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 website\(^1\)). Hence lemmas involving strong normalization, constitute by far the biggest part of this theory. One of those is Newman’s lemma.


c\text{Contents}
c\begin{center}
\textbf{1 Infinite Sequences} & \textbf{2} \\
1.1 Operations on Infinite Sequences & \textbf{2} \\
1.2 Predicates on Natural Numbers & \textbf{4} \\
1.3 Assembling Infinite Words from Finite Words & \textbf{7} \\
\hline
\textbf{2 Abstract Rewrite Systems} & \textbf{13} \\
2.1 Definitions & \textbf{13} \\
2.2 Properties of ARSs & \textbf{18} \\
2.3 Newman’s Lemma & \textbf{37} \\
2.4 Commutation & \textbf{43} \\
2.5 Strong Normalization & \textbf{47} \\
2.6 Terminating part of a relation & \textbf{62} \\
\hline
\textbf{3 Relative Rewriting} & \textbf{72} \\
\hline
\textbf{4 Strongly Normalizing Orders} & \textbf{109} \\
\end{center}

\(^1\)\url{http://cl-informatik.uibk.ac.at/software/ceta}
1 Infinite Sequences

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

Infinite sequences are represented by functions of type nat ⇒ 'a.

type-synonym 'a seq = nat ⇒ '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 ⇒ 'a ⇒ bool) ⇒ 'a seq ⇒ bool where
  chainp $P$ $S$ ≡ ∀$i$. $P$ ($S$ $i$) ($S$ (Suc $i$))

Special version for relations.

abbreviation (input) chain :: 'a rel ⇒ 'a seq ⇒ bool where
  chain $r$ $S$ ≡ chainp ($λ$x y. ($x$, $y$) ∈ $r$) $S$

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

Special version for relations.

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

A chain admits arbitrary transitive steps.

lemma chainp-imp-relpowp:
  assumes chain $P$ $S$ shows ($P$ $^*$ $j$) ($S$ $i$) ($S$ ($i$ + $j$))
proof (induct \( i + j \) arbitrary: \( j \))
  case (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 (Suc \( n \)) thus ?case using assms by (cases \( j \)) auto
qed simp

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

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

A chain admits arbitrary reflexive and transitive steps.

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

lemma chainp-imp-rtrancl:
  assumes chainp \( P S \) and \( i \leq j \) shows \((S i, (S j)) \in r^*\)
proof (induct \( j - i \) arbitrary: \( j \))
  from \( \text{assms}(2) \) obtain \( n \) where \( j = i + n \) by (induct \( j - i \) arbitrary: \( j \)) force+
  with chainp-imp-relpow[of \( P S \) \( \text{OF} \ \text{assms}(1) \), \( \text{of} \ n \) \( i \)]
  show \( \text{thesis} \)
  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 \( \forall i \geq n \cdot \text{nat}. \ f \ i \geq i \ \land \ P (S i) (S (f i)) \)
  shows chainp \( P (\lambda i. S ((f ^\sim i) n)) \) (is chainp \( P \) ?T)
proof
  fix i
  from assms have \((f \ ^\ ^\ i) \ n \geq n\) by (induct i) auto
  with assms[THEN spec[of \((f \ ^\ ^\ i) \ n\)]]
  show \(P \ (? i) \ (? i \ (Suc i))\) by simp
qed

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

lemma subchain:
  assumes \(\forall i::\text{nat}>n. \ \exists j>i. \ P \ (f \ i) \ (f \ j)\)
  shows \(\exists \varphi. \ (\forall i. \ i < j \rightarrow \varphi \ i < \varphi \ j) \land (\forall i. \ P \ (f \ (\varphi \ i)) \ (f \ (\varphi \ (Suc \ i))))\)
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]: \(\varphi \ i = (g \ ^\ ^\ i) \ (Suc \ n)\) for i
  from * have ***: \(\forall i. \ \varphi \ i > n\) by (induct-tac i) auto
  then have \(\forall i. \ \varphi \ i < \varphi \ (Suc \ i)\) using * by (induct-tac i) auto
  then have \(\forall i. \ i < j \Rightarrow \varphi \ i < \varphi \ j\) by (rule lift-Suc-mono-less)
  moreover have \(\forall i. \ P \ (f \ (\varphi \ i)) \ (f \ (\varphi \ (Suc \ i)))\) using ** and *** by simp
  ultimately show \(\exists \text{thesis by blast}\)
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-imp-chainp':
  assumes \(\forall i\geq n::\text{nat}. \ \exists j\geq i. \ P \ (S \ i) \ (S \ j)\) shows \(\exists T. \ \text{chainp} \ P \ T\)
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 \(\forall i \geq n. \ f \ i \geq i \land P \ (S \ i) \ (S \ (f \ i))\) by auto
  from stepfun-imp-chainp'[of \(n \ f \ P \ S\), OF this] show \(\exists \text{thesis by fast}\)
qed

lemma steps-imp-chainp:
  assumes \(\forall i\geq n::\text{nat}. \ \exists j>i. \ P \ (S \ i) \ (S \ j)\) shows \(\exists T. \ \text{chainp} \ P \ T\)
using steps-imp-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-many =
  fixes \(p::\text{nat} \Rightarrow \text{bool}\)

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 ∧ 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 $k \geq \text{Suc (index n)} \land p k$ by auto
with LeastI[of $\lambda k. p k \land k > \text{index n} \ k$] show ?case by auto
qed

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

lemma index-not-p-between:
assumes i1: $\text{index n} < i$
and i2: $i < \text{index} (\text{Suc n})$
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 $\text{index i} \leq \text{index j}$
proof -
from assms have $j = i + (j - i)$ by auto
then obtain $k$ where $j = i + k$ by auto
have $\text{index i} \leq \text{index} (i + k)$
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 $\exists i, j. \; k = \text{index } i + j \land \text{index } i + j < \text{index } (\text{Suc } i)$
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 $\text{lt: index } i + j < \text{index } (\text{Suc } i)$ by auto
  hence $\text{index } l + u < \text{index } (\text{Suc } i)$ by auto
  show ?case
    proof (cases $\text{index } l + (\text{Suc } u) = \text{index } (\text{Suc } i)$)
      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.

\[ \text{fun } \text{inf-concat-simple} :: (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow (\text{nat} \times \text{nat}) \text{ where} \]
\[ \text{inf-concat-simple} f 0 = (0, 0) \]
\[ \text{inf-concat-simple} f (\text{Suc} \, n) = (\]
\[ \text{let } (i, j) = \text{inf-concat-simple} f \, n \text{ in} \]
\[ \text{if } \text{Suc} \, j < f \, i \text{ then } (i, \text{Suc} \, j) \]
\[ \text{else } (\text{Suc} \, i, 0) ) \]

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

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

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

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

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

lemma $\infconcat\text{-bounds}$:
  assumes $\inf i: \InfM i. n i > 0$
  and $\res: \infconcat n k = (i, j)$
  shows $j < n i$
proof (cases $k$)
  case $0$
  with $\res$ have $i: i = (\LEAST i. n i > 0)$ and $j: j = 0$ by auto
from $\inf[\text{unfolded } \InfM\text{-nat-le}]$ obtain $i' where i': 0 < n i'$ by auto
  have $0 < n (\LEAST i. n i > 0)$
    by (rule $\LeastI$, rule $i'$)
  with $i j$ show $\text{thesis}$ by auto
next
  case $\Suc k'$
  obtain $i' j'$ where $\res': \infconcat n k' = (i' j')$ by force
  note $\res = \res[\text{unfolded } \Suc \infconcat\text{-simp s}]$ Let-def split
  show $\text{thesis}$
proof (cases $Suc j' < n i'$)
case True
  with res show ?thesis by auto
next
case False
  with res have i: \( i = (\text{LEAST } f. \, i' < f \land 0 < n \, f)\) and \( j: \, j = 0 \) by auto
  from \( \text{inf[unfolded INFM-nat]} \) obtain f where i' < f \land 0 < n \, f by auto
  have 0 < n (\( \text{LEAST } f. \, i' < f \land 0 < n \, f)\)
    using LeastI[of \( \lambda f. \, i' < f \land 0 < n \, f \), OF f]
    by auto
  with i j show ?thesis by auto
qed

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

lemma inf-concat-step:
  assumes res: \( \text{inf-concat } n \, k = (i,j) \)
  and j: \( \text{Suc } (j + m) = n \, i \)
  shows \( \text{inf-concat } n \, (k + \text{Suc } m) = (\text{LEAST } i'. \, i' > i \land 0 < n \, i', 0) \)
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 \( \exists k. \, \text{inf-concat } n \, k = (i, 0) \)
proof –
  fix l
  have \( \forall \, j. \, j < l \land 0 < n \, j \longrightarrow (\exists \, k. \, \text{inf-concat } n \, k = (j,0)) \)
    proof (induct l)
      case \( 0 \)
      thus ?case by auto
next
case \( \text{Suc } l \)
show \( ?\text{case} \)
proof \( (\text{intro allI impI, elim conjE}) \)
  fix \( j \)
  assume \( j: j < \text{Suc } l \) 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 \( ?\text{thesis} \) by auto
next
  case False
  with \( j \) have \( j: j = l \) by auto
  show \( ?\text{thesis} \)
proof (cases \( \exists j'. j' < l \land 0 < n j' \))
  case False
  have \( l: (\text{LEAST } i. 0 < n i) = l \)
  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 \( ?\text{thesis} \) 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 \( ll = \text{Suc } lll \) by (cases \( l \), auto)
  from \( lll \) \( ll \) have \( lll = ll - (ll - lll) \) by auto
  let \( ?l' = \text{LEAST } d. 0 < n (ll - d) \)
  have \( nlll: 0 < n (ll - ?l' ) \)
  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 \( k: \text{inf-concat } n k = (ll - ?l', 0) \)
  unfolding \( l \) by auto
  from \( nlll \) \( off \) obtain \( \text{off: Suc } (0 + \text{off}) = n (ll - ?l') \) by (cases \( n (ll - ?l'), \text{ auto} \))
  from \( \text{inf-concat-step}[\text{OF } k, \text{ OF } off]\) have \( \text{id: inf-concat } n (k + \text{Suc } \text{off}) = (\text{LEAST } i', ll - ?l' < i' \land 0 < n i', 0) \) (is \( = (?l, 0) \)).
  have \( ll: \exists l = l \) unfolding \( l \)
  proof (rule Least-equality)
    show \( ll - ?l' < \text{Suc } ll \land 0 < n (\text{Suc } ll) \) using \( nj \[ unfolded j \ l \] \) by simp
  next
    fix \( l' \)
    assume \( \text{ass: } ll - ?l' < l' \land 0 < n l' \)
    show \( \text{Suc } ll \leq l' \)
    proof (rule ccontr)
      assume \( \text{not: } \neg ?\text{thesis} \)
hence \( l' \leq ll \) by auto

hence \( ll = l' + (ll - l') \) by auto

then obtain \( k \) where \( ll = k + l' \) by auto

from \( \text{ass have } l' + k = ll \) unfolding \( ll \) by auto

hence \( kl' \leq k \) by auto

have \( \theta < n (ll - k) \) using \( \text{ass unfolding } ll \) by simp

from \( \text{Least-le[of } \lambda k. \theta < n (ll - k), \text{OF this} ] kl' \)

show \( \text{False by auto} \)

qed

show \( \text{thesis unfolding } j \)

by (rule \text{exI[of - } k + \text{Suc of } \text{off}], \text{unfold id } ll, \text{simp})

qed

qed

qed

qed

qed

}\)

with \( \text{assms show } \text{thesis by auto} \)

qed

lemma \text{inf-concat-surj}:

\text{assumes } j: j < n i

\text{shows } \exists k. \text{inf-concat } n k = (i, j)

\text{proof --}

from \( j \) have \( 0 < n i \) by auto

from \( \text{inf-concat-surj-zero[of } n, \text{OF this] } \)

obtain \( k \) where \( \text{inf-concat } n k = (i, 0) \) by auto

from \( \text{inf-concat-add[of } n, \text{OF this, of } j \] j

show \( \text{thesis by auto} \)

qed

lemma \text{inf-concat-mono}:

\text{assumes } \theta: \text{INFM } i. \text{ n i } > \theta

\text{and resk: } \text{inf-concat } n k = (i, j)

\text{and reskp: } \text{inf-concat } n k' = (i', j')

\text{and lt: } i < i'

\text{shows } k < k'

\text{proof --}

note \( \text{bounds } = \text{inf-concat-bounds[of } \text{OF inf] } \)

\{\}

assume \( k' \leq k \)

hence \( k = k' + (k - k') \) by auto

then obtain \( l \) where \( k = k' + l \) by auto

have \( i' \leq \text{fst (inf-concat } n (k' + l)) \)

proof (induct \( l \))

\text{case } 0

with \( \text{reskp show } \text{case by auto} \)

next

\text{case } (\text{Suc } l)
obtain $i''$ $j''$ where $l$: \text{inf-concat} \ n \ (k' + l) = (i'', j'')$ \textbf{by force}
with $Suc$ have one: $i' \leq i''$ \textbf{by auto}
from \textit{bounds}[OF \ l] \ have \ $j'': j'' < n \ i''$ \textbf{by auto}
show \ ?case
proof (cases $Suc \ j'' < n \ i''$
  
  case $True$
  show \ ?thesis \textbf{by (simp add: l True one)}

  qed

  qed

  with \ \textit{resk} \ k \ \textit{lt} \ have \ $False$ \textbf{by auto}
}

thus \ ?thesis \textbf{by arith}

qed

\textbf{lemma \ \textit{inf-concat-Suc}:}

assumes \ \textit{inf}: $INFM \ i. \ n \ i > 0$
  and $f$: $\forall i. \ f \ i \ (n \ i) = f \ (Suc \ i) \ 0$
  and \ \textit{resk}: $\text{inf-concat} \ n \ k = (i, j)$
  and \ \textit{ressk}: $\text{inf-concat} \ n \ (Suc \ k) = (i', j')$
shows \ $f \ i' \ j' = f \ i \ (Suc \ j)$

proof (cases $Suc \ j < n \ i$
  
  case $True$
  with \ \textit{ressk} \ resk
  show \ ?thesis \textbf{by simp}

  next

  case $False$
  let $?i = \text{LEAST} \ i'. \ i'' < i' \land 0 < n \ i'$
  from \ \textit{inf}[unfolded \ \textit{INFM-nat}] \ \textit{obtain} \ k \ \textbf{where} \ i'' < k \land 0 < n \ k \ \textbf{by auto}
  from \ \textit{LeastI}[of \ \lambda \ k. \ i'' < k \land 0 < n \ k, \ \textit{OF} \ \textit{this}]
  have \ $i'' < ?i$ \textbf{by auto}
  with \ \textit{one} \ show \ ?thesis \textbf{by (simp add: l False)}

  qed

  qed

  with \ \textit{resk} \ k \ \textit{lt} \ have \ $False$ \textbf{by auto}
}

thus \ ?thesis \textbf{by arith}

qed
have \( f(\text{Suc}\ i)\ 0 = f\ ?i\ 0 \) unfolding \( \bar{ii}' \) using \( d' \)

proof (induct \( d \))
  case 0
  show \(?case\ by\ simp\)
  next
  case (Suc \( d \))
  hence \( f(\text{Suc} \ i)\ 0 = f(\text{Suc} \ (i + d))\ 0 \) by auto
  also have ... = \( f(\text{Suc}(\text{Suc} \ i + d))\ 0 \)
    unfolding \( f[\text{symmetric}] \)
    using \( \text{Suc}(2)[\text{of}\ d] \) 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:
  \( r \subseteq s \Rightarrow r^+ \subseteq s^+ \)
  by (blast intro: trancl-mono)

lemma relpow-mono:
  fixes \( r :: \ 'a\ rel \)
  assumes \( r \subseteq r' \) shows \( r \^ n \subseteq r' \^ n \)
  using assms by (induct \( n \)) auto

lemma refl-inv-image:
  refl \( R \Rightarrow\) refl (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 :: \( 'a\ rel \Rightarrow 'a\ rel \) \( ((\cdot)^\star)\ \[1000\] \[999\] \) where
  \( A^\star = A^*\ O\ (A^{-1})^\star \)

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 ((-) [1000] 999) where
\[ A^\dagger = (A^{-1})^* \ O \ A^* \]

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

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

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

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

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 ((-) [1000] 999) where
\[ A^! = \{ (a, b). (a, b) \in A^* \land b \in NF \ A \} \]

**notation** (ASCII)

symcl ((-'<->) [1000] 999) and
conversion ((-'<->+>*) [1000] 999) and
normalizability ((-)! [1000] 999)

**lemma** symcl-converse:
\[ (A^{**})^{-1} = A^{**} \text{ by auto} \]

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

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

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

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

**lemma** joinI-right: \((b, a) \in A^* \Rightarrow (a, b) \in A^\dagger\)
by (rule joinI) auto

**lemma** joinE:
assumes \((a, b) \in A^\dagger\)
obtains \( c \) where \((a, c) \in A^*\) and \((b, c) \in A^*\)
using assms by (auto simp: join-def rtrancl-converse)

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

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

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

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

lemma conversionI: \((a, b) \in (A^\leftrightarrow)^\ast \Longrightarrow (a, b) \in A^\leftrightarrow^\ast\)
  by (simp add: conversion-def)

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

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

lemma rtrancl-comp-trancl-conv:
  \(r^+ O r = r^+\) by regexp

lemma trancl-o-refl-is-trancl:
  \(r^+ O r^* = r^+\) by regexp

lemma conversionE:
  \((a, b) \in A^\star^\ast \Longrightarrow ((a, b) \in (A^\ast)^\ast \Longrightarrow P) \Longrightarrow P\)
  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**)
unfolding trans-def
proof (intro allI impI)
fix a b c assume (a, b) ∈ A** and (b, c) ∈ A**
then show (a, c) ∈ A*** unfolding conversion-def
proof (induct)
case base then show ?case by simp
next
case (step b c')
from ⟨(b, c') ∈ A**⟩ and ⟨(c', c) ∈ (A**)⟩;
have (b, c) ∈ (A**)' by (rule converse-rtrancl-into-rtrancl)
with step show ?case by simp
qed
qed

lemma conversion-sym:
sym (A**)
unfolding sym-def
proof (intro allI impI)
fix a b assume (a, b) ∈ A** then show (b, a) ∈ A*** unfolding conversion-def
proof (induct)
case base then show ?case by simp
next
case (step b c)
then have (c, b) ∈ A** by blast
from ⟨(c, b) ∈ A**⟩ and ⟨(b, a) ∈ (A**)⟩
show ?case by (rule converse-rtrancl-into-rtrancl)
qed
qed

lemma conversion-inv:
(x, y) ∈ R*** ↔ (y, x) ∈ R***
by (auto simp: conversion-def)
  (metis (full-types) rtrancl-converseD symcl-converse)
lemma conversion-converse [simp]:
\((A^{*\ast})^{-1} = A^{*\ast}\) by (metis conversion-sym sym-conv-converse-eq)

lemma conversion-rtrancl [simp]:
\((A^{*\ast})^* = A^{*\ast}\) by (metis conversion-def rtrancl-idemp)

lemma rtrancl-join-join:
assumes \((a, b) \in A^* \land (b, c) \in A\downarrow\) shows \((a, c) \in A\downarrow\)
proof
\[\begin{align*}
&\quad \text{from } (b, c) \in A\downarrow \text{ obtain } b' \text{ where } (b, b') \in A^* \land (b', c) \in (A^{-1})^* \\
&\quad \text{unfolding join-def by blast} \\
&\quad \text{with } (a, b) \in A^*, \text{ have } (a, b') \in A^* \text{ by simp} \\
&\quad \text{with } (b', c) \in (A^{-1})^* \text{ show } \text{thesis unfolding join-def by blast}
\end{align*}\]
qed

lemma join-rtrancl-join:
assumes \((a, b) \in A\downarrow \land (c, b) \in A^*\) shows \((a, c) \in A\downarrow\)
proof
\[\begin{align*}
&\quad \text{from } (c, b) \in A^* \text{ have } (b, c) \in (A^{-1})^* \text{ unfolding rtrancl-converse by simp} \\
&\quad \text{from } (a, b) \in A\downarrow \text{ obtain } a' \text{ where } (a, a') \in A^* \land (a', b) \in (A^{-1})^* \\
&\quad \text{unfolding join-def by best} \\
&\quad \text{with } (b, c) \in (A^{-1})^* \text{ have } (a', c) \in (A^{-1})^* \text{ by simp} \\
&\quad \text{with } (a, a') \in A^* \text{ show } \text{thesis unfolding join-def by blast}
\end{align*}\]
qed

lemma NF-I: \((\forall b. (a, b) \notin A) \implies a \in NF A\) 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
\[\begin{align*}
&\quad \text{from } \text{assms have } \forall b. (a, b) \notin A \text{ by auto} \\
&\quad \text{show } \text{thesis} \\
&\quad \text{proof (intro allI notI)}
\end{align*}\]

17
fix $b$ assume $(a, b) \in A^+$
then show False by (induct) (auto simp: $\forall b. \ (a, b) \notin A$)
qed
qed

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

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

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

lemma normalizability-I [simp]: $(a, b) \in A^* \Longrightarrow b \in \text{NF} A \Longrightarrow (a, b) \in A$
by (simp add: normalizability-def)

lemma normalizability-I [intro]: $(a, b) \in A^* \Longrightarrow (b, c) \in A \Longrightarrow (a, c) \in A$
by (auto simp add: normalizability-def)

lemma normalizability-E [simp]: $(a, b) \in A \Longrightarrow ((a, b) \in A^* \Longrightarrow b \in \text{NF} A \Longrightarrow P) \Longrightarrow P$
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 :: '$a \ rel \Rightarrow \ 'a \ set \Rightarrow \ bool$ where
$CR$-on $r A \leftarrow\rightarrow (\forall a \in A. \ \forall b. c. \ (a, b) \in r^* \land (a, c) \in r^* \rightarrow (b, c) \in \text{join} \ r)$

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

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

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

Alternative definition of SN.

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

definition UNF-on :: '$a \ rel \Rightarrow \ 'a \ set \Rightarrow \ bool$ where
$UNF$-on $r A \leftarrow\rightarrow (\forall a \in A. \ \forall b. c. \ (a, b) \in r^\prime \land (a, c) \in r^\prime \rightarrow b = c)$
abbreviation UNF :: 'a rel ⇒ bool where UNF r ≡ UNF-on r UNIV

definition WCR-on :: 'a rel ⇒ 'a set ⇒ bool where
  WCR-on r A ←→ (∀ a∈A. ∀ b. (a, b) ∈ r ∧ (a, c) ∈ r −→ (b, c) ∈ join r)

abbreviation WCR :: 'a rel ⇒ bool where WCR r ≡ WCR-on r UNIV

definition WN-on :: 'a rel ⇒ 'a set ⇒ bool where
  WN-on r A ←→ (∀ a∈A. ∃ b. (a, b) ∈ r)

abbreviation WN :: 'a rel ⇒ bool where
  WN r ≡ WN-on r UNIV

lemmas CR-defs = CR-on-def
lemmas SN-defs = SN-on-def
lemmas UNF-defs = UNF-on-def
lemmas WCR-defs = WCR-on-def
lemmas WN-defs = WN-on-def

definition complete-on :: 'a rel ⇒ 'a set ⇒ bool where
  complete-on r A ←→ SN-on r A ∧ CR-on r A

abbreviation complete :: 'a rel ⇒ bool where
  complete r ≡ complete-on r UNIV

definition semi-complete-on :: 'a rel ⇒ 'a set ⇒ bool where
  semi-complete-on r A ←→ WN-on r A ∧ CR-on r A

abbreviation semi-complete :: 'a rel ⇒ bool where
  semi-complete r ≡ semi-complete-on r UNIV

lemmas complete-defs = complete-on-def
lemmas semi-complete-defs = semi-complete-on-def

Unique normal forms with respect to conversion.

definition UNC :: 'a rel ⇒ bool where
  UNC A ←→ (∀ a b. a ∈ NF A ∧ b ∈ NF A ∧ (a, b) ∈ A*** −→ a = b)

lemma complete-onI:
  SN-on r A −→ CR-on r A −→ complete-on r A
  by (simp add: complete-defs)

lemma complete-onE:
  complete-on r A −→ (SN-on r A −→ CR-on r A −→ P) −→ P
  by (simp add: complete-defs)

lemma CR-onI:
  (∀ a b c. a ∈ A −→ (a, b) ∈ r∗ −→ (a, c) ∈ r∗ −→ (b, c) ∈ join r) −→ CR-on
lemma CR-on-singletonI:
\((\forall b \ c. \ (a, b) \in r^* \implies (a, c) \in r^* \implies (b, c) \in \text{join} \ r) \implies \text{CR-on} \ r \ \{a\}\) 
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:
\((\forall a \ b. \ a \in \text{NF} \ A \implies b \in \text{NF} \ A \implies (a, b) \in A^{**} \implies a = b) \implies \text{UNC} \ A\) 
by (simp add: UNC-def)

lemma UNC-E:
\[\text{UNC} \ A; \ a = b \implies P; \ a \notin \text{NF} \ A \implies P; \ b \notin \text{NF} \ A \implies P; \ (a, b) \notin A^{***} \implies P\] \implies P 
unfolding UNC-def by blast

lemma UNF-onI: \((\forall a \ b \ c. \ a \in A \implies (a, b) \in r^l \implies (a, c) \in r^l \implies b = c) \implies \text{UNF-on} \ r \ A\) 
by (simp add: UNF-defs)

lemma UNF-onE:
\[
UNF-on\ r\ A \Rightarrow a \in A \Rightarrow (b = c \Rightarrow P) \Rightarrow ((a, b) \not\in r' \Rightarrow P) \Rightarrow ((a, c) \\
\not\in r' \Rightarrow P) \Rightarrow P
\]

unfolding \textit{UNF-on-def} by blast

\textbf{lemma} \textit{UNF-onD}:
\[
UNF-on\ r\ A \Rightarrow a \in A \Rightarrow (a, b) \in r' \Rightarrow (a, c) \in r' \Rightarrow b = c
\]
by \textit{(blast elim: UNF-onE)}

\textbf{declare} \textit{UNF-onI} \texttt{[intro]}
\textbf{declare} \textit{UNF-onD} \texttt{[dest]}
\textbf{declare} \textit{UNF-onE} \texttt{[elim]}

\textbf{lemma} \textit{SN-onI}:
\[
\text{assumes } \forall f. [f 0 \in A; \text{ chain } r f] \Rightarrow False
\text{ shows } SN-on\ r\ A
\]
using \textit{assms unfolding \textit{SN-defs} by blast}

\textbf{lemma} \textit{SN-I}: \text{(\(\forall a. \text{SN-on } A \{a\}\))} \Rightarrow SN A
unfolding \textit{SN-on-def} by blast

\textbf{lemma} \textit{SN-on-trancl-imp-SN-on}:
\[
\text{assumes } \text{SN-on } (R^+) T \text{ shows } SN-on\ R\ T
\]
\text{proof} (\textit{rule ccontr})
\[
\text{assume } \neg \text{SN-on } R\ T
\text{ then obtain } s \text{ where } s 0 \in T \text{ and } \text{chain } R\ s \text{ unfolding } \textit{SN-defs} \text{ by } \textit{auto}
\text{ then have } \text{chain } (R^+) s \text{ by } \textit{auto}
\text{ with } (s 0 \in T); \text{ have } \neg \text{SN-on } (R^+) T \text{ unfolding } \textit{SN-defs} \text{ by } \textit{auto}
\text{ with } \textit{assms show } False \text{ by } \textit{simp}
\textbf{qed}

\textbf{lemma} \textit{SN-onE}:
\[
\text{assumes } \text{SN-on } r\ A
\text{ and } \neg (\exists f. f 0 \in A \wedge \text{chain } r f) \Rightarrow P
\text{ shows } P
\]
using \textit{assms unfolding \textit{SN-defs} by \textit{simp}}

\textbf{lemma} \textit{not-SN-onE}:
\[
\text{assumes } \neg \text{SN-on } r\ A
\text{ and } \forall f. [f 0 \in A; \text{ chain } r f] \Rightarrow P
\text{ shows } P
\]
using \textit{assms unfolding \textit{SN-defs} by \textit{simp}}

\textbf{declare} \textit{SN-onI} \texttt{[intro]}
\textbf{declare} \textit{SN-onE} \texttt{[elim]}
\textbf{declare} \textit{not-SN-onE} \texttt{[Pure.elim, elim]}

\textbf{lemma} \textit{refl-not-SN}:
\[
(x, x) \in R \Rightarrow \neg \text{SN } R
\]
unfolding \textit{SN-defs} by \textit{force}

21
lemma SN-on-irrefl:
assumes SN-on r A
shows ∀ a ∈ A. (a, a) /∈ r
proof (intro ballI notI)
  fix a assume a ∈ A and (a, a) ∈ r
  with assms show False unfolding SN-defs by auto
qed

lemma WCR-onI: (⋀ a b c. a ∈ A ⇒ (a, b) ∈ r ⇒ (a, c) ∈ r ⇒ (b, c) ∈ join r) ⇒ WCR-on r A
  by (simp add: WCR-defs)

lemma WCR-onE: WCR-on r A ⇒ a ∈ A ⇒ ((b, c) ∈ join r ⇒ P) ⇒ ((a, b) /∈ r ⇒ P) ⇒
((a, c) /∈ r ⇒ P) ⇒ P
  unfolding WCR-on-def by blast

lemma SN-nat-bounded: SN { (x, y :: nat). x < y ∧ y ≤ b } (is SN ?R)
proof
  fix f
  assume chain ?R f
  then have steps: ⋀ i. (f i, f (Suc i)) ∈ ?R ..
  { fix i
    have inc: f 0 + i ≤ f i
      proof (induct i)
        case 0 then show ?case by auto
        next
        case (Suc i)
        have f 0 + Suc i ≤ f i + Suc 0 using Suc by simp
        also have ... ≤ f (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: WCR-on r A ⇒ a ∈ A ⇒ (a, b) ∈ r ⇒ (a, c) ∈ r ⇒ (b, c) ∈ join r
  by (blast elim: WCR-onE)

lemma WN-onI: (⋀ a. a ∈ A ⇒ ∃ b. (a, b) ∈ r') ⇒ WN-on r A
  by (auto simp: WN-defs)

lemma WN-onE: WN-on r A ⇒ a ∈ A ⇒ (⋀ b. (a, b) ∈ r' ⇒ P) ⇒ P
  unfolding WN-defs by blast
lemma WN-onD: WN-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) | a b. (a, b) \in r \land SN-on s {a}}

lemma SN-restrict-SN-idemp [simp]: SN (restrict-SN A A)
by (auto simp: restrict-SN-def SN-defs)

lemma SN-on-Image:
assumes SN-on r A
shows SN-on r (r '' A)
proof
fix f
assume f 0 \in r '' A and chain: chain r f
then obtain a where a \in A and 1: (a, f 0) \in r by auto
let ?g = case-nat a f
from cons-chain [OF 1 chain] have chain r ?g .
moreover have ?g 0 \in A by (simp add: ⟨a \in A⟩)
ultimately have ¬ SN-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-on r B
shows SN-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-on r {a}
shows SN-on r {b}
using 1 and SN-on-Image [OF 2] and SN-on-subset2 [of {b} r '' {a}] by auto

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

lemma relpow-seq:
assumes \((x, y) \in r^\ast n\)
shows \(\exists f. f \, 0 = x \land f \, n = y \land (\forall i<n. (f \, i, f \, (\text{Suc} \, i)) \in r)\)
using assms
proof (induct \(n\) arbitrary: \(y\))
case 0 then show \(?case\) by auto
next
case (Suc \(n\))
then obtain \(z\) where \((x, z) \in r^\ast n\) and \((z, y) \in r\) by auto
obtain \(f\) where \(f \, 0 = x\) and \(f \, n = z\) and \(\forall i<n. (f \, i, f \, (\text{Suc} \, i)) \in r\)
using assms unfolded rtrancl-power [of \(x\) \(y\) - \(r\)] by blast

lemma rtrancl-imp-seq:
assumes \((x, y) \in r^\ast\)
shows \(\exists f \, n. f \, 0 = x \land f \, n = y \land (\forall i<n. (f \, i, f \, (\text{Suc} \, i)) \in r)\)
using assms [unfolded rtrancl-power] and relpow-seq [of \(x\) \(y\) - \(r\)] by blast

lemma SN-on-Image-rtrancl:
assumes \(\text{SN-on} \, r \, A\)
shows \(\text{SN-on} \, r \, (r^\ast \, '' \, A)\)
proof
fix \(f\)
assume \(f0: f \, 0 \in r^\ast \, '' \, A\) and chain: \(\text{chain} \, r \, f\)
then obtain \(a\) where \(a \in A\) and \((a, f \, 0) \in r^\ast\) by auto
then obtain \(n\) where \((a, f \, 0) \in r^\ast n\) unfolding rtrancl-power by auto
show False
proof (cases \(n\))
  case 0
  with \((a, f \, 0) \in r^\ast n\) 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 \(n\))
from relpow-seq [OF \((a, f \, 0) \in r^\ast n\)]
obtain \(g\) where \(g0: g \, 0 = a\) and \(g \, n = f \, 0\)
and gseq: \(\forall i<n. (g \, i, g \, (\text{Suc} \, i)) \in r\) by auto
let \(?if = \lambda i. \text{if} \, i < n\) then \(g \, i\) else \(f \, (i - n)\)
have chain \(?if\)
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 :Suc \ i > n 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: gseq) 
  from :Suc \ i = n 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 :Suc \ i = n 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 (\a, \f 0) \in r^\sim \n 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-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 

 25
show \( \text{?rhs} \subseteq \text{?lhs} \)

proof

fix \( a \) \( b \) assume \( (a, b) \in \text{?rhs} \)
then have \( (a, b) \in A^+ \) and \( \text{SN-on} \ A \{a\} \) unfolding \( \text{restrict-SN-def} \) by \( \text{auto} \)
then show \( (a, b) \in \text{?lhs} \)

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

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

lemma \( UNC-imp-UNF : \)
assumes \( UNC \ r \) shows \( UNF \ r \)
proof
\{ 
fix \( x \ y \ z \) assume \( (x, y) \in r^1 \) and \( (x, z) \in r^1 \)
then have \( (x, y) \in r^* \) and \( (x, z) \in r^* \) and \( y \in NF \ r \) and \( z \in NF \ r \) by \( \text{auto} \)
then have \( (x, y) \in r^{**} \) and \( (x, z) \in r^{**} \) by \( \text{auto} \)
then have \( (z, x) \in r^{**} \) using \( \text{conversion-sym} \) unfolding \( \text{sym-def} \) by \( \text{best} \)
with \( (x, y) \in r^{**} \) have \( (z, y) \in r^{**} \) using \( \text{conversion-trans} \) unfolding \( \text{trans-def} \) by \( \text{best} \)
from \( \text{assms} \) and this and \( (z \in NF \ r) \) and \( (y \in NF \ r) \) have \( z = y \) unfolding \( UNC-def \) by \( \text{auto} \)
\} then show \( \text{?thesis} \) by \( \text{auto} \)
qed

lemma \( \text{join-NF-imp-eq} : \)
assumes \( (x, y) \in r^1 \) and \( x \in NF \ r \) and \( y \in NF \ r \)
shows \( x = y \)
proof –
from \((x, y) \in r^\downarrow\) 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 \((x \in \text{NF} \ r)\) have \((x, z) \notin r^\uparrow\) using NF-no-trancl-step by best
then have \(x = z\) using rtranclD \[\text{OF} \ ((x, z) \in r^*)\] by auto
with \((x = z)\) show ?thesis by simp
qed

lemma rtrancl-Restr:
assumes \((x, y) \in (\text{Restr} \ r \ A)^*\)
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 \[\text{OF} \ assms\] by (auto simp: join-def rtrancl-converse)

lemma CR-iff-meet-subset-join: \(\text{CR} \ r = (r^\uparrow \subseteq r^\downarrow)\)
proof
assume \(\text{CR} \ r\) 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 \((\text{CR} \ r)\) 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 \((y, z) \in r^\downarrow\) unfolding meet-def rtrancl-converse by auto
with \((r^\uparrow \subseteq r^\downarrow)\) have \((y, z) \in r^\downarrow\) by auto
\}
then show \(\text{CR} \ r\) by (auto simp: CR-defs)
qed

lemma CR-divergence-imp-join:
assumes \(\text{CR} \ r\) 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 \((z, y) \in r^{\leftrightarrow}\) have \((y, z) \in r^{\leftrightarrow}\) using conversion-sym unfolding sym-def by best
with \((x, y) \in r^{**}\) show \((x, z) \in r^{**}\) using conversion-trans unfolding trans-def by best

qed

lemma meet-imp-conversion: \(r^1 \subseteq r^{**}\)
proof (rule subrelI)
  fix \(y\) \(z\) assume \((y, z) \in r^1\)
  then obtain \(x\) where \((x, y) \in r^*\) and \((x, z) \in r^*\) by auto
  then have \((x, y) \in r^{**}\) and \((x, z) \in r^{**}\) by auto
from \((x, y) \in r^{**}\) have \((y, x) \in r^{**}\) using conversion-sym unfolding sym-def by best
with \((x, z) \in r^{**}\) show \((y, z) \in r^{**}\) 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 CR \(r\) and \((x, y) \in r^1\)
    then obtain \(n\) where \((x, y) \in r^\hat{n}\)
of unfolding normalizability-def by auto
from assms and \((x, y) \in r^*\) and \((x, z) \in r^*\) have \((y, z) \in r^1\)
    by (rule CR-divergence-imp-join)
from this and \((y \in NF \ r)\) and \((z \in NF \ r)\) have \(y = z\) by (rule join-NF-imp-eq)

qed

lemma CR-iff-conversion-imp-join:
assumes CR \(r\) \((r^{**} \subseteq r^1)\)
proof (intro iffI subrelI)
  fix \(x\) \(y\) assume CR \(r\) and \((x, y) \in r^{**}\)
  then obtain \(n\) where \((x, y) \in (r^{**})^{-n}\)
of unfolding conversion-def rtrancl-is-UN-relpow by auto
by (rule CR-divergence-imp-join)
from this and \((y \in NF \ r)\) and \((z \in NF \ r)\) have \(y = z\) by (rule join-NF-imp-eq)

qed
then have \((z, x) \in r^*\) by \textbf{simp}
from \((z, y) \in r^\downarrow\) obtain \(z'\) where \((z, z') \in r^*\) and \((y, z') \in r^*\) by \textbf{auto}
from \((CR \ r)\) and \((z, x) \in r^\downarrow\) and \((z, z') \in r^*\) have \((x, z') \in r^\downarrow\)
   by \textbf{(rule CR-divergence-imp-join)}
then obtain \(x'\) where \((x, x') \in r^*\) and \((z', x') \in r^*\) by \textbf{auto}
with \((y, z') \in r^*\) show \(?\text{thesis}\) by \textbf{auto}
\textbf{qed}
\textbf{next}

assume \(r^{***} \subseteq r^\downarrow\) then show \(CR \ r\) \textbf{unfolding} \(CR\text{-iff-meet-subset-join}\)
   using \textbf{meet-imp-conversion} by \textbf{auto}
\textbf{qed}

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

\textbf{lemma} \(sym\text{-join}:: \text{sym} (join \ r)\) by \textbf{(auto simp: sym-def)}

\textbf{lemma} \(join\text{-sym}:: (s, t) \in A^\downarrow \Longrightarrow (t, s) \in A^\downarrow\) by \textbf{auto}

\textbf{lemma} \(CR\text{-join-left-1}::
\textbf{assumes} \(CR \ r\) and \((x, y) \in r^*\) and \((x, z) \in r^\downarrow\) \textbf{shows} \((y, z) \in r^\downarrow\)
\textbf{proof} –
\textbf{from} \((x, z) \in r^\downarrow\) obtain \(x'\) where \((x, x') \in r^*\) and \((z, x') \in r^\downarrow\) by \textbf{auto}
\textbf{from} \((CR \ r)\) and \((x, x') \in r^*\) and \((x, y) \in r^*\) have \((x, y) \in r^\downarrow\) by \textbf{auto}
\textbf{then have} \((y, x) \in r^\downarrow\) \textbf{using} \textbf{join-sym} by \textbf{best}
\textbf{from} \((CR \ r)\) \textbf{have} \(r^{***} = r^\downarrow\) by \textbf{(rule CR-imp-conversionIff-join)}
\textbf{from} \((y, x) \in r^\downarrow\) and \((x, z) \in r^\downarrow\) \textbf{show} \(?\text{thesis}\) \textbf{using} \textbf{conversion-trans unfolding} \textbf{trans-def} \(r^{***} = r^\downarrow\) \textbf{[symmetric]} by \textbf{best}
\textbf{qed}

\textbf{lemma} \(CR\text{-join-right-1}::
\textbf{assumes} \(CR \ r\) and \((x, y) \in r^\downarrow\) and \((y, z) \in r^*\) \textbf{shows} \((x, z) \in r^\downarrow\)
\textbf{proof} –
\textbf{have} \(r^{***} = r^\downarrow\) by \textbf{(rule CR-imp-conversionIff-join \ OF \ (CR \ r))}
\textbf{from} \((y, z) \in r^\downarrow\) \textbf{have} \((y, z) \in r^{***}\) by \textbf{auto}
\textbf{with} \((x, y) \in r^\downarrow\) \textbf{show} \(?\text{thesis}\) \textbf{unfolding} \(r^{***} = r^\downarrow\) \textbf{[symmetric]} \textbf{using} \textbf{conversion-trans unfolding} \textbf{trans-def} by \textbf{fast}
\textbf{qed}

\textbf{lemma} \(NF\text{-not-suc}::
\textbf{assumes} \((x, y) \in r^*\) and \(x \in NF \ r\) \textbf{shows} \(x = y\)
\textbf{proof} –

29
from \( x \in NF r \) have \( \forall y. (x, y) \notin r \) using NF-no-step by auto
then have \( x \notin Domain r \) using Domain-unfold by simp
from \( (x, y) \in r^* \) show \( ?thesis \) unfolding Not-Domain-rtrancl [OF \( x \notin Domain r \)] by simp
qed

lemma semi-complete-imp-conversionIff-same-NF:
assumes semi-complete r
shows \( ((x, y) \in r^{***}) = (\forall u v. (x, u) \in r^1 \land (y, v) \in r^1 \rightarrow u = v) \)
proof –
from \( \text{assms} \) have \( WN r \) and \( CR r \) unfolding semi-complete-defs by auto
then have \( r^{***} = r^1 \) using CR-imp-conversionIff-join by auto
show \( ?thesis \)
proof
assume \( (x, y) \in r^{***} \)
from \( (x, y) \in r^{***} \) have \( (x, y) \in r^1 \) unfolding \( r^{***} = r^1 \) .
show \( \forall u v. (x, u) \in r^1 \land (y, v) \in r^1 \rightarrow u = v \)
proof (intro all_impl, elim conjE)
fix \( u v \) assume \( (x, u) \in r^1 \) and \( (y, v) \in r^1 \)
then have \( (x, u) \in r^* \) and \( (y, v) \in r^* \) and \( u \in NF r \) and \( v \in NF r \) by auto

from \( CR r \) and \( (x, u) \in r^* \) and \( (x, y) \in r^1 \) have \( (u, y) \in r^1 \)
by (auto intro: CR-join-left-I)
then have \( (y, u) \in r^1 \) using join-sym by best
with \( (x, y) \in r^1 \) have \( (x, u) \in r^1 \) unfolding \( r^{***} = r^1 \) [symmetric]
using conversion-trans unfolding trans-def by best
from \( CR r \) and \( (x, y) \in r^1 \) and \( (y, v) \in r^1 \) have \( (x, v) \in r^1 \)
by (auto intro: CR-join-right-I)
then have \( (v, x) \in r^1 \) using join-sym unfolding sym-def by best
with \( (x, u) \in r^1 \) have \( (v, u) \in r^1 \) unfolding \( r^{***} = r^1 \) [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 \( (u, v') \in r^* \) and \( u \in NF r \) have \( u = v' \) by (rule NF-not-suc)
from \( (v, v') \in r^* \) and \( v \in NF r \) have \( v = v' \) by (rule NF-not-suc)
then show \( u = v \) unfolding \( u = v' \) by simp
qed
next
assume equal-NF: \( \forall u v. (x, u) \in r^1 \land (y, v) \in r^1 \rightarrow u = v \)
from \( WN r \) obtain \( u \) where \( (x, u) \in r^1 \) by auto
from \( WN r \) obtain \( v \) where \( (y, v) \in r^1 \) by auto
from \( (x, u) \in r^1 \) and \( (y, v) \in r^1 \) have \( u = v \) using equal-NF by simp
from \( (x, u) \in r^1 \) and \( (y, v) \in r^1 \) have \( (x, v) \in r^* \) and \( (y, v) \in r^* \)
unfolding \( u = v \) by auto
then have \( (x, v) \in r^{***} \) and \( (y, v) \in r^{***} \) by auto
from \( (y, v) \in r^{***} \) have \( (v, y) \in r^{***} \) using conversion-sym unfolding sym-def by best
with \( (x, v) \in r^{***} \) show \( (x, y) \in r^{***} \) using conversion-trans unfolding trans-def by best
qed

30
proof (diamond-imp-semi-confluence)

lemma CR-imp-UNC:

assumes CR r shows UNC r

proof - { fix x y assume x ∈ NF r and y ∈ NF r and (x, y) ∈ r∗∗∗ have r∗∗∗ = r by (rule CR-imp-conversionIff-join [OF assms]) from (x, y) ∈ r∗∗∗ have (x, y) ∈ r† unfolding (r∗∗∗ = r†) by simp 
then obtain x′ where (x, x′) ∈ r* and (y, x′) ∈ r* by best 
then have x = y unfolding (x = x′ by (rule UNF-imp-CR)) 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) ∈ r* and (x, z) ∈ r* from assms obtain y′ where (y, y′) ∈ r† unfolding WN-defs by best with (x, y) ∈ r† have (x, y′) ∈ r† by auto 
then have y′ = z using (UNF r) unfolding UNF-defs by auto 
then show ?thesis by (auto simp: UNF-defs) 
qed

definition diamond :: 'a rel ⇒ bool (◊) where ◊ r =⇒ (r−1 O r) ⊆ (r O r−1)

lemma diamond-I [intro]: (r−1 O r) ⊆ (r O r−1) ⇒ ◊ r unfolding diamond-def by simp

lemma diamond-E [elim]: ◊ r =⇒ ((r−1 O r) ⊆ (r O r−1) ⇒ P) ⇒ P unfolding diamond-def by simp

lemma diamond-imp-semi-confluence:

assumes ◊ r shows (r−1 O r*) ⊆ r†

proof (rule subrell)

fix y z assume (y, z) ∈ r−1 O r* 
then obtain x where (x, y) ∈ r and (x, z) ∈ r* by best 
then obtain n where (x, z) ∈ r−"n using rtrancl-imp-UN-relpow by best 
with (x, y) ∈ r show (y, z) ∈ r† by best 
proof (induct n arbitrary: x z y)

case 0 then show ?case by auto 
next 

case (Suc n) 

31
from $(x, z) \in r^{-}\text{Suc } n$ obtain $x'$ where $(x, x') \in r$ and $(x', z) \in r^{-}n$
using relpow-Suc-D2 by best
with $(x, y) \in r$ have $(y, x') \in (r^{-1} \text{ O } r)$ by auto
with $(\mathcal{O} r)$ 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 $(x', z) \in r^{-}n$ have $(y', z) \in r^{+}$ by auto
with $(y, y') \in r$ show ?thesis by (auto intro: rtrancl-join-join)
qed

lemma semi-confluence-imp-CR:
assumes $(r^{-1} \text{ O } r^{*}) \subseteq r^{+}$ shows CR $r$
proof \{ 
fix $x \ y \ z$ assume $(x, y) \in r^{*}$ and $(x, z) \in r^{*}$
then obtain $n$ where $(x, z) \in r^{-}n$ using rtrancl-imp-UN-relpow by best
with $(x, y) \in r^{*}$ have $(y, z) \in r^{+}$
proof (induct $n$ arbitrary: $x \ y \ z$)
case 0 then show ?case by auto
next
case $(\text{Suc } n)$
from $(x, z) \in r^{-}\text{Suc } n$ obtain $x'$ where $(x, x') \in r$ and $(x', z) \in r^{-}n$
using relpow-Suc-D2 by best
from $(x, x') \in r$ and $(x, y) \in r^{*}$ have $(x', y) \in (r^{-1} \text{ O } r^{*})$ by auto
with assms have $(x', y) \in r^{+}$ by auto
then obtain $y'$ where $(x', y') \in r^{*}$ and $(y, y') \in r^{*}$ by best
with Suc and $(x', z) \in r^{-}n$ have $(y', z) \in r^{+}$ by simp
then obtain $u$ where $(z, u) \in r^{*}$ and $(y', u) \in r^{*}$ by best
from $(y, y') \in r^{*}$ and $(y', u) \in r^{*}$ have $(y, u) \in r^{+}$ by auto
with $(z, u) \in r^{*}$ show ?case by best
qed

lemma diamond-imp-CR:
assumes $\mathcal{O} \ r$ shows CR $r$
using assms by (rule diamond-imp-semi-confluence [THEN semi-confluence-imp-CR])

lemma diamond-imp-CR':
assumes $\mathcal{O} \ s$ and $r \subseteq s$ and $s \subseteq r^{*}$ shows CR $r$
unfolding CR-iff-meet-subset-join
proof \-
from $\mathcal{O} \ s$ have CR $s$ by (rule diamond-imp-CR)
then have $s^{+} \subseteq s^{+}$ unfolding CR-iff-meet-subset-join by simp
from $(r \subseteq s)$ have $r^{*} \subseteq s^{*}$ by (rule rtrancl-mono)
from $(s \subseteq r^{*})$ have $s^{*} \subseteq (r^{*})^{*}$ by (rule rtrancl-mono)
then have $s^{*} \subseteq r^{*}$ by simp
with $(r^{*} \subseteq s^{*})$ have $r^{*} = s^{*}$ by simp
show $r^{+} \subseteq r^{+}$ unfolding meet-def join-def rtrancl-converse $(r^{*} = s^{*})$
unfolding rtrancl-converse [symmetric] meet-def [symmetric]

32
lemma SN-imp-minimal:
assumes SN A
shows \( \forall x. x \in Q \rightarrow (\exists y. (z, y) \in A \rightarrow y \notin Q) \)
proof (rule ccontr)
assume \( \forall x. x \in Q \rightarrow (\exists y. (z, y) \in A \rightarrow y \notin Q) \)
then obtain Q x where x \in Q and \( \forall z \in Q. \exists y. ((z, y) \in A \land y \in Q) \) by auto
then have \( \forall z. \exists y. z \in Q \rightarrow (z, y) \in A \land y \in Q \) by auto
then have \( \exists f. \forall x. x \in Q \rightarrow (x, f x) \in A \land f x \in Q \) by (rule choice)
then obtain f where a:\( \forall x. x \in Q \rightarrow (x, f x) \in A \land f x \in Q \) (is \( \forall x. ?P x \))
by best
let \( ?S = \lambda i. (f ^^ i) x \)
have \( ?S 0 = x \) by simp
have \( \forall i. (\forall i. ?S i, ?S (Suc i)) \in A \land ?S (Suc i) \in Q \)
proof
fix i show \( (\forall i. ?S i, ?S (Suc i)) \in A \land ?S (Suc i) \in Q \)
by (induct i) (auto simp: \( \forall x. ?S x \))
qed
with \( ?S 0 = x \) have \( \exists S. S 0 = x \land \text{chain} A S \) by fast
with assms show False by auto
qed

lemma SN-on-imp-on-minimal:
assumes SN-on r \{x\}
shows \( \forall Q. x \in Q \rightarrow (\exists y. (z, y) \in r \rightarrow y \notin Q) \)
proof (rule ccontr)
assume \( \forall Q. x \in Q \rightarrow (\exists y. (z, y) \in r \rightarrow y \notin Q) \)
then obtain Q where x \in Q and \( \forall z \in Q. \exists y. ((z, y) \in r \land y \in Q) \) by auto
then have \( \forall z. \exists y. z \in Q \rightarrow (z, y) \in r \land y \in Q \) by auto
then have \( \exists f. \forall x. x \in Q \rightarrow (x, f x) \in r \land f x \in Q \) by (rule choice)
then obtain f where a:\( \forall x. x \in Q \rightarrow (x, f x) \in r \land f x \in Q \) (is \( \forall x. ?P x \))
by best
let \( ?S = \lambda i. (f ^^ i) x \)
have \( ?S 0 = x \) by simp
have \( \forall i. (\forall i. ?S i, ?S(Suc i)) \in r \land ?S(Suc i) \in Q \)
proof
fix i show \( (\forall i. ?S i, ?S(Suc i)) \in r \land ?S(Suc i) \in Q \) by (induct i) (auto simp: \( \forall x. ?S x \))
qed
with \( ?S 0 = x \) have \( \exists S. S 0 = x \land \text{chain} r S \) by fast
with assms show False by auto
qed

lemma minimal-imp-uf:
assumes \( \forall Q. x \in Q \rightarrow (\exists y. (z, y) \in r \rightarrow y \notin Q) \)
shows \( uf(r^{-1}) \)
proof (rule ccontr)
assume \( \neg \text{wf}(r^{-1}) \)
then have \( \exists P. (\forall x. (\forall y. (x, y) \in r \implies P y) \implies P x) \land (\exists x. \neg P x) \)
unfolding \( \text{wf-def} \) by \( \text{simp} \)
then obtain \( P x \) where \( \text{suc} : \forall x. (\forall y. (x, y) \in r \implies P y) \implies P x \) and \( \neg P x \)
by \( \text{auto} \)
let \( ?Q = \{ x. \neg P x \} \)
from \( \langle \text{wf} (r^{-1}) \rangle \) have \( x \in ?Q \) by \( \text{simp} \)
then obtain \( z \) where \( z \in ?Q \) and \( \neg P z \)
by \( \text{best} \)
from \( \langle z \in ?Q \rangle \) have \( \neg P z \) by \( \text{simp} \)
with \( \langle \text{suc} \rangle \) obtain \( y \) where \( (z, y) \in r \) and \( \neg P y \)
by \( \text{best} \)
then have \( y \in ?Q \) by \( \text{simp} \)
with \( \langle (z, y) \in r \rangle \) and \( \text{min} \) show \( \text{False} \) by \( \text{simp} \)
qed

lemmas \( \text{SN-imp-wf} = \text{SN-imp-minimal} \) [\( \text{THEN} \) \( \text{minimal-imp-wf} \)]

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

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

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

lemma SN-induct:
assumes sn: SN $r$ and step: $\forall a. (\forall b. (a, b) \in r \Longrightarrow P b) \Longrightarrow P a$
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-on $R$ $A$
  and $s \in A$
  and imp: $\forall t. (\forall u. (t, u) \in R \Longrightarrow P u) \Longrightarrow P t$
shows $P s$
proof
  let $?R = \text{restrict-SN } R R$
  let $?P = \lambda t. \text{SN-on } R \{t\} \Longrightarrow P t$
  have SN-on $R \{s\} \Longrightarrow P s$
proof (rule SN-induct [OF SN-restrict-SN-idemp [of $R$], of $?P$])
    fix $a$
    assume ind: $\forall b. (a, b) \in ?R \Longrightarrow \text{SN-on } R \{b\} \Longrightarrow P b$
    show $\text{SN-on } R \{a\} \Longrightarrow P a$
proof
      assume SN: SN-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
with SN show $P s$ using $(s \in A)$ unfolding SN-on-def by blast
qed

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

35
fix \( x \) assume \( x \in A \)
from assms \( \text{[OF this]} \)
have \( \text{SN-on} \ \{(y, z). \ g \ z \ y \} \ \{x\} \)
proof (induct rule: accp.induct)
  case (accI \( x \))
  show \( \text{?case} \)
  proof
    fix \( f \)
    assume \( x \): \( f \ 0 \in \{x\} \) and \( \text{steps}: \forall \ i. \ (f \ i, f \ (\text{Suc} \ i)) \in \{a. \ (\lambda(y, z). \ g \ z \ y) \ a\} \)
    then have \( g \ (f \ 1) \ x \) by auto
    from accI \( \text{[2]} \text{[OF this]} \) \( \text{steps} \ x \) show \( \text{False} \) unfolding \( \text{SN-on-def} \) by auto
  qed
  qed
} then show \( \text{?thesis} \) unfolding \( \text{SN-on-def} \) by blast
qed

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

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

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

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

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

2.3 Newman’s Lemma

lemma \( \text{rtrancl-len-E [elim]} \):
assumes \( (x, y) \in r^{*} \) obtains \( n \) where \( (x, y) \in r^{-^n} \)
using \( \text{rtrancl-imp-UN-relpow [OF assms]} \) by \( \text{best} \)

lemma \( \text{relpow-Suc-E2' [elim]} \):
assumes \( (x, z) \in A^{-^\text{Suc} n} \) obtains \( y \) where \( (x, y) \in A \) and \( (y, z) \in A^{*} \)
proof –
assume \( \text{assm}: \bigwedge y. (x, y) \in A \implies (y, z) \in A^{*} \implies \text{thesis} \)
from \( \text{relpow-Suc-E2 [OF assms]} \) obtain \( y \) where \( (x, y) \in A \) and \( (y, z) \in A^{-^n} \)
by \( \text{auto} \)
then have \( (y, z) \in A^{*} \) using \( \text{relpow-imp-rtrancl by auto} \)
from \( \text{assm [OF } (x, y) \in A \text{ this]} \) show \( \text{thesis .} \)
qed

lemmas \( \text{SN-on-induct} \)' [\( \text{consumes 1, case-names IH} \) = \( \text{SN-on-induct} \) [\( \text{OF - singletonI} \)]]

lemma \( \text{Newman-local:} \)
assumes \( \text{SN-on } r \ X \) and \( \text{WCR: } \text{WCR-on } r \ \{ x. \ \text{SN-on } r \ \{ x \} \} \)
shows \( \text{CR-on } r \ X \)
proof –
\{ 
fix \( x \)
assume \( x \in X \)
with \( \text{assms have} \ \text{SN-on } r \ \{ x \} \) unfolding \( \text{SN-on-def} \) by \( \text{auto} \)
with \( \text{this have} \ \text{CR-on } r \ \{ x \} \)
proof (induct rule: \( \text{SN-on-induct}' \))
case (IH \( x \)) show \( ?\text{case} \)
proof
fix \( y \) \( z \)
assume \( (x, y) \in r^{*} \) and \( (x, z) \in r^{*} \)
from \( (x, y) \in r^{*} \) obtain \( m \) where \( (x, y) \in r^{-^m} \) ..
from \( (x, z) \in r^{*} \) obtain \( n \) where \( (x, z) \in r^{-^n} \) ..
show \( (y, z) \in r^{+} \)
proof (cases \( n \))
case 0
from \( (x, z) \in r^{-^n} \) have \( eq: x = z \) by \( \text{(simp add: 0)} \)
from \( (x, y) \in r^{*} \) show \( ?\text{thesis unfolding eq .} \)
next
case (Suc \( n \'))
from \( (x, z) \in r^{-^n} \) [\( \text{unfolded Suc} \) obtain \( t \) where \( (x, t) \in r \) and \( (t, z) \in r^{*} \)
show \( ?\text{thesis} \)
proof (cases \( m \))
case 0
from \( (x, y) \in r^{-^m} \) have \( eq: x = y \) by \( \text{(simp add: 0)} \)

37
from \((x, z) \in r^*\) show \(^?\)thesis unfolding eq ..

next

\[\begin{aligned}
&\text{case } (\text{Suc } m) \\
&\text{from } \langle x, y \rangle \in r^* m \ [\text{unfolded Suc}] \text{ obtain } s \text{ where } (x, s) \in r \text{ and } (s, y) \in r^* ..
\end{aligned}\]

from \(WCR\ IH(2)\) have \(WCR\ r \{x\} \) unfolding \(WCR\text{-on-def} \) by auto

with \(\langle x, s \rangle \in r \) and \(\langle x, t \rangle \in r \) have \(\langle s, t \rangle \in r^i \) by auto

then obtain \(u\) where \(\langle s, u \rangle \in r^* \) and \(\langle t, u \rangle \in r^* ..

from \(\langle x, s \rangle \in r \) \(IH(2)\) have \(SN\text{-on } r \{s\} \) by \((\text{rule step-preserves-SN-on})

from \(IH(1)\) \([OF \ (\langle x, s \rangle \in r \ this\] have \(CR\text{-on } r \{s\} \).

from this and \(\langle s, u \rangle \in r^* \) and \(\langle s, y \rangle \in r^* \) have \(\langle u, y \rangle \in r^i \) by auto

then obtain \(v\) where \(\langle u, v \rangle \in r^* \) and \(\langle y, v \rangle \in r^* ..

from \(\langle x, t \rangle \in r \) \(IH(2)\) have \(SN\text{-on } r \{t\} \) by \((\text{rule step-preserves-SN-on})

from \(IH(1)\) \([OF \ (\langle x, t \rangle \in r \ this\] have \(CR\text{-on } r \{t\} \).

moreover from \(\langle t, u \rangle \in r^* \) and \(\langle u, v \rangle \in r^* \) have \(\langle t, v \rangle \in r^i \) by auto

ultimately have \(\langle z, v \rangle \in r^i \) using \(\langle t, z \rangle \in r^* \) by auto

then obtain \(w\) where \(\langle z, w \rangle \in r^* \) and \(\langle v, w \rangle \in r^* ..

from \(\langle y, v \rangle \in r^* \) and \(\langle v, w \rangle \in r^* \) have \(\langle y, w \rangle \in r^* \) by auto

with \(\langle z, w \rangle \in r^* \) show \(^?\)thesis by auto

qed

\textbf{lemma} Newman: \(SN\ r \implies WCR\ r \implies CR\ r\)

using Newman-local \([of\ r\ \text{UNIV}]\)

unfolding \(WCR\text{-on-def} \) by auto

\textbf{lemma} Image-SN-on:

assumes \(SN\text{-on } r \ (r`` A)\)

shows \(SN\text{-on } r\ A\)

proof

fix \(f\)

assume \(f\ 0 \in A\) and chain: \(\text{chain } r\ f\)

then have \(f\ (\text{Suc } 0) \in r`` A\) by auto

with \(\text{assms} \) have \(SN\text{-on } r\ \{f\ (\text{Suc } 0)\} \) by \((\text{auto simp add: } f\ 0 \in A ; \text{SN-defs})\)

moreover have \(\neg\ SN\text{-on } r\ \{f\ (\text{Suc } 0)\} \)

proof

have \(f\ (\text{Suc } 0) \in \{f\ (\text{Suc } 0)\} \) by simp

moreover from chain have \(\text{chain } r\ (f \circ \text{Suc})\) by auto

ultimately show \(^?\)thesis by auto

qed

ultimately show \(\text{False}\) by simp

qed

\textbf{lemma} SN-on-Image-conv: \(SN\text{-on } r \ (r`` A) = SN\text{-on } r\ A\)

38
If all successors are terminating, then the current element is also terminating.

**Lemma step-reflects-SN-on:**

- **Assumes** \( (\forall b. (a, b) \in r \implies SN r \{b\}) \)
- **Shows** \( SN r \{a\} \)
- **Using** \( assms \) and \( Image-SN-on \) of \( r \{a\} \) by \( (auto \ simp: SN-defs) \)

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

- **SN-on** \( r \{a\} = (\forall b. (a, b) \in r \implies SN r \{b\}) \)
- **Using** \( SN-on-Image-conv \) of \( r \{a\} \) by \( (auto \ simp: SN-defs) \)

**Lemma SN-imp-SN-trancl:**

- **Assumes** \( SN (R^+) \)
- **Shows** \( SN R \)
- **Using** \( assms \) by \( (rule SN-on-trancl-imp-SN-on) \)

**Lemma SN-trancl-SN-conv:**

- **SN** \( (R^+) = SN R \)
- **Using** \( SN-trancl-imp-SN \) of \( R \) by \( blast \)

**Lemma SN-inv-image:**

- **SN** \( (inv-image R f) \)
- **Unfolding** \( SN-defs \) by \( simp \)

**Lemma SN-subset:**

- **SN** \( R = \implies R' \subseteq R \implies SN R' \)
- **Unfolding** \( SN-defs \) by \( blast \)

**Lemma SN-pow-imp-SN:**

- **Assumes** \( SN (A^{Suc n}) \)
- **Shows** \( SN A \)
- **Proof** \( (rule ccontr) \)
  - **Assume** \( SN A \)
  - **Then obtain** \( S \) where \( chain A S \)
  - **Unfolding** \( SN-defs \) by \( auto \)
  - **From** \( chain-imp-relpow \) of \( this \)
    - **Have** \( step: \bigwedge i. (S i, S (i + (Suc n))) \in A^{Suc n} \).
  - **Let** \( ?T = \lambda i. S (i * (Suc n)) \)
  - **Have** \( chain (A^{Suc n}) ?T \)
  - **Proof**
    - **Fix** \( i \) show \( (?T i, ?T (Suc i)) \in A^{Suc n} \)
    - **Unfolding** \( mult-Suc \)
      - **Using** \( step \) of \( i * (Suc n) \) by \( (simp \ only: add.commute) \)
  - **Qed**
  - **Then have** \( SN (A^{Suc n}) \)
  - **Unfolding** \( SN-defs \) by \( fast \)
  - **With** \( assms \) show \( False \) by \( simp \)
  - **Qed**

**Lemma pow-Suc-subset-trancl:**

- **Pow** \( R^{Suc n} \subseteq R^+ \)
- **Using** \( trancl-power \) of \( R \) by \( blast \)
lemma SN-imp-SN-pow:
assumes SN R shows SN (R "^\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 \iff SN (R \^\Suc n)
by (rule iffI, rule SN-imp-SN-pow, assumption, rule SN-pow-imp-SN)

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 \(\neg SN-on (r\^+) A\)
  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 \(\forall i. SN-on r \{ f i \}\)
  using steps-preserve-SN-on [of f 0 - r]
  and (f 0 \in A)
  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-on (\(?r\^+) \{ f 0 \}\) by auto
  with \(\neg SN (\(?r\^+))\) have False by (simp add: SN-defs)
  then show \(\neg SN-on r A\) by simp
  qed

lemma SN-on-trancl-SN-on-cone: SN-on (R\^+) T = SN-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 \land y \in S \land (x, y) \in r\}
lemma \(SN\text{-}on\text{-}restrict\): 
assumes \(SN\text{-}on\ r\ A\)
shows \(SN\text{-}on\ (\text{restrict } r\ S)\ A\) (is \(SN\text{-}on\ ?r\ A\))
proof (rule ccontr)
assume \(\neg\ SN\text{-}on\ ?r\ A\)
then have \(\exists f.\ f\ 0 \in A \land \text{chain } ?r\ f\) by auto
then have \(\exists f.\ f\ 0 \in A \land \text{chain } r\ f\) unfolding restrict-def by auto
with \(\langle SN\text{-}on\ r\ A\rangle\) show False by auto
qed

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

lemma \(rtrancl\text{-}Image\text{-}step\): 
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 \(c \in A\) show \(?thesis\) by auto
qed

lemma \(WCR\text{-}SN\text{-}on\text{-}imp\text{-}CR\text{-}on\): 
assumes \(WCR\ r\) and \(SN\text{-}on\ r\ A\)
shows \(CR\text{-}on\ r\ A\)
proof – 
let \(?S\) = \(r\)\(^*\) \(\{" A\)
let \(?r\) = \(\text{restrict } r\ ?S\)
have \(\forall x.\ SN\text{-}on\ ?r\ \{x\}\)
proof 
fix \(y\) have \(y \notin ?S\ \lor\ y \in ?S\) by simp
then show \(SN\text{-}on\ ?r\ \{y\}\)
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 \(SN\text{-}on\text{-}Image\text{-}rtrancl\) [OF \(\langle SN\text{-}on\ r\ A\rangle\)]
have \(SN\text{-}on\ r\ \{y\}\) using \(SN\text{-}on\text{-}subset2\) [of \(\{y\}\) \(r\)\(^*\) \(\{" A\) by blast
then show \(?thesis\) by (rule \(SN\text{-}on\text{-}restrict\))
qed
qed
then have \(SN\ ?r\) unfolding \(SN\text{-}defs\) by auto
\{ 
fix \(x\ y\) assume \((x, y) \in r\)\(^*\) and \(x \in ?S\) and \(y \in ?S\)

41
then obtain \( n \) where \((x, y) \in r^{-n}\) and \(x \in \mathcal{S}^n \) and \(y \in \mathcal{S}^n\)
using `rtrancl-imp-UN-relpow` by `best`
then have \((x, y) \in \hat{r}^*\)
proof (induct \( n \) arbitrary: \( x y \))
  case 0 then show \(?case by simp\)
next
case (Suc \( n \))
  from \((x, y) \in \hat{r}^{-n}\) obtain \( x' \) where \((x, x') \in \hat{r}^{-n}\)
  using `relpow-Suc-D2` by `best`
  then have \((x, x') \in \hat{r}^*\) by `simp`
with \((x \in \mathcal{S})\) have \( x' \in \mathcal{S} \) by (rule `rtrancl-Image-step`)
with Suc and \((x', y) \in \hat{r}^{-n}\) have \((x', y) \in \hat{r}^*\) by `simp`
from \((x, x') \in \hat{r}^{-n}\) and \((x \in \mathcal{S})\) and \((x' \in \mathcal{S})\) have \((x, x') \in \hat{r}^*\)
unfolding `restrict-def` by `simp`
with \((x', y) \in \hat{r}^*\) show \(?case by simp\)
qed

\{ \}
then have \(a: \forall x\ y. \ (x, y) \in \hat{r}^* \land x \in \mathcal{S} \land y \in \mathcal{S} \rightarrow (x, y) \in \hat{r}^*\) by `simp`
\{ \}
  fix \( x'\ y\ z \) assume \((x', y) \in \hat{r}^*\) and \((x', z) \in \hat{r}^*\)
  then have \( x' \in \mathcal{S} \) and \( y \in \mathcal{S} \) and \( z \in \mathcal{S} \) and \((x', y) \in \hat{r}^*\) and \((x', z) \in \hat{r}^*\)
  unfolding `restrict-def` by `auto`
  with \(\langle WCR\ \hat{r}^*\rangle\) have \((y, z) \in \hat{r}^*\) by `auto`
  then obtain \( u \) where \((y, u) \in \hat{r}^*\) and \((z, u) \in \hat{r}^*\) by `auto`
  from \(\langle x' \in \mathcal{S}\rangle\) obtain \( x \) where \( x \in A\) and \((x, x') \in \hat{r}^*\) by `auto`
  from \(\langle x', y \rangle \in \hat{r}^*\) have \((x', y) \in \hat{r}^*\) by `auto`
  with \(\langle y, u \rangle \in \hat{r}^*\) have \((x', u) \in \hat{r}^*\) by `auto`
  with \(\langle x, x' \rangle \in \hat{r}^*\) have \((x, u) \in \hat{r}^*\) by `simp`
  then have \( u \in \mathcal{S} \) using \( x \in A\) by `auto`
  from \(\langle y \in \mathcal{S}\rangle\) and \(\langle u \in \mathcal{S}\rangle\) and \(\langle (y, u) \in \hat{r}^*\rangle\) have \((y, u) \in \hat{r}^*\) using \( a\) by `auto`
  from \(\langle z \in \mathcal{S}\rangle\) and \(\langle u \in \mathcal{S}\rangle\) and \(\langle (z, u) \in \hat{r}^*\rangle\) have \((z, u) \in \hat{r}^*\) using \( a\) by `auto`
  with \(\langle (y, u) \in \hat{r}^*\rangle\) have \((y, z) \in \hat{r}^*\) by `auto`
\} then have \( WCR\ \hat{r}^*\) by `auto`
  have \( \hat{r}\) using Newman [OF \( \langle SN\ \hat{r}\rangle\) \( \langle WCR\ \hat{r}\rangle\)] by `simp`
  \{ \}
  fix \( x\ y\ z \) assume \( x \in A\) and \((x, y) \in \hat{r}^*\) and \((x, z) \in \hat{r}^*\)
  then have \( y \in \mathcal{S} \) and \( z \in \mathcal{S} \) by `auto`
  have \( x \in \mathcal{S} \) using \( x \in A\) by `auto`
  from \( a\) and \(\langle (x, y) \in \hat{r}^*\rangle\) and \( x \in \mathcal{S} \) and \( y \in \mathcal{S} \) have \((x, y) \in \hat{r}^*\) by `simp`
  from \( a\) and \(\langle (x, z) \in \hat{r}^*\rangle\) and \( x \in \mathcal{S} \) and \( z \in \mathcal{S} \) have \((x, z) \in \hat{r}^*\) by `simp`
  with \(\langle CR\ \hat{r}\rangle\) and \(\langle (x, y) \in \hat{r}^*\rangle\) have \((y, z) \in \hat{r}^*\) by `auto`
  then obtain \( u\) where \((y, u) \in \hat{r}^*\) and \((z, u) \in \hat{r}^*\) by `best`
  then have \(\langle y, u \rangle \in \hat{r}^*\) and \(\langle z, u \rangle \in \hat{r}^*\) using `restrict-rtrancl` by `auto`
  then have \(\langle y, z \rangle \in \hat{r}^*\) by `auto`
} then show \(?\)thesis by auto
qed

lemma \(\text{SN-on-Image-normalizable}\):
  assumes \(\text{SN-on} \ r \ A\)
  shows \(\forall a \in A. \ \exists b. \ b \in r^\leftarrow \ A\)
proof
  fix \(a\) assume \(a \in A\)
  show \(\exists b. \ b \in r^\leftarrow \ A\)
  proof (rule ccontr)
    assume \(\neg (\exists b. \ b \in r^\leftarrow \ A)\)
    then have \(A \colon \forall b. \ (a, b) \in r^* \rightarrow b \notin NF \ r\) using \(a\) by auto
    then have \(a \notin NF \ r\) by auto
    let \(\{c \colon (a, c) \in r^* \rightarrow c \notin NF \ r\}\)
    have \(\exists c \in \{Q \colon (a, c) \in r \land b \in \{Q\}\}\)
    proof
      fix \(c\)
      assume \(c \in \{Q\}\)
      then have \((a, c) \in r^* \land c \notin NF \ r\) by auto
      then obtain \(d\) where \((a, d) \in r^*\) by simp
      with \(A\) have \(d \notin NF \ r\) by simp
      with \((c, d) \in r\) and \((a, c) \in r^*\)
      show \(\exists b. \ (c, b) \in r \land b \in \{Q\}\) by auto
    qed
    with \(\neg (\forall c \in \{Q\}. \ a \in \{Q\} \land (\exists b. \ (c, b) \in r \land b \in \{Q\})\) by auto
    then have \(\exists Q \colon a \in \{Q\} \land (\forall c \in \{Q\}. \ \exists b. \ (c, b) \in r \land b \in \{Q\}\) by (rule exI [of - \(\{Q\}\)])
    then have \(\neg (\forall Q. \ a \in Q \rightarrow (\exists c \in Q. \forall b. \ (c, b) \in r \rightarrow b \notin \{Q\}))\) by simp
    with \(\text{SN-on-imp-on-minimal} \ [\text{of} \ r \ a]\) have \(\neg \text{SN-on} \ r \ \{a\}\) by blast
    with assms and \(a \in A\) and \(\text{SN-on-subset2} \ [\text{of} \ \{a\} \ A \ r]\) show \(\text{False}\) by simp
  qed
qed

lemma \(\text{SN-on-imp-normalizability}\):
  assumes \(\text{SN-on} \ r \ \{a\}\) shows \(\exists b. \ (a, b) \in r^\leftarrow\)
  using \(\text{SN-on-Image-normalizable} \ [\text{OF} \ \text{assms}]\) by auto

2.4 Commutation

definition commute :: \('a\ rel \Rightarrow 'a\ rel \Rightarrow \text{bool}\)
  where \(\text{commute} \ r \ s \leftarrow \ ((r^{-1})^* \ O \ s^*) \subseteq (s^* \ O \ (r^{-1})^*)\)

lemma \(\text{CR-iff-self-commute}\): \(\text{CR} \ r = \text{commute} \ r \ r\)
  unfolding commute-def \(\text{CR-iff-meet-subset-join} \ \text{meet-def} \ \text{join-def}\)
  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 \((y, z) \in r\) and \((r \in I)\) have \((y, z) \in ?r^*\) by auto
    with \((x, y) \in ?r^*) show \(?case by auto\)
  qed

definition quasi-commute :: \('a rel \Rightarrow 'a rel \Rightarrow bool\) where
  quasi-commute \(r \leq s\) \iff (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 \(?thesis by auto\)
    next
      case (step y z)
      then have \((y, z) \in r \cup (y, z) \in s^+\) 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 \(?thesis using rtrancl-Union-subset by auto\)
      qed
    with \((x, y) \in (r \cup s)^*\) show \(?case by simp\)
  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
  qed

44
with ⟨(x, y) ∈ (r ∪ s)^*⟩, 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) ∈ s^+ O r⟩
  then obtain y where (x, y) ∈ s^+ and (y, z) ∈ r by best
  then show ⟨(x, z) ∈ r ∪ (r ∪ s)^*⟩
  proof (induct arbitrary: z)
    case (base y)
    then have ⟨(x, z) ∈ (s O r)⟩ by auto
    with assms have ⟨(x, z) ∈ r ∪ (r ∪ 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) ∈ (s O r)⟩ by auto
    with assms have ⟨(a, z) ∈ r ∪ (r ∪ s)^*⟩ unfolding quasi-commute-def by auto
    then obtain u where ⟨(a, u) ∈ r and (u, z) ∈ (r ∪ s)^*⟩ by best
    with ⟨(u, z) ∈ (r ∪ s)^*⟩ have ⟨(v, z) ∈ (r ∪ s)^*⟩ by auto
    with ⟨(x, v) ∈ r⟩ show ?case by auto
  qed
qed

lemma steps-reflect-SN-on:
assumes ¬ SN-on r {b} and (a, b) ∈ r^*
shows ¬ SN-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 ¬ SN-on r {f i}
proof –
  let ?f = λj. f (i + j)
  have ?f 0 ∈ {f i} by simp
  moreover have chain r ?f using assms by auto
  ultimately have ?f 0 ∈ {f i} ∧ chain r ?f by blast
  then have ∃g. g 0 ∈ {f i} ∧ chain r g by (rule exI [of ?f])
  then show ?thesis unfolding SN-defs by auto
qed

45
lemma quasi-commute-imp-SN:
assumes $SN \ r$ and $SN \ s$ and quasi-commute $r \ s$
shows $SN \ (r \cup s)$

proof
\begin{itemize}
\item have quasi-commute $r \ (s^+)$ by (rule qc-imp-qc-trancl \cite{OF \ quasi-commute \ r \ s})
\item let $\mathcal{B} = \{a. \neg SN-on \ (r \cup s) \ \{a\}\}$
\item \begin{itemize}
\item assume $\neg SN(r \cup s)$
\item then obtain $a$ where $a \in \mathcal{B}$ unfolding $SN-defs$ by fast
\item from $\langle SN r \rangle$ have $\forall Q. \ \exists zQ. \ \forall y. \ (z, y) \in r \longrightarrow y \notin Q$
\item by (rule SN-imp-minimal)
\item then have $\forall x. \ x \in \mathcal{B} \longrightarrow (\exists z \in S. \ \forall y. \ (z, y) \in r \longrightarrow y \notin \mathcal{B})$ by (rule spec [where $x = \mathcal{B}$])
\item with $\langle a \in \mathcal{B} \rangle$ obtain $b$ where $b \in \mathcal{B}$ and min: $\forall y. \ (b, y) \in r \longrightarrow y \notin \mathcal{B}$ by auto
\end{itemize}
\item from $\langle b \in \mathcal{B} \rangle$ obtain $S$ where $S 0 = b$ and
\item chain: chain $(r \cup s) S$ unfolding $SN-on-def$ by auto
\item let $\bar{S} = \lambda i. \ S(Suc i)$
\item have $\langle \bar{S} 0 = S / i \rangle$ by simp
\item from chain have chain $(r \cup s) \bar{S}$ by auto
\item with $\langle \bar{S} 0 = S / i \rangle$ have $\neg SN-on \ (r \cup s) \ \{S / i\}$ unfolding $SN-on-def$ by auto
\item from $\langle S 0 = b \rangle$ and chain have $(b, S / i) \in r \cup s$ by auto
\item with min and $\neg SN-on \ (r \cup s) \ \{S / i\}$: have $(b, S / i) \in s$ by auto
\item let $\bar{i} = LEAST i. \ (S / i, S(Suc i)) \notin s$
\item \begin{itemize}
\item assume chain $s \ S$
\item with $\langle S 0 = b \rangle$ have $\neg SN-on \ s \ \{b\}$ unfolding $SN-on-def$ by auto
\item with $\langle SN \ s \rangle$ have $False$ unfolding $SN-defs$ by auto
\end{itemize}
\item then have $\exists i. \ (S / i, S(Suc i)) \notin s$ by auto
\item then have $(S \ ?i, S(Suc \ ?i)) \notin s$ by (rule LeastI-ex)
\item with chain have $(S \ ?i, S(Suc \ ?i)) \in r$ by auto
\item have ini: $\forall i < \bar{i}. \ (S / i, S(Suc i)) \in s$ using not-less-Least by auto
\item \begin{itemize}
\item fix $i$ assume $i < \bar{i}$ then have $(b, S(Suc i)) \in s^+$
\item proof (induct $i$)
\item case $0$ then show ?case using $\langle (b, S / i) \in s \rangle$ and $\langle S 0 = b \rangle$ by auto
\item next
\item case $(Suc k)$
\item then have $(b, S(Suc k)) \in s^+$ and $Suc k < \bar{i}$ by auto
\item with $\forall i < \bar{i}. \ (S / i, S(Suc i)) \in s$ have $(S(Suc k), S(Suc(Suc k))) \in s$ by fast
\item with $\langle (b, S(Suc k)) \in s^+ \rangle$ show ?case by auto
\item qed
\end{itemize}
\item then have pref: $\forall i < \bar{i}. \ (b, S(Suc i)) \in s^+$ by auto
\item from $\langle (b, S / i) \in s \rangle$ and $\langle S 0 = b \rangle$ have $(S 0, S(Suc 0)) \in s$ by auto
\item \begin{itemize}
\item assume $\bar{i} = 0$
\end{itemize}
\end{itemize}
\end{itemize}
from \texttt{ex} have \((S \; \exists i, S(Suc \; \exists i)) \not\in s\) by \texttt{(rule \; LeastI-ex)}
with \((S \; 0, S(Suc \; 0)) \in s\) have \texttt{False} unfolding \(\exists i = 0\) by \texttt{simp} \}
then have \(0 < \exists i\) by \texttt{auto}
then obtain \(j\) where \(\exists i = Suc \; j\) unfolding \texttt{gr0-conv-Suc} by \texttt{best}
with \texttt{ini} have \((S(\exists i - Suc \; 0), S(Suc(\exists i - Suc \; 0))) \in s\) by \texttt{auto}
with \texttt{pref} have \((b, S(Suc \; j)) \in s^+\) unfolding \(\exists i = Suc \; j\) by \texttt{auto}
then have \((b, S \; \exists i) \in s^+\) unfolding \(\exists i = Suc \; j\) by \texttt{auto}
with \texttt{with} \((S \; \exists i, S(Suc \; \exists i)) \in r\) have \((b, S(Suc \; \exists i)) \in (s^+ \; O \; r)\) by \texttt{auto}
with \texttt{with} \((\texttt{quasi-commute} \; r \; (s^+))\) have \((b, S(Suc \; \exists i)) \in r \; O \; (r \; \cup \; s^+)\)
unfolding \texttt{quasi-commute-def} by \texttt{auto}
then obtain \(c\) where \((b, c) \in r\) and \((c, S(Suc \; \exists i)) \in (r \; \cup \; s^+)\) by \texttt{best}
from \((b, c) \in r\) have \((b, c) \in (r \; \cup \; s)^*\) by \texttt{auto}
from \texttt{with} \((c, S(Suc \; \exists i)) \in (r \; \cup \; s)^*\) have \((c, S(Suc \; \exists i)) \in (r \; \cup \; s)^*\)
unfolding \texttt{rtrancl-union-subset-rtrancl-union-trancl} by \texttt{auto}
with \texttt{with} \((\neg \; SN-on \; (r \; \cup \; s) \; \{S(Suc \; \exists i)\})\) by \texttt{auto}
then have \(c \in ?B\) by \texttt{simp}
with \((b, c) \in r\) and \texttt{min} have \texttt{False} by \texttt{auto} \}
then show \(?thesis\) by \texttt{auto}
qed

\subsection*{2.5 Strong Normalization}

\textbf{lemma} \texttt{non-strict-into-strict}:
assumes \texttt{compat} \(NS \; O \; S \subseteq S\)
and \texttt{steps} \((s, t) \in (NS^*) \; O \; S\)
shows \((s, t) \in S\)
using \texttt{steps} \texttt{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 \texttt{auto}
then show \(?thesis\)
proof \texttt{induct} \texttt{rule:} \texttt{rtrancl.induct} \texttt{case} \texttt{rtrancl-refl} \texttt{x} \then show \(?case\).
next \case \texttt{rtrancl-into-rtrancl} \texttt{a} \texttt{b} \texttt{c}
with \texttt{compat} \texttt{show} \(?case\) by \texttt{auto}
qed

\textbf{lemma} \texttt{comp-trancl}:
assumes \texttt{R \; O \; S \subseteq S} shows \texttt{R \; O \; S^+ \subseteq S^+}
proof \texttt{(rule subrelI)}
fix \(w\) \(z\) assume \((w, z) \in R \; O \; S^+\)
then obtain \(x\) where \texttt{R-step} \((w, x) \in R\) and \texttt{S-seq} \((x, z) \in S^+\) by \texttt{best}
from tranclD [OF S-seq] obtain y where S-step: \((x, y) \in S\) and S-seq': \((y, z) \in S^*\) by auto
from R-step and S-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: \(R O S \subseteq S\)
and seq: \((s, t) \in (R \cup S)^* O S\)
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 \((b, c) \in S\) by auto
with ⟨\((x, a) \in (s \cup r)^*\) ⟩ show ?thesis by simp
next
assume \((b, c) \in R\)
with comp-trancl [OF comp] rtrancl-into-rtrancl show ?thesis by auto
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 \(\langle(x, a) \in (s \cup r)^*\) show ?case by auto
qed
qed

lemma restrict-SN-subset: restrict-SN R S \(\subseteq R\)
proof (rule subrelI)
\textbf{proof} \ \ \ \textbf{lemma} \ chain-Un-SN-on-imp-first-step: \\
assumes \ (R \cup S) \ t \ \text{and} \ SN-on \ S \ \{t \ 0\} \\
\shows \ \exists i. \ ((t, i, t \ (Suc \ i)) \in R \ \land \ \forall j<i. \ ((t, j, t \ (Suc \ j)) \in S \ \land \ (t, j, t \ (Suc \ j)) \notin R) \\
\textbf{proof} – \\
\textbf{from} \ (SN-on \ S \ \{t \ 0\}) \ \text{obtain} \ i \ \text{where} \ ((t, i, t \ (Suc \ i)) \notin S \ \text{by} \ \text{blast} \\
\ \text{with} \ \text{assms have} \ ((t, i, t \ (Suc \ i)) \in R \ (is \ ?P \ i) \ \text{by} \ \text{auto} \\
\ \text{let} \ ?i = \ Least \ ?P \\
\ \text{from} \ (?P \ ?i \ \text{have} \ ?P \ ?i \ \text{by} \ \text{rule} \ \text{LeastI}) \\
\ \text{have} \ \forall j<i. \ ((t, j, t \ (Suc \ j)) \notin R \ \text{using} \ \text{not-less-Least} \ \text{by} \ \text{auto} \\
\ \text{moreover with} \ \text{assms have} \ \forall j<i. \ ((t, j, t \ (Suc \ j)) \in S \ \text{by} \ \text{best} \\
\ \text{ultimately have} \ \forall j<i. \ ((t, j, t \ (Suc \ j)) \in S \ \land \ (t, j, t \ (Suc \ j)) \notin R \ \text{by} \ \text{best} \\
\ \text{with} \ (?P \ ?i) \ \text{show} \ ?thesis \ \text{by} \ \text{best} \\
\textbf{qed} \\

\textbf{lemma} \ first-step: \\
\assumes \ C \in A \cup B \ \text{and} \ steps: \ (x, y) \in C^* \ \text{and} \ Bstep: \ (y, z) \in B \\
\shows \ \exists y. \ (x, y) \in A^* \ O B \\
\textbf{proof} \ \text{(induct rule: converse-rtrancl-induct)} \\
\ \text{case} \ base \\
\ \text{show} \ ?case \ \text{using} \ Bstep \ \text{by} \ \text{auto} \\
\textbf{next} \\
\ \text{case} \ (step \ u \ x) \\
\ \text{from} \ \text{step(1)|unfolded} \ C \ \\
\ \text{show} \ ?case \\
\ \textbf{proof} \\
\ \text{assume} \ (u, x) \in B \\
\ \text{then show} \ ?thesis \ \text{by} \ \text{auto} \\
\textbf{next} \\
\ \text{assume} \ uw: \ (u, x) \in A \\
\ \text{from} \ \text{step(3)} \ \text{obtain} \ y \ \text{where} \ (x, y) \in A^* \ O B \ \text{by} \ \text{auto} \\
\ \text{then obtain} \ z \ \text{where} \ (x, z) \in A^* \ \text{and} \ \text{step}: \ (z, y) \in B \ \text{by} \ \text{auto} \\
\ \text{with} \ \text{uw have} \ (u, z) \in A^* \ \text{by} \ \text{auto} \\
\ \text{with} \ \text{step have} \ (u, y) \in A^* \ O B \ \text{by} \ \text{auto} \\
\ \text{then show} \ ?thesis \ \text{by} \ \text{auto} \\
\textbf{qed} \\
\textbf{qed} \\

\textbf{lemma} \ first-step-O: \\
\assumes \ C \in A \cup B \ \text{and} \ steps: \ (x, y) \in C^* \ O B \\
\shows \ \exists y. \ (x, y) \in A^* \ O B \\
\textbf{proof} – \\
\ \text{from} \ \text{steps obtain} \ z \ \text{where} \ (x, z) \in C^* \ \text{and} \ (z, y) \in B \ \text{by} \ \text{auto} \\
\ \text{from} \ \text{first-step} \ [OF \ C \ \text{this}] \ \text{show} \ ?thesis .
lemma firstStep:
  assumes LSR: \( L = S \cup R \) and \( xyL: (x, y) \in L^* \)
  shows \( (x, y) \in R^* \lor (x, y) \in R^* O S O L^* \)
proof (cases \( (x, y) \in R^* \))
  case True
  then show \( \text{thesis} \) by simp
next
  case False
  let \( ?SR = S \cup R \)
  from xyL and LSR have \( (x, y) \in \text{SR}^* \) by simp
  from this and False have \( (x, y) \in R^* O S O \text{SR}^* \) by simp
proof (induct rule: rtrancl-induct)
  case base then show \( \text{case} \) by simp
next
  case (step y z)
  then show \( \text{case} \) by simp
  proof (cases \( (x, y) \in R^* \))
    case False with step have \( (x, y) \in R^* O S O \text{SR}^* \) by simp
    from this obtain \( u \) where \( xu: (x, u) \in R^* O S \) and \( uy: (u, y) \in \text{SR}^* \) by force
    from \( (y, z) \in \text{SR} \) have \( (y, z) \in S \) by blast
    with xu show \( \text{thesis} \) by auto
  qed
  with True show \( \text{thesis} \) by auto
  qed
  qed
  with LSR show \( \text{thesis} \) by simp
qed

lemma non-strict-ending:
  assumes chain: chain \( (R \cup S) t \)
  and comp: \( R O S \subseteq S \) and \( SN: SN-on S \{t 0\} \)
  shows \( \exists j. \forall i \geq j. (t i, t (Suc i)) \in R \setminus S \)
proof (rule ccontr)
  assume \( \neg \text{thesis} \)
  with chain have \( \forall i. \exists j. j \geq i \land (t j, t (Suc j)) \in S \) by blast
  from choice \( \text{OF this} \) obtain \( f \) where S-steps: \( \forall i. i \leq f i \land (t (f i), t (Suc f i)) \in R \setminus S \) by blast
  qed
  qed
  qed
  with LSR show \( \text{thesis} \) by simp
qed
let \( \text{?t} = \lambda i. ((\text{Suc } \circ f)^\langle i \rangle \text{ ?}) \)

have S-chain: \( \forall i. (t i, t (\text{Suc } f i)) \in S^+ \)

proof
  fix i
  from S-steps have leg: \( i \leq f i \) and step: \( (t f i), t(\text{Suc } f i)) \in S \) by auto
  from chain-imp-rtrancl [OF chain leg] have \( (t i, t(\text{Suc } f i)) \in (R \cup S)^+ \).
  with step have \( (t i, t(\text{Suc } f i)) \in (R \cup S)^+ \) \( O S \) by auto
  from comp-rtrancl-trancl [OF comp this] show \( (t i, t(\text{Suc } f i)) \in S^+ \).
qed

then have chain \((S^+)^?t\) by simp
moreover have \( \text{SN-on} (S^+) \{ \text{?t } 0 \} \) using \( \text{SN-on-trancl} \) [OF \( \text{SN} \)] by simp
ultimately show False unfolding \( \text{SN-defs} \) by best
qed

lemma SN-on-subset1:
  assumes \( \text{SN-on} \ r \ A \) and \( s \subseteq r \)
  shows \( \text{SN-on} \ s \ A \)
  using assms unfolding \( \text{SN-defs} \) by blast

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

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

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

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

lemma compatible-SN:
  assumes \( \text{SN}: \text{SN } S \)
  and \( \text{compat}: \text{NS } O S \subseteq S \)
  shows \( \text{SN} (S O S^* O \text{NS }^*) \) (is \( \text{SN } ?A \))
proof
  fix F assume chain: \( \text{chain } ?A F \)
  from compat compat-tr-compat have \( \text{tr-compat }: \text{NS }^* O S \subseteq S \) by blast

51
have \( \forall i. (\exists y. z. (F i, y) \in S \land (y, z) \in S^* \land (z, F (Suc i)) \in NS^*) \)

proof

fix \( i \)

from chain have \((F i, F (Suc i)) \in (S O S^* O NS^*)\) by auto
then show \( \exists y. z. (F i, y) \in S \land (y, z) \in S^* \land (z, F (Suc i)) \in NS^* \)
unfolding relcomp-def using mem-Collect-eq by auto

qed

then have \( \exists f. (\forall i. (\exists z. (F i, f i) \in S \land ((f i, z) \in S^*) \land (z, F (Suc i)) \in NS^*)) \)
by (rule choice)
then obtain \( f \)
where \( \forall i. (\exists z. (F i, f i) \in S \land ((f i, z) \in S^*) \land (z, F (Suc i)) \in NS^*) \).
then have \( \exists g. \forall i. (F i, f i) \in S \land (f i, g i) \in S^* \land (g i, F (Suc i)) \in NS^* \)
by (rule choice)
then obtain \( g \) where \( \forall i. (F i, f i) \in S \land (f i, g i) \in S^* \land (g i, F (Suc i)) \in NS^* \).
then have \( \forall i. (f i, g i) \in S^* \land (g i, F (Suc i)) \in NS^* \land (F (Suc i), f (Suc i)) \in S \)
by auto
then have \( \forall i. (f i, g i) \in S^* \land (g i, F (Suc i)) \in S^* \) unfolding relcomp-def
using tr-compat by auto
then have all:\( \forall i. (f i, g i) \in S^* \land (g i, F (Suc i)) \in S^* \)
by auto
have \( \forall i. (f i, f (Suc i)) \in S^* \)
proof

fix \( i \)
from all have \( (f i, g i) \in S^* \land (g i, F (Suc i)) \in S^* \).
then show \( (f i, f (Suc i)) \in S^* \) using transitive-closure-trans by auto

qed

then have \( \exists x. f 0 = x \land chain (S^+) \) by auto
then obtain \( x \) where \( f 0 = x \land chain (S^+) \) by auto
then have \( \exists f. f 0 = x \land chain (S^+) \) by auto
then have \( \neg SN\text{-}on (S^+) \{x\} \) by auto
then have \( \neg SN \) (S^+) unfolding SN-defs by auto
then have \( wf\text{-}S\text{-}S\text{-}Conv\)\( : \neg wf ((S^+)\text{-}1)\) using SN-iff-wf by auto
from \( SN \) have \( wf (S\text{-}1) \) using SN-imp-wf \( \text{[where } \neg \text{R}=S\text{] by simp} \)
with \( wf\text{-}converse\text{-}trancl \) \( wf\text{-}Conv \) show \( False \) by auto

qed

lemma compatible-rtrancl-split:

assumes \( \text{compat: } NS O S \subseteq S \)
and steps: \( (x, y) \in (NS \cup S)^* \)
shows \( (x, y) \in S \) \( O \) \( S^* \) \( O \) \( NS^* \) \( \cup \) \( NS^* \)
proof

from steps have \( \exists n. (x, y) \in (NS \cup S)^{n} \) using rtrancl-imp-relpow \( \text{[where } R=NS \cup S\text{] by auto} \)
then obtain \( n \) where \( (x, y) \in (NS \cup S)^{n} \) by auto
then show \( (x, y) \in S \) \( O \) \( S^* \) \( O \) \( NS^* \) \( \cup \) \( NS^* \)
proof (induct \( n \) arbitrary: \( x \), simp)

case \( Suc \) \( n \)

52
assume \((x, y) \in (NS \cup S)^\sim \) (Suc \(m\))
then have \(\exists \ z. (x, z) \in (NS \cup S) \land (z, y) \in (NS \cup S)^\sim \) \(m\)
using relpow-Suc-D2 \[\text{where} \ ?R=NS \cup S\] 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 \(\text{True}\)
    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 \(xz; (x, y) \in (S O S^* O NS^*) \cup (NS O NS^*)\) 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 \(\text{False}\)
    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

lemma compatible-conv:
  assumes \(\text{compat; } NS O S \subseteq S\)
  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 \(\?NSS \subseteq ?midS\) by regexp
  have \(\?NSuS^* O S \subseteq (?NSS \cup NS^*) \ O S\)
    using compatible-rtrancl-split \[\text{where} \ S = S \ \text{and} \ NS = NS\] compat by blast
  also have \(\ldots \subseteq ?NSS O S \cup NS^* \ O S\) by auto
  also have \(\ldots \subseteq ?NSS O S \cup S\) using compat compat-tr-compat \[\text{where} \ S = S\)
  and \(\text{NS = NS}\) by auto
  also have \(\ldots \subseteq S O \ ?NSuS^*\) by regexp
  finally have \(?midS \subseteq S O ?NSuS^* \ O ?NSuS^*\) by blast
  also have \(\ldots \subseteq S O (?NSS \cup NS^*)\)
    using compatible-rtrancl-split \[\text{where} \ S = S \ \text{and} \ NS = NS\] compat by blast
  also have \(\ldots \subseteq ?NSS\) by regexp
  finally have \(two; ?midS \subseteq ?NSS\).
  from \(\text{one two}\) show \(?thesis\) by auto
qed

lemma compatible-\(\text{SN}'\):

assumes $\text{compat}: NS O S \subseteq S$ and $\text{SN}: SN S$
shows $\text{SN}((NS \cup S)^* O S O (NS \cup S)^*)$

using $\text{compatible-conv}$ and $\text{SN-empty}$

$\text{SN-on-weakening}$

lemma $\text{rtrancl-diff-decomp}$:

assumes $(x, y) \in A^* - B^*$
shows $(x, y) \in A^* O (A - B) O A^*$

proof –

from $\text{assms}$ have $A: (x, y) \in A^* \text{ and } B: (x, y) \notin B^*$ by auto
from $A$ have $\exists k. (x, y) \in A^{\sim k}$ by (rule $\text{rtrancl-imp-relpow}$)
then obtain $k$ where $Ak: (x, y) \in A^{\sim k}$ by auto
from $Ak B$ show $(x, y) \in A^* O (A - B) O A^*$

proof (induct $k$ arbitrary: $x$)

next

<table>
<thead>
<tr>
<th>case ${ \text{Suc i} }</th>
</tr>
</thead>
</table>

then have $B: (x, y) \notin B^* \text{ and } ASk: (x, y) \in A^{\sim i}$ \text{ by auto}

from $ASk$ have $\exists z. (x, z) \in A \land (z, y) \in A^{\sim i}$ using $\text{relpow-Suc-D2}$

\text{where } ?R=A \text{ by auto}

then obtain $z$ where $xz: (x, z) \in A \text{ and } (z, y) \in A^{\sim i}$ \text{ by auto}

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

from $xz$ show $(x, y) \in A^* O (A - B) O A^*$

proof (cases $(x, z) \in B$)

case $\{ \text{False} \}$

with $xz \ yz$ show $(x, y) \in A^* O (A - B) O A^*$ \text{ by auto}

next

<table>
<thead>
<tr>
<th>case ${ \text{True} }</th>
</tr>
</thead>
</table>

then have $(x, z) \in B^*$ \text{ by auto}

have $[(x, z) \in B^*; (z, y) \in B^*] \Rightarrow (x, y) \in B^*$ using $\text{rtrancl-trans}$ [of $x \ y$ $B$]

\text{by auto}

with $\{ \langle x, z \rangle \in B^*; (x, y) \notin B^* \}$ have $(z, y) \notin B^*$ \text{ by auto}

with $\text{Suc} \langle (z, y) \in A^{\sim i} \rangle$ have $(z, y) \in A^* O (A - B) O A^*$ \text{ by auto}

with $xz$ have $zy: (x, y) \in A O A^* O (A - B) O A^*$ \text{ by auto}

have $A O A^* O (A - B) O A^* \subseteq A^* O (A - B) O A^*$ \text{ by regexp}

from this $xz$ show $(x, y) \in A^* O (A - B) O A^*$

using $\text{subsetD}$ \text{ where } ?A=A \ O A^* O (A - B) O A^* \text{ by auto}

qed

qed

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

lemma $\text{SN-on-weakening}$:

assumes $\text{SN-on } R1 \ A$
shows $\text{SN-on } (R1 \cap R2) \ A$

proof –

{}
assume ∃S. S 0 ∈ A ∧ chain (R1 ∩ R2) S
then obtain S where
 S0: S 0 ∈ A and
 SN: chain (R1 ∩ 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 ⇒ 'a rel ⇒ ('a ⇒ bool) ⇒ bool where
 ideriv R S as (⇒ (∀ i. (as i, as (Suc i)) ∈ R ∪ S) ∧ (INFM i. (as i, as (Suc i)) ∈ R))

lemma ideriv-mono: R ⊆ R' ⇒ S ⊆ S' ⇒ ideriv R S as ⇒ ideriv R' S' as
 unfolding ideriv-def INFM-nat by blast

fun
 shift :: ('a ⇒ bool) ⇒ 'a ⇒ 'a
 where
 shift f j = (λ i. f (i+j))

lemma ideriv-split:
 assumes ideriv: ideriv R S as
 and nideriv: ¬ ideriv (D ∩ (R ∪ S)) (R ∪ S − D) as
 shows ∃ i. ideriv (R − D) (S − D) (shift as i)
 proof –
 have RS: R − D ∪ (S − D) = R ∪ S − D by auto
 from ideriv [unfolded ideriv-def]
 have as: j i. (as i, as (Suc i)) ∈ R ∪ S
 and inf: INFM i. (as i, as (Suc i)) ∈ R by auto
 show ?thesis
 proof (cases INFM i. (as i, as (Suc i)) ∈ D ∩ (R ∪ S))
 case True
 have ideriv (D ∩ (R ∪ S)) (R ∪ 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: j. i < j =⇒ (as j, as (Suc j)) ∉ D ∩ (R ∪ S)
 by auto
 from Dn as have as: j i. i < j =⇒ (as j, as (Suc j)) ∈ R ∪ 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)

55
fix m
from inf [unfolded INFM-nat] obtain j where j: j > Suc i + m
    and R: (as j, as (Suc j)) ∈ R by auto
with as [of j] have RD: (as j, as (Suc j)) ∈ R − D by auto
show ∃ j > m. (shift as (Suc i) j, shift as (Suc i) (Suc j)) ∈ 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 ⊆ S
    and R: R ⊆ NS ∪ S
  shows ¬ ideriv (S ∩ R) (R − S) as
proof
  assume ideriv (S ∩ R) (R − S) as
  with R have steps: ∀ i. (as i, as (Suc i)) ∈ NS ∪ S
    and inf: INFM i. (as i, as (Suc i)) ∈ S ∩ R unfolding ideriv-def by auto
  from non-strict-ending [OF steps compat] SN
  obtain i where i: ∃ j. j ≥ i ⇒ (as j, as (Suc j)) ∈ NS − S by fast
  from inf [unfolded INFM-nat] obtain j where j > i and (as j, as (Suc j)) ∈ S by auto
  with i [of j] show False by auto
qed

next
assume ?O
show ?S
  unfolding INFM-nat-le
proof
  fix m
  from (?S) [unfolded INFM-nat-le]
  obtain k where k: k ≥ m and p: P (shift f n k) by auto
  show ∃ k ≥ m. P (f k)
    by (rule exI [of - k + n], insert k p, auto)
qed
next
assume ?O
show ?S
  unfolding INFM-nat-le
proof
  fix m
  from (?O) [unfolded INFM-nat-le]
  obtain k where k: k ≥ m + n and p: P (f k) by auto
  show ∃ k ≥ m. P (shift f n k)
    by (rule exI [of - k - n], insert k p, auto)
qed
**lemma** rtrancl-list-conv:

\[(s, t) \in R^* \iff (\exists ts. \text{last} (s \# ts) = t \land (\forall i < \text{length} ts. ((s \# ts) ! i, (s \# ts) ! \text{Suc} i) \in R))\]

(is \(t = t\))

**proof**

assume \(?r\)
then obtain \(ts\) where \(\text{last} (s \# ts) = t \land (\forall i < \text{length} ts. ((s \# ts) ! i, (s \# ts) ! \text{Suc} i) \in R)\)
then show \(?l\)
proof (induct \(ts\) arbitrary; \(s\), simp)

**next**

assume \(?l\)
from rtrancl-imp-seq [OF this]

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

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

**qed**

**lemma** SN-reaches-NF:

assumes \(\text{SN-on } r \{x\}\)
shows \(\exists y. (x, y) \in r^* \land y \in \text{NF } r\)

**proof** (induct rule: \(\text{SN-on-induct'}\))

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

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 \(\exists! \ y. (x, y) \in r^* \land y \in NF \ r\)

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

definition \(some\text{-}NF:: \('a \ rel \Rightarrow 'a \Rightarrow 'a\) where
some\text{-}NF \(r\ x = (SOME \ y. (x, y) \in r^* \land y \in NF \ r)\)

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

lemma \(some\text{-}NF\text{-}WCR:\)
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 \(\lambda y. (x, y) \in r^* \land y \in NF \ r\)
from \(SN\text{-}WCR\text{-}reaches-NF \ [OF SN WCR]\)
have one: \(\exists! \ y. \ ?p \ y\).
from \(steps NF\) have \(y: \ ?p \ y\) ..

58
from some-NF [OF SN] have some: ?p (some-NF r x).
from one some y show ?thesis by auto
qed

lemma some-NF-UNF:
  assumes UNF: UNF r
  and steps: (x, y) ∈ r^*
  and NF: y ∈ NF r
  shows y = some-NF r x
proof –
  let ?p = λ y. (x, y) ∈ r^* ∧ y ∈ NF r
  from steps NF have py: ?p y by simp
  then have pNF: ?p (some-NF r x) unfolding some-NF-def
  by (rule someI)
  from py have y: (x, y) ∈ r! by auto
  from pNF have nf: (x, some-NF r x) ∈ 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) ∈ A!)

custom
  fixes A
  assumes SN: SN A and CR: CR A
begin
lemma the-NF: (a, the-NF A a) ∈ A!
proof –
  obtain b where ab: (a, b) ∈ A! using SN by (meson SN-imp-WN UNIV-I
WN-onE)
  moreover have (a, c) ∈ A! ⇒ c = b for c
  using CR and ab by (meson CR-divergence-imp-join join-NF-imp-eq normalizability-E)
  ultimately have ∃ b. (a, b) ∈ A! by blast
  then show ?thesis unfolding the-NF-def by (rule theI')
qed

lemma the-NF-NF: the-NF A a ∈ NF A
using the-NF by (auto simp: normalizability-def)

lemma the-NF-step:
  assumes (a, b) ∈ A
  shows the-NF A a = the-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) ∈ A^*
  shows the-NF A a = the-NF A b
using assms by (induct) (auto dest: the-NF-step)
lemma the-NF-conv:
  assumes \((a, b) \in A^{**}\)
  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\n  w\Diamond r \longleftrightarrow (r^{-1} O r) - Id \subseteq (r O r^{-1})\nlemma 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 \(\text{step}: (z, x) \in r\) and \(\text{steps}: (z, y) \in r^*\) by auto
  from steps
  have \(\exists u. (x, u) \in r^* \land (y, u) \in r^=\)
proof (induct)
  case base
  by (rule exI [of - x], insert step, auto)
next
  case (step y' y)
  from step(3) obtain \(u\) where \(\text{xu}: (x, u) \in r^*\) and \(\text{yu}: (y', u) \in r^=\) by auto
  from y'u have \((y', u) \in r \lor y' = u\) 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
  \(\text{wd [unfolded weak-diamond-def]}\) obtain \(u'\) where \(uu': (u, u') \in r\)
  and \(yu': (y, u') \in r\) by auto
from xu uu' have xu: (x, u') ∈ r* by auto
show ?thesis by (intro exI conjI, rule xu, insert yu', auto)
qed
qed
qed
then show (x, y) ∈ r↓ by auto
qed

lemma steps-imp-not-SN-on:
fixes t :: 'a ⇒ 'b
and R :: 'b rel
assumes steps: ∃ x. (t x, t (f x)) ∈ R
shows ¬ SN-on R {t x}
proof
let ?U = range t
assume SN-on R {t x}
from SN-on-imp-on-minimal [OF this, rule-format, of ?U]
obtain tz where tz: tz ∈ range t and min: ∃ y. (tz, y) ∈ R ⇒ y ∉ range t
by auto
from tz obtain z where tz: 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 ⇒ 'b
and R :: 'b rel
assumes steps: ∃ x. (t x, t (f x)) ∈ R
shows ¬ 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: (∀ t u R . P t ⇒ Q R ⇒ (t, u) ∈ R ⇒ P u ∧ (f t, f u) ∈ g R
and t: P t
and R: Q R
and S: Q S
shows ((t, u) ∈ R* ⇒ (f t, f u) ∈ (g R)*)
∧ ((t, u) ∈ R* O S O R* ⇒ (f t, f u) ∈ (g R)^* O (g S) O (g R)^*)
proof
{ fix t u
assume (t, u) ∈ R* and P t
then have P u ∧ (f t, f u) ∈ (g R)^*
proof (induct)
case (step u v)
from step(3)(OF step(4)]) have Pa: P u and steps: (f t, f u) ∈ (g R)^* by
from $f\ g$ [OF $Ps$ $R$ \(\text{step}(2)\)] have $Pv$: $P\ v$ and \(\text{step}: (f\ u, f\ v) \in (g\ R)^*\) by auto
with steps have $(f\ t, f\ v) \in (g\ R)^*$ by auto
with $Pv$ show $?\text{case}\ by\ \text{simp}$
qed simp
}
} note main = this
note main[t] = main [OF - t]
from main [of u] have one: $(t, u) \in R^* \longrightarrow (f\ t, f\ u) \in (g\ R)^*$ by simp
show $?\text{thesis}$
proof (rule conjI [OF one \text{impl}])
assume $(t, u) \in R^* \ O\ S\ O\ R^*$
then obtain $s\ v$ where $ts: (t, s) \in R^*$ and $sv: (s, v) \in S$ and $vu: (v, u) \in R^*$ by auto
from main [OF $ts$] have $Ps$: $P\ s$ and $ts: (f\ t, f\ s) \in (g\ R)^*$ by auto
from $f\ g$ [OF $Ps\ sv$] have $Pv$: $P\ v$ and $sv: (f\ s, f\ v) \in g\ S$ by auto
from main [OF $vu\ Pv$] have $vu$: $(f\ v, f\ u) \in (g\ R)^*$ by auto
qed

2.6 Terminating part of a relation

inductive-set
$SN\text{-part}:: \ 'a\ rel \Rightarrow \ 'a\ set$
for $r:: \ 'a\ rel$
where
$SN\text{-partI}: (\land y. (x, y) \in r \Rightarrow y \in SN\text{-part} \ r) \Rightarrow x \in SN\text{-part} \ r$

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

Characterization of $SN\text{-on}$ via terminating part.

lemma $SN\text{-on-SN}\text{-part-conv}$:
$SN\text{-on} \ r\ A \longleftrightarrow A \subseteq SN\text{-part} \ r$
proof –
\{
\begin{align*}
\text{fix } x & \text{ assume } SN\text{-on} \ r\ A \text{ and } x \in A \\
& \text{ then have } x \in SN\text{-part} \ r \text{ by (induct) (auto intro: } SN\text{-partI})
\end{align*}
\}
moreover \{
\begin{align*}
\text{fix } x & \text{ assume } x \in A \text{ and } A \subseteq SN\text{-part} \ r \\
& \text{ then have } x \in SN\text{-part} \ r \text{ by auto}
\end{align*}
\}
then have $SN\text{-on} \ r \ \{x\}$ by (induct) (auto intro: step-reflects-SN-on)
\}
ultimately show $?\text{thesis}\ by\ (force\ simp: \text{SN-defs})$
qed

Special case for “full” termination.

lemma $SN\text{-SN}\text{-part-UNIV-conv}$:
$SN\ r \longleftrightarrow SN\text{-part} \ r = UNIV$
using $SN\text{-on-SN}\text{-part-conv} \ [\text{of } r \ \text{UNIV}]$ by auto

62
lemma closed-imp-rtrancl-closed: assumes $L \subseteq A$
and $R: R \subseteq A$
shows $\{ t \mid s, s \in L \land (s,t) \in R^* \} \subseteq A$

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

lemma trancl-steps-relpow: assumes $a \subseteq b^+$
shows $(x,y) \in a^\cdot n \implies \exists m. m \geq n \land (x,y) \in b^\cdot m$

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 $xy: (x,y) \in a^\cdot n$ and $yz: (y,z) \in a$ by auto
  from $Suc(1)(OF xy)$ obtain $m$ where $m: m \geq n$ and $xy: (x,y) \in b^\cdot m$ by auto
  from $yz$ assms have $(y,z) \in b^\cdot (m + k)$ unfolding relpow-add by auto
  with $k$ $m$ show $?case$ by (intro exI [of - $m + k$], auto)
qed

lemma relpow-image: assumes $f: \:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\\
assumes sn: SN-on R {a}
and IH: \( \forall x. \text{SN-on R } \{ x \} \Rightarrow [\forall y. (x, y) \in R \Rightarrow P y] \Rightarrow P x \)
shows \( P a \)

proof
from sn SN-on-cone-acc [of \( R^{-1} \) a] have a: a \in \text{termi } R by auto

show \( \ldots \)
proof (rule Wellfounded.acc.induct [OF a, of P], rule IH)
fix x
assume \( \forall y. (y, x) \in R^{-1} \Rightarrow y \in \text{termi } R \)
from this [folded SN-on-cone-acc]
show SN-on-R \( \{ x \} \) by simp fast
qed auto

qed

lemma partially-localize-CR:
\( CR r \iff (\forall x y z. (x, y) \in r \wedge (x, z) \in r^* \rightarrow (y, z) \in \text{join } r) \)

proof
assume CR r
thus \( \forall x y z. (x, y) \in r \wedge (x, z) \in r^* \rightarrow (y, z) \in \text{join } r \) by auto

next
assume 1: \( \forall x y z. (x, y) \in r \wedge (x, z) \in r^* \rightarrow (y, z) \in \text{join } r \)
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 \( (a, c) \in r^m \) using rtrancl-is-UN-relpow by fast
with 2 3 have \( (b, c) \in \text{join } r \)
proof (induct n arbitrary: a b c)

case 0 thus \( \ldots \) by auto

next
case (Suc m)
from Suc(4) obtain d where 5: \( (a, d) \in r^m \) and 6: \( (d, c) \in r \) by auto
from Suc(1) [OF Suc(2) Suc(3) ad] have \( (b, d) \in \text{join } r \).
with 1 6 have \( (b, c) \in \text{join } r \)
proof (induct n arbitrary: a b c)

case 0 thus \( \ldots \) by metis

qed

qed

definition strongly-confluent-on :: 'a rel \Rightarrow 'a set \Rightarrow bool
where
\( \text{strongly-confluent-on } r A \leftrightarrow (\forall x \in A. \forall y z. (x, y) \in r \wedge (x, z) \in r \rightarrow (\exists u. (y, u) \in r^* \wedge (z, u) \in r^*)) \)

abbreviation strongly-confluent :: 'a rel \Rightarrow bool
where
\( \text{strongly-confluent } r \equiv \text{strongly-confluent-on } r \text{ UNIV} \)

lemma strongly-confluent-on-E11:

64
\[
\exists u. \ (y, u) \in r^* \land (z, u) \in r^=
\]

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

**lemma** strongly-confluentI [intro]:
\[ [\forall x \ y \ z. \ (x, y) \in r \Rightarrow (x, z) \in r \Rightarrow \exists u. \ (y, u) \in r^* \land (z, u) \in r^=] \Rightarrow \text{strongly-confluent } r \]

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

**lemma** strongly-confluent-E1n:
assumes scr: strongly-confluent r
shows \((x, y) \in r^= \Rightarrow (x, z) \in r^* \Rightarrow \exists u. \ (y, u) \in r^* \land (z, u) \in r^=\)
proof (induct n arbitrary: \(x \ y \ z\))
  case (Suc \(m\))
  from Suc(3) obtain \(w\) where \(xw: (x, w) \in r^m\) 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 \[ OF \ scr, \ of \ wz \ yu \ wz \] \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 \((x, y) \in r \Rightarrow (x, z) \in r^* \Rightarrow (y, z) \in \text{join } r\)
      **by** (cases \(x = y\), insert strongly-confluent-E1n \[ OF \ assms, \ blast+\]) \}
  then show \(\text{CR } r\) **using** partially-localize-CR **by** blast
qed

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

**lemma** NF-imp-SN-on: \(a \in NF R \Rightarrow SN-on R \{a\}\) **unfolding** SN-on-def NF-def **by** blast

**lemma** Union-sym: \((s, t) \in \bigcup i \leq n. \ (S \ i)^* \iff (t, s) \in \bigcup i \leq n. \ (S \ i)^*\) **by** auto

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

**lemma** CR-NF-conv:
assumes \(\text{CR } r\) and \(t \in NF r\) and \((u, t) \in r^***\)
shows \((u, t) \in r^1\)
using assms
**unfolding** CR-imp-conversionIff-join \[ OF \ :CR \ r\] **by** (auto simp: NF-iff-no-step normalizability-def)
lemma NF-join-imp-reach:
  assumes y ∈ NF A and (x, y) ∈ A↓
  shows (x, y) ∈ A∗
using assms by (auto simp: join-def) (metis NF-not-suc rtrancl-converseD)

lemma conversion-O-conversion [simp]:
  A*** O A*** = A***
  by (force simp: converse-def)

lemma trans-O-iff: trans A ᵗ A O A ⊆ A unfolding trans-def by auto
lemma refl-O-iff: refl A ᵗ Id ⊆ A unfolding refl-on-def by auto

lemma relpow-Suc: r ᵗ Suc n = r O r ᵗ n
proof (induct n)
case (Suc n)
  show ?case unfolding relpow.simps(2)[of - r ᵗ - n]
  by (simp add: Suc converse-relcomp)
qed simp

lemma conversion-mono: A ⊆ B ⇒ A*** ⊆ B***
  by (auto simp: conversion-def intro: rtrancl-mono)

lemma conversion-conversion-idemp [simp]: (A***)'*** = A***
by auto

lemma lower-set-imp-not-SN-on:
  assumes s ∈ X ∀ t ∈ X. ∃ u ∈ X. (t, u) ∈ R shows ¬ SN-on R {s}
by (meson SN-on-imp-on-minimal assms)

lemma SN-on-Image-rtrancl-iff[simp]: SN-on R (R* ᵗ X) ←→ SN-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 ⊆ R’ ⇒ S O R ⊆ S O R’ by auto
lemma O-mono2: R ⊆ R’ ⇒ R O T ⊆ 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) ∈ (S O R)* O S
then obtain \( n \) where \((x, y) \in (S \circ O R)^n \circ O S\) by blast
then show \((x, y) \in S \circ O (R \circ O S)^*\)
proof (induct \( n \) arbitrary: \( y \))
case \( IH: (\text{Suc } n) \)
then obtain \( z \) where \((x, z) \in (S \circ O R)^n \circ O S\) and \((z, y) \in R \circ O S\) by auto
from \( IH \).hyps[OF \( xz \)] \( zy \) have \((x, y) \in S \circ O (R \circ O S)^* \circ O S\) by auto
then show ?case by (fold trancl-unfold-right, auto)
qed auto
next
fix \( x \, y \)
assume \((x, y) \in S \circ O (R \circ O S)^*\)
then obtain \( n \) where \((x, y) \in (S \circ O R)^n \circ O S\) by blast
then show \((x, y) \in (S \circ O (R \circ O S)^* \circ O S\) by auto
proof (induct \( n \) arbitrary: \( y \))
case \( IH: (\text{Suc } n) \)
then obtain \( z \) where \((x, z) \in (S \circ O R)^n \circ O S\) and \((z, y) \in R \circ O S\) by auto
from \( IH \).hyps[OF \( xz \)] \( zy \) have \((x, y) \in S \circ O (R \circ O S)^* \circ O S\) by auto
then show ?case by (fold trancl-unfold-right, auto)
qed auto
qed

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

lemma \( SN-on-subset-SN-terms\):
assumes \( SN: SN-on R X \) shows \( X \subseteq \{ x. \, SN-on R \{ x \}\} \)
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 auto

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

lemma \( SN-on-UN\):
assumes \( \forall x. \, SN-on R (X \times X) \) shows \( SN-on R (\bigcup x. \, X x)\)
using \( \text{assms} \) by fast

lemma \( Image-subsetI\): \( R \subseteq R' \Longrightarrow R \cup X \subseteq R' \cup X \) by auto

lemma \( SN-on-O-comm\):
assumes \( SN: SN-on ((R :: (\'a \times \'b) \ set) \circ (S :: (\'b \times \'a) \ set)) \) \( (S \cup X) \)
shows \( SN-on (S \circ O R) X\)
proof
fix \( \text{seq :: nat} \Rightarrow \ 'b \) assume \( \text{seq0}: \, \text{seq} \ 0 \in X \) and \( \text{chain: chain (S O R) seq}\)

67
from SN have SN: SN-on (R O S) ((R O S)+ " S " X) by simp
\{ fix i a \
assume ia: (seq i,a) ∈ S and aSi: (a,seq (Suc i)) ∈ 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 ia aSi have b ∈ ((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.prems 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

country 
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 \(s t. (s,t) ∈ S ^* n O R \Rightarrow (s,t) ∈ R O S^* \)
  proof (induct n) 
    case (Suc n) 
  qed 
\}
then obtain $u$ where $(s,u) \in S$ $(u,t) \in R \ O \ S^*$ unfolding relpow-Suc by blast

then have $(s,t) \in S \ O \ R \ O \ S^*$ by auto
also have $\ldots \subseteq R \ O \ S^* \ O \ S^*$ using push by blast
also have $\ldots \subseteq R \ O \ S^*$ by auto
finally show $?case.

qed auto

thus $?thesis$ by blast

qed

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

proof (intro equalityI subrelI)
fix $x$ $y$
assume $(x,y) \in (S \cup R)^*$
also have $\ldots \subseteq (S^* \ O \ R)^* \ O \ S^*$ by regexp
finally obtain $z$ where $xz$ $(x,z) \in (S^* \ O \ R)^*$ and $zy$ $(z,y) \in S^*$ by auto
from $xz$ have $(x,z) \in R^* \ O \ S^*$
proof (induct rule: $rtrancl-induct$)

case (step $z$ $w$)
then have $(x,w) \in R^* \ O \ S^* \ O \ S^* \ O \ R$ by auto
also have $\ldots \subseteq R^* \ O \ S^* \ O \ R$ by regexp
also have $\ldots \subseteq R^* \ O \ R \ O \ S^*$ using $rtrancl-O-push$ by auto
also have $\ldots \subseteq R^* \ O \ S^*$ by regexp
finally show $?case.

qed auto

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

lemma $SN-on-O-push$:
assumes $SN$: $SN-on \ R \ X$ shows $SN-on \ (R \ O \ S^*) \ X$

proof
fix $seq$
have $SN$: $SN-on \ R \ (R^* \ X)$ using $SN-on-Image-rtrancl[OF $SN$].
moreover assume $seq \ (\emptyset::nat) \in X$
then have $seq \ 0 \in R^* \ X$ by auto
ultimately show chain $(R \ O \ S^*) \ seq \Rightarrow False$

proof (induct $seq \ 0$ arbitrary; seg rule: $SN-on-induct$)

case IH
then have $01$: $(seq \ 0, \ seq \ 1) \in R \ O \ S^*$
and $12$: $(seq \ 1, \ seq \ 2) \in R \ O \ S^*$
and $23$: $(seq \ 2, \ seq \ 3) \in R \ O \ S^*$ 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 $(s,t) \in S^* \ O \ R$ by auto
with $rtrancl-O-push$ have $st$: $(s,t) \in R \ O \ S^*$ by auto
from $t2 \ 23$ have $(t, \ seq \ 3) \in S^* \ O \ R \ O \ S^*$ by auto
also from $rtrancl-O-push$ have $\ldots \subseteq R \ O \ S^* \ O \ S^*$ by blast
finally have t3: (t, seq 3) ∈ R O S* by regexp
let ?seq = λi. case i of 0 ⇒ s | Suc 0 ⇒ t | i ⇒ seq (Suc i)
show ?case
proof (rule IH)
from s show (seq 0, ?seq 0) ∈ R by auto
show chain (R O S*) ?seq
proof (intro allI)
  fix i show (?seq i, ?seq (Suc i)) ∈ 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

lemma SN-on-Image-push:
  assumes SN: SN-on R X shows SN-on R (S* " X)
proof –
  { fix n
    have SN-on R ((S^n) " 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 ^ n) " X).
      from SN-on-Image[OF this]
      have SN-on (R O S*) ((R O S*) " (S ^ n) " X).
      then have SN-on R ((R O S*) " (S ^ n) " X) by (rule SN-on-mono, auto)
      from SN-on-subset2[OF Image-mono[OF push subset-refl] this]
      have SN-on R (R " (S ^ Suc n) " 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 ∈ X =⇒ chain R f =⇒ ¬ 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 ∪ B) = Id-on A ∪Id-on B unfolding Id-on-def
  by auto

lemma relpow-union-cases: (a,d) ∈ (A ∪ B) ^ n =⇒ (a,d) ∈ B ^ n ∨ (∃ b c k m.
\[(a, b) \in B \setminus k \land (b, c) \in A \land (c, d) \in (A \cup B) \setminus m \land n = \text{Suc} \ (k + m)\]

**proof** *(induct \(n\) arbitrary: \(a\) \(d)*)

**case** *(Suc \(n\) \(a\) \(e)*)

let \(?AB = A \cup B*

from \(\text{Suc}(2)\) obtain \(b\) where \(ab: (a, b) \in ?AB\) and \(be: (b, e) \in ?AB \setminus n\) by (rule relpow-Suc-E2)

from \(ab\)
show \(?case\)
proof
assume \((a, b) \in A\)
show \(?thesis\)
proof (rule disjI2, intro exI conjI)
show \(\text{Suc} \ n = \text{Suc} \ (0 + n)\) by simp
show \((a, b) \in A\) by fact
qed (insert \(be\), auto)

next
assume \(ab: (a, b) \in B\)
from \(\text{Suc}(1)[OF be]\)
show \(?thesis\)
proof
assume \((b, c) \in B \setminus n\)
with \(ab\) show \(?thesis\)
by (intro disjI1 relpow-Suc-I2)

next
assume \(\exists\ c \ d \ k \ m. (b, c) \in B \setminus k \land (c, d) \in A \land (d, e) \in ?AB \setminus m \land n = \text{Suc} \ (k + m)\)

then obtain \(c \ d \ m \ n\) where \((b, c) \in B \setminus k\) and \(\ast: (c, d) \in A\) \((d, e) \in ?AB\)

with \(ab\) have \(ac: (a, c) \in B \setminus (\text{Suc} \ k)\) by (intro relpow-Suc-I2)
show \(?thesis\)
by (intro disjI2 exI conjI, rule \(ac\), (rule \(\ast\))+, simp add: \(\ast\))
qed
qed
qed

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

assumes **trans** \(r\) refl \(r\)

shows \(r^* = r\)

**proof**

show \(r^* \subseteq r\)

**proof**

fix \(x\) \(y\)

assume \((x, y) \in r^*\)
thus \((x, y) \in r\)
by (induct, insert assms, unfold refl-on-def trans-def, blast+)

qed

qed

lemma **trans-refl-imp-O-id**:
assumes trans r refl r
shows r O r = r
proof
  show r O r ⊆ r by (fact trans-O-subset[OF assms(1)])
  have r ⊆ r O Id by auto
  moreover have Id ⊆ r by (fact assms(2)[unfolded refl-O-iff])
  ultimately show r ⊆ r O r by auto
qed

lemma relcomp3-I:
  assumes (t, u) ∈ A and (s, t) ∈ B and (u, v) ∈ B
  shows (s, v) ∈ B O A O B
  using assms by blast

lemma relcomp3-transI:
  assumes trans B and (t, u) ∈ B O A O B and (s, t) ∈ B and (u, v) ∈ B
  shows (s, v) ∈ B O A O B
  using assms by (auto simp: trans-def intro: relcomp3-I)

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

lemma qc-SN-relto-iff:
  assumes r O s ⊆ s O (s ∪ r)*
  shows SN (r* O s O r*) = SN s
proof
  from converse-mono [THEN iffD2 , OF assms]
  have *: s⁻¹ O r⁻¹ ⊆ (s⁻¹ ∪ r⁻¹)* O s⁻¹ unfolding converse-inward .
  have (r* O s O r*)⁻¹ = (r⁻¹)* O s⁻¹ O (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**)* = r** 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) $\text{relto} :: \langle a \Rightarrow a \Rightarrow a \Rightarrow a \rangle$

$\text{relto } R S \equiv S^* \cup O R O S^*$

**Definition** $\text{SN-rel-on} :: \langle a \Rightarrow a \Rightarrow a \Rightarrow a \Rightarrow bool \rangle$

$\text{SN-rel-on } R S \equiv \text{SN-on (relto } R S)$

**Definition** $\text{SN-rel-on-alt} :: \langle a \Rightarrow a \Rightarrow a \Rightarrow a \Rightarrow bool \rangle$

$\text{SN-rel-on-alt } R S T = (\forall f. \text{chain } (R \cup S) f \wedge f 0 \in T \longrightarrow \neg (\text{INF } j. (f j, f (\text{Suc } j)) \in R))$

**Abbreviation** $\text{SN-rel} :: \langle a \Rightarrow a \Rightarrow a \Rightarrow a \Rightarrow bool \rangle$

$\text{SN-rel } R S \equiv \text{SN-rel-on } R S \cup \text{UNIV}$

**Abbreviation** $\text{SN-rel-alt} :: \langle a \Rightarrow a \Rightarrow a \Rightarrow a \Rightarrow bool \rangle$

$\text{SN-rel-alt } R S \equiv \text{SN-rel-on-alt } R S \cup \text{UNIV}$

**Lemma** $\text{relto-absorb [simp]}: \text{relto } R E O E^* = \text{relto } R E E^* O \text{ relto } R E = \text{relto } R E$

**Using** $O$-assoc and $\text{rtrancl-idemp-self-comp}$ by (metis)+

**Lemma** $\text{steps-preserve-SN-on-relto}$:

**Assumes** steps: $(a, b) \in (R \cup S)^*$

**And** $\text{SN: } \text{SN-on (relto } R S) \{a\}$

**Shows** $\text{SN-on (relto } R S) \{b\}$

**Proof** –

**Let** $?RS = \text{relto } R S$

**Have** $(R \cup S)^* \subseteq S^* \cup ?RS^*$ by regexp

**With** steps have $(a,b) \in S^* \lor (a,b) \in ?RS^*$ 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** $f 0 = b$ and steps: $\forall i. (f i, f (\text{Suc } i)) \in ?RS$ by auto

**Let** $?g = \lambda i. \text{if } i = 0 \text{ then } a \text{ else } f i$

**Have** $\neg \text{SN-on } ?RS \{a\}$ unfolding $\text{SN-on-def not-not}$

**Proof** (rule ext[of $?a$], unfolding $\text{SN-on-def not-not}$)

**Fix** $i$

**Show** $(?g i, ?g (\text{Suc } i)) \in ?RS$

**Proof** (cases $i$)

**Case** $(\text{Suc } j)$

**Show** $?thesis$ using steps[of $i$] unfolding $\text{Suc}$ by simp

next
case 0
  from steps[of 0, unfolded f0] Ssteps have steps: (a, f (Suc 0)) \in S^* O

\textit{RS} by blast
  have (a, f (Suc 0)) \in \textit{RS}
  by (rule subsetD[OF - steps], regexp)
  thus \textit{thesis} unfolding \theta by simp
  qed
  qed simp
  with \textit{SN} show False by simp
  qed

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

\textit{lemma} SN-rel-on-imp-SN-rel-on-alt: SN-rel-on R S T \implies SN-rel-on-alt R S T
\textit{proof} (unfold SN-rel-on-def)
  assume \textit{SN}: SN-on (relto R S) T
  show \textit{?thesis}
  proof (unfold SN-rel-on-alt-def, intro allI impI)
    fix f
    assume steps: chain (R \cup S) f \land f 0 \in T
    with \textit{SN} have \textit{SN}: \textit{SN-on} (relto R S) {f 0}
    and steps: \( \forall i. (f i, f (Suc i)) \in R \cup S \) unfolding \textit{SN-defs} by auto
    obtain r where r: \( \forall j. r j \equiv (f j, f (Suc j)) \in R \) by auto
    show \( \neg (INFM j. (f j, f (Suc j)) \in R) \)
    proof (rule ccontr)
      assume \( \neg \textit{?thesis} \)
      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]
      infinitely-many.index-not-p-between[OF ih]
      have r-index: \( \forall i. r (r-index i) \land r-index i < r-index (Suc i) \land (\forall j. r-index j < r-index (Suc i) \implies \neg r j) \) by auto
      obtain g where g: \( \forall i. g i \equiv f (r-index i) \) ..
      { 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 and rsi: rsi = ?rsi by auto
        with r-index[of i] steps have inter: \( \forall j. ri < j \land j < rsi \implies (f j, f (Suc j)) \in S \) unfolding r by auto
        from ri isi rsi have rsi: 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 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)}
  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


lemma SN-rel-on-alt-imp-SN-rel-on-alt-imp-SN-rel-on-alt: SN-rel-on-alt R S T ⟷ SN-rel-on R S T

proof (unfold SN-rel-on-alt)
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 ai. (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)

  75
{ 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) 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 n 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 trancl-fan-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 \(g \circ f \in T\) have \(h_0 \in T\) unfolding start by auto

\{
    fix \(i\)
    have \((h_i, h (\text{Suc} i)) \in R \cup S\)
    proof (cases \(i < n\))
        case True
        from steps[of \(i\)] \(g \circ f \in T\) True show \(?thesis\) by auto
    next
        case False
        hence \(i = n + (i - n)\) by auto
        then obtain \(k\) where \(i = n + k\) by auto
        from gsteps[of \(k\)] show \(?thesis\) unfolding \(h\) by simp
    qed
\}

note hsteps = this

from \(\text{SN-unfolded SN-rel-on-alt-def, rule-format, OF conjI[OF allI[OF hsteps] h0]}\)
have \(?INFM j. (h_j, h (\text{Suc} j)) \in R\).
moreover have \(?INFM j. (h_j, h (\text{Suc} j)) \in R\) unfolding \(\text{INFM-nat-le}\)
proof (rule)
    fix \(m\)
    from infR[unfolded \(\text{INFM-nat-le}\), rule-format, of \(m\)]
    obtain \(i\) where \(i \geq m\) and \(g\): \((g \circ f \in T\) by auto
    show \(?\exists n \geq m. (h_n, h (\text{Suc} n)) \in R\)
      by (rule exI[of - \(i + n\)], unfold \(h\), insert \(g \circ f\), auto)
    qed
ultimately show \(?thesis\) ..
qed
qed

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

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

lemma \(\text{SN-rel-on-alt-r-empty}: \text{SN-rel-on-alt} \{\} S T\)
unfolding \(\text{SN-rel-defs}\) by auto

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

lemma \(\text{SN-rel-on-mono'}:\)
assumes \(R: R \subseteq R'\) and \(S: S \subseteq R' \cup S'\) and \(\text{SN}: \text{SN-rel-on} R' S' T\)
shows \(\text{SN-rel-on} R S T\)
proof
    note conv = \(\text{SN-rel-on-conv SN-rel-on-alt-def INFM-nat-le}\)
    show \(?thesis\) unfolding \(\text{conv}\)
      proof (intro allI impI)
      fix \(f\)

78
assume chain \((R \cup S) f \land f 0 \in T\)
with \(R S\) have chain \((R' \cup S') f \land f 0 \in T\) by auto
from SN[unfolded conv, rule-format, OF this]
show \(\lnot (\forall m. \exists n \geq m. (f n, f (Suc n)) \in R)\) 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 \(SN: SN-rel-on R' S' T\)
shows \(SN-rel-on R S T\)
using \(SN\)
unfolding \(SN-rel-on-def\) using \(SN-on-mono[OF - relto-mono[OF R S]]\) by blast

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

lemma SN-rel-on-imp-SN-on:
assumes \(SN-rel-on R S T\)
shows \(SN-on R T\)
proof
fix \(f\)
assume \(chain R f\)
and \(f0: f 0 \in T\)

hence \(\land i. (f i, f (Suc i)) \in relto R S\) 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 \(SN-rel-on R (S \cup Id) T = SN-rel-on R S T\)
unfolding \(SN-rel-on-def\) by (simp only: relto-Id)

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

lemma SN-rel-on-ideriv: \(SN-rel-on R S T = (\lnot (\exists as. ideriv R S as \land as 0 \in T)) (is ?L = ?R)\)
proof
assume \(?L\)
show \(?R\)
proof
assume \(\exists as. ideriv R S as \land as 0 \in T\)
then obtain \(as\) where \(id: ideriv R S as\) and \(T: as 0 \in T\) by auto
note \(id = id[unfolded ideriv-def]\)
\textbf{from} \(?L[unfolded\ SN-rel-on-conv\ SN-rel-on-alt-def,\ THEN\ spec[of\ -\ as]]\)
\textbf{id} \(T\) \textbf{obtain} \(i\ where\ i: \land j.\ j \geq i \implies (as\ j,\ as\ (Suc\ j)) \notin R\) \textbf{by} \textbf{auto}
\textbf{with} \textbf{id}[unfolded\ INFM-nat,\ THEN\ conjunct2,\ THEN\ spec[of\ -\ Suc\ i]] \textbf{show}
\textbf{False} \textbf{by} \textbf{auto}
\textbf{qed}
\textbf{next}
\textbf{assume} \(?R\)
\textbf{show} \(?L\)
\textbf{unfolding} \(SN-rel-on-conv\ SN-rel-on-alt-def\)
\textbf{proof}\ (\textbf{intro allI impI})
\textbf{fix} \(f\)
\textbf{presume} \textbf{steps:} \textbf{chain} \((R \cup S)\) \(f\)
\textbf{obtain} \(r\ \textbf{where}\ r: \land j.\ r j \equiv (f\ j,\ f\ (Suc\ j)) \in R\) \textbf{by} \textbf{auto}
\textbf{show} \((INFM\ j.\ (f\ j,\ f\ (Suc\ j)) \in R)\)
\textbf{proof}\ (\textbf{rule ccontr})
\textbf{assume} \textbf{¬} \(?thesis\)
\textbf{hence} \(ih:\ \text{infinitely-many}\ r\ \textbf{unfolding}\ \text{infinitely-many-def}\ r\ \textbf{by}\ \textbf{blast}\)
\textbf{obtain} \(r-index\ \textbf{where}\ \text{r-index} = \text{infinitely-many-index}\ r\ \textbf{by}\ \text{simp}\)
\textbf{with} \textbf{infinitely-many-index-p}[\text{OF}\ ih] \text{infinitely-many-index-ordered}[\text{OF}\ ih]
\textbf{infinitely-many-index-not-p-between}[\text{OF}\ ih]
\textbf{have} \textbf{r-index:} \land i,\ r\ \text{r-index\ i} \land \text{r-index\ i} < \text{r-index\ (Suc\ i)} \land (\forall\ j,\ \text{r-index\ i} < j \land j < \text{r-index\ (Suc\ i)} \implies \textbf{¬}\ r\ j)\ \textbf{by}\ \textbf{auto}\)
\textbf{obtain} \(g\ \textbf{where}\ g: \land i,\ g\ i \equiv f\ \text{r-index\ i}\ ..\)
\{\textbf{fix} \(i\)
\textbf{let} \(?ri = r-index\ i\)
\textbf{let} \(?rsi = r-index\ (Suc\ i)\)
\textbf{from} \textbf{r-index\ have} \(isi:\ ?ri < ?rsi\ \textbf{by}\ \textbf{auto}\)
\textbf{obtain} \(ri\ rsi\ \textbf{where}\ ri: r = ?ri\ \textbf{and}\ rsi: rsi = ?rsi\ \textbf{by}\ \textbf{auto}\)
\textbf{with} \textbf{r-index}[\text{OF}\ i] \textbf{steps} \textbf{have} \textbf{inter:} \land j,\ ri < j \land j < rsi \implies (f\ j,\ f\ (Suc\ j)) \in S\ \textbf{by}\ \textbf{auto}\)
\textbf{from} \(ri\ rsi\ \textbf{have} \textbf{risi:} ri < rsi\ \textbf{by}\ \textbf{simp}\)
\{
lemma SN-rel-alt-to-SN-rel : SN-rel-alt R S → 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 : λ 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. ∃ ai bi. ?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
  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+
  qed
}
from $S$[unfolded $rtrancl$-list-conv] obtain $li$ where last $(a \ i \ # \ li) = b \ (\text{Suc } i) \land \forall j < \text{length } li, ((a \ i \ # \ li) \ # j, \ (a \ i \ # \ li) \ # \text{Suc } j) \in S$ ..
with $R$ have $\text{prop } i \ li$ by blast
hence $\exists \ li. \ \text{prop } i \ li$ ..

hence $\forall i. \ \exists \ li. \ \text{prop } i \ li$ ..
from choice[OF this] obtain $l$ where steps: $\land i. \ \text{prop } i \ (l \ i)$ by auto
let $?p = \lambda i. \ \text{prop } i \ (l \ i)$ from steps have steps: $\land i. \ ?p \ i$ by blast
let $?l' = \lambda i. \ \text{length } (?l \ i)$
let $?g = \lambda i. \ \text{inf-concat-simple } ?l' \ i$
obtain $g$ where $g: \land i. \ g \ i = (\text{let } (ii, jj) = (?g \ i \in ?l \ ii) \ # jj) \ by$ auto
have len: $\land i j n. \ g \ n = (i,j) \implies j < \text{length } (?l \ i)$
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 $\text{thesis by simp$
next
  case $\text{Suc } n$
  obtain $ii \ jj$ where $nn: \ ?g \ nn = (ii, jj)$ by (cases $\ ?g \ nn, \ auto$)
  show $\text{thesis}
  proof (cases $\text{Suc } jj < \text{length } (?l \ ii)$
    case True
      with $nn$ have $\ ?g \ n = (ii, \text{Suc } jj)$ by auto
      with $n$ True show $\text{thesis by simp$
next
    case False
      with $nn$ have $\ ?g \ n = (\text{Suc } ii, 0)$ by auto
      with $n$ show $\text{thesis by simp$
qed
qed

have $\text{gsteps: } \land i. \ (g \ i, \ g \ (\text{Suc } i)) \in R \cup S$
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 $\ ?g n = (?l \ i \ # j \ and \ gsn: \ ?g \ (\text{Suc } n) = (?l \ i \ # (\text{Suc } j) \ unfolding \ g \ by \ auto
        thus $\text{thesis using steps[of i] True by auto$
next

82
\begin{document}

\begin{enumerate}
\item \textbf{case} \textit{False} \\
\textbf{with} \textit{n} \textbf{have} \( ?g \) \((\text{Suc} \ n) = (\text{Suc} \ i, 0) \) \textbf{by} auto \\
\textbf{with} \textit{n} \textbf{have} \( gn = g n = ?l i j \) \textit{and} \( gsn = g (\text{Suc} \ n) = a (\text{Suc} \ i) \) \\
\textbf{unfolding} \textit{g} \textbf{by} auto \\
\textbf{from} \( gm \) \textbf{len}[\text{OF} \ n] \textbf{False} \textbf{have} \( j = \text{length} (\forall i) - 1 \) \textbf{by} auto \\
\textbf{with} \textit{gm} \textbf{have} \( gn = g n = \text{last} (\forall i) \) \textbf{using} \textit{last-cone-nth}[\text{OF} \ i] \textbf{by} auto \\
\textbf{from} \( gm gsn \) \textbf{show} \( \textit{thesis} \) \textbf{using} \textit{steps}[\text{OF} \ i] \textbf{steps}[\text{OF} \ Suc \ i] \textbf{by} auto \\
\textbf{qed} \\
\textbf{qed} \\
\textbf{have} \textit{infR: INFM \ j.} \ (g j, g (Suc \ j)) \in \textit{R} \textbf{unfolding} \textit{INFM-nat-le} \\
\textbf{proof} \\
\textbf{fix} \textit{n} \\
\textbf{obtain} \textit{i j where} \textit{n:} \ ?g \ n = (i,j) \textbf{by} (\textit{cases} \ ?g \ n, \textit{auto}) \\
\textbf{from} \textit{len}[\text{OF} \ n] \textbf{have} \( j: j < ?l i \) \\
\textbf{let} \ ?k = ?l i - 1 - j \\
\textbf{obtain} \textit{k where} \textit{k:} \ k = j + ?k \textbf{by} auto \\
\textbf{from} \textit{j k have} \ k2; \ k = ?l i - 1 \textbf{and} \ k3; \ j + ?k < ?l i \textbf{by} auto \\
\textbf{from} \textit{inf-concat-simple-add}[\text{OF} \ n, \textit{OF} \ k, \textit{OF} \ k3] \textit{have} \textit{gk:} \ ?g \ (n + ?k) = (i, k) \textbf{by} (\textit{simp only:} \ k) \\
\textbf{hence} \ (n + ?k) = ?l i + k \textbf{unfolding} \textit{g} \textbf{by} auto \\
\textbf{hence} \ ?gk2; \ ?g \ (n + ?k) = \textit{last} (\forall i) \textbf{using} \textit{last-conv-nth}[\text{OF} \ ?l i] \textit{k2} \textbf{by} auto \\
\textbf{from} \textit{k2 gk have} \ ?g \ (\text{Suc} \ (n+?k)) = (\text{Suc} \ i, 0) \textbf{by} auto \\
\textbf{hence} \ ?gsk2; \ ?g \ (\text{Suc} \ (n+?k)) = a (\text{Suc} \ i) \textbf{unfolding} \textit{g} \textbf{by} auto \\
\textbf{from} \textit{steps}[\text{OF} \ i] \textbf{steps}[\text{OF} \ Suc \ i] \textbf{have} \textit{main:} \ (g (n+?k), g (\text{Suc} \ (n+?k))) \in \textit{R} \\
\textbf{by} (\textit{simpl only:} \ ?gk2 \ ?gsk2) \\
\textbf{show} \ \exists \ j \geq n. \ ((g j, g (Suc \ j)) \in \textit{R} \\
\textbf{by} (\textit{rule cxI[of - n + ?k], auto simp: main[simplified]}) \\
\textbf{qed} \\
\textbf{from} \textit{SN[unfolded \ SN-rel-on-alt-def]} \textit{gsteps infR} \textbf{show} \textit{False} \textbf{by} blast \\
\textbf{qed} \\
\textbf{qed} \\
\textbf{lemma} \textit{SN-rel-alt-r-empty} : \textit{SN-rel-alt {} S} \\
\textbf{unfolding} \textit{SN-rel-defs} \textbf{by} auto \\
\textbf{lemma} \textit{SN-rel-alt-s-empty} : \textit{SN-rel-alt R {} = SN R} \\
\textbf{unfolding} \textit{SN-rel-defs \ SN-defs} \textbf{by} auto \\
\textbf{lemma} \textit{SN-rel-mono\'}: \\
R \subseteq R' \Rightarrow S \subseteq R' \cup S' \Rightarrow \textit{SN-rel} R' S' \Rightarrow \textit{SN-rel} R S \\
\textbf{unfolding} \textit{SN-rel-on-conv \ SN-rel-defs \ INFM-nat-le} \\
\textbf{by} (\textit{metis contru-subsetD sup.left-idem sup.mono}) \\
\textbf{lemma} \textit{SN-rel-mono} : \\
\textbf{assumes} \textit{R:} \ R \subseteq R' \textbf{and} \textit{S:} \ S \subseteq S' \textbf{and} \textit{SN:} \ \textit{SN-rel} R' S' \\
\textbf{shows} \textit{SN-rel} R S \\
\textbf{using} \textit{SN} \textbf{unfolding} \textit{SN-rel-defs \ using} \textit{SN-subset[OF - relto-mono[OF \ R \ S]}]) \textbf{by} blast \\
\end{enumerate}

\end{document}
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 \( \bigwedge i. (f i, f (Suc i)) \in \text{relto } R S \) by blast
  thus False using assms unfolding SN-rel-defs SN-defs by fast
qed

lemma relto-trancl-conv : (relto R S)\(^*\) = ((R \cup S))\(^*\) O R O ((R \cup S))\(^*\) 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\(^*\)) = 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 = (\( \neg (\exists as. \text{ideriv } R S as) \)) (is \( \_L = \_R \))
proof
  assume \_L
  show \_R unfolding SN-rel-on-conv SN-rel-defs
  proof (intro allI impI)
    fix as
    presume chain (R \cup S) as
    with \_R[unfolded ideriv-def] have \( \neg (\text{INFM } i. (as i, as (Suc i)) \in R) \) 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 \( \neg (\text{INFM } j. (as j, as (Suc j)) \in R) \) unfolding INFM-nat using i by blast
  qed simp
next
  assume \_R
  show \_L unfolding SN-rel-on-conv SN-rel-defs
  proof (intro allI implI)
    fix as
    presume chain (R \cup S) as
    with \_L[unfolded ideriv-def] have \( \neg (\text{INFM } i. (as i, as (Suc i)) \in R) \) by auto
    from this[unfolded INFM-nat] obtain i where i: \( \bigwedge j. i \geq j \implies (as j, as (Suc j)) \notin R \) by auto
    show \( \neg (\text{INFM } j. (as j, as (Suc j)) \in R) \) unfolding INFM-nat using i by blast
  qed simp

84
lemma \textit{SN-rel-map}:
fixes $R$, $R'$, $Rw$, $\cdot'$ :: 'a rel
defines $A$ :: $\equiv R' \cup Rw'$
assumes $SN$ :: $SN$-$rel$ $R'$, $Rw'$
and $R$ :: $\forall s \ t. \ (s, t) \in R \Longrightarrow (f s, f t) \in A \ast O R' O A \ast$
and $Rw$ :: $\forall s \ t. \ (s, t) \in Rw \Longrightarrow (f s, f t) \in A \ast$
shows $SN$-$rel$ $R$, $Rw$
unfolding $SN$-$rel$-$defs$
proof
fix $g$
assume steps :: chain (relto $R$, $Rw$) $g$
let $\equiv f i = \lambda i. \ (f s, f t)$
obtain $h$ where $h = \equiv f$ by auto

\{ fix $i$
let $\equiv m = \lambda (x, y). \ (f x, f y)$
\{ fix $s \ t$
assume $(s, t) \in Rw \ast$
hence $\equiv m (s, t) \in A \ast$
proof (induct)
  case base show $\equiv m (s, t) \in A \ast$
  by simp
next
  case (step $t \ u$
  from $Rw[\OF step(2)]$ step(3)
  show $\equiv m (s, t) \in A \ast$
  by auto
\} note $Rw = this$
from steps have $(g i, g (Suc i)) \in relto R$, $Rw$ ..
from this
obtain $s \ t$ where $gs$ :: $(g i, s) \in Rw \ast$ and $st$ :: $(s, t) \in R$ and $tg$ :: $(t, g (Suc i))$
\mathrel{\in Rw \ast}$ by auto
from $Rw[\OF gs] R[\OF st] Rw[\OF tg]$
have step :: $\equiv m (Suc i) \in A \ast O (A \ast O R' O A \ast) O A \ast$
  by fast
have $\equiv m (Suc i) \in A \ast O R' O A \ast$
  by (rule subsetD[OF - step], regexp)
hence $h i, h (Suc i) \in (relto R' Rw') \ast$
  unfolding $A h$ relto-trancl-conv .
\}
hence $\sim SN ((relto R' Rw') \ast)$ by auto
with $SN$-$imp$-$SN$-$trancl$[OF $SN$[unfolded $SN$-$rel$-$on-def$]]
  show False by simp
qed

datatype $SN$-$rel$-$ext$-type = top-s | top-ns | normal-s | normal-ns
fun SN-rel-ext-step :: 'a rel ⇒ 'a rel ⇒ 'a rel ⇒ SN-rel-ext-type ⇒ 'a rel
where
  SN-rel-ext-step P Pw R Rw top-s = P
  SN-rel-ext-step P Pw R Rw top-ns = Pw
  SN-rel-ext-step P Pw R Rw normal-s = R
  SN-rel-ext-step P Pw R Rw normal-ns = Rw

definition SN-rel-ext :: 'a rel ⇒ 'a rel ⇒ 'a rel ⇒ 'a rel ⇒ ('a ⇒ bool) ⇒ bool
where
  SN-rel-ext P Pw R Rw M ≡ (¬ (∃ f t.
    (∀ i. (f i, f (Suc i)) ∈ SN-rel-ext-step P Pw R Rw (t i))
    ∧ (∀ i. M (f i))
    ∧ (INFM i. t i ∈ {top-s, top-ns}))
    ∧ (INFM i. t i ∈ {top-s, normal-s})))

lemma SN-rel-ext-step-mono: assumes P ⊆ P' Pw ⊆ Pw' R ⊆ R' Rw ⊆ Rw'
shows SN-rel-ext-step P Pw R Rw t ⊆ SN-rel-ext-step P' Pw' R' Rw' t
using assms
by (cases t, auto)

lemma SN-rel-ext-mono: assumes subset: P ⊆ P' Pw ⊆ Pw' R ⊆ R' Rw ⊆ 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 rel and M :: 'a ⇒ bool
  defines M': M' ≡ {(s,t). M t}
  defines A: A ≡ (P ∪ Pw ∪ R ∪ Rw) ∩ M'
  assumes SN-rel-ext P Pw R Rw M
  shows SN-rel-ext (A' ∩ O (P ∩ M')) O A' ∩ O (A' ∩ O ((P ∪ Pw) ∩ M' ∩ O A' ∩ O (A' ∩ O (P ∪ R) ∩ M')) O A' ∩ O (A' ∩ O M (is SN-rel-ext P Pw R Rw M))
proof (rule ccontr)
let ?relt = SN-rel-ext-step P Pw R Rw
let ?rel = SN-rel-ext-step P Pw R Rw
assumes ¬ ?thesis
from this[unfolded SN-rel-ext-def]
obtain f ty
  where steps: (∀ i. (f i, f (Suc i)) ∈ ?relt (ty i))
      and min: (∀ i. M (f i))
and inf1: INFM i. ty i ∈ {top-s, top-ns}
and inf2: INFM i. ty i ∈ {top-s, normal-s}
by auto
let ?Un = λ tt. ∪ (?rel' tt)
let ?UnM = λ tt. (∪ (?rel' tt)) ∩ M'
let ?A = ?UnM {top-s, top-ns, normal-s, normal-ns}
let ?P' = ?UnM {top-s}
let ?Pw' = ?UnM {top-s, top-ns}
let $R' = \uplus M \{\text{top-s,normal-s}\}$
let $Rw' = \uplus M \{\text{top-s,top-ns,normal-s,normal-ns}\}$

have $A: A = \uplus A$ unfolding $A$ by auto

have $P: (P \cap M') = \uplus P'$ by auto
have $Pw: (P \cup Pw) \cap M' = \uplus Pw'$ by auto
have $R: (P \cup R) \cap M' = \uplus R'$ by auto
have $Rw: A = \uplus Rw'$ unfolding $A$ ..

{ fix $s$ $t$ $tt$
  assume $m: M s$ and $st: (s,t) \in \uplus M tt$
  hence $\exists \text{typ } \in tt. (s,t) \in \rel \text{typ } \wedge M s \wedge M t$ unfolding $M'$ by auto
}

note one-step = this

let $\lambda s t g n ty. s = g 0 \wedge t = g n \wedge (\forall i < n. (g i, g (Suc i)) \in \rel (ty i)) \wedge (\forall i \leq n. M (g i))$

{ fix $s$ $t$
  assume $m: M s$ and $st: (s,t) \in A^*$
  from $st[\text{unfolded rtrancl-fun-conv}]$
  obtain $gn$ where $g0: g 0 = s$ and $gn: g n = t$ and steps: $\wedge i. i < n \Rightarrow (g i, g (Suc i)) \in \rel (ty i)) \wedge (\forall i \leq n. M (g i))$
}

{ 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 $i \leq n$ have $j < n$ by auto
    from steps[OF this] show $?thesis$ unfolding $Suc M'$ by auto
  qed
}

{ fix $i$
  assume $i: i < n$ hence $i': i \leq n$ by auto
  from $i'$ one-step[OF min steps[OF $i$]]
  have $\exists \text{ty. } (g i, g (Suc i)) \in \rel \text{ty}$ by blast
}

hence $\forall i. (\exists \text{ty. } i < n \Rightarrow (g i, g (Suc i)) \in \rel \text{ty})$ by auto
from choice[OF this]
obtain $tt$ where steps: $\wedge i. i < n \Rightarrow (g i, g (Suc i)) \in \rel (tt i)$ 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 $A\text{-steps} = this$
  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) ∈ A^* O UnM tt O A^*
then obtain u v where su: (s,u) ∈ A^* and uv: (u,v) ∈ 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 ∈ 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 = 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 = ?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)

88
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-steps[OF min this]
  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 inf[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 θ < 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 \( \neg SN\text{-rel-ext} \) \( P \) \( Pw \) \( R \) \( Rw \) \( M \)

unfolding \( SN\text{-rel-ext-def} \) simp-thms

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

fix \( k \)

obtain \( i \) \( j \) where \( ik\): \( ?\text{ind } k = (i,j) \) by force

from bounds[OF this] have \( j < n \) \( i \) by auto

show \( M \ (ik) \) unfolding \( ik \) using partial[of \( i \)] \( j \) by auto

next

fix \( k \)

obtain \( i \) \( j \) where \( ik\): \( ?\text{ind } k = (i,j) \) by force

from bounds[OF this] have \( j < n \) \( i \) by auto

from partial[of \( i \)] \( j \) have step: \( (g \ i \ j, \ g \ i \ (Suc \ j)) \in ?rel \ (ty' \ i \ j) \) by auto

obtain \( i' \) \( j' \) where \( isk: \ ?\text{ind } (Suc \ k) = (i'\ j') \) 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 \( \ldots = g \ (Suc \ i) \ \emptyset \) using partial[of Suc \( i \)] by simp

finally show \( g \ i \ (n \ i) = g \ (Suc \ i) \ \emptyset \).

qed

show \( (?g \ k, \ ?\text{rel} \ (Suc \ k)) \in ?rel \ (\ ?ty \ k) \)

unfolding \( ik \) isk split \( i'\ j' \)

by (rule step)

next

show \( \text{INFM} \ i. \ ?\text{ty} \ i \in \{\text{top-s, top-ns}\} \)

unfolding \( \text{INFM-nat-le} \)

proof (intro allI)

fix \( k \)

obtain \( i \) \( j \) where \( ik: \ ?\text{ind } k = (i,j) \) by force

from inf1[unfolded \( \text{INFM-nat} \)] obtain \( i' \) where \( i'\): \( i' > i \) and \( ty: ty' \ i' \in \{\text{top-s, top-ns}\} \) by auto

from partial[of \( i' \)] \( ty \) obtain \( j' \) where \( j': j' < n \) \( i' \) and \( ty': ty' \ i' \ j' \in \{\text{top-s, top-ns}\} \) by auto

from inf-concat-surj[of - \( n \), \( OF \ j' \)] obtain \( k' \) where \( ik'\): \( ?\text{ind } k' = (i'\ j') \)

from inf-mono[OF \( ik' \) \( i' \)] have \( k: k \leq k' \) by simp

show \( \exists k' \geq k. \ ?\text{ty} \ k' \in \{\text{top-s, top-ns}\} \)

by (intro exI conjI, rule \( k \), unfold ik' split, rule \( ty' \))

qed

next

show \( \text{INFM} \ i. \ ?\text{ty} \ i \in \{\text{top-s, normal-s}\} \)

unfolding \( \text{INFM-nat-le} \)

proof (intro allI)

fix \( k \)

obtain \( i \) \( j \) where \( ik: \ ?\text{ind } k = (i,j) \) by force

from inf2[unfolded \( \text{INFM-nat} \)] obtain \( i' \) where \( i': i' > i \) and \( ty: ty' \ i' \in \{\text{top-s, normal-s}\} \) by auto

from partial[of \( i' \)] \( ty \) obtain \( j' \) where \( j': j' < n \) \( i' \) and \( ty': ty' \ i' \ j' \in \{\text{top-s, normal-s}\} \) by auto
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' †] have k: k ≤ k' by simp
  show ∃ k' ≥ k. ?ty k' ∈ {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 ⇒ bool
  defines Ms: Ms ≡ {(s, t). M' t}
defines A: A ≡ (P' ⊔ Pw' ⊔ R' ⊔ Rw') ∩ Ms
assumes SN: SN-rel-ext P' Pw' R' Rw' M'
and P: ∃ s t. M s → M t ⊢ (s, t) ∈ P ⇒ (f s, f t) ∈ (A ° O (P' ∩ Ms) O A °)
and Pw: ∃ s t. M s → M t ⊢ (s, t) ∈ Pw ⇒ (f s, f t) ∈ (A ° O ((P' ∪ Pw') ∩ Ms) O A °)
and R: ∃ s t. I s → M s → M t ⊢ (s, t) ∈ R ⇒ (f s, f t) ∈ (A ° O ((P' ∪ R') ∩ Ms) O A °) ∩ I t
and Rw: ∃ s t. I s → M s → M t ⊢ (s, t) ∈ Rw ⇒ (f s, f t) ∈ A ° ∩ I t
tools SN-rel-ext P Pw R Rw M
proof
  note SN = SN-rel-ext-trans[OF SN]
let ?P = (A ° O (P' ∩ Ms) O A °)
let ?Pw = (A ° O ((P' ∪ Pw') ∩ Ms) O A °)
let ?R = (A ° O ((P' ∪ R') ∩ Ms) O A °)
let ?Rw = A °
let ?rel = SN-rel-ext-step P Pw R Rw
shows ?thesis
proof (rule ccontr)
  assume ¬ ?thesis
  from this[unfolded SN-rel-ext-def]
  obtain g ty
    where steps: ∃ i. (g i, g (Suc i)) ∈ ?rel (ty i)
    and min: ∃ i. M (g i)
  and inf1: INFM i. ty i ∈ {top-s, top-ns}
  and inf2: INFM i. ty i ∈ {top-s, normal-s}
    by auto
  from inf1[unfolded INFM-nat] obtain k where k: ty k ∈ {top-s, top-ns} by auto
    let ?k = Suc k
    let ?i = shift id ?k
    let ?f = λ i. f (shift g ?k i)
    let ?ty = shift ty ?k
{
fix i
assume ty: ty i ∈ \{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))) ∈ ?relt (ty i) ∧ 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))) ∈ ?relt (ty i) ∧ 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 ¬ SN-rel-ext ?P ?Pw ?R ?Rw M'
  unfolding SN-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)) ∈ ?relt (?ty i)
    using stepsI[OF I[of i]] by auto
next
  show INFM i. ?ty i ∈ \{top-s, top-ns\}
    unfolding Infm-shift[of λi. i ∈ \{top-s, top-ns\} ty ?k]
    by (rule inf1)
next
  show INFM i. ?ty i ∈ \{top-s, normal-s\}
    unfolding Infm-shift[of λi. i ∈ \{top-s, normal-s\} ty ?k]
    by (rule inf2)
92
next

fix i

have A: A ⊆ Ms unfolding A by auto
from rtrancl-mono[of this] have A*: A* ⊆ Ms* by auto
have PM: ?P ⊆ Ms* O Ms O Ms* using As by auto
have PwM: ?Pw ⊆ Ms* O Ms O Ms* using As by auto
have RM: ?R ⊆ Ms* O Ms O Ms* using As by auto
have RwM: ?Rw ⊆ Ms* using As by auto

from PM PwM RM have ?P ∪ ?Pw ∪ ?R ⊆ Ms* O Ms O Ms* (is ?PPR ⊆ -) by auto
also have ... ⊆ Ms*+ by regexp
also have ... = Ms

proof
  have Ms*+ ⊆ Ms* O Ms by regexp
  also have ... ⊆ Ms unfolding Ms by auto
  finally show Ms*+ ⊆ Ms .
qed regexp

finally have PPR: ?PPR ⊆ Ms .
show M' (?f i)
proof (induct i)
  case 0
  from stepsP[of k] k have (f (g k), f (g (Suc k))) ∈ ?PPR by (cases ty k, auto)
  with PPR show ?case unfolding Ms by simp blast
next
  case (Suc i)
  show ?thesis
  proof (cases ?ty i = normal-ns)
    case False
    hence ?ty i ∈ {top-s,top-ns,normal-s}
      by (cases ?ty i, auto)
    with stepsI[of I[of i]] have (?if i, ?f (Suc i)) ∈ ?PPR
      by auto
    from subsetD[of PPR this] have (?if i, ?f (Suc i)) ∈ Ms .
    thus ?thesis unfolding Ms by auto
next
  case True
  with stepsI[of I[of i]] have (?if i, ?f (Suc i)) ∈ ?Rw by auto
  with RwM have mem: (?if i, ?f (Suc i)) ∈ Ms* by auto
  thus ?thesis
  proof (cases)
    case Suc show ?thesis by simp
next
  case step
  thus ?thesis unfolding Ms by simp
qed
qed

qed

93
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 ⇒ bool

defines Ms: Ms ≡ {(s,t). M' t}
defines A: A ≡ P' ∩ Ms ∪ Pw' ∩ Ms ∪ R' ∪ Rw'

assumes SN: SN-rel-ext P' Pw' R' Rw' M'

and M: \( \forall s. M s \Rightarrow M'(f t) \)

and M': \( \forall s. M' s \Rightarrow (s,t) \in R' \cup Rw' \Rightarrow M' t \)

and P: \( \forall s. M s \Rightarrow M' (f s) \Rightarrow M'(f t) \Rightarrow (s,t) \in P \Rightarrow (f s, f t) \in (A^* O (P' ∩ Ms) O A^*) \cap I \)

and Pw: \( \forall s. M s \Rightarrow M' (f s) \Rightarrow M'(f t) \Rightarrow (s,t) \in Pw \Rightarrow (f s, f t) \in (A^* O (P' ∩ Ms ∪ Pw') ∩ Ms) O A^* ) \cap I \)

and R: \( \forall s. I s \Rightarrow M s \Rightarrow M' (f s) \Rightarrow M'(f t) \Rightarrow (s,t) \in R \Rightarrow (f s, f t) \in (A^* O (P' ∩ Ms ∪ R') O A^* ) \cap I \)

and Rw: \( \forall s. I s \Rightarrow M s \Rightarrow M t \Rightarrow M' (f s) \Rightarrow M'(f t) \Rightarrow (s,t) \in Rw \Rightarrow (f s, f t) \in A^* \cap I \)

shows SN-rel-ext P Pw R Rw M

proof —

let ?Ms = {(s,t). M' t}
let ?A = (P' ∩ Pw' ∪ R' ∪ Rw') ∩ ?Ms

{ fix s t
  assume s: M' s and (s,t) ∈ A
  with M'(OF s, of t) have (s,t) ∈ ?A ∧ M' t unfolding Ms A by auto
  } note Aone = this

{ fix s t
  assume s: M' s and steps: (s,t) ∈ A^*
  from steps have (s,t) ∈ ?A^* ∧ 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) ∈ ?A^* O ?A by blast
    have (s,u) ∈ ?A^*
      by (rule subsetD[OF - steps], regexp)
    with one show ?case by simp
  qed
  } note Amany = this
let ?P = (A^* O (P' ∩ Ms) O A^*)
let ?Pw = (A^* O (P' ∩ Ms ∪ Pw' ∩ Ms) O A^*)
let \( ?R = (A \ast O (P' \cap Ms \cup R') \ O A \ast) \)
let \( ?Rw = A \ast \)
let \( ?P' = (?A \ast O (P' \cap ?Ms) \ O ?A \ast) \)
let \( ?Pw' = (?A \ast O ((P' \cup Pw') \cap ?Ms) \ O ?A \ast) \)
let \( ?R' = (?A \ast O ((P' \cup R') \cap ?Ms) \ O ?A \ast) \)
let \( ?Rw' = ?A \ast \)

show \(?\text{thesis}\)

proof (rule SN-rel-ext-map[OF SN])

fix \( s t \)
assume \( s: M s \text{ and } t: M t \text{ and } \text{step: } (s,t) \in P \)
from \( P[OF \ s \ t \ M[OF \ s] \ M[OF \ t] \text{ step}] \)
have \( (f s, f t) \in ?P \text{ and } I: I t \text{ by } \text{auto} \)
then obtain \( u v \text{ where } su: (f s, u) \in A \ast \text{ and } uv: (u,v) \in P' \cap Ms \)
and \( vt: (v,f t) \in A \ast \text{ by } \text{auto} \)
from \( \text{Amany}\[OF \ M[OF \ s] \ su]\text{ have } su: (f s, u) \in ?A \ast \text{ and } w: M' \text{ a } \text{by } \text{auto} \)
from \( uv \text{ have } v: M' \text{ v unfolding } Ms \text{ by } \text{auto} \)
from \( \text{Amany}\[OF \ v \ vt]\text{ have } vt: (v, f t) \in ?A \ast \text{ by } \text{auto} \)
from \( su \text{ uw } vt I \)
show \( (f s, f t) \in ?P' \wedge I t \text{ unfolding } Ms \text{ by } \text{auto} \)

next
fix \( s t \)
assume \( s: M s \text{ and } t: M t \text{ and } \text{step: } (s,t) \in Pw \)
from \( Pw[OF s t M[OF s] M[OF t] \text{ step}] \)
have \( (f s, f t) \in ?Pw \text{ and } I: I t \text{ by } \text{auto} \)
then obtain \( u v \text{ where } su: (f s, u) \in A \ast \text{ and } uv: (u,v) \in P' \cap Ms \cup Pw' \)
\cap Ms
and \( vt: (v,f t) \in A \ast \text{ by } \text{auto} \)
from \( \text{Amany}\[OF \ M[OF s] \ su]\text{ have } su: (f s, u) \in ?A \ast \text{ and } w: M' \text{ a } \text{by } \text{auto} \)
from \( uv \text{ have } w: (u,v) \in (P' \cup Pw') \cap ?Ms \text{ and } v: M' \text{ v unfolding } Ms \text{ by } \text{auto} \)
from \( su \text{ uw } vt I \)
show \( (f s, f t) \in ?Pw' \wedge I t \text{ by } \text{auto} \)

next
fix \( s t \)
assume \( I: I s \text{ and } s: M s \text{ and } t: M t \text{ and } \text{step: } (s,t) \in R \)
from \( R[OF s t M[OF s] M[OF t] \text{ step}] \)
have \( (f s, f t) \in ?R \text{ and } I: I t \text{ by } \text{auto} \)
then obtain \( u v \text{ where } su: (f s, u) \in A \ast \text{ and } uv: (u,v) \in P' \cap Ms \cup R' \)
and \( vt: (v,f t) \in A \ast \text{ by } \text{auto} \)
from \( \text{Amany}\[OF \ M[OF s] \ su]\text{ have } su: (f s, u) \in ?A \ast \text{ and } w: M' \text{ a } \text{by } \text{auto} \)
from \( uv \text{ have } w: (u,v) \in (P' \cup R') \cap ?Ms \text{ and } v: M' \text{ v unfolding } Ms \text{ by } \text{auto} \)
from \( \text{Amany}\[OF \ v \ vt]\text{ have } vt: (v, f t) \in ?A \ast \text{ by } \text{auto} \)
from \( su \text{ uw } vt I \)
show \( (f s, f t) \in ?R' \wedge I t \text{ by } \text{auto} \)

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) ∈ ?Rw' ∧ I t by auto
qed
qed

lemma SN-relto-imp-SN-rel: SN (relto R S) ⇒ 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 ∪ 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 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]
    have r-index: j. 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: j. 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
      }
    }
  }
qed
from this[of rsi – ri – 1] risi have
\((f \ (\text{Suc } ri), f \ rsi) \in S^* \text{ by simp} \)
with ri rsi have ssteps: \((f \ (\text{Suc } ?ri), f \ ?rsi) \in S^* \text{ by simp} \)
with r-index[of i] have \((f \ ?ri, f \ ?rsi) \in R \ O S^* \text{ unfolding r by auto} \)
hence \((g \ i, g \ (\text{Suc } i)) \in S^* \ O R \ O S^* \) using rtrancl-refl unfolding g
by auto
hence \(- SN\ (S^* \ O R \ O S^*) \text{ unfolding SN-defs by blast} \)
with SN show False by simp
qed
qed

lemma rtrancl-list-conv:
\(((s,t) \in R^*) \Rightarrow 
  (\exists \text{list. last (s \\# list) = t} \wedge (\forall i. i < \text{length list} \rightarrow ((s \\# list) ! i, (s \\# list) ! \text{Suc } i) \in R)) \) (is \(\text{?l = ?r}) \)
proof
  assume ?r
  then obtain list where last (s \\# list) = t \wedge (\forall i. i < \text{length list} \rightarrow ((s \\# list) ! i, (s \\# list) ! \text{Suc } i) \in R) \).
  thus ?l.
  proof (induct list arbitrary: s, simp)
    case (Cons u ll)
    hence last (u \\# ll) = t \wedge (\forall i. i < \text{length ll} \rightarrow ((u \\# ll) ! i, (u \\# ll) ! \text{Suc } i) \in R) \) by auto
    from Cons \(\text{1[OF this]} \) have rec: \((u,t) \in R^* \).
    from Cons have \((s, u) \in R \text{ by auto} \).
    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 ?list = map \((\lambda i. S \ (\text{Suc } i)) [0 <. n] \)
  show ?r
  proof (rule exI[of - ?list], intro conj1,
    cases n, simp add: s[symmetric], simp add: t[symmetric])
  show \(\forall i < \text{length } ?\text{list}. ((s \\# ?\text{list}) ! i, (s \\# ?\text{list}) ! \text{Suc } i) \in R \)
  proof (intro allI impI)
    fix i
    assume i: i < length ?list
    thus \((s \\# ?\text{list}) ! i, (s \\# ?\text{list}) ! \text{Suc } i) \in R \)
  proof (cases i, simp add: s[symmetric] steps)
    case (Suc j)
    with i steps show ?thesis by simp
  qed
qed

97
\[\text{fun choice :: (nat ⇒ 'a list) ⇒ nat ⇒ (nat × nat) where}\]
\[\text{choice} f 0 = (0,0)\]
\[| \text{choice} f \text{ (Suc } n) = \text{ (let } (i, j) = \text{ choice } f n \text{ in}
\text{if } \text{Suc } j < \text{ length } (f i)
\text{ then } (i, \text{Suc } j)
\text{ else } (\text{Suc } i, 0))\]

\textbf{lemma} \text{SN-rel-imp-SN-relto} : \text{SN-rel R S} \implies \text{SN (relto R S)}
\textbf{proof –}
\text{assume} \text{SN: SN-rel R S}
\text{show} \text{SN (relto R S)}
\textbf{proof}
\text{fix } f
\text{assume} \forall i. \text{ (f i, f (Suc } i) \in \text{ relto R S)}
\text{hence} \text{ steps: } \forall i. \text{ (f i, f (Suc } i) \in S^* \times R \times S^* \text{ by auto}}
\text{let} \text{ ?prop = } \lambda i. \text{ ai } \text{ bi. (f i, b i) } \in S^* \times (\text{ (bi, ai)} \in R \times \text{ (ai, f (Suc } i)})) \in S^*
\{ \text{fix } i \}
\text{from} \text{ steps obtain} \text{ bi ai where} \text{ ?prop i ai bi by blast}
\text{hence} \exists ai bi. \text{ ?prop i ai bi by blast}
\}
\text{hence} \forall i. \exists ai bi. \text{ ?prop i ai bi by blast}
\text{from} \text{ choice[OF this]} \text{ obtain} \text{ b where} \forall i. \exists ai. \text{ ?prop i ai (b i) by blast}
\text{from} \text{ choice[OF this]} \text{ obtain} \text{ a where} \text{ steps: } \forall i. \text{ ?prop i (a i) (b i) by blast}
\text{let} \text{ ?prop = } \lambda i. \text{ ai li. (b i, a i) } \in R \times \text{ (\forall j < length } li. ((a i \# li) \land j, (a i \# li)}}
\text{Suc } j) \in S) \land \text{last} \text{ (a i \# li) } = b \text{ (Suc } i)
\{ \text{fix } i \}
\text{from} \text{ steps[of i]} \text{ steps[of Suc i] have} \text{ (a i, f (Suc } i) \in S^* \times \text{ (S (Suc } i)}),
\text{b (Suc } i) \in S^* \text{ by auto}}
\text{from} \text{ rtrancl-trans[OF this]} \text{ steps[of i] have} \text{ R: (b i, a i) } \in R \text{ and } S: (a i, b
\text{(Suc i)) } \in S^* \text{ by blast+}}
\text{from} \text{ S[unfolded rtrancl-list-cone] obtain} \text{ li where last} \text{ (a i \# li) } = b \text{ (Suc } i)
\land \forall j < \text{ length } li. ((a i \# li) \land j, (a i \# li) \land \text{Suc } j) \in S) \ldots
\text{with} \text{ R have} \text{ ?prop i li by blast}
\text{hence} \exists li. \text{ ?prop i li ..}
\}
\text{hence} \forall i. \exists li. \text{ ?prop i li ..}
\text{from} \text{ choice[OF this]} \text{ obtain} \text{ l where} \text{ steps: } \forall i. \text{ ?prop i (l i) by auto}
\text{let} \text{ ?p = } \lambda i. \text{ ?prop i (l i)}
\text{from} \text{ steps have steps: } \forall i. \text{ ?p i by blast}
\text{let} \text{ ?l = } \lambda i. \text{ a i \# li i}
\text{let} \text{ ?g = } \lambda i. \text{ choice } (\lambda j. \text{ ?l } j) \text{ i}
\text{obtain } g \text{ where} g: \forall i. \text{ g i = } (\text{let (i j) = ?g i in } \text{ ?l i i } j) \text{ by auto}
\text{have} \text{ len: } \forall i. j. \text{ ?g n = (i, j) } \implies j < \text{ 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

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 jj) by auto
with n have gn: g n = ?l i ! j and gsn: g (Suc n) = ?l i ! (Suc jj) unfolding g by auto
thus ?thesis using steps[of i] True by auto
next
case False
with n have ?g (Suc n) = (Suc ii, 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

have infR: ∀ n. ∃ j ≥ n. (g j, g (Suc j)) ∈ R
proof
fix n
obtain i j where n: ?g n = (i, j) by (cases ?g n, auto)
from len[OF n] have j: j ≤ length (?l i) − 1 by simp
let \( k = \text{length } (\_ i) - 1 - j \)

obtain \( k \) where \( k = j + \_ k \) by auto

from \( j k \) have \( k2: k = \text{length } (\_ i) - 1 \) and \( k3: j + \_ k < \text{length } (\_ i) \) by auto

\(
\begin{align*}
\text{fix } & n \ i \ j \ k \ l \\
\text{assume } & n: \text{choice } l \ n = (i, j) \text{ and } j + k < \text{length } (l i) \\
\text{hence } & \text{choice } l (n + k) = (i, j + k) \\
\text{by } & \text{(induct } k \text{ arbitrary: } j, \text{ simp, auto)}
\end{align*}
\)

from this\([OF n, of \_ k] \) 

have \( gnk: \exists g (n + \_ k) = (i, k) \) by (simp only: \( k \))

hence \( g (n + \_ k) = ?l i ! k \) unfolding \( g \) by auto

hence \( gnk2: g (n + \_ k) = \text{last } (\_ l i) \) using \( \text{last-conv-nth[of } \_ l i \_ k2 \) by auto

from \( k2 gnk \) have \( \exists g (\text{Suc } (n + \_ k)) = (\text{Suc } i, 0) \) by auto

hence \( gnk2: g (\text{Suc } (n + \_ k)) = a (\text{Suc } i) \) unfolding \( g \) by auto

from \( \text{steps[of } i \} \text{ steps[of } \_ Suc i \) have \( \text{main: } (g (n + \_ k), g (\text{Suc } (n + \_ k))) \in R \)

by (simp only: \( gnk2 gnk2 \))

show \( \exists j \geq n. (g j, g (\text{Suc } j)) \in R \)

by (rule \( \text{exI[of - } n + \_ k, \text{ auto simp: main[simplified]} \))

qed

from \( \text{SN[ simplified } \text{ SN-rel-on-conv } \text{ SN-rel-defs] gsteps infR show False \)

unfolding \( \text{INFM-nat-le by fast} \)

qed

hide-const choice

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

by (blast intro: \( \text{SN-relto-imp-SN-rel } \text{SN-rel-imp-SN-relto} \))

lemma \( \text{SN-rel-empty1: } \text{SN-rel } \{\} S \)

unfolding \( \text{SN-rel-defs by auto} \)

lemma \( \text{SN-rel-empty2: } \text{SN-rel } R \{\} = \text{SN } R \)

unfolding \( \text{SN-rel-defs SN-defs by auto} \)

lemma \( \text{SN-relto-mono:} \)

assumes \( R: R \subseteq R ‘ \) and \( S: S \subseteq S ‘ \)

and \( \text{SN: } \text{SN } (\text{relto } R ‘ S ‘) \)

shows \( \text{SN } (\text{relto } R S) \)

using \( \text{SN SN-subset[of } R ‘ \text{ relto-mono[of } R S \]} \) by blast

lemma \( \text{SN-relto-imp-SN:} \)

assumes \( \text{SN } (\text{relto } R S) \) shows \( \text{SN } R \)

proof

fix \( f \)

assume \( \forall i. (f i, f (\text{Suc } i)) \in R \)

hence \( \forall i. (f i, f (\text{Suc } i)) \in \text{relto } R S \) by blast
thus False using assms unfolding SN-defs by blast

qed

lemma SN-relo-Id:
SN (relto R (S ∪ 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 ⊆ (r ∪ s) and SN r and SN s
shows SN R

proof
fix f :: nat ⇒ 'a
assuming f 0 ∈ UNIV
and chain: chain R f

have star: ∃i j. i < j ⇒ (f i, f j) ∈ r ∪ s
using assms and chain-imp-trancl [OF chain] by auto

let M = {i. (∀j > i. (f i, f j) /∈ r)

show False
proof (cases finite M)
let n = Max M

assume finite M
with Max-le have ∃i∈M. i ≤ n by simp
then have ∃k≥Suc n. f k, f (Suc k) ∈ r by auto
with steps-imp-chainp [of Suc n λ x y. (x, y) ∈ r] and assms
show False by auto
next
assume infinite M
then have INFM j. j ∈ M by (simp add: Inf-many-def)
then interpret infinitely-many λi. i ∈ M by (unfold-locales) assumption

define g where simp: g = index

have ∃i. (f (g i), f (g (Suc i))) ∈ s

proof
fix i
have less: g i < g (Suc i) using index-ordered-less [of i Suc i] by simp
have g i ∈ M using index-p by simp
then have (f (g i), f (g (Suc i))) /∈ r using less by simp
moreover have (f (g i), f (g (Suc i))) ∈ r ∪ s using * [OF less] by simp
ultimately show (f (g i), f (g (Suc i))) ∈ s by blast
qed
with ⟨SN s⟩ show False by (auto simp: SN-defs)
qed

qed

lemma SN-Un-conv:
assumes trans (r ∪ s)
shows (r ∪ s) ↔ SN r ∧ SN s
(is SN ?r ↔ ?rhs)

proof
assume $SN \ (r \cup s)$ thus $SN \ r \land SN \ s$

using $SN$-subset[of $?r$] by blast

next

assume $SN \ r \land SN \ s$

with trans-subset-$SN$[OF assms subset-refl] show $SN \ {?r}$

by blast

qed

lemma $SN$-relto-Un:

$SN \ (relto \ (R \cup S) \ Q) \iff SN \ (relto \ R \ (S \cup Q)) \land SN \ (relto \ S \ Q)$

(is $SN \ ?a \iff SN \ ?b \land SN \ ?c$)

proof

have eq: $?a^+ = ?b^+ \cup ?c^+$ by regexp

from $SN$-Un-conv[of $?b^+ \cup ?c^+$, unfolded eq[symmetric]]

show $?thesis$ unfolding $SN$-trancl-$SN$-conv by simp

qed

lemma $SN$-relto-split:

assumes $SN \ (relto \ r \ (s \cup q2) \cup relto \ q1 \ (s \cup q2))$ (is $SN \ ?a$)

and $SN \ (relto \ s \ q2)$ (is $SN \ ?b$)

shows $SN \ (relto \ r \ (q1 \cup q2) \cup relto \ s \ (q1 \cup q2))$ (is $SN \ ?c$)

proof

have $?c^+ \subseteq ?a^+ \cup ?b^+$ by regexp

from trans-subset-$SN$[OF - this, unfolded $SN$-trancl-$SN$-conv, OF - assms]

show $?thesis$ by simp

qed

lemma relto-trancl-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 $\ldots \subseteq c^+$

by (rule trancl-mono-set, insert assms, auto)

finally show $?thesis$.

qed

An explicit version of relto which mentions all intermediate terms

inductive relto-fun :: $\ 'a \ rel \Rightarrow 'a \ rel \Rightarrow \ nat \Rightarrow \ nat \Rightarrow \ 'a \times 'a \Rightarrow \ bool$ where

relto-fun: as $0 = a \Rightarrow as \ m = b \Rightarrow$

$(\land i. i < m \Rightarrow$

$(sel \ i \rightarrow (as \ i, \ as \ (Suc \ i)) \in A) \land \ (\neg \ sel \ i \rightarrow (as \ i, \ as \ (Suc \ i)) \in B))$

$\Rightarrow n = \ card \ \{i . \ i < m \land sel \ i\}$

$\Rightarrow (n = 0 \leftarrow\rightarrow m = 0) \Rightarrow relto-fun \ A \ B \ n \ as \ sel \ m \ (a,b)$

lemma relto-funD:

assumes relto-fun $A \ B \ n \ as \ sel \ m \ (a,b)$

shows as $0 = a \Rightarrow as \ m = b$

$(\land i. i < m \Rightarrow sel \ i \equiv (as \ i, \ as \ (Suc \ i)) \in A)$

$(\land i. i < m \Rightarrow \neg \ sel \ i \equiv (as \ i, \ as \ (Suc \ i)) \in B)$

$n = \ card \ \{i . \ i < m \land sel \ i\}$

$n = 0 \leftarrow\rightarrow m = 0$

102
using assms[unfolded relto-fun.simps] by blast+

lemma relto-fun-refl: \( \exists \) sel. relto-fun \( A \ B \ 0 \) as sel \( 0 \) (\( a,a \))
by (rule exI[of - \( \lambda \cdot \cdot \cdot a \)], rule exI, rule relto-fun, auto)

lemma relto-fun-trans: assumes \( (a,b) \in \text{relto} \ A \ B \)
shows \( \exists \) as sel m. relto-fun \( A \ B \) (\( \text{Suc} \ 0 \)) as sel \( m \) (\( a,b \))
proof –
  from assms obtain \( a' \ b' \) where \( aa: (a,a') \in B^-\ast \) and \( ab: (a',b') \in A \)
  and \( bb: (b',b) \in B^-\ast \) by auto
  from \( aa[\text{unfolded rtrancl-fun-conv}] \) obtain \( f1 \) \( n1 \) where
    \( f1: f1 \ 0 = a f1 \ n1 = a' \land i. i < n1 \implies (f1 \ i, f1 (\text{Suc} \ i)) \in B \) by auto
  from \( bb[\text{unfolded rtrancl-fun-conv}] \) obtain \( f2 \) \( n2 \) where
    \( f2: f2 \ 0 = b' f2 \ n2 = b \land i. i < n2 \implies (f2 \ i, f2 (\text{Suc} \ i)) \in B \) by auto
  let \( \?gen = \lambda \ aa \ aa \ bb \ i. \text{if} \ i < n1 \text{ then} aa \ i \text{ else} if \ i = n1 \text{ then} ab \ i \text{ else} bb \ i \text{ (i - Suc} \ n1) \)
  let \( \?f = \?gen \ f1 \ a' \ f2 \)
  let \( \?sel = \?gen (\lambda \cdot \cdot \cdot \text{False}) \text{ True (\lambda \cdot \cdot \cdot \text{False})} \)
  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 \( (\?sel \ i \longrightarrow (\?f \ i, \?f (\text{Suc} \ i)) \in A) \land (\neg \?sel \ i \longrightarrow (\?f \ i, \?f (\text{Suc} \ i)) \in B) \)
      proof (cases \( i < n1 \))
        case True
        with \( f1(\text{OF} \ this) \) \( f1 \ (2) \) show \( \?thesis \) by (cases 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 \( \text{True unfolding} \ j\text{-def} \) by auto
          thus \( \?thesis \) using \( f2 \) by auto
          qed
        qed
      qed (insert \( f1 \ f2 \), auto)
      qed

lemma relto-fun-conv: assumes \( ab: \text{relto-fun} \ A \ B \ a1 \ b1 \) \( \text{sel1} \ m1 \) (\( a,b \))
and \( bc: \text{relto-fun} \ A \ B \ a2 \ b2 \) \( \text{sel2} \ m2 \) (\( b,c \))
shows \( \exists \) as sel. relto-fun \( A \ B \) (\( n1 + n2 \)) as sel \( (m1 + m2) \) (\( a,c \))
proof —

from relto-funD[OF ab]
have 1: as1 0 = a as1 m1 = b
  \( \Lambda i. i < m1 \implies (sel1 i \mapsto (as1 i, as1 (Suc i))) \in A) \land (\neg sel1 i \mapsto (as1 i, as1 (Suc i))) \in B) \)
  \( n1 = 0 \iff m1 = 0 \) and card1: \( \text{card} \{ i. i < m1 \land sel1 i \} \) by blast+

from relto-funD[OF bc]
have 2: as2 0 = b as2 m2 = c
  \( \Lambda i. i < m2 \implies (sel2 i \mapsto (as2 i, as2 (Suc i))) \in A) \land (\neg sel2 i \mapsto (as2 i, as2 (Suc i))) \in B) \)
  \( n2 = 0 \iff m2 = 0 \) and card2: \( \text{card} \{ i. i < m2 \land sel2 i \} \) by blast+

let ?as = \( \lambda i. \text{if } i < m1 \text{ then } as1 i \text{ else } as2 (i - m1) \)
let ?sel = \( \lambda i. \text{if } i < m1 \text{ then } sel1 i \text{ else } sel2 (i - m1) \)
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 \land ?sel i \} = \{ i. i < m1 \land sel1 i \} \cup (+) m1 \{ i. i < m2 \land sel2 i \} by force
  have card (?A \cup ?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 \land ?sel i \} unfolding card1 card2 id by simp

next
fix i
assume i: \( i < ?m \)

show (?sel i \mapsto (as i, as (Suc i))) \in A) \land (\neg ?sel i \mapsto (as i, as (Suc i))) \in 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 \( (a,b) \in (relto A B) \) \^\^ \( n \)
shows \( \exists \text{ as sel m, relto-fun A B n as sel m (a,b) } \)
using assms
proof (induct n arbitrary: b)
case \((0 \ b)\)
hence \(b = a\) by auto
show \(?case\ unfolding \ b\ using \ relto-fun-refl[\ of \ A\ B\ a]\ by\ blast\)

next
case \((\text{Suc } n \ c)\)
from relpow-Suc-E[\ OF \ Suc(2)]
obtain \(b\ where\ (a, b) \in (\text{relto } A\ B)^{\sim n}\ and\ bc\ : (b, c) \in \text{relto } A\ B\ by\ auto\)
from Suc(1)[\ OF \ ab]\ obtain as \(sel\ m\) where
IH: \(\text{relto-fun } A\ B\ n\ as\ sel\ m\ (a, b)\ by\ auto\)
from \(\text{relto-into-relto-fun}\ [\ OF \ bc]\ obtain\ as\ sel\ m\\ where\ \text{relto-fun } A\ B\ \text{(Suc } 0)\ as\ sel\ m\ (b, c)\ by\ blast\)
from \(\text{relto-fun-trans}\ [\ OF \ IH\ this]\ show \ ?case\ by\ auto\)
qed

lemma \(\text{relto-fun-into-reltos:}\ assumes\ \text{relto-fun } A\ B\ n\ as\ sel\ m\ (a,b)\)
shows \((a,b) \in (\text{relto } A\ B)^{\sim n}\)
proof
note \(* = \text{relto-funD[OF \ assms]}\)
\{
  fix \(m'\)
  let \(?c = \lambda m'.\ \text{card } \{i.\ i < m' \land sel\ i\}\)
  assume \(m' \leq m\)
hence \((?c m' > 0 \rightarrow (as\ 0, as\ m') \in (\text{relto } A\ B)^{\sim} ?c m') \land (?c m' = 0 \rightarrow (as\ 0, as\ m') \in B^*)\)
proof (induct \(m')\)
case \((\text{Suc } m')\)
let \(?x = as\ 0\)
let \(?y = as\ m'\)
let \(?z = as\ (\text{Suc } m')\)
let \(?C = ?c (\text{Suc } m')\)
have \(C: ?C = ?c m' + (\text{if } \text{sel } m' \text{ then } 1 \text{ else } 0)\)
proof
  have \(\text{id: } \{i.\ i < \text{Suc } m' \land sel\ i\} = \{i.\ i < m' \land sel\ i\} \cup (\text{if } \text{sel } m' \text{ then } \{\} \text{ else } \{\})\)
  by (cases \text{sel } m',\ auto,\ case-tac x = m',\ auto)
  show \(?thesis\ unfolding\ \text{id}\ by\ auto\)
qed
from Suc(2) have \(m' \leq m\ and\ \text{lt: } m' < m\ by\ auto\)
from Suc(1)[\ OF \ m'] have IH: \(?c m' > 0 \rightarrow (?x, ?y) \in (\text{relto } A\ B)^{\sim} ?c m'\)
\(?c m' = 0 \rightarrow (?x, ?y) \in B^*\) by auto
from *(3-4)[\ OF \ \text{lt}] have \(yz: \text{sel } m' \Rightarrow (?y, ?z) \in A \dashv \text{sel } m' \Rightarrow (?y, ?z)\)
in \(B\ by\ auto\)
show \(?case\)
proof (cases \(?c m' = 0\))
case \(\text{True note } c = \text{this}\)
from IH(2)[\ OF \ this]\ have \(xy: (?x, ?y) \in B^*\) by auto
show \(?thesis\)
proof (cases \text{sel } m')
case False
  from \(xy yz\)(2)[OF False] have \(xz\): \((?x, ?z) \in B^*\) by auto
from False c have \(C: ?C = 0\) unfolding C by simp
from \(xz\) show \(\text{thesis unfolding } C\) by auto
next
case True
from \(xy yz\)(1)[OF True] have \(xz\): \((?x, ?z) \in \text{relto } A B\) by auto
from True c have \(C: ?C = 1\) unfolding C by simp
from \(xz\) show \(\text{thesis unfolding } C\) by auto
qed
next
case False
  hence c: \(?c m' > 0\) \((?c m' = 0) = False\) by arith+
from \(IH(1)\)[OF c(1)] have \(xy: (?x, ?y) \in (\text{relto } A B)^* \cdot \cdot (?c m').\)
  show \(\text{thesis}\)
proof (cases sel m')
case False
  from \(c\) obtain \(k\) where \(ck: ?c m' = \text{Suc } k\) by (cases \(?c m'\), auto)
  from relpow_Suc_E[OF \(xy\)[unfolded this]] obtain \(u\) where \(xu: (?x, u) \in (\text{relto } A B)^* \cdot k\) and \(uy: (u, ?y) \in \text{relto } A B\) by auto
  from uy \(yz\)(2)[OF False] have \(uz: (u, ?z) \in \text{relto } A B\) by force
  with xu have \(xz\): \((?x, ?z) \in (\text{relto } A B)^* \cdot (?c m')\) unfolding \(ck\) by auto
from \(xz\) show \(\text{thesis unfolding } C\) by auto
next
case True
from \(xy yz\)(1)[OF True] have \(xz\): \((?x, ?z) \in (\text{relto } A B)^* \cdot \cdot (\text{Suc } (?c m'))\) by auto
from \(c\) True have \(C: ?C = \text{Suc } (?c m')\) unfolding \(C\) by simp
from \(xz\) show \(\text{thesis unfolding } C\) by auto
qed
qed simp

} from this[of m] * show \(\text{thesis by auto}\)
qed

lemma relto-relto-fun-conv: \((a, b) \in (\text{relto } A B)^* \cdot n) = (\exists \ as \ \text{sel } m. \text{ relto-fun } A B \ n \ as \ \text{sel } m \ (a, b))\)
using relto-fun-into-reltos[of A B n - - - a b] retlos-into-relto-fun[of a b n B A] by blast

lemma relto-fun-intermediate: assumes \(A \subseteq C\) and \(B \subseteq C\)
and \(rf: \text{relto-fun } A B \ n \ as \ \text{sel } m \ (a, b)\)
shows \(i \leq m \implies (a, as \ i) \in C^*\)
proof (induct i)
case 0
from relto-funD[of \(rf\)] show \(\text{case by simp}\)
next
case (Suc i)
hence IH: (a, as i) ∈ C’∗ and im: i < m by auto
from relto-funD(3−j) OF rf im
assms have (as i, as (Suc i)) ∈ C by auto
with IH show ?case by auto

qed

lemma not-SN-on-rel-succ:
assumes ¬ SN-on (relto R E) {s}
shows ∃ t u. (s, t) ∈ E∗ ∧ (t, u) ∈ R ∧ ¬ SN-on (relto R E) {u}
proof –
obtain v where (s, v) ∈ relto R E and v: ¬ SN-on (relto R E) {v}
using assms by fast
moreover then obtain t and u
where (s, t) ∈ E∗ and (t, u) ∈ R and uv: (u, v) ∈ E∗ by auto
moreover from uv have uv: (u, v) ∈ (R ∪ E)∗ by regexp
moreover have ¬ SN-on (relto R E) {u} using
v steps-preserve-SN-on-relto[OF uv] by auto
ultimately show ?thesis by auto

qed

lemma SN-on-relto-relcomp: SN-on (relto R S) T = SN-on (S∗ O R) T (is ?L T = ?R T)
proof
assume L: ?L T
{ fix t assume t ∈ T hence ?L {t} using L by fast }
thus ?R T by fast
next
{ fix s
have SN-on (relto R S) {s} = SN-on (S∗ O R) {s}
proof
let ?X = {s. ¬SN-on (relto R S) {s}}
{ assume ¬ ?L {s}
hence s ∈ ?X by auto
hence ¬ ?R {s}
proof(rule lower-set-imp-not-SN-on, intro ballI)
fix s assume s ∈ ?X
then obtain t u where (s, t) ∈ S∗ (t, u) ∈ R and u: u ∈ ?X
unfolding mem-Collect-eq by (metis not-SN-on-rel-succ)
hence (s, u) ∈ S∗ O R by auto
with u show ∃ u ∈ ?X. (s, u) ∈ S∗ O R by auto
qed }
thus ?R {s} ⇒ ?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 ∈ T hence ?L {t} unfolding main using R by fast }

107
thus \( ?L T \) by fast

\[ \text{qed} \]

\textbf{lemma trans-relto:}
\textit{assumes trans: trans } \( R \) \textit{and } \( S \cap R \subseteq R \cap S \)
\textit{shows } trans (relto \( R \) \( S \))

\textbf{proof}
\textit{fix} \( a \), \( b \), \( c \)
\textit{assume} \( ab: (a, b) \in S^* O R O S^* \) \textit{and } \( bc: (b, c) \in S^* O R O S^* \)
\textit{from} \( rtrancl-O-push \) \textit{[of } \( S R \) \( ] \) \textit{have comm: } \( S^* O R \subseteq R O S^* \) \textit{by blast}
\textit{from} ab \textit{obtain } \( d e \) \textit{where } de: \( (a, d) \in S^* (d, e) \in R (e, b) \in S^* \) \textit{by auto}
\textit{from} bc \textit{obtain } \( f g \) \textit{where } fg: \( (b, f) \in S^* (f, g) \in R (g, c) \in S^* \) \textit{by auto}
\textit{from} de(3) \textit{fg(1)} \textit{have } \( (e, f) \in S^* \) \textit{by auto}
\textit{with} fg(2) \textit{comm have } \( (e, g) \in R O S^* \) \textit{by blast}
\textit{then} \textit{obtain } h \textit{where } h: \( (e, h) \in R (h, g) \in S^* \) \textit{by auto}
\textit{with} de(2) \textit{trans have } dh: \( (d, h) \in R \) \textit{unfolding} \textit{trans-def by blast}
\textit{from} fg(3) \textit{h(2)} \textit{have } \( (h, c) \in S^* \) \textit{by auto}
\textit{with} de(1) \textit{dh(1)} \textit{show } \( (a, c) \in S^* O R O S^* \) \textit{by auto}

\[ \text{qed} \]

\textbf{lemma relative-ending:}
\textit{assumes chain: chain } \( (R \cup S) \cap t \)
\textit{and } \( \emptyset: t \emptyset \in X \)
\textit{and } \( SN: SN-on \) \( (relto \) \( R \) \( S) \) \( ) \( X \)
\textit{shows } \( \exists j. \forall i \geq j. (t i, t (\text{Suc } i)) \in S \cap R \)

\textbf{proof} \textit{rule ccontr}
\textit{assume } \( \neg \) \textit{thesis}
\textit{with} \textit{chain} \textit{have } \( \forall i, \exists j. j \geq i \land (t j, t (\text{Suc } j)) \in R \) \textit{by blast}
\textit{from} \textit{choice } \textit{OF this} \textit{obtain } \( f \) \textit{where } R\textit{-steps: } \( \forall i, i \leq f i \land (t (f i), t (\text{Suc } (f i))) \in R \) \textit{by ..}
\textit{let } \( ?t = \lambda i. t (((\text{Suc } \circ f) \ 	ext{Suc } i) \emptyset ) \)
\textit{have } \( \forall i. (t i, t (\text{Suc } (f i))) \in (\text{relto } R S)^+ \)

\textbf{proof}
\textit{fix } \( i \)
\textit{from} \textit{R\text{-steps have } leq}: \( i \leq f i \) \textit{and } \( step: (t (f i), t (\text{Suc } (f i))) \in R \) \textit{by auto}
\textit{from} \textit{chain-imp-rtrancl } \textit{[OF } \textit{chain } \textit{leq]} \textit{have } \( (t i, t (f i)) \in (R \cup S)^+ \).
\textit{with} \textit{step have } \( (t i, t (\text{Suc } (f i))) \in (R \cup S)^+ O R \) \textit{by auto}
\textit{then} \textit{show } \( (t i, t (\text{Suc } f i)) \in (\text{relto } R S)^+ \) \textit{by regexp}

\textbf{qed}

\textit{then} \textit{have } \( \text{chain } (\text{relto } R S)^+ \) \( \neg \) \( ?t \) \textit{by simp}
\textit{with } \( t \emptyset \) \textit{have } \( \neg \) \textit{SN-on } \( (\text{relto } R S)^+ \) \( X \) \textit{by } \( \text{unfolding } \text{SN-on-def, auto intro: exI[of - } ?t]) \)
\textit{with} \textit{SN-on-trancl[OF } \textit{SN]} \textit{show } \textit{False } \textit{by auto}

\[ \text{qed} \]

\textit{from} \textit{Geser’s thesis } \textit{[p.32, Corollary-1], generalized for } \textit{SN-on.}

\textbf{lemma } \textit{SN-on-relto-Un:}
\textit{assumes closure: relto } \( R \cup R' \) \textit{S } \( \subseteq X \)
\textit{shows } \( \text{SN-on } (\text{relto } (R \cup R') \cap S) \) \( X \leftarrow \text{SN-on } (\text{relto } R (R' \cup S)) \) \( X \cap \text{SN-on} \)

108
\[(\mathsf{relto} R' S) X\]

(is \(?c \iff ?a \land ?b\))

**proof (safe)**

assumes \(SN: ?a\) and \(SN': ?b\)

from \(SN\) have \(SN: \mathsf{SN-on} (\mathsf{relto} (\mathsf{relto} R S) (\mathsf{relto} R' S)) X\) by (rule \(\mathsf{SN-on-subset1}\))

regexp

shows \(?c\)

**proof**

fix \(f\)

assumes \(f0: f 0 \in X\) and \(chain: \mathsf{chain} (\mathsf{relto} (R \cup R') S) f\)

then have \(\mathsf{chain} (\mathsf{relto} R S \cup \mathsf{relto} R' S) f\) by auto

from \(relative-ending[OF this f0 SN]\)

have \(\exists j. \forall i \geq j. (f i, f (Suc i)) \in \mathsf{relto} R' S - \mathsf{relto} R S\) by auto

then obtain \(j\) where \(\forall i \geq j. (f i, f (Suc i)) \in \mathsf{relto} R' S\) by auto

then have \(\mathsf{chain} (\mathsf{relto} R' S) (\mathsf{shift} f j)\) by auto

moreover have \(f j \in X\)

**proof (induct \(j\))**

- case \(0\) from \(f0\) shows \(?case\) by simp

next

- case \((Suc j)\)

  let \(?s = (f j, f (Suc j))\)

  from \(chain\) have \(?s \in \mathsf{relto} (R \cup R') S\) by auto

  with Image-closed-trancl[\(OF closure\)] Suc show \(f (Suc j) \in X\) by blast

  qed

then have \(\mathsf{shift} f j 0 \in X\) by auto

ultimately have \(\neg \mathsf{SN-on} (\mathsf{relto} R' S) X\) by (intro \(\mathsf{not-SN-onI}\))

with \(SN'\) show \(\mathsf{False}\) by auto

qed

next

assumes \(SN: ?c\)

then show \(?b\) by (rule \(\mathsf{SN-on-subset1}\), auto)

moreover

from \(SN\) have \(\mathsf{SN-on} ((\mathsf{relto} (R \cup R') S)') X\) by (unfold \(\mathsf{SN-on-trancl-SN-on-conv}\))

then show \(?a\) by (rule \(\mathsf{SN-on-subset1}\))

regexp

qed

**lemma** \(\mathsf{SN-on-Un}: (R \cup R') ^+ X \subseteq X \implies \mathsf{SN-on} (R \cup R') X \iff \mathsf{SN-on} (\mathsf{relto} R R') X \land \mathsf{SN-on} R' X\)

using \(\mathsf{SN-on-relto-Un[of \{\}\]}\) by simp

end

4 Strongly Normalizing Orders

theory \(\mathsf{SN-Orders}\)

imports \(\mathsf{Abstract-Rewriting}\)

begin

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

class non-strict-order = ord +
  assumes ge-refl: \( x \geq (x :: 'a) \)
  and ge-trans[trans]: \( [x \geq y; (y :: 'a) \geq z] \implies x \geq z \)
  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: \( z \geq 0 \implies x \geq y \implies x \times z \geq y \times z \)
  and times-right-mono: \( x \geq 0 \implies y \geq z \implies x \times y \geq x \times z \)
  and times-left-anti-mono: \( x \geq y \implies 0 \geq z \implies y \times z \geq x \times z \)

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 \( \succ \))
  and default :: 'a
  assumes compat[trans]: \( [x \geq y; y \succ z] \implies x \succ z \)
  and compat2[trans]: \( [x \succ y; y \geq z] \implies x \succ z \)
  and gt-imp-ge: \( x \succ y \implies x \geq y \)
  and default-ge-zero: \( \text{default} \geq 0 \)
begin
  lemma gt-trans[trans]: \( [x \succ y; y \succ z] \implies x \succ z \)
    by (rule compat[OF gt-imp-ge])
end

110
locale one-mono-ordered-semiring-1 = order-pair gt
  for gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool (infix :> 50) +
  assumes plus-gt-left-mono: y :> y ⇒ x + z :> y + z
  and default-gt-zero: default :> 0
begin
lemma plus-gt-right-mono: y :> y ⇒ a + x :> a + y
  unfolding add.commute[of a] by (rule plus-gt-left-mono)

lemma plus-gt-bot-right-mono: y :> y ⇒ a + x :> a + 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 +
  assumes SN: SN{(x,y). y ≥ 0 ∧ x :> y} +
locale SN-strict-mono-ordered-semiring-1 = SN-one-mono-ordered-semiring-1 +
  fixes mono :: 'a :: ordered-semiring-1 ⇒ bool
  assumes mono: [ mono x; y :> z; x ≥ 0 ] ⇒ x * y :> x * z
locale both-mono-ordered-semiring-1 = order-pair gt
  for gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool (infix :> 50) +
  assumes plus-gt-bot-right-mono: y :> y ⇒ x + z :> y + z
  and times-gt-left-mono: x :> y ⇒ x * z :> y * z
  and times-gt-right-mono: y :> y ⇒ x * y :> x * z
  and zero-leastI: x :> 0
  and zero-leastII: 0 :> x ⇒ x = 0
  and zero-leastIII: (x :: 'a) ≥ 0
  and arc-pos-one: arc-pos (1 :: 'a)
  and arc-pos-default: arc-pos default
  and arc-pos-zero: ¬ arc-pos 0
  and arc-pos-plus: arc-pos x ⇒ arc-pos (x + y)
  and arc-pos-mult: [arc-pos x; arc-pos y] ⇒ arc-pos (x * y)
  and not-all-ge: ∃ c d. arc-pos d ⇒ ∃ e. e ≥ 0 ∧ arc-pos e ∧ ¬ (c ≥ 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 ∧ x :> y} +
locale weak-SN-strict-mono-ordered-semiring-1 =
  fixes weak-gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool
  and default :: 'a
and mono :: 'a ⇒ bool
assumes weak-gt-mono: ∀ x y. (x,y) ∈ set xys → weak-gt x y ⇒ ∃ gt.
SN-strict-mono-ordered-semiring-1 default gt mono ∧ (∀ x y. (x,y) ∈ set xys →

locale weak-SN-both-mono-ordered-semiring-1 =
fixes weak-gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool and
default :: 'a and arc-pos :: 'a ⇒ bool
assumes weak-gt-both-mono:
∀ x y. (x,y) ∈ set xys → weak-gt x y =⇒ ∃ gt.
SN-both-mono-ordered-semiring-1 default gt arc-pos ∧ (∀ x y. (x,y) ∈ set xys →

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 ≻ 50) +
fixes power-mono :: bool and
discrete :: bool
assumes times-gt-mono: [ y ≻ z ; x ≥ 1 ] ⇒ y * x ≻ z * x
and power-mono: power-mono ⇒ x ≻ y ⇒ y ≥ 0 ⇒ n ≥ 1 ⇒ x ^ n ≻ y
^ n
and discrete: discrete ⇒ x ≥ y ⇒ ∃ k. x = (((+ (1 ^ k)) # k) * y)
class large-ordered-semiring-1 = poly-carrier +
assumes ex-large-of-nat: ∃ x. of-nat x ≥ y

context ordered-semiring-1

begin

lemma pow-mono: assumes ab: a ≥ b and b: b ≥ 0
shows a ^ n ≥ b ^ n ∧ b ^ n ≥ 0
proof (induct n)
case 0
show ?case by (auto simp: ge-refl one-ge-zero)
next
case (Suc n)
hence abn: a ^ n ≥ b ^ n and bnn: b ^ n ≥ 0 by auto
have bsn: b ^ Suc n ≥ 0 unfolding power-Suc
  using times-left-mono[OF bnn] by auto
have a ^ Suc n = a * a ^ n unfolding power-Suc by simp
also have ... ≥ b * a ^ n
  by (rule times-left-mono[OF ge-trans[OF abn bnn] ab])
also have b * a ^ n ≥ b * b ^ 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 ≥ (0 :: 'a)
shows a ^ n ≥ 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 ≥ (0 :: 'a :: 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) ≥ (1 :: 'a) by simp
  also have (1 :: 'a) ≥ 0 using one-ge-zero.
  finally show ?case .
qed

lemma mult-ge-zero[intro]: (a :: 'a :: ordered-semiring-1) ≥ 0 ⇒ b ≥ 0 ⇒ a * b ≥ 0
  using times-left-mono[of b 0 a] by auto

lemma pow-mono-one: assumes a: a ≥ (1 :: 'a :: ordered-semiring-1)
  shows a ^ n ≥ 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: a ≥ (1 :: 'a :: ordered-semiring-1)
  shows n ≥ m ⇒ a ^ n ≥ a ^ m
proof (induct m arbitrary: n)
  case 0
  show ?case using pow-mono-one[OF a] by auto
next
  case (Suc m mn)
  then obtain n where nn: nn = Suc n by (cases nn, auto)
  note Suc = Suc[unfolded nn]
  hence rec: a ^ n ≥ 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 :: ordered-semiring-1) ≥ 1
and \( b: b \geq 1 \)
shows \( a \ast b \geq 1 \)
proof –
from \( \text{ge-trans}\{\text{OF } b \text{ one-ge-zero}\} \) have \( b0: b \geq 0 \).
from \( \text{times-left-mono}\{\text{OF } b0 \text{ a}\} \) have \( a \ast b \geq b \) by \( \text{simp} \)
from \( \text{ge-trans}\{\text{OF } this \text{ b}\} \) show \(?\text{thesis}\).
qed

lemma \( \text{sum-list-ge-mono}\): \( \text{fixes } as : \langle\forall a :: \text{ordered-semiring-0}\rangle \text{ list} \)
assumes \( \text{length as} = \text{length bs} \)
and \( \forall i. i < \text{length bs} \Rightarrow \text{as} ! i \geq \text{bs} ! i \)
shows \( \text{sum-list as} \geq \text{sum-list bs} \)
using \( \text{assms} \)
proof (induct as arbitrary: \( bs \))

\( \text{case } (\text{Nil } bs) \)
from \( \text{Nil}(1) \) show \(?\text{case}\) by \( \text{(simp add: ge-refl)} \)

next
\( \text{case } (\text{Cons } a \text{ as } bbs) \)
from \( \text{Cons}(2) \) obtain \( b \text{ bs} \) where \( bbs: bbs = b \# \text{bs} \) and \( \text{len: length as} = \text{length bs} \)
by (cases \( bbs \), \( \text{auto} \))

\( \text{note } \text{ge} = \text{Cons}(3)[\text{unfolded } bbs] \)
\{ \( \text{fix } i \)
assume \( i < \text{length bs} \)
\( \text{hence } \text{Suc } i < \text{length (b \# bs)} \) by \( \text{simp} \)
from \( \text{ge}[\text{OF this}] \) have \( \text{as} ! i \geq \text{bs} ! i \) by \( \text{simp} \)
\} \( \text{from } \text{Cons}(1)[\text{OF len this}] \) have \( \text{IH: sum-list as} \geq \text{sum-list bs} \).
from \( \text{ge}[\text{of } 0] \) have \( ab: a \geq b \) by \( \text{simp} \)
from \( \text{ge-trans}\{\text{OF plus-left-mono}\{\text{OF } ab\} \text{ plus-right-mono}\{\text{OF } \text{IH}\}\} \)
show \(?\text{case}\) unfolding \( bbs \) by \( \text{simp} \)
qed

lemma \( \text{sum-list-ge-0-nth}\): \( \text{fixes } xs : \langle\forall a :: \text{ordered-semiring-0}\rangle \text{ list} \)
assumes \( \text{ge: } \forall i. i < \text{length xs} \Rightarrow xs ! i \geq 0 \)
shows \( \text{sum-list xs} \geq 0 \)
proof –
let \( ?! = \text{replicate } \langle\text{length xs}\rangle (0 :: 'a) \)
\( \text{have length xs} = \text{length } ?! \) by \( \text{simp} \)
from \( \text{sum-list-ge-mono}\{\text{OF this}\} \) \( \text{ge} \) have \( \text{sum-list xs} \geq \text{sum-list } ?! \) by \( \text{simp} \)
also have \( \text{sum-list } ?! = 0 \) using \( \text{sum-list-0}[\text{of } ?!] \) by \( \text{auto} \)
finally show \(?\text{thesis}\).
qed

lemma \( \text{sum-list-ge-0}\): \( \text{fixes } xs : \langle\forall a :: \text{ordered-semiring-0}\rangle \text{ list} \)
assumes \( \text{ge: } \forall x. x \in \text{set xs} \Rightarrow x \geq 0 \)
shows \( \text{sum-list xs} \geq 0 \)
by (rule \( \text{sum-list-ge-0-nth}\), \( \text{insert } \text{ge[unfolded set-conv-nth]} \), \( \text{auto} \))
lemma foldr-max: \( a \in \text{set as} \Rightarrow \text{foldr max as b} \geq (a :: 'a :: \text{ordered-ab-semigroup}) \)

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 \( \geq a \) by auto
    from ge-trans[OF - this] show ?thesis by auto
  qed
qed

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

proof 
  let \(?n = \text{of-nat :: nat} \Rightarrow 'a\)
  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 \( = \text{Suc n} \) by (cases nn, auto)
    note Suc = Suc[unfolded nn]
    hence rec: \(?n \geq ?n m \) by simp
    show ?thesis 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 \Rightarrow bool
where non-inf r \( \equiv \forall a f. \exists i. (f i, f (\text{Suc} i)) \notin r \vee (f i, a) \notin r \)

lemma non-inf[intro]: assumes \( \land a f. [ \land i. (f i, f (\text{Suc} i)) \in r ] \Rightarrow \exists i. (f i, a) \notin r \)
  shows non-inf r
  using assms unfolding non-inf-def by blast

lemma non-infE[elim]: assumes non-inf r and \( \land i. (f i, f (\text{Suc} i)) \notin r \vee (f i, a) \notin r \Rightarrow P \)
  shows P
  using assms unfolding non-inf-def by blast

115
lemma non-inf-image:
assumes ni: non-inf r and image: \( \forall a b. (a,b) \in s \implies (f a, f b) \in r \)
shows non-inf s
proof
fix a g
assume s: \( \forall i. (g i, g (Suc i)) \in s \)
define h where h = f o g
from image[OF s] have h: \( \forall i. (h i, h (Suc i)) \in r \) unfolding h-def comp-def .
from non-infE[OF ni, of h] have \( \exists a. \exists i. (h i, a) \notin r \) using h by blast
thus \( \exists i. (g i, a) \notin s \) using image unfolding h-def comp-def by blast
qed

lemma SN-imp-non-inf: SN r \implies non-inf r
by (intro non-infI, auto)

lemma non-inf-imp-SN-bound: non-inf r \implies SN \( \{(a,b). (b,c) \in r \land (a,b) \in r\} \)
by (rule, auto)

end

5 Carriers of Strongly Normalizing Orders

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

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

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

5.1 The standard semiring over the naturals

instantiation nat :: large-ordered-semiring-1
begin
instance by (intro-classes, auto)
end

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

interpretation nat-SN: SN-strict-mono-ordered-semiring-1 1 (> ) :: nat \Rightarrow nat
\Rightarrow bool nat-mono
by (unfold-locales, insert SN-nat-gt, auto simp: nat-mono-def)

interpretation nat-poly: poly-order-carrier 1 (> ) :: nat \Rightarrow nat \Rightarrow bool True
discrete
proof (unfold-locales)
fix x y :: nat
assume ge: x ≥ y
obtain k where k: x − y = k by auto
show ∃ k. x = ((+) 1 "" k) y
proof (rule exI[of - k])
from ge k have x = k + y by simp
also have ... = ((+) 1 "" k) y
by (induct k, auto)
finally show x = ((+) 1 "" k) y.
qed
qed (auto simp: field-simps power-strict-mono)

5.2 The standard semiring over the Archimedean fields using
delta-orderings
definition delta-gt :: 'a :: floor-ceiling ⇒ 'a ⇒ bool where
delta-gt δ ≡ (λ x y. x − y ≥ δ)

lemma non-inf-delta-gt: assumes delta: δ > 0
sows non-inf {(a,b) . delta-gt δ a b} (is non-inf ?r)
proof
let ?gt = delta-gt δ
fix a :: 'a and f
assume ⋀ i. (f i, f (Suc i)) ∈ ?r
hence gt: ⋀ i. ?gt (f i) (f (Suc i)) by simp
{
  fix i
  have f i ≤ f 0 − δ * 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
}

note fi = this
{
  fix r :: 'a
  have of-nat (nat (ceiling r)) ≥ 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 ceil-elim = this
define i where i = nat (ceiling ((f 0 − a) / δ))
from fi[of i] have f i − f 0 ≤ − δ * of-nat (nat (ceiling ((f 0 − a) / δ)))
unfolding i-def by simp
also have ... ≤ − δ * ((f 0 − a) / δ) using ceil-elim[of (f 0 − a) / δ] delta
  by (metis le-imp-neg-le minus-mult-commute mult-le-cancel-left-pos)
also have ... = − f 0 + a using delta by auto
also have ... < − f 0 + a + δ using delta by auto
finally have ¬ ?gt (f i) a unfolding delta-gt-def by arith
thus ∃ i. (f i, a) /∈ ?r by blast
lemma delta-gt-SN: assumes dpos: δ > 0 shows SN \{(x,y). 0 ≤ y ∧ delta-gt δ 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 ⇒ bool where
delta-mono x ≡ x ≥ 1
subclass (in floor-ceiling) large-ordered-semiring-1
proof
  fix x :: 'a
  from ex-le-of-int[of x] obtain z where x: x ≤ of-int z by auto
  have z ≤ int (nat z) by auto
  with x have x ≤ of-int (int (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 ... = of-nat (nat z) unfolding of-int-of-nat-eq..
  finally
  show ∃ y. x ≤ 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: δ > 0 and default: δ ≤ def
  shows SN-strict-mono-ordered-semiring-1 def (delta-gt δ) delta-mono
proof
  from dpos default have defz: 0 ≤ def by auto
  show ?thesis
  proof (unfold-locales)
    show SN \{(x,y). y ≥ 0 ∧ delta-gt δ x y\} by (rule delta-gt-SN[OF dpos])
  next
    fix x y z :: 'a
    assume delta-mono x and yz: delta-gt δ y z
    hence x: 1 ≤ x unfolding delta-mono-def by simp
    have ∃ d > 0, delta-gt δ = (λ x y. d ≤ 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 δ = (λ x y. d ≤ x − y)
    by auto
    from yz have yzd: d ≤ y − z by (simp add: rat)
    show delta-gt δ (x * y) (x * z)
    proof (simp only: rat)
      let ?p = (x − 1) * (y − z)
      from x have x1: 0 ≤ x − 1 by auto
      from yzd d have yzd0: 0 ≤ y − z by auto
      have 0 ≤ ?p
      by (rule mult-nonneg-nonneg[OF x1 yzd0])
      have x * y − x * z = x * (y − z) using right-diff-distrib[of x y z] by auto
  qed
also have \( \ldots = (x - 1 + 1) \times (y - z) \) by auto
also have \( \ldots = ?p + 1 \times (y - z) \) by (rule ring-distrib(2))
also have \( \ldots = ?p + (y - z) \) by simp
also have \( \ldots \geq (\theta + d) \) using gcd (\( \theta \leq ?p \)) by auto
finally
show \( d \leq x \times y - x \times z \) by auto
qed

qed (insert dpos, auto simp: delta-gt-def default defz)

lemma delta-poly: assumes dpos: \( \delta > 0 \) and default: \( \delta \leq \text{def} \)
sows 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 \( \delta \) \( y \times z \) 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 \( \delta \) \((y \times x)\) \((z \times x)\)
  using mono[OF m gt ge] by (auto simp: field-simps)
next
  fix \( x \) \( y \)::'a
  assume False
  thus \( \exists k \) \( x = ((+) 1 \sim k) y \) by simp
qed

show \(?thesis
proof (unfold-locales)
  fix \( x \) \( y \)::'a and \( n :: \text{nat} \)
  assume one: \( 1 \leq \delta \) and gt: delta-gt \( \delta \) \( x \times y \) and \( y \geq 0 \) and \( n :: 1 \leq n \)
  then obtain \( p \) where \( n = \text{Suc} \ p \) 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 \( \delta \) \((x \sim n)\) \((y \sim n)\)
  proof (simp only: \( n \), induct \( p \), simp add: gt)
    case (Suc \( p \))
    from times-gt-mono[OF this \( x \)]
    have one: delta-gt \( \delta \) \((x \sim \text{Suc} \ (\text{Suc} \ p))\) \((x \times y \sim \text{Suc} \ p)\) by (auto simp: field-simps)
    also have \( \ldots \geq y \times y \sim \text{Suc} \ p \)
    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

119
fix x y :: 'a
assume False
thus ∃ k. x = ((+ 1 ^^ k)) y by simp
qed (rule times-gt-mono, auto)
qed

lemma delta-minimal-delta: assumes (∃ x y. (x,y) ∈ set xys =⇒ x > y) shows (∃ δ > 0. ∀ x y. (x,y) ∈ set xys =⇒ delta-gt δ 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 ≤ x - y by auto
hence xy: delta-gt d1 x y unfolding delta-gt-def by auto
from Cons obtain d2 where d2pos: d2 > 0 and xys: ∀ x y. (x, y) ∈ set xys =⇒ 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 delta-gt d x y unfolding delta-gt-def by auto
with xys d Pair have ∀ x y. (x,y) ∈ set (xy # xys) =⇒ 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 × 'a) list
assume orient: ∀ x y. (x,y) ∈ set xysp =⇒ x > y
obtain xys where xys = (1,0) ≠ xysp by auto
with orient have (∃ x y. (x,y) ∈ set xys =⇒ x > y) by auto
with delta-minimal-delta have (∃ δ > 0. ∀ x y. (x, y) ∈ set xys =⇒ delta-gt δ x y)
by auto
then obtain δ where dpos: δ > 0 and orient: (∃ x y. (x,y) ∈ set xys =⇒ delta-gt δ x y by auto
from orient have orient1: ∀ x y. (x,y) ∈ set xysp =⇒ delta-gt δ x y and orient2: delta-gt δ 1 0 unfolding xys by auto
from orient2 have oned: δ ≤ 1 unfolding delta-gt-def by auto
show (∃ gt. SN-strict-mono-ordered-semiring-1 1 gt delta-mono ∧ (∀ x y. (x, y) ∈ set xysp =⇒ gt x y)
by (intro exI conjI, rule delta-interpretation[OF dpos oned], rule orient1)
5.3 The standard semiring over the integers

**Definition** \( \text{int-mono} :: \text{int} \Rightarrow \text{bool} \) where \( \text{int-mono} \ x \equiv x \geq 1 \)

**Instantiation** \( \text{int} :: \text{large-ordered-semiring-1} \)

**Proof**
- Fix \( y :: \text{int} \)
- Show \( \exists \ x. \text{of-nat} \ x \geq y \)
  - By (rule exI[of - nat y], simp)
- QED (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg)

**Lemma** \( \text{non-inf-int-gt} : \text{non-inf} \{(a,b :: \text{int}) . \ a > b\} \) by (rule non-inf-image[OF non-inf-delta-gt, of 1 - rat-of-int], auto simp: delta-gt-def)

**Interpretation** \( \text{int-SN} : \text{SN-strict-mono-ordered-semiring-1} 1 (>) :: \text{int} \Rightarrow \text{int} \Rightarrow \text{bool} \text{ int-mono} \)
- Have [simp]: \( \forall x :: \text{int}. \ (-1 < x) = (0 \leq x) \) by auto
- Show \( \text{SN} \{(x,y). \ y \geq 0 \land (y :: \text{int}) < x\} \)
  - 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** \( \text{int-poly} : \text{poly-order-carrier} 1 (>) :: \text{int} \Rightarrow \text{int} \Rightarrow \text{bool} \text{ True discrete} \)
- Fix \( x y :: \text{int} \)
- Assume \( gc : x \geq y \)
- Then obtain \( k \) where \( k : x - y = k \text{ and } kp : \theta \leq k \) by auto
- Then obtain \( nk \) where \( nk : nk = \text{nat} k \text{ and } k : x - y = \text{int} nk \) by auto
- Show \( \exists k. \ x = ((+) 1 ^ k) y \)
- Proof (rule exI[of - nk])
  - From \( k \) have \( x = \text{int} nk + y \) by simp
  - Also have \( ... = ((+) 1 ^ nk) y \)
    - By (induct nk, auto)
  - Finally show \( x = ((+) 1 ^ nk) y \).
- 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** \( \text{arctic} = \text{MinInfty} | \text{Num-arc} \text{ int} \)

**Instantiation** \( \text{arctic} :: \text{ord} \)
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)

end
show \( x \times 1 = x \)
by (cases \( x \), simp-all add: one-arctic-def)

show \( 0 + x = x \)
by (simp add: zero-arctic-def)

show \( 0 \times x = 0 \)
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 \( ge: x \geq y \)
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 \( x \times z \geq y \times z \)
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 \( x y z :: \text{arctic} \)
assume \( x \geq y \) and \( y \geq z \)
thus \( x \geq z \)
by (cases \( x \), cases \( y \), cases \( y \), cases \( z \), auto, cases \( z \), auto, cases \( z \), auto)

next
fix \( x y z :: \text{arctic} \)
assume \( y \geq z \)
thus \( x \times y \geq x \times z \)
by (cases \( x \), cases \( y \), cases \( z \), auto, cases \( y \), cases \( z \), auto, cases \( z \), auto)

next
fix \( x y z :: \text{arctic} \)
show \( x \geq y \implies 0 \geq z \implies y \times z \geq x \times z \)
by (cases \( z \), cases \( x \), auto simp: zero-arctic-def)

qed

derived
where pos-arctic MinInfty = False
    | pos-arctic (Num-arc n) = (0 <= n)

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

proof
  fix x y z :: arctic
  assume x >= 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 >= 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 >= 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) >= 0 unfolding zero-arctic-def by simp
  show x >= 0 unfolding zero-arctic-def by simp
  show ~ 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 \((x \ast y)\) by \((\text{cases } x, \text{ simp}, \text{ cases } y, \text{ auto})\)

next

show \(\{ (x,y). \text{pos-arctic } y \land x > y \} \) \((\text{is } \text{SN } ?\text{rel})\)

proof = \{

fix \(x\)

assume \(\exists f : f \, 0 = x \land (\forall i. \langle f \, i, f \, (\text{Suc } i) \rangle \in ?\text{rel})\)

from \(\text{this} \) obtain \(f \) where \(f \, 0 = x \) and seq: \(\forall i. \langle f \, i, f \, (\text{Suc } i) \rangle \in ?\text{rel}\) by \(\text{auto}\)

from \(\text{seq} \) have \(\text{steps}: \forall i. f \, i > (\text{Suc } i) \land \text{pos-arctic } (f \, (\text{Suc } i)) \) by \(\text{auto}\)

from \(i \) obtain \(n \) where \(f \, i = \text{Num-arc } n \) by \((\text{cases } f \, (\text{Suc } i), \text{ simp}, \text{ cases } z, \text{ simp}, \text{ cases } y, \text{ auto})\)

with \(i \) have \(\text{gz}: 0 \leq m \) by \(\text{simp}\)

from \(i \) fi fsi have \(n > m \) by \(\text{auto}\)

with \(f \) fsi \(\text{gz}\)

show \(\forall i. g \, (\text{Suc } i) \geq 0 \land g \, i > g \, (\text{Suc } i)\) by \(\text{auto}\)

qed

from \(\text{this} \) obtain \(g \) where \(\forall i. g \, (\text{Suc } i) \geq 0 \land (\forall i. \langle f \, i, f \, (\text{Suc } i) \rangle \in \{(x,y). y \geq 0 \land x > y\})\) by \(\text{auto}\)

with \(\text{int-SN} \, \text{SN} \) have \(\text{False}\) unfolding \(\text{SN-defs}\) by \(\text{auto}\)

\}

thus ?\text{thesis} unfolding \(\text{SN-defs}\) by \(\text{auto}\)

qed

next

fix \(y\) \(z\) \(x\) :: \text{arctic}

assume \(y > z\)

thus \(x \ast y > x \ast z\)

by \((\text{cases } x, \text{ simp}, \text{ cases } z, \text{ simp}, \text{ cases } y, \text{ auto})\)

next

fix \(c\) \(d\)

assume \(\text{pos-arctic } d\)

then obtain \(n \) where \(d = \text{Num-arc } n \) and \(n: 0 \leq n\)

by \((\text{cases } d, \text{ auto})\)

show \(\exists e. e \geq 0 \land \text{pos-arctic } e \land \neg c \geq d \ast e\)

proof \((\text{cases } c)\)

case \(\text{MinInfty}\)

show ?\text{thesis}

by \((\text{rule exI[of - \text{Num-arc } 0]},\)

unfold \(\text{d MinInfty zero-arctic-def, simp})\)

next

case \((\text{Num-arc } m)\)

125
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 ⇒ 'a arctic-delta ⇒ bool where
  less-eq-arctic-delta MinInfty-delta x = True
| less-eq-arctic-delta (Num-arc-delta y) MinInfty-delta = False
| less-eq-arctic-delta (Num-arc-delta y) (Num-arc-delta x) = (y ≤ x)

fun less-arctic-delta :: 'a arctic-delta ⇒ 'a arctic-delta ⇒ bool where
  less-arctic-delta MinInfty-delta x = True
| less-arctic-delta (Num-arc-delta y) 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 ⇒ 'a arctic-delta ⇒ '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 ⇒ 'a arctic-delta ⇒ '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 \( (x \ast y) \ast z = x \ast (y \ast z) \)
   by (cases \( x \), auto, cases \( y \), auto, cases \( z \), auto)
show \( x \ast 0 = 0 \)
   by (cases \( x \), auto simp: zero-arctic-delta-def)
show \( x \ast (y + z) = x \ast y + x \ast z \)
   by (cases \( x \), auto, cases \( y \), auto, cases \( z \), auto)
show \( (x + y) \ast z = x \ast z + y \ast z \)
   by (cases \( x \), auto, cases \( y \), cases \( z \), auto, cases \( z \), auto)
show \( 1 \ast x = x \)
   by (cases \( x \), simp-all add: one-arctic-delta-def)
show \( x \ast 1 = x \)
   by (cases \( x \), simp-all add: one-arctic-delta-def)
show \( 0 + x = x \)
   by (simp add: zero-arctic-delta-def)
show \( 0 \ast x = 0 \)
   by (simp add: zero-arctic-delta-def)
show \( (0 :: 'a arctic-delta) \neq 1 \)
   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 \( (1 :: 'a arctic-delta) \geq 0 \)
   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 \( \geq: x \geq y \)
from \( \geq \) show \( x + z \geq y + z \)
   by (cases \( x \), cases \( y \), cases \( z \), auto, cases \( y \), cases \( z \), auto, cases \( z \), auto)
from \( \geq \) show \( x \ast z \geq y \ast z \)
   by (cases \( x \), cases \( y \), cases \( z \), auto, cases \( y \), cases \( z \), auto, cases \( z \), auto)
from \( \geq \) show \( \max x y = x \) unfolding max-def
   by (cases \( x \), (cases \( y \), auto)+)
from \( \geq \) show \( \max x y \geq \max z y \) unfolding max-def
   by (cases \( z \), cases \( x \), auto, cases \( x \), (cases \( y \), auto)+)

next
fix \( x y z :: 'a arctic-delta \)
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 \( x y z :: 'a arctic-delta \)
assume \( y \geq z \)
thus \( x \ast y \geq x \ast z \)
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: δ > 0 shows SN-both-mono-ordered-semiring-1
  δ (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, 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 δ x y
  thus gt-arctic-delta δ (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 δ 0 x
  thus x = 0
    by (cases x, auto simp: zero-arctic-delta-def)

next
  fix x y :: 'a arctic-delta
  assume pos-arctic-delta x
  thus pos-arctic-delta (x + y)
    by (cases x, auto)

next
  show pos-arctic-delta 1 unfolding one-arctic-delta-def

next
  show SN {(x, y). pos-arctic-delta y ∧ gt-arctic-delta δ 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. gt-arctic-delta δ (f i) (f (Suc i)) ∧ pos-arctic-delta
    (f (Suc i)) by auto
    let ?g = λ i. get-arctic-delta-num (f i)
    have ∀ i. ?g (Suc i) ≥ 0 ∧ delta-gt δ (?g i) (?g (Suc i))
    proof
      fix i
      from steps have i: gt-arctic-delta δ (f i) (f (Suc i)) ∧ 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 ≤ m by simp
      from i fi fsi have delta-gt δ n m by auto
      with fi fsi gz
show \(?g \, (\text{Suc} \, i) \geq 0 \land \text{delta-gt} \, \delta \, (?g \, i) \, (?g \, (\text{Suc} \, i))\) by auto
qed
from this obtain \(g\) where \(\forall \, i. \, g \, (\text{Suc} \, i) \geq 0 \land \text{delta-gt} \, \delta \, (g \, i) \, (g \, (\text{Suc} \, i))\)
by auto
hence \(\exists \, f. \, f \, 0 \, = \, g \, 0 \land (\forall \, i. \, (f \, i, \, f \, (\text{Suc} \, i)) \in \{(x,y). \, y \geq 0 \land \text{delta-gt} \, \delta \, x \, y\})\) by auto
with SN have False unfolding SN-defs by auto
}
thus \(?thesis\) unfolding SN-defs by auto
qed
next
fix \(c \, d\) :: \('a \text{ arctic-delta}\)
assume \(\text{pos-arctic-delta} \, d\)
then obtain \(n\) where \(d \, = \, \text{Num-arc-delta} \, n\) and \(n: \, 0 \leq \, n\)
by (cases \(d\), auto)
show \(\exists \, e. \, e \geq 0 \land \text{pos-arctic-delta} \, e \land \neg \, c \geq \, d \ast \, e\)
proof (cases \(c\))
case \(\text{MinInfty-delta}\)
show \(?thesis\)
by (rule exI[of - \, \text{Num-arc-delta} \, 0],
unfold \(d\) MinInfty-delta zero-arctic-delta-def, simp)
next
case \(\text{Num-arc-delta} \, m\)
show \(?thesis\)
by (rule exI[of - \, \text{Num-arc-delta} \, (abs \, m \, + \, 1)],
insert \(n\),
unfold \(d\) MinInfty-delta zero-arctic-delta-def, simp)
qed
next
fix \(x \, y \, z\)
assume \(\text{gt}: \, \text{gt-arctic-delta} \, \delta \, y \, z\)
{ 
fix \(x \, y \, z\)
assume \(\text{gt}: \, \text{delta-gt} \, \delta \, y \, z\)
have \(\text{delta-gt} \, \delta \, (x \, + \, y) \, (x \, + \, z)\)
using plus-gt-left-mono[OF \(\text{gt}\)] by (auto simp: field-simps)
}
with \(\text{gt}\) 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 ⇒ 'a arctic-delta ⇒ 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: ∀ x y. (x,y) ∈ set xys → weak-gt-arctic-delta x y
obtain xysp where xysp: xysp = map (λ (ax, ay). (case ax of Num-arc-delta x ⇒ x , case ay of Num-arc-delta y ⇒ y)) \((\text{filter} (λ (ax,ay). ax \neq \text{MinInfty-delta} \land ay \neq \text{MinInfty-delta}) xys)\)
(is - = map ?f -)
by auto
have ∀ x y. (x,y) ∈ set xysp → x > y
proof (intro allI implI)
fix x y
assume (x,y) ∈ set xysp
with xysp obtain az ay where (az,ay) ∈ set xys and ax \neq \text{MinInfty-delta}
and ay \neq \text{MinInfty-delta} and (x,y) = ?f (az,ay) by auto
hence (Num-arc-delta x, Num-arc-delta y) ∈ set xys by (cases ax, simp, cases ay, auto)
with orient show x > y by force
qed

with delta-minimal-delta[of xysp] obtain δ where dpos: δ > 0 and orient2: \(\forall x y. (x,y) \in \text{set xysp} \implies \text{delta-gt}\ \delta\ \ x\ \ y\) by auto
have orient: ∀ x y. (x,y) ∈ set xys → gt-arctic-delta \(\delta\ \ x\ \ y\)
proof(intro allI implI)
fix ax ay
assume axay: (ax,ay) ∈ set xys
with orient have orient: weak-gt-arctic-delta ax ay by auto
show gt-arctic-delta \(\delta\ \ ax\ \ ay\)
proof (cases ax, simp)
case (Num-arc-delta y) note ay = this
show ?thesis
proof (cases ax)
case \text{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) ∈ set xysp unfolding xysp by force
from ax ay orient2 [OF this] show ?thesis by simp
qed
qed
qed
show ∃ gt. SN-both-mono-ordered-semiring-1 \(\text{gt}\ \text{pos-arctic-delta} \land (\forall x y. (x, y) \in \text{set xys} \implies \text{gt}\ \ x\ \ y)\)
by (intro exI conjI, rule arctic-delta-interpretation[OF dpos], rule orient)
qed

end
References

