

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	7
2	Abstract Rewrite Systems	13
2.1	Definitions	13
2.2	Properties of ARSs	18
2.3	Newman's Lemma	37
2.4	Commutation	43
2.5	Strong Normalization	47
2.6	Terminating part of a relation	62
3	Relative Rewriting	72
4	Strongly Normalizing Orders	109

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

5	Carriers of Strongly Normalizing Orders	116
5.1	The standard semiring over the naturals	116
5.2	The standard semiring over the Archimedean fields using delta-orderings	117
5.3	The standard semiring over the integers	121
5.4	The arctic semiring over the integers	121
5.5	The arctic semiring over an arbitrary archimedean field . . .	126

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 $\text{nat} \Rightarrow 'a$.

```

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
  fix i show P (?S i) (?S (Suc i)) using assms by (cases i) simp-all
qed

```

Special version for relations.

```

lemma cons-chain:
  assumes (x, S 0)  $\in$  r and chain r S shows chain r (case-nat x S)
  using cons-chainp[of  $\lambda x y. (x, y) \in r, OF$  assms] .

```

A chain admits arbitrary transitive steps.

```

lemma chainp-imp-relpow:
  assumes chainp P S shows (P~j) (S i) (S (i + j))

```

proof *(induct i + j arbitrary: j)*
case $(\text{Suc } n)$ **thus** *?case using assms by (cases j) auto*
qed simp

lemma *chain-imp-relpow:*
assumes *chain r S shows* $(S\ i, S\ (i + j)) \in r^{\sim j}$
proof *(induct i + j arbitrary: j)*
case $(\text{Suc } n)$ **thus** *?case using assms by (cases j) auto*
qed simp

lemma *chainp-imp-tranclp:*
assumes *chainp P S and* $i < j$ **shows** $P^{\hat{++}}(S\ i)\ (S\ j)$
proof –
from *less-imp-Suc-add[OF assms(2)] obtain n where* $j = i + \text{Suc } n$ **by auto**
with *chainp-imp-relpow[of P S Suc n i, OF assms(1)]*
show *?thesis*
unfolding *trancl-power[of (S i, S j), to-pred]*
by force
qed

lemma *chain-imp-trancl:*
assumes *chain r S and* $i < j$ **shows** $(S\ i, S\ j) \in r^{\hat{+}}$
proof –
from *less-imp-Suc-add[OF assms(2)] obtain n where* $j = i + \text{Suc } n$ **by auto**
with *chain-imp-relpow[OF assms(1), of i Suc n]*
show *?thesis unfolding trancl-power by force*
qed

A chain admits arbitrary reflexive and transitive steps.

lemma *chainp-imp-rtranclp:*
assumes *chainp P S and* $i \leq j$ **shows** $P^{\hat{**}}(S\ i)\ (S\ j)$
proof –
from *assms(2) obtain n where* $j = i + n$ **by** *(induct j - i arbitrary: j) force+*
with *chainp-imp-relpow[of P S, OF assms(1), of n i] show* *?thesis*
by *(simp add: relpow-imp-rtrancl[of (S i, S (i + n)), to-pred])*
qed

lemma *chain-imp-rtrancl:*
assumes *chain r S and* $i \leq j$ **shows** $(S\ i, S\ j) \in r^{\hat{*}}$
proof –
from *assms(2) obtain n where* $j = i + n$ **by** *(induct j - i arbitrary: j) force+*
with *chain-imp-relpow[OF assms(1), of i n] show* *?thesis by (simp add: relpow-imp-rtrancl)*
qed

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 $\forall i \geq n::\text{nat}. f\ i \geq i \wedge P\ (S\ i)\ (S\ (f\ i))$
shows *chainp P* $(\lambda i. S\ ((f\ \hat{\sim}\ i)\ n))$ **(is** *chainp P ?T***)**

```

proof
  fix  $i$ 
  from  $assms$  have  $(f \text{ } \sim i) n \geq n$  by  $(induct\ i)$   $auto$ 
  with  $assms[THEN\ spec[of\ -\ (f \text{ } \sim i)\ n]]$ 
  show  $P\ (?T\ i)\ (?T\ (Suc\ i))$  by  $simp$ 
qed

```

```

lemma  $stepfun\text{-}imp\text{-}chainp$ :
  assumes  $\forall i \geq n :: nat. f\ i > i \wedge P\ (S\ i)\ (S\ (f\ i))$ 
  shows  $chainp\ P\ (\lambda i. S\ ((f \text{ } \sim i)\ n))$  (is  $chainp\ P\ ?T$ )
  using  $stepfun\text{-}imp\text{-}chainp'[of\ n\ f\ P\ S]$  and  $assms$  by  $force$ 

```

```

lemma  $subchain$ :
  assumes  $\forall i :: nat > n. \exists j > i. P\ (f\ i)\ (f\ j)$ 
  shows  $\exists \varphi. (\forall i\ j. i < j \longrightarrow \varphi\ i < \varphi\ j) \wedge (\forall i. P\ (f\ (\varphi\ i))\ (f\ (\varphi\ (Suc\ i))))$ 
proof -
  from  $assms$  have  $\forall i \in \{i. i > n\}. \exists j > i. P\ (f\ i)\ (f\ j)$  by  $simp$ 
  from  $bchoice\ [OF\ this]$  obtain  $g$ 
  where  $*$ :  $\forall i > n. g\ i > i$ 
  and  $**$ :  $\forall i > n. P\ (f\ i)\ (f\ (g\ i))$  by  $auto$ 
  define  $\varphi$  where  $[simp]$ :  $\varphi\ i = (g \text{ } \sim i)\ (Suc\ n)$  for  $i$ 
  from  $*$  have  $***$ :  $\bigwedge i. \varphi\ i > n$  by  $(induct\text{-}tac\ i)\ auto$ 
  then have  $\bigwedge i. \varphi\ i < \varphi\ (Suc\ i)$  using  $*$  by  $(induct\text{-}tac\ i)\ auto$ 
  then have  $\bigwedge i\ j. i < j \implies \varphi\ i < \varphi\ j$  by  $(rule\ lift\text{-}Suc\text{-}mono\text{-}less)$ 
  moreover have  $\bigwedge i. P\ (f\ (\varphi\ i))\ (f\ (\varphi\ (Suc\ i)))$  using  $**$  and  $***$  by  $simp$ 
  ultimately show  $?thesis$  by  $blast$ 
qed

```

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  $steps\text{-}imp\text{-}chainp'$ :
  assumes  $\forall i \geq n :: nat. \exists j \geq i. P\ (S\ i)\ (S\ j)$  shows  $\exists T. chainp\ P\ T$ 
proof -
  from  $assms$  have  $\forall i \in \{i. i \geq n\}. \exists j \geq i. P\ (S\ i)\ (S\ j)$  by  $auto$ 
  from  $bchoice\ [OF\ this]$ 
  obtain  $f$  where  $\forall i \geq n. f\ i \geq i \wedge P\ (S\ i)\ (S\ (f\ i))$  by  $auto$ 
  from  $stepfun\text{-}imp\text{-}chainp'[of\ n\ f\ P\ S, OF\ this]$  show  $?thesis$  by  $fast$ 
qed

```

```

lemma  $steps\text{-}imp\text{-}chainp$ :
  assumes  $\forall i \geq n :: nat. \exists j > i. P\ (S\ i)\ (S\ j)$  shows  $\exists T. chainp\ P\ T$ 
  using  $steps\text{-}imp\text{-}chainp'[of\ n\ P\ S]$  and  $assms$  by  $force$ 

```

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  $infinitely\text{-}many =$ 
  fixes  $p :: nat \Rightarrow bool$ 

```

```

    assumes infinite: INFM j. p j
  begin

  lemma inf:  $\exists j \geq i. p j$  using infinite[unfolded INFM-nat-le] by auto

  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 (induct n)
    case 0
    from inf obtain j where p j by auto
    with LeastI[of p j] show ?case by auto
  next
    case (Suc n)
    from inf obtain k where k  $\geq$  Suc (index n)  $\wedge$  p k by auto
    with LeastI[of  $\lambda k. p k \wedge k > index n$ ] show ?case by auto
  qed

  lemma index-ordered: index n < index (Suc n)
  proof -
    from inf obtain k where k  $\geq$  Suc (index n)  $\wedge$  p k by auto
    with LeastI[of  $\lambda k. p k \wedge k > index n$ ] show ?thesis by auto
  qed

  lemma index-not-p-between:
    assumes i1: index n < i
      and i2: i < index (Suc n)
    shows  $\neg p i$ 
  proof -
    from not-less-Least[OF i2[simplified]] i1 show ?thesis by auto
  qed

  lemma index-ordered-le:
    assumes i  $\leq$  j shows index i  $\leq$  index j
  proof -
    from assms have j = i + (j - i) by auto
    then obtain k where j: j = i + k by auto
    have index i  $\leq$  index (i + k)
    proof (induct k)
      case (Suc k)
      with index-ordered[of i + k]
      show ?case by auto
    qed simp
    thus ?thesis unfolding j .
  qed

  lemma index-surj:

```

```

assumes  $k \geq \text{index } l$ 
shows  $\exists i j. k = \text{index } i + j \wedge \text{index } i + j < \text{index } (\text{Suc } i)$ 
proof -
  from assms have  $k = \text{index } l + (k - \text{index } l)$  by auto
  then obtain  $u$  where  $k = \text{index } l + u$  by auto
  show ?thesis unfolding  $k$ 
  proof (induct u)
    case  $0$ 
    show ?case
    by (intro exI conjI, rule refl, insert index-ordered[of l], simp)
  next
    case (Suc u)
    then obtain  $i j$ 
    where  $lu: \text{index } l + u = \text{index } i + j$  and  $lt: \text{index } i + j < \text{index } (\text{Suc } i)$  by
auto
    hence  $\text{index } l + u < \text{index } (\text{Suc } i)$  by auto
    show ?case
    proof (cases index l + (Suc u) = index (Suc i))
      case False
      show ?thesis
      by (rule exI[of - i], rule exI[of - Suc j], insert lu lt False, auto)
    next
      case True
      show ?thesis
      by (rule exI[of - Suc i], rule exI[of - 0], insert True index-ordered[of Suc i],
auto)
    qed
  qed
qed

```

```

lemma index-ordered-less:
  assumes  $i < j$  shows  $\text{index } i < \text{index } j$ 
proof -
  from assms have  $\text{Suc } i \leq j$  by auto
  from index-ordered-le[OF this]
  have  $\text{index } (\text{Suc } i) \leq \text{index } j$  .
  with index-ordered[of i] show ?thesis by auto
qed

```

```

lemma index-not-p-start: assumes  $i: i < \text{index } 0$  shows  $\neg p \ i$ 
proof -
  from i[simplified index.simps] have  $i < \text{Least } p$  .
  from not-less-Least[OF this] show ?thesis .
qed

```

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-concat-simple f 0 = (0, 0)
| inf-concat-simple f (Suc n) = (
  let (i, j) = inf-concat-simple f n in
  if Suc j < f i then (i, Suc j)
  else (Suc i, 0))
```

```
lemma inf-concat-simple-add:
assumes ck: inf-concat-simple f k = (i, j)
and jl: j + l < f i
shows inf-concat-simple f (k + l) = (i, j + l)
using jl
proof (induct l)
  case 0
  thus ?case using ck by simp
next
  case (Suc l)
  hence c: inf-concat-simple f (k + l) = (i, j + l) by auto
  show ?case
  by (simp add: c, insert Suc(2), auto)
qed
```

```
lemma inf-concat-simple-surj-zero:  $\exists$  k. inf-concat-simple f k = (i, 0)
proof (induct i)
  case 0
  show ?case
  by (rule exI[of - 0], simp)
next
  case (Suc i)
  then obtain k where ck: inf-concat-simple f k = (i, 0) by auto
  show ?case
  proof (cases f i)
    case 0
    show ?thesis
    by (rule exI[of - Suc k], simp add: ck 0)
  next
    case (Suc n)
    hence 0 + n < f i by auto
    from inf-concat-simple-add[OF ck, OF this] Suc
    show ?thesis
    by (intro exI[of - k + Suc n], auto)
  qed
qed
```

```
lemma inf-concat-simple-surj:
assumes j < f i
```

shows $\exists k. \text{inf-concat-simple } f \ k = (i, j)$
proof –
from *assms* **have** $0 + j < f \ i$ **by** *auto*
from *inf-concat-simple-surj-zero* **obtain** k **where** $\text{inf-concat-simple } f \ k = (i, 0)$
by *auto*
from *inf-concat-simple-add*[*OF this, OF j*] **show** *?thesis* **by** *auto*
qed

lemma *inf-concat-simple-mono*:
assumes $k \leq k'$ **shows** $\text{fst } (\text{inf-concat-simple } f \ k) \leq \text{fst } (\text{inf-concat-simple } f \ k')$
proof –
from *assms* **have** $k' = k + (k' - k)$ **by** *auto*
then obtain l **where** $k' = k + l$ **by** *auto*
show *?thesis* **unfolding** k'
proof (*induct l*)
case (*Suc l*)
obtain $i \ j$ **where** $\text{ccl}: \text{inf-concat-simple } f \ (k+l) = (i, j)$ **by** (*cases inf-concat-simple f (k+l), auto*)
with *Suc* **have** $\text{fst } (\text{inf-concat-simple } f \ k) \leq i$ **by** *auto*
also have $\dots \leq \text{fst } (\text{inf-concat-simple } f \ (k + \text{Suc } l))$
by (*simp add: ccl*)
finally show *?case* .
qed *simp*
qed

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

lemma *inf-concat-bounds*:
assumes $\text{inf}: \text{INFM } i. n \ i > 0$
and $\text{res}: \text{inf-concat } n \ k = (i, j)$
shows $j < n \ i$
proof (*cases k*)
case 0
with res **have** $i: i = (\text{LEAST } i. n \ i > 0)$ **and** $j: j = 0$ **by** *auto*
from *inf*[*unfolded INFM-nat-le*] **obtain** i' **where** $0 < n \ i'$ **by** *auto*
have $0 < n$ (*LEAST i. n i > 0*)
by (*rule LeastI, rule i'*)
with $i \ j$ **show** *?thesis* **by** *auto*
next
case (*Suc k'*)
obtain $i' \ j'$ **where** $\text{res}': \text{inf-concat } n \ k' = (i', j')$ **by** *force*
note $\text{res} = \text{res}$ [*unfolded Suc inf-concat.simps res' Let-def split*]
show *?thesis*
proof (*cases Suc j' < n i'*)


```

    case True
    with res show ?thesis by auto
  next
  case False
  with res have i: i = (LEAST f. i' < f ∧ 0 < n f) and j: j = 0 by auto
  from inf[unfolded INFM-nat] obtain f where f: i' < f ∧ 0 < n f by auto
  have 0 < n (LEAST f. i' < f ∧ 0 < n f)
    using LeastI[of λ f. i' < f ∧ 0 < n f, OF f]
    by auto
  with i j show ?thesis by auto
qed
qed

```

```

lemma inf-concat-add:
  assumes res: inf-concat n k = (i,j)
    and j: j + m < n i
  shows inf-concat n (k + m) = (i,j+m)
  using j
proof (induct m)
  case 0 show ?case using res by auto
next
  case (Suc m)
  hence inf-concat n (k + m) = (i, j+m) by auto
  with Suc(2)
  show ?case by auto
qed

```

```

lemma inf-concat-step:
  assumes res: inf-concat n k = (i,j)
    and j: Suc (j + m) = n i
  shows inf-concat n (k + Suc m) = (LEAST i'. i' > i ∧ 0 < n i', 0)
proof -
  from j have j + m < n i by auto
  note res = inf-concat-add[OF res, OF this]
  show ?thesis by (simp add: res j)
qed

```

```

lemma inf-concat-surj-zero:
  assumes 0 < n i
  shows ∃ k. inf-concat n k = (i, 0)
proof -
  {
    fix l
    have ∀ j. j < l ∧ 0 < n j → (∃ k. inf-concat n k = (j,0))
    proof (induct l)
      case 0
      thus ?case by auto
    next
      case (Suc l)

```

```

show ?case
proof (intro allI impI, elim conjE)
  fix j
  assume j: j < Suc l and nj: 0 < n j
  show  $\exists k. \text{inf-concat } n \ k = (j, 0)$ 
  proof (cases j < l)
    case True
      from Suc[THEN spec[of - j]] True nj show ?thesis by auto
    next
      case False
        with j have j: j = l by auto
        show ?thesis
        proof (cases  $\exists j'. j' < l \wedge 0 < n \ j'$ )
          case False
            have l: (LEAST i. 0 < n i) = l
            proof (rule Least-equality, rule nj[unfolded j])
              fix l'
              assume 0 < n l'
              with False have  $\neg l' < l$  by auto
              thus  $l \leq l'$  by auto
            qed
          show ?thesis
          by (rule exI[of - 0], simp add: l j)
        next
          case True
            then obtain lll where lll: lll < l and nlll: 0 < n lll by auto
            then obtain ll where l: l = Suc ll by (cases l, auto)
            from lll l have lll: lll = ll - (ll - lll) by auto
            let ?l' = LEAST d. 0 < n (ll - d)
            have nl': 0 < n (ll - ?l')
            proof (rule LeastI)
              show 0 < n (ll - (ll - lll)) using lll nlll by auto
            qed
            with Suc[THEN spec[of - ll - ?l']] obtain k where k:
              inf-concat n k = (ll - ?l', 0) unfolding l by auto
            from nl' obtain off where off: Suc (0 + off) = n (ll - ?l') by (cases
n (ll - ?l'), auto)
            from inf-concat-step[OF k, OF off]
            have id: inf-concat n (k + Suc off) = (LEAST i'. ll - ?l' < i'  $\wedge$  0 < n
i', 0) (is - = (?l', 0)) .
            have ll: ?l = l unfolding l
            proof (rule Least-equality)
              show ll - ?l' < Suc ll  $\wedge$  0 < n (Suc ll) using nj[unfolded j l] by simp
            next
              fix l'
              assume ass: ll - ?l' < l'  $\wedge$  0 < n l'
              show Suc ll  $\leq$  l'
              proof (rule ccontr)
                assume not:  $\neg$  ?thesis

```

hence $l' \leq ll$ by *auto*
 hence $ll = l' + (ll - l')$ by *auto*
 then obtain k where $ll: ll = l' + k$ by *auto*
 from *ass* have $l' + k - ?l' < l'$ unfolding ll by *auto*
 hence $kl': k < ?l'$ by *auto*
 have $0 < n (ll - k)$ using *ass* unfolding ll by *simp*
 from *Least-le*[of $\lambda k. 0 < n (ll - k)$, *OF this*] kl'
 show *False* by *auto*
 qed
 qed
 show *?thesis* unfolding j
 by (*rule exI*[of $- k + Suc$ off], *unfold id ll, simp*)
 qed
 qed
 qed
 qed
 }
 with *assms* show *?thesis* by *auto*
 qed

lemma *inf-concat-surj*:
 assumes $j: j < n$ i
 shows $\exists k. \text{inf-concat } n \ k = (i, j)$
proof –
 from j have $0 < n$ i by *auto*
 from *inf-concat-surj-zero*[of n , *OF this*]
 obtain k where $\text{inf-concat } n \ k = (i, 0)$ by *auto*
 from *inf-concat-add*[*OF this*, of j] j
 show *?thesis* by *auto*
 qed

lemma *inf-concat-mono*:
 assumes *inf*: $INFM \ i. n \ i > 0$
 and *resk*: $\text{inf-concat } n \ k = (i, j)$
 and *reskp*: $\text{inf-concat } n \ k' = (i', j')$
 and *lt*: $i < i'$
 shows $k < k'$
proof –
 note *bounds* = *inf-concat-bounds*[*OF inf*]
 {
 assume $k' \leq k$
 hence $k = k' + (k - k')$ by *auto*
 then obtain l where $k: k = k' + l$ by *auto*
 have $i' \leq \text{fst } (\text{inf-concat } n \ (k' + l))$
proof (*induct l*)
 case 0
 with *reskp* show *?case* by *auto*
 next
 case (*Suc l*)

```

obtain  $i'' j''$  where  $l: \text{inf-concat } n (k' + l) = (i'', j'')$  by force
with  $\text{Suc}$  have one:  $i' \leq i''$  by auto
from  $\text{bounds}[OF l]$  have  $j'': j'' < n i''$  by auto
show  $?case$ 
proof ( $\text{cases } \text{Suc } j'' < n i''$ )
  case  $\text{True}$ 
    show  $?thesis$  by ( $\text{simp add: } l \text{ True one}$ )
  next
    case  $\text{False}$ 
      let  $?i = \text{LEAST } i'. i'' < i' \wedge 0 < n i'$ 
      from  $\text{inf}[\text{unfolded } \text{INFM-nat}]$  obtain  $k$  where  $i'' < k \wedge 0 < n k$  by auto
      from  $\text{LeastI}[\text{of } \lambda k. i'' < k \wedge 0 < n k, OF \text{ this}]$ 
      have  $i'' < ?i$  by auto
      with one show  $?thesis$  by ( $\text{simp add: } l \text{ False}$ )
    qed
  qed
with  $\text{resk } k \text{ lt}$  have  $\text{False}$  by auto
}
thus  $?thesis$  by arith
qed

```

lemma inf-concat-Suc :

```

assumes  $\text{inf: INFM } i. n i > 0$ 
  and  $f: \bigwedge i. f i (n i) = f (\text{Suc } i) 0$ 
  and  $\text{resk: inf-concat } n k = (i, j)$ 
  and  $\text{ressk: inf-concat } n (\text{Suc } k) = (i', j')$ 
shows  $f i' j' = f i (\text{Suc } j)$ 
proof –
  note  $\text{bounds} = \text{inf-concat-bounds}[OF \text{ inf}]$ 
  from  $\text{bounds}[OF \text{ resk}]$  have  $j: j < n i$  .
  show  $?thesis$ 
  proof ( $\text{cases } \text{Suc } j < n i$ )
    case  $\text{True}$ 
      with  $\text{ressk } \text{resk}$ 
      show  $?thesis$  by simp
    next
      case  $\text{False}$ 
        let  $?p = \lambda i'. i < i' \wedge 0 < n i'$ 
        let  $?i' = \text{LEAST } i'. ?p i'$ 
        from  $\text{False } j$  have  $\text{id: Suc } (j + 0) = n i$  by auto
        from  $\text{inf-concat-step}[OF \text{ resk}, OF \text{ id}]$   $\text{ressk}$ 
        have  $i': i' = ?i'$  and  $j': j' = 0$  by auto
        from  $\text{id}$  have  $j: \text{Suc } j = n i$  by simp
        from  $\text{inf}[\text{unfolded } \text{INFM-nat}]$  obtain  $k$  where  $?p k$  by auto
        from  $\text{LeastI}[\text{of } ?p, OF \text{ this}]$  have  $?p ?i'$  .
        hence  $?i' = \text{Suc } i + (?i' - \text{Suc } i)$  by simp
        then obtain  $d$  where  $ii': ?i' = \text{Suc } i + d$  by auto
        from  $\text{not-less-Least}[\text{of } - ?p, \text{unfolded } ii']$  have  $d': \bigwedge d'. d' < d \implies n (\text{Suc } i + d') = 0$  by auto

```

```

have f (Suc i) 0 = f ?i' 0 unfolding ii' using d'
proof (induct d)
  case 0
  show ?case by simp
next
  case (Suc d)
  hence f (Suc i) 0 = f (Suc i + d) 0 by auto
  also have ... = f (Suc (Suc i + d)) 0
    unfolding f[symmetric]
    using Suc(2)[of d] by simp
  finally show ?case by simp
qed
thus ?thesis unfolding i' j' j f by simp
qed
end

```

2 Abstract Rewrite Systems

```

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

```

```

lemma trancl-mono-set:
   $r \subseteq s \implies r^+ \subseteq s^+$ 
  by (blast intro: trancl-mono)

```

```

lemma relpow-mono:
  fixes r :: 'a rel
  assumes r  $\subseteq$  r' shows  $r \overset{\sim}{\sim} n \subseteq r' \overset{\sim}{\sim} n$ 
  using assms by (induct n) auto

```

```

lemma refl-inv-image:
   $\text{refl } R \implies \text{refl } (\text{inv-image } R f)$ 
  by (simp add: inv-image-def refl-on-def)

```

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 $((-\widehat{\leftarrow-\rightarrow})$ [1000] 999) **and**
conversion $((-\widehat{\leftarrow-\rightarrow}^*)$ [1000] 999) **and**
normalizability $((-\downarrow)$ [1000] 999)

lemma *symcl-converse*:
 $(A^{\leftrightarrow})^{-1} = A^{\leftrightarrow}$ **by** *auto*

lemma *symcl-Un*: $(A \cup B)^{\leftrightarrow} = A^{\leftrightarrow} \cup B^{\leftrightarrow}$ **by** *auto*

lemma *no-step*:
assumes $A \text{ “ } \{a\} = \{\}$ **shows** $a \in NF\ A$
using *assms* **by** (*auto simp: NF-def*)

lemma *joinI*:
 $(a, c) \in A^* \Longrightarrow (b, c) \in A^* \Longrightarrow (a, b) \in A^\downarrow$
by (*auto simp: join-def rtrancl-converse*)

lemma *joinI-left*:
 $(a, b) \in A^* \Longrightarrow (a, b) \in A^\downarrow$
by (*auto simp: join-def*)

lemma *joinI-right*: $(b, a) \in A^* \Longrightarrow (a, b) \in A^\downarrow$
by (*rule joinI*) *auto*

lemma *joinE*:
assumes $(a, b) \in A^\downarrow$
obtains c **where** $(a, c) \in A^*$ **and** $(b, c) \in A^*$

```

using assms by (auto simp: join-def rtrancl-converse)

lemma joinD:
   $(a, b) \in A^\downarrow \implies \exists c. (a, c) \in A^* \wedge (b, c) \in A^*$ 
  by (blast elim: joinE)

lemma meetI:
   $(a, b) \in A^* \implies (a, c) \in A^* \implies (b, c) \in A^\uparrow$ 
  by (auto simp: meet-def rtrancl-converse)

lemma meetE:
  assumes  $(b, c) \in A^\uparrow$ 
  obtains a where  $(a, b) \in A^*$  and  $(a, c) \in A^*$ 
  using assms by (auto simp: meet-def rtrancl-converse)

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

lemma conversionI:  $(a, b) \in (A^{\leftrightarrow})^* \implies (a, b) \in A^{\leftrightarrow*}$ 
  by (simp add: conversion-def)

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

lemma conversionI':
  assumes  $(a, b) \in A^*$  shows  $(a, b) \in A^{\leftrightarrow*}$ 
  using assms
  proof (induct)
    case base then show ?case by simp
  next
    case (step b c)
    then have  $(b, c) \in A^{\leftrightarrow}$  by simp
    with  $\langle (a, b) \in A^{\leftrightarrow*} \rangle$  show ?case unfolding conversion-def by (rule rtrancl.intros)
  qed

lemma rtrancl-comp-trancl-conv:
   $r^* \circ r = r^+$  by regex

lemma trancl-o-refl-is-trancl:
   $r^+ \circ r^= = r^+$  by regex

lemma conversionE:
   $(a, b) \in A^{\leftrightarrow*} \implies ((a, b) \in (A^{\leftrightarrow})^* \implies P) \implies P$ 
  by (simp add: conversion-def)

```

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 ( $A^{\leftrightarrow*}$ )
  unfolding trans-def
proof (intro allI impI)
  fix a b c assume  $(a, b) \in A^{\leftrightarrow*}$  and  $(b, c) \in A^{\leftrightarrow*}$ 
  then show  $(a, c) \in A^{\leftrightarrow*}$  unfolding conversion-def
  proof (induct)
    case base then show ?case by simp
  next
    case (step b c')
    from  $\langle (b, c') \in A^{\leftrightarrow} \rangle$  and  $\langle (c', c) \in (A^{\leftrightarrow})^* \rangle$ 
    have  $(b, c) \in (A^{\leftrightarrow})^*$  by (rule converse-rtrancl-into-rtrancl)
    with step show ?case by simp
  qed
qed

```

lemma *conversion-sym*:

```

  sym ( $A^{\leftrightarrow*}$ )
  unfolding sym-def
proof (intro allI impI)
  fix a b assume  $(a, b) \in A^{\leftrightarrow*}$  then show  $(b, a) \in A^{\leftrightarrow*}$  unfolding conversion-def
  proof (induct)
    case base then show ?case by simp
  next
    case (step b c)
    then have  $(c, b) \in A^{\leftrightarrow}$  by blast
    from  $\langle (c, b) \in A^{\leftrightarrow} \rangle$  and  $\langle (b, a) \in (A^{\leftrightarrow})^* \rangle$ 
    show ?case by (rule converse-rtrancl-into-rtrancl)
  qed
qed

```

lemma *conversion-inv*:

```

 $(x, y) \in R^{\leftrightarrow*} \iff (y, x) \in R^{\leftrightarrow*}$ 
by (auto simp: conversion-def)
  (metis (full-types) rtrancl-converseD symcl-converse)+

```


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

$$(A^{\leftrightarrow*})^{-1} = A^{\leftrightarrow*}$$

by (*metis conversion-sym sym-conv-converse-eq*)

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

$$(A^{\leftrightarrow*})^* = A^{\leftrightarrow*}$$

by (*metis conversion-def rtrancl-idemp*)

lemma *rtrancl-join-join*:

assumes $(a, b) \in A^*$ **and** $(b, c) \in A^\downarrow$ **shows** $(a, c) \in A^\downarrow$

proof –

from $\langle (b, c) \in A^\downarrow \rangle$ **obtain** b' **where** $(b, b') \in A^*$ **and** $(b', c) \in (A^{-1})^*$

unfolding *join-def* **by** *blast*

with $\langle (a, b) \in A^* \rangle$ **have** $(a, b') \in A^*$ **by** *simp*

with $\langle (b', c) \in (A^{-1})^* \rangle$ **show** *?thesis* **unfolding** *join-def* **by** *blast*

qed

lemma *join-rtrancl-join*:

assumes $(a, b) \in A^\downarrow$ **and** $(c, b) \in A^*$ **shows** $(a, c) \in A^\downarrow$

proof –

from $\langle (c, b) \in A^* \rangle$ **have** $(b, c) \in (A^{-1})^*$ **unfolding** *rtrancl-converse* **by** *simp*

from $\langle (a, b) \in A^\downarrow \rangle$ **obtain** a' **where** $(a, a') \in A^*$ **and** $(a', b) \in (A^{-1})^*$

unfolding *join-def* **by** *best*

with $\langle (b, c) \in (A^{-1})^* \rangle$ **have** $(a', c) \in (A^{-1})^*$ **by** *simp*

with $\langle (a, a') \in A^* \rangle$ **show** *?thesis* **unfolding** *join-def* **by** *blast*

qed

lemma *NF-I*: $(\bigwedge b. (a, b) \notin A) \implies a \in NF\ A$ **by** (*auto intro: no-step*)

lemma *NF-E*: $a \in NF\ A \implies ((a, b) \notin A \implies P) \implies P$ **by** (*auto simp: NF-def*)

declare *NF-I* [*intro*]

declare *NF-E* [*elim*]

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

lemma *NF-anti-mono*:

assumes $A \subseteq B$ **shows** $NF\ B \subseteq NF\ A$

using *assms* **by** *auto*

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

lemma *NF-no-trancl-step*:

assumes $a \in NF\ A$ **shows** $\forall b. (a, b) \notin A^+$

proof –

from *assms* **have** $\forall b. (a, b) \notin A$ **by** *auto*

show *?thesis*

proof (*intro allI notI*)

fix b **assume** $(a, b) \in A^+$
then show $False$ **by** (*induct*) (*auto simp*: $\langle \forall b. (a, b) \notin A \rangle$)
qed
qed

lemma *NF-Id-on-fst-image* [*simp*]: $NF (Id-on (fst \text{ ` } A)) = NF A$ **by** *force*

lemma *fst-image-NF-Id-on* [*simp*]: $fst \text{ ` } R = Q \implies NF (Id-on Q) = NF R$ **by** *force*

lemma *NF-empty* [*simp*]: $NF \{\} = UNIV$ **by** *auto*

lemma *normalizability-I*: $(a, b) \in A^* \implies b \in NF A \implies (a, b) \in A^!$
by (*simp add: normalizability-def*)

lemma *normalizability-I'*: $(a, b) \in A^* \implies (b, c) \in A^! \implies (a, c) \in A^!$
by (*auto simp add: normalizability-def*)

lemma *normalizability-E*: $(a, b) \in A^! \implies ((a, b) \in A^* \implies b \in NF A \implies P) \implies P$
by (*simp add: normalizability-def*)

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

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

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

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

Alternative definition of *SN*.

lemma *SN-def*: $SN \ r = (\forall x. SN-on \ r \ \{x\})$
unfolding *SN-on-def* **by** *blast*

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

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

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

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

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 $complete\text{-on}I$:
 $SN\text{-on } r \ A \Longrightarrow CR\text{-on } r \ A \Longrightarrow complete\text{-on } r \ A$
by (*simp add: complete-defs*)

lemma $complete\text{-on}E$:
 $complete\text{-on } r \ A \Longrightarrow (SN\text{-on } r \ A \Longrightarrow CR\text{-on } r \ A \Longrightarrow P) \Longrightarrow P$
by (*simp add: complete-defs*)

lemma $CR\text{-on}I$:
 $(\bigwedge a \ b \ c. a \in A \Longrightarrow (a, b) \in r^* \Longrightarrow (a, c) \in r^* \Longrightarrow (b, c) \in \text{join } r) \Longrightarrow CR\text{-on}$

$r A$

by (*simp add: CR-defs*)

lemma *CR-on-singletonI*:

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

by (*simp add: CR-defs*)

lemma *CR-onE*:

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

unfolding *CR-defs* **by** *blast*

lemma *CR-onD*:

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

by (*blast elim: CR-onE*)

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

by (*simp add: semi-complete-defs*)

lemma *semi-complete-onE*:

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

by (*simp add: semi-complete-defs*)

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*:

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

by (*simp add: UNC-def*)

lemma *UNC-E*:

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

unfolding *UNC-def* **by** *blast*

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

by (*simp add: UNF-defs*)

lemma *UNF-onE*:

$UNF\text{-on } r A \implies a \in A \implies (b = c \implies P) \implies ((a, b) \notin r^! \implies P) \implies ((a, c) \notin r^! \implies P) \implies P$

unfolding *UNF-on-def* **by** *blast*

lemma *UNF-onD*:

$UNF\text{-on } r A \implies a \in A \implies (a, b) \in r^! \implies (a, c) \in r^! \implies b = c$
by (*blast elim*: *UNF-onE*)

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* $r \ A$
using *assms* **unfolding** *SN-defs* **by** *blast*

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

unfolding *SN-on-def* **by** *blast*

lemma *SN-on-trancl-imp-SN-on*:

assumes *SN-on* $(R^+) \ T$ **shows** *SN-on* $R \ T$
proof (*rule ccontr*)
assume $\neg \text{SN-on } R \ T$
then obtain s **where** $s \ 0 \in T$ **and** *chain* $R \ s$ **unfolding** *SN-defs* **by** *auto*
then have *chain* $(R^+) \ s$ **by** *auto*
with $\langle s \ 0 \in T \rangle$ **have** $\neg \text{SN-on } (R^+) \ T$ **unfolding** *SN-defs* **by** *auto*
with *assms* **show** *False* **by** *simp*

qed

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
using *assms* **unfolding** *SN-defs* **by** *simp*

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
using *assms* **unfolding** *SN-defs* **by** *blast*

declare *SN-onI* [*intro*]

declare *SN-onE* [*elim*]

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

lemma *refl-not-SN*: $(x, x) \in R \implies \neg \text{SN } R$

unfolding *SN-defs* **by** *force*

lemma *SN-on-irrefl*:
assumes *SN-on r A*
shows $\forall a \in A. (a, a) \notin r$
proof (*intro ballI notI*)
fix *a* **assume** $a \in A$ **and** $(a, a) \in r$
with *assms* **show** *False* **unfolding** *SN-defs* **by** *auto*
qed

lemma *WCR-onI*: $(\bigwedge a b c. a \in A \implies (a, b) \in r \implies (a, c) \in r \implies (b, c) \in \text{join } r) \implies \text{WCR-on } r A$
by (*simp add: WCR-defs*)

lemma *WCR-onE*:
 $\text{WCR-on } r A \implies a \in A \implies ((b, c) \in \text{join } r \implies P) \implies ((a, b) \notin r \implies P) \implies ((a, c) \notin r \implies P) \implies P$
unfolding *WCR-on-def* **by** *blast*

lemma *SN-nat-bounded*: *SN* $\{(x, y :: \text{nat}). x < y \wedge y \leq b\}$ (**is** *SN ?R*)
proof
fix *f*
assume *chain ?R f*
then have *steps*: $\bigwedge i. (f i, f (\text{Suc } i)) \in ?R ..$
{
fix *i*
have *inc*: $f 0 + i \leq f i$
proof (*induct i*)
case *0* **then show** *?case* **by** *auto*
next
case (*Suc i*)
have $f 0 + \text{Suc } i \leq f i + \text{Suc } 0$ **using** *Suc* **by** *simp*
also have $\dots \leq f (\text{Suc } i)$ **using** *steps* [*of i*]
by *auto*
finally show *?case* **by** *simp*
qed
}
from *this* [*of Suc b*] *steps* [*of b*]
show *False* **by** *simp*
qed

lemma *WCR-onD*:
 $\text{WCR-on } r A \implies a \in A \implies (a, b) \in r \implies (a, c) \in r \implies (b, c) \in \text{join } r$
by (*blast elim: WCR-onE*)

lemma *WN-onI*: $(\bigwedge a. a \in A \implies \exists b. (a, b) \in r^!) \implies \text{WN-on } r A$
by (*auto simp: WN-defs*)

lemma *WN-onE*: $\text{WN-on } r A \implies a \in A \implies (\bigwedge b. (a, b) \in r^! \implies P) \implies P$
unfolding *WN-defs* **by** *blast*

lemma *WN-onD*: $WN\text{-on } r \ A \implies a \in A \implies \exists b. (a, b) \in r^!$
by (*blast elim: WN-onE*)

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 $r \ s = \{(a, b) \mid a \ b. (a, b) \in r \wedge SN\text{-on } s \ \{a\}\}$

lemma *SN-restrict-SN-idemp* [*simp*]: $SN \ (restrict\text{-}SN \ A \ A)$
by (*auto simp: restrict-SN-def SN-defs*)

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

proof

fix f
assume $f \ 0 \in r \text{ `` } A$ **and** *chain: chain* $r \ f$
then obtain a **where** $a \in A$ **and** $1: (a, f \ 0) \in r$ **by** *auto*
let $?g = \text{case-nat } a \ f$
from *cons-chain* [*OF* $1 \ \text{chain}$] **have** *chain* $r \ ?g$.
moreover have $?g \ 0 \in A$ **by** (*simp add: <a ∈ A>*)
ultimately have $\neg SN\text{-on } r \ A$ **unfolding** *SN-defs* **by** *best*
with *assms* **show** *False* **by** *simp*

qed

lemma *SN-on-subset2*:
assumes $A \subseteq B$ **and** $SN\text{-on } r \ B$
shows $SN\text{-on } r \ A$
using *assms* **unfolding** *SN-on-def* **by** *blast*

lemma *step-preserves-SN-on*:
assumes $1: (a, b) \in r$
and $2: SN\text{-on } r \ \{a\}$
shows $SN\text{-on } r \ \{b\}$
using 1 **and** *SN-on-Image* [*OF* 2] **and** *SN-on-subset2* [*of* $\{b\} \ r \text{ `` } \{a\}$] **by** *auto*

lemma *steps-preserve-SN-on*: $(a, b) \in A^* \implies SN\text{-on } A \ \{a\} \implies SN\text{-on } A \ \{b\}$
by (*induct rule: rtrancl.induct*) (*auto simp: step-preserves-SN-on*)

lemma *relpow-seq*:

assumes $(x, y) \in r^{\sim n}$
shows $\exists f. f\ 0 = x \wedge f\ n = y \wedge (\forall i < n. (f\ i, f\ (Suc\ i)) \in r)$
using *assms*
proof (*induct n arbitrary: y*)
case 0 then show *?case by auto*
next
case (*Suc n*)
then obtain *z where* $(x, z) \in r^{\sim n}$ **and** $(z, y) \in r$ **by** *auto*
from *Suc(1)[OF <(x, z) ∈ r[~]n>]*
obtain *f where* $f\ 0 = x$ **and** $f\ n = z$ **and** *seq:* $\forall i < n. (f\ i, f\ (Suc\ i)) \in r$ **by** *auto*
let *?n = Suc n*
let *?f = λi. if i = ?n then y else f i*
have *?f ?n = y* **by** *simp*
from $\langle f\ 0 = x \rangle$ **have** $\langle ?f\ 0 = x \rangle$ **by** *simp*
from *seq* **have** *seq':* $\forall i < n. (?f\ i, ?f\ (Suc\ i)) \in r$ **by** *auto*
with $\langle f\ n = z \rangle$ **and** $\langle (z, y) \in r \rangle$ **have** $\forall i < ?n. (?f\ i, ?f\ (Suc\ i)) \in r$ **by** *auto*
with $\langle ?f\ 0 = x \rangle$ **and** $\langle ?f\ ?n = y \rangle$ **show** *?case by best*
qed

lemma *rtrancl-imp-seq:*

assumes $(x, y) \in r^*$
shows $\exists f n. f\ 0 = x \wedge f\ n = y \wedge (\forall i < n. (f\ i, f\ (Suc\ i)) \in r)$
using *assms [unfolded rtrancl-power]* **and** *relpow-seq [of x y - r]* **by** *blast*

lemma *SN-on-Image-rtrancl:*

assumes *SN-on r A*
shows *SN-on r (r* “ A)*

proof

fix *f*
assume *f0:* $f\ 0 \in r^* \text{ “ } A$ **and** *chain:* *chain r f*
then obtain *a where* $a: a \in A$ **and** $(a, f\ 0) \in r^*$ **by** *auto*
then obtain *n where* $(a, f\ 0) \in r^{\sim n}$ **unfolding** *rtrancl-power* **by** *auto*
show *False*

proof (*cases n*)

case *0*

with $\langle (a, f\ 0) \in r^{\sim n} \rangle$ **have** $f\ 0 = a$ **by** *simp*
then have $f\ 0 \in A$ **by** (*simp add: a*)
with *chain* **have** $\neg \text{SN-on } r\ A$ **by** *auto*
with *assms* **show** *False by simp*

next

case (*Suc m*)

from *relpow-seq [OF <(a, f 0) ∈ r[~]n>]*
obtain *g where* $g\ 0 = a$ **and** $g\ n = f\ 0$
and *gseq:* $\forall i < n. (g\ i, g\ (Suc\ i)) \in r$ **by** *auto*
let *?f = λi. if i < n then g i else f (i - n)*
have *chain r ?f*

proof

fix *i*


```

{
  assume  $Suc\ i < n$ 
  then have  $(?f\ i, ?f\ (Suc\ i)) \in r$  by (simp add: gseq)
}
moreover
{
  assume  $Suc\ i > n$ 
  then have eq:  $Suc\ (i - n) = Suc\ i - n$  by arith
  from chain have  $(f\ (i - n), f\ (Suc\ (i - n))) \in r$  by simp
  then have  $(f\ (i - n), f\ (Suc\ i - n)) \in r$  by (simp add: eq)
  with  $\langle Suc\ i > n \rangle$  have  $(?f\ i, ?f\ (Suc\ i)) \in r$  by simp
}
moreover
{
  assume  $Suc\ i = n$ 
  then have eq:  $f\ (Suc\ i - n) = g\ n$  by (simp add:  $\langle g\ n = f\ 0 \rangle$ )
  from  $\langle Suc\ i = n \rangle$  have eq':  $i = n - 1$  by arith
  from gseq have  $(g\ i, f\ (Suc\ i - n)) \in r$  unfolding eq by (simp add:  $Suc$ 
eq^)
  then have  $(?f\ i, ?f\ (Suc\ i)) \in r$  using  $\langle Suc\ i = n \rangle$  by simp
}
ultimately show  $(?f\ i, ?f\ (Suc\ i)) \in r$  by simp
qed
moreover have  $?f\ 0 \in A$ 
proof (cases n)
  case 0
  with  $\langle (a, f\ 0) \in r \sim n \rangle$  have eq:  $a = f\ 0$  by simp
  from a show ?thesis by (simp add: eq 0)
next
  case (Suc m)
  then show ?thesis by (simp add: a g0)
qed
ultimately have  $\neg\ SN\text{-on}\ r\ A$  unfolding SN-defs by best
with assms show False by simp
qed
qed

```

declare subrelI [Pure.intro]

lemma restrict-SN-trancl-simp [simp]: $(restrict\ SN\ A\ A)^+ = restrict\ SN\ (A^+)\ A$
(is ?lhs = ?rhs)

```

proof
  show ?lhs  $\subseteq$  ?rhs
  proof
    fix a b assume  $(a, b) \in ?lhs$  then show  $(a, b) \in ?rhs$ 
    unfolding restrict-SN-def by (induct rule: trancl.induct) auto
  qed
next

```

show $?rhs \subseteq ?lhs$
proof
fix $a\ b$ **assume** $(a, b) \in ?rhs$
then have $(a, b) \in A^+$ **and** *SN-on A {a}* **unfolding** *restrict-SN-def* **by** *auto*
then show $(a, b) \in ?lhs$
proof (*induct rule: trancl.induct*)
case (*r-into-trancl x y*) **then show** *?case* **unfolding** *restrict-SN-def* **by** *auto*
next
case (*trancl-into-trancl a b c*)
then have *IH: (a, b) ∈ ?lhs* **by** *auto*
from *trancl-into-trancl* **have** $(a, b) \in A^*$ **by** *auto*
from *this* **and** $\langle SN\text{-on } A\ \{a\} \rangle$ **have** *SN-on A {b}* **by** (*rule steps-preserve-SN-on*)
with $\langle (b, c) \in A \rangle$ **have** $(b, c) \in ?lhs$ **unfolding** *restrict-SN-def* **by** *auto*
with *IH* **show** *?case* **by** *simp*
qed
qed
qed

lemma *SN-imp-WN*:
assumes *SN A* **shows** *WN A*
proof –
from $\langle SN\ A \rangle$ **have** *wf (A⁻¹)* **by** (*simp add: SN-defs wf-iff-no-infinite-down-chain*)
show *WN A*
proof
fix a
show $\exists b. (a, b) \in A^!$ **unfolding** *normalizability-def NF-def Image-def*
by (*rule wfE-min [OF ⟨wf (A⁻¹)⟩, of a A* “{a}, simplified]*)
(auto intro: rtrancl-into-rtrancl)
qed
qed

lemma *UNC-imp-UNF*:
assumes *UNC r* **shows** *UNF r*
proof – {
fix $x\ y\ z$ **assume** $(x, y) \in r^!$ **and** $(x, z) \in r^!$
then have $(x, y) \in r^*$ **and** $(x, z) \in r^*$ **and** $y \in NF\ r$ **and** $z \in NF\ r$ **by** *auto*
then have $(x, y) \in r^{\leftrightarrow*}$ **and** $(x, z) \in r^{\leftrightarrow*}$ **by** *auto*
then have $(z, x) \in r^{\leftrightarrow*}$ **using** *conversion-sym* **unfolding** *sym-def* **by** *best*
with $\langle (x, y) \in r^{\leftrightarrow*} \rangle$ **have** $(z, y) \in r^{\leftrightarrow*}$ **using** *conversion-trans* **unfolding**
trans-def **by** *best*
from *assms* **and** *this* **and** $\langle z \in NF\ r \rangle$ **and** $\langle y \in NF\ r \rangle$ **have** $z = y$ **unfolding**
UNC-def **by** *auto*
} **then show** *?thesis* **by** *auto*
qed

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 –

from $\langle (x, y) \in r^\downarrow \rangle$ **obtain** z **where** $(x, z) \in r^*$ **and** $(z, y) \in (r^{-1})^*$ **unfolding**
join-def by auto
then have $(y, z) \in r^*$ **unfolding** *rtrancl-converse by simp*
from $\langle x \in NF\ r \rangle$ **have** $(x, z) \notin r^+$ **using** *NF-no-trancl-step by best*
then have $x = z$ **using** *rtranclD [OF $\langle (x, z) \in r^* \rangle$] by auto*
from $\langle y \in NF\ r \rangle$ **have** $(y, z) \notin r^+$ **using** *NF-no-trancl-step by best*
then have $y = z$ **using** *rtranclD [OF $\langle (y, z) \in r^* \rangle$] by auto*
with $\langle x = z \rangle$ **show** *?thesis by simp*
qed

lemma *rtrancl-Restr*:
assumes $(x, y) \in (Restr\ r\ A)^*$
shows $(x, y) \in r^*$
using *assms by induct auto*

lemma *join-mono*:
assumes $r \subseteq s$
shows $r^\downarrow \subseteq s^\downarrow$
using *rtrancl-mono [OF assms] by (auto simp: join-def rtrancl-converse)*

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

proof
assume $CR\ r$ **show** $r^\uparrow \subseteq r^\downarrow$
proof (*rule subrelI*)
fix $x\ y$ **assume** $(x, y) \in r^\uparrow$
then obtain z **where** $(z, x) \in r^*$ **and** $(z, y) \in r^*$ **using** *meetD by best*
with $\langle CR\ r \rangle$ **show** $(x, y) \in r^\downarrow$ **by** (*auto simp: CR-defs*)
qed
next
assume $r^\uparrow \subseteq r^\downarrow$ {
fix $x\ y\ z$ **assume** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
then have $(y, z) \in r^\uparrow$ **unfolding** *meet-def rtrancl-converse by auto*
with $\langle r^\uparrow \subseteq r^\downarrow \rangle$ **have** $(y, z) \in r^\downarrow$ **by** *auto*
} **then show** $CR\ r$ **by** (*auto simp: CR-defs*)
qed

lemma *CR-divergence-imp-join*:
assumes $CR\ r$ **and** $(x, y) \in r^*$ **and** $(x, z) \in r^*$
shows $(y, z) \in r^\downarrow$
using *assms by auto*

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

proof
fix $x\ z$ **assume** $(x, z) \in r^\downarrow$
then obtain y **where** $(x, y) \in r^*$ **and** $(z, y) \in r^*$ **by** *auto*
then have $(x, y) \in r^{\leftrightarrow*}$ **and** $(z, y) \in r^{\leftrightarrow*}$ **by** *auto*
from $\langle (z, y) \in r^{\leftrightarrow*} \rangle$ **have** $(y, z) \in r^{\leftrightarrow*}$ **using** *conversion-sym unfolding sym-def*
by *best*

with $\langle (x, y) \in r^{\leftrightarrow*} \rangle$ **show** $(x, z) \in r^{\leftrightarrow*}$ **using** *conversion-trans unfolding trans-def* **by best**
qed

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

proof (*rule subrelI*)

fix $y z$ **assume** $(y, z) \in r^\uparrow$

then obtain x **where** $(x, y) \in r^*$ **and** $(x, z) \in r^*$ **by** *auto*

then have $(x, y) \in r^{\leftrightarrow*}$ **and** $(x, z) \in r^{\leftrightarrow*}$ **by** *auto*

from $\langle (x, y) \in r^{\leftrightarrow*} \rangle$ **have** $(y, x) \in r^{\leftrightarrow*}$ **using** *conversion-sym unfolding sym-def* **by best**

with $\langle (x, z) \in r^{\leftrightarrow*} \rangle$ **show** $(y, z) \in r^{\leftrightarrow*}$ **using** *conversion-trans unfolding trans-def* **by best**

qed

lemma *CR-imp-UNF*:

assumes *CR* r **shows** *UNF* r

proof – {

fix $x y z$ **assume** $(x, y) \in r^\downarrow$ **and** $(x, z) \in r^\downarrow$

then have $(x, y) \in r^*$ **and** $y \in NF\ r$ **and** $(x, z) \in r^*$ **and** $z \in NF\ r$

unfolding *normalizability-def* **by** *auto*

from *assms* **and** $\langle (x, y) \in r^* \rangle$ **and** $\langle (x, z) \in r^* \rangle$ **have** $(y, z) \in r^\downarrow$

by (*rule CR-divergence-imp-join*)

from *this* **and** $\langle y \in NF\ r \rangle$ **and** $\langle z \in NF\ r \rangle$ **have** $y = z$ **by** (*rule join-NF-imp-eq*)

} **then show** *?thesis* **by** *auto*

qed

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

proof (*intro iffI subrelI*)

fix $x y$ **assume** *CR* r **and** $(x, y) \in r^{\leftrightarrow*}$

then obtain n **where** $(x, y) \in (r^{\leftrightarrow})^{\sim n}$ **unfolding** *conversion-def rtrancl-is-UN-relpow* **by** *auto*

then show $(x, y) \in r^\downarrow$

proof (*induct n arbitrary: x*)

case 0

assume $(x, y) \in r^{\leftrightarrow} \sim 0$ **then have** $x = y$ **by** *simp*

show *?case* **unfolding** $\langle x = y \rangle$ **by** *auto*

next

case (*Suc* n)

from $\langle (x, y) \in r^{\leftrightarrow} \sim \text{Suc } n \rangle$ **obtain** z **where** $(x, z) \in r^{\leftrightarrow}$ **and** $(z, y) \in r^{\leftrightarrow} \sim n$

using *relpow-Suc-D2* **by** *best*

with *Suc* **have** $(z, y) \in r^\downarrow$ **by** *simp*

from $\langle (x, z) \in r^{\leftrightarrow} \rangle$ **show** *?case*

proof

assume $(x, z) \in r$ **with** $\langle (z, y) \in r^\downarrow \rangle$ **show** *?thesis* **by** (*auto intro: rtrancl-join-join*)

next

assume $(x, z) \in r^{-1}$

then have $(z, x) \in r^*$ **by** *simp*

from $\langle z, y \rangle \in r^\downarrow$ **obtain** z' **where** $\langle z, z' \rangle \in r^*$ **and** $\langle y, z' \rangle \in r^*$ **by** *auto*
from $\langle CR\ r \rangle$ **and** $\langle z, x \rangle \in r^*$ **and** $\langle z, z' \rangle \in r^*$ **have** $\langle x, z' \rangle \in r^\downarrow$
by (*rule CR-divergence-imp-join*)
then obtain x' **where** $\langle x, x' \rangle \in r^*$ **and** $\langle z', x' \rangle \in r^*$ **by** *auto*
with $\langle y, z' \rangle \in r^*$ **show** *?thesis* **by** *auto*
qed
qed
next
assume $r^{\leftrightarrow*} \subseteq r^\downarrow$ **then show** *CR r unfolding CR-iff-meet-subset-join*
using *meet-imp-conversion* **by** *auto*
qed

lemma *CR-imp-conversionIff-join*:
assumes *CR r* **shows** $r^{\leftrightarrow*} = r^\downarrow$
proof
show $r^{\leftrightarrow*} \subseteq r^\downarrow$ **using** *CR-iff-conversion-imp-join assms* **by** *auto*
next
show $r^\downarrow \subseteq r^{\leftrightarrow*}$ **by** (*rule join-imp-conversion*)
qed

lemma *sym-join*: *sym (join r)* **by** (*auto simp: sym-def*)

lemma *join-sym*: $\langle s, t \rangle \in A^\downarrow \implies \langle t, s \rangle \in A^\downarrow$ **by** *auto*

lemma *CR-join-left-I*:
assumes *CR r* **and** $\langle x, y \rangle \in r^*$ **and** $\langle x, z \rangle \in r^\downarrow$ **shows** $\langle y, z \rangle \in r^\downarrow$
proof –
from $\langle x, z \rangle \in r^\downarrow$ **obtain** x' **where** $\langle x, x' \rangle \in r^*$ **and** $\langle z, x' \rangle \in r^\downarrow$ **by** *auto*
from $\langle CR\ r \rangle$ **and** $\langle x, x' \rangle \in r^*$ **and** $\langle x, y \rangle \in r^*$ **have** $\langle x, y \rangle \in r^\downarrow$ **by** *auto*
then have $\langle y, x \rangle \in r^\downarrow$ **using** *join-sym* **by** *best*
from $\langle CR\ r \rangle$ **have** $r^{\leftrightarrow*} = r^\downarrow$ **by** (*rule CR-imp-conversionIff-join*)
from $\langle y, x \rangle \in r^\downarrow$ **and** $\langle x, z \rangle \in r^\downarrow$ **show** *?thesis* **using** *conversion-trans*
unfolding *trans-def* $\langle r^{\leftrightarrow*} = r^\downarrow \rangle$ [*symmetric*] **by** *best*
qed

lemma *CR-join-right-I*:
assumes *CR r* **and** $\langle x, y \rangle \in r^\downarrow$ **and** $\langle y, z \rangle \in r^*$ **shows** $\langle x, z \rangle \in r^\downarrow$
proof –
have $r^{\leftrightarrow*} = r^\downarrow$ **by** (*rule CR-imp-conversionIff-join [OF <CR r>]*)
from $\langle y, z \rangle \in r^*$ **have** $\langle y, z \rangle \in r^{\leftrightarrow*}$ **by** *auto*
with $\langle x, y \rangle \in r^\downarrow$ **show** *?thesis* **unfolding** $\langle r^{\leftrightarrow*} = r^\downarrow \rangle$ [*symmetric*] **using**
conversion-trans
unfolding *trans-def* **by** *fast*
qed

lemma *NF-not-suc*:
assumes $\langle x, y \rangle \in r^*$ **and** $x \in NF\ r$ **shows** $x = y$
proof –
from $\langle x \in NF\ r \rangle$ **have** $\forall y. \langle x, y \rangle \notin r$ **using** *NF-no-step* **by** *auto*

then have $x \notin \text{Domain } r$ **unfolding** *Domain-unfold* **by** *simp*
 from $\langle (x, y) \in r^* \rangle$ **show** *?thesis* **unfolding** *Not-Domain-rtrancl* [*OF* $\langle x \notin \text{Domain } r \rangle$] **by** *simp*
qed

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 –

from *assms* have *WN* r and *CR* r **unfolding** *semi-complete-defs* **by** *auto*

then have $r^{\leftrightarrow*} = r^\downarrow$ **using** *CR-imp-conversionIff-join* **by** *auto*

show *?thesis*

proof

assume $(x, y) \in r^{\leftrightarrow*}$

from $\langle (x, y) \in r^{\leftrightarrow*} \rangle$ have $(x, y) \in r^\downarrow$ **unfolding** $\langle r^{\leftrightarrow*} = r^\downarrow \rangle$.

show $\forall u v. (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v$

proof (*intro allI impI, elim conjE*)

fix $u v$ assume $(x, u) \in r^!$ and $(y, v) \in r^!$

then have $(x, u) \in r^*$ and $(y, v) \in r^*$ and $u \in \text{NF } r$ and $v \in \text{NF } r$ **by** *auto*

from $\langle \text{CR } r \rangle$ and $\langle (x, u) \in r^* \rangle$ and $\langle (y, v) \in r^* \rangle$ have $(u, v) \in r^\downarrow$

by (*auto intro: CR-join-left-I*)

then have $(y, u) \in r^\downarrow$ **using** *join-sym* **by** *best*

with $\langle (x, y) \in r^\downarrow \rangle$ have $(x, u) \in r^\downarrow$ **unfolding** $\langle r^{\leftrightarrow*} = r^\downarrow \rangle$ [*symmetric*]

using *conversion-trans* **unfolding** *trans-def* **by** *best*

from $\langle \text{CR } r \rangle$ and $\langle (x, y) \in r^\downarrow \rangle$ and $\langle (y, v) \in r^* \rangle$ have $(x, v) \in r^\downarrow$

by (*auto intro: CR-join-right-I*)

then have $(v, x) \in r^\downarrow$ **using** *join-sym* **unfolding** *sym-def* **by** *best*

with $\langle (x, u) \in r^\downarrow \rangle$ have $(v, u) \in r^\downarrow$ **unfolding** $\langle r^{\leftrightarrow*} = r^\downarrow \rangle$ [*symmetric*]

using *conversion-trans* **unfolding** *trans-def* **by** *best*

then obtain v' where $(v, v') \in r^*$ and $(u, v') \in r^*$ **by** *auto*

from $\langle (u, v') \in r^* \rangle$ and $\langle u \in \text{NF } r \rangle$ have $u = v'$ **by** (*rule NF-not-suc*)

from $\langle (v, v') \in r^* \rangle$ and $\langle v \in \text{NF } r \rangle$ have $v = v'$ **by** (*rule NF-not-suc*)

then show $u = v$ **unfolding** $\langle u = v' \rangle$ **by** *simp*

qed

next

assume *equal-NF*: $\forall u v. (x, u) \in r^! \wedge (y, v) \in r^! \longrightarrow u = v$

from $\langle \text{WN } r \rangle$ obtain u where $(x, u) \in r^!$ **by** *auto*

from $\langle \text{WN } r \rangle$ obtain v where $(y, v) \in r^!$ **by** *auto*

from $\langle (x, u) \in r^! \rangle$ and $\langle (y, v) \in r^! \rangle$ have $u = v$ **using** *equal-NF* **by** *simp*

from $\langle (x, u) \in r^! \rangle$ and $\langle (y, v) \in r^! \rangle$ have $(x, v) \in r^*$ and $(y, v) \in r^*$

unfolding $\langle u = v \rangle$ **by** *auto*

then have $(x, v) \in r^{\leftrightarrow*}$ and $(y, v) \in r^{\leftrightarrow*}$ **by** *auto*

from $\langle (y, v) \in r^{\leftrightarrow*} \rangle$ have $(v, y) \in r^{\leftrightarrow*}$ **using** *conversion-sym* **unfolding** *sym-def* **by** *best*

with $\langle (x, v) \in r^{\leftrightarrow*} \rangle$ **show** $(x, y) \in r^{\leftrightarrow*}$ **using** *conversion-trans* **unfolding** *trans-def* **by** *best*

qed

qed

lemma *CR-imp-UNC*:

assumes *CR* *r* shows *UNC* *r*

proof – {

fix *x y* assume $x \in NF\ r$ and $y \in NF\ r$ and $(x, y) \in r^{\leftrightarrow*}$

have $r^{\leftrightarrow*} = r^\downarrow$ by (rule *CR-imp-conversionIff-join* [*OF* *assms*])

from $\langle (x, y) \in r^{\leftrightarrow*} \rangle$ have $(x, y) \in r^\downarrow$ unfolding $\langle r^{\leftrightarrow*} = r^\downarrow \rangle$ by *simp*

then obtain *x'* where $(x, x') \in r^*$ and $(y, x') \in r^*$ by *best*

from $\langle (x, x') \in r^* \rangle$ and $\langle x \in NF\ r \rangle$ have $x = x'$ by (rule *NF-not-suc*)

from $\langle (y, x') \in r^* \rangle$ and $\langle y \in NF\ r \rangle$ have $y = x'$ by (rule *NF-not-suc*)

then have $x = y$ unfolding $\langle x = x' \rangle$ by *simp*

} then show *?thesis* by (auto *simp*: *UNC-def*)

qed

lemma *WN-UNF-imp-CR*:

assumes *WN* *r* and *UNF* *r* shows *CR* *r*

proof – {

fix *x y z* assume $(x, y) \in r^*$ and $(x, z) \in r^*$

from *assms* obtain *y'* where $(y, y') \in r^\downarrow$ unfolding *WN-defs* by *best*

with $\langle (x, y) \in r^* \rangle$ have $(x, y') \in r^\downarrow$ by *auto*

from *assms* obtain *z'* where $(z, z') \in r^\downarrow$ unfolding *WN-defs* by *best*

with $\langle (x, z) \in r^* \rangle$ have $(x, z') \in r^\downarrow$ by *auto*

with $\langle (x, y') \in r^\downarrow \rangle$ have $y' = z'$ using $\langle UNF\ r \rangle$ unfolding *UNF-defs* by *auto*

from $\langle (y, y') \in r^\downarrow \rangle$ and $\langle (z, z') \in r^\downarrow \rangle$ have $(y, z) \in r^\downarrow$ unfolding $\langle y' = z' \rangle$ by

auto

} then show *?thesis* by *auto*

qed

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

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

lemma *diamond-I* [*intro*]: $(r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \implies \diamond\ r$ unfolding *diamond-def* by *simp*

lemma *diamond-E* [*elim*]: $\diamond\ r \implies ((r^{-1}\ O\ r) \subseteq (r\ O\ r^{-1}) \implies P) \implies P$ unfolding *diamond-def* by *simp*

lemma *diamond-imp-semi-confluence*:

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

proof (rule *subrelI*)

fix *y z* assume $(y, z) \in r^{-1}\ O\ r^*$

then obtain *x* where $(x, y) \in r$ and $(x, z) \in r^*$ by *best*

then obtain *n* where $(x, z) \in r^{\sim n}$ using *rtrancl-imp-UN-relpow* by *best*

with $\langle (x, y) \in r \rangle$ show $(y, z) \in r^\downarrow$

proof (induct *n* arbitrary: $x\ z\ y$)

case 0 then show *?case* by *auto*

next

case (*Suc* *n*)

from $\langle (x, z) \in r^{\sim Suc\ n} \rangle$ obtain *x'* where $(x, x') \in r$ and $(x', z) \in r^{\sim n}$ using *relpow-Suc-D2* by *best*

with $\langle (x, y) \in r \rangle$ **have** $(y, x') \in (r^{-1} O r)$ **by** *auto*
with $\langle \diamond r \rangle$ **have** $(y, x') \in (r O r^{-1})$ **by** *auto*
then obtain y' **where** $(x', y') \in r$ **and** $(y, y') \in r$ **by** *best*
with *Suc* **and** $\langle (x', z) \in r \widetilde{n} \rangle$ **have** $(y', z) \in r^\downarrow$ **by** *auto*
with $\langle (y, y') \in r \rangle$ **show** *?case* **by** (*auto intro: rtrancl-join-join*)
qed
qed

lemma *semi-confluence-imp-CR*:

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

proof – {

fix $x y z$ **assume** $(x, y) \in r^*$ **and** $(x, z) \in r^*$

then obtain n **where** $(x, z) \in r \widetilde{n}$ **using** *rtrancl-imp-UN-relpow* **by** *best*

with $\langle (x, y) \in r^* \rangle$ **have** $(y, z) \in r^\downarrow$

proof (*induct n arbitrary: x y z*)

case 0 **then show** *?case* **by** *auto*

next

case (*Suc n*)

from $\langle (x, z) \in r \widetilde{Suc\ n} \rangle$ **obtain** x' **where** $(x, x') \in r$ **and** $(x', z) \in r \widetilde{n}$
using *relpow-Suc-D2* **by** *best*

from $\langle (x, x') \in r \rangle$ **and** $\langle (x, y) \in r^* \rangle$ **have** $(x', y) \in (r^{-1} O r^*)$ **by** *auto*

with *assms* **have** $(x', y) \in r^\downarrow$ **by** *auto*

then obtain y' **where** $(x', y') \in r^*$ **and** $(y, y') \in r^*$ **by** *best*

with *Suc* **and** $\langle (x', z) \in r \widetilde{n} \rangle$ **have** $(y', z) \in r^\downarrow$ **by** *simp*

then obtain u **where** $(z, u) \in r^*$ **and** $(y', u) \in r^*$ **by** *best*

from $\langle (y, y') \in r^* \rangle$ **and** $\langle (y', u) \in r^* \rangle$ **have** $(y, u) \in r^*$ **by** *auto*

with $\langle (z, u) \in r^* \rangle$ **show** *?case* **by** *best*

qed

} **then show** *?thesis* **by** *auto*

qed

lemma *diamond-imp-CR*:

assumes $\diamond r$ **shows** *CR* r

using *assms* **by** (*rule diamond-imp-semi-confluence [THEN semi-confluence-imp-CR]*)

lemma *diamond-imp-CR'*:

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

unfolding *CR-iff-meet-subset-join*

proof –

from $\langle \diamond s \rangle$ **have** *CR* s **by** (*rule diamond-imp-CR*)

then have $s^\uparrow \subseteq s^\downarrow$ **unfolding** *CR-iff-meet-subset-join* **by** *simp*

from $\langle r \subseteq s \rangle$ **have** $r^* \subseteq s^*$ **by** (*rule rtrancl-mono*)

from $\langle s \subseteq r^* \rangle$ **have** $s^* \subseteq (r^*)^*$ **by** (*rule rtrancl-mono*)

then have $s^* \subseteq r^*$ **by** *simp*

with $\langle r^* \subseteq s^* \rangle$ **have** $r^* = s^*$ **by** *simp*

show $r^\uparrow \subseteq r^\downarrow$ **unfolding** *meet-def join-def rtrancl-converse* $\langle r^* = s^* \rangle$

unfolding *rtrancl-converse [symmetric] meet-def [symmetric]*

join-def [symmetric] **by** (*rule* $\langle s^\uparrow \subseteq s^\downarrow \rangle$)

qed

lemma *SN-imp-minimal*:

assumes *SN A*

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

proof (*rule ccontr*)

assume $\neg (\forall Q. x \in Q \longrightarrow (\exists z \in Q. \forall y. (z, y) \in A \longrightarrow y \notin Q))$

then obtain *Q x* **where** $x \in Q$ **and** $\forall z \in Q. \exists y. (z, y) \in A \wedge y \in Q$ **by** *auto*

then have $\forall z. \exists y. z \in Q \longrightarrow (z, y) \in A \wedge y \in Q$ **by** *auto*

then have $\exists f. \forall x. x \in Q \longrightarrow (x, f x) \in A \wedge f x \in Q$ **by** (*rule choice*)

then obtain *f* **where** $a: \forall x. x \in Q \longrightarrow (x, f x) \in A \wedge f x \in Q$ (**is** $\forall x. ?P x$)

by *best*

let $?S = \lambda i. (f \sim i) x$

have $?S 0 = x$ **by** *simp*

have $\forall i. (?S i, ?S (Suc i)) \in A \wedge ?S (Suc i) \in Q$

proof

fix *i* **show** $(?S i, ?S (Suc i)) \in A \wedge ?S (Suc i) \in Q$

by (*induct i*) (*auto simp: <x ∈ Q> a*)

qed

with $<?S 0 = x>$ **have** $\exists S. S 0 = x \wedge \text{chain } A S$ **by** *fast*

with *assms* **show** *False* **by** *auto*

qed

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 (*rule ccontr*)

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

then obtain *Q* **where** $x \in Q$ **and** $\forall z \in Q. \exists y. (z, y) \in r \wedge y \in Q$ **by** *auto*

then have $\forall z. \exists y. z \in Q \longrightarrow (z, y) \in r \wedge y \in Q$ **by** *auto*

then have $\exists f. \forall x. x \in Q \longrightarrow (x, f x) \in r \wedge f x \in Q$ **by** (*rule choice*)

then obtain *f* **where** $a: \forall x. x \in Q \longrightarrow (x, f x) \in r \wedge f x \in Q$ (**is** $\forall x. ?P x$)

by *best*

let $?S = \lambda i. (f \sim i) x$

have $?S 0 = x$ **by** *simp*

have $\forall i. (?S i, ?S (Suc i)) \in r \wedge ?S (Suc i) \in Q$

proof

fix *i* **show** $(?S i, ?S (Suc i)) \in r \wedge ?S (Suc i) \in Q$ **by** (*induct i*) (*auto simp: <x ∈ Q> a*)

qed

with $<?S 0 = x>$ **have** $\exists S. S 0 = x \wedge \text{chain } r S$ **by** *fast*

with *assms* **show** *False* **by** *auto*

qed

lemma *minimal-imp-wf*:

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

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

proof (*rule ccontr*)

assume $\neg \text{wf}(r^{-1})$

then have $\exists P. (\forall y. (x, y) \in r \longrightarrow P y) \longrightarrow P x \wedge (\exists x. \neg P x)$ **unfolding**

wf-def **by** *simp*
then obtain $P x$ **where** $suc: \forall x. (\forall y. (x, y) \in r \longrightarrow P y) \longrightarrow P x$ **and** $\neg P x$
by *auto*
let $?Q = \{x. \neg P x\}$
from $\langle \neg P x \rangle$ **have** $x \in ?Q$ **by** *simp*
from *assms* **have** $\forall x. x \in ?Q \longrightarrow (\exists z \in ?Q. \forall y. (z, y) \in r \longrightarrow y \notin ?Q)$ **by** (*rule*
allE [**where** $x = ?Q$])
with $\langle x \in ?Q \rangle$ **obtain** z **where** $z \in ?Q$ **and** $min: \forall y. (z, y) \in r \longrightarrow y \notin ?Q$
by *best*
from $\langle z \in ?Q \rangle$ **have** $\neg P z$ **by** *simp*
with *suc* **obtain** y **where** $(z, y) \in r$ **and** $\neg P y$ **by** *best*
then have $y \in ?Q$ **by** *simp*
with $\langle (z, y) \in r \rangle$ **and** *min* **show** *False* **by** *simp*
qed

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

lemma *wf-imp-SN*:
assumes *wf* (A^{-1}) **shows** *SN* A
proof – {
fix a
let $?P = \lambda a. \neg(\exists S. S 0 = a \wedge chain A S)$
from $\langle wf (A^{-1}) \rangle$ **have** $?P a$
proof *induct*
case (*less* a)
then have *IH*: $\bigwedge b. (a, b) \in A \implies ?P b$ **by** *auto*
show $?P a$
proof (*rule ccontr*)
assume $\neg ?P a$
then obtain S **where** $S 0 = a$ **and** *chain* $A S$ **by** *auto*
then have $(S 0, S 1) \in A$ **by** *auto*
with *IH* **have** $?P (S 1)$ **unfolding** $\langle S 0 = a \rangle$ **by** *auto*
with $\langle chain A S \rangle$ **show** *False* **by** *auto*
qed
qed
then have *SN-on* $A \{a\}$ **unfolding** *SN-defs* **by** *auto*
} **then show** *?thesis* **by** *fast*
qed

lemma *SN-nat-gt*: *SN* $\{(a, b :: nat) . a > b\}$
proof –
from *wf-less* **have** *wf* $(\{(x, y) . (x :: nat) > y\}^{-1})$ **unfolding** *converse-unfold*
by *auto*
from *wf-imp-SN* [*OF this*] **show** *?thesis* .
qed

lemma *SN-iff-wf*: *SN* $A = wf (A^{-1})$ **by** (*auto simp: SN-imp-wf wf-imp-SN*)

lemma *SN-imp-acyclic*: $SN\ R \implies acyclic\ R$
using *wf-acyclic* [of R^{-1} , unfolded *SN-iff-wf* [symmetric]] **by** *auto*

lemma *SN-induct*:
assumes *sn*: $SN\ r$ **and** *step*: $\bigwedge a. (\bigwedge b. (a, b) \in r \implies P\ b) \implies P\ a$
shows $P\ a$
using *sn* **unfolding** *SN-iff-wf* **proof** *induct*
case (*less a*)
with *step* **show** *?case* **by** *best*
qed

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*: $SN\text{-on}\ R\ A$
and $s \in A$
and *imp*: $\bigwedge t. (\bigwedge u. (t, u) \in R \implies P\ u) \implies P\ t$
shows $P\ s$
proof –
let $?R = restrict\text{-}SN\ R\ R$
let $?P = \lambda t. SN\text{-on}\ R\ \{t\} \longrightarrow P\ t$
have $SN\text{-on}\ R\ \{s\} \longrightarrow P\ s$
proof (*rule* *SN-induct* [OF *SN-restrict-SN-idemp* [of R], of $?P$])
fix a
assume *ind*: $\bigwedge b. (a, b) \in ?R \implies SN\text{-on}\ R\ \{b\} \longrightarrow P\ b$
show $SN\text{-on}\ R\ \{a\} \longrightarrow P\ a$
proof
assume *SN*: $SN\text{-on}\ R\ \{a\}$
show $P\ a$
proof (*rule* *imp*)
fix b
assume $(a, b) \in R$
with *SN* *step-preserves-SN-on* [OF *this* *SN*]
show $P\ b$ **using** *ind* [of b] **unfolding** *restrict-SN-def* **by** *auto*
qed
qed
qed
with *SN* **show** $P\ s$ **using** $\langle s \in A \rangle$ **unfolding** *SN-on-def* **by** *blast*
qed

lemma *accp-imp-SN-on*:
assumes $\bigwedge x. x \in A \implies Wellfounded.\ accp\ g\ x$
shows $SN\text{-on}\ \{(y, z). g\ z\ y\}\ A$
proof – {
fix x **assume** $x \in A$
from *assms* [OF *this*]

```

have SN-on  $\{(y, z). g z y\} \{x\}$ 
proof (induct rule: accp.induct)
  case (accI x)
  show ?case
  proof
    fix f
    assume x: f 0  $\in \{x\}$  and steps:  $\forall i. (f i, f (Suc i)) \in \{a. (\lambda(y, z). g z y) a\}$ 
    then have g (f 1) x by auto
    from accI(2)[OF this] steps x show False unfolding SN-on-def by auto
  qed
qed
}
then show ?thesis unfolding SN-on-def by blast
qed

```

```

lemma SN-on-imp-accp:
  assumes SN-on  $\{(y, z). g z y\} A$ 
  shows  $\forall x \in A. Wellfounded.accp\ g\ x$ 
proof
  fix x assume x  $\in A$ 
  with assms show Wellfounded.accp g x
  proof (induct rule: SN-on-induct)
    case (IH x)
    show ?case
    proof
      fix y
      assume g y x
      with IH show Wellfounded.accp g y by simp
    qed
  qed
qed

```

```

lemma SN-on-conv-accp:
  SN-on  $\{(y, z). g z y\} \{x\} = Wellfounded.accp\ g\ x$ 
  using SN-on-imp-accp [of g {x}]
  accp-imp-SN-on [of {x} g]
  by auto

```

```

lemma SN-on-conv-acc: SN-on  $\{(y, z). (z, y) \in r\} \{x\} \longleftrightarrow x \in Wellfounded.acc\ r$ 
unfolding SN-on-conv-accp accp-acc-eq ..

```

```

lemma acc-imp-SN-on:
  assumes x  $\in Wellfounded.acc\ r$  shows SN-on  $\{(y, z). (z, y) \in r\} \{x\}$ 
  using assms unfolding SN-on-conv-acc by simp

```

```

lemma SN-on-imp-acc:
  assumes SN-on  $\{(y, z). (z, y) \in r\} \{x\}$  shows x  $\in Wellfounded.acc\ r$ 
  using assms unfolding SN-on-conv-acc by simp

```

2.3 Newman's Lemma

lemma *rtrancl-len-E* [elim]:

assumes $(x, y) \in r^*$ **obtains** n **where** $(x, y) \in r^{\sim n}$

using *rtrancl-imp-UN-relpow* [OF assms] **by** *best*

lemma *relpow-Suc-E2'* [elim]:

assumes $(x, z) \in A^{\sim \text{Suc } n}$ **obtains** y **where** $(x, y) \in A$ **and** $(y, z) \in A^*$

proof –

assume *assm*: $\bigwedge y. (x, y) \in A \implies (y, z) \in A^* \implies \text{thesis}$

from *relpow-Suc-E2* [OF assms] **obtain** y **where** $(x, y) \in A$ **and** $(y, z) \in A^{\sim n}$

by *auto*

then have $(y, z) \in A^*$ **using** *relpow-imp-rtrancl* **by** *auto*

from *assm* [OF $\langle (x, y) \in A \rangle$ *this*] **show** *thesis* .

qed

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

lemma *Newman-local*:

assumes *SN-on* r X **and** *WCR*: *WCR-on* r $\{x. \text{SN-on } r \{x\}\}$

shows *CR-on* r X

proof – {

fix x

assume $x \in X$

with *assms* **have** *SN-on* r $\{x\}$ **unfolding** *SN-on-def* **by** *auto*

with *this* **have** *CR-on* r $\{x\}$

proof (*induct rule: SN-on-induct'*)

case (*IH* x) **show** *?case*

proof

fix $y z$ **assume** $(x, y) \in r^*$ **and** $(x, z) \in r^*$

from $\langle (x, y) \in r^* \rangle$ **obtain** m **where** $(x, y) \in r^{\sim m}$..

from $\langle (x, z) \in r^* \rangle$ **obtain** n **where** $(x, z) \in r^{\sim n}$..

show $(y, z) \in r^\downarrow$

proof (*cases* n)

case 0

from $\langle (x, z) \in r^{\sim n} \rangle$ **have** *eq: x = z* **by** (*simp add: 0*)

from $\langle (x, y) \in r^* \rangle$ **show** *?thesis* **unfolding** *eq* ..

next

case (*Suc* n')

from $\langle (x, z) \in r^{\sim n} \rangle$ [*unfolded Suc*] **obtain** t **where** $(x, t) \in r$ **and** $(t, z) \in r^*$..

show *?thesis*

proof (*cases* m)

case 0

from $\langle (x, y) \in r^{\sim m} \rangle$ **have** *eq: x = y* **by** (*simp add: 0*)

from $\langle (x, z) \in r^* \rangle$ **show** *?thesis* **unfolding** *eq* ..

next

case (*Suc* m')

from $\langle (x, y) \in r^{\sim m} \rangle$ [*unfolded Suc*] **obtain** s **where** $(x, s) \in r$ **and** $(s,$

lemma *step-reflects-SN-on*:
assumes $(\bigwedge b. (a, b) \in r \implies \text{SN-on } r \{b\})$
shows $\text{SN-on } r \{a\}$
using *assms* **and** *Image-SN-on [of r {a}]* **by** (*auto simp: SN-defs*)

lemma *SN-on-all-reducts-SN-on-conv*:
 $\text{SN-on } r \{a\} = (\forall b. (a, b) \in r \longrightarrow \text{SN-on } r \{b\})$
using *SN-on-Image-conv [of r {a}]* **by** (*auto simp: SN-defs*)

lemma *SN-imp-SN-trancl*: $\text{SN } R \implies \text{SN } (R^+)$
unfolding *SN-iff-wf* **by** (*rule wf-converse-trancl*)

lemma *SN-trancl-imp-SN*:
assumes $\text{SN } (R^+)$ **shows** $\text{SN } R$
using *assms* **by** (*rule SN-on-trancl-imp-SN-on*)

lemma *SN-trancl-SN-conv*: $\text{SN } (R^+) = \text{SN } R$
using *SN-trancl-imp-SN [of R]* *SN-imp-SN-trancl [of R]* **by** *blast*

lemma *SN-inv-image*: $\text{SN } R \implies \text{SN } (\text{inv-image } R \ f)$ **unfolding** *SN-iff-wf* **by** *simp*

lemma *SN-subset*: $\text{SN } R \implies R' \subseteq R \implies \text{SN } R'$ **unfolding** *SN-defs* **by** *blast*

lemma *SN-pow-imp-SN*:
assumes $\text{SN } (A \rightsquigarrow \text{Suc } n)$ **shows** $\text{SN } A$
proof (*rule ccontr*)
assume $\neg \text{SN } A$
then obtain S **where** *chain A S* **unfolding** *SN-defs* **by** *auto*
from *chain-imp-relpow [OF this]*
have *step*: $\bigwedge i. (S \ i, S \ (i + (\text{Suc } n))) \in A \rightsquigarrow \text{Suc } n .$
let $?T = \lambda i. S \ (i * (\text{Suc } n))$
have *chain* $(A \rightsquigarrow \text{Suc } n) \ ?T$
proof
fix i **show** $(?T \ i, ?T \ (\text{Suc } i)) \in A \rightsquigarrow \text{Suc } n$ **unfolding** *mult-Suc*
using *step [of i * Suc n]* **by** (*simp only: add.commute*)
qed
then have $\neg \text{SN } (A \rightsquigarrow \text{Suc } n)$ **unfolding** *SN-defs* **by** *fast*
with *assms* **show** *False* **by** *simp*
qed

lemma *pow-Suc-subset-trancl*: $R \rightsquigarrow (\text{Suc } n) \subseteq R^+$
using *trancl-power [of - R]* **by** *blast*

lemma *SN-imp-SN-pow*:
assumes $\text{SN } R$ **shows** $\text{SN } (R \rightsquigarrow \text{Suc } n)$
using *SN-subset [where R=R+, OF SN-imp-SN-trancl [OF assms] pow-Suc-subset-trancl]*
by *simp*

lemma *SN-pow*: $SN\ R \longleftrightarrow SN\ (R \rightsquigarrow Suc\ n)$
 by (rule *iffI*, rule *SN-imp-SN-pow*, assumption, rule *SN-pow-imp-SN*, assumption)

lemma *SN-on-trancl*:
 assumes *SN-on* $r\ A$ shows *SN-on* $(r^+)\ A$
 using *assms*
proof (rule *contrapos-pp*)
 let $?r = restrict\ SN\ r\ r$
 assume $\neg\ SN\text{-on}\ (r^+)\ A$
 then obtain f where $f\ 0 \in A$ and *chain*: *chain* $(r^+)\ f$ by *auto*
 have *SN* $?r$ by (rule *SN-restrict-SN-idemp*)
 then have *SN* $(?r^+)$ by (rule *SN-imp-SN-trancl*)
 have $\forall i. (f\ 0, f\ i) \in r^*$
proof
 fix i show $(f\ 0, f\ i) \in r^*$
proof (*induct* i)
 case 0 show *?case* ..
 next
 case (*Suc* i)
 from *chain* have $(f\ i, f\ (Suc\ i)) \in r^+ ..$
 with *Suc* show *?case* by *auto*
 qed
 qed
 with *assms* have $\forall i. SN\text{-on}\ r\ \{f\ i\}$
 using *steps-preserve-SN-on* [*of* $f\ 0 - r$]
 and $\langle f\ 0 \in A \rangle$
 and *SN-on-subset2* [*of* $\{f\ 0\}\ A$] by *auto*
 with *chain* have *chain* $(?r^+)\ f$
 unfolding *restrict-SN-trancl-simp*
 unfolding *restrict-SN-def* by *auto*
 then have $\neg\ SN\text{-on}\ (?r^+)\ \{f\ 0\}$ by *auto*
 with $\langle SN\ (?r^+) \rangle$ have *False* by (*simp* *add*: *SN-defs*)
 then show $\neg\ SN\text{-on}\ r\ A$ by *simp*
 qed

lemma *SN-on-trancl-SN-on-conv*: $SN\text{-on}\ (R^+)\ T = SN\text{-on}\ R\ T$
 using *SN-on-trancl-imp-SN-on* [*of* R] *SN-on-trancl* [*of* R] by *blast*

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* $?r\ A$)
proof (rule *ccontr*)

assume \neg *SN-on* $?r$ *A*
then have $\exists f. f 0 \in A \wedge \text{chain } ?r f$ **by** *auto*
then have $\exists f. f 0 \in A \wedge \text{chain } r f$ **unfolding** *restrict-def* **by** *auto*
with $\langle \text{SN-on } r A \rangle$ **show** *False* **by** *auto*
qed

lemma *restrict-rtrancl*: $(\text{restrict } r S)^* \subseteq r^*$ (**is** $?r^* \subseteq r^*$)
proof – {
 fix $x y$ **assume** $(x, y) \in ?r^*$ **then have** $(x, y) \in r^*$ **unfolding** *restrict-def* **by**
 induct auto
} **then show** *?thesis* **by** *auto*
qed

lemma *rtrancl-Image-step*:
 assumes $a \in r^*$ “ *A*
 and $(a, b) \in r^*$
 shows $b \in r^*$ “ *A*
proof –
 from *assms(1)* **obtain** c **where** $c \in A$ **and** $(c, a) \in r^*$ **by** *auto*
 with *assms* **have** $(c, b) \in r^*$ **by** *auto*
 with $\langle c \in A \rangle$ **show** *?thesis* **by** *auto*
qed

lemma *WCR-SN-on-imp-CR-on*:
 assumes *WCR* r **and** *SN-on* r *A* **shows** *CR-on* r *A*
proof –
 let $?S = r^*$ “ *A*
 let $?r = \text{restrict } r ?S$
 have $\forall x. \text{SN-on } ?r \{x\}$
 proof
 fix y **have** $y \notin ?S \vee y \in ?S$ **by** *simp*
 then show *SN-on* $?r \{y\}$
 proof
 assume $y \notin ?S$ **then show** *?thesis* **unfolding** *restrict-def* **by** *auto*
 next
 assume $y \in ?S$
 then have $y \in r^*$ “ *A* **by** *simp*
 with *SN-on-Image-rtrancl* [*OF* $\langle \text{SN-on } r A \rangle$]
 have *SN-on* $r \{y\}$ **using** *SN-on-subset2* [*of* $\{y\}$ r^* “ *A*] **by** *blast*
 then show *?thesis* **by** (*rule* *SN-on-restrict*)
 qed
qed
then have *SN* $?r$ **unfolding** *SN-defs* **by** *auto*
 {
 fix $x y$ **assume** $(x, y) \in r^*$ **and** $x \in ?S$ **and** $y \in ?S$
 then obtain n **where** $(x, y) \in r^{\sim n}$ **and** $x \in ?S$ **and** $y \in ?S$
 using *rtrancl-imp-UN-relpow* **by** *best*
 then have $(x, y) \in ?r^*$
 proof (*induct* n *arbitrary: x y*)
 }

```

    case 0 then show ?case by simp
  next
  case (Suc n)
  from ⟨(x, y) ∈ r~Suc n⟩ obtain x' where (x, x') ∈ r and (x', y) ∈ r~n
    using relpow-Suc-D2 by best
  then have (x, x') ∈ r* by simp
  with ⟨x ∈ ?S⟩ have x' ∈ ?S by (rule rtrancl-Image-step)
  with Suc and ⟨(x', y) ∈ r~n⟩ have (x', y) ∈ ?r* by simp
  from ⟨(x, x') ∈ r⟩ and ⟨x ∈ ?S⟩ and ⟨x' ∈ ?S⟩ have (x, x') ∈ ?r
    unfolding restrict-def by simp
  with ⟨(x', y) ∈ ?r*⟩ show ?case by simp
qed
}
then have a: ∀ x y. (x, y) ∈ r* ∧ x ∈ ?S ∧ y ∈ ?S ⟶ (x, y) ∈ ?r* by simp
{
  fix x' y z assume (x', y) ∈ ?r and (x', z) ∈ ?r
  then have x' ∈ ?S and y ∈ ?S and z ∈ ?S and (x', y) ∈ r and (x', z) ∈ r
    unfolding restrict-def by auto
  with ⟨WCR r⟩ have (y, z) ∈ r↓ by auto
  then obtain u where (y, u) ∈ r* and (z, u) ∈ r* by auto
  from ⟨x' ∈ ?S⟩ obtain x where x ∈ A and (x, x') ∈ r* by auto
  from ⟨(x', y) ∈ r⟩ have (x', y) ∈ r* by auto
  with ⟨(y, u) ∈ r*⟩ have (x', u) ∈ r* by auto
  with ⟨(x, x') ∈ r*⟩ have (x, u) ∈ r* by simp
  then have u ∈ ?S using ⟨x ∈ A⟩ by auto
  from ⟨y ∈ ?S⟩ and ⟨u ∈ ?S⟩ and ⟨(y, u) ∈ r*⟩ have (y, u) ∈ ?r* using a by
  auto
  from ⟨z ∈ ?S⟩ and ⟨u ∈ ?S⟩ and ⟨(z, u) ∈ r*⟩ have (z, u) ∈ ?r* using a by
  auto
  with ⟨(y, u) ∈ ?r*⟩ have (y, z) ∈ ?r↓ by auto
}
then have WCR ?r by auto
have CR ?r using Newman [OF ⟨SN ?r⟩ ⟨WCR ?r⟩] by simp
{
  fix x y z assume x ∈ A and (x, y) ∈ r* and (x, z) ∈ r*
  then have y ∈ ?S and z ∈ ?S by auto
  have x ∈ ?S using ⟨x ∈ A⟩ by auto
  from a and ⟨(x, y) ∈ r*⟩ and ⟨x ∈ ?S⟩ and ⟨y ∈ ?S⟩ have (x, y) ∈ ?r* by
  simp
  from a and ⟨(x, z) ∈ r*⟩ and ⟨x ∈ ?S⟩ and ⟨z ∈ ?S⟩ have (x, z) ∈ ?r* by
  simp
  with ⟨CR ?r⟩ and ⟨(x, y) ∈ ?r*⟩ have (y, z) ∈ ?r↓ by auto
  then obtain u where (y, u) ∈ ?r* and (z, u) ∈ ?r* by best
  then have (y, u) ∈ r* and (z, u) ∈ r* using restrict-rtrancl by auto
  then have (y, z) ∈ r↓ by auto
}
}
then show ?thesis by auto
qed

```

lemma *SN-on-Image-normalizable*:

assumes *SN-on* r A

shows $\forall a \in A. \exists b. b \in r^! \text{ `` } A$

proof

fix a **assume** $a: a \in A$

show $\exists b. b \in r^! \text{ `` } A$

proof (*rule ccontr*)

assume $\neg (\exists b. b \in r^! \text{ `` } A)$

then have $A: \forall b. (a, b) \in r^* \longrightarrow b \notin NF\ r$ **using** a **by** *auto*

then have $a \notin NF\ r$ **by** *auto*

let $?Q = \{c. (a, c) \in r^* \wedge c \notin NF\ r\}$

have $a \in ?Q$ **using** $\langle a \notin NF\ r \rangle$ **by** *simp*

have $\forall c \in ?Q. \exists b. (c, b) \in r \wedge b \in ?Q$

proof

fix c

assume $c \in ?Q$

then have $(a, c) \in r^*$ **and** $c \notin NF\ r$ **by** *auto*

then obtain d **where** $(c, d) \in r$ **by** *auto*

with $\langle (a, c) \in r^* \rangle$ **have** $(a, d) \in r^*$ **by** *simp*

with A **have** $d \notin NF\ r$ **by** *simp*

with $\langle (c, d) \in r \rangle$ **and** $\langle (a, c) \in r^* \rangle$

show $\exists b. (c, b) \in r \wedge b \in ?Q$ **by** *auto*

qed

with $\langle a \in ?Q \rangle$ **have** $a \in ?Q \wedge (\forall c \in ?Q. \exists b. (c, b) \in r \wedge b \in ?Q)$ **by** *auto*

then have $\exists Q. a \in Q \wedge (\forall c \in Q. \exists b. (c, b) \in r \wedge b \in Q)$ **by** (*rule exI [of - ?Q]*)

then have $\neg (\forall Q. a \in Q \longrightarrow (\exists c \in Q. \forall b. (c, b) \in r \longrightarrow b \notin Q))$ **by** *simp*

with *SN-on-imp-on-minimal [of r a]* **have** $\neg SN\text{-on}\ r\ \{a\}$ **by** *blast*

with *assms* **and** $\langle a \in A \rangle$ **and** *SN-on-subset2 [of {a} A r]* **show** *False* **by** *simp*

qed

qed

lemma *SN-on-imp-normalizability*:

assumes *SN-on* $r\ \{a\}$ **shows** $\exists b. (a, b) \in r^!$

using *SN-on-Image-normalizable [OF assms]* **by** *auto*

2.4 Commutation

definition *commute* $:: 'a\ rel \Rightarrow 'a\ rel \Rightarrow bool$ **where**

commute $r\ s \longleftrightarrow ((r^{-1})^* \circ s^*) \subseteq (s^* \circ (r^{-1})^*)$

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

unfolding *commute-def* *CR-iff-meet-subset-join* *meet-def* *join-def*

by *simp*

lemma *rtrancl-imp-rtrancl-UN*:

assumes $(x, y) \in r^*$ **and** $r \in I$

shows $(x, y) \in (\bigcup r \in I. r)^*$ **(is** $(x, y) \in ?r^*$ **)**

```

using assms proof induct
  case base then show ?case by simp
next
  case (step y z)
  then have  $(x, y) \in ?r^*$  by simp
  from  $\langle (y, z) \in r \rangle$  and  $\langle r \in I \rangle$  have  $(y, z) \in ?r^*$  by auto
  with  $\langle (x, y) \in ?r^* \rangle$  show ?case by auto
qed

```

definition *quasi-commute* :: $'a \text{ rel} \Rightarrow 'a \text{ rel} \Rightarrow \text{bool}$ **where**
quasi-commute $r \ s \longleftrightarrow (s \ O \ r) \subseteq r \ O \ (r \cup s)^*$

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

```

show  $(r \cup s^+)^* \subseteq (r \cup s)^*$ 
proof (rule subrelI)
  fix x y assume  $(x, y) \in (r \cup s^+)^*$ 
  then show  $(x, y) \in (r \cup s)^*$ 
  proof (induct)
    case base then show ?case by auto
  next
    case (step y z)
    then have  $(y, z) \in r \vee (y, z) \in s^+$  by auto
    then have  $(y, z) \in (r \cup s)^*$ 
    proof
      assume  $(y, z) \in r$  then show ?thesis by auto
    next
      assume  $(y, z) \in s^+$ 
      then have  $(y, z) \in s^*$  by auto
      then have  $(y, z) \in r^* \cup s^*$  by auto
      then show ?thesis using rtrancl-Un-subset by auto
    qed
  with  $\langle (x, y) \in (r \cup s)^* \rangle$  show ?case by simp
  qed
qed
next
show  $(r \cup s)^* \subseteq (r \cup s^+)^*$ 
proof (rule subrelI)
  fix x y assume  $(x, y) \in (r \cup s)^*$ 
  then show  $(x, y) \in (r \cup s^+)^*$ 
  proof (induct)
    case base then show ?case by auto
  next
    case (step y z)
    then have  $(y, z) \in (r \cup s^+)^*$  by auto
    with  $\langle (x, y) \in (r \cup s)^* \rangle$  show ?case by auto
  qed
qed
qed

```

lemma *qc-imp-qc-trancl*:
assumes *quasi-commute r s* **shows** *quasi-commute r (s⁺)*
unfolding *quasi-commute-def*
proof (*rule subrelI*)
fix *x z* **assume** $(x, z) \in s^+ O r$
then obtain *y* **where** $(x, y) \in s^+$ **and** $(y, z) \in r$ **by** *best*
then show $(x, z) \in r O (r \cup s^+)^*$
proof (*induct arbitrary: z*)
case (*base y*)
then have $(x, z) \in (s O r)$ **by** *auto*
with *assms* **have** $(x, z) \in r O (r \cup s)^*$ **unfolding** *quasi-commute-def* **by** *auto*
then show *?case* **using** *rtrancl-union-subset-rtrancl-union-trancl* **by** *auto*
next
case (*step a b*)
then have $(a, z) \in (s O r)$ **by** *auto*
with *assms* **have** $(a, z) \in r O (r \cup s)^*$ **unfolding** *quasi-commute-def* **by** *auto*
then obtain *u* **where** $(a, u) \in r$ **and** $(u, z) \in (r \cup s)^*$ **by** *best*
then have $(u, z) \in (r \cup s^+)^*$ **using** *rtrancl-union-subset-rtrancl-union-trancl*
by *auto*
from $\langle (a, u) \in r \rangle$ **and** *step* **have** $(x, u) \in r O (r \cup s^+)^*$ **by** *auto*
then obtain *v* **where** $(x, v) \in r$ **and** $(v, u) \in (r \cup s^+)^*$ **by** *best*
with $\langle (u, z) \in (r \cup s^+)^* \rangle$ **have** $(v, z) \in (r \cup s^+)^*$ **by** *auto*
with $\langle (x, v) \in r \rangle$ **show** *?case* **by** *auto*
qed
qed

lemma *steps-reflect-SN-on*:
assumes $\neg SN\text{-on } r \{b\}$ **and** $(a, b) \in r^*$
shows $\neg SN\text{-on } r \{a\}$
using *SN-on-Image-rtrancl* [*of r {a}*]
and *assms* **and** *SN-on-subset2* [*of {b} r** “*{a} r*”] **by** *blast*

lemma *chain-imp-not-SN-on*:
assumes *chain r f*
shows $\neg SN\text{-on } r \{f i\}$
proof –
let $?f = \lambda j. f (i + j)$
have $?f 0 \in \{f i\}$ **by** *simp*
moreover **have** *chain r ?f* **using** *assms* **by** *auto*
ultimately **have** $?f 0 \in \{f i\} \wedge \text{chain } r ?f$ **by** *blast*
then **have** $\exists g. g 0 \in \{f i\} \wedge \text{chain } r g$ **by** (*rule exI* [*of - ?f*])
then show *?thesis* **unfolding** *SN-defs* **by** *auto*
qed

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

have *quasi-commute* $r (s^+)$ **by** (*rule qc-imp-qc-trancl* [*OF* \langle *quasi-commute* $r s$ \rangle])
let $?B = \{a. \neg SN\text{-on} (r \cup s) \{a\}\}$
{
 assume $\neg SN(r \cup s)$
 then obtain a **where** $a \in ?B$ **unfolding** *SN-defs* **by** *fast*
 from $\langle SN r \rangle$ **have** $\forall Q x. x \in Q \longrightarrow (\exists z \in Q. \forall y. (z, y) \in r \longrightarrow y \notin Q)$
 by (*rule SN-imp-minimal*)
 then have $\forall x. x \in ?B \longrightarrow (\exists z \in ?B. \forall y. (z, y) \in r \longrightarrow y \notin ?B)$ **by** (*rule spec*
 [*where* $x = ?B$])
 with $\langle a \in ?B \rangle$ **obtain** b **where** $b \in ?B$ **and** $min: \forall y. (b, y) \in r \longrightarrow y \notin ?B$
by *auto*
 from $\langle b \in ?B \rangle$ **obtain** S **where** $S 0 = b$ **and**
 chain: chain $(r \cup s) S$ **unfolding** *SN-on-def* **by** *auto*
 let $?S = \lambda i. S(Suc i)$
 have $?S 0 = S 1$ **by** *simp*
 from *chain* **have** *chain* $(r \cup s) ?S$ **by** *auto*
 with $\langle ?S 0 = S 1 \rangle$ **have** $\neg SN\text{-on} (r \cup s) \{S 1\}$ **unfolding** *SN-on-def* **by** *auto*
 from $\langle S 0 = b \rangle$ **and** *chain* **have** $(b, S 1) \in r \cup s$ **by** *auto*
 with min **and** $\langle \neg SN\text{-on} (r \cup s) \{S 1\} \rangle$ **have** $(b, S 1) \in s$ **by** *auto*
 let $?i = LEAST i. (S i, S(Suc i)) \notin s$
 {
 assume *chain* $s S$
 with $\langle S 0 = b \rangle$ **have** $\neg SN\text{-on} s \{b\}$ **unfolding** *SN-on-def* **by** *auto*
 with $\langle SN s \rangle$ **have** *False* **unfolding** *SN-defs* **by** *auto*
 }
 then have $ex: \exists i. (S i, S(Suc i)) \notin s$ **by** *auto*
 then have $(S ?i, S(Suc ?i)) \notin s$ **by** (*rule LeastI-ex*)
 with *chain* **have** $(S ?i, S(Suc ?i)) \in r$ **by** *auto*
 have $ini: \forall i < ?i. (S i, S(Suc i)) \in s$ **using** *not-less-Least* **by** *auto*
 {
 fix i **assume** $i < ?i$ **then have** $(b, S(Suc i)) \in s^+$
 proof (*induct i*)
 case 0 **then show** $?case$ **using** $\langle (b, S 1) \in s \rangle$ **and** $\langle S 0 = b \rangle$ **by** *auto*
 next
 case $(Suc k)$
 then have $(b, S(Suc k)) \in s^+$ **and** $Suc k < ?i$ **by** *auto*
 with $\langle \forall i < ?i. (S i, S(Suc i)) \in s \rangle$ **have** $(S(Suc k), S(Suc(Suc k))) \in s$ **by**
 fast
 with $\langle (b, S(Suc k)) \in s^+ \rangle$ **show** $?case$ **by** *auto*
 qed
 }
 then have $pref: \forall i < ?i. (b, S(Suc i)) \in s^+$ **by** *auto*
 from $\langle (b, S 1) \in s \rangle$ **and** $\langle S 0 = b \rangle$ **have** $(S 0, S(Suc 0)) \in s$ **by** *auto*
 {
 assume $?i = 0$
 from ex **have** $(S ?i, S(Suc ?i)) \notin s$ **by** (*rule LeastI-ex*)
 with $\langle (S 0, S(Suc 0)) \in s \rangle$ **have** *False* **unfolding** $\langle ?i = 0 \rangle$ **by** *simp*
 }
 then have $0 < ?i$ **by** *auto*

then obtain j where $?i = \text{Suc } j$ unfolding gr0-conv-Suc by best
with ini have $(S(?i - \text{Suc } 0), S(\text{Suc} (?i - \text{Suc } 0))) \in s$ by auto
with pref have $(b, S(\text{Suc } j)) \in s^+$ unfolding $\langle ?i = \text{Suc } j \rangle$ by auto
then have $(b, S ?i) \in s^+$ unfolding $\langle ?i = \text{Suc } j \rangle$ by auto
with $\langle (S ?i, S(\text{Suc } ?i)) \in r \rangle$ have $(b, S(\text{Suc } ?i)) \in (s^+ O r)$ by auto
with $\langle \text{quasi-commute } r (s^+) \rangle$ have $(b, S(\text{Suc } ?i)) \in r O (r \cup s^+)^*$
unfolding quasi-commute-def by auto
then obtain c where $(b, c) \in r$ and $(c, S(\text{Suc } ?i)) \in (r \cup s^+)^*$ by best
from $\langle (b, c) \in r \rangle$ have $(b, c) \in (r \cup s^+)^*$ by auto
from $\text{chain-imp-not-SN-on}$ [of $S r \cup s$]
and chain have $\neg \text{SN-on } (r \cup s) \{S (\text{Suc } ?i)\}$ by auto
from $\langle (c, S(\text{Suc } ?i)) \in (r \cup s^+)^* \rangle$ have $(c, S(\text{Suc } ?i)) \in (r \cup s)^*$
unfolding $\text{rtrancl-union-subset-rtrancl-union-trancl}$ by auto
with $\text{steps-reflect-SN-on}$ [of $r \cup s$]
and $\langle \neg \text{SN-on } (r \cup s) \{S(\text{Suc } ?i)\} \rangle$ have $\neg \text{SN-on } (r \cup s) \{c\}$ by auto
then have $c \in ?B$ by simp
with $\langle (b, c) \in r \rangle$ and min have False by auto
}
then show $?thesis$ by auto
qed

2.5 Strong Normalization

lemma $\text{non-strict-into-strict}$:

assumes $\text{compat}: NS O S \subseteq S$
and $\text{steps}: (s, t) \in (NS^*) O S$
shows $(s, t) \in S$

using steps proof

fix $x u z$

assume $(s, t) = (x, z)$ and $(x, u) \in NS^*$ and $(u, z) \in S$

then have $(s, u) \in NS^*$ and $(u, t) \in S$ by auto

then show $?thesis$

proof (induct rule: rtrancl.induct)

case ($\text{rtrancl-refl } x$) then show $?case$.

next

case ($\text{rtrancl-into-rtrancl } a b c$)

with compat show $?case$ by auto

qed

qed

lemma comp-trancl :

assumes $R O S \subseteq S$ shows $R O S^+ \subseteq S^+$

proof (rule subrelI)

fix $w z$ assume $(w, z) \in R O S^+$

then obtain x where $R\text{-step}: (w, x) \in R$ and $S\text{-seq}: (x, z) \in S^+$ by best

from tranclD [OF $S\text{-seq}$] obtain y where $S\text{-step}: (x, y) \in S$ and $S\text{-seq}': (y, z) \in S^*$ by auto

from $R\text{-step}$ and $S\text{-step}$ have $(w, y) \in R O S$ by auto

with assms have $(w, y) \in S$ by auto

with S -seq' **show** $(w, z) \in S^+$ **by** *simp*
qed

lemma *comp-rtrancl-trancl*:

assumes *comp*: $R \circ S \subseteq S$
and *seq*: $(s, t) \in (R \cup S)^* \circ S$
shows $(s, t) \in S^+$

using *seq* **proof**

fix $x \ u \ z$

assume $(s, t) = (x, z)$ **and** $(x, u) \in (R \cup S)^*$ **and** $(u, z) \in S$

then have $(s, u) \in (R \cup S)^*$ **and** $(u, t) \in S^+$ **by** *auto*

then show *?thesis*

proof (*induct rule: rtrancl.induct*)

case (*rtrancl-refl x*) **then show** *?case* .

next

case (*rtrancl-into-rtrancl a b c*)

then have $(b, c) \in R \cup S$ **by** *simp*

then show *?case*

proof

assume $(b, c) \in S$

with *rtrancl-into-rtrancl*

have $(b, t) \in S^+$ **by** *simp*

with *rtrancl-into-rtrancl* **show** *?thesis* **by** *simp*

next

assume $(b, c) \in R$

with *comp-trancl [OF comp]* *rtrancl-into-rtrancl*

show *?thesis* **by** *auto*

qed

qed

qed

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

proof (*rule subrelI*)

fix $x \ y$ **assume** $(x, y) \in r^+$ **then show** $(x, y) \in (s \cup r)^+$

proof (*induct*)

case *base* **then show** *?case* **by** *auto*

next

case (*step a b*)

then have $(a, b) \in (s \cup r)^+$ **by** *auto*

with $\langle (x, a) \in (s \cup r)^+ \rangle$ **show** *?case* **by** *auto*

qed

qed

lemma *restrict-SN-subset*: *restrict-SN* $R \ S \subseteq R$

proof (*rule subrelI*)

fix $a \ b$ **assume** $(a, b) \in \text{restrict-SN } R \ S$ **then show** $(a, b) \in R$ **unfolding**
restrict-SN-def **by** *simp*

qed

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 –

from $\langle SN\text{-on}\ S \{t\ 0\} \rangle$ **obtain** i **where** $(t\ i, t\ (Suc\ i)) \notin S$ **by** *blast*

with *assms* **have** $(t\ i, t\ (Suc\ i)) \in R$ **(is** $?P\ i$ **)** **by** *auto*

let $?i = Least\ ?P$

from $\langle ?P\ i \rangle$ **have** $?P\ ?i$ **by** (*rule* *LeastI*)

have $\forall j < ?i. (t\ j, t\ (Suc\ j)) \notin R$ **using** *not-less-Least* **by** *auto*

moreover **with** *assms* **have** $\forall j < ?i. (t\ j, t\ (Suc\ j)) \in S$ **by** *best*

ultimately **have** $\forall j < ?i. (t\ j, t\ (Suc\ j)) \in S \wedge (t\ j, t\ (Suc\ j)) \notin R$ **by** *best*

with $\langle ?P\ ?i \rangle$ **show** *?thesis* **by** *best*

qed

lemma *first-step*:

assumes $C: C = A \cup B$ **and** *steps*: $(x, y) \in C^*$ **and** *Bstep*: $(y, z) \in B$

shows $\exists y. (x, y) \in A^* \ O\ B$

using *steps*

proof (*induct* *rule*: *converse-rtrancl-induct*)

case *base*

show *?case* **using** *Bstep* **by** *auto*

next

case $(step\ u\ x)$

from $step(1)[unfolded\ C]$

show *?case*

proof

assume $(u, x) \in B$

then **show** *?thesis* **by** *auto*

next

assume $ux: (u, x) \in A$

from $step(3)$ **obtain** y **where** $(x, y) \in A^* \ O\ B$ **by** *auto*

then **obtain** z **where** $(x, z) \in A^*$ **and** *step*: $(z, y) \in B$ **by** *auto*

with ux **have** $(u, z) \in A^*$ **by** *auto*

with *step* **have** $(u, y) \in A^* \ O\ B$ **by** *auto*

then **show** *?thesis* **by** *auto*

qed

qed

lemma *first-step-O*:

assumes $C: C = A \cup B$ **and** *steps*: $(x, y) \in C^* \ O\ B$

shows $\exists y. (x, y) \in A^* \ O\ B$

proof –

from *steps* **obtain** z **where** $(x, z) \in C^*$ **and** $(z, y) \in B$ **by** *auto*

from *first-step* [*OF* C *this*] **show** *?thesis* .

qed

lemma *firstStep*:

assumes *LSR*: $L = S \cup R$ **and** $xyL: (x, y) \in L^*$

```

shows  $(x, y) \in R^* \vee (x, y) \in R^* O S O L^*$ 
proof (cases  $(x, y) \in R^*$ )
  case True
    then show ?thesis by simp
  next
    case False
      let  $?SR = S \cup R$ 
      from xyL and LSR have  $(x, y) \in ?SR^*$  by simp
      from this and False have  $(x, y) \in R^* O S O ?SR^*$ 
      proof (induct rule: rtrancl-induct)
        case base then show ?case by simp
      next
        case (step  $y z$ )
          then show ?case
          proof (cases  $(x, y) \in R^*$ )
            case False with step have  $(x, y) \in R^* O S O ?SR^*$  by simp
            from this obtain  $u$  where  $xu: (x, u) \in R^* O S$  and  $uy: (u, y) \in ?SR^*$  by
force
            from  $\langle (y, z) \in ?SR \rangle$  have  $(y, z) \in ?SR^*$  by auto
            with  $uy$  have  $(u, z) \in ?SR^*$  by (rule rtrancl-trans)
            with  $xu$  show ?thesis by auto
          next
            case True
              have  $(y, z) \in S$ 
              proof (rule ccontr)
                assume  $(y, z) \notin S$  with  $\langle (y, z) \in ?SR \rangle$  have  $(y, z) \in R$  by auto
                with True have  $(x, z) \in R^*$  by auto
                with  $\langle (x, z) \notin R^* \rangle$  show False ..
              qed
              with True show ?thesis by auto
            qed
          qed
        with LSR show ?thesis by simp
      qed

```

lemma *non-strict-ending*:

```

assumes chain:  $\text{chain } (R \cup S) t$ 
  and comp:  $R O S \subseteq S$ 
  and SN: SN-on  $S \{t 0\}$ 
shows  $\exists j. \forall i \geq j. (t i, t (Suc i)) \in R - S$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  with chain have  $\forall i. \exists j. j \geq i \wedge (t j, t (Suc j)) \in S$  by blast
  from choice [OF this] obtain  $f$  where S-steps:  $\forall i. i \leq f i \wedge (t (f i), t (Suc (f i))) \in S$  ..
  let  $?t = \lambda i. t (((Suc \circ f) \rightsquigarrow i) 0)$ 
  have S-chain:  $\forall i. (t i, t (Suc (f i))) \in S^+$ 
  proof

```

fix i
from S -steps **have** $leq: i \leq f i$ **and** $step: (t(f i), t(Suc(f i))) \in S$ **by** *auto*
from *chain-imp-rtrancl* [*OF chain leq*] **have** $(t i, t(f i)) \in (R \cup S)^*$.
with $step$ **have** $(t i, t(Suc(f i))) \in (R \cup S)^* O S$ **by** *auto*
from *comp-rtrancl-trancl* [*OF comp this*] **show** $(t i, t(Suc(f i))) \in S^+$.
qed
then **have** *chain* (S^+) **?tby** *simp*
moreover **have** *SN-on* (S^+) $\{?t 0\}$ **using** *SN-on-trancl* [*OF SN*] **by** *simp*
ultimately **show** *False* **unfolding** *SN-defs* **by** *best*
qed

lemma *SN-on-subset1*:
assumes *SN-on* $r A$ **and** $s \subseteq r$
shows *SN-on* $s A$
using *assms* **unfolding** *SN-defs* **by** *blast*

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

lemma *rtrancl-fun-conv*:
 $((s, t) \in R^*) = (\exists f n. f 0 = s \wedge f n = t \wedge (\forall i < n. (f i, f (Suc i)) \in R))$
unfolding *rtrancl-is-UN-relpow* **using** *relpow-fun-conv* [**where** $R = R$]
by *auto*

lemma *compat-tr-compat*:
assumes $NS O S \subseteq S$ **shows** $NS^* O S \subseteq S$
using *non-strict-into-strict* [**where** $S = S$ **and** $NS = NS$] *assms* **by** *blast*

lemma *right-comp-S* [*simp*]:
assumes $(x, y) \in S O (S O S^* O NS^* \cup NS^*)$
shows $(x, y) \in (S O S^* O NS^*)$
proof –
from *assms* **have** $(x, y) \in (S O S O S^* O NS^*) \cup (S O NS^*)$ **by** *auto*
then **have** $xy:(x, y) \in (S O (S O S^*) O NS^*) \cup (S O NS^*)$ **by** *auto*
have $S O S^* \subseteq S^*$ **by** *auto*
with xy **have** $(x, y) \in (S O S^* O NS^*) \cup (S O NS^*)$ **by** *auto*
then **show** $(x, y) \in (S O S^* O NS^*)$ **by** *auto*
qed

lemma *compatible-SN*:
assumes *SN*: $SN S$
and *compat*: $NS O S \subseteq S$
shows *SN* $(S O S^* O NS^*)$ (**is** *SN* $?A$)
proof
fix F **assume** *chain*: *chain* $?A F$
from *compat* *compat-tr-compat* **have** $NS^* O S \subseteq S$ **by** *blast*
have $\forall i. (\exists y z. (F i, y) \in S \wedge (y, z) \in S^* \wedge (z, F (Suc i)) \in NS^*)$
proof
fix i
from *chain* **have** $(F i, F (Suc i)) \in (S O S^* O NS^*)$ **by** *auto*

then show $\exists y z. (F i, y) \in S \wedge (y, z) \in S^* \wedge (z, F (Suc i)) \in NS^*$
unfolding relcomp-def using mem-Collect-eq by auto
qed
then have $\exists f. (\forall i. (\exists z. (F i, f i) \in S \wedge ((f i, z) \in S^*) \wedge (z, F (Suc i)) \in NS^*))$
NS^)*
by (rule choice)
then obtain f
where $\forall i. (\exists z. (F i, f i) \in S \wedge ((f i, z) \in S^*) \wedge (z, F (Suc i)) \in NS^*) ..$
then have $\exists g. \forall i. (F i, f i) \in S \wedge (f i, g i) \in S^* \wedge (g i, F (Suc i)) \in NS^*$
by (rule choice)
then obtain g where $\forall i. (F i, f i) \in S \wedge (f i, g i) \in S^* \wedge (g i, F (Suc i))$
∈ NS^ ..*
then have $\forall i. (f i, g i) \in S^* \wedge (g i, F (Suc i)) \in NS^* \wedge (F (Suc i), f (Suc$
i)) \in S
by auto
then have $\forall i. (f i, g i) \in S^* \wedge (g i, f (Suc i)) \in S$ **unfolding relcomp-def**
using tr-compat by auto
then have all: $\forall i. (f i, g i) \in S^* \wedge (g i, f (Suc i)) \in S^+$ **by auto**
have $\forall i. (f i, f (Suc i)) \in S^+$
proof
fix i
from all have $(f i, g i) \in S^* \wedge (g i, f (Suc i)) \in S^+ ..$
then show $(f i, f (Suc i)) \in S^+$ **using transitive-closure-trans by auto**
qed
then have $\exists x. f 0 = x \wedge chain (S^+) f$ **by auto**
then obtain x where $f 0 = x \wedge chain (S^+) f$ **by auto**
then have $\exists f. f 0 = x \wedge chain (S^+) f$ **by auto**
then have $\neg SN\text{-on} (S^+) \{x\}$ **by auto**
then have $\neg SN (S^+)$ **unfolding SN-defs by auto**
then have $wfSconv: \neg wf ((S^+)^{-1})$ **using SN-iff-wf by auto**
from SN have $wf (S^{-1})$ **using SN-imp-wf [where ?r=S] by simp**
with wf-converse-trancl wfSconv show False by auto
qed

lemma compatible-rtrancl-split:

assumes compat: $NS \ O \ S \subseteq S$

and steps: $(x, y) \in (NS \cup S)^*$

shows $(x, y) \in S \ O \ S^* \ O \ NS^* \cup NS^*$

proof –

from steps have $\exists n. (x, y) \in (NS \cup S)^{\sim n}$ **using rtrancl-imp-relpow [where**
?R=NS ∪ S] by auto

then obtain n where $(x, y) \in (NS \cup S)^{\sim n}$ **by auto**

then show $(x, y) \in S \ O \ S^* \ O \ NS^* \cup NS^*$

proof (induct n arbitrary: x, simp)

case (Suc m)

assume $(x, y) \in (NS \cup S)^{\sim (Suc m)}$

then have $\exists z. (x, z) \in (NS \cup S) \wedge (z, y) \in (NS \cup S)^{\sim m}$

using relpow-Suc-D2 [where ?R=NS ∪ S] by auto

then obtain z where $xz:(x, z) \in (NS \cup S)$ **and** $zy:(z, y) \in (NS \cup S)^{\sim m}$ **by**

auto
with *Suc* **have** $zy:(z, y) \in S O S^* O NS^* \cup NS^*$ **by** *auto*
then show $(x, y) \in S O S^* O NS^* \cup NS^*$
proof (*cases* $(x, z) \in NS$)
 case *True*
 from *compat compat-tr-compat* **have** $trCompat: NS^* O S \subseteq S$ **by** *blast*
 from *zy True* **have** $(x, y) \in (NS O S O S^* O NS^*) \cup (NS O NS^*)$ **by** *auto*
 then have $(x, y) \in ((NS O S) O S^* O NS^*) \cup (NS O NS^*)$ **by** *auto*
 then have $(x, y) \in ((NS^* O S) O S^* O NS^*) \cup (NS O NS^*)$ **by** *auto*
 with *trCompat* **have** $xy:(x, y) \in (S O S^* O NS^*) \cup (NS O NS^*)$ **by** *auto*
 have $NS O NS^* \subseteq NS^*$ **by** *auto*
 with *xy* **show** $(x, y) \in (S O S^* O NS^*) \cup NS^*$ **by** *auto*
 next
 case *False*
 with *xz* **have** $xz:(x, z) \in S$ **by** *auto*
 with *zy* **have** $(x, y) \in S O (S O S^* O NS^* \cup NS^*)$ **by** *auto*
 then show $(x, y) \in (S O S^* O NS^*) \cup NS^*$ **using** *right-comp-S* **by** *simp*
 qed
qed
qed

lemma *compatible-conv*:

assumes *compat*: $NS O S \subseteq S$
shows $(NS \cup S)^* O S O (NS \cup S)^* = S O S^* O NS^*$
proof –
 let $?NSuS = NS \cup S$
 let $?NSS = S O S^* O NS^*$
 let $?midS = ?NSuS^* O S O ?NSuS^*$
 have one: $?NSS \subseteq ?midS$ **by** *regex*
 have $?NSuS^* O S \subseteq (?NSS \cup NS^*) O S$
 using *compatible-rtrancl-split* [**where** $S = S$ **and** $NS = NS$] *compat* **by** *blast*
 also have $\dots \subseteq ?NSS O S \cup NS^* O S$ **by** *auto*
 also have $\dots \subseteq ?NSS O S \cup S$ **using** *compat compat-tr-compat* [**where** $S = S$
and $NS = NS$] **by** *auto*
 also have $\dots \subseteq S O ?NSuS^*$ **by** *regex*
 finally have $?midS \subseteq S O ?NSuS^* O ?NSuS^*$ **by** *blast*
 also have $\dots \subseteq S O ?NSuS^*$ **by** *regex*
 also have $\dots \subseteq S O (?NSS \cup NS^*)$
 using *compatible-rtrancl-split* [**where** $S = S$ **and** $NS = NS$] *compat* **by** *blast*
 also have $\dots \subseteq ?NSS$ **by** *regex*
 finally have two: $?midS \subseteq ?NSS$.
 from one two show *?thesis* **by** *auto*
qed

lemma *compatible-SN'*:

assumes *compat*: $NS O S \subseteq S$ **and** *SN*: $SN S$
shows $SN((NS \cup S)^* O S O (NS \cup S)^*)$
using *compatible-conv* [**where** $S = S$ **and** $NS = NS$]
 compatible-SN [**where** $S = S$ **and** $NS = NS$] *assms* **by** *force*

lemma *rtrancl-diff-decomp*:
assumes $(x, y) \in A^* - B^*$
shows $(x, y) \in A^* O (A - B) O A^*$
proof –
from *assms* **have** $A: (x, y) \in A^*$ **and** $B: (x, y) \notin B^*$ **by** *auto*
from A **have** $\exists k. (x, y) \in A \overset{\sim}{\sim} k$ **by** (*rule rtrancl-imp-relpow*)
then obtain k **where** $Ak: (x, y) \in A \overset{\sim}{\sim} k$ **by** *auto*
from $Ak B$ **show** $(x, y) \in A^* O (A - B) O A^*$
proof (*induct k arbitrary: x*)
 case 0
 with $\langle (x, y) \notin B^* \rangle 0$ **show** *?case* **using** *ccontr* **by** *auto*
 next
 case (*Suc i*)
 then have $B: (x, y) \notin B^*$ **and** $ASk: (x, y) \in A \overset{\sim}{\sim} \text{Suc } i$ **by** *auto*
 from ASk **have** $\exists z. (x, z) \in A \wedge (z, y) \in A \overset{\sim}{\sim} i$ **using** *relpow-Suc-D2* [**where**
?R=A] **by** *auto*
 then obtain z **where** $xz: (x, z) \in A$ **and** $(z, y) \in A \overset{\sim}{\sim} i$ **by** *auto*
 then have $zy: (z, y) \in A^*$ **using** *relpow-imp-rtrancl* **by** *auto*
 from xz **show** $(x, y) \in A^* O (A - B) O A^*$
 proof (*cases (x, z) \in B*)
 case *False*
 with $xz zy$ **show** $(x, y) \in A^* O (A - B) O A^*$ **by** *auto*
 next
 case *True*
 then have $(x, z) \in B^*$ **by** *auto*
 have $[(x, z) \in B^*; (z, y) \in B^*] \implies (x, y) \in B^*$ **using** *rtrancl-trans* [*of x z*
B] **by** *auto*
 with $\langle (x, z) \in B^* \rangle \langle (x, y) \notin B^* \rangle$ **have** $(z, y) \notin B^*$ **by** *auto*
 with $\text{Suc } \langle (z, y) \in A \overset{\sim}{\sim} i \rangle$ **have** $(z, y) \in A^* O (A - B) O A^*$ **by** *auto*
 with xz **have** $xy: (x, y) \in A O A^* O (A - B) O A^*$ **by** *auto*
 have $A O A^* O (A - B) O A^* \subseteq A^* O (A - B) O A^*$ **by** *regexp*
 from *this xy* **show** $(x, y) \in A^* O (A - B) O A^*$
 using *subsetD* [**where** *?A=A O A^* O (A - B) O A^**] **by** *auto*
 qed
qed
qed

lemma *SN-empty [simp]*: $SN \ \{\}$ **by** *auto*

lemma *SN-on-weakening*:

assumes *SN-on R1 A*

shows *SN-on (R1 \cap R2) A*

proof –

{

assume $\exists S. S \ 0 \in A \wedge \text{chain } (R1 \cap R2) S$

then obtain S **where**

$S0: S \ 0 \in A$ **and**

$SN: \text{chain } (R1 \cap R2) S$

```

    by auto
  from SN have SN': chain R1 S by simp
  with S0 and assms have False by auto
}
then show ?thesis by force
qed

```

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*: $R \subseteq R' \Longrightarrow S \subseteq S' \Longrightarrow ideriv\ R\ S\ as \Longrightarrow ideriv\ R'\ S'\ as$
unfolding *ideriv-def* *INFM-nat* **by** *blast*

```

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  $\exists i. ideriv\ (R - D)\ (S - D)\ (shift\ as\ i)$ 

```

proof -

```

have RS:  $R - D \cup (S - D) = R \cup S - D$  by auto
from ideriv [unfolded ideriv-def]
have as:  $\bigwedge i. (as\ i, as\ (Suc\ i)) \in R \cup S$ 
and inf:  $INFM\ i. (as\ i, as\ (Suc\ i)) \in R$  by auto
show ?thesis
proof (cases  $INFM\ i. (as\ i, as\ (Suc\ i)) \in D \cap (R \cup S)$ )
  case True
    have ideriv  $(D \cap (R \cup S))\ (R \cup S - D)\ as$ 
      unfolding ideriv-def
      using as True by auto
    with nideriv show ?thesis ..

```

next

```

  case False
    from False [unfolded INFM-nat]
    obtain i where Dn:  $\bigwedge j. i < j \Longrightarrow (as\ j, as\ (Suc\ j)) \notin D \cap (R \cup S)$ 
      by auto
    from Dn as have as:  $\bigwedge j. i < j \Longrightarrow (as\ j, as\ (Suc\ j)) \in R \cup S - D$  by auto
    show ?thesis
    proof (rule exI [of - Suc i], unfold ideriv-def RS, insert as, intro conjI, simp,
      unfold INFM-nat, intro allI)
      fix m
      from inf [unfolded INFM-nat] obtain j where j:  $j > Suc\ i + m$ 
      and R:  $(as\ j, as\ (Suc\ j)) \in R$  by auto
      with as [of j] have RD:  $(as\ j, as\ (Suc\ j)) \in R - D$  by auto

```

show $\exists j > m. (\text{shift as } (Suc\ i)\ j, \text{shift as } (Suc\ i)\ (Suc\ j)) \in R - D$
by (*rule exI [of - j - Suc i], insert j RD, auto*)
qed
qed
qed

lemma *ideriv-SN*:

assumes *SN*: $SN\ S$
and *compat*: $NS\ O\ S \subseteq S$
and *R*: $R \subseteq NS \cup S$
shows $\neg \text{ideriv } (S \cap R)\ (R - S)\ \text{as}$
proof
assume *ideriv* $(S \cap R)\ (R - S)\ \text{as}$
with *R* **have** *steps*: $\forall i. (\text{as } i, \text{as } (Suc\ i)) \in NS \cup S$
and *inf*: *INFM* *i*. $(\text{as } i, \text{as } (Suc\ i)) \in S \cap R$ **unfolding** *ideriv-def* **by** *auto*
from *non-strict-ending* [*OF steps compat*] *SN*
obtain *i* **where** *i*: $\bigwedge j. j \geq i \implies (\text{as } j, \text{as } (Suc\ j)) \in NS - S$ **by** *fast*
from *inf* [*unfolded INFM-nat*] **obtain** *j* **where** $j > i$ **and** $(\text{as } j, \text{as } (Suc\ j)) \in S$
by *auto*
with *i* [*of j*] **show** *False* **by** *auto*
qed

lemma *Infm-shift*: $(INFM\ i.\ P\ (\text{shift } f\ n\ i)) = (INFM\ i.\ P\ (f\ i))$ (**is** $?S = ?O$)

proof

assume *?S*
show *?O*
unfolding *INFM-nat-le*
proof
fix *m*
from $\langle ?S \rangle$ [*unfolded INFM-nat-le*]
obtain *k* **where** $k \geq m$ **and** *p*: $P\ (\text{shift } f\ n\ k)$ **by** *auto*
show $\exists k \geq m. P\ (f\ k)$
by (*rule exI [of - k + n], insert k p, auto*)
qed

qed

next

assume *?O*
show *?S*
unfolding *INFM-nat-le*
proof
fix *m*
from $\langle ?O \rangle$ [*unfolded INFM-nat-le*]
obtain *k* **where** $k \geq m + n$ **and** *p*: $P\ (f\ k)$ **by** *auto*
show $\exists k \geq m. P\ (\text{shift } f\ n\ k)$
by (*rule exI [of - k - n], insert k p, auto*)
qed

qed

qed

lemma *rtrancl-list-conv*:

$(s, t) \in R^* \longleftrightarrow$

($\exists ts. \text{last } (s \# ts) = t \wedge (\forall i < \text{length } ts. ((s \# ts) ! i, (s \# ts) ! \text{Suc } i) \in R)$)
(is ?l = ?r)

proof

assume ?r

then obtain ts where last (s # ts) = t \wedge ($\forall i < \text{length } ts. ((s \# ts) ! i, (s \# ts) ! \text{Suc } i) \in R$) ..

then show ?l

proof (induct ts arbitrary: s, simp)

case (Cons u ll)

then have last (u # ll) = t \wedge ($\forall i < \text{length } ll. ((u \# ll) ! i, (u \# ll) ! \text{Suc } i) \in R$) by auto

from Cons(1)[OF this] have rec: (u, t) $\in R^*$.

from Cons have (s, u) $\in R$ by auto

with rec show ?case by auto

qed

next

assume ?l

from rtrancl-imp-seq [OF this]

obtain S n where s: S 0 = s and t: S n = t and steps: $\forall i < n. (S i, S (\text{Suc } i)) \in R$ by auto

let ?ts = map ($\lambda i. S (\text{Suc } i)$) [0 ..< n]

show ?r

proof (rule exI [of - ?ts], intro conjI,

cases n, simp add: s [symmetric] t [symmetric], simp add: t [symmetric])

show $\forall i < \text{length } ?ts. ((s \# ?ts) ! i, (s \# ?ts) ! \text{Suc } i) \in R$

proof (intro allI impI)

fix i

assume i: i < length ?ts

then show ((s # ?ts) ! i, (s # ?ts) ! Suc i) $\in R$

proof (cases i, simp add: s [symmetric] steps)

case (Suc j)

with i steps show ?thesis by simp

qed

qed

qed

qed

lemma SN-reaches-NF:

assumes SN-on r {x}

shows $\exists y. (x, y) \in r^* \wedge y \in \text{NF } r$

using assms

proof (induct rule: SN-on-induct')

case (IH x)

show ?case

proof (cases x $\in \text{NF } r$)

case True

then show ?thesis by auto

next

case False

then obtain y **where** $step: (x, y) \in r$ **by** *auto*
from *IH* [*OF this*] **obtain** z **where** $steps: (y, z) \in r^*$ **and** $NF: z \in NF\ r$ **by**
auto
show *?thesis*
by (*intro exI*, *rule conjI* [*OF - NF*], *insert step steps*, *auto*)
qed
qed

lemma *SN-WCR-reaches-NF*:

assumes $SN: SN\text{-on } r\ \{x\}$
and $WCR: WCR\text{-on } r\ \{x.\ SN\text{-on } r\ \{x\}\}$
shows $\exists! y. (x, y) \in r^* \wedge y \in NF\ r$
proof –
from *SN-reaches-NF* [*OF SN*] **obtain** y **where** $steps: (x, y) \in r^*$ **and** $NF: y \in$
 $NF\ r$ **by** *auto*
show *?thesis*
proof(*rule*, *rule conjI* [*OF steps NF*])
fix z
assume $steps': (x, z) \in r^* \wedge z \in NF\ r$
from *Newman-local* [*OF SN WCR*] **have** $CR\text{-on } r\ \{x\}$ **by** *auto*
from *CR-onD* [*OF this - steps*] $steps'$ **have** $(y, z) \in r^\downarrow$ **by** *simp*
from *join-NF-imp-eq* [*OF this NF*] $steps'$ **show** $z = y$ **by** *simp*
qed
qed

definition *some-NF* :: $'a\ rel \Rightarrow 'a \Rightarrow 'a$ **where**

$some\text{-NF } r\ x = (SOME\ y. (x, y) \in r^* \wedge y \in NF\ r)$

lemma *some-NF*:

assumes $SN: SN\text{-on } r\ \{x\}$
shows $(x, some\text{-NF } r\ x) \in r^* \wedge some\text{-NF } r\ x \in NF\ r$
using *someI-ex* [*OF SN-reaches-NF* [*OF SN*]]
unfolding *some-NF-def* .

lemma *some-NF-WCR*:

assumes $SN: SN\text{-on } r\ \{x\}$
and $WCR: WCR\text{-on } r\ \{x.\ SN\text{-on } r\ \{x\}\}$
and $steps: (x, y) \in r^*$
and $NF: y \in NF\ r$
shows $y = some\text{-NF } r\ x$
proof –
let $?p = \lambda y. (x, y) \in r^* \wedge y \in NF\ r$
from *SN-WCR-reaches-NF* [*OF SN WCR*]
have *one*: $\exists! y. ?p\ y$.
from *steps NF* **have** $y: ?p\ y$..
from *some-NF* [*OF SN*] **have** *some*: $?p\ (some\text{-NF } r\ x)$.
from *one some y* **show** *?thesis* **by** *auto*
qed

lemma *some-NF-UNF*:
assumes *UNF*: $UNF\ r$
and *steps*: $(x, y) \in r^*$
and *NF*: $y \in NF\ r$
shows $y = some\text{-}NF\ r\ x$
proof –
let $?p = \lambda\ y. (x, y) \in r^* \wedge y \in NF\ r$
from *steps* *NF* **have** *py*: $?p\ y$ **by** *simp*
then **have** *pNF*: $?p\ (some\text{-}NF\ r\ x)$ **unfolding** *some-NF-def*
by (*rule someI*)
from *py* **have** *y*: $(x, y) \in r^!$ **by** *auto*
from *pNF* **have** *nf*: $(x, some\text{-}NF\ r\ x) \in r^!$ **by** *auto*
from *UNF* [*unfolded UNF-on-def*] *y nf* **show** *?thesis* **by** *auto*
qed

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*: $(a, the\text{-}NF\ A\ a) \in A^!$
proof –
obtain b **where** *ab*: $(a, b) \in A^!$ **using** *SN* **by** (*meson SN-imp-WN UNIV-I WN-onE*)
moreover **have** $(a, c) \in A^! \implies c = b$ **for** c
using *CR* **and** *ab* **by** (*meson CR-divergence-imp-join join-NF-imp-eq normalizability-E*)
ultimately **have** $\exists!b. (a, b) \in A^!$ **by** *blast*
then **show** *?thesis* **unfolding** *the-NF-def* **by** (*rule theI'*)
qed

lemma *the-NF-NF*: $the\text{-}NF\ A\ a \in NF\ A$
using *the-NF* **by** (*auto simp: normalizability-def*)

lemma *the-NF-step*:
assumes $(a, b) \in A$
shows $the\text{-}NF\ A\ a = the\text{-}NF\ A\ b$
using *the-NF* **and** *assms*
by (*meson CR SN SN-imp-WN conversionI' r-into-rtrancl semi-complete-imp-conversionIff-same-NF semi-complete-onI*)

lemma *the-NF-steps*:
assumes $(a, b) \in A^*$
shows $the\text{-}NF\ A\ a = the\text{-}NF\ A\ b$
using *assms* **by** (*induct*) (*auto dest: the-NF-step*)

lemma *the-NF-conv*:
assumes $(a, b) \in A^{\leftrightarrow*}$

shows *the-NF A a = the-NF A b*
using *assms*
by (*meson CR WN-on-def the-NF semi-complete-imp-conversionIff-same-NF semi-complete-onI*)

end

definition *weak-diamond* :: 'a rel \Rightarrow bool ($w\Diamond$) **where**
 $w\Diamond r \longleftrightarrow (r^{-1} O r) - Id \subseteq (r O r^{-1})$

lemma *weak-diamond-imp-CR*:

assumes *wd*: $w\Diamond r$
shows *CR r*
proof (*rule semi-confluence-imp-CR, rule*)
fix *x y*
assume $(x, y) \in r^{-1} O r^*$
then obtain *z* **where** *step*: $(z, x) \in r$ **and** *steps*: $(z, y) \in r^*$ **by** *auto*
from *steps*
have $\exists u. (x, u) \in r^* \wedge (y, u) \in r^=$
proof (*induct*)
case *base*
show *?case*
by (*rule exI [of - x], insert step, auto*)
next
case (*step y' y*)
from *step(3)* **obtain** *u* **where** *xu*: $(x, u) \in r^*$ **and** *y'u*: $(y', u) \in r^=$ **by** *auto*
from *y'u* **have** $(y', u) \in r \vee y' = u$ **by** *auto*
then show *?case*
proof
assume *y'u*: $y' = u$
with *xu* *step(2)* **have** *xy*: $(x, y) \in r^*$ **by** *auto*
show *?thesis*
by (*intro exI conjI, rule xy, simp*)
next
assume $(y', u) \in r$
with *step(2)* **have** *uy*: $(u, y) \in r^{-1} O r$ **by** *auto*
show *?thesis*
proof (*cases u = y*)
case *True*
show *?thesis*
by (*intro exI conjI, rule xu, unfold True, simp*)
next
case *False*
with *uy*
wd [*unfolded weak-diamond-def*] **obtain** *u'* **where** *uu'*: $(u, u') \in r$
and *yu'*: $(y, u') \in r$ **by** *auto*
from *xu uu'* **have** *xu*: $(x, u') \in r^*$ **by** *auto*
show *?thesis*
by (*intro exI conjI, rule xu, insert yu', auto*)

qed
qed
qed
then show $(x, y) \in r^\downarrow$ **by auto**
qed

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

fixes $t :: 'a \Rightarrow 'b$
and $R :: 'b \text{ rel}$
assumes *steps*: $\bigwedge x. (t x, t (f x)) \in R$
shows $\neg \text{SN-on } R \{t x\}$
proof
let $?U = \text{range } t$
assume *SN-on* $R \{t x\}$
from *SN-on-imp-on-minimal* [*OF this, rule-format, of ?U*]
obtain tz **where** $tz \in \text{range } t$ **and** *min*: $\bigwedge y. (tz, y) \in R \implies y \notin \text{range } t$
by auto
from tz **obtain** z **where** $tz = t z$ **by auto**
from *steps* [*of z*] *min* [*of t (f z)*] **show** *False* **unfolding** tz **by auto**
qed

lemma *steps-imp-not-SN*:

fixes $t :: 'a \Rightarrow 'b$
and $R :: 'b \text{ rel}$
assumes *steps*: $\bigwedge x. (t x, t (f x)) \in R$
shows $\neg \text{SN } R$
proof –
from *steps-imp-not-SN-on* [*of t f R, OF steps*]
show *?thesis* **unfolding** *SN-def* **by blast**
qed

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 $((t, u) \in R^* \longrightarrow (f t, f u) \in (g R)^*)$
 $\wedge ((t, u) \in R^* O S O R^* \longrightarrow (f t, f u) \in (g R)^* O (g S) O (g R)^*)$
proof –
{
fix $t u$
assume $(t, u) \in R^*$ **and** $P t$
then have $P u \wedge (f t, f u) \in (g R)^*$
proof (*induct*)
case (*step u v*)
from *step(3)*[*OF step(4)*] **have** $Pu: P u$ **and** *steps*: $(f t, f u) \in (g R)^*$ **by**
auto
from *fg* [*OF Pu R step(2)*] **have** $Pv: P v$ **and** *step*: $(f u, f v) \in g R$ **by auto**
with *steps* **have** $(f t, f v) \in (g R)^*$ **by auto**
}

```

    with Pv show ?case by simp
  qed simp
} note main = this
note maint = main [OF - t]
from maint [of u] have one: (t, u) ∈ R* ⟶ (f t, f u) ∈ (g R)* by simp
show ?thesis
proof (rule conjI [OF one impI])
  assume (t, u) ∈ R* O S O R*
  then obtain s v where ts: (t, s) ∈ R* and sv: (s, v) ∈ S and vu: (v, u) ∈
R* by auto
  from maint [OF ts] have Ps: P s and ts: (f t, f s) ∈ (g R)* by auto
  from fg [OF Ps S sv] have Pv: P v and sv: (f s, f v) ∈ g S by auto
  from main [OF vu Pv] have vu: (f v, f u) ∈ (g R)* by auto
  from ts sv vu show (f t, f u) ∈ (g R)* O g S O (g R)* by auto
qed
qed

```

2.6 Terminating part of a relation

inductive-set

```

SN-part :: 'a rel ⇒ 'a set
for r :: 'a rel

```

where

```

SN-partI: (∧y. (x, y) ∈ r ⟹ y ∈ SN-part r) ⟹ x ∈ SN-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 $(\bigwedge y. (y, ?x) \in ?r \implies y \in \text{Wellfounded.acc } ?r) \implies ?x \in \text{Wellfounded.acc } ?r$.

Characterization of *SN-on* via terminating part.

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

```

SN-on r A ⟷ A ⊆ SN-part r

```

proof –

```

{
  fix x assume SN-on r A and x ∈ A
  then have x ∈ SN-part r by (induct) (auto intro: SN-partI)
} moreover {
  fix x assume x ∈ A and A ⊆ SN-part r
  then have x ∈ SN-part r by auto
  then have SN-on r {x} by (induct) (auto intro: step-reflects-SN-on)
} ultimately show ?thesis by (force simp: SN-defs)
qed

```

Special case for “full” termination.

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

```

SN r ⟷ SN-part r = UNIV
using SN-on-SN-part-conv [of r UNIV] by auto

```

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

and $R: R \text{ “ } A \subseteq A$

shows $\{t \mid s. s \in L \wedge (s,t) \in R^{\widehat{*}}\} \subseteq A$
proof –
 {
 fix $s t$
 assume $(s,t) \in R^{\widehat{*}}$ **and** $s \in L$
 hence $t \in A$
 by (*induct, insert L R, auto*)
 }
thus *?thesis* **by** *auto*
qed

lemma *trancl-steps-relpow*: **assumes** $a \subseteq b^{\widehat{+}}$
shows $(x,y) \in a^{\widehat{\sim}n} \implies \exists m. m \geq n \wedge (x,y) \in b^{\widehat{\sim}m}$
proof (*induct n arbitrary: y*)
case 0 **thus** *?case* **by** (*intro exI[of - 0], auto*)
next
case (*Suc n z*)
from *Suc(2)* **obtain** y **where** $xy: (x,y) \in a^{\widehat{\sim}n}$ **and** $yz: (y,z) \in a$ **by** *auto*
from *Suc(1)[OF xy]* **obtain** m **where** $m: m \geq n$ **and** $xy: (x,y) \in b^{\widehat{\sim}m}$ **by** *auto*
from yz **assms** **have** $(y,z) \in b^{\widehat{+}}$ **by** *auto*
from *this[unfolded trancl-power]* **obtain** k **where** $k: k > 0$ **and** $yz: (y,z) \in b^{\widehat{\sim}k}$ **by** *auto*
from $xy yz$ **have** $(x,z) \in b^{\widehat{\sim}(m+k)}$ **unfolding** *relpow-add* **by** *auto*
with $k m$ **show** *?case* **by** (*intro exI[of - m + k], auto*)
qed

lemma *relpow-image*: **assumes** $f: \bigwedge s t. (s,t) \in r \implies (f s, f t) \in r'$
shows $(s,t) \in r^{\widehat{\sim}n} \implies (f s, f t) \in r'^{\widehat{\sim}n}$
proof (*induct n arbitrary: t*)
case (*Suc n u*)
from *Suc(2)* **obtain** t **where** $st: (s,t) \in r^{\widehat{\sim}n}$ **and** $tu: (t,u) \in r$ **by** *auto*
from *Suc(1)[OF st] f[OF tu]* **show** *?case* **by** *auto*
qed *auto*

lemma *relpow-refl-mono*:
assumes *refl*: $\bigwedge x. (x,x) \in Rel$
shows $m \leq n \implies (a,b) \in Rel^{\widehat{\sim}m} \implies (a,b) \in Rel^{\widehat{\sim}n}$
proof (*induct rule:dec-induct*)
case (*step i*)
hence $abi: (a, b) \in Rel^{\widehat{\sim}i}$ **by** *auto*
from *refl[of b] abi relpowp-Suc-I[of i λ x y. (x,y) ∈ Rel]* **show** $(a, b) \in Rel^{\widehat{\sim}(i+1)}$
Suc i **by** *auto*
qed

lemma *SN-on-induct-acc-style* [*consumes 1, case-names IH*]:
assumes *sn*: *SN-on* $R \{a\}$
and *IH*: $\bigwedge x. SN\text{-on } R \{x\} \implies \llbracket \bigwedge y. (x, y) \in R \implies P y \rrbracket \implies P x$
shows $P a$

proof –
from *sn SN-on-conv-acc* [of R^{-1} a] **have** $a: a \in \text{termi } R$ **by** *auto*
show *?thesis*
proof (*rule Wellfounded.acc.induct* [OF a , of P], *rule IH*)
fix x
assume $\bigwedge y. (y, x) \in R^{-1} \implies y \in \text{termi } R$
from *this* [*folded SN-on-conv-acc*]
show *SN-on* $R \{x\}$ **by** *simp fast*
qed *auto*
qed

lemma *partially-localize-CR*:

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

proof

assume $CR \ r$

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

next

assume $1: \forall x \ y \ z. (x, y) \in r \wedge (x, z) \in r^* \longrightarrow (y, z) \in \text{join } r$

show $CR \ r$

proof

fix $a \ b \ c$

assume $2: a \in UNIV$ **and** $3: (a, b) \in r^*$ **and** $4: (a, c) \in r^*$

then obtain n **where** $(a, c) \in r^{\sim n}$ **using** *rtrancl-is-UN-relpow* **by** *fast*

with $2 \ 3$ **show** $(b, c) \in \text{join } r$

proof (*induct n arbitrary: a b c*)

case 0 **thus** *?case* **by** *auto*

next

case (*Suc m*)

from *Suc(4)* **obtain** d **where** $ad: (a, d) \in r^{\sim m}$ **and** $dc: (d, c) \in r$ **by** *auto*

from *Suc(1)* [*OF Suc(2) Suc(3) ad*] **have** $(b, d) \in \text{join } r$.

with $1 \ dc \ \text{joinE} \ \text{joinI}$ [*of b - r c*] *join-rtrancl-join* **show** *?case* **by** *metis*

qed

qed

qed

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

where

strongly-confluent-on $r \ A \longleftrightarrow$

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

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

where

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

lemma *strongly-confluent-on-E11*:

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

$\exists u. (y, u) \in r^* \wedge (z, u) \in r^{\equiv}$

unfolding *strongly-confluent-on-def* **by** *blast*

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

unfolding *strongly-confluent-on-def* **by** *auto*

lemma *strongly-confluent-E1n*:

assumes *scr*: *strongly-confluent* r

shows $(x, y) \in r^- \implies (x, z) \in r \overset{\sim}{\sim} n \implies \exists u. (y, u) \in r^* \wedge (z, u) \in r^-$

proof (*induct* n *arbitrary*: $x y z$)

case (*Suc* m)

from *Suc*(3) **obtain** w **where** xw : $(x, w) \in r \overset{\sim}{\sim} m$ **and** wz : $(w, z) \in r$ **by** *auto*

from *Suc*(1) [*OF* *Suc*(2) xw] **obtain** u **where** yu : $(y, u) \in r^*$ **and** wu : $(w, u) \in r^-$ **by** *auto*

from *strongly-confluent-on-E11* [*OF* *scr*, *of* w] $wz yu wu$ **show** *?case*

by (*metis* *UnE* *converse-rtrancl-into-rtrancl* *iso-tuple-UNIV-I* *pair-in-Id-conv* *rtrancl-trans*)

qed *auto*

lemma *strong-confluence-imp-CR*:

assumes *strongly-confluent* r

shows *CR* r

proof –

{ **fix** $x y z$

have $(x, y) \in r \implies (x, z) \in r^* \implies (y, z) \in \text{join } r$

by (*cases* $x = y$, *insert* *strongly-confluent-E1n* [*OF* *assms*], *blast+*) }

then show *CR* r **using** *partially-localize-CR* **by** *blast*

qed

lemma *WCR-alt-def*: $WCR A \longleftrightarrow A^{-1} O A \subseteq A^\downarrow$ **by** (*auto* *simp*: *WCR-defs*)

lemma *NF-imp-SN-on*: $a \in NF R \implies SN\text{-on } R \{a\}$ **unfolding** *SN-on-def* *NF-def* **by** *blast*

lemma *Union-sym*: $(s, t) \in (\bigcup_{i \leq n}. (S i)^{\leftrightarrow}) \longleftrightarrow (t, s) \in (\bigcup_{i \leq n}. (S i)^{\leftrightarrow})$ **by** *auto*

lemma *peak-iff*: $(x, y) \in A^{-1} O B \longleftrightarrow (\exists u. (u, x) \in A \wedge (u, y) \in B)$ **by** *auto*

lemma *CR-NF-conv*:

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

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

using *assms*

unfolding *CR-imp-conversionIff-join* [*OF* $\langle CR r \rangle$]

by (*auto* *simp*: *NF-iff-no-step* *normalizability-def*)

(*metis* (*mono-tags*) *converse-rtranclE* *joinE*)

lemma *NF-join-imp-reach*:

assumes $y \in NF\ A$ **and** $(x, y) \in A^\downarrow$
shows $(x, y) \in A^*$
using *assms* **by** (*auto simp: join-def*) (*metis NF-not-suc rtrancl-converseD*)

lemma *conversion-O-conversion* [*simp*]:
 $A^{\leftrightarrow*} \ O \ A^{\leftrightarrow*} = A^{\leftrightarrow*}$
by (*force simp: converse-def*)

lemma *trans-O-iff*: $trans\ A \ \longleftrightarrow \ A \ O \ A \subseteq A$ **unfolding** *trans-def* **by** *auto*
lemma *refl-O-iff*: $refl\ A \ \longleftrightarrow \ Id \subseteq A$ **unfolding** *refl-on-def* **by** *auto*

lemma *relpow-Suc*: $r \ \overset{\sim}{\sim} \ Suc\ n = r \ O \ r \ \overset{\sim}{\sim} \ n$
using *relpow-add[of 1 n r]* **by** *auto*

lemma *converse-power*: **fixes** $r :: 'a\ rel$ **shows** $(r^{-1}) \ \overset{\sim}{\sim} \ n = (r \ \overset{\sim}{\sim} \ n)^{-1}$
proof (*induct n*)
case (*Suc n*)
show *?case* **unfolding** *relpow.simps(2)[of - r^{-1}] relpow-Suc[of - r]*
by (*simp add: Suc converse-relcomp*)
qed *simp*

lemma *conversion-mono*: $A \subseteq B \implies A^{\leftrightarrow*} \subseteq B^{\leftrightarrow*}$
by (*auto simp: conversion-def intro!: rtrancl-mono*)

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

lemma *lower-set-imp-not-SN-on*:
assumes $s \in X \ \forall t \in X. \ \exists u \in X. \ (t, u) \in R$ **shows** $\neg SN\text{-on}\ R \ \{s\}$
by (*meson SN-on-imp-on-minimal assms*)

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

proof (*intro iffI*)
assume *?l* **show** *?r* **by** (*rule SN-on-subset2[OF - <?l], auto*)
qed (*fact SN-on-Image-rtrancl*)

lemma *O-mono1*: $R \subseteq R' \implies S \ O \ R \subseteq S \ O \ R'$ **by** *auto*

lemma *O-mono2*: $R \subseteq R' \implies R \ O \ T \subseteq R' \ O \ T$ **by** *auto*

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

proof (*intro equalityI subrelI*)
fix $x\ y$
assume $(x, y) \in (S \ O \ R)^* \ O \ S$
then obtain n **where** $(x, y) \in (S \ O \ R) \ \overset{\sim}{\sim} \ n \ O \ S$ **by** *blast*
then show $(x, y) \in S \ O \ (R \ O \ S)^*$
proof (*induct n arbitrary: y*)

```

    case IH: (Suc n)
    then obtain z where xz: (x,z) ∈ (S O R) ~^n O S and zy: (z,y) ∈ R O S by
auto
    from IH.hyps[OF xz] zy have (x,y) ∈ S O (R O S)* O R O S by auto
    then show ?case by(fold trancl-unfold-right, auto)
qed auto
next
fix x y
assume (x,y) ∈ S O (R O S)*
then obtain n where (x,y) ∈ S O (R O S) ~^n by blast
then show (x,y) ∈ (S O R)* O S
proof(induct n arbitrary: y)
  case IH: (Suc n)
  then obtain z where xz: (x,z) ∈ S O (R O S) ~^n and zy: (z,y) ∈ R O S by
auto
  from IH.hyps[OF xz] zy have (x,y) ∈ ((S O R)* O S O R) O S by auto
  from this[folded trancl-unfold-right]
  show ?case by (rule rev-subsetD[OF - O-mono2], auto simp: O-assoc)
qed auto
qed

```

lemma *O-rtrancl-O-O*: $R O (S O R)^* O S = (R O S)^+$
 by (unfold rtrancl-O-shift trancl-unfold-left, auto)

lemma *SN-on-subset-SN-terms*:
 assumes *SN*: *SN-on* $R X$ shows $X \subseteq \{x. \text{SN-on } R \{x\}\}$
proof(intro subsetI, unfold mem-Collect-eq)
 fix x assume $x \in X$
 show *SN-on* $R \{x\}$ by (rule *SN-on-subset2*[OF - *SN*], insert x , auto)
qed

lemma *SN-on-Un2*:
 assumes *SN-on* $R X$ and *SN-on* $R Y$ shows *SN-on* $R (X \cup Y)$
 using *assms* by fast

lemma *SN-on-UN*:
 assumes $\bigwedge x. \text{SN-on } R (X x)$ shows *SN-on* $R (\bigcup x. X x)$
 using *assms* by fast

lemma *Image-subsetI*: $R \subseteq R' \implies R \text{ `` } X \subseteq R' \text{ `` } X$ by auto

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

proof
 fix $seq :: nat \Rightarrow 'b$ assume $seq0: seq\ 0 \in X$ and *chain*: *chain* $(S O R) seq$
 from *SN* have *SN*: *SN-on* $(R O S) ((R O S)^* \text{ `` } S \text{ `` } X)$ by simp
 { fix $i a$
 assume $ia: (seq\ i, a) \in S$ and $aSi: (a, seq\ (Suc\ i)) \in R$

```

have seq i ∈ (S O R)* “ X
proof (induct i)
  case 0 from seq0 show ?case by auto
next
  case (Suc i) with chain have seq (Suc i) ∈ ((S O R)* O S O R) “ X by
blast
  also have ... ⊆ (S O R)* “ X by (fold trancl-unfold-right, auto)
  finally show ?case.
qed
with ia have a ∈ ((S O R)* O S) “ X by auto
then have a: a ∈ ((R O S)* “ S “ X by (auto simp: rtrancl-O-shift)
with ia aSi have False
proof(induct a arbitrary: i rule: SN-on-induct[OF SN])
  case 1 show ?case by (fact a)
next
  case IH: (2 a)
  from chain obtain b
  where *: (seq (Suc i), b) ∈ S (b, seq (Suc (Suc i))) ∈ R by auto
  with IH have ab: (a,b) ∈ R O S by auto
  with ⟨a ∈ (R O S)* “ S “ X⟩ have b ∈ ((R O S)* O R O S) “ S “ X by
auto
  then have b ∈ (R O S)* “ S “ X
  by (rule rev-subsetD, intro Image-subsetI, fold trancl-unfold-right, auto)
  from IH.hyps[OF ab * this] IH.prem ab show False by auto
qed
}
with chain show False by auto
qed

lemma SN-O-comm: SN (R O S) ⟷ SN (S O R)
  by (intro iffI; rule SN-on-O-comm[OF SN-on-subset2], auto)

lemma chain-mono: assumes R' ⊆ R chain R' seq shows chain R seq
  using assms by auto

context
  fixes S R
  assumes push: S O R ⊆ R O S*
begin

lemma rtrancl-O-push: S* O R ⊆ R O S*
proof –
  { fix n
    have ∧s t. (s,t) ∈ S ~ n O R ⟹ (s,t) ∈ R O S*
    proof(induct n)
      case (Suc n)
      then obtain u where (s,u) ∈ S (u,t) ∈ R O S* unfolding relpow-Suc by
blast
      then have (s,t) ∈ S O R O S* by auto
    }
  }

```

also have $\dots \subseteq R \circ S^* \circ S^*$ using *push by blast*
 also have $\dots \subseteq R \circ S^*$ by *auto*
 finally show *?case*.
 qed *auto*
 }
 thus *?thesis* by *blast*
 qed

lemma *rtrancl-U-push*: $(S \cup R)^* = R^* \circ S^*$

proof(*intro equalityI subrelI*)

fix $x\ y$

assume $(x,y) \in (S \cup R)^*$

also have $\dots \subseteq (S^* \circ R)^* \circ S^*$ by *regexp*

finally obtain z where $xz: (x,z) \in (S^* \circ R)^*$ and $zy: (z,y) \in S^*$ by *auto*

from xz have $(x,z) \in R^* \circ S^*$

proof (*induct rule: rtrancl-induct*)

case (*step z w*)

then have $(x,w) \in R^* \circ S^* \circ S^* \circ R$ by *auto*

also have $\dots \subseteq R^* \circ S^* \circ R$ by *regexp*

also have $\dots \subseteq R^* \circ R \circ S^*$ using *rtrancl-O-push* by *auto*

also have $\dots \subseteq R^* \circ S^*$ by *regexp*

finally show *?case*.

qed *auto*

with zy show $(x,y) \in R^* \circ S^*$ by *auto*

qed *regexp*

lemma *SN-on-O-push*:

assumes *SN*: *SN-on* $R\ X$ shows *SN-on* $(R \circ S^*)\ X$

proof

fix *seq*

have *SN*: *SN-on* $R\ (R^* \text{ `` } X)$ using *SN-on-Image-rtrancl[OF SN]*.

moreover assume $seq\ 0::nat) \in X$

then have $seq\ 0 \in R^* \text{ `` } X$ by *auto*

ultimately show $chain\ (R \circ S^*)\ seq \implies False$

proof(*induct seq 0 arbitrary: seq rule: SN-on-induct*)

case *IH*

then have $01: (seq\ 0, seq\ 1) \in R \circ S^*$

and $12: (seq\ 1, seq\ 2) \in R \circ S^*$

and $23: (seq\ 2, seq\ 3) \in R \circ S^*$ by (*auto simp: eval-nat-numeral*)

then obtain $s\ t$

where $s: (seq\ 0, s) \in R$ and $s1: (s, seq\ 1) \in S^*$

and $t: (seq\ 1, t) \in R$ and $t2: (t, seq\ 2) \in S^*$ by *auto*

from $s1\ t$ have $(s,t) \in S^* \circ R$ by *auto*

with *rtrancl-O-push* have $st: (s,t) \in R \circ S^*$ by *auto*

from $t2\ 23$ have $(t, seq\ 3) \in S^* \circ R \circ S^*$ by *auto*

also from *rtrancl-O-push* have $\dots \subseteq R \circ S^* \circ S^*$ by *blast*

finally have $t3: (t, seq\ 3) \in R \circ S^*$ by *regexp*

let $?seq = \lambda i. case\ i\ of\ 0 \Rightarrow s \mid Suc\ 0 \Rightarrow t \mid i \Rightarrow seq\ (Suc\ i)$

show *?case*

```

proof(rule IH)
  from  $s$  show  $(seq\ 0, ?seq\ 0) \in R$  by auto
  show  $chain\ (R\ O\ S^*)\ ?seq$ 
  proof (intro allI)
    fix  $i$  show  $(?seq\ i, ?seq\ (Suc\ i)) \in R\ O\ S^*$ 
    proof (cases i)
      case  $0$  with  $st$  show ?thesis by auto
    next
      case  $(Suc\ i)$  with  $t3\ IH$  show ?thesis by (cases i, auto simp: eval-nat-numeral)
    qed
  qed
qed
qed
qed

```

```

lemma SN-on-Image-push:
  assumes  $SN$ : SN-on  $R\ X$  shows SN-on  $R\ (S^* \text{ `` } X)$ 
proof -
  { fix  $n$ 
    have SN-on  $R\ ((S \text{ `` } ^n) \text{ `` } X)$ 
    proof(induct n)
      case  $0$  from  $SN$  show ?case by auto
      case  $(Suc\ n)$ 
        from SN-on-O-push[OF this] have SN-on  $(R\ O\ S^*)\ ((S \text{ `` } ^n) \text{ `` } X)$ .
        from SN-on-Image[OF this]
        have SN-on  $(R\ O\ S^*)\ ((R\ O\ S^*) \text{ `` } (S \text{ `` } ^n) \text{ `` } X)$ .
        then have SN-on  $R\ ((R\ O\ S^*) \text{ `` } (S \text{ `` } ^n) \text{ `` } X)$  by (rule SN-on-mono,
auto)
        from SN-on-subset2[OF Image-mono[OF push subset-refl] this]
        have SN-on  $R\ (R \text{ `` } (S \text{ `` } ^{Suc\ n}) \text{ `` } X)$  by (auto simp: relcomp-Image)
        then show ?case by fast
      qed
    }
  then show ?thesis by fast
qed

end

```

```

lemma not-SN-onI[intro]:  $f\ 0 \in X \implies chain\ R\ f \implies \neg\ SN-on\ R\ X$ 
  by (unfold SN-on-def not-not, intro exI conjI)
lemma shift-comp[simp]:  $shift\ (f\ o\ seq)\ n = f\ o\ (shift\ seq\ n)$  by auto

```

```

lemma Id-on-union:  $Id-on\ (A\ \cup\ B) = Id-on\ A\ \cup\ Id-on\ B$  unfolding Id-on-def
by auto

```

```

lemma relpow-union-cases:  $(a, d) \in (A\ \cup\ B) \text{ `` } ^n \implies (a, d) \in B \text{ `` } ^n \vee (\exists\ b\ c\ k\ m.$ 
 $(a, b) \in B \text{ `` } ^k \wedge (b, c) \in A \wedge (c, d) \in (A\ \cup\ B) \text{ `` } ^m \wedge n = Suc\ (k + m))$ 
proof (induct n arbitrary: a d)
  case  $(Suc\ n\ a\ e)$ 

```

```

let ?AB = A ∪ B
from Suc(2) obtain b where ab: (a,b) ∈ ?AB and be: (b,e) ∈ ?AB~n by (rule
relpow-Suc-E2)
from ab
show ?case
proof
  assume (a,b) ∈ A
  show ?thesis
  proof (rule disjI2, intro exI conjI)
    show Suc n = Suc (0 + n) by simp
    show (a,b) ∈ A by fact
  qed (insert be, auto)
next
assume ab: (a,b) ∈ B
from Suc(1)[OF be]
show ?thesis
proof
  assume (b,e) ∈ B~n
  with ab show ?thesis
  by (intro disjI1 relpow-Suc-I2)
next
assume ∃ c d k m. (b, c) ∈ B~k ∧ (c, d) ∈ A ∧ (d, e) ∈ ?AB~m ∧ n
= Suc (k + m)
then obtain c d k m where (b, c) ∈ B~k and *: (c, d) ∈ A (d, e) ∈ ?AB
~m n = Suc (k + m) by blast
with ab have ac: (a,c) ∈ B~(Suc k) by (intro relpow-Suc-I2)
show ?thesis
by (intro disjI2 exI conjI, rule ac, (rule *)+, simp add: *)
qed
qed
qed simp

```

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

```

assumes trans r refl r
shows r* = r
proof
  show r* ⊆ r
  proof
    fix x y
    assume (x,y) ∈ r*
    thus (x,y) ∈ r
    by (induct, insert assms, unfold refl-on-def trans-def, blast+)
  qed
qed regexp

```

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

```

assumes trans r refl r
shows r O r = r
proof (intro equalityI)

```

show $r \circ r \subseteq r$ **by** (*fact trans-O-subset* [*OF assms*(1)])
have $r \subseteq r \circ Id$ **by** *auto*
moreover have $Id \subseteq r$ **by** (*fact assms*(2)) [*unfolded refl-O-iff*]
ultimately show $r \subseteq r \circ r$ **by** *auto*
qed

lemma *relcomp3-I*:
assumes $(t, u) \in A$ **and** $(s, t) \in B$ **and** $(u, v) \in B$
shows $(s, v) \in B \circ A \circ B$
using *assms* **by** *blast*

lemma *relcomp3-transI*:
assumes *trans* B **and** $(t, u) \in B \circ A \circ B$ **and** $(s, t) \in B$ **and** $(u, v) \in B$
shows $(s, v) \in B \circ A \circ B$
using *assms* **by** (*auto simp: trans-def intro: relcomp3-I*)

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

lemma *qc-SN-relto-iff*:
assumes $r \circ s \subseteq s \circ (s \cup r)^*$
shows $SN (r^* \circ s \circ r^*) = SN s$
proof –
from *converse-mono* [*THEN iffD2*, *OF assms*]
have $*$: $s^{-1} \circ r^{-1} \subseteq (s^{-1} \cup r^{-1})^* \circ s^{-1}$ **unfolding** *converse-inward* .
have $(r^* \circ s \circ r^*)^{-1} = (r^{-1})^* \circ s^{-1} \circ (r^{-1})^*$
by (*simp only: converse-relcomp O-assoc rtrancl-converse*)
with *qc-wf-relto-iff* [*OF **]
show *?thesis* **by** (*simp add: SN-iff-wf*)
qed

lemma *conversion-empty* [*simp*]: *conversion* $\{\}$ = *Id*
by (*auto simp: conversion-def*)

lemma *symcl-idemp* [*simp*]: $(r^{\leftrightarrow})^{\leftrightarrow} = r^{\leftrightarrow}$ **by** *auto*

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 $R S \equiv S^{\wedge*} \circ R \circ S^{\wedge*}$

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

definition *SN-rel-on-alt* :: 'a rel \Rightarrow 'a rel \Rightarrow 'a set \Rightarrow bool **where**
SN-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-rel* :: 'a rel \Rightarrow 'a rel \Rightarrow bool **where**
SN-rel R S \equiv *SN-rel-on* R S UNIV

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

lemma *relto-absorb [simp]*: relto R E O E* = relto R E E* O relto R E = relto R E
using *O-assoc* **and** *rtrancl-idemp-self-comp* **by** (*metis*)+

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

proof –

let ?RS = relto R S

have (R \cup S)^{*} \subseteq S^{*} \cup ?RS^{*} **by** *regexp*

with *steps* **have** (a,b) \in S^{*} \vee (a,b) \in ?RS^{*} **by** *auto*

thus ?thesis

proof

assume (a,b) \in ?RS^{*}

from *steps-preserve-SN-on*[*OF this SN*] **show** ?thesis .

next

assume *Ssteps*: (a,b) \in S^{*}

show ?thesis

proof

fix f

assume f 0 \in {b} **and** chain ?RS f

hence f 0: f 0 = b **and** *steps*: $\bigwedge i$. (f i, f (Suc i)) \in ?RS **by** *auto*

let ?g = λi . if i = 0 then a else f i

have \neg *SN-on* ?RS {a} **unfolding** *SN-on-def not-not*

proof (*rule exI*[*of* - ?g], *intro conjI allI*)

fix i

show (?g i, ?g (Suc i)) \in ?RS

proof (*cases i*)

case (Suc j)

show ?thesis **using** *steps*[*of i*] **unfolding** *Suc* **by** *simp*

next

case 0

from *steps*[*of 0, unfolded f0*] *Ssteps* **have** *steps*: (a,f (Suc 0)) \in S^{*} O
?RS **by** *blast*

```

    have (a, f (Suc 0)) ∈ ?RS
      by (rule subsetD[OF - steps], regexp)
    thus ?thesis unfolding 0 by simp
  qed
qed simp
with SN show False by simp
qed
qed
qed

```

lemma *step-preserves-SN-on-relto*: **assumes** $st: (s, t) \in R \cup E$
and $SN: SN\text{-on } (relto\ R\ E)\ \{s\}$
shows $SN\text{-on } (relto\ R\ E)\ \{t\}$
by (rule *steps-preserve-SN-on-relto*[OF - SN], *insert st, auto*)

lemma *SN-rel-on-imp-SN-rel-on-alt*: $SN\text{-rel-on } R\ S\ T \implies SN\text{-rel-on-alt } R\ S\ T$

```

proof (unfold SN-rel-on-def)
  assume  $SN: SN\text{-on } (relto\ R\ S)\ T$ 
  show ?thesis
  proof (unfold SN-rel-on-alt-def, intro allI impI)
    fix f
    assume steps:  $chain\ (R \cup S)\ f \wedge f\ 0 \in T$ 
    with SN have  $SN: SN\text{-on } (relto\ R\ S)\ \{f\ 0\}$ 
      and steps:  $\bigwedge i. (f\ i, f\ (Suc\ i)) \in R \cup S$  unfolding SN-defs by auto
    obtain r where  $r: \bigwedge j. r\ j \equiv (f\ j, f\ (Suc\ j)) \in R$  by auto
    show  $\neg (INFM\ j. (f\ j, f\ (Suc\ j)) \in R)$ 
    proof (rule ccontr)
      assume  $\neg ?thesis$ 
      hence ih: infinitely-many r unfolding infinitely-many-def r by blast
      obtain r-index where  $r\text{-index} = \text{infinitely-many.index } r$  by simp
      with infinitely-many.index-p[OF ih] infinitely-many.index-ordered[OF ih] infinitely-many.index-not-p-between[OF ih]
        have  $r\text{-index}: \bigwedge i. r\ (r\text{-index } i) \wedge r\text{-index } i < r\text{-index } (Suc\ i) \wedge (\forall j. r\text{-index } i < j \wedge j < r\text{-index } (Suc\ i) \longrightarrow \neg r\ j)$  by auto
      obtain g where  $g: \bigwedge i. g\ i \equiv f\ (r\text{-index } i) ..$ 
      {
        fix i
        let ?ri =  $r\text{-index } i$ 
        let ?rsi =  $r\text{-index } (Suc\ i)$ 
        from r-index have isi:  $?ri < ?rsi$  by auto
        obtain ri rsi where  $ri: ri = ?ri$  and  $rsi: rsi = ?rsi$  by auto
        with r-index[of i] steps have inter:  $\bigwedge j. ri < j \wedge j < rsi \implies (f\ j, f\ (Suc\ j)) \in S$  unfolding r by auto
        from ri isi rsi have risi:  $ri < rsi$  by simp
        {
          fix n
          assume  $Suc\ n \leq rsi - ri$ 
          hence  $(f\ (Suc\ ri), f\ (Suc\ (n + ri))) \in S^{\widehat{*}}$ 
          proof (induct n, simp)

```

```

      case (Suc n)
      hence stepps: (f (Suc ri), f (Suc (n+ri))) ∈ S∗ by simp
      have (f (Suc (n+ri)), f (Suc (Suc n + ri))) ∈ S
        using inter[of Suc n + ri] Suc(2) by auto
      with stepps show ?case by simp
    qed
  }
  from this[of rsi - ri - 1] risi have
    (f (Suc ri), f rsi) ∈ S∗ by simp
  with ri risi have ssteps: (f (Suc ?ri), f ?rsi) ∈ S∗ by simp
  with r-index[of i] have (f ?ri, f ?rsi) ∈ R O S∗ unfolding r by auto
  hence (g i, g (Suc i)) ∈ S∗ O R O S∗ using rtrancl-refl unfolding g by
auto
  }
  hence nSN: ¬ SN-on (S∗ O R O S∗) {g 0} unfolding SN-defs by blast
  have SN: SN-on (S∗ O R O S∗) {f (r-index 0)}
  proof (rule steps-preserve-SN-on-relto[OF - SN])
    show (f 0, f (r-index 0)) ∈ (R ∪ S)∗
      unfolding rtrancl-fun-conv
      by (rule exI[of - f], rule exI[of - r-index 0], insert steps, auto)
    qed
  with nSN show False unfolding g ..
  qed
  qed
  qed
  qed

lemma SN-rel-on-alt-imp-SN-rel-on: SN-rel-on-alt R S T ⇒ SN-rel-on R S T
proof (unfold SN-rel-on-def)
  assume SN: SN-rel-on-alt R S T
  show SN-on (relto R S) T
  proof
    fix f
    assume start: f 0 ∈ T and chain (relto R S) f
    hence steps: ∧ i. (f i, f (Suc i)) ∈ S∗ O R O S∗ by auto
    let ?prop = λ i ai bi. (f i, bi) ∈ S∗ ∧ (bi, ai) ∈ R ∧ (ai, f (Suc i)) ∈ S∗
    {
      fix i
      from steps obtain bi ai where ?prop i ai bi by blast
      hence ∃ ai bi. ?prop i ai bi by blast
    }
    hence ∀ i. ∃ bi ai. ?prop i ai bi by blast
    from choice[OF this] obtain b where ∀ i. ∃ ai. ?prop i ai (b i) by blast
    from choice[OF this] obtain a where steps: ∧ i. ?prop i (a i) (b i) by blast
    from steps[of 0] have fa0: (f 0, a 0) ∈ S∗ O R by auto
    let ?prop = λ i li. (b i, a i) ∈ R ∧ (∀ j < length li. ((a i # li) ! j, (a i # li) !
Suc j) ∈ S) ∧ last (a i # li) = b (Suc i)
    {
      fix i
      from steps[of i] steps[of Suc i] have (a i, f (Suc i)) ∈ S∗ and (f (Suc i), b

```

```

(Suc i) ∈ S∧* by auto
  from rtrancl-trans[OF this] steps[of i] have R: (b i, a i) ∈ R and S: (a i, b
(Suc i) ∈ S∧* by blast+
  from S[unfolded rtrancl-list-conv] obtain li where last (a i # li) = b (Suc
i) ∧ (∀ j < length li. ((a i # li) ! j, (a i # li) ! Suc j) ∈ S) ..
  with R have ?prop i li by blast
  hence ∃ li. ?prop i li ..
}
hence ∀ i. ∃ li. ?prop i li ..
from choice[OF this] obtain l where steps: ∧ i. ?prop i (l i) by auto
let ?p = λ i. ?prop i (l i)
from steps have steps: ∧ i. ?p i by blast
let ?l = λ i. a i # l i
let ?l' = λ i. length (?l i)
let ?g = λ i. inf-concat-simple ?l' i
obtain g where g: ∧ i. g i = (let (ii,jj) = ?g i in ?l ii ! jj) by auto
have g0: g 0 = a 0 unfolding g Let-def by simp
with fa0 have fg0: (f 0, g 0) ∈ S∧* O R by auto
have fg0: (f 0, g 0) ∈ (R ∪ S)∧*
  by (rule subsetD[OF - fg0], regexp)
have len: ∧ i j n. ?g n = (i,j) ⇒ j < length (?l i)
proof -
  fix i j n
  assume n: ?g n = (i,j)
  show j < length (?l i)
  proof (cases n)
    case 0
    with n have j = 0 by auto
    thus ?thesis by simp
  next
    case (Suc nn)
    obtain ii jj where nn: ?g nn = (ii,jj) by (cases ?g nn, auto)
    show ?thesis
    proof (cases Suc jj < length (?l ii))
      case True
      with nn Suc have ?g n = (ii, Suc jj) by auto
      with n True show ?thesis by simp
    next
      case False
      with nn Suc have ?g n = (Suc ii, 0) by auto
      with n show ?thesis by simp
    qed
  qed
qed
have gsteps: ∧ i. (g i, g (Suc i)) ∈ R ∪ S
proof -
  fix n
  obtain i j where n: ?g n = (i, j) by (cases ?g n, auto)
  show (g n, g (Suc n)) ∈ R ∪ S

```

```

proof (cases Suc j < length (?l i))
  case True
    with n have ?g (Suc n) = (i, Suc j) by auto
    with n have gn: g n = ?l i ! j and gsn: g (Suc n) = ?l i ! (Suc j) unfolding
g by auto
    thus ?thesis using steps[of i] True by auto
  next
    case False
    with n have ?g (Suc n) = (Suc i, 0) by auto
    with n have gn: g n = ?l i ! j and gsn: g (Suc n) = a (Suc i) unfolding
g by auto
    from gn len[OF n] False have j = length (?l i) - 1 by auto
    with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
    from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
  qed
qed
have infR: INFM j. (g j, g (Suc j)) ∈ R unfolding INFM-nat-le
proof
  fix n
  obtain i j where n: ?g n = (i,j) by (cases ?g n, auto)
  from len[OF n] have j: j < ?l' i .
  let ?k = ?l' i - 1 - j
  obtain k where k: k = j + ?k by auto
  from j k have k2: k = ?l' i - 1 and k3: j + ?k < ?l' i by auto
  from inf-concat-simple-add[OF n, of ?k, OF k3]
  have gnk: ?g (n + ?k) = (i, k) by (simp only: k)
  hence g (n + ?k) = ?l i ! k unfolding g by auto
  hence gnk2: g (n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
  from k2 gnk have ?g (Suc (n+?k)) = (Suc i, 0) by auto
  hence gnsk2: g (Suc (n+?k)) = a (Suc i) unfolding g by auto
  from steps[of i] steps[of Suc i] have main: (g (n+?k), g (Suc (n+?k))) ∈ R
  by (simp only: gnk2 gnsk2)
  show ∃ j ≥ n. (g j, g (Suc j)) ∈ R
  by (rule exI[of - n + ?k], auto simp: main[simplified])
qed
from fg0[unfolded rtrancl-fun-conv] obtain gg n where start: gg 0 = f 0
  and n: gg n = g 0 and steps: ∧ i. i < n ⇒ (gg i, gg (Suc i)) ∈ R ∪ S by
auto
  let ?h = λ i. if i < n then gg i else g (i - n)
  obtain h where h: h = ?h by auto
  {
    fix i
    assume i: i ≤ n
    have h i = gg i using i unfolding h
    by (cases i < n, auto simp: n)
  } note gg = this
from gg[of 0] ⟨f 0 ∈ T⟩ have h0: h 0 ∈ T unfolding start by auto
  {
    fix i

```

```

have (h i, h (Suc i)) ∈ R ∪ S
proof (cases i < n)
  case True
    from steps[of i] gg[of i] gg[of Suc i] True show ?thesis by auto
  next
    case False
      hence i = n + (i - n) by auto
      then obtain k where i: i = n + k by auto
      from gsteps[of k] show ?thesis unfolding h i by simp
    qed
  } note hsteps = this
from SN[unfolded SN-rel-on-alt-def, rule-format, OF conjI[OF allI[OF hsteps]
h0]]
have ¬ (INFM j. (h j, h (Suc j)) ∈ R) .
moreover have INFM j. (h j, h (Suc j)) ∈ R unfolding INFM-nat-le
proof (rule)
  fix m
  from infR[unfolded INFM-nat-le, rule-format, of m]
  obtain i where i: i ≥ m and g: (g i, g (Suc i)) ∈ R by auto
  show ∃ n ≥ m. (h n, h (Suc n)) ∈ R
  by (rule exI[of - i + n], unfold h, insert g i, auto)
qed
ultimately show False ..
qed
qed

```

lemma SN-rel-on-conv: SN-rel-on = SN-rel-on-alt
by (intro ext) (blast intro: SN-rel-on-imp-SN-rel-on-alt SN-rel-on-alt-imp-SN-rel-on)

lemmas SN-rel-defs = SN-rel-on-def SN-rel-on-alt-def

lemma SN-rel-on-alt-r-empty : SN-rel-on-alt {} S T
unfolding SN-rel-defs **by** auto

lemma SN-rel-on-alt-s-empty : SN-rel-on-alt R {} = SN-on R
by (intro ext, unfold SN-rel-defs SN-defs, auto)

lemma SN-rel-on-mono':
assumes R: R ⊆ R' **and** S: S ⊆ R' ∪ S' **and** SN: SN-rel-on R' S' T
shows SN-rel-on R S T

proof –
note conv = SN-rel-on-conv SN-rel-on-alt-def INFM-nat-le
show ?thesis **unfolding** conv
proof(intro allI impI)
fix f
assume chain (R ∪ S) f ∧ f 0 ∈ T
with R S **have** chain (R' ∪ S') f ∧ f 0 ∈ T **by** auto
from SN[unfolded conv, rule-format, OF this]

show $\neg (\forall m. \exists n \geq m. (f\ n, f\ (Suc\ n)) \in R)$ **using** R **by** *auto*
qed
qed

lemma *relto-mono*:
assumes $R \subseteq R'$ **and** $S \subseteq S'$
shows $relto\ R\ S \subseteq relto\ R'\ S'$
using *assms rtrancl-mono* **by** *blast*

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 $SN\text{-rel-on}\ R\ S\ T$
using SN
unfolding *SN-rel-on-def* **using** $SN\text{-on-mono}[OF\ \text{-}\ relto\text{-mono}[OF\ R\ S]]$ **by** *blast*

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
fix f
assume $chain\ R\ f$
and $f0: f\ 0 \in T$
hence $\bigwedge i. (f\ i, f\ (Suc\ i)) \in relto\ R\ S$ **by** *blast*
thus *False* **using** *assms f0* **unfolding** *SN-rel-on-def* *SN-defs* **by** *blast*
qed

lemma *relto-Id*: $relto\ R\ (S \cup Id) = relto\ R\ S$ **by** *simp*

lemma *SN-rel-on-Id*:
shows $SN\text{-rel-on}\ R\ (S \cup Id)\ T = SN\text{-rel-on}\ R\ S\ T$
unfolding *SN-rel-on-def* **by** (*simp only: relto-Id*)

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

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
assume $?L$
show $?R$
proof
assume $\exists as. ideriv\ R\ S\ as \wedge as\ 0 \in T$
then obtain as **where** $id: ideriv\ R\ S\ as$ **and** $T: as\ 0 \in T$ **by** *auto*
note $id = id[unfolded\ ideriv-def]$
from $\langle ?L \rangle[unfolded\ SN\text{-rel-on-conv}\ SN\text{-rel-on-alt-def, THEN}\ spec[of\ \text{-}\ as]]$
id T **obtain** i **where** $i: \bigwedge j. j \geq i \implies (as\ j, as\ (Suc\ j)) \notin R$ **by** *auto*
with $id[unfolded\ INFM\text{-nat, THEN}\ conjunct2, THEN]\ spec[of\ \text{-}\ Suc\ i]$ **show**

```

False by auto
qed
next
  assume ?R
  show ?L
    unfolding SN-rel-on-conv SN-rel-on-alt-def
    proof(intro allI impI)
      fix as
      assume chain (R ∪ S) as ∧ as 0 ∈ T
      with ⟨?R⟩[unfolded ideriv-def] have ¬ (INFM i. (as i, as (Suc i)) ∈ R) by
auto
      from this[unfolded INFM-nat] obtain i where i: ∧j. i < j ⇒ (as j, as (Suc
j)) ∉ R by auto
      show ¬ (INFM j. (as j, as (Suc j)) ∈ R) unfolding INFM-nat using i by
blast
    qed
  qed
qed

lemma SN-rel-to-SN-rel-alt: SN-rel R S ⇒ SN-rel-alt R S
proof (unfold SN-rel-on-def)
  assume SN: SN (relto R S)
  show ?thesis
  proof (unfold SN-rel-on-alt-def, intro allI impI)
    fix f
    presume steps: chain (R ∪ S) f
    obtain r where r: ∧j. r j ≡ (f j, f (Suc j)) ∈ R by auto
    show ¬ (INFM j. (f j, f (Suc j)) ∈ R)
    proof (rule ccontr)
      assume ¬ ?thesis
      hence ih: infinitely-many r unfolding infinitely-many-def r by blast
      obtain r-index where r-index = infinitely-many.index r by simp
      with infinitely-many.index-p[OF ih] infinitely-many.index-ordered[OF ih] in-
finitely-many.index-not-p-between[OF ih]
      have r-index: ∧ i. r (r-index i) ∧ r-index i < r-index (Suc i) ∧ (∀ j. r-index
i < j ∧ j < r-index (Suc i) ⇒ ¬ r j) by auto
      obtain g where g: ∧ i. g i ≡ f (r-index i) ..
      {
        fix i
        let ?ri = r-index i
        let ?rsi = r-index (Suc i)
        from r-index have isi: ?ri < ?rsi by auto
        obtain ri rsi where ri: ri = ?ri and rsi: rsi = ?rsi by auto
        with r-index[of i] steps have inter: ∧ j. ri < j ∧ j < rsi ⇒ (f j, f (Suc
j)) ∈ S unfolding r by auto
        from ri isi rsi have risi: ri < rsi by simp
        {
          fix n
          assume Suc n ≤ rsi - ri
          hence (f (Suc ri), f (Suc (n + ri))) ∈ S∞

```



```

proof (induct n, simp)
  case (Suc n)
  hence stepps: (f (Suc ri), f (Suc (n+ri))) ∈ S∗ by simp
  have (f (Suc (n+ri)), f (Suc (Suc n + ri))) ∈ S
    using inter[of Suc n + ri] Suc(2) by auto
  with stepps show ?case by simp
qed
}
from this[of rsi - ri - 1] rsi have
  (f (Suc ri), f rsi) ∈ S∗ by simp
with ri rsi have ssteps: (f (Suc ?ri), f ?rsi) ∈ S∗ by simp
with r-index[of i] have (f ?ri, f ?rsi) ∈ R O S∗ unfolding r by auto
hence (g i, g (Suc i)) ∈ S∗ O R O S∗ using rtrancl-refl unfolding g by
auto
}
hence ¬ SN (S∗ O R O S∗) unfolding SN-defs by blast
with SN show False by simp
qed
qed simp
qed

```

lemma SN-rel-alt-to-SN-rel : SN-rel-alt R S \implies SN-rel R S

```

proof (unfold SN-rel-on-def)
  assume SN: SN-rel-alt R S
  show SN (relto R S)
  proof
    fix f
    assume chain (relto R S) f
    hence steps:  $\bigwedge i. (f i, f (Suc i)) \in S^{\ast} O R O S^{\ast}$  by auto
    let ?prop =  $\lambda i ai bi. (f i, bi) \in S^{\ast} \wedge (bi, ai) \in R \wedge (ai, f (Suc i)) \in S^{\ast}$ 
    {
      fix i
      from steps obtain bi ai where ?prop i ai bi by blast
      hence  $\exists ai bi. ?prop i ai bi$  by blast
    }
    hence  $\forall i. \exists bi ai. ?prop i ai bi$  by blast
    from choice[OF this] obtain b where  $\forall i. \exists ai. ?prop i ai (b i)$  by blast
    from choice[OF this] obtain a where steps:  $\bigwedge i. ?prop i (a i) (b i)$  by blast
    let ?prop =  $\lambda i li. (b i, a i) \in R \wedge (\forall j < length li. ((a i \# li) ! j, (a i \# li) !$ 
    Suc j) ∈ S)  $\wedge last (a i \# li) = b (Suc i)$ 
    {
      fix i
      from steps[of i] steps[of Suc i] have (a i, f (Suc i)) ∈ S∗ and (f (Suc i), b
      (Suc i)) ∈ S∗ by auto
      from rtrancl-trans[OF this] steps[of i] have R: (b i, a i) ∈ R and S: (a i, b
      (Suc i)) ∈ S∗ by blast+
      from S[unfolded rtrancl-list-conv] obtain li where last (a i # li) = b (Suc
      i)  $\wedge (\forall j < length li. ((a i \# li) ! j, (a i \# li) ! Suc j) \in S)$  ..
      with R have ?prop i li by blast
    }
  }

```

```

    hence  $\exists li. ?prop\ i\ li\ ..$ 
  }
  hence  $\forall i. \exists li. ?prop\ i\ li\ ..$ 
  from choice[OF this] obtain  $l$  where steps:  $\bigwedge i. ?prop\ i\ (l\ i)$  by auto
  let  $?p = \lambda i. ?prop\ i\ (l\ i)$ 
  from steps have steps:  $\bigwedge i. ?p\ i$  by blast
  let  $?l = \lambda i. a\ i\ \# l\ i$ 
  let  $?l' = \lambda i. length\ (?l\ i)$ 
  let  $?g = \lambda i. inf-concat-simple\ ?l'\ i$ 
  obtain  $g$  where  $g: \bigwedge i. g\ i = (let\ (ii,jj) = ?g\ i\ in\ ?l\ ii\ !\ jj)$  by auto
  have len:  $\bigwedge i\ j\ n. ?g\ n = (i,j) \implies j < length\ (?l\ i)$ 
  proof -
    fix  $i\ j\ n$ 
    assume  $n: ?g\ n = (i,j)$ 
    show  $j < length\ (?l\ i)$ 
    proof (cases  $n$ )
      case 0
      with  $n$  have  $j = 0$  by auto
      thus ?thesis by simp
    next
      case (Suc  $nn$ )
      obtain  $ii\ jj$  where  $nn: ?g\ nn = (ii,jj)$  by (cases  $?g\ nn$ , auto)
      show ?thesis
      proof (cases  $Suc\ jj < length\ (?l\ ii)$ )
        case True
        with  $nn\ Suc$  have  $?g\ n = (ii, Suc\ jj)$  by auto
        with  $n\ True$  show ?thesis by simp
      next
        case False
        with  $nn\ Suc$  have  $?g\ n = (Suc\ ii, 0)$  by auto
        with  $n$  show ?thesis by simp
      qed
    qed
  qed
  have gsteps:  $\bigwedge i. (g\ i, g\ (Suc\ i)) \in R \cup S$ 
  proof -
    fix  $n$ 
    obtain  $i\ j$  where  $n: ?g\ n = (i, j)$  by (cases  $?g\ n$ , auto)
    show  $(g\ n, g\ (Suc\ n)) \in R \cup S$ 
    proof (cases  $Suc\ j < length\ (?l\ i)$ )
      case True
      with  $n$  have  $?g\ (Suc\ n) = (i, Suc\ j)$  by auto
      with  $n$  have  $gn: g\ n = ?l\ i\ !\ j$  and  $gsn: g\ (Suc\ n) = ?l\ i\ !\ (Suc\ j)$  unfolding
g by auto
      thus ?thesis using steps[of i] True by auto
    next
      case False
      with  $n$  have  $?g\ (Suc\ n) = (Suc\ i, 0)$  by auto
      with  $n$  have  $gn: g\ n = ?l\ i\ !\ j$  and  $gsn: g\ (Suc\ n) = a\ (Suc\ i)$  unfolding

```

g by *auto*
from gn $len[OF\ n]$ *False* **have** $j = length\ (?l\ i) - 1$ **by** *auto*
with gn **have** $gn: g\ n = last\ (?l\ i)$ **using** *last-conv-nth[of ?l i]* **by** *auto*
from gn gsn **show** *?thesis* **using** *steps[of i] steps[of Suc i]* **by** *auto*
qed
qed
have *infR: INFM j. (g j, g (Suc j)) ∈ R* **unfolding** *INFM-nat-le*
proof
fix n
obtain $i\ j$ **where** $n: ?g\ n = (i, j)$ **by** (*cases ?g n, auto*)
from $len[OF\ n]$ **have** $j: j < ?l'\ i$.
let $?k = ?l'\ i - 1 - j$
obtain k **where** $k: k = j + ?k$ **by** *auto*
from $j\ k$ **have** $k2: k = ?l'\ i - 1$ **and** $k3: j + ?k < ?l'\ i$ **by** *auto*
from *inf-concat-simple-add[OF n, of ?k, OF k3]*
have $gnk: ?g\ (n + ?k) = (i, k)$ **by** (*simp only: k*)
hence $g\ (n + ?k) = ?l\ i\ !\ k$ **unfolding** g **by** *auto*
hence $gnk2: g\ (n + ?k) = last\ (?l\ i)$ **using** *last-conv-nth[of ?l i] k2* **by** *auto*
from $k2\ gnk$ **have** $?g\ (Suc\ (n + ?k)) = (Suc\ i, 0)$ **by** *auto*
hence $gnsk2: g\ (Suc\ (n + ?k)) = a\ (Suc\ i)$ **unfolding** g **by** *auto*
from *steps[of i] steps[of Suc i]* **have** *main: (g (n + ?k), g (Suc (n + ?k))) ∈ R*
by (*simp only: gnk2 gnsk2*)
show $\exists\ j \geq n. (g\ j, g\ (Suc\ j)) \in R$
by (*rule exI[of - n + ?k], auto simp: main[simplified]*)
qed
from *SN[unfolded SN-rel-on-alt-def] gsteps infR* **show** *False* **by** *blast*
qed
qed

lemma *SN-rel-alt-r-empty : SN-rel-alt {} S*
unfolding *SN-rel-defs* **by** *auto*

lemma *SN-rel-alt-s-empty : SN-rel-alt R {} = SN R*
unfolding *SN-rel-defs SN-defs* **by** *auto*

lemma *SN-rel-mono'*:
 $R \subseteq R' \implies S \subseteq R' \cup S' \implies SN\text{-rel}\ R'\ S' \implies SN\text{-rel}\ R\ S$
unfolding *SN-rel-on-conv SN-rel-defs INFM-nat-le*
by (*metis contra-subsetD sup.left-idem sup.mono*)

lemma *SN-rel-mono*:
assumes $R: R \subseteq R'$ **and** $S: S \subseteq S'$ **and** $SN: SN\text{-rel}\ R'\ S'$
shows *SN-rel R S*
using SN **unfolding** *SN-rel-defs* **using** *SN-subset[OF - relto-mono[OF R S]]* **by**
blast

lemmas *SN-rel-alt-mono = SN-rel-mono[unfolded SN-rel-on-conv]*

lemma *SN-rel-imp-SN : assumes SN-rel R S shows SN R*

proof
fix f
assume $\forall i. (f\ i, f\ (Suc\ i)) \in R$
hence $\bigwedge i. (f\ i, f\ (Suc\ i)) \in relto\ R\ S$ **by** *blast*
thus *False* **using** *assms* **unfolding** *SN-rel-defs* *SN-defs* **by** *fast*
qed

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

lemma *SN-rel-Id*:
shows $SN-rel\ R\ (S \cup Id) = SN-rel\ R\ S$
unfolding *SN-rel-defs* **by** (*simp* *only*: *relto-Id*)

lemma *relto-rtrancl*: $relto\ R\ (S^{\wedge*}) = relto\ R\ S$ **by** *regexp*

lemma *SN-rel-empty[simp]*: $SN-rel\ R\ \{\} = SN\ R$
unfolding *SN-rel-defs* **by** *auto*

lemma *SN-rel-ideriv*: $SN-rel\ R\ S = (\neg (\exists\ as. ideriv\ R\ S\ as))$ (**is** $?L = ?R$)
proof
assume $?L$
show $?R$
proof
assume $\exists\ as. ideriv\ R\ S\ as$
then obtain as **where** $id: ideriv\ R\ S\ as$ **by** *auto*
note $id = id[unfolded\ ideriv-def]$
from $\langle ?L \rangle[unfolded\ SN-rel-on-conv\ SN-rel-defs, THEN\ spec[of - as]]$
id obtain i **where** $i: \bigwedge j. j \geq i \implies (as\ j, as\ (Suc\ j)) \notin R$ **by** *auto*
with $id[unfolded\ INFM-nat, THEN\ conjunct2, THEN\ spec[of - Suc\ i]]$ **show**
False **by** *auto*
qed
next
assume $?R$
show $?L$
unfolding *SN-rel-on-conv* *SN-rel-defs*
proof (*intro* *allI* *impI*)
fix as
presume $chain\ (R \cup S)\ as$
with $\langle ?R \rangle[unfolded\ ideriv-def]$ **have** $\neg (INFM\ i. (as\ i, as\ (Suc\ i)) \in R)$ **by**
auto
from $this[unfolded\ INFM-nat]$ **obtain** i **where** $i: \bigwedge j. i < j \implies (as\ j, as\ (Suc\ j)) \notin R$ **by** *auto*
show $\neg (INFM\ j. (as\ j, as\ (Suc\ j)) \in R)$ **unfolding** *INFM-nat* **using** i **by**
blast
qed *simp*
qed

lemma *SN-rel-map*:

```

fixes  $R\ Rw\ R'\ Rw' :: 'a\ rel$ 
defines  $A: A \equiv R' \cup Rw'$ 
assumes  $SN: SN\text{-rel}\ R'\ Rw'$ 
and  $R: \bigwedge s\ t. (s,t) \in R \implies (f\ s, f\ t) \in A^{\widehat{*}}\ O\ R'\ O\ A^{\widehat{*}}$ 
and  $Rw: \bigwedge s\ t. (s,t) \in Rw \implies (f\ s, f\ t) \in A^{\widehat{*}}$ 
shows  $SN\text{-rel}\ R\ Rw$ 
unfolding  $SN\text{-rel}\text{-defs}$ 
proof
  fix  $g$ 
  assume  $steps: chain\ (relto\ R\ Rw)\ g$ 
  let  $?f = \lambda i. (f\ (g\ i))$ 
  obtain  $h$  where  $h: h = ?f$  by  $auto$ 
  {
    fix  $i$ 
    let  $?m = \lambda (x,y). (f\ x, f\ y)$ 
    {
      fix  $s\ t$ 
      assume  $(s,t) \in Rw^{\widehat{*}}$ 
      hence  $?m\ (s,t) \in A^{\widehat{*}}$ 
      proof ( $induct$ )
        case  $base$  show  $?case$  by  $simp$ 
      next
        case ( $step\ t\ u$ )
        from  $Rw[OF\ step(2)]\ step(3)$ 
        show  $?case$  by  $auto$ 
      qed
    }
    note  $Rw = this$ 
    from  $steps$  have  $(g\ i, g\ (Suc\ i)) \in relto\ R\ Rw ..$ 
    from  $this$ 
    obtain  $s\ t$  where  $gs: (g\ i, s) \in Rw^{\widehat{*}}$  and  $st: (s, t) \in R$  and  $tg: (t, g\ (Suc\ i))$ 
     $\in Rw^{\widehat{*}}$  by  $auto$ 
    from  $Rw[OF\ gs]\ R[OF\ st]\ Rw[OF\ tg]$ 
    have  $step: (?f\ i, ?f\ (Suc\ i)) \in A^{\widehat{*}}\ O\ (A^{\widehat{*}}\ O\ R'\ O\ A^{\widehat{*}})\ O\ A^{\widehat{*}}$ 
    by  $fast$ 
    have  $(?f\ i, ?f\ (Suc\ i)) \in A^{\widehat{*}}\ O\ R'\ O\ A^{\widehat{*}}$ 
    by ( $rule\ subsetD[OF\ step],\ regexp$ )
    hence  $(h\ i, h\ (Suc\ i)) \in (relto\ R'\ Rw')^{\widehat{+}}$ 
    unfolding  $A\ h\ relto\text{-trancl}\text{-conv} .$ 
  }
  hence  $\neg SN\ ((relto\ R'\ Rw')^{\widehat{+}})$  by  $auto$ 
  with  $SN\text{-imp}\text{-}SN\text{-trancl}[OF\ SN[unfolding\ SN\text{-rel}\text{-on}\text{-def}]]$ 
  show  $False$  by  $simp$ 
qed

```

```

datatype  $SN\text{-rel}\text{-ext}\text{-type} = top\text{-}s \mid top\text{-}ns \mid normal\text{-}s \mid normal\text{-}ns$ 

```

```

fun  $SN\text{-rel}\text{-ext}\text{-step} :: 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel \Rightarrow SN\text{-rel}\text{-ext}\text{-type} \Rightarrow 'a\ rel$ 
where
   $SN\text{-rel}\text{-ext}\text{-step}\ P\ Pw\ R\ Rw\ top\text{-}s = P$ 

```

| *SN-rel-ext-step* $P \ Pw \ R \ Rw \ top\text{-}ns = Pw$
| *SN-rel-ext-step* $P \ Pw \ R \ Rw \ normal\text{-}s = R$
| *SN-rel-ext-step* $P \ Pw \ R \ Rw \ normal\text{-}ns = Rw$

definition *SN-rel-ext* :: 'a rel \Rightarrow 'a rel \Rightarrow 'a rel \Rightarrow 'a rel \Rightarrow ('a \Rightarrow bool) \Rightarrow bool
where

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

lemma *SN-rel-ext-step-mono*: **assumes** $P \subseteq P' \ Pw \subseteq Pw' \ R \subseteq R' \ Rw \subseteq Rw'$
shows *SN-rel-ext-step* $P \ Pw \ R \ Rw \ t \subseteq \text{SN-rel-ext-step } P' \ Pw' \ R' \ Rw' \ t$
using *assms*
by (*cases t, auto*)

lemma *SN-rel-ext-mono*: **assumes** *subset*: $P \subseteq P' \ Pw \subseteq Pw' \ R \subseteq R' \ Rw \subseteq Rw'$
and

SN: *SN-rel-ext* $P' \ Pw' \ R' \ Rw' \ M$ **shows** *SN-rel-ext* $P \ Pw \ R \ Rw \ M$
using *SN-rel-ext-step-mono*[*OF subset*] *SN unfolding SN-rel-ext-def* **by** *blast*

lemma *SN-rel-ext-trans*:

fixes $P \ Pw \ R \ Rw :: 'a \text{ rel}$ **and** $M :: 'a \Rightarrow \text{bool}$
defines $M' : M' \equiv \{(s, t). M \ t\}$
defines $A : A \equiv (P \cup Pw \cup R \cup Rw) \cap M'$
assumes *SN-rel-ext* $P \ Pw \ R \ Rw \ M$
shows *SN-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-rel-ext* $?P \ ?Pw \ ?R \ ?Rw \ M$)
proof (*rule ccontr*)
let $?relt = \text{SN-rel-ext-step } ?P \ ?Pw \ ?R \ ?Rw$
let $?rel = \text{SN-rel-ext-step } P \ Pw \ R \ Rw$
assume $\neg ?thesis$
from *this*[*unfolded SN-rel-ext-def*]
obtain $f \ ty$
where *steps*: $\bigwedge i. (f \ i, f \ (Suc \ i)) \in ?relt \ (ty \ i)$
and *min*: $\bigwedge i. M \ (f \ i)$
and *inf1*: $\text{INFM } i. ty \ i \in \{top\text{-}s, top\text{-}ns\}$
and *inf2*: $\text{INFM } i. ty \ i \in \{top\text{-}s, normal\text{-}s\}$
by *auto*
let $?Un = \lambda \ tt. \bigcup (\ ?rel \ ' \ tt)$
let $?UnM = \lambda \ tt. (\bigcup (\ ?rel \ ' \ tt)) \cap M'$
let $?A = ?UnM \ \{top\text{-}s, top\text{-}ns, normal\text{-}s, normal\text{-}ns\}$
let $?P' = ?UnM \ \{top\text{-}s\}$
let $?Pw' = ?UnM \ \{top\text{-}s, top\text{-}ns\}$
let $?R' = ?UnM \ \{top\text{-}s, normal\text{-}s\}$
let $?Rw' = ?UnM \ \{top\text{-}s, top\text{-}ns, normal\text{-}s, normal\text{-}ns\}$
have $A : A = ?A$ **unfolding** A **by** *auto*

have $P: (P \cap M') = ?P'$ **by** *auto*
have $Pw: (P \cup Pw) \cap M' = ?Pw'$ **by** *auto*
have $R: (P \cup R) \cap M' = ?R'$ **by** *auto*
have $Rw: A = ?Rw'$ **unfolding** A ..
{
 fix $s\ t\ tt$
 assume $m: M\ s$ **and** $st: (s,t) \in ?UnM\ tt$
 hence $\exists\ typ \in tt. (s,t) \in ?rel\ typ \wedge M\ s \wedge M\ t$ **unfolding** M' **by** *auto*
} **note** *one-step* = *this*
let $?seq = \lambda\ s\ t\ g\ n\ ty. s = g\ 0 \wedge t = g\ n \wedge (\forall\ i < n. (g\ i, g\ (Suc\ i)) \in ?rel\ (ty\ i)) \wedge (\forall\ i \leq n. M\ (g\ i))$
{
 fix $s\ t$
 assume $m: M\ s$ **and** $st: (s,t) \in A^{\widehat{*}}$
 from $st[unfolded\ rtrancl\ fun\ conv]$
 obtain $g\ n$ **where** $g0: g\ 0 = s$ **and** $gn: g\ n = t$ **and** $steps: \bigwedge\ i. i < n \implies (g\ i, g\ (Suc\ i)) \in ?A$ **unfolding** A **by** *auto*
 {
 fix i
 assume $i \leq n$
 have $M\ (g\ i)$
 proof (*cases* i)
 case 0
 show $?thesis$ **unfolding** $0\ g0$ **by** (*rule* m)
 next
 case ($Suc\ j$)
 with $\langle i \leq n \rangle$ **have** $j < n$ **by** *auto*
 from $steps[OF\ this]$ **show** $?thesis$ **unfolding** $Suc\ M'$ **by** *auto*
 } **qed**
 } **note** $min = this$
 {
 fix i
 assume $i: i < n$ **hence** $i': i \leq n$ **by** *auto*
 from i' $one\ step[OF\ min\ steps[OF\ i]]$
 have $\exists\ ty. (g\ i, g\ (Suc\ i)) \in ?rel\ ty$ **by** *blast*
 }
 hence $\forall\ i. (\exists\ ty. i < n \longrightarrow (g\ i, g\ (Suc\ i)) \in ?rel\ ty)$ **by** *auto*
 from $choice[OF\ this]$
 obtain tt **where** $steps: \bigwedge\ i. i < n \implies (g\ i, g\ (Suc\ i)) \in ?rel\ (tt\ i)$ **by** *auto*
 from $g0\ gn\ steps\ min$
 have $?seq\ s\ t\ g\ n\ tt$ **by** *auto*
 hence $\exists\ g\ n\ tt. ?seq\ s\ t\ g\ n\ tt$ **by** *blast*
} **note** $A\ steps = this$
let $?seqtt = \lambda\ s\ t\ tt\ g\ n\ ty. s = g\ 0 \wedge t = g\ n \wedge n > 0 \wedge (\forall\ i < n. (g\ i, g\ (Suc\ i)) \in ?rel\ (ty\ i)) \wedge (\forall\ i \leq n. M\ (g\ i)) \wedge (\exists\ i < n. ty\ i \in tt)$
{
 fix $s\ t\ tt$
 assume $m: M\ s$ **and** $st: (s,t) \in A^{\widehat{*}}\ O\ ?UnM\ tt\ O\ A^{\widehat{*}}$
 then **obtain** $u\ v$ **where** $su: (s,u) \in A^{\widehat{*}}$ **and** $uv: (u,v) \in ?UnM\ tt$ **and** $vt:$

```

(v,t) ∈ A∗
  by auto
  from A-steps[OF m su] obtain g1 n1 ty1 where seq1: ?seq s u g1 n1 ty1 by
  auto
  from uv have M v unfolding M' by auto
  from A-steps[OF this vt] obtain g2 n2 ty2 where seq2: ?seq v t g2 n2 ty2 by
  auto
  from seq1 have M u by auto
  from one-step[OF this uv] obtain ty where ty: ty ∈ tt and uv: (u,v) ∈ ?rel
  ty by auto
  let ?g = λ i. if i ≤ n1 then g1 i else g2 (i - (Suc n1))
  let ?ty = λ i. if i < n1 then ty1 i else if i = n1 then ty else ty2 (i - (Suc n1))
  let ?n = Suc (n1 + n2)
  have ex: ∃ i < ?n. ?ty i ∈ tt
    by (rule exI[of - n1], simp add: ty)
  have steps: ∀ i < ?n. (?g i, ?g (Suc i)) ∈ ?rel (?ty i)
  proof (intro allI impI)
    fix i
    assume i < ?n
    show (?g i, ?g (Suc i)) ∈ ?rel (?ty i)
    proof (cases i ≤ n1)
      case True
        with seq1 seq2 uv show ?thesis by auto
      next
      case False
        hence i = Suc n1 + (i - Suc n1) by auto
        then obtain k where i: i = Suc n1 + k by auto
        with ⟨i < ?n⟩ have k < n2 by auto
        thus ?thesis using seq2 unfolding i by auto
    qed
  qed
  from steps seq1 seq2 ex
  have seq: ?seqtt s t tt ?g ?n ?ty by auto
  have ∃ g n ty. ?seqtt s t tt g n ty
    by (intro exI, rule seq)
} note A-tt-A = this
let ?tycon = λ ty1 ty2 tt ty' n. ty1 = ty2 → (∃ i < n. ty' i ∈ tt)
let ?seqt = λ i ty g n ty'. f i = g 0 ∧ f (Suc i) = g n ∧ (∀ j < n. (g j, g (Suc
j)) ∈ ?rel (ty' j)) ∧ (∀ j ≤ n. M (g j))
  ∧ (?tycon (ty i) top-s {top-s} ty' n)
  ∧ (?tycon (ty i) top-ns {top-s,top-ns} ty' n)
  ∧ (?tycon (ty i) normal-s {top-s,normal-s} ty' n)
{
  fix i
  have ∃ g n ty'. ?seqt i ty g n ty'
  proof (cases ty i)
    case top-s
      from steps[of i, unfolded top-s]
      have (f i, f (Suc i)) ∈ ?P by auto

```



```

    from A-tt-A[OF min this[unfolded P]]
    show ?thesis unfolding top-s by auto
  next
    case top-ns
    from steps[of i, unfolded top-ns]
    have (f i, f (Suc i)) ∈ ?Pw by auto
    from A-tt-A[OF min this[unfolded Pw]]
    show ?thesis unfolding top-ns by auto
  next
    case normal-s
    from steps[of i, unfolded normal-s]
    have (f i, f (Suc i)) ∈ ?R by auto
    from A-tt-A[OF min this[unfolded R]]
    show ?thesis unfolding normal-s by auto
  next
    case normal-ns
    from steps[of i, unfolded normal-ns]
    have (f i, f (Suc i)) ∈ ?Rw by auto
    from A-steps[OF min this]
    show ?thesis unfolding normal-ns by auto
qed
}
hence ∃ i. ∃ g n ty'. ?seqt i ty g n ty' by auto
from choice[OF this] obtain g where ∃ i. ∃ n ty'. ?seqt i ty (g i) n ty' by auto
from choice[OF this] obtain n where ∃ i. ∃ ty'. ?seqt i ty (g i) (n i) ty' by
auto
from choice[OF this] obtain ty' where ∃ i. ?seqt i ty (g i) (n i) (ty' i) by auto
hence partial: ∧ i. ?seqt i ty (g i) (n i) (ty' i) ..

let ?ind = inf-concat n
let ?g = λ k. (λ (i,j). g i j) (?ind k)
let ?ty = λ k. (λ (i,j). ty' i j) (?ind k)
have inf: INFM i. 0 < n i
  unfolding INFM-nat-le
proof (intro allI)
  fix m
  from inf1[unfolded INFM-nat-le]
  obtain k where k: k ≥ m and ty: ty k ∈ {top-s, top-ns} by auto
  show ∃ k ≥ m. 0 < n k
  proof (intro exI conjI, rule k)
    from partial[of k] ty show 0 < n k by (cases n k, auto)
  qed
qed
note bounds = inf-concat-bounds[OF inf]
note inf-Suc = inf-concat-Suc[OF inf]
note inf-mono = inf-concat-mono[OF inf]
have ¬ SN-rel-ext P Pw R Rw M
  unfolding SN-rel-ext-def simp-thms
proof (rule exI[of - ?g], rule exI[of - ?ty], intro conjI allI)

```

```

fix k
obtain i j where ik: ?ind k = (i,j) by force
from bounds[OF this] have j: j < n i by auto
show M (?g k) unfolding ik using partial[of i] j by auto
next
fix k
obtain i j where ik: ?ind k = (i,j) by force
from bounds[OF this] have j: j < n i by auto
from partial[of i] j have step: (g i j, g i (Suc j)) ∈ ?rel (ty' i j) by auto
obtain i' j' where isk: ?ind (Suc k) = (i',j') by force
have i'j': g i' j' = g i (Suc j)
proof (rule inf-Suc[OF - ik isk])
  fix i
  from partial[of i]
  have g i (n i) = f (Suc i) by simp
  also have ... = g (Suc i) 0 using partial[of Suc i] by simp
  finally show g i (n i) = g (Suc i) 0 .
qed
show (?g k, ?g (Suc k)) ∈ ?rel (?ty k)
  unfolding ik isk split i'j'
  by (rule step)
next
show INFM i. ?ty i ∈ {top-s, top-ns}
  unfolding INFM-nat-le
proof (intro allI)
  fix k
  obtain i j where ik: ?ind k = (i,j) by force
  from inf1[unfolded INFM-nat] obtain i': i' > i and ty: ty i' ∈
{top-s, top-ns} by auto
  from partial[of i'] ty obtain j' where j': j' < n i' and ty': ty' i' j' ∈ {top-s,
top-ns} by auto
  from inf-concat-surj[of - n, OF j'] obtain k' where ik': ?ind k' = (i',j') ..

  from inf-mono[OF ik ik' i'] have k: k ≤ k' by simp
  show ∃ k' ≥ k. ?ty k' ∈ {top-s, top-ns}
    by (intro exI conjI, rule k, unfold ik' split, rule ty')
qed
next
show INFM i. ?ty i ∈ {top-s, normal-s}
  unfolding INFM-nat-le
proof (intro allI)
  fix k
  obtain i j where ik: ?ind k = (i,j) by force
  from inf2[unfolded INFM-nat] obtain i': i' > i and ty: ty i' ∈
{top-s, normal-s} by auto
  from partial[of i'] ty obtain j' where j': j' < n i' and ty': ty' i' j' ∈ {top-s,
normal-s} by auto
  from inf-concat-surj[of - n, OF j'] obtain k' where ik': ?ind k' = (i',j') ..
  from inf-mono[OF ik ik' i'] have k: k ≤ k' by simp

```

```

    show  $\exists k' \geq k. ?ty\ k' \in \{top-s, normal-s\}$ 
      by (intro exI conjI, rule k, unfold ik' split, rule ty')
    qed
  qed
  with assms show False by auto
  qed

```

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

```

proof –

```

  note SN = SN-rel-ext-trans[OF SN]
  let ?P = (A  $\hat{*} O (P' \cap Ms) O A \hat{*}$ )
  let ?Pw = (A  $\hat{*} O ((P' \cup Pw') \cap Ms) O A \hat{*}$ )
  let ?R = (A  $\hat{*} O ((P' \cup R') \cap Ms) O A \hat{*}$ )
  let ?Rw = A  $\hat{*}$ 
  let ?relt = SN-rel-ext-step ?P ?Pw ?R ?Rw
  let ?rel = SN-rel-ext-step P Pw R Rw
  show ?thesis
  proof (rule ccontr)
    assume  $\neg ?thesis$ 
    from this[unfolded SN-rel-ext-def]
    obtain g ty
      where steps:  $\bigwedge i. (g\ i, g\ (Suc\ i)) \in ?rel\ (ty\ i)$ 
      and min:  $\bigwedge i. M\ (g\ i)$ 
      and inf1: INFM i. ty i  $\in \{top-s, top-ns\}$ 
      and inf2: INFM i. ty i  $\in \{top-s, normal-s\}$ 
      by auto
    from inf1[unfolded INFM-nat] obtain k where k: ty k  $\in \{top-s, top-ns\}$  by

```

auto

```

  let ?k = Suc k
  let ?i = shift id ?k
  let ?f =  $\lambda i. f\ (shift\ g\ ?k\ i)$ 
  let ?ty = shift ty ?k
  {
    fix i
    assume ty: ty i  $\in \{top-s, top-ns\}$ 
    note m = min[of i]

```

```

note  $ms = \min[of\ Suc\ i]$ 
from  $P[OF\ m\ ms]$ 
   $Pw[OF\ m\ ms]$ 
   $steps[of\ i]$ 
   $ty$ 
have  $(f\ (g\ i), f\ (g\ (Suc\ i))) \in ?relt\ (ty\ i) \wedge I\ (g\ (Suc\ i))$ 
  by  $(cases\ ty\ i,\ auto)$ 
} note  $stepsP = this$ 
{
  fix  $i$ 
  assume  $I: I\ (g\ i)$ 
  note  $m = \min[of\ i]$ 
  note  $ms = \min[of\ Suc\ i]$ 
  from  $P[OF\ m\ ms]$ 
     $Pw[OF\ m\ ms]$ 
     $R[OF\ I\ m\ ms]$ 
     $Rw[OF\ I\ m\ ms]$ 
     $steps[of\ i]$ 
  have  $(f\ (g\ i), f\ (g\ (Suc\ i))) \in ?relt\ (ty\ i) \wedge I\ (g\ (Suc\ i))$ 
  by  $(cases\ ty\ i,\ auto)$ 
} note  $stepsI = this$ 
{
  fix  $i$ 
  have  $I\ (g\ (?i\ i))$ 
  proof  $(induct\ i)$ 
    case  $0$ 
    show  $?case$  using  $stepsP[OF\ k]$  by  $simp$ 
  next
    case  $(Suc\ i)$ 
    from  $stepsI[OF\ Suc]$  show  $?case$  by  $simp$ 
  qed
} note  $I = this$ 
have  $\neg SN\text{-rel-ext}\ ?P\ ?Pw\ ?R\ ?Rw\ M'$ 
  unfolding  $SN\text{-rel-ext-def}\ simp\ thms$ 
proof  $(rule\ exI[of\ -\ ?f], rule\ exI[of\ -\ ?ty], intro\ allI\ conjI)$ 
  fix  $i$ 
  show  $(?f\ i, ?f\ (Suc\ i)) \in ?relt\ (?ty\ i)$ 
  using  $stepsI[OF\ I[of\ i]]$  by  $auto$ 
next
  show  $INFM\ i.\ ?ty\ i \in \{top\text{-}s, top\text{-}ns\}$ 
  unfolding  $Infm\text{-shift}[of\ \lambda i. i \in \{top\text{-}s, top\text{-}ns\}\ ty\ ?k]$ 
  by  $(rule\ inf1)$ 
next
  show  $INFM\ i.\ ?ty\ i \in \{top\text{-}s, normal\text{-}s\}$ 
  unfolding  $Infm\text{-shift}[of\ \lambda i. i \in \{top\text{-}s, normal\text{-}s\}\ ty\ ?k]$ 
  by  $(rule\ inf2)$ 
next
  fix  $i$ 
  have  $A: A \subseteq Ms$  unfolding  $A$  by  $auto$ 

```

```

from rtrancl-mono[OF this] have As:  $A^{\hat{*}} \subseteq Ms^{\hat{*}}$  by auto
have PM:  $?P \subseteq Ms^{\hat{*}} \circ Ms \circ Ms^{\hat{*}}$  using As by auto
have PwM:  $?Pw \subseteq Ms^{\hat{*}} \circ Ms \circ Ms^{\hat{*}}$  using As by auto
have RM:  $?R \subseteq Ms^{\hat{*}} \circ Ms \circ Ms^{\hat{*}}$  using As by auto
have RwM:  $?Rw \subseteq Ms^{\hat{*}}$  using As by auto
from PM PwM RM have  $?P \cup ?Pw \cup ?R \subseteq Ms^{\hat{*}} \circ Ms \circ Ms^{\hat{*}}$  (is ?PPR
 $\subseteq$  -) by auto
also have  $\dots \subseteq Ms^{\hat{+}}$  by regex
also have  $\dots = Ms$ 
proof
  have  $Ms^{\hat{+}} \subseteq Ms^{\hat{*}} \circ Ms$  by regex
  also have  $\dots \subseteq Ms$  unfolding Ms by auto
  finally show  $Ms^{\hat{+}} \subseteq Ms$  .
qed regex
finally have PPR:  $?PPR \subseteq Ms$  .
show  $M' (?f i)$ 
proof (induct i)
  case 0
  from stepsP[OF k] k
  have  $(f (g k), f (g (Suc k))) \in ?PPR$  by (cases ty k, auto)
  with PPR show ?case unfolding Ms by simp blast
next
  case (Suc i)
  show ?case
  proof (cases ?ty i = normal-ns)
    case False
    hence  $?ty i \in \{top-s, top-ns, normal-s\}$ 
    by (cases ?ty i, auto)
    with stepsI[OF I[of i]] have  $(?f i, ?f (Suc i)) \in ?PPR$ 
    by auto
    from subsetD[OF PPR this] have  $(?f i, ?f (Suc i)) \in Ms$  .
    thus ?thesis unfolding Ms by auto
  next
  case True
  with stepsI[OF I[of i]] have  $(?f i, ?f (Suc i)) \in ?Rw$  by auto
  with RwM have mem:  $(?f i, ?f (Suc i)) \in Ms^{\hat{*}}$  by auto
  thus ?thesis
  proof (cases)
    case base
    with Suc show ?thesis by simp
  next
  case step
  thus ?thesis unfolding Ms by simp
  qed
qed
qed
with SN
show False unfolding A Ms by simp

```

qed
qed

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

proof –

let $?Ms = \{(s,t). M' t\}$
let $?A = (P' \cup Pw' \cup R' \cup Rw') \cap ?Ms$
{
 fix $s t$
 assume $s: M' s$ **and** $(s,t) \in A$
 with $M'[OF\ s, of\ t]$ **have** $(s,t) \in ?A \wedge M' t$ **unfolding** $Ms\ A$ **by** *auto*
} **note** $Aone = this$
{
 fix $s t$
 assume $s: M' s$ **and** $steps: (s,t) \in A \hat{*}$
 from $steps$ **have** $(s,t) \in ?A \hat{*} \wedge M' t$
 proof (*induct*)
 case *base* **from** s **show** $?case$ **by** *simp*
 next
 case (*step* $t\ u$)
 note $one = Aone[OF\ step(3)[THEN\ conjunct2]\ step(2)]$
 from $step(3)$ one
 have $steps: (s,u) \in ?A \hat{*} O ?A$ **by** *blast*
 have $(s,u) \in ?A \hat{*}$
 by (*rule* $subsetD[OF - steps],\ regexp$)
 with one **show** $?case$ **by** *simp*
 qed
} **note** $Amany = this$
let $?P = (A \hat{*} O (P' \cap Ms) O A \hat{*})$
let $?Pw = (A \hat{*} O (P' \cap Ms \cup Pw' \cap Ms) O A \hat{*})$
let $?R = (A \hat{*} O (P' \cap Ms \cup R') O A \hat{*})$
let $?Rw = A \hat{*}$
let $?P' = (?A \hat{*} O (P' \cap ?Ms) O ?A \hat{*})$

```

let ?Pw' = (?A^* O ((P' ∪ Pw') ∩ ?Ms) O ?A^*)
let ?R' = (?A^* O ((P' ∪ R') ∩ ?Ms) O ?A^*)
let ?Rw' = ?A^*
show ?thesis
proof (rule SN-rel-ext-map[OF SN])
  fix s t
  assume s: M s and t: M t and step: (s,t) ∈ P
  from P[OF s t M[OF s] M[OF t] step]
  have (f s, f t) ∈ ?P and I: I t by auto
  then obtain u v where su: (f s, u) ∈ A^* and uv: (u,v) ∈ P' ∩ Ms
    and vt: (v,f t) ∈ A^* by auto
  from Amany[OF M[OF s] su] have su: (f s, u) ∈ ?A^* and u: M' u by auto
  from uv have v: M' v unfolding Ms by auto
  from Amany[OF v vt] have vt: (v, f t) ∈ ?A^* by auto
  from su uv vt I
  show (f s, f t) ∈ ?P' ∧ I t unfolding Ms by auto
next
fix s t
assume s: M s and t: M t and step: (s,t) ∈ Pw
from Pw[OF s t M[OF s] M[OF t] step]
have (f s, f t) ∈ ?Pw and I: I t by auto
then obtain u v where su: (f s, u) ∈ A^* and uv: (u,v) ∈ P' ∩ Ms ∪ Pw' ∩ Ms
  and vt: (v,f t) ∈ A^* by auto
from Amany[OF M[OF s] su] have su: (f s, u) ∈ ?A^* and u: M' u by auto
from uv have uv: (u,v) ∈ (P' ∪ Pw') ∩ ?Ms and v: M' v unfolding Ms
  by auto
from Amany[OF v vt] have vt: (v, f t) ∈ ?A^* by auto
from su uv vt I
show (f s, f t) ∈ ?Pw' ∧ I t by auto
next
fix s t
assume I: I s and s: M s and t: M t and step: (s,t) ∈ R
from R[OF I s t M[OF s] M[OF t] step]
have (f s, f t) ∈ ?R and I: I t by auto
then obtain u v where su: (f s, u) ∈ A^* and uv: (u,v) ∈ P' ∩ Ms ∪ R'
  and vt: (v,f t) ∈ A^* by auto
from Amany[OF M[OF s] su] have su: (f s, u) ∈ ?A^* and u: M' u by auto
from uv M'[OF u, of v] have uv: (u,v) ∈ (P' ∪ R') ∩ ?Ms and v: M' v
unfolding Ms
  by auto
from Amany[OF v vt] have vt: (v, f t) ∈ ?A^* by auto
from su uv vt I
show (f s, f t) ∈ ?R' ∧ I t by auto
next
fix s t
assume I: I s and s: M s and t: M t and step: (s,t) ∈ Rw
from Rw[OF I s t M[OF s] M[OF t] step]
have steps: (f s, f t) ∈ ?Rw and I: I t by auto

```

from $Amany[OF\ M[OF\ s]\ steps]\ I$
show $(f\ s, f\ t) \in ?Rw' \wedge I\ t$ **by** *auto*
qed
qed

lemma $SN\ relto\ imp\ SN\ rel: SN\ (relto\ R\ S) \implies SN\ rel\ R\ S$

proof –

assume $SN: SN\ (relto\ R\ S)$

show *?thesis*

proof (*simp only: SN-rel-on-conv SN-rel-defs, intro allI impI*)

fix f

presume $steps: chain\ (R \cup S)\ f$

obtain r **where** $r: \bigwedge j. r\ j \equiv (f\ j, f\ (Suc\ j)) \in R$ **by** *auto*

show $\neg (INFM\ j. (f\ j, f\ (Suc\ j)) \in R)$

proof (*rule ccontr*)

assume $\neg ?thesis$

hence $ih: infinitely\ many\ r$ **unfolding** *infinitely-many-def r INFM-nat-le* **by** *blast*

obtain $r\ index$ **where** $r\ index = infinitely\ many\ index\ r$ **by** *simp*

with *infinitely-many.index-p[OF ih] infinitely-many.index-ordered[OF ih] infinitely-many.index-not-p-between[OF ih]*

have $r\ index: \bigwedge i. r\ (r\ index\ i) \wedge r\ index\ i < r\ index\ (Suc\ i) \wedge (\forall j. r\ index\ i < j \wedge j < r\ index\ (Suc\ i) \longrightarrow \neg r\ j)$ **by** *auto*

obtain g **where** $g: \bigwedge i. g\ i \equiv f\ (r\ index\ i) ..$

{

fix i

let $?ri = r\ index\ i$

let $?rsi = r\ index\ (Suc\ i)$

from $r\ index$ **have** $isi: ?ri < ?rsi$ **by** *auto*

obtain $ri\ rsi$ **where** $ri: ri = ?ri$ **and** $rsi: rsi = ?rsi$ **by** *auto*

with $r\ index[of\ i]\ steps$ **have** $inter: \bigwedge j. ri < j \wedge j < rsi \implies (f\ j, f\ (Suc\ j)) \in S$ **unfolding** r **by** *auto*

from $ri\ isi\ rsi$ **have** $risi: ri < rsi$ **by** *simp*

{

fix n

assume $Suc\ n \leq rsi - ri$

hence $(f\ (Suc\ ri), f\ (Suc\ (n + ri))) \in S^*$

proof (*induct n, simp*)

case $(Suc\ n)$

hence $steps: (f\ (Suc\ ri), f\ (Suc\ (n+ri))) \in S^*$ **by** *simp*

have $(f\ (Suc\ (n+ri)), f\ (Suc\ (Suc\ n + ri))) \in S$

using $inter[of\ Suc\ n + ri]\ Suc(2)$ **by** *auto*

with $steps$ **show** *?case* **by** *simp*

qed

}

from $this[of\ rsi - ri - 1]\ risi$ **have**

$(f\ (Suc\ ri), f\ rsi) \in S^*$ **by** *simp*

with $ri\ rsi$ **have** $ssteps: (f\ (Suc\ ?ri), f\ ?rsi) \in S^*$ **by** *simp*

with $r\text{-index}[of\ i]$ have $(f\ ?ri, f\ ?rsi) \in R\ O\ S^*$ unfolding r by *auto*
 hence $(g\ i, g\ (Suc\ i)) \in S^* O R O S^*$ using *rtrancl-refl* unfolding g by
auto
 }
 hence $\neg SN\ (S^* O R O S^*)$ unfolding *SN-defs* by *blast*
 with *SN* show *False* by *simp*
 qed
 qed *simp*
 qed

lemma *rtrancl-list-conv*:

$((s, t) \in R^*) =$
 $(\exists\ list.\ last\ (s\ \#\ list) = t \wedge (\forall\ i.\ i < length\ list \longrightarrow ((s\ \#\ list)\ i, (s\ \#\ list)\ i) \in R))$ (is $?l = ?r$)

proof

assume $?r$

then obtain *list* where $last\ (s\ \#\ list) = t \wedge (\forall\ i.\ i < length\ list \longrightarrow ((s\ \#\ list)\ i, (s\ \#\ list)\ i) \in R)$..

thus $?l$

proof (*induct list arbitrary: s, simp*)

case (*Cons u ll*)

hence $last\ (u\ \#\ ll) = t \wedge (\forall\ i.\ i < length\ ll \longrightarrow ((u\ \#\ ll)\ i, (u\ \#\ ll)\ i) \in R)$ by *auto*

from *Cons(1)[OF this]* have $rec: (u, t) \in R^*$.

from *Cons* have $(s, u) \in R$ by *auto*

with *rec* show $?case$ by *auto*

qed

next

assume $?l$

from *rtrancl-imp-seq[OF this]*

obtain $S\ n$ where $s: S\ 0 = s$ and $t: S\ n = t$ and *steps*: $\forall\ i < n.\ (S\ i, S\ (Suc\ i)) \in R$ by *auto*

let $?list = map\ (\lambda\ i.\ S\ (Suc\ i))\ [0 ..< n]$

show $?r$

proof (*rule exI[of - ?list], intro conjI,*

cases n, simp add: s[symmetric] t[symmetric], simp add: t[symmetric])

show $\forall\ i < length\ ?list.\ ((s\ \#\ ?list)\ i, (s\ \#\ ?list)\ i) \in R$

proof (*intro allI impI*)

fix i

assume $i: i < length\ ?list$

thus $((s\ \#\ ?list)\ i, (s\ \#\ ?list)\ i) \in R$

proof (*cases i, simp add: s[symmetric] steps*)

case (*Suc j*)

with i steps show $?thesis$ by *simp*

qed

qed

qed

qed

```

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-rel-imp-SN-relto : SN-rel R S \implies SN (relto R S)

proof –

assume SN: SN-rel R S

show SN (relto R S)

proof

fix f

assume $\forall i. (f i, f (Suc i)) \in \text{relto } R S$

hence steps: $\bigwedge i. (f i, f (Suc i)) \in S^{\widehat{*}} O R O S^{\widehat{*}}$ **by** auto

let ?prop = $\lambda i ai bi. (f i, bi) \in S^{\widehat{*}} \wedge (bi, ai) \in R \wedge (ai, f (Suc i)) \in S^{\widehat{*}}$

{

fix i

from steps **obtain** bi ai **where** ?prop i ai bi **by** blast

hence $\exists ai bi. ?prop i ai bi$ **by** blast

}

hence $\forall i. \exists bi ai. ?prop i ai bi$ **by** blast

from choice[OF this] **obtain** b **where** $\forall i. \exists ai. ?prop i ai (b i)$ **by** blast

from choice[OF this] **obtain** a **where** steps: $\bigwedge i. ?prop i (a i) (b i)$ **by** blast

let ?prop = $\lambda i li. (b i, a i) \in R \wedge (\forall j < \text{length } li. ((a i \# li) ! j, (a i \# li) ! Suc j) \in S) \wedge \text{last } (a i \# li) = b (Suc i)$

{

fix i

from steps[of i] steps[of Suc i] **have** $(a i, f (Suc i)) \in S^{\widehat{*}}$ **and** $(f (Suc i), b (Suc i)) \in S^{\widehat{*}}$ **by** auto

from rtrancl-trans[OF this] steps[of i] **have** R: $(b i, a i) \in R$ **and** S: $(a i, b (Suc i)) \in S^{\widehat{*}}$ **by** blast+

from S[unfolded rtrancl-list-conv] **obtain** li **where** $\text{last } (a i \# li) = b (Suc i) \wedge (\forall j < \text{length } li. ((a i \# li) ! j, (a i \# li) ! Suc j) \in S)$..

with R **have** ?prop i li **by** blast

hence $\exists li. ?prop i li$..

}

hence $\forall i. \exists li. ?prop i li$..

from choice[OF this] **obtain** l **where** steps: $\bigwedge i. ?prop i (l i)$ **by** auto

let ?p = $\lambda i. ?prop i (l i)$

from steps **have** steps: $\bigwedge i. ?p i$ **by** blast

let ?l = $\lambda i. a i \# l i$

let ?g = $\lambda i. \text{choice } (\lambda j. ?l j) i$

obtain g **where** g: $\bigwedge i. g i = (\text{let } (ii, jj) = ?g i \text{ in } ?l ii ! jj)$ **by** auto

have len: $\bigwedge i j n. ?g n = (i, j) \implies j < \text{length } (?l i)$

proof –

fix i j n

assume n: ?g n = (i, j)

```

show  $j < \text{length } (?l\ i)$ 
proof (cases  $n$ )
  case 0
  with  $n$  have  $j = 0$  by auto
  thus ?thesis by simp
next
case (Suc  $nn$ )
obtain  $ii\ jj$  where  $nn: ?g\ nn = (ii, jj)$  by (cases ? $g\ nn$ , auto)
show ?thesis
proof (cases  $\text{Suc } jj < \text{length } (?l\ ii)$ )
  case True
  with  $nn\ \text{Suc}$  have  $?g\ n = (ii, \text{Suc } jj)$  by auto
  with  $n\ \text{True}$  show ?thesis by simp
next
case False
  with  $nn\ \text{Suc}$  have  $?g\ n = (\text{Suc } ii, 0)$  by auto
  with  $n$  show ?thesis by simp
qed
qed
qed
have  $gsteps: \bigwedge i. (g\ i, g\ (\text{Suc } i)) \in R \cup S$ 
proof -
  fix  $n$ 
  obtain  $i\ j$  where  $n: ?g\ n = (i, j)$  by (cases ? $g\ n$ , auto)
  show  $(g\ n, g\ (\text{Suc } n)) \in R \cup S$ 
  proof (cases  $\text{Suc } j < \text{length } (?l\ i)$ )
    case True
    with  $n$  have  $?g\ (\text{Suc } n) = (i, \text{Suc } j)$  by auto
    with  $n$  have  $gn: g\ n = ?l\ i\ !\ j$  and  $gsn: g\ (\text{Suc } n) = ?l\ i\ !\ (\text{Suc } j)$  unfolding
  g by auto
    thus ?thesis using steps[of  $i$ ] True by auto
  next
  case False
  with  $n$  have  $?g\ (\text{Suc } n) = (\text{Suc } i, 0)$  by auto
  with  $n$  have  $gn: g\ n = ?l\ i\ !\ j$  and  $gsn: g\ (\text{Suc } n) = a\ (\text{Suc } i)$  unfolding
  g by auto
  from  $gn\ \text{len}[OF\ n]\ \text{False}$  have  $j = \text{length } (?l\ i) - 1$  by auto
  with  $gn$  have  $gn: g\ n = \text{last } (?l\ i)$  using last-conv-nth[of ? $l\ i$ ] by auto
  from  $gn\ gsn$  show ?thesis using steps[of  $i$ ] steps[of  $\text{Suc } i$ ] by auto
qed
qed
have  $\text{inf}R: \forall n. \exists j \geq n. (g\ j, g\ (\text{Suc } j)) \in R$ 
proof
  fix  $n$ 
  obtain  $i\ j$  where  $n: ?g\ n = (i, j)$  by (cases ? $g\ n$ , auto)
  from  $\text{len}[OF\ n]$  have  $j: j \leq \text{length } (?l\ i) - 1$  by simp
  let ? $k = \text{length } (?l\ i) - 1 - j$ 
  obtain  $k$  where  $k: k = j + ?k$  by auto
  from  $j\ k$  have  $k2: k = \text{length } (?l\ i) - 1$  and  $k3: j + ?k < \text{length } (?l\ i)$  by

```

```

auto
{
  fix n i j k l
  assume n: choice l n = (i,j) and j + k < length (l i)
  hence choice l (n + k) = (i, j + k)
    by (induct k arbitrary: j, simp, auto)
}
from this[OF n, of ?k, OF k3]
have gnk: ?g (n + ?k) = (i, k) by (simp only: k)
hence g (n + ?k) = ?l i ! k unfolding g by auto
hence gnk2: g (n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
from k2 gnk have ?g (Suc (n+?k)) = (Suc i, 0) by auto
hence gnsk2: g (Suc (n+?k)) = a (Suc i) unfolding g by auto
from steps[of i] steps[of Suc i] have main: (g (n+?k), g (Suc (n+?k))) ∈ R
  by (simp only: gnk2 gnsk2)
show ∃ j ≥ n. (g j, g (Suc j)) ∈ R
  by (rule exI[of - n + ?k], auto simp: main[simplified])
qed
from SN[simplified SN-rel-on-conv SN-rel-defs] gsteps infR show False
  unfolding INFM-nat-le by fast
qed
qed

hide-const choice

lemma SN-relto-SN-rel-conv: SN (relto R S) = SN-rel R S
  by (blast intro: SN-relto-imp-SN-rel SN-rel-imp-SN-relto)

lemma SN-rel-empty1: SN-rel {} S
  unfolding SN-rel-defs by auto

lemma SN-rel-empty2: SN-rel R {} = SN R
  unfolding SN-rel-defs SN-defs by auto

lemma SN-relto-mono:
  assumes R: R ⊆ R' and S: S ⊆ S'
  and SN: SN (relto R' S')
  shows SN (relto R S)
  using SN SN-subset[OF - relto-mono[OF R S]] by blast

lemma SN-relto-imp-SN:
  assumes SN (relto R S) shows SN R
proof
  fix f
  assume ∀ i. (f i, f (Suc i)) ∈ R
  hence ∧ i. (f i, f (Suc i)) ∈ relto R S by blast
  thus False using assms unfolding SN-defs by blast
qed

```

lemma *SN-relto-Id*:

$SN (relto R (S \cup Id)) = SN (relto R S)$

by (*simp only: relto-Id*)

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*

proof

fix $f :: nat \Rightarrow 'a$

assume $f 0 \in UNIV$

and *chain: chain R f*

have $*$: $\bigwedge i j. i < j \implies (f i, f j) \in r \cup s$

using *assms* **and** *chain-imp-trancl* [*OF chain*] **by** *auto*

let $?M = \{i. \forall j > i. (f i, f j) \notin r\}$

show *False*

proof (*cases finite ?M*)

let $?n = Max ?M$

assume *finite ?M*

with *Max-ge* **have** $\forall i \in ?M. i \leq ?n$ **by** *simp*

then **have** $\forall k \geq Suc ?n. \exists k' > k. (f k, f k') \in r$ **by** *auto*

with *steps-imp-chainp* [*of Suc ?n* $\lambda x y. (x, y) \in r$] **and** *assms*

show *False* **by** *auto*

next

assume *infinite ?M*

then **have** *INFM j. j* $\in ?M$ **by** (*simp add: Inf-many-def*)

then **interpret** *infinitely-many* $\lambda i. i \in ?M$ **by** (*unfold-locales*) *assumption*

define g **where** [*simp*]: $g = index$

have $\forall i. (f (g i), f (g (Suc i))) \in s$

proof

fix i

have *less*: $g i < g (Suc i)$ **using** *index-ordered-less* [*of i Suc i*] **by** *simp*

have $g i \in ?M$ **using** *index-p* **by** *simp*

then **have** $(f (g i), f (g (Suc i))) \notin r$ **using** *less* **by** *simp*

moreover **have** $(f (g i), f (g (Suc i))) \in r \cup s$ **using** $*$ [*OF less*] **by** *simp*

ultimately **show** $(f (g i), f (g (Suc i))) \in s$ **by** *blast*

qed

with $\langle SN s \rangle$ **show** *False* **by** (*auto simp: SN-defs*)

qed

qed

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*)

proof

assume $SN (r \cup s)$ **thus** $SN r \wedge SN s$

using *SN-subset*[*of ?r*] **by** *blast*

next

assume $SN\ r \wedge SN\ s$
with $trans\text{-}subset\text{-}SN[OF\ assms\ subset\text{-}refl]$ **show** $SN\ ?r$ **by** $simp$
qed

lemma $SN\text{-}relto\text{-}Un$:
 $SN\ (relto\ (R \cup S)\ Q) \longleftrightarrow SN\ (relto\ R\ (S \cup Q)) \wedge SN\ (relto\ S\ Q)$
(is $SN\ ?a \longleftrightarrow SN\ ?b \wedge SN\ ?c$ **)**

proof –
have $eq: ?a^+ = ?b^+ \cup ?c^+$ **by** $regexp$
from $SN\text{-}Un\text{-}conv[of\ ?b^+ ?c^+,\ unfolded\ eq[symmetric]]$
show $?thesis$ **unfolding** $SN\text{-}trancl\text{-}SN\text{-}conv$ **by** $simp$
qed

lemma $SN\text{-}relto\text{-}split$:
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** $SN\ ?c$ **)**

proof –
have $?c^+ \subseteq ?a^+ \cup ?b^+$ **by** $regexp$
from $trans\text{-}subset\text{-}SN[OF\ \text{-}\ this,\ unfolded\ SN\text{-}trancl\text{-}SN\text{-}conv,\ OF\ \text{-}\ assms]$
show $?thesis$ **by** $simp$
qed

lemma $relto\text{-}trancl\text{-}subset$: **assumes** $a \subseteq c$ **and** $b \subseteq c$ **shows** $relto\ a\ b \subseteq c^+$
proof –
have $relto\ a\ b \subseteq (a \cup b)^+$ **by** $regexp$
also $have \dots \subseteq c^+$
by $(rule\ trancl\text{-}mono\text{-}set,\ insert\ assms,\ auto)$
finally **show** $?thesis$.
qed

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

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

lemma $relto\text{-}funD$: **assumes** $relto\text{-}fun\ A\ B\ n\ as\ sel\ m\ (a,b)$
shows $as\ 0 = a\ as\ m = b$
 $\bigwedge i. i < m \Longrightarrow sel\ i \Longrightarrow (as\ i,\ as\ (Suc\ i)) \in A$
 $\bigwedge i. i < m \Longrightarrow \neg\ sel\ i \Longrightarrow (as\ i,\ as\ (Suc\ i)) \in B$
 $n = card\ \{i . i < m \wedge sel\ i\}$
 $n = 0 \longleftrightarrow m = 0$
using $assms[unfolded\ relto\text{-}fun.\text{simps}]$ **by** $blast+$

lemma $relto\text{-}fun\text{-}refl$: $\exists\ as\ sel. relto\text{-}fun\ A\ B\ 0\ as\ sel\ 0\ (a,a)$

by (rule exI[of - $\lambda \cdot a$], rule exI, rule relto-fun, auto)

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

proof –

from *assms* **obtain** $a' b'$ **where** $aa: (a,a') \in B^*$ **and** $ab: (a',b') \in A$
and $bb: (b',b) \in B^*$ **by** *auto*

from $aa[\text{unfolded } r\text{trancl-fun-conv}]$ **obtain** $f1\ n1$ **where**

$f1: f1\ 0 = a\ f1\ n1 = a' \wedge i. i < n1 \implies (f1\ i, f1\ (\text{Suc } i)) \in B$ **by** *auto*

from $bb[\text{unfolded } r\text{trancl-fun-conv}]$ **obtain** $f2\ n2$ **where**

$f2: f2\ 0 = b'\ f2\ n2 = b \wedge i. i < n2 \implies (f2\ i, f2\ (\text{Suc } i)) \in B$ **by** *auto*

let $?gen = \lambda aa\ ab\ bb\ i. \text{if } i < n1 \text{ then } aa\ i \text{ else if } i = n1 \text{ then } ab \text{ else } bb\ (i - \text{Suc } n1)$

let $?f = ?gen\ f1\ a'\ f2$

let $?sel = ?gen\ (\lambda \cdot \text{False})\ \text{True}\ (\lambda \cdot \text{False})$

let $?m = \text{Suc } (n1 + n2)$

show $?thesis$

proof (rule exI[of - $?f$], rule exI[of - $?sel$], rule exI[of - $?m$], rule relto-fun)

fix i

assume $i: i < ?m$

show $(?sel\ i \longrightarrow (?f\ i, ?f\ (\text{Suc } i)) \in A) \wedge (\neg ?sel\ i \longrightarrow (?f\ i, ?f\ (\text{Suc } i)) \in B)$

proof (cases $i < n1$)

case *True*

with $f1(3)[\text{OF } \text{this}] f1(2)$ **show** $?thesis$ **by** (cases $\text{Suc } i = n1$, *auto*)

next

case *False* **note** $nle = \text{this}$

show $?thesis$

proof (cases $i > n1$)

case *False*

with nle **have** $i = n1$ **by** *auto*

thus $?thesis$ **using** $f1\ f2\ ab$ **by** *auto*

next

case *True*

define j **where** $j = i - \text{Suc } n1$

have $i: i = \text{Suc } n1 + j$ **and** $j: j < n2$ **using** i *True* **unfolding** $j\text{-def}$ **by**

auto

thus $?thesis$ **using** $f2$ **by** *auto*

qed

qed

qed (*insert* $f1\ f2$, *auto*)

qed

lemma relto-fun-trans: **assumes** $ab: \text{relto-fun } A B\ n1\ as1\ sel1\ m1 (a,b)$

and $bc: \text{relto-fun } A B\ n2\ as2\ sel2\ m2 (b,c)$

shows $\exists \text{ as sel. relto-fun } A B (n1 + n2) \text{ as sel } (m1 + m2) (a,c)$

proof –

from $relto-funD[\text{OF } ab]$

have $1: as1\ 0 = a\ as1\ m1 = b$

$\wedge i. i < m1 \implies (sel1\ i \longrightarrow (as1\ i, as1\ (\text{Suc } i)) \in A) \wedge (\neg sel1\ i \longrightarrow (as1\ i,$

```

as1 (Suc i) ∈ B
  n1 = 0 ↔ m1 = 0 and card1: n1 = card {i. i < m1 ∧ sel1 i} by blast+
from relto-funD[OF bc]
  have 2: as2 0 = b as2 m2 = c
    ∧ i. i < m2 ⇒ (sel2 i → (as2 i, as2 (Suc i)) ∈ A) ∧ (¬ sel2 i → (as2 i,
as2 (Suc i)) ∈ B)
  n2 = 0 ↔ m2 = 0 and card2: n2 = card {i. i < m2 ∧ sel2 i} by blast+
let ?as = λ i. if i < m1 then as1 i else as2 (i - m1)
let ?sel = λ i. if i < m1 then sel1 i else sel2 (i - m1)
let ?m = m1 + m2
let ?n = n1 + n2
show ?thesis
proof (rule exI[of - ?as], rule exI[of - ?sel], rule relto-fun)
  have id: {i . i < ?m ∧ ?sel i} = {i . i < m1 ∧ sel1 i} ∪ ((+) m1) ‘ {i. i
< m2 ∧ sel2 i}
    (is - = ?A ∪ ?f ‘ ?B)
    by force
  have card (?A ∪ ?f ‘ ?B) = card ?A + card (?f ‘ ?B)
    by (rule card-Un-disjoint, auto)
  also have card (?f ‘ ?B) = card ?B
    by (rule card-image, auto simp: inj-on-def)
  finally show ?n = card {i . i < ?m ∧ ?sel i} unfolding card1 card2 id by
simp
next
fix i
assume i: i < ?m
show (?sel i → (?as i, ?as (Suc i)) ∈ A) ∧ (¬ ?sel i → (?as i, ?as (Suc i))
∈ B)
proof (cases i < m1)
  case True
    from 1 2 have [simp]: as2 0 = as1 m1 by simp
    from True 1(3)[of i] 1(2) show ?thesis by (cases Suc i = m1, auto)
  next
    case False
      define j where j = i - m1
      have i: i = m1 + j and j: j < m2 using i False unfolding j-def by auto
      thus ?thesis using False 2(3)[of j] by auto
    qed
  qed (insert 1 2, auto)
qed

```

lemma *reltos-into-relto-fun*: **assumes** (*a,b*) ∈ (*relto A B*)^{~n}
shows ∃ *as sel m*. *relto-fun A B n as sel m* (*a,b*)
using *assms*
proof (*induct n arbitrary: b*)
case (0 *b*)
hence *b*: *b* = *a* **by** *auto*
show *?case* **unfolding** *b* **using** *relto-fun-refl*[*of A B a*] **by** *blast*
next

case ($Suc\ n\ c$)
from $relpow\text{-}Suc\text{-}E[OF\ Suc(2)]$
obtain b **where** $ab: (a,b) \in (relto\ A\ B) \sim\sim n$ **and** $bc: (b,c) \in relto\ A\ B$ **by** $auto$
from $Suc(1)[OF\ ab]$ **obtain** $as\ sel\ m$ **where**
 $IH: relto\text{-}fun\ A\ B\ n\ as\ sel\ m\ (a,\ b)$ **by** $auto$
from $relto\text{-}into\text{-}relto\text{-}fun[OF\ bc]$ **obtain** $as\ sel\ m$ **where** $relto\text{-}fun\ A\ B\ (Suc\ 0)$
 $as\ sel\ m\ (b,c)$ **by** $blast$
from $relto\text{-}fun\text{-}trans[OF\ IH\ this]$ **show** $?case$ **by** $auto$
qed

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

proof –

note $*$ = $relto\text{-}funD[OF\ assms]$
 $\{$
 $\ \ \ \mathbf{fix}\ m'$
 $\ \ \ \mathbf{let}\ ?c = \lambda\ m'.\ card\ \{i.\ i < m' \wedge sel\ i\}$
 $\ \ \ \mathbf{assume}\ m' \leq m$
 $\ \ \ \mathbf{hence}\ (?c\ m' > 0 \longrightarrow (as\ 0,\ as\ m') \in (relto\ A\ B) \sim\sim ?c\ m') \wedge (?c\ m' = 0 \longrightarrow$
 $(as\ 0,\ as\ m') \in B \hat{\sim} *)$
 $\ \ \ \mathbf{proof}\ (induct\ m')$
 $\ \ \ \ \ \ \mathbf{case}\ (Suc\ m')$
 $\ \ \ \ \ \ \ \ \ \mathbf{let}\ ?x = as\ 0$
 $\ \ \ \ \ \ \ \ \ \mathbf{let}\ ?y = as\ m'$
 $\ \ \ \ \ \ \ \ \ \mathbf{let}\ ?z = as\ (Suc\ m')$
 $\ \ \ \ \ \ \ \ \ \mathbf{let}\ ?C = ?c\ (Suc\ m')$
 $\ \ \ \ \ \ \ \ \ \mathbf{have}\ C: ?C = ?c\ m' + (if\ (sel\ m')\ then\ 1\ else\ 0)$
 $\ \ \ \ \ \ \ \ \ \mathbf{proof}\ \text{--}$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{have}\ id: \{i.\ i < Suc\ m' \wedge sel\ i\} = \{i.\ i < m' \wedge sel\ i\} \cup (if\ sel\ m'\ then$
 $\{m'\}\ else\ \{\})$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{by}\ (cases\ sel\ m',\ auto,\ case\text{-}tac\ x = m',\ auto)$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{show}\ ?thesis\ \mathbf{unfolding}\ id\ \mathbf{by}\ auto$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{qed}$
 $\ \ \ \ \ \ \ \ \ \mathbf{from}\ Suc(2)\ \mathbf{have}\ m': m' \leq m\ \mathbf{and}\ lt: m' < m\ \mathbf{by}\ auto$
 $\ \ \ \ \ \ \ \ \ \mathbf{from}\ Suc(1)[OF\ m']\ \mathbf{have}\ IH: ?c\ m' > 0 \implies (?x,\ ?y) \in (relto\ A\ B) \sim\sim ?c$
 m'
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \ ?c\ m' = 0 \implies (?x,\ ?y) \in B \hat{\sim} * \mathbf{by}\ auto$
 $\ \ \ \ \ \ \ \ \ \mathbf{from}\ *(3\text{-}4)[OF\ lt]\ \mathbf{have}\ yz: sel\ m' \implies (?y,\ ?z) \in A \neg sel\ m' \implies (?y,\ ?z)$
 $\in B\ \mathbf{by}\ auto$
 $\ \ \ \ \ \ \ \ \ \mathbf{show}\ ?case$
 $\ \ \ \ \ \ \ \ \ \mathbf{proof}\ (cases\ ?c\ m' = 0)$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{case}\ True\ \mathbf{note}\ c = this$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{from}\ IH(2)[OF\ this]\ \mathbf{have}\ xy: (?x,\ ?y) \in B \hat{\sim} * \mathbf{by}\ auto$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{show}\ ?thesis$
 $\ \ \ \ \ \ \ \ \ \mathbf{proof}\ (cases\ sel\ m')$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{case}\ False$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{from}\ xy\ yz(2)[OF\ False]\ \mathbf{have}\ xz: (?x,\ ?z) \in B \hat{\sim} * \mathbf{by}\ auto$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{from}\ False\ c\ \mathbf{have}\ C: ?C = 0\ \mathbf{unfolding}\ C\ \mathbf{by}\ simp$
 $\ \ \ \ \ \ \ \ \ \ \ \ \ \mathbf{from}\ xz\ \mathbf{show}\ ?thesis\ \mathbf{unfolding}\ C\ \mathbf{by}\ auto$

```

next
  case True
  from xy yz(1)[OF True] have xz: (?x,?z) ∈ relto A B by auto
  from True c have C: ?C = 1 unfolding C by simp
  from xz show ?thesis unfolding C by auto
qed
next
case False
hence c: ?c m' > 0 (?c m' = 0) = False by arith+
from IH(1)[OF c(1)] have xy: (?x, ?y) ∈ (relto A B) ~ ?c m' .
show ?thesis
proof (cases sel m')
  case False
  from c obtain k where ck: ?c m' = Suc k by (cases ?c m', auto)
  from relpow-Suc-E[OF xy[unfolded this]] obtain
    u where xu: (?x, u) ∈ (relto A B) ~ k and uy: (u, ?y) ∈ relto A B by
auto
  from uy yz(2)[OF False] have uz: (u, ?z) ∈ relto A B by force
  with xu have xz: (?x,?z) ∈ (relto A B) ~ ?c m' unfolding ck by auto
  from False c have C: ?C = ?c m' unfolding C by simp
  from xz show ?thesis unfolding C c by auto
next
case True
from xy yz(1)[OF True] have xz: (?x,?z) ∈ (relto A B) ~ (Suc (?c m'))
by auto
from c True have C: ?C = Suc (?c m') unfolding C by simp
from xz show ?thesis unfolding C by auto
qed
qed
qed simp
}
from this[of m] * show ?thesis by auto
qed

```

lemma *relto-relto-fun-conv*: $((a,b) \in (\text{relto } A B) \sim^n) = (\exists \text{ as sel } m. \text{relto-fun } A B n \text{ as sel } m (a,b))$
using *relto-fun-into-reltos*[of A B n - - - a b] *reltos-into-relto-fun*[of a b n B A]
by *blast*

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 $i \leq m \implies (a, \text{as } i) \in C^*$

proof (*induct* i)

case 0

from *relto-funD*[OF *rf*] **show** ?*case* **by** *simp*

next

case (*Suc* i)

hence *IH*: $(a, \text{as } i) \in C^*$ **and** *im*: $i < m$ **by** *auto*

from *relto-funD*(3-4)[OF *rf im*] *assms* **have** $(\text{as } i, \text{as } (\text{Suc } i)) \in C$ **by** *auto*

with *IH* show *?case* by *auto*
 qed

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 –
 obtain *v* where $(s, v) \in \text{relto } R \ E$ and $v: \neg \text{SN-on } (\text{relto } R \ E) \ \{v\}$
 using *assms* by *fast*
 moreover then obtain *t* and *u*
 where $(s, t) \in E^*$ and $(t, u) \in R$ and $uv: (u, v) \in E^*$ by *auto*
 moreover from *uv* have $uv: (u, v) \in (R \cup E)^*$ by *regexp*
 moreover have $\neg \text{SN-on } (\text{relto } R \ E) \ \{u\}$ using
 $v \text{ steps-preserve-SN-on-relto} [OF \ uv]$ by *auto*
 ultimately show *?thesis* by *auto*
 qed

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

assume *L*: *?L T*
 { fix *t* assume $t \in T$ hence *?L* {*t*} using *L* by *fast* }
 thus *?R T* by *fast*
 next
 { fix *s*
 have $\text{SN-on } (\text{relto } R \ S) \ \{s\} = \text{SN-on } (S^* \ O \ R) \ \{s\}$
 proof
 let *?X* = $\{s. \neg \text{SN-on } (\text{relto } R \ S) \ \{s\}\}$
 { assume $\neg ?L \ \{s\}$
 hence $s \in ?X$ by *auto*
 hence $\neg ?R \ \{s\}$
 proof(rule *lower-set-imp-not-SN-on, intro ballI*)
 fix *s* assume $s \in ?X$
 then obtain *t u* where $(s, t) \in S^* \ (t, u) \in R$ and $u: u \in ?X$
 unfolding *mem-Collect-eq* by (*metis not-SN-on-rel-succ*)
 hence $(s, u) \in S^* \ O \ R$ by *auto*
 with *u* show $\exists u \in ?X. (s, u) \in S^* \ O \ R$ by *auto*
 qed
 }
 thus *?R* {*s*} $\implies ?L \ \{s\}$ by *auto*
 assume *?L* {*s*} thus *?R* {*s*} by (*rule SN-on-mono, auto*)
 qed
 } note *main* = *this*
 assume *R*: *?R T*
 { fix *t* assume $t \in T$ hence *?L* {*t*} unfolding *main* using *R* by *fast* }
 thus *?L T* by *fast*
 qed

lemma *trans-relto*:

assumes *trans*: $\text{trans } R \text{ and } S \circ R \subseteq R \circ S$
shows *trans* (*relto* $R \ S$)
proof
fix $a \ b \ c$
assume $ab: (a, b) \in S^* \circ R \circ S^*$ **and** $bc: (b, c) \in S^* \circ R \circ S^*$
from *rtrancl-O-push* [*of* $S \ R$] *assms*(2) **have** $\text{comm}: S^* \circ R \subseteq R \circ S^*$ **by** *blast*
from ab **obtain** $d \ e$ **where** $de: (a, d) \in S^* \ (d, e) \in R \ (e, b) \in S^*$ **by** *auto*
from bc **obtain** $f \ g$ **where** $fg: (b, f) \in S^* \ (f, g) \in R \ (g, c) \in S^*$ **by** *auto*
from de (3) fg (1) **have** $(e, f) \in S^*$ **by** *auto*
with fg (2) *comm* **have** $(e, g) \in R \circ S^*$ **by** *blast*
then **obtain** h **where** $h: (e, h) \in R \ (h, g) \in S^*$ **by** *auto*
with de (2) *trans* **have** $dh: (d, h) \in R$ **unfolding** *trans-def* **by** *blast*
from fg (3) h (2) **have** $(h, c) \in S^*$ **by** *auto*
with de (1) dh (1) **show** $(a, c) \in S^* \circ R \circ S^*$ **by** *auto*
qed

lemma *relative-ending*:

assumes *chain*: $\text{chain } (R \cup S) \ t$
and $t0: t \ 0 \in X$
and $SN: SN\text{-on } (\text{relto } R \ S) \ X$
shows $\exists j. \forall i \geq j. (t \ i, t \ (\text{Suc } i)) \in S - R$
proof (*rule ccontr*)
assume $\neg ?thesis$
with *chain* **have** $\forall i. \exists j. j \geq i \wedge (t \ j, t \ (\text{Suc } j)) \in R$ **by** *blast*
from *choice* [*OF this*] **obtain** f **where** *R-steps*: $\forall i. i \leq f \ i \wedge (t \ (f \ i), t \ (\text{Suc } (f \ i))) \in R \ ..$
let $?t = \lambda i. t \ (((\text{Suc } \circ f) \ \sim i) \ 0)$
have $\forall i. (t \ i, t \ (\text{Suc } (f \ i))) \in (\text{relto } R \ S)^+$
proof
fix i
from *R-steps* **have** $\text{leq}: i \leq f \ i$ **and** $\text{step}: (t \ (f \ i), t \ (\text{Suc } (f \ i))) \in R$ **by** *auto*
from *chain-imp-rtrancl* [*OF chain leq*] **have** $(t \ i, t \ (f \ i)) \in (R \cup S)^*$.
with step **have** $(t \ i, t \ (\text{Suc } (f \ i))) \in (R \cup S)^* \circ R$ **by** *auto*
then **show** $(t \ i, t \ (\text{Suc } (f \ i))) \in (\text{relto } R \ S)^+$ **by** *regexp*
qed
then **have** $\text{chain } ((\text{relto } R \ S)^+) \ ?t$ **by** *simp*
with $t0$ **have** $\neg SN\text{-on } ((\text{relto } R \ S)^+) \ X$ **by** (*unfold SN-on-def, auto intro: exI[of - ?t]*)
with *SN-on-trancl*[*OF SN*] **show** *False* **by** *auto*
qed

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

lemma *SN-on-relto-Un*:

assumes *closure*: $\text{relto } (R \cup R') \ S \ \text{`` } X \subseteq X$
shows $SN\text{-on } (\text{relto } (R \cup R') \ S) \ X \longleftrightarrow SN\text{-on } (\text{relto } R \ (R' \cup S)) \ X \wedge SN\text{-on } (\text{relto } R' \ S) \ X$
(is $?c \longleftrightarrow ?a \wedge ?b$
proof(*safe*)
assume $SN: ?a$ **and** $SN': ?b$

```

from SN have SN: SN-on (relto (relto R S) (relto R' S)) X by (rule SN-on-subset1)
regexp
show ?c
proof
  fix f
  assume f0: f 0 ∈ X and chain: chain (relto (R ∪ R') S) f
  then have chain (relto R S ∪ relto R' S) f by auto
  from relative-ending[OF this f0 SN]
  have ∃ j. ∀ i ≥ j. (f i, f (Suc i)) ∈ relto R' S – relto R S by auto
  then obtain j where ∀ i ≥ j. (f i, f (Suc i)) ∈ relto R' S by auto
  then have chain (relto R' S) (shift f j) by auto
  moreover have f j ∈ X
  proof(induct j)
    case 0 from f0 show ?case by simp
  next
    case (Suc j)
    let ?s = (f j, f (Suc j))
    from chain have ?s ∈ relto (R ∪ R') S by auto
    with Image-closed-trancl[OF closure] Suc show f (Suc j) ∈ X by blast
  qed
  then have shift f j 0 ∈ X by auto
  ultimately have ¬ SN-on (relto R' S) X by (intro not-SN-onI)
  with SN' show False by auto
  qed
next
  assume SN: ?c
  then show ?b by (rule SN-on-subset1, auto)
  moreover
  from SN have SN-on ((relto (R ∪ R') S)+) X by (unfold SN-on-trancl-SN-on-conv)
  then show ?a by (rule SN-on-subset1) regexp
qed

lemma SN-on-Un: (R ∪ R') “X ⊆ X ⇒ SN-on (R ∪ R') X ⇔ SN-on (relto R
R') X ∧ SN-on R' X
  using SN-on-relto-Un[of {}] by simp

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-reft:  $x \geq (x :: 'a)$ 
  and ge-trans[trans]:  $\llbracket x \geq y; (y :: 'a) \geq z \rrbracket \implies x \geq z$ 
  and max-comm:  $\max x y = \max y x$ 
  and max-ge-x[intro]:  $\max x y \geq x$ 
  and max-id:  $x \geq y \implies \max x y = x$ 
  and max-mono:  $x \geq y \implies \max z x \geq \max z y$ 
begin
lemma max-ge-y[intro]:  $\max x y \geq y$ 
  unfolding max-comm[of x y] ..

lemma max-mono2:  $x \geq y \implies \max x z \geq \max y z$ 
  unfolding max-comm[of - z] by (rule max-mono)
end

class ordered-ab-semigroup = non-strict-order + ab-semigroup-add + monoid-add
+
  assumes plus-left-mono:  $x \geq y \implies x + z \geq y + z$ 

lemma plus-right-mono:  $y \geq (z :: 'a :: ordered-ab-semigroup) \implies x + y \geq x + z$ 
  by (simp add: add.commute[of x], rule plus-left-mono, auto)

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  $\delta$ 

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]:  $\llbracket x \succ y; y \succ z \rrbracket \implies x \succ z$ 
  by (rule compat[OF gt-imp-ge])
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:  $x \succ y \implies a + x \succ a + y$ 
  unfolding add commute[of a] by (rule plus-gt-left-mono)

lemma plus-gt-both-mono:  $x \succ y \implies a \succ b \implies x + a \succ y + b$ 
  by (rule gt-trans[OF plus-gt-left-mono plus-gt-right-mono])
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:  $\llbracket mono \ x; y \succ z; x \geq 0 \rrbracket \implies x * y \succ x * z$ 

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: arc-pos (1 :: 'a)
  and arc-pos-default: arc-pos default
  and arc-pos-zero:  $\neg$  arc-pos 0
  and arc-pos-plus: arc-pos x  $\implies$  arc-pos (x + y)
  and arc-pos-mult:  $\llbracket arc-pos \ x; arc-pos \ y \rrbracket \implies arc-pos \ (x * y)$ 
  and not-all-ge:  $\bigwedge c \ d. arc-pos \ d \implies \exists e. e \geq 0 \wedge arc-pos \ e \wedge \neg (c \geq d * e)$ 
begin
lemma max0-id:  $max \ 0 \ (x :: 'a) = x$ 
  unfolding max-comm[of 0]
  by (rule max-id[OF zero-leastIII])
end

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

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

```

```

locale weak-SN-both-mono-ordered-semiring-1 =
  fixes weak-gt :: 'a :: ordered-semiring-1  $\Rightarrow$  'a  $\Rightarrow$  bool
  and default :: 'a
  and arc-pos :: 'a  $\Rightarrow$  bool
  assumes weak-gt-both-mono:  $\forall x y. (x,y) \in \text{set } xys \longrightarrow \text{weak-gt } x y \Longrightarrow \exists gt.$ 
  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 :: poly-carrier and gt (infix  $\succ$  50) +
  fixes power-mono :: bool
  and discrete :: bool
  assumes times-gt-mono:  $\llbracket y \succ z; x \geq 1 \rrbracket \Longrightarrow y * x \succ z * x$ 
  and power-mono: power-mono  $\Longrightarrow x \succ y \Longrightarrow y \geq 0 \Longrightarrow n \geq 1 \Longrightarrow x \wedge^n \succ y$ 
   $\wedge^n$ 
  and discrete: discrete  $\Longrightarrow x \geq y \Longrightarrow \exists k. x = (((+ 1) \wedge^k) y)$ 

```

```

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  $a \wedge^n \geq b \wedge^n \wedge b \wedge^n \geq 0$ 
proof (induct n)
  case 0
  show ?case by (auto simp: ge-refl one-ge-zero)
next
  case (Suc n)
  hence abn:  $a \wedge^n \geq b \wedge^n$  and bn:  $b \wedge^n \geq 0$  by auto
  have bsn:  $b \wedge \text{Suc } n \geq 0$  unfolding power-Suc
    using times-left-mono[OF bn b] by auto
  have  $a \wedge \text{Suc } n = a * a \wedge^n$  unfolding power-Suc by simp
  also have  $\dots \geq b * a \wedge^n$ 
    by (rule times-left-mono[OF ge-trans[OF abn bn] ab])
  also have  $b * a \wedge^n \geq b * b \wedge^n$ 
    by (rule times-right-mono[OF b abn])
  finally show ?case using bsn unfolding power-Suc by simp
qed

```

```

lemma pow-ge-zero[intro]: assumes a:  $a \geq (0 :: 'a)$ 
  shows  $a \wedge^n \geq 0$ 
proof (induct n)
  case 0
  from one-ge-zero show ?case by simp
next
  case (Suc n)

```


show ?case **using** times-left-mono[OF Suc a] **by** simp
qed
end

lemma of-nat-ge-zero[*intro,simp*]: of-nat $n \geq (0 :: 'a :: \text{ordered-semiring-1})$
proof (induct n)
 case 0
 show ?case **by** (simp add: ge-refl)
next
 case (Suc n)
 from plus-right-mono[OF Suc, of 1] **have** of-nat (Suc n) $\geq (1 :: 'a)$ **by** simp
 also have $(1 :: 'a) \geq 0$ **using** one-ge-zero .
 finally show ?case .
qed

lemma mult-ge-zero[*intro*]: $(a :: 'a :: \text{ordered-semiring-1}) \geq 0 \implies b \geq 0 \implies a * b \geq 0$
using times-left-mono[of b 0 a] **by** auto

lemma pow-mono-one: **assumes** $a \geq (1 :: 'a :: \text{ordered-semiring-1})$
 shows $a^n \geq 1$
proof (induct n)
 case (Suc n)
 show ?case **unfolding** power-Suc
 using ge-trans[OF times-right-mono[OF ge-trans[OF a one-ge-zero] Suc], of 1]
 a
 by (auto simp: field-simps)
qed (auto simp: ge-refl)

lemma pow-mono-exp: **assumes** $a \geq (1 :: 'a :: \text{ordered-semiring-1})$
 shows $n \geq m \implies a^n \geq a^m$
proof (induct m arbitrary: n)
 case 0
 show ?case **using** pow-mono-one[OF a] **by** auto
next
 case (Suc m nn)
 then obtain n **where** $nn = \text{Suc } n$ **by** (cases nn, auto)
 note $\text{Suc} = \text{Suc}[\text{unfolded } nn]$
 hence $\text{rec}: a^n \geq a^m$ **by** auto
 show ?case **unfolding** nn power-Suc
 by (rule times-right-mono[OF ge-trans[OF a one-ge-zero] rec])
qed

lemma mult-ge-one[*intro*]: **assumes** $a: (a :: 'a :: \text{ordered-semiring-1}) \geq 1$
 and $b: b \geq 1$
 shows $a * b \geq 1$
proof –
 from ge-trans[OF b one-ge-zero] **have** $b0: b \geq 0$.
 from times-left-mono[OF b0 a] **have** $a * b \geq b$ **by** simp

from *ge-trans*[*OF this b*] **show** *?thesis* .
qed

lemma *sum-list-ge-mono*: **fixes** *as* :: ('a :: *ordered-semiring-0*) *list*
assumes *length as = length bs*
and $\bigwedge i. i < \text{length } bs \implies as ! i \geq bs ! i$
shows *sum-list as* \geq *sum-list bs*
using *assms*
proof (*induct as arbitrary: bs*)
case (*Nil bs*)
from *Nil(1)* **show** *?case* **by** (*simp add: ge-refl*)
next
case (*Cons a as bbs*)
from *Cons(2)* **obtain** *b bs* **where** *bbs: bbs = b # bs* **and** *len: length as = length bs*
by (*cases bbs, auto*)
note *ge = Cons(3)[unfolded bbs]*
{
fix *i*
assume *i < length bs*
hence *Suc i < length (b # bs)* **by** *simp*
from *ge[OF this]* **have** *as ! i* \geq *bs ! i* **by** *simp*
}
from *Cons(1)[OF len this]* **have** *IH: sum-list as* \geq *sum-list bs* .
from *ge[of 0]* **have** *ab: a* \geq *b* **by** *simp*
from *ge-trans[OF plus-left-mono[OF ab] plus-right-mono[OF IH]]*
show *?case* **unfolding** *bbs* **by** *simp*
qed

lemma *sum-list-ge-0-nth*: **fixes** *xs* :: ('a :: *ordered-semiring-0*)*list*
assumes *ge: $\bigwedge i. i < \text{length } xs \implies xs ! i \geq 0$*
shows *sum-list xs* ≥ 0
proof –
let *?l = replicate (length xs) (0 :: 'a)*
have *length xs = length ?l* **by** *simp*
from *sum-list-ge-mono[OF this] ge* **have** *sum-list xs* \geq *sum-list ?l* **by** *simp*
also have *sum-list ?l = 0* **using** *sum-list-0[of ?l]* **by** *auto*
finally show *?thesis* .
qed

lemma *sum-list-ge-0*: **fixes** *xs* :: ('a :: *ordered-semiring-0*)*list*
assumes *ge: $\bigwedge x. x \in \text{set } xs \implies x \geq 0$*
shows *sum-list xs* ≥ 0
by (*rule sum-list-ge-0-nth, insert ge[unfolded set-conv-nth], auto*)

lemma *foldr-max*: *a* \in *set as* \implies *foldr max as b* \geq (*a* :: 'a :: *ordered-ab-semigroup*)
proof (*induct as arbitrary: b*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons c as*)

```

show ?case
proof (cases a = c)
  case True
  show ?thesis unfolding True by auto
next
  case False
  with Cons have foldr max as b ≥ a by auto
  from ge-trans[OF - this] show ?thesis by auto
qed

```

```

lemma of-nat-mono[intro]: assumes n ≥ m shows (of-nat n :: 'a :: ordered-semiring-1)
≥ of-nat m
proof -
  let ?n = of-nat :: nat ⇒ 'a
  from assms
  show ?thesis
  proof (induct m arbitrary: n)
    case 0
    show ?case by auto
  next
    case (Suc m nn)
    then obtain n where nn: nn = Suc n by (cases nn, auto)
    note Suc = Suc[unfolded nn]
    hence rec: ?n n ≥ ?n m by simp
    show ?case unfolding nn of-nat-Suc
    by (rule plus-right-mono[OF rec])
  qed
qed

```

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

```

definition non-inf :: 'a rel ⇒ bool
where non-inf r ≡ ∀ a f. ∃ i. (f i, f (Suc i)) ∉ r ∨ (f i, a) ∉ r

```

```

lemma non-infI[intro]: assumes ∧ a f. [ [ ∧ i. (f i, f (Suc i)) ∈ r ] ⇒ ∃ i. (f i,
a) ∉ r
shows non-inf r
using assms unfolding non-inf-def by blast

```

```

lemma non-infE[elim]: assumes non-inf r and ∧ i. (f i, f (Suc i)) ∉ r ∨ (f i,
a) ∉ r ⇒ P
shows P
using assms unfolding non-inf-def by blast

```

```

lemma non-inf-image:
assumes ni: non-inf r and image: ∧ a b. (a,b) ∈ s ⇒ (f a, f b) ∈ r
shows non-inf s
proof
  fix a g

```

```

assume  $s: \bigwedge i. (g\ i, g\ (Suc\ i)) \in s$ 
define  $h$  where  $h = f\ o\ g$ 
from  $image[OF\ s]$  have  $h: \bigwedge i. (h\ i, h\ (Suc\ i)) \in r$  unfolding  $h\text{-def}\ comp\text{-def}$  .
from  $non\text{-infE}[OF\ ni, of\ h]$  have  $\bigwedge a. \exists i. (h\ i, a) \notin r$  using  $h$  by  $blast$ 
thus  $\exists i. (g\ i, a) \notin s$  using  $image$  unfolding  $h\text{-def}\ comp\text{-def}$  by  $blast$ 
qed

```

```

lemma  $SN\text{-imp}\text{-non}\text{-inf}: SN\ r \implies non\text{-inf}\ r$ 
by  $(intro\ non\text{-infI, auto)}$ 

```

```

lemma  $non\text{-inf}\text{-imp}\text{-SN}\text{-bound}: non\text{-inf}\ r \implies SN\ \{(a,b). (b,c) \in r \wedge (a,b) \in r\}$ 
by  $(rule, auto)$ 

```

end

5 Carriers of Strongly Normalizing Orders

```

theory  $SN\text{-Order}\text{-Carrier}$ 
imports
   $SN\text{-Orders}$ 
   $HOL.Rat$ 
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  $nat :: large\text{-ordered}\text{-semiring}\text{-1}$ 
begin
instance by  $(intro\text{-classes, auto})$ 
end

```

```

definition  $nat\text{-mono} :: nat \Rightarrow bool$  where  $nat\text{-mono}\ x \equiv x \neq 0$ 

```

```

interpretation  $nat\text{-SN}: SN\text{-strict}\text{-mono}\text{-ordered}\text{-semiring}\text{-1}\ 1\ (>) :: nat \Rightarrow nat \Rightarrow$ 
 $bool\ nat\text{-mono}$ 
by  $(unfold\text{-locales, insert}\ SN\text{-nat}\text{-gt, auto}\ simp: nat\text{-mono}\text{-def})$ 

```

```

interpretation  $nat\text{-poly}: poly\text{-order}\text{-carrier}\ 1\ (>) :: nat \Rightarrow nat \Rightarrow bool\ True\ dis\text{-}$ 
 $crete$ 

```

```

proof  $(unfold\text{-locales})$ 
fix  $x\ y :: nat$ 
assume  $ge: x \geq y$ 
obtain  $k$  where  $k: x - y = k$  by  $auto$ 

```

```

show  $\exists k. x = ((+) 1 \widehat{\sim} k) y$ 
proof (rule exI[of - k])
  from ge k have  $x = k + y$  by simp
  also have  $\dots = ((+) 1 \widehat{\sim} k) y$ 
  by (induct k, auto)
  finally show  $x = ((+) 1 \widehat{\sim} k) y$  .
qed
qed (auto simp: field-simps power-strict-mono)

```

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

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

lemma non-inf-delta-gt: **assumes** $\text{delta}: \delta > 0$

shows $\text{non-inf } \{(a,b) . \text{delta-gt } \delta a b\}$ (**is non-inf ?r**)

proof

let $?gt = \text{delta-gt } \delta$

fix $a :: 'a$ **and** f

assume $\bigwedge i. (f i, f (\text{Suc } i)) \in ?r$

hence $gt: \bigwedge i. ?gt (f i) (f (\text{Suc } i))$ **by** *simp*

{

fix i

have $f i \leq f 0 - \delta * \text{of-nat } i$

proof (*induct i*)

case (*Suc i*)

thus $?case$ **using** gt [of i , *unfolded delta-gt-def*] **by** (*auto simp: field-simps*)

qed *simp*

} **note** $fi = \text{this}$

{

fix $r :: 'a$

have $\text{of-nat } (\text{nat } (\text{ceiling } r)) \geq r$

by (*metis ceiling-le-zero le-of-int-ceiling less-le-not-le nat-0-iff not-less of-nat-0*)

of-nat-nat)

} **note** $\text{ceil-elim} = \text{this}$

define i **where** $i = \text{nat } (\text{ceiling } ((f 0 - a) / \delta))$

from fi [of i] **have** $f i - f 0 \leq -\delta * \text{of-nat } (\text{nat } (\text{ceiling } ((f 0 - a) / \delta)))$

unfolding i -def **by** *simp*

also have $\dots \leq -\delta * ((f 0 - a) / \delta)$ **using** ceil-elim [of $(f 0 - a) / \delta$] *delta*

by (*metis le-imp-neg-le minus-mult-commute mult-le-cancel-left-pos*)

also have $\dots = -f 0 + a$ **using** *delta* **by** *auto*

also have $\dots < -f 0 + a + \delta$ **using** *delta* **by** *auto*

finally have $\neg ?gt (f i) a$ **unfolding** delta-gt-def **by** *arith*

thus $\exists i. (f i, a) \notin ?r$ **by** *blast*

qed

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

proof –

from *non-inf-imp-SN-bound*[*OF non-inf-delta-gt*[*OF dpos*], *of - δ*]

show *?thesis unfolding delta-gt-def by auto*

qed

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

subclass (in *floor-ceiling*) *large-ordered-semiring-1*

proof

fix $x :: 'a$

from *ex-le-of-int*[*of x*] **obtain** z **where** $x \leq \text{of-int } z$ **by** *auto*

have $z \leq \text{int } (\text{nat } z)$ **by** *auto*

with x **have** $x \leq \text{of-int } (\text{int } (\text{nat } z))$

by (*metis (full-types) le-cases of-int-0-le-iff of-int-of-nat-eq of-nat-0-le-iff of-nat-nat order-trans*)

also have $\dots = \text{of-nat } (\text{nat } z)$ **unfolding** *of-int-of-nat-eq ..*

finally

show $\exists y. x \leq \text{of-nat } y$ **by** *blast*

qed (*auto simp: mult-right-mono mult-left-mono mult-right-mono-neg max-def*)

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*

proof –

from *dpos default* **have** $\text{defz}: 0 \leq \text{def}$ **by** *auto*

show *?thesis*

proof (*unfold-locales*)

show *SN $\{(x,y). y \geq 0 \wedge \text{delta-gt } \delta x y\}$ by (rule delta-gt-SN[*OF dpos*])*

next

fix $x y z :: 'a$

assume *delta-mono x and yz: delta-gt $\delta y z$*

hence $x: 1 \leq x$ **unfolding** *delta-mono-def by simp*

have $\exists d > 0. \text{delta-gt } \delta = (\lambda x y. d \leq x - y)$

by (*rule exI[*of - δ*], auto simp: dpos delta-gt-def*)

from *this* **obtain** d **where** $d: 0 < d$ **and** *rat: delta-gt $\delta = (\lambda x y. d \leq x - y)$*

by *auto*

from yz **have** $yzd: d \leq y - z$ **by** (*simp add: rat*)

show *delta-gt $\delta (x * y) (x * z)$*

proof (*simp only: rat*)

let $?p = (x - 1) * (y - z)$

from x **have** $x1: 0 \leq x - 1$ **by** *auto*

from $yzd d$ **have** $yz0: 0 \leq y - z$ **by** *auto*

have $0 \leq ?p$

by (*rule mult-nonneg-nonneg[*OF x1 yz0*]*)

have $x * y - x * z = x * (y - z)$ **using** *right-diff-distrib[*of x y z*]* **by** *auto*

also have $\dots = ((x - 1) + 1) * (y - z)$ **by** *auto*

also have $\dots = ?p + 1 * (y - z)$ **by** (*rule ring-distrib(2)*)

also have $\dots = ?p + (y - z)$ **by** *simp*

also have $\dots \geq (0 + d)$ **using** $yzd \langle 0 \leq ?p \rangle$ **by** *auto*

```

    finally
      show  $d \leq x * y - x * z$  by auto
    qed
  qed (insert dpos, auto simp: delta-gt-def default defz)
qed

lemma delta-poly: assumes dpos:  $\delta > 0$  and default:  $\delta \leq def$ 
  shows poly-order-carrier def (delta-gt  $\delta$ ) ( $1 \leq \delta$ ) False
proof -
  from delta-interpretation[OF dpos default]
  interpret SN-strict-mono-ordered-semiring-1 def delta-gt  $\delta$  delta-mono .
  interpret poly-order-carrier def delta-gt  $\delta$  False False
  proof (unfold-locales)
    fix  $y z x :: 'a$ 
    assume gt: delta-gt  $\delta$   $y z$  and ge:  $x \geq 1$ 
    from ge have ge:  $x \geq 0$  and m: delta-mono  $x$  unfolding delta-mono-def by
  auto
    show delta-gt  $\delta$  ( $y * x$ ) ( $z * x$ )
      using mono[OF m gt ge] by (auto simp: field-simps)
    next
      fix  $x y :: 'a$  and  $n :: nat$ 
      assume False thus delta-gt  $\delta$  ( $x \wedge n$ ) ( $y \wedge n$ ) ..
    next
      fix  $x y :: 'a$ 
      assume False
      thus  $\exists k. x = ((+) 1 \wedge k) y$  by simp
    qed
    show ?thesis
    proof (unfold-locales)
      fix  $x y :: 'a$  and  $n :: nat$ 
      assume one:  $1 \leq \delta$  and gt: delta-gt  $\delta$   $x y$  and y:  $y \geq 0$  and n:  $1 \leq n$ 
      then obtain  $p$  where  $n = Suc p$  and  $x \geq 1$  and  $y2: 0 \leq y$  and  $xy: x$ 
       $\geq y$  by (cases  $n$ , auto simp: delta-gt-def)
      show delta-gt  $\delta$  ( $x \wedge n$ ) ( $y \wedge n$ )
      proof (simp only:  $n$ , induct  $p$ , simp add: gt)
        case (Suc  $p$ )
          from times-gt-mono[OF this  $x$ ]
          have one: delta-gt  $\delta$  ( $x \wedge Suc (Suc p)$ ) ( $x * y \wedge Suc p$ ) by (auto simp:
  field-simps)
          also have ...  $\geq y * y \wedge Suc p$ 
            by (rule times-left-mono[OF -  $xy$ ], auto simp: zero-le-power[OF  $y2$ , of  $Suc$ 
   $p$ , simplified])
          finally show ?case by auto
        qed
      next
        fix  $x y :: 'a$ 
        assume False
        thus  $\exists k. x = ((+) 1 \wedge k) y$  by simp
      qed (rule times-gt-mono, auto)

```

qed

lemma *delta-minimal-delta*: **assumes** $\bigwedge x y. (x,y) \in \text{set } xys \implies x > y$
shows $\exists \delta > 0. \forall x y. (x,y) \in \text{set } xys \longrightarrow \text{delta-gt } \delta x y$
using *assms*
proof (*induct xys*)
 case *Nil*
 show *?case* **by** (*rule exI[of - 1], auto*)
next
 case (*Cons xy xys*)
 show *?case*
 proof (*cases xy*)
 case (*Pair x y*)
 with *Cons* **have** $x > y$ **by** *auto*
 then obtain *d1* **where** $d1 = x - y$ **and** *d1pos*: $d1 > 0$ **and** $d1 \leq x - y$ **by**
 auto
 hence *xy*: $\text{delta-gt } d1 x y$ **unfolding** *delta-gt-def* **by** *auto*
 from *Cons* **obtain** *d2* **where** *d2pos*: $d2 > 0$ **and** *xys*: $\forall x y. (x, y) \in \text{set } xys$
 $\longrightarrow \text{delta-gt } d2 x y$ **by** *auto*
 obtain *d* **where** $d = \min d1 d2$ **by** *auto*
 with *d1pos d2pos xy* **have** *dpos*: $d > 0$ **and** $\text{delta-gt } d x y$ **unfolding** *delta-gt-def*
 by *auto*
 with *xys d Pair* **have** $\forall x y. (x,y) \in \text{set } (xy \# xys) \longrightarrow \text{delta-gt } d x y$ **unfolding**
 delta-gt-def **by** *force*
 with *dpos* **show** *?thesis* **by** *auto*
 qed
 qed

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

fix *xysp* :: ($'a \times 'a$) *list*
 assume *orient*: $\forall x y. (x,y) \in \text{set } xysp \longrightarrow x > y$
 obtain *xys* **where** *xsy*: $xys = (1,0) \# xysp$ **by** *auto*
 with *orient* **have** $\bigwedge x y. (x,y) \in \text{set } xys \implies x > y$ **by** *auto*
 with *delta-minimal-delta* **have** $\exists \delta > 0. \forall x y. (x,y) \in \text{set } xys \longrightarrow \text{delta-gt } \delta x$
 y **by** *auto*
 then obtain δ **where** *dpos*: $\delta > 0$ **and** *orient*: $\bigwedge x y. (x,y) \in \text{set } xys \implies \text{delta-gt } \delta x y$
 by *auto*
 from *orient* **have** *orient1*: $\forall x y. (x,y) \in \text{set } xysp \longrightarrow \text{delta-gt } \delta x y$ **and** *orient2*:
 delta-gt $\delta 1 0$ **unfolding** *xsy* **by** *auto*
 from *orient2* **have** *oned*: $\delta \leq 1$ **unfolding** *delta-gt-def* **by** *auto*
 show $\exists \text{gt. } \text{SN-strict-mono-ordered-semiring-1 } 1 \text{ gt } \text{delta-mono} \wedge (\forall x y. (x, y) \in \text{set } xysp \longrightarrow \text{gt } x y)$
 by (*intro exI conjI, rule delta-interpretation[OF dpos oned], rule orient1*)
 qed

5.3 The standard semiring over the integers

definition *int-mono* :: *int* ⇒ *bool* **where** *int-mono* *x* ≡ *x* ≥ 1

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

begin

instance

proof

fix *y* :: *int*

show ∃ *x*. *of-nat* *x* ≥ *y*

by (*rule* *exI*[*of - nat y*], *simp*)

qed (*auto simp: mult-right-mono mult-left-mono mult-right-mono-neg*)

end

lemma *non-inf-int-gt*: *non-inf* {(*a, b* :: *int*) . *a* > *b*} (**is** *non-inf* ?*r*)

by (*rule* *non-inf-image*[*OF non-inf-delta-gt, of 1 - rat-of-int*], *auto simp: delta-gt-def*)

interpretation *int-SN*: *SN-strict-mono-ordered-semiring-1* 1 (>) :: *int* ⇒ *int* ⇒ *bool int-mono*

proof (*unfold-locales*)

have [*simp*]: ∧ *x* :: *int* . (*-1* < *x*) = (*0* ≤ *x*) **by** *auto*

show *SN* {(*x, y*). *y* ≥ 0 ∧ (*y* :: *int*) < *x*}

using *non-inf-imp-SN-bound*[*OF non-inf-int-gt, of -1*] **by** *auto*

qed (*auto simp: mult-strict-left-mono int-mono-def*)

interpretation *int-poly*: *poly-order-carrier* 1 (>) :: *int* ⇒ *int* ⇒ *bool True discrete*

proof (*unfold-locales*)

fix *x y* :: *int*

assume *ge*: *x* ≥ *y*

then obtain *k* **where** *k*: *x - y = k* **and** *kp*: *0* ≤ *k* **by** *auto*

then obtain *nk* **where** *nk*: *nk = nat k* **and** *k*: *x - y = int nk* **by** *auto*

show ∃ *k*. *x = ((+)* 1 $\overset{\sim}{\sim}$ *k*) *y*

proof (*rule* *exI*[*of - nk*])

from *k* **have** *x = int nk + y* **by** *simp*

also have ... = ((+) 1 $\overset{\sim}{\sim}$ *nk*) *y*

by (*induct nk, auto*)

finally show *x = ((+)* 1 $\overset{\sim}{\sim}$ *nk*) *y* .

qed

qed (*auto simp: field-simps power-strict-mono*)

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* ⇒ *arctic* ⇒ *bool* **where**

```

    less-eq-arctic MinInfty x = True
| less-eq-arctic (Num-arc -) MinInfty = False
| less-eq-arctic (Num-arc y) (Num-arc x) = (y ≤ x)

fun less-arctic :: arctic ⇒ arctic ⇒ bool where
    less-arctic MinInfty x = True
| less-arctic (Num-arc -) MinInfty = False
| less-arctic (Num-arc y) (Num-arc x) = (y < x)

instance ..
end

instantiation arctic :: ordered-semiring-1
begin
fun plus-arctic :: arctic ⇒ arctic ⇒ 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 ⇒ arctic ⇒ 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
    fix x y z :: arctic
    show x + y = y + x
      by (cases x, cases y, auto, cases y, auto)
    show (x + y) + z = x + (y + z)
      by (cases x, auto, cases y, auto, cases z, auto)
    show (x * y) * z = x * (y * z)
      by (cases x, auto, cases y, auto, cases z, auto)
    show x * 0 = 0
      by (cases x, auto simp: zero-arctic-def)
    show x * (y + z) = x * y + x * z
      by (cases x, auto, cases y, auto, cases z, auto)
    show (x + y) * z = x * z + y * z
      by (cases x, auto, cases y, cases z, auto, cases z, auto)
    show 1 * x = x
      by (cases x, simp-all add: one-arctic-def)
    show x * 1 = x
      by (cases x, simp-all add: one-arctic-def)

```

```

show  $0 + x = x$ 
  by (simp add: zero-arctic-def)
show  $0 * x = 0$ 
  by (simp add: zero-arctic-def)
show  $(0 :: \text{arctic}) \neq 1$ 
  by (simp add: zero-arctic-def one-arctic-def)
show  $x + 0 = x$  by (cases x, auto simp: zero-arctic-def)
show  $x \geq x$ 
  by (cases x, auto)
show  $(1 :: \text{arctic}) \geq 0$ 
  by (simp add: zero-arctic-def one-arctic-def)
show  $\max x y = \max y x$  unfolding max-def
  by (cases x, (cases y, auto)+)
show  $\max x y \geq x$  unfolding max-def
  by (cases x, (cases y, auto)+)
assume ge: x ≥ y
from ge show  $x + z \geq y + z$ 
  by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
from ge show  $x * z \geq y * z$ 
  by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
from ge show  $\max x y = x$  unfolding max-def
  by (cases x, (cases y, auto)+)
from ge show  $\max z x \geq \max z y$  unfolding max-def
  by (cases z, cases x, auto, cases x, (cases y, auto)+)
next
  fix x y z :: arctic
  assume  $x \geq y$  and  $y \geq z$ 
  thus  $x \geq z$ 
    by (cases x, cases y, auto, cases y, cases z, auto, cases z, auto)
next
  fix x y z :: arctic
  assume  $y \geq z$ 
  thus  $x * y \geq x * z$ 
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
next
  fix x y z :: arctic
  show  $x \geq y \implies 0 \geq z \implies y * z \geq x * z$ 
    by (cases z, cases x, auto simp: zero-arctic-def)
qed
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) =  $(0 \leq n)$ 

```

```

interpretation arctic-SN: SN-both-mono-ordered-semiring-1 1 (>) pos-arctic
proof
  fix x y z :: arctic
  assume  $x \geq y$  and  $y > z$ 
  thus  $x > z$ 
    by (cases z, simp, cases y, simp, cases x, auto)
next
  fix x y z :: arctic
  assume  $x > y$  and  $y \geq z$ 
  thus  $x > z$ 
    by (cases z, simp, cases y, simp, cases x, auto)
next
  fix x y z :: arctic
  assume  $x > y$ 
  thus  $x \geq y$ 
    by (cases x, (cases y, auto)+)
next
  fix x y z u :: arctic
  assume  $x > y$  and  $z > u$ 
  thus  $x + z > y + u$ 
    by (cases y, cases u, simp, cases z, auto, cases x, auto, cases u, auto, cases z,
    auto, cases x, auto, cases x, auto, cases z, auto, cases x, auto)
next
  fix x y z :: arctic
  assume  $x > y$ 
  thus  $x * z > y * z$ 
    by (cases y, simp, cases z, simp, cases x, auto)
next
  fix x :: arctic
  assume  $0 > x$ 
  thus  $x = 0$ 
    by (cases x, auto simp: zero-arctic-def)
next
  fix x :: arctic
  show pos-arctic 1 unfolding one-arctic-def by simp
  show  $x > 0$  unfolding zero-arctic-def by simp
  show  $(1 :: arctic) \geq 0$  unfolding zero-arctic-def by simp
  show  $x \geq 0$  unfolding zero-arctic-def by simp
  show  $\neg$  pos-arctic 0 unfolding zero-arctic-def by simp
next
  fix x y
  assume pos-arctic x
  thus pos-arctic (x + y) by (cases x, simp, cases y, auto)
next
  fix x y
  assume pos-arctic x and pos-arctic y
  thus pos-arctic (x * y) by (cases x, simp, cases y, auto)
next

```

```

show SN {(x,y). pos-arctic y ∧ x > y} (is SN ?rel)
proof - {
  fix x
  assume ∃ f . f 0 = x ∧ (∀ i. (f i, f (Suc i)) ∈ ?rel)
  from this obtain f where f 0 = x and seq: ∀ i. (f i, f (Suc i)) ∈ ?rel by
auto
  from seq have steps: ∀ i. f i > f (Suc i) ∧ pos-arctic (f (Suc i)) by auto
  let ?g = λ i. get-arctic-num (f i)
  have ∀ i. ?g (Suc i) ≥ 0 ∧ ?g i > ?g (Suc i)
  proof
    fix i
    from steps have i: f i > f (Suc i) ∧ pos-arctic (f (Suc i)) by auto
    from i obtain n where fi: f i = Num-arc n by (cases f (Suc i), simp, cases
f i, auto)
    from i obtain m where fsi: f (Suc i) = Num-arc m by (cases f (Suc i),
auto)
    with i have gz: 0 ≤ m by simp
    from i fi fsi have n > m by auto
    with fi fsi gz
    show ?g (Suc i) ≥ 0 ∧ ?g i > ?g (Suc i) by auto
  qed
  from this obtain g where ∀ i. g (Suc i) ≥ 0 ∧ ((>) :: int ⇒ int ⇒ bool) (g
i) (g (Suc i)) by auto
  hence ∃ f. f 0 = g 0 ∧ (∀ i. (f i, f (Suc i)) ∈ {(x,y). y ≥ 0 ∧ x > y}) by
auto
  with int-SN.SN have False unfolding SN-defs by auto
}
thus ?thesis unfolding SN-defs by auto
qed
next
fix y z x :: arctic
assume y > z
thus x * y > x * z
by (cases x, simp, cases z, simp, cases y, auto)
next
fix c d
assume pos-arctic d
then obtain n where d: d = Num-arc n and n: 0 ≤ n
by (cases d, auto)
show ∃ e. e ≥ 0 ∧ pos-arctic e ∧ ¬ c ≥ d * e
proof (cases c)
  case MinInfty
  show ?thesis
  by (rule exI[of - Num-arc 0],
unfold d MinInfty zero-arctic-def, simp)
next
  case (Num-arc m)
  show ?thesis
  by (rule exI[of - Num-arc (abs m + 1)], insert n,

```

```

    unfold d Num-arc zero-arctic-def, simp)
  qed
  qed

```

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 ..

```

```

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

```

```

  fix x y z :: 'a arctic-delta

```

```

  show x + y = y + x

```

```

  by (cases x, cases y, auto, cases y, auto)

```

```

show  $(x + y) + z = x + (y + z)$ 
  by (cases x, auto, cases y, auto, cases z, auto)
show  $(x * y) * z = x * (y * z)$ 
  by (cases x, auto, cases y, auto, cases z, auto)
show  $x * 0 = 0$ 
  by (cases x, auto simp: zero-arctic-delta-def)
show  $x * (y + z) = x * y + x * z$ 
  by (cases x, auto, cases y, auto, cases z, auto)
show  $(x + y) * z = x * z + y * z$ 
  by (cases x, auto, cases y, cases z, auto, cases z, auto)
show  $1 * x = x$ 
  by (cases x, simp-all add: one-arctic-delta-def)
show  $x * 1 = x$ 
  by (cases x, simp-all add: one-arctic-delta-def)
show  $0 + x = x$ 
  by (simp add: zero-arctic-delta-def)
show  $0 * x = 0$ 
  by (simp add: zero-arctic-delta-def)
show  $(0 :: 'a \text{ arctic-delta}) \neq 1$ 
  by (simp add: zero-arctic-delta-def one-arctic-delta-def)
show  $x + 0 = x$  by (cases x, auto simp: zero-arctic-delta-def)
show  $x \geq x$ 
  by (cases x, auto)
show  $(1 :: 'a \text{ arctic-delta}) \geq 0$ 
  by (simp add: zero-arctic-delta-def one-arctic-delta-def)
show  $\max x y = \max y x$  unfolding max-def
  by (cases x, (cases y, auto)+)
show  $\max x y \geq x$  unfolding max-def
  by (cases x, (cases y, auto)+)
assume ge: x ≥ y
from ge show  $x + z \geq y + z$ 
  by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
from ge show  $x * z \geq y * z$ 
  by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
from ge show  $\max x y = x$  unfolding max-def
  by (cases x, (cases y, auto)+)
from ge show  $\max z x \geq \max z y$  unfolding max-def
  by (cases z, cases x, auto, cases x, (cases y, auto)+)
next
  fix  $x y z :: 'a \text{ arctic-delta}$ 
  assume  $x \geq y$  and  $y \geq z$ 
  thus  $x \geq z$ 
  by (cases x, cases y, auto, cases y, cases z, auto, cases z, auto)
next
  fix  $x y z :: 'a \text{ arctic-delta}$ 
  assume  $y \geq z$ 
  thus  $x * y \geq x * z$ 
  by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
next

```

```

fix x y z :: 'a arctic-delta
show x ≥ y ⇒ 0 ≥ z ⇒ y * z ≥ x * z
  by (cases z, cases x, auto simp: zero-arctic-delta-def)
qed
end

```

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

```

fun gt-arctic-delta :: 'a :: floor-ceiling ⇒ 'a arctic-delta ⇒ 'a arctic-delta ⇒ 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 ⇒ 'a
where get-arctic-delta-num (Num-arc-delta n) = n

```

```

fun pos-arctic-delta :: ('a :: floor-ceiling) arctic-delta ⇒ bool
where pos-arctic-delta MinInfty-delta = False
  | pos-arctic-delta (Num-arc-delta n) = (0 ≤ 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 –

from delta-interpretation[OF dpos] **interpret** SN-strict-mono-ordered-semiring-1
δ delta-gt δ delta-mono **by** simp

show ?thesis

proof

fix x y z :: 'a arctic-delta

assume x ≥ y **and** gt-arctic-delta δ y z

thus gt-arctic-delta δ x z

by (cases z, simp, cases y, simp, cases x, simp, simp add: compat)

next

fix x y z :: 'a arctic-delta

assume gt-arctic-delta δ x y **and** y ≥ z

thus gt-arctic-delta δ x z

by (cases z, simp, cases y, simp, cases x, simp, simp add: compat2)

next

fix x y :: 'a arctic-delta

assume gt-arctic-delta δ x y

thus x ≥ y

by (cases x, insert dpos, (cases y, auto simp: delta-gt-def)+)

next

fix x y z u

assume gt-arctic-delta δ x y **and** gt-arctic-delta δ z u

thus gt-arctic-delta δ (x + z) (y + u)

by (cases y, cases u, simp, cases z, simp, cases x, simp, simp add: delta-gt-def,

cases z, cases x, simp, cases u, simp, simp, cases x, simp, cases z, simp,
cases u, simp add: delta-gt-def, simp add: delta-gt-def)


```

next
  fix x y z
  assume gt-arctic-delta  $\delta$  x y
  thus gt-arctic-delta  $\delta$  (x * z) (y * z)
    by (cases y, simp, cases z, simp, cases x, simp, simp add: plus-gt-left-mono)
next
  fix x
  assume gt-arctic-delta  $\delta$  0 x
  thus x = 0
    by (cases x, auto simp: zero-arctic-delta-def)
next
  fix x
  show pos-arctic-delta 1 unfolding one-arctic-delta-def by simp
  show gt-arctic-delta  $\delta$  x 0 unfolding zero-arctic-delta-def by simp
  show (1 :: 'a arctic-delta)  $\geq$  0 unfolding zero-arctic-delta-def by simp
  show x  $\geq$  0 unfolding zero-arctic-delta-def by simp
  show  $\neg$  pos-arctic-delta 0 unfolding zero-arctic-delta-def by simp
next
  fix x y :: 'a arctic-delta
  assume pos-arctic-delta x
  thus pos-arctic-delta (x + y) by (cases x, simp, cases y, auto)
next
  fix x y :: 'a arctic-delta
  assume pos-arctic-delta x and pos-arctic-delta y
  thus pos-arctic-delta (x * y) by (cases x, simp, cases y, auto)
next
  show SN {(x,y). pos-arctic-delta y  $\wedge$  gt-arctic-delta  $\delta$  x y} (is SN ?rel)
  proof - {
    fix x
    assume  $\exists f . f 0 = x \wedge (\forall i. (f i, f (Suc i)) \in ?rel)$ 
    from this obtain f where f 0 = x and seq:  $\forall i. (f i, f (Suc i)) \in ?rel$  by
  auto
    from seq have steps:  $\forall i. gt-arctic-delta \delta (f i) (f (Suc i)) \wedge pos-arctic-delta$ 
    (f (Suc i)) by auto
    let ?g =  $\lambda i. get-arctic-delta-num (f i)$ 
    have  $\forall i. ?g (Suc i) \geq 0 \wedge delta-gt \delta (?g i) (?g (Suc i))$ 
    proof
      fix i
      from steps have i:  $gt-arctic-delta \delta (f i) (f (Suc i)) \wedge pos-arctic-delta$ 
    (f (Suc i)) by auto
      from i obtain n where fi:  $f i = Num-arc-delta n$  by (cases f (Suc i), simp,
    cases f i, auto)
      from i obtain m where fsi:  $f (Suc i) = Num-arc-delta m$  by (cases f (Suc
    i), auto)
      with i have gz:  $0 \leq m$  by simp
      from i fi fsi have delta-gt  $\delta n m$  by auto
      with fi fsi gz
      show  $?g (Suc i) \geq 0 \wedge delta-gt \delta (?g i) (?g (Suc i))$  by auto
    qed
  }

```

```

    from this obtain g where  $\forall i. g (Suc i) \geq 0 \wedge \text{delta-gt } \delta (g i) (g (Suc i))$ 
  by auto
    hence  $\exists f. f 0 = g 0 \wedge (\forall i. (f i, f (Suc i)) \in \{(x,y). y \geq 0 \wedge \text{delta-gt } \delta x y\})$  by auto
    with SN have False unfolding SN-defs by auto
  }
  thus ?thesis unfolding SN-defs by auto
qed
next
fix c d :: 'a arctic-delta
assume pos-arctic-delta d
then obtain n where d:  $d = \text{Num-arc-delta } n$  and  $n: 0 \leq n$ 
  by (cases d, auto)
show  $\exists e. e \geq 0 \wedge \text{pos-arctic-delta } e \wedge \neg c \geq d * e$ 
proof (cases c)
  case MinInfty-delta
  show ?thesis
    by (rule exI[of - Num-arc-delta 0],
        unfold d MinInfty-delta zero-arctic-delta-def, simp)
  next
  case (Num-arc-delta m)
  show ?thesis
    by (rule exI[of - Num-arc-delta (abs m + 1)], insert n,
        unfold d Num-arc-delta zero-arctic-delta-def, simp)
qed
next
fix x y z
assume gt:  $\text{gt-arctic-delta } \delta y z$ 
{
  fix x y z
  assume gt:  $\text{delta-gt } \delta y z$ 
  have  $\text{delta-gt } \delta (x + y) (x + z)$ 
    using plus-gt-left-mono[OF gt] by (auto simp: field-simps)
}
with gt show  $\text{gt-arctic-delta } \delta (x * y) (x * z)$ 
  by (cases x, simp, cases z, simp, cases y, simp-all)
qed
qed

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
  fix xys
  assume orient:  $\forall x y. (x,y) \in \text{set } xys \longrightarrow \text{weak-gt-arctic-delta } x y$ 

```

```

obtain xysp where xysp: xysp = map ( $\lambda$  (ax, ay). (case ax of Num-arc-delta x
 $\Rightarrow$  x, case ay of Num-arc-delta y  $\Rightarrow$  y)) (filter ( $\lambda$  (ax,ay). ax  $\neq$  MinInfty-delta  $\wedge$ 
ay  $\neq$  MinInfty-delta) xys)
  (is - = map ?f -)
  by auto
have  $\forall$  x y. (x,y)  $\in$  set xysp  $\longrightarrow$  x > y
proof (intro allI impI)
  fix x y
  assume (x,y)  $\in$  set xysp
  with xysp obtain ax ay where (ax,ay)  $\in$  set xys and ax  $\neq$  MinInfty-delta
and ay  $\neq$  MinInfty-delta and (x,y) = ?f (ax,ay) by auto
  hence (Num-arc-delta x, Num-arc-delta y)  $\in$  set xys by (cases ax, simp, cases
ay, auto)
  with orient show x > y by force
qed
with delta-minimal-delta[of xysp] obtain  $\delta$  where dpos:  $\delta$  > 0 and orient2:  $\wedge$ 
x y. (x, y)  $\in$  set xysp  $\Longrightarrow$  delta-gt  $\delta$  x y by auto
have orient:  $\forall$  x y. (x,y)  $\in$  set xys  $\longrightarrow$  gt-arctic-delta  $\delta$  x y
proof(intro allI impI)
  fix ax ay
  assume axay: (ax,ay)  $\in$  set xys
  with orient have orient: weak-gt-arctic-delta ax ay by auto
  show gt-arctic-delta  $\delta$  ax ay
  proof (cases ay, simp)
    case (Num-arc-delta y) note ay = this
    show ?thesis
    proof (cases ax)
      case MinInfty-delta
      with ay orient show ?thesis by auto
    next
      case (Num-arc-delta x) note ax = this
      from ax ay axay have (x,y)  $\in$  set xysp unfolding xysp by force
      from ax ay orient2[OF this] show ?thesis by simp
  qed
qed
qed
show  $\exists$  gt. SN-both-mono-ordered-semiring-1 1 gt pos-arctic-delta  $\wedge$  ( $\forall$  x y. (x, y)
 $\in$  set xys  $\longrightarrow$  gt x y)
  by (intro exI conjI, rule arctic-delta-interpretation[OF dpos], rule orient)
qed
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.