ys1 \Longrightarrow x' \in set\ ys2 \Longrightarrow f\ x\ x' = g\ x\ x'$
  shows *mul-ext f xs1 xs2* = *mul-ext g ys1 ys2*
  ⟨*proof*⟩

lemma *all-nstri-imp-mul-nstri*:
  assumes $\forall\, i<length\ ys.\ snd\ (f\ (xs\ !\ i)\ (ys\ !\ i))$
    and *length xs* = *length ys*
  shows *snd* (*mul-ext f xs ys*)
⟨*proof*⟩

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

The next lemma is a local version of strong-normalization of the multiset 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*).
  $(\forall\, y{\in}set\ ys.\ SN\text{-}on\ \{(s\ ,t).\ fst\ (g\ s\ t)\}\ \{y\}) \wedge$
  $fst\ (mul\text{-}ext\ g\ ys\ xs)\}$

⟨*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*
  λs *ns*. {(*as*, *bs*). (*mset as*, *mset bs*) $\in$ *s-mul-ext ns s*}
  λs *ns*. {(*as*, *bs*). (*mset as*, *mset bs*) $\in$ *ns-mul-ext ns s*}
⟨*proof*⟩

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

lemma *ns-mul-ext-singleton* [*simp*, *intro*]:
  $(a, b) \in ns \Longrightarrow$ ({#*a*#}, {#*b*#}) $\in$ *ns-mul-ext ns s*
  ⟨*proof*⟩

lemma *ns-mul-ext-singleton2*:
  $(a, b) \in s \Longrightarrow$ ({#*a*#}, {#*b*#}) $\in$ *ns-mul-ext ns s*
  ⟨*proof*⟩

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

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

lemma *s-mul-ext-extend-left*:
assumes $(B, C) \in$ *s-mul-ext* $W\ S$
shows $(A + B, C) \in$ *s-mul-ext* $W\ S$
⟨*proof*⟩

lemma *mul-ext-mono*:
assumes $\bigwedge x\ y.\ [\![x \in \mathit{set}\ \mathit{xs};\ y \in \mathit{set}\ \mathit{ys};\ \mathit{fst}\ (P\ x\ y)]\!] \Longrightarrow \mathit{fst}\ (P'\ x\ y)$
and $\bigwedge x\ y.\ [\![x \in \mathit{set}\ \mathit{xs};\ y \in \mathit{set}\ \mathit{ys};\ \mathit{snd}\ (P\ x\ y)]\!] \Longrightarrow \mathit{snd}\ (P'\ x\ y)$
shows
$\mathit{fst}\ (\mathit{mul\text{-}ext}\ P\ \mathit{xs}\ \mathit{ys}) \Longrightarrow \mathit{fst}\ (\mathit{mul\text{-}ext}\ P'\ \mathit{xs}\ \mathit{ys})$
$\mathit{snd}\ (\mathit{mul\text{-}ext}\ P\ \mathit{xs}\ \mathit{ys}) \Longrightarrow \mathit{snd}\ (\mathit{mul\text{-}ext}\ P'\ \mathit{xs}\ \mathit{ys})$
⟨*proof*⟩

4.5 Special case: non-strict order is equality

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

lemma *ns-mul-ext-IdI*:
assumes $M = X + Z$ **and** $N = Y + Z$ **and** $\forall y \in$ *set-mset* $Y.\ \exists x \in$ *set-mset* $X.\ (x, y) \in R$
shows $(M, N) \in$ *ns-mul-ext Id* R
⟨*proof*⟩

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

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

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

lemma *ns-mul-ext-Id-eq*:
  ns-mul-ext Id R = (*s-mul-ext Id R*)$^=$
  ⟨*proof*⟩

lemma *subseteq-mset-imp-ns-mul-ext-Id*:
  assumes *A* ⊆# *B*
  shows (*B*, *A*) ∈ *ns-mul-ext Id R*
⟨*proof*⟩

lemma *subset-mset-imp-s-mul-ext-Id*:
  assumes *A* ⊂# *B*
  shows (*B*, *A*) ∈ *s-mul-ext Id R*
  ⟨*proof*⟩

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*: ∃ *s ns*. *list-order-extension-impl s ns mul-ext*
⟨*proof*⟩

context fixes *sns* :: *'a* ⇒ *'a* ⇒ *bool* × *bool*
begin

fun *mul-ext-impl* :: *'a list* ⇒ *'a list* ⇒ *bool* × *bool*
and *mul-ex-dom* :: *'a list* ⇒ *'a list* ⇒ *'a* ⇒ *'a list* ⇒ *bool* × *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*, -) ⇒ *if snd* (*mul-ext-impl* (*xs* @ *xs'*) (*filter* (λ*y*. ¬ *fst* (*sns x y*)) *ys*))
then (*True*, *True*)

else mul-ex-dom xs $(x \# xs')$ *y ys*
| (*False*, *True*) ⇒ *or2* (*mul-ext-impl* (*xs* @ *xs'*) *ys*) (*mul-ex-dom xs* $(x \# xs')$ *y ys*)
| - ⇒ *mul-ex-dom xs* $(x \# xs')$ *y ys*)

end

context
begin
lemma *mul-ext-impl-sound0*:
mul-ext-impl sns xs ys = *mult2-impl* ($\lambda x\ y.$ *sns y x*) *ys xs*
mul-ex-dom sns xs xs' y ys = *mult2-ex-dom* ($\lambda x\ y.$ *sns y x*) *y ys xs xs'*
⟨*proof*⟩ **definition** *cond1* **where**
cond1 f bs y xs ys ≡
((∃ *b*. *b* ∈ *set bs* ∧ *fst* (*f b y*) ∧ *snd* (*mul-ext f* (*remove1 b xs*) [*y*←*ys* . ¬ *fst* (*f b y*)]))
∨ (∃ *b*. *b* ∈ *set bs* ∧ *snd* (*f b y*) ∧ *fst* (*mul-ext f* (*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 ≡ (*cond1 f bs y xs ys*
∨ (∃ *b*. *b* ∈ *set bs* ∧ *snd* (*f b y*) ∧ *snd* (*mul-ext f* (*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*
⟨*proof*⟩

lemma *mul-ext-impl-sound*:
shows *mul-ext-impl f xs ys* = *mul-ext f xs ys*
⟨*proof*⟩

lemma *mul-ext-code* [*code*]: *mul-ext* = *mul-ext-impl*
⟨*proof*⟩

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

fun *ass-list-to-single-list* :: ($'a \times nat$) *list* ⇒ *'a list*
where
ass-list-to-single-list [] = []

| *ass-list-to-single-list* $((x,\ n)\ \#\ xs) = \mathit{replicate}\ n\ x\ @\ \mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ xs$

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

lemma *count-mset-replicate* [*simp*]:
$\mathit{count}\ (\mathit{mset}\ (\mathit{replicate}\ n\ x))\ x = n$
⟨*proof*⟩

lemma *count-mset-lal-ge*:
$(x,\ n) \in \mathit{set}\ xs \Longrightarrow \mathit{count}\ (\mathit{mset}\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ xs))\ x \geq n$
⟨*proof*⟩

lemma *count-of-count-mset-lal* [*simp*]:
$\mathit{distinct}\ (\mathit{map}\ \mathit{fst}\ y) \Longrightarrow \mathit{count\text{-}of}\ y\ x = \mathit{count}\ (\mathit{mset}\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ y))\ x$
⟨*proof*⟩

lemma *Bag-mset*: $\mathit{Bag}\ xs = \mathit{mset}\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ (\mathit{DAList.impl\text{-}of}\ xs))$
⟨*proof*⟩

lemma *Bag-Alist-Cons*:
$x \notin \mathit{fst}\ `\ \mathit{set}\ xs \Longrightarrow \mathit{distinct}\ (\mathit{map}\ \mathit{fst}\ xs) \Longrightarrow$
$\mathit{Bag}\ (\mathit{Alist}\ ((x,\ n)\ \#\ xs)) = \mathit{mset}\ (\mathit{replicate}\ n\ x) + \mathit{Bag}\ (\mathit{Alist}\ xs)$
⟨*proof*⟩

lemma *mset-lal* [*simp*]:
$\mathit{distinct}\ (\mathit{map}\ \mathit{fst}\ xs) \Longrightarrow \mathit{mset}\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ xs) = \mathit{Bag}\ (\mathit{Alist}\ xs)$
⟨*proof*⟩

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

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

lemma *smulextp-code*[*code*]:
$\mathit{smulextp}\ f\ (\mathit{Bag}\ xs)\ (\mathit{Bag}\ ys) \longleftrightarrow \mathit{fst}\ (\mathit{mul\text{-}ext}\ f\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ (\mathit{DAList.impl\text{-}of}$
$xs))\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ (\mathit{DAList.impl\text{-}of}\ ys)))$
⟨*proof*⟩

lemma *nsmulextp-code*[*code*]:
$\mathit{nsmulextp}\ f\ (\mathit{Bag}\ xs)\ (\mathit{Bag}\ ys) \longleftrightarrow \mathit{snd}\ (\mathit{mul\text{-}ext}\ f\ (\mathit{ass\text{-}list\text{-}to\text{-}single\text{-}list}\ (\mathit{DAList.impl\text{-}of}$

xs)) (*ass-list-to-single-list* (*DAList.impl-of ys*)))
  ⟨*proof*⟩

lemma *mulextp-code*[*code*]:
  mulextp f (*Bag xs*) (*Bag ys*) = *mul-ext f* (*ass-list-to-single-list* (*DAList.impl-of xs*)) (*ass-list-to-single-list* (*DAList.impl-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
    First-Order-Terms.Subterm-and-Context
    Knuth-Bendix-Order.Order-Pair
    Polynomial-Factorization.Missing-List
    Status
    Precedence
    Multiset-Extension2
    HOL.Zorn
begin

datatype order-tag = Lex | Mul

locale wpo =
  fixes n :: nat
    and S NS :: ('f, 'v) term rel
    and prc :: ('f × nat ⇒ 'f × nat ⇒ bool × bool)
    and prl :: 'f × nat ⇒ bool
    and σσ :: 'f status
    and c :: 'f × nat ⇒ order-tag
    and ssimple :: bool
    and large :: 'f × nat ⇒ bool
begin

fun wpo :: ('f, 'v) term ⇒ ('f, 'v) term ⇒ bool × bool
  where
    wpo s t = (if (s,t) ∈ S then (True, True) else
        if (s,t) ∈ NS then (case s of
      Var x ⇒ (False,
        (case t of
          Var y ⇒ x = y
        | Fun g ts ⇒ status σσ (g, length ts) = [] ∧ prl (g, length ts)))
    | Fun f ss ⇒
```

$if\ \exists\ i \in set\ (status\ \sigma\sigma\ (f,\ length\ ss)).\ snd\ (wpo\ (ss\ !\ i)\ t)\ then\ (True,\ True)$
$else$
$(case\ t\ of$
$Var\ \text{-} \Rightarrow (False,\ ssimple \wedge large\ (f,\ length\ ss))$
$|\ Fun\ g\ ts \Rightarrow$
$(case\ prc\ (f,\ length\ ss)\ (g,\ length\ ts)\ of\ (prs,\ prns) \Rightarrow$
$if\ prns \wedge (\forall\ j \in set\ (status\ \sigma\sigma\ (g,\ length\ ts)).\ fst\ (wpo\ s\ (ts\ !\ j)))\ then$
$if\ prs\ then\ (True,\ True)$
$else\ let\ ss' = map\ (\lambda\ i.\ ss\ !\ i)\ (status\ \sigma\sigma\ (f,\ length\ ss));$
$ts' = map\ (\lambda\ i.\ ts\ !\ i)\ (status\ \sigma\sigma\ (g,\ length\ ts));$
$cf = c\ (f,length\ ss);$
$cg = c\ (g,length\ ts)$
$in\ if\ cf = Lex \wedge cg = Lex$
$then\ lex\text{-}ext\ wpo\ n\ ss'\ ts'$
$else\ if\ cf = Mul \wedge cg = Mul$
$then\ mul\text{-}ext\ wpo\ ss'\ ts'$
$else\ (length\ ss' \neq 0 \wedge length\ ts' = 0,\ length\ ts' = 0)$
$else\ (False,\ False))))$
$else\ (False,\ False))$

declare *wpo.simps* [*simp del*]

abbreviation *wpo-s* (**infix** ‹≻› *50*) **where** $s \succ t \equiv fst\ (wpo\ s\ t)$
abbreviation *wpo-ns* (**infix** ‹⪰› *50*) **where** $s \succeq t \equiv snd\ (wpo\ s\ t)$

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

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

lemma *S-imp-wpo-s*: $(s,t) \in S \Longrightarrow s \succ t$ ⟨*proof*⟩

end

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

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

definition *trans-precedence* **where** $trans\text{-}precedence\ prc = (\forall\ f\ g\ h.$
$(fst\ (prc\ f\ g) \longrightarrow snd\ (prc\ g\ h) \longrightarrow fst\ (prc\ f\ h)) \wedge$
$(snd\ (prc\ f\ g) \longrightarrow fst\ (prc\ g\ h) \longrightarrow fst\ (prc\ f\ h)) \wedge$
$(snd\ (prc\ f\ g) \longrightarrow snd\ (prc\ g\ h) \longrightarrow snd\ (prc\ f\ h)))$

locale *wpo-with-basic-assms* = *wpo* +
order-pair + *irrefl-precedence* +
constrains S :: $('f,\ 'v)$ *term rel* **and** NS :: -
and *prc* :: $'f \times nat \Rightarrow 'f \times nat \Rightarrow bool \times bool$
and *prl* :: $'f \times nat \Rightarrow bool$
and *ssimple* :: *bool*
and *large* :: $'f \times nat \Rightarrow bool$
and c :: $'f \times nat \Rightarrow$ *order-tag*
and n :: *nat*
and $\sigma\sigma$ :: $'f$ *status*
assumes *subst-S*: $(s,t) \in S \Longrightarrow (s \cdot \sigma,\ t \cdot \sigma) \in S$
and *subst-NS*: $(s,t) \in NS \Longrightarrow (s \cdot \sigma,\ t \cdot \sigma) \in NS$
and *irrefl-S*: *irrefl* S
and *S-imp-NS*: $S \subseteq NS$
and *ss-status*: $ssimple \Longrightarrow i \in set\ (status\ \sigma\sigma\ fn) \Longrightarrow simple\text{-}arg\text{-}pos\ S\ fn\ i$
and *large*: $ssimple \Longrightarrow large\ fn \Longrightarrow fst\ (prc\ fn\ gm) \lor snd\ (prc\ fn\ gm) \land status\ \sigma\sigma\ gm = []$
and *large-trans*: $ssimple \Longrightarrow large\ fn \Longrightarrow snd\ (prc\ gm\ fn) \Longrightarrow large\ gm$
and *ss-S-non-empty*: $ssimple \Longrightarrow S \neq \{\}$
begin
abbreviation $\sigma \equiv status\ \sigma\sigma$

lemma *ss-NS-not-UNIV*: $ssimple \Longrightarrow NS \neq UNIV$
⟨*proof*⟩

lemmas $\sigma = status[of\ \sigma\sigma]$
lemma σE: $i \in set\ (\sigma\ (f,\ length\ ss)) \Longrightarrow ss\ !\ i \in set\ ss$ ⟨*proof*⟩

lemma *wpo-ns-imp-NS*: $s \succeq t \Longrightarrow (s,t) \in NS$
⟨*proof*⟩

lemma *wpo-s-imp-NS*: $s \succ t \Longrightarrow (s,t) \in NS$
⟨*proof*⟩

lemma *wpo-least-1*: **assumes** $prl\ (f,length\ ss)$
and $(t,\ Fun\ f\ ss) \in NS$
and $\sigma\ (f,length\ ss) = []$
shows $t \succeq Fun\ f\ ss$
⟨*proof*⟩

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 $\neg\ Fun\ f\ ss \succ t$
⟨*proof*⟩

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*: σ $(f,length\ ss) = []$
and *S*: $\bigwedge x.\ (Fun\ f\ ss,\ x) \notin S$
and *u*: $u = Var\ x$
shows $u \succeq t$
⟨*proof*⟩

lemma *wpo-compat*: $(s \succeq t \wedge t \succ u \longrightarrow s \succ u) \wedge$
$(s \succ t \wedge t \succeq u \longrightarrow s \succ u) \wedge$
$(s \succeq t \wedge t \succeq u \longrightarrow s \succeq u)$ (**is** *?tran s t u*)
⟨*proof*⟩

context
assumes *ssimple*: *strictly-simple-status* $\sigma\sigma$ *NS*
begin
lemma *NS-arg′*:
assumes *i*: $i \in set\ (\sigma\ (f,length\ ts))$
shows $(Fun\ f\ ts,\ ts\ !\ i) \in NS$
⟨*proof*⟩

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

context
fixes *f s t bef aft*
assumes *ctxt-NS*: $(s,t) \in NS \Longrightarrow (Fun\ f\ (bef\ @\ s\ \#\ aft),\ Fun\ f\ (bef\ @\ t\ \#\ aft)) \in NS$
begin

lemma *wpo-ns-pre-mono′*:
defines $\sigma f \equiv \sigma\ (f,\ Suc\ (length\ bef + length\ aft))$
assumes *rel*: $(wpo\text{-}ns\ s\ t)$
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*⟩

end
end
end

locale *wpo-with-assms* = *wpo-with-basic-assms* + *order-pair* +
constrains *S* :: ('*f*, '*v*) *term rel* **and** *NS* :: -
and *prc* :: '*f* × *nat* ⇒ '*f* × *nat* ⇒ *bool* × *bool*
and *prl* :: '*f* × *nat* ⇒ *bool*
and *ssimple* :: *bool*
and *large* :: '*f* × *nat* ⇒ *bool*
and *c* :: '*f* × *nat* ⇒ *order-tag*
and *n* :: *nat*
and *σσ* :: '*f status*
assumes *ctxt-NS*: (*s*,*t*) ∈ *NS* ⟹ (*Fun f* (*bef* @ *s* # *aft*), *Fun f* (*bef* @ *t* # *aft*)) ∈ *NS*
and *ws-status*: *i* ∈ *set* (*status σσ fn*) ⟹ *simple-arg-pos NS fn i*
begin

lemma *ssimple*: *strictly-simple-status σσ NS*
⟨*proof*⟩

lemma *trans-prc*: *trans-precedence prc*
⟨*proof*⟩

lemma *NS-arg*: **assumes** *i*: *i* ∈ *set* (*σ* (*f*,*length ts*))
shows (*Fun f ts*, *ts* ! *i*) ∈ *NS*
⟨*proof*⟩

lemma *NS-subterm*: **assumes** *all*: $\bigwedge$ *f k*. *set* (*σ* (*f*,*k*)) = {*0* ..< *k*}
shows $s \trianglerighteq t$ ⟹ (*s*,*t*) ∈ *NS*
⟨*proof*⟩

lemma *wpo-ns-refl*: $s \succeq s$
⟨*proof*⟩

lemma *subterm-wpo-s-arg*: **assumes** *i*: *i* ∈ *set* (*σ* (*f*,*length ss*))
shows *Fun f ss* $\succ$ *ss* ! *i*
⟨*proof*⟩

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

lemma *wpo-irrefl*: $\neg$ $(s \succ s)$
$\langle proof \rangle$

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

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* σf. *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* σf. ((*map* ((!) (*bef* @ *s* # *aft*)) σf) ! *i*) $\succeq$ ((*map* ((!) (*bef* @ *t* # *aft*)) σf) ! *i*))
  $\langle proof \rangle$

lemma *wpo-stable*: **fixes** δ :: $('f,'v)$*subst*
  shows $(s \succ t \longrightarrow s \cdot \delta \succ t \cdot \delta) \wedge (s \succeq t \longrightarrow s \cdot \delta \succeq t \cdot \delta)$
  $\langle proof \rangle$

theorem *wpo-order-pair*: *order-pair WPO-S WPO-NS*
$\langle proof \rangle$

theorem *WPO-S-subst*: (*s*,*t*) $\in$ *WPO-S* $\Longrightarrow$ $(s \cdot \sigma, t \cdot \sigma)$ $\in$ *WPO-S* **for** σ
  $\langle proof \rangle$

theorem *WPO-NS-subst*: (*s*,*t*) $\in$ *WPO-NS* $\Longrightarrow$ $(s \cdot \sigma, t \cdot \sigma)$ $\in$ *WPO-NS* **for** σ
  $\langle proof \rangle$

theorem *WPO-NS-ctxt*: (*s*,*t*) $\in$ *WPO-NS* $\Longrightarrow$ (*Fun f* (*bef* @ *s* # *aft*), *Fun f* (*bef* @ *t* # *aft*)) $\in$ *WPO-NS*
  $\langle proof \rangle$

theorem *WPO-S-subset-WPO-NS*: *WPO-S* $\subseteq$ *WPO-NS*
  $\langle proof \rangle$

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

lemma *subterm-wpo-s*: $s \rhd t \Longrightarrow s \succ t$
$\langle proof \rangle$

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

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

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

theorem *supt-subset-WPO-S*: $\{\rhd\} \subseteq$ *WPO-S*
⟨*proof*⟩

theorem *supteq-subset-WPO-NS*: $\{\unrhd\} \subseteq$ *WPO-NS*
⟨*proof*⟩

end
end

If we demand strong normalization of the underlying order and the precedence, then also WPO is strongly normalizing.

locale *wpo-with-SN-assms* = *wpo-with-assms* + *SN-order-pair* + *precedence* +
constrains *S* :: ('*f*, '*v*) *term rel* **and** *NS* :: -
and *prc* :: '*f* × *nat* ⇒ '*f* × *nat* ⇒ *bool* × *bool*
and *prl* :: '*f* × *nat* ⇒ *bool*
and *ssimple* :: *bool*
and *large* :: '*f* × *nat* ⇒ *bool*
and *c* :: '*f* × *nat* ⇒ *order-tag*
and *n* :: *nat*
and *σσ* :: '*f status*
begin

lemma *Var-not-S*[*simp*]: (*Var x*, *t*) $\notin$ *S*
⟨*proof*⟩

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

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

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 × nat ⇒ 'f × nat ⇒ bool × bool
    and prl :: 'f × nat ⇒ bool
    and c :: 'f × nat ⇒ order-tag
    and n :: nat
begin

fun rpo :: ('f, 'v) term ⇒ ('f, 'v) term ⇒ bool × bool
  where
    rpo (Var x) (Var y) = (False, x = y) |
    rpo (Var x) (Fun g ts) = (False, ts = [] ∧ prl (g,0)) |
     rpo (Fun f ss) (Var y) = (let con = (∃ s ∈ set ss. snd (rpo s (Var y))) in
(con,con)) |
    rpo (Fun f ss) (Fun g ts) = (
    if (∃ s ∈ 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 ∧ (∀ t ∈ set ts. fst (rpo (Fun f ss) t))
         then if prs
              then (True,True)
              else if c (f,length ss) = Lex ∧ c (g,length ts) = Lex
                   then lex-ext rpo n ss ts
                   else if c (f,length ss) = Mul ∧ c (g,length ts) = Mul
                        then mul-ext rpo ss ts
                        else (length ss ≠ 0 ∧ length ts = 0, length ts = 0)
          else (False,False)))
end

locale rpo-with-assms = precedence prc prl
  for prc :: 'f × nat ⇒ 'f × nat ⇒ bool × bool
    and prl :: 'f × nat ⇒ bool
    and c :: 'f × nat ⇒ order-tag
    and n :: nat
begin

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

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

lemma rpo-eq-wpo: rpo-pr s t = wpo s t
```

⟨*proof*⟩

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

theorem *RPO-SN-order-pair*: *SN-order-pair RPO-S RPO-NS*
⟨*proof*⟩

theorem *RPO-S-subst*: $(s,t) \in RPO\text{-}S \Longrightarrow (s \cdot \sigma,\ t \cdot \sigma) \in RPO\text{-}S$ **for** σ :: $('f,'a)subst$
⟨*proof*⟩

theorem *RPO-NS-subst*: $(s,t) \in RPO\text{-}NS \Longrightarrow (s \cdot \sigma,\ t \cdot \sigma) \in RPO\text{-}NS$ **for** σ :: $('f,'a)subst$
⟨*proof*⟩

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

theorem *RPO-S-ctxt*: (*s*,*t*) ∈ *RPO-S* ⟹ (*Fun f* (*bef* @ *s* # *aft*), *Fun f* (*bef* @ *t* # *aft*)) ∈ *RPO-S*
⟨*proof*⟩

theorem *RPO-S-subset-RPO-NS*: *RPO-S* ⊆ *RPO-NS*
⟨*proof*⟩

theorem *supt-subset-RPO-S*: {⊳} ⊆ *RPO-S*
⟨*proof*⟩

theorem *supteq-subset-RPO-NS*: {⊵} ⊆ *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 (∅, UNIV) and the status is the full one, i.e., taking parameters [0,..,n-1] for each n-ary symbol.

theory *LPO*
imports

WPO
begin

context
fixes *pr* :: ('*f* × *nat* ⇒ '*f* × *nat* ⇒ *bool* × *bool*)
and *prl* :: '*f* × *nat* ⇒ *bool*
and *n* :: *nat*
begin
fun *lpo* :: ('*f*, '*v*) *term* ⇒ ('*f*, '*v*) *term* ⇒ *bool* × *bool*
where
lpo (*Var x*) (*Var y*) = (*False*, *x* = *y*) |
lpo (*Var x*) (*Fun g ts*) = (*False*, *ts* = [] ∧ *prl* (*g*,*0*)) |
lpo (*Fun f ss*) (*Var y*) = (*let con* = (∃ *s* ∈ *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-SN-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*: $(s,t) \in$ *LPO-S* $\Longrightarrow (s \cdot \sigma,\ t \cdot \sigma) \in$ *LPO-S* **for** σ :: $('f,'a)subst$
⟨*proof*⟩

theorem *LPO-NS-subst*: $(s,t) \in$ *LPO-NS* $\Longrightarrow (s \cdot \sigma,\ t \cdot \sigma) \in$ *LPO-NS* **for** σ :: $('f,'a)subst$
⟨*proof*⟩

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

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

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

theorem *supt-subset-LPO-S*: $\{\rhd\} \subseteq$ *LPO-S*
⟨*proof*⟩

theorem *supteq-subset-LPO-NS*: $\{\unrhd\} \subseteq$ *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 precende *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 precendence 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* × *nat*)*set* **where** *lConsts* = { $(f,0)$ | f. *least* f}
definition *pr-strict'* **where** *pr-strict'* f g = (f ∉ *lConsts* ∧ (*pr-strict* f g ∨ g ∈ *lConsts*))
definition *pr-weak'* **where** *pr-weak'* f g = ((f ∉ *lConsts* ∧ *pr-weak* f g) ∨ g ∈ *lConsts*)

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

lemma *least-pr-weak'*: *least* f ⟹ *pr-weak'* g $(f,0)$ ⟨*proof*⟩

lemma *least-pr-weak'-trans*: *least* f ⟹ *pr-weak'* $(f,0)$ g ⟹ *least* (*fst* g) ∧ *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* $\equiv$
(*vars-term-ms* (*SCF s*) ⊆# *vars-term-ms* (*SCF t*) ∧ *weight s* ≤ *weight t*)

definition *weight-less s t* $\equiv$
(*vars-term-ms* (*SCF s*) ⊆# *vars-term-ms* (*SCF t*) ∧ *weight s* < *weight t*)

lemma *weight-le-less-iff*: *weight-le s t* ⟹ *weight-less s t* ⟷ *weight s* < *weight t*
⟨*proof*⟩

lemma *weight-less-iff*: *weight-less s t* ⟹ *weight-le s t* ∧ *weight s* < *weight t*
⟨*proof*⟩

abbreviation *weight-NS* ≡ {(*t*,*s*). *weight-le s t*}

abbreviation *weight-S* ≡ {(*t*,*s*). *weight-less s t*}

lemma *weight-le-mono-one*:
assumes *S*: *weight-le s t*
shows *weight-le* (*Fun f* (*ss1* @ *s* # *ss2*)) (*Fun f* (*ss1* @ *t* # *ss2*)) (**is** *weight-le* *?s ?t*)
⟨*proof*⟩

lemma *weight-le-ctxt*: *weight-le s t* ⟹ *weight-le* (*C*⟨*s*⟩) (*C*⟨*t*⟩)
⟨*proof*⟩

lemma *SCF-stable*:
assumes *vars-term-ms* (*SCF s*) ⊆# *vars-term-ms* (*SCF t*)
shows *vars-term-ms* (*SCF* (*s* · σ)) ⊆# *vars-term-ms* (*SCF* (*t* · σ))
⟨*proof*⟩

lemma *SN-weight-S*: *SN weight-S*
⟨*proof*⟩

lemma *weight-less-imp-le*: *weight-less s t* ⟹ *weight-le s t* ⟨*proof*⟩

lemma *weight-le-Var-Var*: *weight-le* (*Var x*) (*Var y*) $\longleftrightarrow$ *x* = *y*
⟨*proof*⟩
end

context *kbo* **begin**

lemma *kbo-altdef*:
kbo s t = (*if weight-le t s*
then if weight-less t s
then (*True*, *True*)
else (*case s of*
Var y ⇒ (*False*, (*case t of Var x* ⇒ *x* = *y* | *Fun g ts* ⇒ *ts* = [] ∧ *least g*))
| *Fun f ss* ⇒ (*case t of*
Var x ⇒ (*True*, *True*)
| *Fun g ts* ⇒ *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* ($s \cdot \sigma$) ($t \cdot \sigma$)
⟨*proof*⟩

lemma *weight-less-stable*:
assumes *weight-less s t*
shows *weight-less* ($s \cdot \sigma$) ($t \cdot \sigma$)
⟨*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* ⊆ *weight-S* **and** *weight-S O weight-NS* ⊆ *weight-S*
⟨*proof*⟩

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

context

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

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

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

lemma *kbo-as-wpo-with-assms*: **assumes** *bounded-arity* n (*funas-term* t)
shows $kbo\ s\ t = wpo.wpo\ s\ t$
⟨*proof*⟩
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-SN-assms weight-S weight-NS prec prl full-status False* $(\lambda f.\ False)$
and
kbo-as-wpo: *bounded-arity* n (*funas-term* t) $\Longrightarrow kbo\ s\ t = wpo.wpo\ n$ *weight-S weight-NS prec prl full-status* $(\lambda\text{-}.\ Lex)$ *False* $(\lambda f.\ False)\ s\ t$
⟨*proof*⟩

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 $t_1 > t_2 > t_3 > ...$ 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*: $S\ s\ t \Longrightarrow S\ (s \cdot (\sigma :: ('f,'a)\ subst))\ (t \cdot \sigma)$
⟨*proof*⟩

lemma *weight-is-arity-bound*: *weight t* $\leq$ *b* $\Longrightarrow$ *bounded-arity b* (*funas-term t*)
⟨*proof*⟩

lemma *KBO-SN-via-WPO*: *SN* {(*s*,*t*). *S s t*}
⟨*proof*⟩

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, 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.