

Abstract Rewriting

Christian Sternagel and René Thiemann

June 3, 2024

Abstract

We present an Isabelle formalization of abstract rewriting (see, e.g., [1]). First, we define standard relations like *joinability*, *meetability*, *conversion*, etc. Then, we formalize important properties of abstract rewrite systems, e.g., confluence and strong normalization. Our main concern is on strong normalization, since this formalization is the basis of [3] (which is mainly about strong normalization of term rewrite systems; see also `IsaFoR/CeTA`'s website¹). Hence lemmas involving strong normalization, constitute by far the biggest part of this theory. One of those is Newman's lemma.

Contents

1	Infinite Sequences	2
1.1	Operations on Infinite Sequences	2
1.2	Predicates on Natural Numbers	4
1.3	Assembling Infinite Words from Finite Words	4
2	Abstract Rewrite Systems	6
2.1	Definitions	7
2.2	Properties of ARSs	10
2.3	Newman's Lemma	19
2.4	Commutation	21
2.5	Strong Normalization	22
2.6	Terminating part of a relation	27
3	Relative Rewriting	31
4	Strongly Normalizing Orders	39

¹<http://cl-informatik.uibk.ac.at/software/ceta>

5	Carriers of Strongly Normalizing Orders	43
5.1	The standard semiring over the naturals	44
5.2	The standard semiring over the Archimedean fields using delta-orderings	44
5.3	The standard semiring over the integers	45
5.4	The arctic semiring over the integers	45
5.5	The arctic semiring over an arbitrary archimedean field . . .	46

A description of this formalization will be available in [2].

1 Infinite Sequences

```

theory Seq
imports
  Main
  HOL-Library.Infinite-Set
begin

```

Infinite sequences are represented by functions of type $nat \Rightarrow 'a$.

```

type-synonym 'a seq = nat  $\Rightarrow$  'a

```

1.1 Operations on Infinite Sequences

An infinite sequence is *linked* by a binary predicate P if every two consecutive elements satisfy it. Such a sequence is called a P -chain.

```

abbreviation (input) chainp :: ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  'a seq  $\Rightarrow$  bool where
  chainp P S  $\equiv$   $\forall i. P (S i) (S (Suc i))$ 

```

Special version for relations.

```

abbreviation (input) chain :: 'a rel  $\Rightarrow$  'a seq  $\Rightarrow$  bool where
  chain r S  $\equiv$  chainp ( $\lambda x y. (x, y) \in r$ ) S

```

Extending a chain at the front.

```

lemma cons-chainp:
  assumes P x (S 0) and chainp P S
  shows chainp P (case-nat x S) (is chainp P ?S)
<proof>

```

Special version for relations.

```

lemma cons-chain:
  assumes (x, S 0)  $\in$  r and chain r S shows chain r (case-nat x S)
<proof>

```

A chain admits arbitrary transitive steps.

```

lemma chainp-imp-relpowp:
  assumes chainp P S shows (P $\wedge^j$ ) (S i) (S (i + j))
<proof>

```

lemma *chain-imp-relpow*:
assumes *chain r S* **shows** $(S\ i, S\ (i + j)) \in r^{\sim j}$
 $\langle proof \rangle$

lemma *chainp-imp-tranclp*:
assumes *chainp P S* **and** $i < j$ **shows** $P^{\sim++} (S\ i)\ (S\ j)$
 $\langle proof \rangle$

lemma *chain-imp-trancl*:
assumes *chain r S* **and** $i < j$ **shows** $(S\ i, S\ j) \in r^{\sim+}$
 $\langle proof \rangle$

A chain admits arbitrary reflexive and transitive steps.

lemma *chainp-imp-rtranclp*:
assumes *chainp P S* **and** $i \leq j$ **shows** $P^{\sim**} (S\ i)\ (S\ j)$
 $\langle proof \rangle$

lemma *chain-imp-rtrancl*:
assumes *chain r S* **and** $i \leq j$ **shows** $(S\ i, S\ j) \in r^{\sim*}$
 $\langle proof \rangle$

If for every i there is a later index $f\ i$ such that the corresponding elements satisfy the predicate P , then there is a P -chain.

lemma *stepfun-imp-chainp'*:
assumes $\forall i \geq n::nat. f\ i \geq i \wedge P\ (S\ i)\ (S\ (f\ i))$
shows *chainp P* $(\lambda i. S\ ((f\ \sim i)\ n))$ **(is chainp P ?T)**
 $\langle proof \rangle$

lemma *stepfun-imp-chainp*:
assumes $\forall i \geq n::nat. f\ i > i \wedge P\ (S\ i)\ (S\ (f\ i))$
shows *chainp P* $(\lambda i. S\ ((f\ \sim i)\ n))$ **(is chainp P ?T)**
 $\langle proof \rangle$

lemma *subchain*:
assumes $\forall i::nat > n. \exists j > i. P\ (f\ i)\ (f\ j)$
shows $\exists \varphi. (\forall i\ j. i < j \longrightarrow \varphi\ i < \varphi\ j) \wedge (\forall i. P\ (f\ (\varphi\ i))\ (f\ (\varphi\ (Suc\ i))))$
 $\langle proof \rangle$

If for every i there is a later index j such that the corresponding elements satisfy the predicate P , then there is a P -chain.

lemma *steps-imp-chainp'*:
assumes $\forall i \geq n::nat. \exists j \geq i. P\ (S\ i)\ (S\ j)$ **shows** $\exists T. chainp\ P\ T$
 $\langle proof \rangle$

lemma *steps-imp-chainp*:
assumes $\forall i \geq n::nat. \exists j > i. P\ (S\ i)\ (S\ j)$ **shows** $\exists T. chainp\ P\ T$
 $\langle proof \rangle$

1.2 Predicates on Natural Numbers

If some property holds for infinitely many natural numbers, obtain an index function that points to these numbers in increasing order.

```
locale infinitely-many =  
  fixes p :: nat  $\Rightarrow$  bool  
  assumes infinite: INFM j. p j  
begin  
  
lemma inf:  $\exists j \geq i. p j$  <proof>  
  
fun index :: nat seq where  
  index 0 = (LEAST n. p n)  
| index (Suc n) = (LEAST k. p k  $\wedge$  k > index n)  
  
lemma index-p: p (index n)  
<proof>  
  
lemma index-ordered: index n < index (Suc n)  
<proof>  
  
lemma index-not-p-between:  
  assumes i1: index n < i  
  and i2: i < index (Suc n)  
  shows  $\neg p i$   
<proof>  
  
lemma index-ordered-le:  
  assumes i  $\leq$  j shows index i  $\leq$  index j  
<proof>  
  
lemma index-surj:  
  assumes k  $\geq$  index l  
  shows  $\exists i j. k = \text{index } i + j \wedge \text{index } i + j < \text{index } (\text{Suc } i)$   
<proof>  
  
lemma index-ordered-less:  
  assumes i < j shows index i < index j  
<proof>  
  
lemma index-not-p-start: assumes i: i < index 0 shows  $\neg p i$   
<proof>  
  
end
```

1.3 Assembling Infinite Words from Finite Words

Concatenate infinitely many non-empty words to an infinite word.

```
fun inf-concat-simple :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  nat  $\Rightarrow$  (nat  $\times$  nat) where
```

$inf\text{-concat-simple } f \ 0 = (0, 0)$
 $| \text{inf-concat-simple } f \ (Suc \ n) = ($
 $\quad \text{let } (i, j) = \text{inf-concat-simple } f \ n \ \text{in}$
 $\quad \text{if } Suc \ j < f \ i \ \text{then } (i, Suc \ j)$
 $\quad \text{else } (Suc \ i, 0))$

lemma *inf-concat-simple-add*:
assumes $ck: \text{inf-concat-simple } f \ k = (i, j)$
and $jl: j + l < f \ i$
shows $\text{inf-concat-simple } f \ (k + l) = (i, j + l)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-simple-surj-zero*: $\exists k. \text{inf-concat-simple } f \ k = (i, 0)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-simple-surj*:
assumes $j < f \ i$
shows $\exists k. \text{inf-concat-simple } f \ k = (i, j)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-simple-mono*:
assumes $k \leq k'$ **shows** $\text{fst } (\text{inf-concat-simple } f \ k) \leq \text{fst } (\text{inf-concat-simple } f \ k')$
 $\langle \text{proof} \rangle$

fun *inf-concat* :: $(\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat} \times \text{nat}$ **where**
 $\text{inf-concat } n \ 0 = (\text{LEAST } j. n \ j > 0, 0)$
 $| \text{inf-concat } n \ (Suc \ k) = (\text{let } (i, j) = \text{inf-concat } n \ k \ \text{in } (\text{if } Suc \ j < n \ i \ \text{then } (i, Suc \ j) \ \text{else } (\text{LEAST } i'. i' > i \wedge n \ i' > 0, 0)))$

lemma *inf-concat-bounds*:
assumes $\text{inf}: \text{INFM } i. n \ i > 0$
and $\text{res}: \text{inf-concat } n \ k = (i, j)$
shows $j < n \ i$
 $\langle \text{proof} \rangle$

lemma *inf-concat-add*:
assumes $\text{res}: \text{inf-concat } n \ k = (i, j)$
and $j + m < n \ i$
shows $\text{inf-concat } n \ (k + m) = (i, j + m)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-step*:
assumes $\text{res}: \text{inf-concat } n \ k = (i, j)$
and $j: Suc \ (j + m) = n \ i$
shows $\text{inf-concat } n \ (k + Suc \ m) = (\text{LEAST } i'. i' > i \wedge 0 < n \ i', 0)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-surj-zero*:
assumes $0 < n$ i
shows $\exists k. \text{inf-concat } n \ k = (i, 0)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-surj*:
assumes $j < n$ i
shows $\exists k. \text{inf-concat } n \ k = (i, j)$
 $\langle \text{proof} \rangle$

lemma *inf-concat-mono*:
assumes $\text{inf}: \text{INFM } i. n \ i > 0$
and $\text{resk}: \text{inf-concat } n \ k = (i, j)$
and $\text{reskp}: \text{inf-concat } n \ k' = (i', j')$
and $\text{lt}: i < i'$
shows $k < k'$
 $\langle \text{proof} \rangle$

lemma *inf-concat-Suc*:
assumes $\text{inf}: \text{INFM } i. n \ i > 0$
and $f: \bigwedge i. f \ i \ (n \ i) = f \ (\text{Suc } i) \ 0$
and $\text{resk}: \text{inf-concat } n \ k = (i, j)$
and $\text{ressk}: \text{inf-concat } n \ (\text{Suc } k) = (i', j')$
shows $f \ i' \ j' = f \ i \ (\text{Suc } j)$
 $\langle \text{proof} \rangle$

end

2 Abstract Rewrite Systems

theory *Abstract-Rewriting*
imports
HOL-Library.Infinite-Set
Regular-Sets.Regexp-Method
Seq
begin

lemma *trancl-mono-set*:
 $r \subseteq s \implies r^+ \subseteq s^+$
 $\langle \text{proof} \rangle$

lemma *relpow-mono*:
fixes $r :: 'a \ \text{rel}$
assumes $r \subseteq r'$ **shows** $r \ \overset{\sim}{\sim} \ n \subseteq r' \ \overset{\sim}{\sim} \ n$
 $\langle \text{proof} \rangle$

lemma *refl-inv-image*:
 $\text{refl } R \implies \text{refl } (\text{inv-image } R \ f)$

<proof>

2.1 Definitions

Two elements are *joinable* (and then have in the joinability relation) w.r.t. A , iff they have a common reduct.

definition *join* :: 'a rel \Rightarrow 'a rel $((\downarrow)$ [1000] 999) **where**
 $A^\downarrow = A^* \circ (A^{-1})^*$

Two elements are *meetable* (and then have in the meetability relation) w.r.t. A , iff they have a common ancestor.

definition *meet* :: 'a rel \Rightarrow 'a rel $((\uparrow)$ [1000] 999) **where**
 $A^\uparrow = (A^{-1})^* \circ A^*$

The *symmetric closure* of a relation allows steps in both directions.

abbreviation *symcl* :: 'a rel \Rightarrow 'a rel $((\leftrightarrow)$ [1000] 999) **where**
 $A^{\leftrightarrow} \equiv A \cup A^{-1}$

A *conversion* is a (possibly empty) sequence of steps in the symmetric closure.

definition *conversion* :: 'a rel \Rightarrow 'a rel $((\leftrightarrow^*)$ [1000] 999) **where**
 $A^{\leftrightarrow^*} = (A^{\leftrightarrow})^*$

The set of *normal forms* of an ARS constitutes all the elements that do not have any successors.

definition *NF* :: 'a rel \Rightarrow 'a set **where**
 $NF\ A = \{a. A \text{ “ } \{a\} = \{\}\}$

definition *normalizability* :: 'a rel \Rightarrow 'a rel $((\downarrow)$ [1000] 999) **where**
 $A^\downarrow = \{(a, b). (a, b) \in A^* \wedge b \in NF\ A\}$

notation (*ASCII*)
symcl $((\widehat{\leftrightarrow})$ [1000] 999) **and**
conversion $((\widehat{\leftrightarrow}^*)$ [1000] 999) **and**
normalizability $((\downarrow)$ [1000] 999)

lemma *symcl-converse*:
 $(A^{\leftrightarrow})^{-1} = A^{\leftrightarrow}$ *<proof>*

lemma *symcl-Un*: $(A \cup B)^{\leftrightarrow} = A^{\leftrightarrow} \cup B^{\leftrightarrow}$ *<proof>*

lemma *no-step*:
assumes $A \text{ “ } \{a\} = \{\}$ **shows** $a \in NF\ A$
<proof>

lemma *joinI*:
 $(a, c) \in A^* \implies (b, c) \in A^* \implies (a, b) \in A^\downarrow$
<proof>

lemma *joinI-left*:

$(a, b) \in A^* \implies (a, b) \in A^\downarrow$
<proof>

lemma *joinI-right*: $(b, a) \in A^* \implies (a, b) \in A^\downarrow$

<proof>

lemma *joinE*:

assumes $(a, b) \in A^\downarrow$
obtains c **where** $(a, c) \in A^*$ **and** $(b, c) \in A^*$
<proof>

lemma *joinD*:

$(a, b) \in A^\downarrow \implies \exists c. (a, c) \in A^* \wedge (b, c) \in A^*$
<proof>

lemma *meetI*:

$(a, b) \in A^* \implies (a, c) \in A^* \implies (b, c) \in A^\uparrow$
<proof>

lemma *meetE*:

assumes $(b, c) \in A^\uparrow$
obtains a **where** $(a, b) \in A^*$ **and** $(a, c) \in A^*$
<proof>

lemma *meetD*: $(b, c) \in A^\uparrow \implies \exists a. (a, b) \in A^* \wedge (a, c) \in A^*$

<proof>

lemma *conversionI*: $(a, b) \in (A^{\leftrightarrow})^* \implies (a, b) \in A^{\leftrightarrow*}$

<proof>

lemma *conversion-refl* [*simp*]: $(a, a) \in A^{\leftrightarrow*}$

<proof>

lemma *conversionI'*:

assumes $(a, b) \in A^*$ **shows** $(a, b) \in A^{\leftrightarrow*}$
<proof>

lemma *rtrancl-comp-trancl-conv*:

$r^* \circ r = r^+$ *<proof>*

lemma *trancl-o-refl-is-trancl*:

$r^+ \circ r^= = r^+$ *<proof>*

lemma *conversionE*:

$(a, b) \in A^{\leftrightarrow*} \implies ((a, b) \in (A^{\leftrightarrow})^* \implies P) \implies P$
<proof>

Later declarations are tried first for ‘proof’ and ‘rule,’ then have the

“main” introduction / elimination rules for constants should be declared last.

declare *joinI-left* [*intro*]
declare *joinI-right* [*intro*]
declare *joinI* [*intro*]
declare *joinD* [*dest*]
declare *joinE* [*elim*]

declare *meetI* [*intro*]
declare *meetD* [*dest*]
declare *meetE* [*elim*]

declare *conversionI'* [*intro*]
declare *conversionI* [*intro*]
declare *conversionE* [*elim*]

lemma *conversion-trans*:
 $trans (A^{\leftrightarrow*})$
 $\langle proof \rangle$

lemma *conversion-sym*:
 $sym (A^{\leftrightarrow*})$
 $\langle proof \rangle$

lemma *conversion-inv*:
 $(x, y) \in R^{\leftrightarrow*} \longleftrightarrow (y, x) \in R^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *conversion-converse* [*simp*]:
 $(A^{\leftrightarrow*})^{-1} = A^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *conversion-rtrancl* [*simp*]:
 $(A^{\leftrightarrow*})^* = A^{\leftrightarrow*}$
 $\langle proof \rangle$

lemma *rtrancl-join-join*:
assumes $(a, b) \in A^*$ **and** $(b, c) \in A^\downarrow$ **shows** $(a, c) \in A^\downarrow$
 $\langle proof \rangle$

lemma *join-rtrancl-join*:
assumes $(a, b) \in A^\downarrow$ **and** $(c, b) \in A^*$ **shows** $(a, c) \in A^\downarrow$
 $\langle proof \rangle$

lemma *NF-I*: $(\bigwedge b. (a, b) \notin A) \implies a \in NF\ A$ $\langle proof \rangle$

lemma *NF-E*: $a \in NF\ A \implies ((a, b) \notin A \implies P) \implies P$ $\langle proof \rangle$

declare *NF-I* [*intro*]
declare *NF-E* [*elim*]

lemma *NF-no-step*: $a \in NF\ A \implies \forall b. (a, b) \notin A$ *<proof>*

lemma *NF-anti-mono*:
assumes $A \subseteq B$ **shows** $NF\ B \subseteq NF\ A$
<proof>

lemma *NF-iff-no-step*: $a \in NF\ A = (\forall b. (a, b) \notin A)$ *<proof>*

lemma *NF-no-trancl-step*:
assumes $a \in NF\ A$ **shows** $\forall b. (a, b) \notin A^+$
<proof>

lemma *NF-Id-on-fst-image* [*simp*]: $NF\ (Id\text{-on}\ (fst\ 'A)) = NF\ A$ *<proof>*

lemma *fst-image-NF-Id-on* [*simp*]: $fst\ 'R = Q \implies NF\ (Id\text{-on}\ Q) = NF\ R$ *<proof>*

lemma *NF-empty* [*simp*]: $NF\ \{\} = UNIV$ *<proof>*

lemma *normalizability-I*: $(a, b) \in A^* \implies b \in NF\ A \implies (a, b) \in A^!$
<proof>

lemma *normalizability-I'*: $(a, b) \in A^* \implies (b, c) \in A^! \implies (a, c) \in A^!$
<proof>

lemma *normalizability-E*: $(a, b) \in A^! \implies ((a, b) \in A^* \implies b \in NF\ A \implies P) \implies P$
<proof>

declare *normalizability-I'* [*intro*]
declare *normalizability-I* [*intro*]
declare *normalizability-E* [*elim*]

2.2 Properties of ARSs

The following properties on (elements of) ARSs are defined: completeness, Church-Rosser property, semi-completeness, strong normalization, unique normal forms, Weak Church-Rosser property, and weak normalization.

definition *CR-on* :: 'a rel \Rightarrow 'a set \Rightarrow bool **where**
 $CR\text{-on}\ r\ A \longleftrightarrow (\forall a \in A. \forall b\ c. (a, b) \in r^* \wedge (a, c) \in r^* \longrightarrow (b, c) \in join\ r)$

abbreviation *CR* :: 'a rel \Rightarrow bool **where**
 $CR\ r \equiv CR\text{-on}\ r\ UNIV$

definition *SN-on* :: 'a rel \Rightarrow 'a set \Rightarrow bool **where**
 $SN\text{-on}\ r\ A \longleftrightarrow \neg (\exists f. f\ 0 \in A \wedge chain\ r\ f)$

abbreviation $SN :: 'a \text{ rel} \Rightarrow \text{bool}$ **where**
 $SN \ r \equiv SN\text{-on } r \ UNIV$

Alternative definition of SN .

lemma $SN\text{-def}$: $SN \ r = (\forall x. SN\text{-on } r \ \{x\})$
 $\langle \text{proof} \rangle$

definition $UNF\text{-on} :: 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $UNF\text{-on } r \ A \longleftrightarrow (\forall a \in A. \forall b \ c. (a, b) \in r^! \wedge (a, c) \in r^! \longrightarrow b = c)$

abbreviation $UNF :: 'a \text{ rel} \Rightarrow \text{bool}$ **where** $UNF \ r \equiv UNF\text{-on } r \ UNIV$

definition $WCR\text{-on} :: 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $WCR\text{-on } r \ A \longleftrightarrow (\forall a \in A. \forall b \ c. (a, b) \in r \wedge (a, c) \in r \longrightarrow (b, c) \in \text{join } r)$

abbreviation $WCR :: 'a \text{ rel} \Rightarrow \text{bool}$ **where** $WCR \ r \equiv WCR\text{-on } r \ UNIV$

definition $WN\text{-on} :: 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $WN\text{-on } r \ A \longleftrightarrow (\forall a \in A. \exists b. (a, b) \in r^!)$

abbreviation $WN :: 'a \text{ rel} \Rightarrow \text{bool}$ **where**
 $WN \ r \equiv WN\text{-on } r \ UNIV$

lemmas $CR\text{-defs} = CR\text{-on-def}$
lemmas $SN\text{-defs} = SN\text{-on-def}$
lemmas $UNF\text{-defs} = UNF\text{-on-def}$
lemmas $WCR\text{-defs} = WCR\text{-on-def}$
lemmas $WN\text{-defs} = WN\text{-on-def}$

definition $complete\text{-on} :: 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $complete\text{-on } r \ A \longleftrightarrow SN\text{-on } r \ A \wedge CR\text{-on } r \ A$

abbreviation $complete :: 'a \text{ rel} \Rightarrow \text{bool}$ **where**
 $complete \ r \equiv complete\text{-on } r \ UNIV$

definition $semi\text{-complete-on} :: 'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **where**
 $semi\text{-complete-on } r \ A \longleftrightarrow WN\text{-on } r \ A \wedge CR\text{-on } r \ A$

abbreviation $semi\text{-complete} :: 'a \text{ rel} \Rightarrow \text{bool}$ **where**
 $semi\text{-complete } r \equiv semi\text{-complete-on } r \ UNIV$

lemmas $complete\text{-defs} = complete\text{-on-def}$
lemmas $semi\text{-complete-defs} = semi\text{-complete-on-def}$

Unique normal forms with respect to conversion.

definition $UNC :: 'a \text{ rel} \Rightarrow \text{bool}$ **where**
 $UNC \ A \longleftrightarrow (\forall a \ b. a \in NF \ A \wedge b \in NF \ A \wedge (a, b) \in A^{\leftrightarrow*} \longrightarrow a = b)$

lemma $complete\text{-onI}$:

$SN\text{-on } r A \implies CR\text{-on } r A \implies complete\text{-on } r A$
<proof>

lemma *complete-onE*:

$complete\text{-on } r A \implies (SN\text{-on } r A \implies CR\text{-on } r A \implies P) \implies P$
<proof>

lemma *CR-onI*:

$(\bigwedge a b c. a \in A \implies (a, b) \in r^* \implies (a, c) \in r^* \implies (b, c) \in join\ r) \implies CR\text{-on } r A$
<proof>

lemma *CR-on-singletonI*:

$(\bigwedge b c. (a, b) \in r^* \implies (a, c) \in r^* \implies (b, c) \in join\ r) \implies CR\text{-on } r \{a\}$
<proof>

lemma *CR-onE*:

$CR\text{-on } r A \implies a \in A \implies ((b, c) \in join\ r \implies P) \implies ((a, b) \notin r^* \implies P) \implies ((a, c) \notin r^* \implies P) \implies P$
<proof>

lemma *CR-onD*:

$CR\text{-on } r A \implies a \in A \implies (a, b) \in r^* \implies (a, c) \in r^* \implies (b, c) \in join\ r$
<proof>

lemma *semi-complete-onI*: $WN\text{-on } r A \implies CR\text{-on } r A \implies semi\text{-complete-on } r A$
<proof>

lemma *semi-complete-onE*:

$semi\text{-complete-on } r A \implies (WN\text{-on } r A \implies CR\text{-on } r A \implies P) \implies P$
<proof>

declare *semi-complete-onI* [*intro*]

declare *semi-complete-onE* [*elim*]

declare *complete-onI* [*intro*]

declare *complete-onE* [*elim*]

declare *CR-onI* [*intro*]

declare *CR-on-singletonI* [*intro*]

declare *CR-onD* [*dest*]

declare *CR-onE* [*elim*]

lemma *UNC-I*:

$(\bigwedge a b. a \in NF\ A \implies b \in NF\ A \implies (a, b) \in A^{\leftrightarrow*} \implies a = b) \implies UNC\ A$
<proof>

lemma *UNC-E*:

$\llbracket \text{UNC } A; a = b \implies P; a \notin \text{NF } A \implies P; b \notin \text{NF } A \implies P; (a, b) \notin A^{\leftrightarrow*} \implies P \rrbracket \implies P$
 <proof>

lemma *UNF-onI*: $(\bigwedge a b c. a \in A \implies (a, b) \in r^! \implies (a, c) \in r^! \implies b = c) \implies \text{UNF-on } r A$
 <proof>

lemma *UNF-onE*:
 $\text{UNF-on } r A \implies a \in A \implies (b = c \implies P) \implies ((a, b) \notin r^! \implies P) \implies ((a, c) \notin r^! \implies P) \implies P$
 <proof>

lemma *UNF-onD*:
 $\text{UNF-on } r A \implies a \in A \implies (a, b) \in r^! \implies (a, c) \in r^! \implies b = c$
 <proof>

declare *UNF-onI* [*intro*]
declare *UNF-onD* [*dest*]
declare *UNF-onE* [*elim*]

lemma *SN-onI*:
assumes $\bigwedge f. \llbracket f \ 0 \in A; \text{chain } r f \rrbracket \implies \text{False}$
shows *SN-on* $r A$
 <proof>

lemma *SN-I*: $(\bigwedge a. \text{SN-on } A \ \{a\}) \implies \text{SN } A$
 <proof>

lemma *SN-on-transl-imp-SN-on*:
assumes *SN-on* $(R^+) T$ **shows** *SN-on* $R T$
 <proof>

lemma *SN-onE*:
assumes *SN-on* $r A$
and $\neg (\exists f. f \ 0 \in A \wedge \text{chain } r f) \implies P$
shows P
 <proof>

lemma *not-SN-onE*:
assumes $\neg \text{SN-on } r A$
and $\bigwedge f. \llbracket f \ 0 \in A; \text{chain } r f \rrbracket \implies P$
shows P
 <proof>

declare *SN-onI* [*intro*]
declare *SN-onE* [*elim*]
declare *not-SN-onE* [*Pure.elim*, *elim*]

lemma *refl-not-SN*: $(x, x) \in R \implies \neg SN R$
 ⟨*proof*⟩

lemma *SN-on-irrefl*:
assumes *SN-on* $r A$
shows $\forall a \in A. (a, a) \notin r$
 ⟨*proof*⟩

lemma *WCR-onI*: $(\bigwedge a b c. a \in A \implies (a, b) \in r \implies (a, c) \in r \implies (b, c) \in join r) \implies WCR-on r A$
 ⟨*proof*⟩

lemma *WCR-onE*:
 $WCR-on r A \implies a \in A \implies ((b, c) \in join r \implies P) \implies ((a, b) \notin r \implies P) \implies ((a, c) \notin r \implies P) \implies P$
 ⟨*proof*⟩

lemma *SN-nat-bounded*: $SN \{(x, y :: nat). x < y \wedge y \leq b\}$ (**is** *SN ?R*)
 ⟨*proof*⟩

lemma *WCR-onD*:
 $WCR-on r A \implies a \in A \implies (a, b) \in r \implies (a, c) \in r \implies (b, c) \in join r$
 ⟨*proof*⟩

lemma *WN-onI*: $(\bigwedge a. a \in A \implies \exists b. (a, b) \in r^!) \implies WN-on r A$
 ⟨*proof*⟩

lemma *WN-onE*: $WN-on r A \implies a \in A \implies (\bigwedge b. (a, b) \in r^! \implies P) \implies P$
 ⟨*proof*⟩

lemma *WN-onD*: $WN-on r A \implies a \in A \implies \exists b. (a, b) \in r^!$
 ⟨*proof*⟩

declare *WCR-onI* [*intro*]
declare *WCR-onD* [*dest*]
declare *WCR-onE* [*elim*]

declare *WN-onI* [*intro*]
declare *WN-onD* [*dest*]
declare *WN-onE* [*elim*]

Restricting a relation r to those elements that are strongly normalizing with respect to a relation s .

definition *restrict-SN* :: $'a rel \Rightarrow 'a rel \Rightarrow 'a rel$ **where**
 $restrict-SN r s = \{(a, b) \mid a b. (a, b) \in r \wedge SN-on s \{a\}\}$

lemma *SN-restrict-SN-idemp* [*simp*]: $SN (restrict-SN A A)$
 ⟨*proof*⟩

lemma *SN-on-Image*:
assumes *SN-on* r A
shows *SN-on* r (r “ A)
 \langle *proof* \rangle

lemma *SN-on-subset2*:
assumes $A \subseteq B$ **and** *SN-on* r B
shows *SN-on* r A
 \langle *proof* \rangle

lemma *step-preserves-SN-on*:
assumes 1: $(a, b) \in r$
and 2: *SN-on* r $\{a\}$
shows *SN-on* r $\{b\}$
 \langle *proof* \rangle

lemma *steps-preserve-SN-on*: $(a, b) \in A^* \implies \text{SN-on } A \{a\} \implies \text{SN-on } A \{b\}$
 \langle *proof* \rangle

lemma *relpow-seq*:
assumes $(x, y) \in r^{\sim n}$
shows $\exists f. f\ 0 = x \wedge f\ n = y \wedge (\forall i < n. (f\ i, f\ (Suc\ i)) \in r)$
 \langle *proof* \rangle

lemma *rtrancl-imp-seq*:
assumes $(x, y) \in r^*$
shows $\exists f\ n. f\ 0 = x \wedge f\ n = y \wedge (\forall i < n. (f\ i, f\ (Suc\ i)) \in r)$
 \langle *proof* \rangle

lemma *SN-on-Image-rtrancl*:
assumes *SN-on* r A
shows *SN-on* r (r^* “ A)
 \langle *proof* \rangle

declare *subrelI* [*Pure.intro*]

lemma *restrict-SN-trancl-simp* [*simp*]: $(\text{restrict-SN } A\ A)^+ = \text{restrict-SN } (A^+) A$
(is $?lhs = ?rhs$ **)**
 \langle *proof* \rangle

lemma *SN-imp-WN*:
assumes *SN* A **shows** *WN* A
 \langle *proof* \rangle

lemma *UNC-imp-UNF*:
assumes *UNC* r **shows** *UNF* r
 \langle *proof* \rangle

lemma *join-NF-imp-eq*:

assumes $(x, y) \in r^\downarrow$ **and** $x \in NF\ r$ **and** $y \in NF\ r$
shows $x = y$

<proof>

lemma *rtrancl-Restr*:

assumes $(x, y) \in (Restr\ r\ A)^*$
shows $(x, y) \in r^*$

<proof>

lemma *join-mono*:

assumes $r \subseteq s$
shows $r^\downarrow \subseteq s^\downarrow$

<proof>

lemma *CR-iff-meet-subset-join*: $CR\ r = (r^\uparrow \subseteq r^\downarrow)$

<proof>

lemma *CR-divergence-imp-join*:

assumes $CR\ r$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
shows $(y, z) \in r^\downarrow$

<proof>

lemma *join-imp-conversion*: $r^\downarrow \subseteq r^{\leftrightarrow*}$

<proof>

lemma *meet-imp-conversion*: $r^\uparrow \subseteq r^{\leftrightarrow*}$

<proof>

lemma *CR-imp-UNF*:

assumes $CR\ r$ **shows** $UNF\ r$

<proof>

lemma *CR-iff-conversion-imp-join*: $CR\ r = (r^{\leftrightarrow*} \subseteq r^\downarrow)$

<proof>

lemma *CR-imp-conversionIff-join*:

assumes $CR\ r$ **shows** $r^{\leftrightarrow*} = r^\downarrow$

<proof>

lemma *sym-join*: $sym\ (join\ r)$ *<proof>*

lemma *join-sym*: $(s, t) \in A^\downarrow \implies (t, s) \in A^\downarrow$ *<proof>*

lemma *CR-join-left-I*:

assumes $CR\ r$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^\downarrow$ **shows** $(y, z) \in r^\downarrow$

<proof>

lemma *CR-join-right-I*:

assumes *CR* r **and** $(x, y) \in r^\downarrow$ **and** $(y, z) \in r^*$ **shows** $(x, z) \in r^\downarrow$
<proof>

lemma *NF-not-suc*:

assumes $(x, y) \in r^*$ **and** $x \in NF\ r$ **shows** $x = y$
<proof>

lemma *semi-complete-imp-conversionIff-same-NF*:

assumes *semi-complete* r
shows $((x, y) \in r^{\leftrightarrow*}) = (\forall u\ v. (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v)$
<proof>

lemma *CR-imp-UNC*:

assumes *CR* r **shows** *UNC* r
<proof>

lemma *WN-UNF-imp-CR*:

assumes *WN* r **and** *UNF* r **shows** *CR* r
<proof>

definition *diamond* :: 'a rel \Rightarrow bool (\diamond) **where**

$\diamond\ r \iff (r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1})$

lemma *diamond-I* [*intro*]: $(r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \implies \diamond\ r$ *<proof>*

lemma *diamond-E* [*elim*]: $\diamond\ r \implies ((r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \implies P) \implies P$
<proof>

lemma *diamond-imp-semi-confluence*:

assumes $\diamond\ r$ **shows** $(r^{-1}\ O\ r^*) \subseteq r^\downarrow$
<proof>

lemma *semi-confluence-imp-CR*:

assumes $(r^{-1}\ O\ r^*) \subseteq r^\downarrow$ **shows** *CR* r
<proof>

lemma *diamond-imp-CR*:

assumes $\diamond\ r$ **shows** *CR* r
<proof>

lemma *diamond-imp-CR'*:

assumes $\diamond\ s$ **and** $r \subseteq s$ **and** $s \subseteq r^*$ **shows** *CR* r
<proof>

lemma *SN-imp-minimal*:

assumes *SN* A
shows $\forall Q\ x. x \in Q \longrightarrow (\exists z \in Q. \forall y. (z, y) \in A \longrightarrow y \notin Q)$

<proof>

lemma *SN-on-imp-on-minimal*:

assumes *SN-on* $r \{x\}$

shows $\forall Q. x \in Q \longrightarrow (\exists z \in Q. \forall y. (z, y) \in r \longrightarrow y \notin Q)$

<proof>

lemma *minimal-imp-wf*:

assumes $\forall Q x. x \in Q \longrightarrow (\exists z \in Q. \forall y. (z, y) \in r \longrightarrow y \notin Q)$

shows $wf(r^{-1})$

<proof>

lemmas *SN-imp-wf = SN-imp-minimal [THEN minimal-imp-wf]*

lemma *wf-imp-SN*:

assumes $wf(A^{-1})$ **shows** *SN* A

<proof>

lemma *SN-nat-gt*: *SN* $\{(a, b :: nat) . a > b\}$

<proof>

lemma *SN-iff-wf*: *SN* $A = wf(A^{-1})$ *<proof>*

lemma *SN-imp-acyclic*: *SN* $R \implies acyclic R$

<proof>

lemma *SN-induct*:

assumes *sn*: *SN* r **and** *step*: $\bigwedge a. (\bigwedge b. (a, b) \in r \implies P b) \implies P a$

shows $P a$

<proof>

lemmas *SN-induct-rule = SN-induct [consumes 1, case-names IH, induct pred: SN]*

lemma *SN-on-induct [consumes 2, case-names IH, induct pred: SN-on]*:

assumes *SN*: *SN-on* $R A$

and $s \in A$

and *imp*: $\bigwedge t. (\bigwedge u. (t, u) \in R \implies P u) \implies P t$

shows $P s$

<proof>

lemma *accp-imp-SN-on*:

assumes $\bigwedge x. x \in A \implies Wellfounded.accp g x$

shows *SN-on* $\{(y, z). g z y\} A$

<proof>

lemma *SN-on-imp-accp*:
assumes *SN-on* $\{(y, z). g z y\} A$
shows $\forall x \in A. \text{Wellfounded. accp } g x$
 $\langle \text{proof} \rangle$

lemma *SN-on-conv-accp*:
 $\text{SN-on } \{(y, z). g z y\} \{x\} = \text{Wellfounded. accp } g x$
 $\langle \text{proof} \rangle$

lemma *SN-on-conv-acc*: $\text{SN-on } \{(y, z). (z, y) \in r\} \{x\} \longleftrightarrow x \in \text{Wellfounded. acc } r$
 $\langle \text{proof} \rangle$

lemma *acc-imp-SN-on*:
assumes $x \in \text{Wellfounded. acc } r$ **shows** $\text{SN-on } \{(y, z). (z, y) \in r\} \{x\}$
 $\langle \text{proof} \rangle$

lemma *SN-on-imp-acc*:
assumes $\text{SN-on } \{(y, z). (z, y) \in r\} \{x\}$ **shows** $x \in \text{Wellfounded. acc } r$
 $\langle \text{proof} \rangle$

2.3 Newman's Lemma

lemma *rtrancl-len-E [elim]*:
assumes $(x, y) \in r^*$ **obtains** n **where** $(x, y) \in r^{\sim n}$
 $\langle \text{proof} \rangle$

lemma *relpow-Suc-E2' [elim]*:
assumes $(x, z) \in A^{\sim \text{Suc } n}$ **obtains** y **where** $(x, y) \in A$ **and** $(y, z) \in A^*$
 $\langle \text{proof} \rangle$

lemmas *SN-on-induct' [consumes 1, case-names IH] = SN-on-induct [OF - singletonI]*

lemma *Newman-local*:
assumes $\text{SN-on } r X$ **and** *WCR*: $\text{WCR-on } r \{x. \text{SN-on } r \{x\}\}$
shows $\text{CR-on } r X$
 $\langle \text{proof} \rangle$

lemma *Newman*: $\text{SN } r \implies \text{WCR } r \implies \text{CR } r$
 $\langle \text{proof} \rangle$

lemma *Image-SN-on*:
assumes $\text{SN-on } r (r \text{ `` } A)$
shows $\text{SN-on } r A$
 $\langle \text{proof} \rangle$

lemma *SN-on-Image-conv*: $\text{SN-on } r (r \text{ `` } A) = \text{SN-on } r A$
 $\langle \text{proof} \rangle$

If all successors are terminating, then the current element is also terminating.

lemma *step-reflects-SN-on*:

assumes $(\bigwedge b. (a, b) \in r \implies \text{SN-on } r \{b\})$

shows $\text{SN-on } r \{a\}$

<proof>

lemma *SN-on-all-reducts-SN-on-conv*:

$\text{SN-on } r \{a\} = (\forall b. (a, b) \in r \longrightarrow \text{SN-on } r \{b\})$

<proof>

lemma *SN-imp-SN-trancl*: $\text{SN } R \implies \text{SN } (R^+)$

<proof>

lemma *SN-trancl-imp-SN*:

assumes $\text{SN } (R^+)$ **shows** $\text{SN } R$

<proof>

lemma *SN-trancl-SN-conv*: $\text{SN } (R^+) = \text{SN } R$

<proof>

lemma *SN-inv-image*: $\text{SN } R \implies \text{SN } (\text{inv-image } R \ f)$ *<proof>*

lemma *SN-subset*: $\text{SN } R \implies R' \subseteq R \implies \text{SN } R'$ *<proof>*

lemma *SN-pow-imp-SN*:

assumes $\text{SN } (A \text{ } \widetilde{\text{Suc}} \ n)$ **shows** $\text{SN } A$

<proof>

lemma *pow-Suc-subset-trancl*: $R \text{ } \widetilde{\text{Suc}} \ n \subseteq R^+$

<proof>

lemma *SN-imp-SN-pow*:

assumes $\text{SN } R$ **shows** $\text{SN } (R \text{ } \widetilde{\text{Suc}} \ n)$

<proof>

lemma *SN-pow*: $\text{SN } R \longleftrightarrow \text{SN } (R \text{ } \widetilde{\text{Suc}} \ n)$

<proof>

lemma *SN-on-trancl*:

assumes $\text{SN-on } r \ A$ **shows** $\text{SN-on } (r^+) \ A$

<proof>

lemma *SN-on-trancl-SN-on-conv*: $\text{SN-on } (R^+) \ T = \text{SN-on } R \ T$

<proof>

Restrict an ARS to elements of a given set.

definition *restrict* :: 'a rel \Rightarrow 'a set \Rightarrow 'a rel **where**
restrict r $S = \{(x, y). x \in S \wedge y \in S \wedge (x, y) \in r\}$

lemma *SN-on-restrict*:
assumes *SN-on* r A
shows *SN-on* (*restrict* r S) A (**is** *SN-on* $?r$ A)
 \langle *proof* \rangle

lemma *restrict-rtrancl*: (*restrict* r S) $^* \subseteq r^*$ (**is** $?r^* \subseteq r^*$)
 \langle *proof* \rangle

lemma *rtrancl-Image-step*:
assumes $a \in r^*$ “ A
and $(a, b) \in r^*$
shows $b \in r^*$ “ A
 \langle *proof* \rangle

lemma *WCR-SN-on-imp-CR-on*:
assumes *WCR* r **and** *SN-on* r A **shows** *CR-on* r A
 \langle *proof* \rangle

lemma *SN-on-Image-normalizable*:
assumes *SN-on* r A
shows $\forall a \in A. \exists b. b \in r^!$ “ A
 \langle *proof* \rangle

lemma *SN-on-imp-normalizability*:
assumes *SN-on* r $\{a\}$ **shows** $\exists b. (a, b) \in r^!$
 \langle *proof* \rangle

2.4 Commutation

definition *commute* :: 'a rel \Rightarrow 'a rel \Rightarrow bool **where**
commute r $s \iff ((r^{-1})^* \circ s^*) \subseteq (s^* \circ (r^{-1})^*)$

lemma *CR-iff-self-commute*: *CR* $r =$ *commute* r r
 \langle *proof* \rangle

lemma *rtrancl-imp-rtrancl-UN*:
assumes $(x, y) \in r^*$ **and** $r \in I$
shows $(x, y) \in (\bigcup r \in I. r)^*$ (**is** $(x, y) \in ?r^*$)
 \langle *proof* \rangle

definition *quasi-commute* :: 'a rel \Rightarrow 'a rel \Rightarrow bool **where**
quasi-commute r $s \iff (s \circ r) \subseteq r \circ (r \cup s)^*$

lemma *rtrancl-union-subset-rtrancl-union-trancl*: $(r \cup s^+)^* = (r \cup s)^*$
 \langle *proof* \rangle

lemma *qc-imp-qc-trancl*:
assumes *quasi-commute r s* **shows** *quasi-commute r (s⁺)*
 ⟨*proof*⟩

lemma *steps-reflect-SN-on*:
assumes \neg *SN-on r {b}* **and** $(a, b) \in r^*$
shows \neg *SN-on r {a}*
 ⟨*proof*⟩

lemma *chain-imp-not-SN-on*:
assumes *chain r f*
shows \neg *SN-on r {f i}*
 ⟨*proof*⟩

lemma *quasi-commute-imp-SN*:
assumes *SN r* **and** *SN s* **and** *quasi-commute r s*
shows *SN (r \cup s)*
 ⟨*proof*⟩

2.5 Strong Normalization

lemma *non-strict-into-strict*:
assumes *compat: NS O S \subseteq S*
and *steps: (s, t) \in (NS*) O S*
shows $(s, t) \in S$
 ⟨*proof*⟩

lemma *comp-trancl*:
assumes *R O S \subseteq S* **shows** *R O S⁺ \subseteq S⁺*
 ⟨*proof*⟩

lemma *comp-rtrancl-trancl*:
assumes *comp: R O S \subseteq S*
and *seq: (s, t) \in (R \cup S)* O S*
shows $(s, t) \in S^+$
 ⟨*proof*⟩

lemma *trancl-union-right*: $r^+ \subseteq (s \cup r)^+$
 ⟨*proof*⟩

lemma *restrict-SN-subset*: *restrict-SN R S \subseteq R*
 ⟨*proof*⟩

lemma *chain-Un-SN-on-imp-first-step*:
assumes *chain (R \cup S) t* **and** *SN-on S {t 0}*
shows $\exists i. (t\ i, t\ (Suc\ i)) \in R \wedge (\forall j < i. (t\ j, t\ (Suc\ j)) \in S \wedge (t\ j, t\ (Suc\ j)) \notin R)$
 ⟨*proof*⟩

lemma *first-step*:

assumes C : $C = A \cup B$ **and** *steps*: $(x, y) \in C^*$ **and** *Bstep*: $(y, z) \in B$
shows $\exists y. (x, y) \in A^* \circ B$
<proof>

lemma *first-step-O*:

assumes C : $C = A \cup B$ **and** *steps*: $(x, y) \in C^* \circ B$
shows $\exists y. (x, y) \in A^* \circ B$
<proof>

lemma *firstStep*:

assumes *LSR*: $L = S \cup R$ **and** *xyL*: $(x, y) \in L^*$
shows $(x, y) \in R^* \vee (x, y) \in R^* \circ S \circ L^*$
<proof>

lemma *non-strict-ending*:

assumes *chain*: $\text{chain } (R \cup S) \ t$
and *comp*: $R \circ S \subseteq S$
and *SN*: *SN-on* $S \ \{t \ 0\}$
shows $\exists j. \forall i \geq j. (t \ i, t \ (\text{Suc } i)) \in R - S$
<proof>

lemma *SN-on-subset1*:

assumes *SN-on* $r \ A$ **and** $s \subseteq r$
shows *SN-on* $s \ A$
<proof>

lemmas *SN-on-mono* = *SN-on-subset1*

lemma *rtrancl-fun-conv*:

$((s, t) \in R^*) = (\exists f \ n. f \ 0 = s \wedge f \ n = t \wedge (\forall i < n. (f \ i, f \ (\text{Suc } i)) \in R))$
<proof>

lemma *compat-tr-compat*:

assumes $NS \circ S \subseteq S$ **shows** $NS^* \circ S \subseteq S$
<proof>

lemma *right-comp-S* [*simp*]:

assumes $(x, y) \in S \circ (S \circ S^* \circ NS^* \cup NS^*)$
shows $(x, y) \in (S \circ S^* \circ NS^*)$
<proof>

lemma *compatible-SN*:

assumes *SN*: *SN* S
and *compat*: $NS \circ S \subseteq S$
shows *SN* $(S \circ S^* \circ NS^*)$ (**is** *SN* $?A$)
<proof>

lemma compatible-rtrancl-split:
assumes *compat*: $NS \ O \ S \subseteq S$
and steps: $(x, y) \in (NS \cup S)^*$
shows $(x, y) \in S \ O \ S^* \ O \ NS^* \cup NS^*$
 $\langle proof \rangle$

lemma compatible-conv:
assumes *compat*: $NS \ O \ S \subseteq S$
shows $(NS \cup S)^* \ O \ S \ O \ (NS \cup S)^* = S \ O \ S^* \ O \ NS^*$
 $\langle proof \rangle$

lemma compatible-SN':
assumes *compat*: $NS \ O \ S \subseteq S$ **and** *SN*: $SN \ S$
shows $SN((NS \cup S)^* \ O \ S \ O \ (NS \cup S)^*)$
 $\langle proof \rangle$

lemma rtrancl-diff-decomp:
assumes $(x, y) \in A^* - B^*$
shows $(x, y) \in A^* \ O \ (A - B) \ O \ A^*$
 $\langle proof \rangle$

lemma SN-empty [simp]: $SN \ \{\}$ $\langle proof \rangle$

lemma SN-on-weakening:
assumes *SN-on* $R1 \ A$
shows *SN-on* $(R1 \cap R2) \ A$
 $\langle proof \rangle$

definition ideriv :: $'a \ rel \Rightarrow 'a \ rel \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$ **where**
 $ideriv \ R \ S \ as \longleftrightarrow (\forall i. (as \ i, as \ (Suc \ i)) \in R \cup S) \wedge (INFM \ i. (as \ i, as \ (Suc \ i)) \in R)$

lemma ideriv-mono: $R \subseteq R' \Longrightarrow S \subseteq S' \Longrightarrow ideriv \ R \ S \ as \Longrightarrow ideriv \ R' \ S' \ as$
 $\langle proof \rangle$

fun
 $shift :: (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow nat \Rightarrow 'a$
where
 $shift \ f \ j = (\lambda \ i. f \ (i+j))$

lemma ideriv-split:
assumes *ideriv*: $ideriv \ R \ S \ as$
and *nideriv*: $\neg ideriv \ (D \cap (R \cup S)) \ (R \cup S - D) \ as$
shows $\exists i. ideriv \ (R - D) \ (S - D) \ (shift \ as \ i)$
 $\langle proof \rangle$

lemma ideriv-SN:

assumes $SN: SN\ S$
and $compat: NS\ O\ S \subseteq S$
and $R: R \subseteq NS \cup S$
shows $\neg\ ideriv\ (S \cap R)\ (R - S)\ as$
 $\langle proof \rangle$

lemma $Infm-shift: (INFM\ i.\ P\ (shift\ f\ n\ i)) = (INFM\ i.\ P\ (f\ i))$ (**is** $?S = ?O$)
 $\langle proof \rangle$

lemma $rtrancl-list-conv:$
 $(s, t) \in R^* \longleftrightarrow$
 $(\exists\ ts.\ last\ (s\ \#\ ts) = t \wedge (\forall\ i < length\ ts.\ ((s\ \#\ ts)\ !\ i,\ (s\ \#\ ts)\ !\ Suc\ i) \in R))$
(is $?l = ?r$)
 $\langle proof \rangle$

lemma $SN-reaches-NF:$
assumes $SN-on\ r\ \{x\}$
shows $\exists\ y.\ (x, y) \in r^* \wedge y \in NF\ r$
 $\langle proof \rangle$

lemma $SN-WCR-reaches-NF:$
assumes $SN: SN-on\ r\ \{x\}$
and $WCR: WCR-on\ r\ \{x.\ SN-on\ r\ \{x\}\}$
shows $\exists!\ y.\ (x, y) \in r^* \wedge y \in NF\ r$
 $\langle proof \rangle$

definition $some-NF :: 'a\ rel \Rightarrow 'a \Rightarrow 'a$ **where**
 $some-NF\ r\ x = (SOME\ y.\ (x, y) \in r^* \wedge y \in NF\ r)$

lemma $some-NF:$
assumes $SN: SN-on\ r\ \{x\}$
shows $(x, some-NF\ r\ x) \in r^* \wedge some-NF\ r\ x \in NF\ r$
 $\langle proof \rangle$

lemma $some-NF-WCR:$
assumes $SN: SN-on\ r\ \{x\}$
and $WCR: WCR-on\ r\ \{x.\ SN-on\ r\ \{x\}\}$
and $steps: (x, y) \in r^*$
and $NF: y \in NF\ r$
shows $y = some-NF\ r\ x$
 $\langle proof \rangle$

lemma $some-NF-UNF:$
assumes $UNF: UNF\ r$
and $steps: (x, y) \in r^*$
and $NF: y \in NF\ r$
shows $y = some-NF\ r\ x$
 $\langle proof \rangle$

definition *the-NF* $A a = (THE\ b.\ (a, b) \in A^!)$

context

fixes A

assumes $SN: SN\ A$ and $CR: CR\ A$

begin

lemma *the-NF*: $(a, the-NF\ A\ a) \in A^!$

<proof>

lemma *the-NF-NF*: $the-NF\ A\ a \in NF\ A$

<proof>

lemma *the-NF-step*:

assumes $(a, b) \in A$

shows $the-NF\ A\ a = the-NF\ A\ b$

<proof>

lemma *the-NF-steps*:

assumes $(a, b) \in A^*$

shows $the-NF\ A\ a = the-NF\ A\ b$

<proof>

lemma *the-NF-conv*:

assumes $(a, b) \in A^{\leftrightarrow*}$

shows $the-NF\ A\ a = the-NF\ A\ b$

<proof>

end

definition *weak-diamond* $:: 'a\ rel \Rightarrow bool\ (w\Diamond)$ **where**

$w\Diamond\ r \iff (r^{-1}\ O\ r) - Id \subseteq (r\ O\ r^{-1})$

lemma *weak-diamond-imp-CR*:

assumes $wd: w\Diamond\ r$

shows $CR\ r$

<proof>

lemma *steps-imp-not-SN-on*:

fixes $t :: 'a \Rightarrow 'b$

and $R :: 'b\ rel$

assumes $steps: \bigwedge x. (t\ x, t\ (f\ x)) \in R$

shows $\neg SN-on\ R\ \{t\ x\}$

<proof>

lemma *steps-imp-not-SN*:

fixes $t :: 'a \Rightarrow 'b$

and $R :: 'b\ rel$

assumes $steps: \bigwedge x. (t\ x, t\ (f\ x)) \in R$

shows $\neg SN R$
 $\langle proof \rangle$

lemma *steps-map*:

assumes $fg: \bigwedge t u R . P t \implies Q R \implies (t, u) \in R \implies P u \wedge (f t, f u) \in g R$
and $t: P t$
and $R: Q R$
and $S: Q S$
shows $((t, u) \in R^* \implies (f t, f u) \in (g R)^*)$
 $\wedge ((t, u) \in R^* O S O R^* \implies (f t, f u) \in (g R)^* O (g S) O (g R)^*)$
 $\langle proof \rangle$

2.6 Terminating part of a relation

inductive-set

$SN\text{-part} :: 'a \text{ rel} \Rightarrow 'a \text{ set}$
for $r :: 'a \text{ rel}$

where

$SN\text{-part} I: (\bigwedge y. (x, y) \in r \implies y \in SN\text{-part } r) \implies x \in SN\text{-part } r$

The accessible part of a relation is the same as the terminating part (just two names for the same definition – modulo argument order). See $(\bigwedge y. (y, ?x) \in ?r \implies y \in Wellfounded.\text{acc } ?r) \implies ?x \in Wellfounded.\text{acc } ?r$.

Characterization of *SN-on* via terminating part.

lemma *SN-on-SN-part-conv*:

$SN\text{-on } r A \iff A \subseteq SN\text{-part } r$
 $\langle proof \rangle$

Special case for “full” termination.

lemma *SN-SN-part-UNIV-conv*:

$SN r \iff SN\text{-part } r = UNIV$
 $\langle proof \rangle$

lemma *closed-imp-rtrancl-closed*: **assumes** $L: L \subseteq A$

and $R: R \text{ “ } A \subseteq A$

shows $\{t \mid s. s \in L \wedge (s, t) \in R^*\} \subseteq A$
 $\langle proof \rangle$

lemma *trancl-steps-relpow*: **assumes** $a \subseteq b^+$

shows $(x, y) \in a^{\sim n} \implies \exists m. m \geq n \wedge (x, y) \in b^{\sim m}$
 $\langle proof \rangle$

lemma *relpow-image*: **assumes** $f: \bigwedge s t. (s, t) \in r \implies (f s, f t) \in r'$

shows $(s, t) \in r^{\sim n} \implies (f s, f t) \in r'^{\sim n}$
 $\langle proof \rangle$

lemma *relpow-refl-mono*:

assumes $refl: \bigwedge x. (x, x) \in Rel$

shows $m \leq n \implies (a, b) \in Rel^{\sim m} \implies (a, b) \in Rel^{\sim n}$

<proof>

lemma *SN-on-induct-acc-style* [*consumes 1, case-names IH*]:

assumes *sn*: *SN-on* $R \{a\}$

and *IH*: $\bigwedge x. \text{SN-on } R \{x\} \implies \llbracket \bigwedge y. (x, y) \in R \implies P y \rrbracket \implies P x$

shows $P a$

<proof>

lemma *partially-localize-CR*:

$CR r \iff (\forall x y z. (x, y) \in r \wedge (x, z) \in r^* \longrightarrow (y, z) \in \text{join } r)$

<proof>

definition *strongly-confluent-on* :: $'a \text{ rel} \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$

where

strongly-confluent-on $r A \iff$

$(\forall x \in A. \forall y z. (x, y) \in r \wedge (x, z) \in r \longrightarrow (\exists u. (y, u) \in r^* \wedge (z, u) \in r^=))$

abbreviation *strongly-confluent* :: $'a \text{ rel} \Rightarrow \text{bool}$

where

strongly-confluent $r \equiv \text{strongly-confluent-on } r \text{ UNIV}$

lemma *strongly-confluent-on-E11*:

strongly-confluent-on $r A \implies x \in A \implies (x, y) \in r \implies (x, z) \in r \implies$

$\exists u. (y, u) \in r^* \wedge (z, u) \in r^=$

<proof>

lemma *strongly-confluentI* [*intro*]:

$\llbracket \bigwedge x y z. (x, y) \in r \implies (x, z) \in r \implies \exists u. (y, u) \in r^* \wedge (z, u) \in r^= \rrbracket \implies$

strongly-confluent r

<proof>

lemma *strongly-confluent-E1n*:

assumes *scr*: *strongly-confluent* r

shows $(x, y) \in r^= \implies (x, z) \in r \smallfrown n \implies \exists u. (y, u) \in r^* \wedge (z, u) \in r^=$

<proof>

lemma *strong-confluence-imp-CR*:

assumes *strongly-confluent* r

shows $CR r$

<proof>

lemma *WCR-alt-def*: $WCR A \iff A^{-1} \circ A \subseteq A^\downarrow$ *<proof>*

lemma *NF-imp-SN-on*: $a \in NF R \implies SN-on R \{a\}$ *<proof>*

lemma *Union-sym*: $(s, t) \in (\bigcup_{i \leq n}. (S i)^{\leftrightarrow}) \iff (t, s) \in (\bigcup_{i \leq n}. (S i)^{\leftrightarrow})$ *<proof>*

lemma *peak-iff*: $(x, y) \in A^{-1} \circ B \iff (\exists u. (u, x) \in A \wedge (u, y) \in B)$ *<proof>*

lemma *CR-NF-conv*:

assumes *CR* r **and** $t \in NF\ r$ **and** $(u, t) \in r^{\leftrightarrow*}$

shows $(u, t) \in r^!$

<proof>

lemma *NF-join-imp-reach*:

assumes $y \in NF\ A$ **and** $(x, y) \in A^\downarrow$

shows $(x, y) \in A^*$

<proof>

lemma *conversion-O-conversion* [*simp*]:

$A^{\leftrightarrow*} \circ A^{\leftrightarrow*} = A^{\leftrightarrow*}$

<proof>

lemma *trans-O-iff*: $trans\ A \iff A \circ A \subseteq A$ *<proof>*

lemma *refl-O-iff*: $refl\ A \iff Id \subseteq A$ *<proof>*

lemma *relpow-Suc*: $r \overset{\sim}{\sim} Suc\ n = r \circ r \overset{\sim}{\sim} n$

<proof>

lemma *converse-power*: **fixes** $r :: 'a\ rel$ **shows** $(r^{-1}) \overset{\sim}{\sim} n = (r \overset{\sim}{\sim} n)^{-1}$

<proof>

lemma *conversion-mono*: $A \subseteq B \implies A^{\leftrightarrow*} \subseteq B^{\leftrightarrow*}$

<proof>

lemma *conversion-conversion-idemp* [*simp*]: $(A^{\leftrightarrow*})^{\leftrightarrow*} = A^{\leftrightarrow*}$

<proof>

lemma *lower-set-imp-not-SN-on*:

assumes $s \in X \ \forall t \in X. \exists u \in X. (t, u) \in R$ **shows** $\neg SN\text{-on}\ R\ \{s\}$

<proof>

lemma *SN-on-Image-rtrancl-iff*[*simp*]: $SN\text{-on}\ R\ (R^* \text{ `` } X) \iff SN\text{-on}\ R\ X$ (**is** ?)

<proof>

lemma *O-mono1*: $R \subseteq R' \implies S \circ R \subseteq S \circ R'$ *<proof>*

lemma *O-mono2*: $R \subseteq R' \implies R \circ T \subseteq R' \circ T$ *<proof>*

lemma *rtrancl-O-shift*: $(S \circ R)^* \circ S = S \circ (R \circ S)^*$

<proof>

lemma *O-rtrancl-O-O*: $R \circ (S \circ R)^* \circ S = (R \circ S)^+$

<proof>

lemma *SN-on-subset-SN-terms*:
assumes $SN: SN\text{-on } R \ X$ **shows** $X \subseteq \{x. SN\text{-on } R \ \{x\}\}$
 $\langle proof \rangle$

lemma *SN-on-Un2*:
assumes $SN\text{-on } R \ X$ **and** $SN\text{-on } R \ Y$ **shows** $SN\text{-on } R \ (X \cup Y)$
 $\langle proof \rangle$

lemma *SN-on-UN*:
assumes $\bigwedge x. SN\text{-on } R \ (X \ x)$ **shows** $SN\text{-on } R \ (\bigcup x. X \ x)$
 $\langle proof \rangle$

lemma *Image-subsetI*: $R \subseteq R' \implies R \text{ `` } X \subseteq R' \text{ `` } X$ $\langle proof \rangle$

lemma *SN-on-O-comm*:
assumes $SN: SN\text{-on } ((R :: ('a \times 'b) \text{ set}) \ O \ (S :: ('b \times 'a) \text{ set})) \ (S \text{ `` } X)$
shows $SN\text{-on } (S \ O \ R) \ X$
 $\langle proof \rangle$

lemma *SN-O-comm*: $SN \ (R \ O \ S) \longleftrightarrow SN \ (S \ O \ R)$
 $\langle proof \rangle$

lemma *chain-mono*: **assumes** $R' \subseteq R$ *chain* R' *seq* **shows** *chain* R *seq*
 $\langle proof \rangle$

context
fixes $S \ R$
assumes *push*: $S \ O \ R \subseteq R \ O \ S^*$
begin

lemma *rtrancl-O-push*: $S^* \ O \ R \subseteq R \ O \ S^*$
 $\langle proof \rangle$

lemma *rtrancl-U-push*: $(S \cup R)^* = R^* \ O \ S^*$
 $\langle proof \rangle$

lemma *SN-on-O-push*:
assumes $SN: SN\text{-on } R \ X$ **shows** $SN\text{-on } (R \ O \ S^*) \ X$
 $\langle proof \rangle$

lemma *SN-on-Image-push*:
assumes $SN: SN\text{-on } R \ X$ **shows** $SN\text{-on } R \ (S^* \text{ `` } X)$
 $\langle proof \rangle$

end

lemma *not-SN-onI[intro]*: $f \ 0 \in X \implies \text{chain } R \ f \implies \neg SN\text{-on } R \ X$
 $\langle proof \rangle$

lemma *shift-comp*[simp]: $\text{shift } (f \circ \text{seq}) \ n = f \circ (\text{shift seq } n)$ $\langle \text{proof} \rangle$

lemma *Id-on-union*: $\text{Id-on } (A \cup B) = \text{Id-on } A \cup \text{Id-on } B$ $\langle \text{proof} \rangle$

lemma *relpow-union-cases*: $(a,d) \in (A \cup B)^{\sim n} \implies (a,d) \in B^{\sim n} \vee (\exists b \ c \ k \ m. (a,b) \in B^{\sim k} \wedge (b,c) \in A \wedge (c,d) \in (A \cup B)^{\sim m} \wedge n = \text{Suc } (k + m))$
 $\langle \text{proof} \rangle$

lemma *trans-refl-imp-rtrancl-id*:

assumes *trans* *r refl* *r*

shows $r^* = r$

$\langle \text{proof} \rangle$

lemma *trans-refl-imp-O-id*:

assumes *trans* *r refl* *r*

shows $r \ O \ r = r$

$\langle \text{proof} \rangle$

lemma *relcomp3-I*:

assumes $(t, u) \in A$ **and** $(s, t) \in B$ **and** $(u, v) \in B$

shows $(s, v) \in B \ O \ A \ O \ B$

$\langle \text{proof} \rangle$

lemma *relcomp3-transI*:

assumes *trans* *B* **and** $(t, u) \in B \ O \ A \ O \ B$ **and** $(s, t) \in B$ **and** $(u, v) \in B$

shows $(s, v) \in B \ O \ A \ O \ B$

$\langle \text{proof} \rangle$

lemmas *converse-inward* = *rtrancl-converse*[*symmetric*] *converse-Un* *converse-UNION*
converse-relcomp

converse-converse *converse-Id*

lemma *qc-SN-relto-iff*:

assumes $r \ O \ s \subseteq s \ O \ (s \cup r)^*$

shows $\text{SN } (r^* \ O \ s \ O \ r^*) = \text{SN } s$

$\langle \text{proof} \rangle$

lemma *conversion-empty* [simp]: $\text{conversion } \{\} = \text{Id}$

$\langle \text{proof} \rangle$

lemma *symcl-idemp* [simp]: $(r^{\leftrightarrow})^{\leftrightarrow} = r^{\leftrightarrow}$ $\langle \text{proof} \rangle$

end

3 Relative Rewriting

theory *Relative-Rewriting*

imports *Abstract-Rewriting*

begin

Considering a relation R relative to another relation S , i.e., R -steps may be preceded and followed by arbitrary many S -steps.

abbreviation (*input*) $relto :: 'a rel \Rightarrow 'a rel \Rightarrow 'a rel$ **where**
 $relto R S \equiv S^* O R O S^*$

definition $SN\text{-rel-on} :: 'a rel \Rightarrow 'a rel \Rightarrow 'a set \Rightarrow bool$ **where**
 $SN\text{-rel-on} R S \equiv SN\text{-on} (relto R S)$

definition $SN\text{-rel-on-alt} :: 'a rel \Rightarrow 'a rel \Rightarrow 'a set \Rightarrow bool$ **where**
 $SN\text{-rel-on-alt} R S T = (\forall f. chain (R \cup S) f \wedge f 0 \in T \longrightarrow \neg (INFM j. (f j, f (Suc j)) \in R))$

abbreviation $SN\text{-rel} :: 'a rel \Rightarrow 'a rel \Rightarrow bool$ **where**
 $SN\text{-rel} R S \equiv SN\text{-rel-on} R S UNIV$

abbreviation $SN\text{-rel-alt} :: 'a rel \Rightarrow 'a rel \Rightarrow bool$ **where**
 $SN\text{-rel-alt} R S \equiv SN\text{-rel-on-alt} R S UNIV$

lemma $relto\text{-absorb}$ [*simp*]: $relto R E O E^* = relto R E E^* O relto R E = relto R E$
 $\langle proof \rangle$

lemma $steps\text{-preserve-SN-on-relto}$:
assumes $steps: (a, b) \in (R \cup S)^*$
and $SN: SN\text{-on} (relto R S) \{a\}$
shows $SN\text{-on} (relto R S) \{b\}$
 $\langle proof \rangle$

lemma $step\text{-preserves-SN-on-relto}$: **assumes** $st: (s,t) \in R \cup E$
and $SN: SN\text{-on} (relto R E) \{s\}$
shows $SN\text{-on} (relto R E) \{t\}$
 $\langle proof \rangle$

lemma $SN\text{-rel-on-imp-SN-rel-on-alt}$: $SN\text{-rel-on} R S T \Longrightarrow SN\text{-rel-on-alt} R S T$
 $\langle proof \rangle$

lemma $SN\text{-rel-on-alt-imp-SN-rel-on}$: $SN\text{-rel-on-alt} R S T \Longrightarrow SN\text{-rel-on} R S T$
 $\langle proof \rangle$

lemma $SN\text{-rel-on-conv}$: $SN\text{-rel-on} = SN\text{-rel-on-alt}$
 $\langle proof \rangle$

lemmas $SN\text{-rel-defs} = SN\text{-rel-on-def} SN\text{-rel-on-alt-def}$

lemma $SN\text{-rel-on-alt-r-empty}$: $SN\text{-rel-on-alt} \{\} S T$
 $\langle proof \rangle$

lemma $SN\text{-rel-on-alt-s-empty}$: $SN\text{-rel-on-alt} R \{\} = SN\text{-on} R$

<proof>

lemma *SN-rel-on-mono'*:

assumes $R: R \subseteq R'$ **and** $S: S \subseteq R' \cup S'$ **and** $SN: SN\text{-rel-on } R' S' T$
shows $SN\text{-rel-on } R S T$

<proof>

lemma *relto-mono*:

assumes $R \subseteq R'$ **and** $S \subseteq S'$
shows $relto R S \subseteq relto R' S'$

<proof>

lemma *SN-rel-on-mono*:

assumes $R: R \subseteq R'$ **and** $S: S \subseteq S'$
and $SN: SN\text{-rel-on } R' S' T$

shows $SN\text{-rel-on } R S T$

<proof>

lemmas $SN\text{-rel-on-alt-mono} = SN\text{-rel-on-mono}[unfolded SN\text{-rel-on-conv}]$

lemma *SN-rel-on-imp-SN-on*:

assumes $SN\text{-rel-on } R S T$ **shows** $SN\text{-on } R T$

<proof>

lemma *relto-Id*: $relto R (S \cup Id) = relto R S$ *<proof>*

lemma *SN-rel-on-Id*:

shows $SN\text{-rel-on } R (S \cup Id) T = SN\text{-rel-on } R S T$

<proof>

lemma *SN-rel-on-empty[simp]*: $SN\text{-rel-on } R \{\} T = SN\text{-on } R T$

<proof>

lemma *SN-rel-on-ideriv*: $SN\text{-rel-on } R S T = (\neg (\exists as. ideriv R S as \wedge as 0 \in T))$

(**is** $?L = ?R$)

<proof>

lemma *SN-rel-to-SN-rel-alt*: $SN\text{-rel } R S \implies SN\text{-rel-alt } R S$

<proof>

lemma *SN-rel-alt-to-SN-rel* : $SN\text{-rel-alt } R S \implies SN\text{-rel } R S$

<proof>

lemma *SN-rel-alt-r-empty* : $SN\text{-rel-alt } \{\} S$

<proof>

lemma *SN-rel-alt-s-empty* : $SN\text{-rel-alt } R \{\} = SN R$

<proof>

lemma *SN-rel-mono'*:

$R \subseteq R' \implies S \subseteq R' \cup S' \implies \text{SN-rel } R' S' \implies \text{SN-rel } R S$

<proof>

lemma *SN-rel-mono*:

assumes $R: R \subseteq R'$ **and** $S: S \subseteq S'$ **and** $\text{SN}: \text{SN-rel } R' S'$

shows $\text{SN-rel } R S$

<proof>

lemmas $\text{SN-rel-alt-mono} = \text{SN-rel-mono}[\text{unfolded SN-rel-on-conv}]$

lemma *SN-rel-imp-SN* : **assumes** $\text{SN-rel } R S$ **shows** $\text{SN } R$

<proof>

lemma *relto-trancl-conv* : $(\text{relto } R S)^{\hat{+}} = ((R \cup S))^{\hat{*}} \circ R \circ ((R \cup S))^{\hat{*}}$

<proof>

lemma *SN-rel-Id*:

shows $\text{SN-rel } R (S \cup \text{Id}) = \text{SN-rel } R S$

<proof>

lemma *relto-rtrancl*: $\text{relto } R (S^{\hat{*}}) = \text{relto } R S$ *<proof>*

lemma *SN-rel-empty[simp]*: $\text{SN-rel } R \{\} = \text{SN } R$

<proof>

lemma *SN-rel-ideriv*: $\text{SN-rel } R S = (\neg (\exists \text{ as. ideriv } R S \text{ as}))$ (**is** $?L = ?R$)

<proof>

lemma *SN-rel-map*:

fixes $R R_w R' R_w' :: 'a \text{ rel}$

defines $A: A \equiv R' \cup R_w'$

assumes $\text{SN}: \text{SN-rel } R' R_w'$

and $R: \bigwedge s t. (s, t) \in R \implies (f s, f t) \in A^{\hat{*}} \circ R' \circ A^{\hat{*}}$

and $R_w: \bigwedge s t. (s, t) \in R_w \implies (f s, f t) \in A^{\hat{*}}$

shows $\text{SN-rel } R R_w$

<proof>

datatype *SN-rel-ext-type* = *top-s* | *top-ns* | *normal-s* | *normal-ns*

fun *SN-rel-ext-step* :: $'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow \text{SN-rel-ext-type} \Rightarrow 'a \text{ rel}$

where

$\text{SN-rel-ext-step } P P_w R R_w \text{ top-s} = P$

| $\text{SN-rel-ext-step } P P_w R R_w \text{ top-ns} = P_w$

| $\text{SN-rel-ext-step } P P_w R R_w \text{ normal-s} = R$

| $\text{SN-rel-ext-step } P P_w R R_w \text{ normal-ns} = R_w$

definition *SN-rel-ext* :: $'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

where

$SN\text{-rel-ext } P \ Pw \ R \ Rw \ M \equiv (\neg (\exists f \ t.$
 $(\forall i. (f \ i, f \ (Suc \ i)) \in SN\text{-rel-ext-step } P \ Pw \ R \ Rw \ (t \ i))$
 $\wedge (\forall i. M \ (f \ i))$
 $\wedge (INFM \ i. t \ i \in \{top\text{-}s, top\text{-}ns\})$
 $\wedge (INFM \ i. t \ i \in \{top\text{-}s, normal\text{-}s\})))$

lemma *SN-rel-ext-step-mono*: **assumes** $P \subseteq P' \ Pw \subseteq Pw' \ R \subseteq R' \ Rw \subseteq Rw'$
shows $SN\text{-rel-ext-step } P \ Pw \ R \ Rw \ t \subseteq SN\text{-rel-ext-step } P' \ Pw' \ R' \ Rw' \ t$
 $\langle proof \rangle$

lemma *SN-rel-ext-mono*: **assumes** *subset*: $P \subseteq P' \ Pw \subseteq Pw' \ R \subseteq R' \ Rw \subseteq Rw'$
and
 $SN: SN\text{-rel-ext } P' \ Pw' \ R' \ Rw' \ M$ **shows** $SN\text{-rel-ext } P \ Pw \ R \ Rw \ M$
 $\langle proof \rangle$

lemma *SN-rel-ext-trans*:

fixes $P \ Pw \ R \ Rw :: 'a \ rel$ **and** $M :: 'a \Rightarrow bool$
defines $M': M' \equiv \{(s, t). M \ t\}$
defines $A: A \equiv (P \cup Pw \cup R \cup Rw) \cap M'$
assumes $SN\text{-rel-ext } P \ Pw \ R \ Rw \ M$
shows $SN\text{-rel-ext } (A \hat{*} \ O \ (P \cap M') \ O \ A \hat{*}) \ (A \hat{*} \ O \ ((P \cup Pw) \cap M') \ O \ A \hat{*})$
 $(A \hat{*} \ O \ ((P \cup R) \cap M') \ O \ A \hat{*}) \ (A \hat{*}) \ M$ (**is** $SN\text{-rel-ext } ?P \ ?Pw \ ?R \ ?Rw \ M$)
 $\langle proof \rangle$

lemma *SN-rel-ext-map*: **fixes** $P \ Pw \ R \ Rw \ P' \ Pw' \ R' \ Rw' :: 'a \ rel$ **and** $M \ M' :: 'a \Rightarrow bool$

defines $Ms: Ms \equiv \{(s, t). M' \ t\}$
defines $A: A \equiv (P' \cup Pw' \cup R' \cup Rw') \cap Ms$
assumes $SN: SN\text{-rel-ext } P' \ Pw' \ R' \ Rw' \ M'$
and $P: \bigwedge s \ t. M \ s \Longrightarrow M \ t \Longrightarrow (s, t) \in P \Longrightarrow (f \ s, f \ t) \in (A \hat{*} \ O \ (P' \cap Ms) \ O \ A \hat{*}) \wedge I \ t$
and $Pw: \bigwedge s \ t. M \ s \Longrightarrow M \ t \Longrightarrow (s, t) \in Pw \Longrightarrow (f \ s, f \ t) \in (A \hat{*} \ O \ ((P' \cup Pw') \cap Ms) \ O \ A \hat{*}) \wedge I \ t$
and $R: \bigwedge s \ t. I \ s \Longrightarrow M \ s \Longrightarrow M \ t \Longrightarrow (s, t) \in R \Longrightarrow (f \ s, f \ t) \in (A \hat{*} \ O \ ((P' \cup R') \cap Ms) \ O \ A \hat{*}) \wedge I \ t$
and $Rw: \bigwedge s \ t. I \ s \Longrightarrow M \ s \Longrightarrow M \ t \Longrightarrow (s, t) \in Rw \Longrightarrow (f \ s, f \ t) \in A \hat{*} \wedge I \ t$
shows $SN\text{-rel-ext } P \ Pw \ R \ Rw \ M$
 $\langle proof \rangle$

lemma *SN-rel-ext-map-min*: **fixes** $P \ Pw \ R \ Rw \ P' \ Pw' \ R' \ Rw' :: 'a \ rel$ **and** $M \ M' :: 'a \Rightarrow bool$

defines $Ms: Ms \equiv \{(s, t). M' \ t\}$
defines $A: A \equiv P' \cap Ms \cup Pw' \cap Ms \cup R' \cup Rw'$
assumes $SN: SN\text{-rel-ext } P' \ Pw' \ R' \ Rw' \ M'$
and $M: \bigwedge t. M \ t \Longrightarrow M' \ (f \ t)$
and $M': \bigwedge s \ t. M' \ s \Longrightarrow (s, t) \in R' \cup Rw' \Longrightarrow M' \ t$

and $P: \bigwedge s t. M s \implies M t \implies M' (f s) \implies M' (f t) \implies (s,t) \in P \implies (f s, f t) \in (A \widehat{*} O (P' \cap Ms) O A \widehat{*}) \wedge I t$
and $Pw: \bigwedge s t. M s \implies M t \implies M' (f s) \implies M' (f t) \implies (s,t) \in Pw \implies (f s, f t) \in (A \widehat{*} O (P' \cap Ms \cup Pw' \cap Ms) O A \widehat{*}) \wedge I t$
and $R: \bigwedge s t. I s \implies M s \implies M t \implies M' (f s) \implies M' (f t) \implies (s,t) \in R \implies (f s, f t) \in (A \widehat{*} O (P' \cap Ms \cup R') O A \widehat{*}) \wedge I t$
and $Rw: \bigwedge s t. I s \implies M s \implies M t \implies M' (f s) \implies M' (f t) \implies (s,t) \in Rw \implies (f s, f t) \in A \widehat{*} \wedge I t$
shows $SN\text{-rel-ext } P Pw R Rw M$
 $\langle proof \rangle$

lemma $SN\text{-relto-imp-SN-rel}: SN (relto R S) \implies SN\text{-rel } R S$
 $\langle proof \rangle$

lemma $rtrancl\text{-list-conv}$:

$((s,t) \in R \widehat{*}) =$
 $(\exists list. last (s \# list) = t \wedge (\forall i. i < length list \longrightarrow ((s \# list) ! i, (s \# list) ! Suc i) \in R))$ (**is** $?l = ?r$)
 $\langle proof \rangle$

fun $choice :: (nat \Rightarrow 'a list) \Rightarrow nat \Rightarrow (nat \times nat)$ **where**
 $choice f 0 = (0,0)$
 $| choice f (Suc n) = (let (i, j) = choice f n in$
 $if Suc j < length (f i)$
 $then (i, Suc j)$
 $else (Suc i, 0))$

lemma $SN\text{-rel-imp-SN-relto} : SN\text{-rel } R S \implies SN (relto R S)$
 $\langle proof \rangle$

hide-const $choice$

lemma $SN\text{-relto-SN-rel-conv}: SN (relto R S) = SN\text{-rel } R S$
 $\langle proof \rangle$

lemma $SN\text{-rel-empty1}: SN\text{-rel } \{\} S$
 $\langle proof \rangle$

lemma $SN\text{-rel-empty2}: SN\text{-rel } R \{\} = SN R$
 $\langle proof \rangle$

lemma $SN\text{-relto-mono}$:

assumes $R: R \subseteq R'$ **and** $S: S \subseteq S'$
and $SN: SN (relto R' S')$
shows $SN (relto R S)$
 $\langle proof \rangle$

lemma *SN-relto-imp-SN*:
assumes SN (*relto* R S) **shows** SN R
 \langle *proof* \rangle

lemma *SN-relto-Id*:
 SN (*relto* R ($S \cup Id$)) = SN (*relto* R S)
 \langle *proof* \rangle

Termination inheritance by transitivity (see, e.g., Geser's thesis).

lemma *trans-subset-SN*:
assumes *trans* R **and** $R \subseteq (r \cup s)$ **and** SN r **and** SN s
shows SN R
 \langle *proof* \rangle

lemma *SN-Un-conv*:
assumes *trans* ($r \cup s$)
shows SN ($r \cup s$) \longleftrightarrow SN r \wedge SN s
(is SN ? r \longleftrightarrow ? rhs)
 \langle *proof* \rangle

lemma *SN-relto-Un*:
 SN (*relto* ($R \cup S$) Q) \longleftrightarrow SN (*relto* R ($S \cup Q$)) \wedge SN (*relto* S Q)
(is SN ? a \longleftrightarrow SN ? b \wedge SN ? c)
 \langle *proof* \rangle

lemma *SN-relto-split*:
assumes SN (*relto* r ($s \cup q2$) \cup *relto* $q1$ ($s \cup q2$)) (is SN ? a)
and SN (*relto* s $q2$) (is SN ? b)
shows SN (*relto* r ($q1 \cup q2$) \cup *relto* s ($q1 \cup q2$)) (is SN ? c)
 \langle *proof* \rangle

lemma *relto-transcl-subset*: **assumes** $a \subseteq c$ **and** $b \subseteq c$ **shows** *relto* a $b \subseteq c^{\hat{+}}$
 \langle *proof* \rangle

An explicit version of *relto* which mentions all intermediate terms

inductive *relto-fun* :: ' a *rel* \Rightarrow ' a *rel* \Rightarrow nat \Rightarrow (nat \Rightarrow ' a) \Rightarrow (nat \Rightarrow $bool$) \Rightarrow nat
 \Rightarrow ' a \times ' a \Rightarrow $bool$ **where**
relto-fun: as $0 = a \Longrightarrow as$ $m = b \Longrightarrow$
(\bigwedge i . $i < m \Longrightarrow$
(sel $i \longrightarrow (as$ i , as (Suc i)) \in A) \wedge (\neg sel $i \longrightarrow (as$ i , as (Suc i)) \in B)
 $\Longrightarrow n = card$ { i . $i < m \wedge sel$ i }
 $\Longrightarrow (n = 0 \longleftrightarrow m = 0) \Longrightarrow relto-fun$ A B n as sel m (a, b)

lemma *relto-funD*: **assumes** *relto-fun* A B n as sel m (a, b)
shows as $0 = a$ as $m = b$
 \bigwedge i . $i < m \Longrightarrow sel$ $i \Longrightarrow (as$ i , as (Suc i)) \in A
 \bigwedge i . $i < m \Longrightarrow \neg sel$ $i \Longrightarrow (as$ i , as (Suc i)) \in B
 $n = card$ { i . $i < m \wedge sel$ i }
 $n = 0 \longleftrightarrow m = 0$

<proof>

lemma *relto-fun-refl*: \exists *as sel. relto-fun A B 0 as sel 0 (a,a)*
<proof>

lemma *relto-into-relto-fun*: **assumes** $(a,b) \in \text{relto } A \ B$
shows \exists *as sel m. relto-fun A B (Suc 0) as sel m (a,b)*
<proof>

lemma *relto-fun-trans*: **assumes** *ab: relto-fun A B n1 as1 sel1 m1 (a,b)*
and *bc: relto-fun A B n2 as2 sel2 m2 (b,c)*
shows \exists *as sel. relto-fun A B (n1 + n2) as sel (m1 + m2) (a,c)*
<proof>

lemma *reltos-into-relto-fun*: **assumes** $(a,b) \in (\text{relto } A \ B)^{\sim n}$
shows \exists *as sel m. relto-fun A B n as sel m (a,b)*
<proof>

lemma *relto-fun-into-reltos*: **assumes** *relto-fun A B n as sel m (a,b)*
shows $(a,b) \in (\text{relto } A \ B)^{\sim n}$
<proof>

lemma *relto-relto-fun-conv*: $((a,b) \in (\text{relto } A \ B)^{\sim n}) = (\exists$ *as sel m. relto-fun A B n as sel m (a,b))
*<proof>**

lemma *relto-fun-intermediate*: **assumes** $A \subseteq C$ **and** $B \subseteq C$
and *rf: relto-fun A B n as sel m (a,b)*
shows $i \leq m \implies (a,as \ i) \in C^{\sim*}$
<proof>

lemma *not-SN-on-rel-succ*:
assumes $\neg \text{SN-on } (\text{relto } R \ E) \ \{s\}$
shows $\exists t \ u. (s, t) \in E^* \wedge (t, u) \in R \wedge \neg \text{SN-on } (\text{relto } R \ E) \ \{u\}$
<proof>

lemma *SN-on-relto-relcomp*: $\text{SN-on } (\text{relto } R \ S) \ T = \text{SN-on } (S^* \ O \ R) \ T$ (**is** *?L T = ?R T*)
<proof>

lemma *trans-relto*:
assumes *trans: trans R and S O R \subseteq R O S*
shows *trans (relto R S)*
<proof>

lemma *relative-ending*:
assumes *chain: chain (R \cup S) t*
and *t0: t 0 \in X*
and *SN: SN-on (relto R S) X*

shows $\exists j. \forall i \geq j. (t\ i, t\ (Suc\ i)) \in S - R$
 ⟨*proof*⟩

from Geser’s thesis [p.32, Corollary-1], generalized for *SN-on*.

lemma *SN-on-relto-Un*:

assumes *closure*: $relto\ (R\ \cup\ R')\ S\ \text{“}\ X\ \subseteq\ X$

shows $SN\text{-on}\ (relto\ (R\ \cup\ R')\ S)\ X\ \longleftrightarrow\ SN\text{-on}\ (relto\ R\ (R'\ \cup\ S))\ X\ \wedge\ SN\text{-on}\ (relto\ R'\ S)\ X$

(**is** $?c\ \longleftrightarrow\ ?a\ \wedge\ ?b$)

⟨*proof*⟩

lemma *SN-on-Un*: $(R\ \cup\ R')\ \text{“}\ X\ \subseteq\ X\ \Longrightarrow\ SN\text{-on}\ (R\ \cup\ R')\ X\ \longleftrightarrow\ SN\text{-on}\ (relto\ R\ R')\ X\ \wedge\ SN\text{-on}\ R'\ X$

⟨*proof*⟩

end

4 Strongly Normalizing Orders

theory *SN-Orders*

imports *Abstract-Rewriting*

begin

We define several classes of orders which are used to build ordered semirings. Note that we do not use Isabelle’s preorders since the condition $x > y = x \geq y \wedge y \not\geq x$ is sometimes not applicable. E.g., for δ -orders over the rationals we have $0.2 \geq 0.1 \wedge 0.1 \not\geq 0.2$, but $0.2 >_\delta 0.1$ does not hold if δ is larger than 0.1.

class *non-strict-order* = *ord* +

assumes *ge-refl*: $x \geq (x :: 'a)$

and *ge-trans*[*trans*]: $\llbracket x \geq y; (y :: 'a) \geq z \rrbracket \Longrightarrow x \geq z$

and *max-comm*: $\max\ x\ y = \max\ y\ x$

and *max-ge-x*[*intro*]: $\max\ x\ y \geq x$

and *max-id*: $x \geq y \Longrightarrow \max\ x\ y = x$

and *max-mono*: $x \geq y \Longrightarrow \max\ z\ x \geq \max\ z\ y$

begin

lemma *max-ge-y*[*intro*]: $\max\ x\ y \geq y$

⟨*proof*⟩

lemma *max-mono2*: $x \geq y \Longrightarrow \max\ x\ z \geq \max\ y\ z$

⟨*proof*⟩

end

class *ordered-ab-semigroup* = *non-strict-order* + *ab-semigroup-add* + *monoid-add* +

assumes *plus-left-mono*: $x \geq y \Longrightarrow x + z \geq y + z$

lemma *plus-right-mono*: $y \geq (z :: 'a :: ordered\text{-ab}\text{-semigroup}) \Longrightarrow x + y \geq x + z$

<proof>

```

class ordered-semiring-0 = ordered-ab-semigroup + semiring-0 +
assumes times-left-mono:  $z \geq 0 \implies x \geq y \implies x * z \geq y * z$ 
and times-right-mono:  $x \geq 0 \implies y \geq z \implies x * y \geq x * z$ 
and times-left-anti-mono:  $x \geq y \implies 0 \geq z \implies y * z \geq x * z$ 

```

```

class ordered-semiring-1 = ordered-semiring-0 + semiring-1 +
assumes one-ge-zero:  $1 \geq 0$ 

```

We do not use a class to define order-pairs of a strict and a weak-order since often we have parametric strict orders, e.g. on rational numbers there are several orders $>$ where $x > y = x \geq y + \delta$ for some parameter δ

```

locale order-pair =
fixes gt :: 'a :: {non-strict-order,zero}  $\Rightarrow$  'a  $\Rightarrow$  bool (infix  $\succ$  50)
and default :: 'a
assumes compat[trans]:  $\llbracket x \geq y; y \succ z \rrbracket \implies x \succ z$ 
and compat2[trans]:  $\llbracket x \succ y; y \geq z \rrbracket \implies x \succ z$ 
and gt-imp-ge:  $x \succ y \implies x \geq y$ 
and default-ge-zero: default  $\geq 0$ 

```

begin

```

lemma gt-trans[trans]:  $\llbracket x \succ y; y \succ z \rrbracket \implies x \succ z$ 

```

<proof>

end

```

locale one-mono-ordered-semiring-1 = order-pair gt
for gt :: 'a :: ordered-semiring-1  $\Rightarrow$  'a  $\Rightarrow$  bool (infix  $\succ$  50) +
assumes plus-gt-left-mono:  $x \succ y \implies x + z \succ y + z$ 
and default-gt-zero: default  $\succ 0$ 

```

begin

```

lemma plus-gt-right-mono:  $x \succ y \implies a + x \succ a + y$ 

```

<proof>

```

lemma plus-gt-both-mono:  $x \succ y \implies a \succ b \implies x + a \succ y + b$ 

```

<proof>

end

```

locale SN-one-mono-ordered-semiring-1 = one-mono-ordered-semiring-1 + order-pair
+
assumes SN: SN  $\{(x,y) . y \geq 0 \wedge x \succ y\}$ 

```

```

locale SN-strict-mono-ordered-semiring-1 = SN-one-mono-ordered-semiring-1 +
fixes mono :: 'a :: ordered-semiring-1  $\Rightarrow$  bool
assumes mono:  $\llbracket \text{mono } x; y \succ z; x \geq 0 \rrbracket \implies x * y \succ x * z$ 

```

```

locale both-mono-ordered-semiring-1 = order-pair gt
for gt :: 'a :: ordered-semiring-1  $\Rightarrow$  'a  $\Rightarrow$  bool (infix  $\succ$  50) +
fixes arc-pos :: 'a  $\Rightarrow$  bool

```



```

assumes plus-gt-both-mono:  $\llbracket x \succ y; z \succ u \rrbracket \implies x + z \succ y + u$ 
and times-gt-left-mono:  $x \succ y \implies x * z \succ y * z$ 
and times-gt-right-mono:  $y \succ z \implies x * y \succ x * z$ 
and zero-leastI:  $x \succ 0$ 
and zero-leastII:  $0 \succ x \implies x = 0$ 
and zero-leastIII:  $(x :: 'a) \geq 0$ 
and arc-pos-one:  $\text{arc-pos } (1 :: 'a)$ 
and arc-pos-default:  $\text{arc-pos default}$ 
and arc-pos-zero:  $\neg \text{arc-pos } 0$ 
and arc-pos-plus:  $\text{arc-pos } x \implies \text{arc-pos } (x + y)$ 
and arc-pos-mult:  $\llbracket \text{arc-pos } x; \text{arc-pos } y \rrbracket \implies \text{arc-pos } (x * y)$ 
and not-all-ge:  $\bigwedge c d. \text{arc-pos } d \implies \exists e. e \geq 0 \wedge \text{arc-pos } e \wedge \neg (c \geq d * e)$ 
begin
lemma max0-id:  $\text{max } 0 (x :: 'a) = x$ 
  <proof>
end

locale SN-both-mono-ordered-semiring-1 = both-mono-ordered-semiring-1 +
  assumes SN:  $SN \{(x,y) . \text{arc-pos } y \wedge x \succ y\}$ 

locale weak-SN-strict-mono-ordered-semiring-1 =
  fixes weak-gt ::  $'a :: \text{ordered-semiring-1} \Rightarrow 'a \Rightarrow \text{bool}$ 
  and default ::  $'a$ 
  and mono ::  $'a \Rightarrow \text{bool}$ 
  assumes weak-gt-mono:  $\forall x y. (x,y) \in \text{set } xys \longrightarrow \text{weak-gt } x y \implies \exists gt.$ 
SN-strict-mono-ordered-semiring-1 default gt mono  $\wedge (\forall x y. (x,y) \in \text{set } xys \longrightarrow$ 
gt x y)

locale weak-SN-both-mono-ordered-semiring-1 =
  fixes weak-gt ::  $'a :: \text{ordered-semiring-1} \Rightarrow 'a \Rightarrow \text{bool}$ 
  and default ::  $'a$ 
  and arc-pos ::  $'a \Rightarrow \text{bool}$ 
  assumes weak-gt-both-mono:  $\forall x y. (x,y) \in \text{set } xys \longrightarrow \text{weak-gt } x y \implies \exists gt.$ 
SN-both-mono-ordered-semiring-1 default gt arc-pos  $\wedge (\forall x y. (x,y) \in \text{set } xys \longrightarrow$ 
gt x y)

class poly-carrier = ordered-semiring-1 + comm-semiring-1

locale poly-order-carrier = SN-one-mono-ordered-semiring-1 default gt
  for default ::  $'a :: \text{poly-carrier}$  and gt (infix  $\succ 50$ ) +
  fixes power-mono ::  $\text{bool}$ 
  and discrete ::  $\text{bool}$ 
  assumes times-gt-mono:  $\llbracket y \succ z; x \geq 1 \rrbracket \implies y * x \succ z * x$ 
  and power-mono:  $\text{power-mono} \implies x \succ y \implies y \geq 0 \implies n \geq 1 \implies x \hat{=} n \succ y$ 
  and discrete:  $\text{discrete} \implies x \geq y \implies \exists k. x = (((+ 1) \hat{=} k) y)$ 

class large-ordered-semiring-1 = poly-carrier +
  assumes ex-large-of-nat:  $\exists x. \text{of-nat } x \geq y$ 

```

context *ordered-semiring-1*
begin
lemma *pow-mono*: **assumes** $ab: a \geq b$ **and** $b: b \geq 0$
shows $a^n \geq b^n \wedge b^n \geq 0$
<proof>

lemma *pow-ge-zero*[*intro*]: **assumes** $a: a \geq (0 :: 'a)$
shows $a^n \geq 0$
<proof>
end

lemma *of-nat-ge-zero*[*intro,simp*]: *of-nat* $n \geq (0 :: 'a :: ordered-semiring-1)$
<proof>

lemma *mult-ge-zero*[*intro*]: $(a :: 'a :: ordered-semiring-1) \geq 0 \implies b \geq 0 \implies a * b \geq 0$
<proof>

lemma *pow-mono-one*: **assumes** $a: a \geq (1 :: 'a :: ordered-semiring-1)$
shows $a^n \geq 1$
<proof>

lemma *pow-mono-exp*: **assumes** $a: a \geq (1 :: 'a :: ordered-semiring-1)$
shows $n \geq m \implies a^n \geq a^m$
<proof>

lemma *mult-ge-one*[*intro*]: **assumes** $a: (a :: 'a :: ordered-semiring-1) \geq 1$
and $b: b \geq 1$
shows $a * b \geq 1$
<proof>

lemma *sum-list-ge-mono*: **fixes** $as :: ('a :: ordered-semiring-0) list$
assumes $length\ as = length\ bs$
and $\bigwedge i. i < length\ bs \implies as\ !\ i \geq bs\ !\ i$
shows $sum-list\ as \geq sum-list\ bs$
<proof>

lemma *sum-list-ge-0-nth*: **fixes** $xs :: ('a :: ordered-semiring-0) list$
assumes $ge: \bigwedge i. i < length\ xs \implies xs\ !\ i \geq 0$
shows $sum-list\ xs \geq 0$
<proof>

lemma *sum-list-ge-0*: **fixes** $xs :: ('a :: ordered-semiring-0) list$
assumes $ge: \bigwedge x. x \in set\ xs \implies x \geq 0$
shows $sum-list\ xs \geq 0$
<proof>

lemma *foldr-max*: $a \in set\ as \implies foldr\ max\ as\ b \geq (a :: 'a :: ordered-ab-semigroup)$

<proof>

lemma *of-nat-mono*[*intro*]: **assumes** $n \geq m$ **shows** (*of-nat* $n :: 'a :: \text{ordered-semiring-1}$)
 \geq *of-nat* m
<proof>

non infinitesimal is the same as in the CADE07 bounded increase paper

definition *non-inf* :: $'a \text{ rel} \Rightarrow \text{bool}$
where $\text{non-inf } r \equiv \forall a f. \exists i. (f i, f (\text{Suc } i)) \notin r \vee (f i, a) \notin r$

lemma *non-infI*[*intro*]: **assumes** $\bigwedge a f. \llbracket \bigwedge i. (f i, f (\text{Suc } i)) \in r \rrbracket \Longrightarrow \exists i. (f i, a) \notin r$
shows $\text{non-inf } r$
<proof>

lemma *non-infE*[*elim*]: **assumes** $\text{non-inf } r$ **and** $\bigwedge i. (f i, f (\text{Suc } i)) \notin r \vee (f i, a) \notin r \Longrightarrow P$
shows P
<proof>

lemma *non-inf-image*:
assumes $\text{ni}: \text{non-inf } r$ **and** *image*: $\bigwedge a b. (a, b) \in s \Longrightarrow (f a, f b) \in r$
shows $\text{non-inf } s$
<proof>

lemma *SN-imp-non-inf*: $\text{SN } r \Longrightarrow \text{non-inf } r$
<proof>

lemma *non-inf-imp-SN-bound*: $\text{non-inf } r \Longrightarrow \text{SN } \{(a, b). (b, c) \in r \wedge (a, b) \in r\}$
<proof>

end

5 Carriers of Strongly Normalizing Orders

theory *SN-Order-Carrier*
imports
 SN-Orders
 HOL.Rat
begin

This theory shows that standard semirings can be used in combination with polynomials, e.g. the naturals, integers, and arbitrary Archimedean fields by using delta-orders.

It also contains the arctic integers and arctic delta-orders where 0 is -infty, 1 is zero, + is max and * is plus.

5.1 The standard semiring over the naturals

instantiation *nat* :: *large-ordered-semiring-1*
begin
instance $\langle \text{proof} \rangle$
end

definition *nat-mono* :: *nat* \Rightarrow *bool* **where** *nat-mono* $x \equiv x \neq 0$

interpretation *nat-SN*: *SN-strict-mono-ordered-semiring-1* 1 ($>$) :: *nat* \Rightarrow *nat* \Rightarrow *bool* *nat-mono*
 $\langle \text{proof} \rangle$

interpretation *nat-poly*: *poly-order-carrier* 1 ($>$) :: *nat* \Rightarrow *nat* \Rightarrow *bool* *True discrete*
 $\langle \text{proof} \rangle$

5.2 The standard semiring over the Archimedean fields using delta-orderings

definition *delta-gt* :: '*a* :: *floor-ceiling* \Rightarrow '*a* \Rightarrow '*a* \Rightarrow *bool* **where**
delta-gt $\delta \equiv (\lambda x y. x - y \geq \delta)$

lemma *non-inf-delta-gt*: **assumes** *delta*: $\delta > 0$
shows *non-inf* $\{(a,b) . \text{delta-gt } \delta a b\}$ (**is** *non-inf* ?*r*)
 $\langle \text{proof} \rangle$

lemma *delta-gt-SN*: **assumes** *dpos*: $\delta > 0$ **shows** *SN* $\{(x,y). 0 \leq y \wedge \text{delta-gt } \delta x y\}$
 $\langle \text{proof} \rangle$

definition *delta-mono* :: '*a* :: *floor-ceiling* \Rightarrow *bool* **where** *delta-mono* $x \equiv x \geq 1$

subclass (**in** *floor-ceiling*) *large-ordered-semiring-1*
 $\langle \text{proof} \rangle$

lemma *delta-interpretation*: **assumes** *dpos*: $\delta > 0$ **and** *default*: $\delta \leq \text{def}$
shows *SN-strict-mono-ordered-semiring-1* *def* (*delta-gt* δ) *delta-mono*
 $\langle \text{proof} \rangle$

lemma *delta-poly*: **assumes** *dpos*: $\delta > 0$ **and** *default*: $\delta \leq \text{def}$
shows *poly-order-carrier* *def* (*delta-gt* δ) ($1 \leq \delta$) *False*
 $\langle \text{proof} \rangle$

lemma *delta-minimal-delta*: **assumes** $\bigwedge x y. (x,y) \in \text{set } xys \implies x > y$
shows $\exists \delta > 0. \forall x y. (x,y) \in \text{set } xys \longrightarrow \text{delta-gt } \delta x y$
 $\langle \text{proof} \rangle$

interpretation *weak-delta-SN*: *weak-SN-strict-mono-ordered-semiring-1* ($>$) 1 *delta-mono*
 \langle *proof* \rangle

5.3 The standard semiring over the integers

definition *int-mono* :: *int* \Rightarrow *bool* **where** *int-mono* $x \equiv x \geq 1$

instantiation *int* :: *large-ordered-semiring-1*

begin

instance

\langle *proof* \rangle

end

lemma *non-inf-int-gt*: *non-inf* $\{(a,b :: \textit{int}) . a > b\}$ (**is** *non-inf ?r*)
 \langle *proof* \rangle

interpretation *int-SN*: *SN-strict-mono-ordered-semiring-1* 1 ($>$) :: *int* \Rightarrow *int* \Rightarrow
bool int-mono
 \langle *proof* \rangle

interpretation *int-poly*: *poly-order-carrier* 1 ($>$) :: *int* \Rightarrow *int* \Rightarrow *bool True discrete*
 \langle *proof* \rangle

5.4 The arctic semiring over the integers

plus is interpreted as max, times is interpreted as plus, 0 is -infinity, 1 is 0

datatype *arctic* = *MinInfty* | *Num-arc int*

instantiation *arctic* :: *ord*

begin

fun *less-eq-arctic* :: *arctic* \Rightarrow *arctic* \Rightarrow *bool* **where**

| *less-eq-arctic* *MinInfty* $x = \textit{True}$

| *less-eq-arctic* (*Num-arc* $-$) *MinInfty* = *False*

| *less-eq-arctic* (*Num-arc* y) (*Num-arc* x) = ($y \leq x$)

fun *less-arctic* :: *arctic* \Rightarrow *arctic* \Rightarrow *bool* **where**

| *less-arctic* *MinInfty* $x = \textit{True}$

| *less-arctic* (*Num-arc* $-$) *MinInfty* = *False*

| *less-arctic* (*Num-arc* y) (*Num-arc* x) = ($y < x$)

instance \langle *proof* \rangle

end

instantiation *arctic* :: *ordered-semiring-1*

begin

fun *plus-arctic* :: *arctic* \Rightarrow *arctic* \Rightarrow *arctic* **where**

| *plus-arctic* *MinInfty* $y = y$

| *plus-arctic* x *MinInfty* = x

| *plus-arctic* (Num-arc x) (Num-arc y) = (Num-arc (max x y))

fun *times-arctic* :: arctic \Rightarrow arctic \Rightarrow arctic **where**
 times-arctic MinInfty y = MinInfty
| *times-arctic* x MinInfty = MinInfty
| *times-arctic* (Num-arc x) (Num-arc y) = (Num-arc (x + y))

definition *zero-arctic* :: arctic **where**
 zero-arctic = MinInfty

definition *one-arctic* :: arctic **where**
 one-arctic = Num-arc 0

instance
 <proof>
end

fun *get-arctic-num* :: arctic \Rightarrow int
where *get-arctic-num* (Num-arc n) = n

fun *pos-arctic* :: arctic \Rightarrow bool
where *pos-arctic* MinInfty = False
 | *pos-arctic* (Num-arc n) = (0 <= n)

interpretation *arctic-SN*: *SN-both-mono-ordered-semiring-1 1 (>)* *pos-arctic*
 <proof>

5.5 The arctic semiring over an arbitrary archimedean field

completely analogous to the integers, where one has to use delta-orderings

datatype 'a arctic-delta = MinInfty-delta | Num-arc-delta 'a

instantiation *arctic-delta* :: (ord) ord
begin

fun *less-eq-arctic-delta* :: 'a arctic-delta \Rightarrow 'a arctic-delta \Rightarrow bool **where**
 less-eq-arctic-delta MinInfty-delta x = True
| *less-eq-arctic-delta* (Num-arc-delta -) MinInfty-delta = False
| *less-eq-arctic-delta* (Num-arc-delta y) (Num-arc-delta x) = (y \leq x)

fun *less-arctic-delta* :: 'a arctic-delta \Rightarrow 'a arctic-delta \Rightarrow bool **where**
 less-arctic-delta MinInfty-delta x = True
| *less-arctic-delta* (Num-arc-delta -) MinInfty-delta = False
| *less-arctic-delta* (Num-arc-delta y) (Num-arc-delta x) = (y < x)

instance <proof>
end

instantiation *arctic-delta* :: (*linordered-field*) *ordered-semiring-1*

begin

fun *plus-arctic-delta* :: '*a* *arctic-delta* \Rightarrow '*a* *arctic-delta* \Rightarrow '*a* *arctic-delta* **where**
| *plus-arctic-delta* *MinInfty-delta* *y* = *y*
| *plus-arctic-delta* *x* *MinInfty-delta* = *x*
| *plus-arctic-delta* (*Num-arc-delta* *x*) (*Num-arc-delta* *y*) = (*Num-arc-delta* (*max* *x* *y*))

fun *times-arctic-delta* :: '*a* *arctic-delta* \Rightarrow '*a* *arctic-delta* \Rightarrow '*a* *arctic-delta* **where**
| *times-arctic-delta* *MinInfty-delta* *y* = *MinInfty-delta*
| *times-arctic-delta* *x* *MinInfty-delta* = *MinInfty-delta*
| *times-arctic-delta* (*Num-arc-delta* *x*) (*Num-arc-delta* *y*) = (*Num-arc-delta* (*x* + *y*))

definition *zero-arctic-delta* :: '*a* *arctic-delta* **where**
zero-arctic-delta = *MinInfty-delta*

definition *one-arctic-delta* :: '*a* *arctic-delta* **where**
one-arctic-delta = *Num-arc-delta* 0

instance

<proof>

end

x >d y is interpreted as *y = -inf* or (*x,y != -inf* and *x >d y*)

fun *gt-arctic-delta* :: '*a* :: *floor-ceiling* \Rightarrow '*a* *arctic-delta* \Rightarrow '*a* *arctic-delta* \Rightarrow *bool*
where *gt-arctic-delta* δ - *MinInfty-delta* = *True*
| *gt-arctic-delta* δ *MinInfty-delta* (*Num-arc-delta* -) = *False*
| *gt-arctic-delta* δ (*Num-arc-delta* *x*) (*Num-arc-delta* *y*) = *delta-gt* δ *x* *y*

fun *get-arctic-delta-num* :: '*a* *arctic-delta* \Rightarrow '*a*
where *get-arctic-delta-num* (*Num-arc-delta* *n*) = *n*

fun *pos-arctic-delta* :: ('*a* :: *floor-ceiling*) *arctic-delta* \Rightarrow *bool*
where *pos-arctic-delta* *MinInfty-delta* = *False*
| *pos-arctic-delta* (*Num-arc-delta* *n*) = (*0* \leq *n*)

lemma *arctic-delta-interpretation*: **assumes** *dpos*: $\delta > 0$ **shows** *SN-both-mono-ordered-semiring-1*
1 (*gt-arctic-delta* δ) *pos-arctic-delta*
<proof>

fun *weak-gt-arctic-delta* :: ('*a* :: *floor-ceiling*) *arctic-delta* \Rightarrow '*a* *arctic-delta* \Rightarrow *bool*
where *weak-gt-arctic-delta* - *MinInfty-delta* = *True*
| *weak-gt-arctic-delta* *MinInfty-delta* (*Num-arc-delta* -) = *False*
| *weak-gt-arctic-delta* (*Num-arc-delta* *x*) (*Num-arc-delta* *y*) = (*x* $>$ *y*)

interpretation *weak-arctic-delta-SN*: *weak-SN-both-mono-ordered-semiring-1* *weak-gt-arctic-delta*
1 *pos-arctic-delta*

<proof>

end

References

- [1] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, Aug. 1999.
- [2] C. Sternagel. *Automatic Certification of Termination Proofs*. PhD thesis, University of Innsbruck, Institute of Computer Science, 2010. not finished yet.
- [3] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, pages 452–468. Springer, 2009.