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.

Contents

1 Infinite Sequences .............................................. 2
   1.1 Operations on Infinite Sequences .......................... 2
   1.2 Predicates on Natural Numbers .............................. 4
   1.3 Assembling Infinite Words from Finite Words ............. 7

2 Abstract Rewrite Systems ........................................ 13
   2.1 Definitions .................................................. 13
   2.2 Properties of ARSs ........................................... 18
   2.3 Newman’s Lemma .............................................. 37
   2.4 Commutation .................................................. 43
   2.5 Strong Normalization ......................................... 47
   2.6 Terminating part of a relation ............................... 62

3 Relative Rewriting ................................................. 72

4 Strongly Normalizing Orders ..................................... 109

\(^1\)http://cl-informatik.uibk.ac.at/software/ceta
Carriers of Strongly Normalizing Orders

5.1 The standard semiring over the naturals
5.2 The standard semiring over the Archimedean fields using delta-orderings
5.3 The standard semiring over the integers
5.4 The arctic semiring over the integers
5.5 The arctic semiring over an arbitrary archimedean field

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

1 Infinite Sequences

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

Infinite sequences are represented by functions of type \( \text{nat} \Rightarrow \text{'a} \).

type-synonym \('a seq = \text{nat} \Rightarrow \text{'a}\)

1.1 Operations on Infinite Sequences

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

abbreviation (input) chainp :: \('a \Rightarrow \text{'a} \Rightarrow \text{bool}\) ⇒ \text{'}a seq ⇒ \text{bool} where
chainp \text{P} \text{S} ≡ \forall \text{i}. \text{P} \text{S}\text{i} \Rightarrow \text{'}a seq ⇒ \text{bool}

Special version for relations.

abbreviation (input) chain :: \('a \Rightarrow \text{'a} \Rightarrow \text{bool}\) ⇒ \text{'}a seq ⇒ \text{bool} where
chain \text{r} \text{S} ≡ \text{chainp} \text{(λ} x y. (x, y) ∈ \text{r}) \text{S}

Extending a chain at the front.

lemma cons-chainp:
  assumes \text{P} \text{x} (\text{S} \text{0}) and \text{chainp} \text{P} \text{S}
  shows \text{chainp} \text{P} (\text{case-nat} \text{x} \text{S}) (\text{is} \text{chainp} \text{P} \text{?S})
proof
  fix \text{i} show \text{P} (?\text{S} \text{i}) (?\text{S} (\text{Suc} \text{i})) using \text{assms} by (cases \text{i}) simp-all qed

Special version for relations.

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

A chain admits arbitrary transitive steps.

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

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

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

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

A chain admits arbitrary reflexive and transitive steps.

lemma chainp-imp-rtranclp:
\[\text{assumes } \text{chainp } \text{P } S \text{ and } i \leq j \text{ shows } \text{P}^{\sim \ast} (S \ i) (S \ j) \]
proof (cases \( j \) arbitrary: \( j \))
\[\text{from } \text{assms} \langle 2 \rangle \text{ obtain } n \text{ where } j = i + n \text{ by } \text{auto} \]
\[\text{with } \text{chain-imp-relpowp}[\text{OF } \text{P } S \text{ Suc } n \ i, \ \text{OF } \text{assms} \langle 1 \rangle], \ \text{of } \text{n i} \]
\[\text{show } ?\text{thesis by (simp add: relpow-imp-rtranclp}[\text{of } \text{S } i, \text{S } (i + n)], \text{to-pred}) \]
qed

lemma chain-imp-rtrancl:
\[\text{assumes } \text{chain } r S \text{ and } i \leq j \text{ shows } (S \ i, S \ j) \in r^{\sim \ast} \]
proof (cases \( j \) arbitrary: \( j \))
\[\text{from } \text{assms} \langle 2 \rangle \text{ obtain } n \text{ where } j = i + n \text{ by } \text{auto} \]
\[\text{with } \text{chain-imp-relpow}[\text{OF } \text{assms} \langle 1 \rangle], \ \text{of } \text{i n} \]
\[\text{show } ?\text{thesis by simp add: relpow-imp-rtranclp} \]
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' :
\[\text{assumes } \forall i \geq n :: \text{nat}. \ f \ i \geq i \ \land \ P (S \ i) (S \ (f \ i)) \]
\[\text{shows } \text{chainp } \text{P } (\text{\lambda} i. \ S \ ((f ^{\sim i} i) \ n)) \ (\text{is chainp } \text{P } ?T) \]
proof
  fix i
  from assms have (f i) n ≥ n by (induct i) auto
with assms[THEN spec[of - (f i) n]]
  show P (?T i) (?T (Suc i)) by simp
qed

lemma stepfun-imp-chainp:
  assumes ∀ i≥n::nat. f i > i ∧ P (S i) (S (f i))
  shows chainp P (λi. S ((f i) n)) (is chainp P ?T)
using stepfun-imp-chainp[of f P S] and assms by force

lemma subchain:
  assumes ∀ i::nat>n. ∃ j>i. P (f i) (f j)
  shows ∃ ϕ. (∀ i j. i < j → ϕ i < ϕ j) ∧ (∀ i. P (f (ϕ i)) (f (ϕ (Suc i))))
proof –
  from assms have ∀ i∈{i. i > n}. ∃ j>i. P (f i) (f j) by simp
from bchoice [OF this] obtain g
  where *: ∀ i>n. g i > i
  and **: ∀ i>n. P (f i) (f (g i)) by auto
define ϕ where [simp]: ϕ i = (g i) (Suc n) for i
from * have ***: ∀ i. ϕ i > n by (induct-tac i) auto
then have ∀ i. ϕ i < ϕ (Suc i) using * by (induct-tac i) auto
then have ∀ i j. i < j → ϕ i < ϕ j by (rule lift-Suc-mono-less)
moreover have ∀ i. P (f (ϕ i)) (f (ϕ (Suc i))) using ** and *** by simp
ultimately show ?thesis by blast
qed

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

lemma steps-imp-chainp':
  assumes ∀ i≥n::nat. ∃ j>i. P (S i) (S j) shows ∃ T. chainp P T
proof –
  from assms have ∀ i∈{i. i ≥ n}. ∃ j>i. P (S i) (S j) by auto
from bchoice [OF this]
  obtain f where ∀ i≥n. f i ≥ i ∧ P (S i) (S (f i)) by auto
from stepfun-imp-chainp[of f P S, OF this] show ?thesis by fast
qed

lemma steps-imp-chainp:
  assumes ∀ i≥n::nat. ∃ j>i. P (S i) (S j) shows ∃ T. 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 :: nat ⇒ bool
assumes infinite: INFM j. p j

begin

lemma inf: ∃ j ≥ 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 ≥ Suc (index n) ∧ p k by auto
  with LeastI[of λ k. p k ∧ k > index n k] show ?case by auto
qed

lemma index-ordered: index n < index (Suc n)
proof –
  from inf obtain k where k ≥ Suc (index n) ∧ p k by auto
  with LeastI[of λ k. p k ∧ k > index n k] show ?thesis by auto
qed

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

lemma index-ordered-le:
  assumes i ≤ j shows index i ≤ index j
proof –
  from assms have j = i + (j - i) by auto
  then obtain k where j: j = i + k by auto
  have index i ≤ 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 $lt$: $\text{index } i + j < \text{index } (\text{Suc } i)$ by auto
  hence $\text{index } l + u < \text{index } (\text{Suc } i)$ by auto
  show ?case
    proof (cases $\text{index } l + (\text{Suc } u) = \text{index } (\text{Suc } i)$)
      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 $i: \text{i < index } 0$ shows $\neg p \text{ i}$
proof –
  from $i$[simplified index.simps] have $i < \text{Least } p$.
  from not-less-Least[OF this] show ?thesis.
qed

end
1.3 Assembling Infinite Words from Finite Words

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

fun inf-concat-simple :: (nat ⇒ nat) ⇒ nat ⇒ (nat × nat) where
  inf-concat-simple f 0 = (0, 0)
  inf-concat-simple f (Suc n) = (
    let (i, j) = inf-concat-simple f n in
    if Suc j < f i then (i, Suc j)
    else (Suc i, 0))

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

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

lemma inf-concat-simple-surj:
  assumes j < f i
shows \( \exists k. \inf-concat-simple f k = (i,j) \)

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

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

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

lemma inf-concat-bounds:
assumes \( \text{inf} : \text{INFM } i. n i > 0 \)
and \( \text{res} : \text{inf-concat n k} = (i,j) \)
sshows \( j < n i \)
proof (cases \( k \))
  case \( 0 \)
  with \( \text{res} \) have \( i : (\text{LEAST } i. n i > 0) \text{ and } j : j = 0 \) by auto
  from \( \text{inf[unfolded INFM-nat-le]} \) obtain \( i' \) where \( i' : 0 < n i' \) by auto
  have \( 0 < n (\text{LEAST } i. n i > 0) \)
  by (rule LeastI, rule \( i' \))
  with \( i \) \( j \) show \( ?\text{thesis} \) by auto
next
  case \( (\text{Suc } k') \)
  obtain \( i' \) \( j' \) where \( \text{res'} : \text{inf-concat n k'} = (i'j') \) by force
  note \( \text{res} = \text{res[unfolded Suc inf-concat.simps res'} Let-def split] \)
  show \( ?\text{thesis} \)
  proof (cases \( \text{Suc } j' < n i' \))
case \textit{True} \\
\textbf{with} \textit{res} \textbf{show} \ ?\textit{thesis} \textbf{by} \textit{auto} \\
\textbf{next} \\
\textbf{case} \textit{False} \\
\textbf{with} \textit{res} \textbf{have} \textit{i}: \textit{i} = (\text{LEAST} \textit{f}. \textit{i}' < \textit{f} \land \textit{0} < \textit{n} \textit{f}) \ \textbf{and} \ \textit{j}: \textit{j} = \textit{0} \textbf{by} \ \textit{auto} \\
\textbf{from} \ \textit{inf}[\text{unfolded INFM-nat}] \ \textbf{obtain} \ \textit{f} \ \textbf{where} \ \textit{f}: \textit{i}' < \textit{f} \land \textit{0} < \textit{n} \textit{f} \ \textbf{by} \ \textit{auto} \\
\textbf{have} \textit{0} < \textit{n} \ (\text{LEAST} \textit{f}. \textit{i}' < \textit{f} \land \textit{0} < \textit{n} \textit{f}) \\
\quad \textbf{using} \ \text{LeastI}[\text{of} \ \lambda \textit{f}. \textit{i}' < \textit{f} \land \textit{0} < \textit{n} \textit{f}, \ \textit{OF} \ \textit{f}] \\
\quad \textbf{by} \ \textit{auto} \\
\textbf{with} \ \textit{i} \ \textit{j} \ \textbf{show} \ ?\textit{thesis} \ \textbf{by} \ \textit{auto} \\
\textbf{qed} \\
\textbf{qed} \\
\textbf{lemma} \ \textit{inf-concat-add}: \\
\textbf{assumes} \ \textit{res}: \ \textit{inf-concat} \ \textit{n} \ \textit{k} = (\textit{i}, \textit{j}) \\
\quad \textbf{and} \ \textit{j}: \ \textit{j} + \textit{m} < \textit{n} \ \textit{i} \\
\textbf{shows} \ \textit{inf-concat} \ \textit{n} \ (\textit{k} + \textit{m}) = (\textit{i}, \textit{j} + \textit{m}) \\
\textbf{using} \ \textit{j} \\
\textbf{proof} \ (\textit{induct} \ \textit{m}) \\
\textbf{case} \ \textit{0} \ \textbf{show} \ ?\textit{case} \ \textbf{using} \ \textit{res} \ \textbf{by} \ \textit{auto} \\
\textbf{next} \\
\textbf{case} \ (\textbf{Suc} \ \textit{m}) \\
\quad \textbf{hence} \ \textit{inf-concat} \ \textit{n} \ (\textit{k} + \textit{m}) = (\textit{i}, \textit{j} + \textit{m}) \ \textbf{by} \ \textit{auto} \\
\quad \textbf{with} \ \textit{Suc}(2) \\
\quad \textbf{show} \ ?\textit{case} \ \textbf{by} \ \textit{auto} \\
\textbf{qed} \\
\textbf{lemma} \ \textit{inf-concat-step}: \\
\textbf{assumes} \ \textit{res}: \ \textit{inf-concat} \ \textit{n} \ \textit{k} = (\textit{i}, \textit{j}) \\
\quad \textbf{and} \ \textit{j}: \ \text{Suc} \ (\textit{j} + \textit{m}) = \textit{n} \ \textit{i} \\
\textbf{shows} \ \textit{inf-concat} \ \textit{n} \ (\textit{k} + \text{Suc} \ \textit{m}) = \ (\text{LEAST} \ \textit{i}''. \ \textit{i}''. > \textit{i} \land \textit{0} < \textit{n} \textit{i}'', \textit{0}) \\
\textbf{proof} - \\
\quad \textbf{from} \ \textit{j} \ \textbf{have} \ \textit{j} + \textit{m} < \textit{n} \ \textit{i} \ \textbf{by} \ \textit{auto} \\
\quad \textbf{note} \ \textit{res} = \ \textit{inf-concat-add}[\text{OF} \ \textit{res}, \ \textit{OF} \ \textit{this}] \\
\quad \textbf{show} \ ?\textit{thesis} \ \textbf{by} \ (\textit{simp add:} \ \textit{res} \ \textit{j}) \\
\textbf{qed} \\
\textbf{lemma} \ \textit{inf-concat-surj-zero}: \\
\textbf{assumes} \ \textit{0} < \textit{n} \ \textit{i} \\
\textbf{shows} \ \exists \textit{k}. \ \textit{inf-concat} \ \textit{n} \ \textit{k} = (\textit{i}, \textit{0}) \\
\textbf{proof} - \\
\quad \textbf{fix} \ \textit{l} \\
\quad \textbf{have} \ \forall \ \textit{j}. \ \textit{j} < \textit{l} \land \textit{0} < \textit{n} \ \textit{j} \ \rightarrow \ (\exists \ \textit{k}. \ \textit{inf-concat} \ \textit{n} \ \textit{k} = (\textit{j},\textit{0})) \\
\quad \textbf{proof} \ (\textit{induct} \ \textit{l}) \\
\quad \textbf{case} \ \textit{0} \\
\quad \quad \textbf{thus} \ ?\textit{case} \ \textbf{by} \ \textit{auto} \\
\quad \textbf{next} \\
\quad \textbf{case} \ (\textbf{Suc} \ \textit{l})
show \(?\)case

proof (intro allI impI, elim conjE)
  
  fix \(j\)

  assume \(j : j < \text{Suc } l \text{ and } nj : 0 < n j\)

  show \(\exists k. \text{inf-concat } n k = (j, 0)\)

  proof (cases \(j < l\))

  case True

  from Suc[THEN spec[of - j]] True nj show \(?\)thesis by auto

  next

  case False

  with \(j\) have \(j : j = l\) by auto

  show \(?\)thesis

  proof (cases \(\exists j'. j' < l \land 0 < n j'\))

  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 \(?\)thesis

  by (rule exI[of - 0], simp add: \(l j\))

  next

  case True

  then obtain \(\text{lil} where \\text{lil}: \text{lil} < l \text{ and } n\text{lil}: 0 < n \text{lil}\) by auto

  then obtain \(\text{ll} where \ \text{ll} : \text{ll} = \text{Suc } \text{ll}\) (cases \(l, \text{auto}\))

  from \(\text{lil}\) have \(\text{ll} : \text{ll} = l - (\text{ll} - \text{lil})\) by auto

  let \(?l' = \text{LEAST } d. 0 < n (l - d)\)

  have \(\text{nlll}: 0 < n (l - ?l')\)

  proof (rule LeastI)

  show \(0 < n (l - (l - \text{lil}))\) using \(\text{lil}\) \(n\text{lil}\) by auto

  qed

  with Suc[THEN spec[of - l - ?l']] obtain \(k\) where \(k:\)

  \(\text{inf-concat } n k = (l - ?l', 0)\) unfolding \(l\) by auto

  from \(?l'\) obtain \(\text{off}\) where \(\text{off}: \text{Suc } (0 + \text{off}) = n (l - ?l')\) by (cases \(n (l - ?l'), \text{auto}\))

  from \(\text{inf-concat-step}[\text{OF } k, \text{OF } \text{off}]\)

  have \(\text{id}: \text{inf-concat } n (k + \text{Suc } \text{off}) = (\text{LEAST } i'. l - ?l' < i' \land 0 < n i', 0)\) (is - = (?l, 0)).

  have \(\text{ll}: \text{ll} = l\) unfolding \(l\)

  proof (rule Least-equality)

  show \(l - ?l' < \text{Suc } ll \land 0 < n (\text{Suc } ll)\) using nj[unfolded \(j\) \(l\)] by simp

  next

  fix \(l'\)

  assume ass: \(l l - ?l' < l' \land 0 < n l'\)

  show \(\text{Suc } ll \leq l'\)

  proof (rule ccontr)

    assume not: \(\neg \?\)thesis

    show \(l - ?l' < ll \land 0 < n ll\)

    proof (rule LeastI)

      show \(\text{Suc } ll \leq ll\)

      (cases \(l - ?l'\), auto)

      from \(\text{inf-concat-step}[\text{OF } k, \text{OF } \text{off}]\)

      have \(\text{id}: \text{inf-concat } n (k + \text{Suc } \text{off}) = (\text{LEAST } i'. l - ?l' < i' \land 0 < n i', 0)\) (is - = (?l, 0)).

      show \(\text{ill}: \text{ill} = ll\) unfolding \(ll\)

      proof (rule Least-equality)

        show \(l - ?l' < \text{Suc } ll \land 0 < n (\text{Suc } ll)\) using nj[unfolded \(j l\)] by simp

      next

      fix \(l'\)

      assume ass: \(l l - ?l' < l' \land 0 < n l'\)

      show \(\text{Suc } ll \leq l'\)

      proof (rule ccontr)

        assume not: \(\neg \?\)thesis
hence $l' \leq ll$ by auto
hence $ll = l' + (ll - l')$ by auto
then obtain $k$ where $ll = l' + k$ by auto
from ass have $l' + k - ?ll < l'$ unfolding $ll$ by auto
hence $kl'$: $k < ?ll$ by auto
have $0 < n$ $(ll - k)$ using ass unfolding $ll$ by simp
from Least-le[of $\lambda k. \; 0 < n$ $(ll - k)$, OF this] $kl'$
show False by auto
qed
qed
show $?thesis$ unfolding $j$
by (rule exI[of - $k + Suc off$], unfold id $ll$, simp)
qed
qed
qed
qed
qed
)
with assms show $?thesis$ by auto
qed

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

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

11
obtain $i''$ $j''$ where $l$: inf-concat $n$ $(k' + l) = (i'', j'')$ by force
with Suc have one: $i' \leq i''$ by auto
from bounds[of $l$] have $j'': j'' < n$ $i''$ by auto
show ?case
proof (cases Suc $j'' < n$ $i''$
  case True
  show ?thesis by (simp add: $l$ True one)
next
  case False
  let $i = \text{LEAST} i'. i'' < i' \land 0 < n$ $i'$
  from inf[of unfolded INFM-nat] obtain $k$ where $i'' < k \land 0 < n$ $k$ by auto
  from LeastI[of $\lambda k. i'' < k \land 0 < n$ $k$, OF this]
  have $i'' < ?i$ by auto
  with one show ?thesis by (simp add: $l$ False)
qed
qed with resk $k$ lt have False by auto
}
thus ?thesis by arith
qed

lemma inf-concat-Suc:
assumes inf: INFM $i$. $n$ $i > 0$
  and $f$: $\lambda i. f$ ($n$ $i$) = $f$ (Suc $i$) $0$
  and resk: inf-concat $n$ $k$ = ($i$, $j$)
  and ressk: inf-concat $n$ (Suc $k$) = ($i'$, $j'$)
shows $f$ $i'$ $j'$ = $f$ ($n$ $i$)
proof (cases Suc $j < n$ $i$)
  case True
  with ressk resk
  show ?thesis by simp
next
  case False
  let $p = \lambda i'. i < i' \land 0 < n$ $i'$
  let $i' = \text{LEAST} i'. ?p i'$
  from False $j$ have id: Suc ($j + 0$) = $n$ $i$ by auto
  from inf-concat-step[of resk, OF id] ressk:
  have $i'$. $i' = ?i'$ and $j'$. $j' = 0$ by auto
  from id have $j$: Suc $j$ = $n$ $i$ by simp
  from inf[of unfolded INFM-nat] obtain $k$ where $\exists p$ $k$ by auto
  from LeastI[of $\exists p$, OF this] have $?p$ $?i'$.
  hence $i' = \text{Suc} i + (?i' - \text{Suc} i)$ by simp
  then obtain $d$ where $ii'$: $i' = \text{Suc} i + d$ by auto
  from not-less-Least[of $\exists p$, unfolded $ii'$] have $d'$: $\exists d'. d' < d \implies n$ (Suc $i$ + $d'$) = $0$ by auto
have \( f \, (Suc \, i) \, 0 = f \, ?i\, 0 \) unfolding \( ii' \) using \( d' \)
proof (induct \( d \))
  case 0
  show \( \text{thesis} \) by simp
next
  case (Suc \( d \))
  hence \( f \, (Suc \, i) \, 0 = f \, (Suc \, (Suc \, i + d)) \, 0 \) by auto
  also have \( ... = f \, (Suc \, (Suc \, i + d)) \, 0 \)
    unfolding \( f[\text{symmetric}] \)
    using Suc(2)[of \( d \)] by simp
  finally show \( \text{thesis} \) unfolding \( i \, j' \, j \) by simp
qed
qed

2 Abstract Rewrite Systems

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

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

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

lemma refl-inv-image:
  refl \( R \) \implies refl \, (inv-image \( R \, f \))
by (simp add: inv-image-def refl-on-def)

2.1 Definitions

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

definition join :: 'a rel \( \Rightarrow \) 'a \, \text{rel} \, (\, (\cdot)^+ [1000] \, 999 \, ) \, \text{where} \)
  \( A^+ = A^* \, O \, (A^{-1})^* \)

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

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

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

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

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

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

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

**definition** NF :: 'a rel \( \Rightarrow \) 'a set where  
\( NF A = \{ a. A '' \{ a \} = \{ \} \} \)

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

**notation** (ASCII)  
symcl \((\cdot\cdot\cdot) [1000] 999) and  
conversion \((\cdot\cdot\cdot) [1000] 999) and  
normalizability \((\cdot\cdot\cdot) [1000] 999) and

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

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

**lemma** no-step:  
assumes \( A '' \{ a \} = \{ \} \) 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:
  \( (a, b) \in A^\downarrow \implies \exists c. (a, c) \in A^* \land (b, c) \in A^* \) 
by (blast elim: joinE)

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

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

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

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

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

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

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

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

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

Later declarations are tried first for ‘proof’ and ‘rule,’ then have the “main” introduction / elimination rules for constants should be declared last.

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

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

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

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

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

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

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

lemma rtrancl-join-join:
assumes \((a, b) \in A^*\) and \((b, c) \in A^\downarrow\)
show \((a, c) \in A^\downarrow\)
proof
- from \(\langle b, c \rangle \in A^\downarrow\) obtain \(b'\) where \((b, b') \in A^*\) and \((b', c) \in (A^{-1})^*\)
  unfolding join-def by blast
with \(\langle a, b \rangle \in A^\downarrow\) have \(\langle a, b' \rangle \in A^*\) by simp
with \(\langle b', c \rangle \in (A^{-1})^*\) show ?thesis unfolding join-def by blast
qed

lemma join-rtrancl-join:
assumes \((a, b) \in A^\downarrow\) and \((c, b) \in A^*\)
show \((a, c) \in A^\downarrow\)
proof
- from \(\langle c, b \rangle \in A^*\) have \(\langle b', c \rangle \in (A^{-1})^*\) by simp
  unfolding join-def by best
from \(\langle a, b \rangle \in A^\downarrow\) obtain \(a'\) where \((a, a') \in A^*\) and \((a', b) \in (A^{-1})^*\)
  unfolding join-def by simp
with \(\langle a', c \rangle \in (A^{-1})^*\) show ?thesis unfolding join-def by blast
qed

lemma NF-I: \((\forall b. (a, b) \notin A) \implies a \in NF A\)
by (auto intro: no-step)
lemma NF-E: \(a \in NF A \iff ((a, b) \notin A \implies P) \implies P\)
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-rtrancl-step:
assumes \(a \in NF A\)
show \(\forall b. (a, b) \notin A^+\)
proof
- from assms have \(\forall b. (a, b) \notin A\) by auto
  show ?thesis
  proof (intro allI notI)
  17
fix \( b \) assume \((a, b) \in A^+\)
then show \( \text{False} \) by (induct) (auto simp: \( \forall b. (a, b) \notin A \))
qed
qed

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

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

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

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

lemma \( \text{normalizability-I'} \) [intro]: \((a, b) \in A^* = \Rightarrow (b, c) \in A\! = \Rightarrow (a, c) \in A\! \) by (auto simp add: normalizability-def)

lemma \( \text{normalizability-E} \) [elim]: \((a, b) \in A\! = \Rightarrow (\ (a, b) \in A^* = \Rightarrow b \in \text{NF} A = \Rightarrow P) = \Rightarrow P \) 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 \( \text{CR-on} :: \ 'a \ rel \Rightarrow 'a \ set \Rightarrow \text{bool} \) where
\( \text{CR-on} \ r A \leftrightarrow (\forall a \in A. \forall b c. (a, b) \in r^* \land (a, c) \in r^* \longrightarrow (b, c) \in \text{join} \ r) \)

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

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

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

Alternative definition of \( \text{SN} \).

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

definition \( \text{UNF-on} :: \ 'a \ rel \Rightarrow 'a \ set \Rightarrow \text{bool} \) where
\( \text{UNF-on} \ r \ A \leftrightarrow (\forall a \in A. \forall b c. (a, b) \in r^1 \land (a, c) \in r^1 \longrightarrow b = c) \)
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 c. (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
\[ r A \]
by (simp add: CR-defs)

**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: 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:
\[
\text{semi-complete-on} r A \implies (\text{WN-on} r A \implies \text{CR-on} r A) \implies P
\]
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 \, UNF-on-def by blast

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

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

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

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

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

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

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

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

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

22
lemma WN-onD: WN-on r A ⟹ a ∈ A ⟹ ∃ b. (a, b) ∈ 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 ⇒ 'a rel ⇒ 'a rel where
restrict-SN r s = {(a, b) | a b. (a, b) ∈ r ∧ 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 ∈ r '' A and chain: chain r f
then obtain a where a ∈ A and 1: (a, f 0) ∈ r by auto
let ?g = case-nat a f
from cons-chain [OF 1 chain] have chain r ?g .
moreover have ?g 0 ∈ A by (simp add: ⟨a ∈ 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 ⊆ 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) ∈ 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) ∈ A* ⟹ SN-on A {a} ⟹ 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 \(\exists f. f \, 0 = x \land f \, n = y \land (\forall i < n. (f \, i, f \, (Succ \, i)) \in r)\)
using assms
proof (induct \(n\) arbitrary: \(y\))
case \(0\) then show \(?case\) by auto
next
case \((Succ \, n)\)
then obtain \(z\) where \((x, z) \in r^\ast n\) and \((z, y) \in r\) by auto
from \((Succ(1)) \, \{OF \, \langle(x, z) \in r^\ast n\rangle\}\)
obtain \(f\) where \(f \, 0 = x\) and \(f \, n = z\) and \(\forall i < n. (f \, i, f \, (Succ \, i)) \in r\)
by auto
let \(?n = Succ \, n\)
let \(?f = \lambda i. \text{if } i = ?n \text{ then } y \text{ else } f \, i\)
have \(?f \, ?n = y\) by simp
from \((f \, 0 = x)\) have \(?f \, 0 = x\) by simp
from \(seq\) have \(seq' \, : \forall i < n. (\forall f \, i, \, f \, (Succ \, i)) \in r\)
by auto
with \((f \, n = z)\) and \((z, y) \in r\) have \(\forall i < ?n. (f \, i, \, f \, (Succ \, i)) \in r\)
by auto
with \((f \, 0 = x)\) and \((f \, ?n = y)\) show \(?case\) by best
qed

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

lemma \(SN-on-Image-rtrancl\):
assumes \(SN-on \, r \, A\)
shows \(SN-on \, r \, (r^\ast \, \{\sim A\})\)
proof
fix \(f\)
assume \(f_0: f \, 0 \in r^\ast \, \{\sim A\} \land \, chain \, : \, chain \, r \, f\)
then obtain \(a\) where \(a : a \in A \land (a, f \, 0) \in r^\ast\)
by auto
then obtain \(n\) where \((a, f \, 0) \in r^\ast n\)
unfolding rtrancl-power by auto
show \((\sim False)\)
proof (cases \(n\))
case \(0\)
with \((a, f \, 0) \in r^\ast \) have \(f \, 0 = a\) by simp
then have \(f \, 0 \in A\) by (simp add: \(a\))
with \(chain\) have \((\sim SN-on \, r \, A)\)
by auto
with \(assms\) show \((\sim False)\) by simp
next
case \((Succ \, n)\)
from \(relpow-seq\) \([OF \, \langle(a, f \, 0) \in r^\ast \rangle]\)
obtain \(g\) where \(g_0 : g \, 0 = a \land g \, n = f \, 0\)
and \(gseq: \forall i < n. (g \, i, g \, (Succ \, i)) \in r\)
by auto
let \(if = \lambda i. \text{if } i < n \text{ then } g \, i \text{ else } f \, (i - n)\)
have \(chain \, r \, if\)
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 \( \text{Suc} \ i > n \) have \( (?f \ i, ?f (Suc \ i)) \in r \) by simp 
}\}

moreover 
\{ 
  assume Suc \ i = n 
  then have eq: \( f (\text{Suc} \ i - n) = g \ n \) by (simp add: \langle g \ n = f \ 0 \rangle) 
  from \( \text{Suc} \ i = n \) have eq': \( i = n - 1 \) by arith 
  from gseq have \( g \ i, f (\text{Suc} \ i - n) \) \in r unfolding eq by (simp add: Suc eq') 
  then have \( (?f \ i, ?f (Suc \ i)) \in r \) using \( \text{Suc} \ i = n \) by simp 
}\}

ultimately show 
\( (?f \ i, ?f (Suc \ i)) \in r \) by simp 
\[ \text{qed} \]

moreover have \( ?f \ 0 \in A \) 
proof (cases \( n \)) 
  case 0 
  with \( \langle a, f \ 0 \rangle \in r^-^n \) have eq: \( a = f \ 0 \) by simp 
  from a show \( ?\text{thesis} \) by (simp add: eq 0) 
  next 
  case (Suc \ m) 
  then show \( ?\text{thesis} \) by (simp add: a g0) 
\[ \text{qed} \]

ultimately have \( \neg \text{SN-on} r A \) unfolding SN-defs by best 
with \( \text{assms} \) show \( \text{False} \) by simp 
\[ \text{qed} \]

\[ \text{qed} \]

declare subrelI [Pure.intro]

**lemma** restrict-SN-trancl-simp [simp]: \((\text{restrict-SN} \ A \ A)^+ = \text{restrict-SN} \ (A^+) \ A\) 
(is \( ?lhs = ?rhs \)) 
proof 
  show \( ?lhs \subseteq ?rhs \) 
  proof 
    fix \( a, b \) assume \( (a, b) \in ?lhs \) then show \( (a, b) \in ?rhs \) 
    unfolding restrict-SN-def by (induct rule: trancl.induct) auto 
  qed 
next
show \( \text{rhs} \subseteq \text{lhs} \)

proof

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

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

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

lemma \( \text{join-NF-imp-eq} \):
assumes \( (x, y) \in r^\downarrow \) and \( x \in \text{NF } r \) and \( y \in \text{NF } r \)
shows \( x \ = \ y \)
proof –
from \((x, y) \in r^\uparrow\) obtain \(z\) where \( (x, z) \in r^*\) and \((z, y) \in (r^{\rightarrow})^*\) unfolding 

\text{join-def by auto}

then have \((y, z) \in r^*\) unfolding \text{rtrancl-converse by simp}

from \(x \in NF r\) have \((x, z) \notin r^\uparrow\) using \text{NF-no-trancl-step by best}

then have \(x = z\) using \text{rtranclD [OF \((x, z) \in r^*\)] by auto}

from \(y \in NF r\) have \((y, z) \notin r^\uparrow\) using \text{NF-no-trancl-step by best}

then have \(y = z\) using \text{rtranclD [OF \((y, z) \in r^*\)] by auto}

with \(x = z\) show \#thesis by simp

qed

\text{lemma rtrancl-Restr:}

\text{assumes} \((x, y) \in (\text{Restr } r A)^*\)

\text{shows} \((x, y) \in r^*

\text{using assms by induct auto}

\text{lemma join-mono:}

\text{assumes} r \subseteq s

\text{shows} r^\downarrow \subseteq s^\downarrow

\text{using rtrancl-mono [OF assms] by (auto simp: join-def rtrancl-converse)}

\text{lemma CR-iff-meet-subset-join:} CR r = (r^\uparrow \subseteq r^\downarrow)

\text{proof}

\text{assume} CR r \text{ show} r^\uparrow \subseteq r^\downarrow

\text{proof \text{(rule subrelI)}}

\text{fix} x y \text{ assume} (x, y) \in r^\uparrow

\text{then obtain} z \text{ where} \((z, x) \in r^*\) and \((z, y) \in r^*\) using \text{meetD by best}

with \(\text{CR r}\) show \((x, y) \in r^\downarrow\) by (auto simp: CR-defs)

qed

next

\text{assume} r^\uparrow \subseteq r^\downarrow \{ 

\text{fix} x y z \text{ assume} (x, y) \in r^* \text{ and} \((x, z) \in r^*\)

\text{then have} \((y, z) \in r^\downarrow\) unfolding \text{meet-def rtrancl-converse by auto}

with \(r^\uparrow \subseteq r^\downarrow\) have \((y, z) \in r^\downarrow\) by auto

\} \text{ then show} CR r \text{ by (auto simp: CR-defs)}

qed

\text{lemma CR-divergence-imp-join:}

\text{assumes} CR r \text{ and} \((x, y) \in r^*\) and \((x, z) \in r^*\)

\text{shows} \((y, z) \in r^\downarrow\) using \text{assms by auto}

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

\text{proof}

\text{fix} x z \text{ assume} \((x, z) \in r^\downarrow\)

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

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

\text{from} \((z, y) \in r^{***}\) have \((y, z) \in r^{***}\) using \text{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: \(r^\uparrow \subseteq r^{***}\)

proof (rule subrelI)
  fix \(y\) \(z\) assume \((y, z) \in r^\uparrow\)
  then obtain \(x\) where \((x, y) \in r^*\) and \((x, z) \in r^*\) by auto
  then have \((x, y) \in r^{***}\) 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 \((x, y) \in r\) and \((x, z) \in r\)
  then have \((x, y) \in r^*\) and \(y \in NF r\) and \((x, z) \in r^*\) and \(z \in NF r\)
    unfolding normalizability-def by auto
  from assms and \((x, y) \in r^*\) and \((x, z) \in r^*\) have \((y, z) \in r^\downarrow\)
    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)
} then show ?thesis by auto

qed

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

proof (intro iffI subrelI)
  fix \(x\) \(y\) assume \(CR r\) and \((x, y) \in r^{***}\)
  then obtain \(n\) where \((x, y) \in r^{\downarrow\neg\neg n}\) unfolding conversion-def rtrancl-is-UN-relpow
    by auto
  then show \((x, y) \in r^\downarrow\)
    proof (induct \(n\) arbitrary: \(x\))
      case 0
      assume \((x, y) \in r^{\neg\neg 0}\) then have \(x = y\) by simp
      show ?case unfolding \(x = y\) by auto
    next
      case (Suc \(n\))
      from \((x, y) \in r^{\neg\neg \text{Suc} n}\) obtain \(z\) where \((x, z) \in r^{\neg\neg \text{Suc} n}\) and \((z, y) \in r^{\neg\neg \text{Suc} n}\)
      unfolding relpow-Suc-D2 by best
      with Suc have \((z, y) \in r^\downarrow\) by simp
      from \((x, z) \in r^{\neg\neg n}\) show ?case
        proof
          assume \((x, z) \in r\) with \((z, y) \in r^\downarrow\) show ?thesis by (auto intro: rtrancl-join-join)
        next
          assume \((x, z) \in r^{-1}\)

28
then have \((z, x) \in r^*\) by simp
from \((z, y) \in r^+\) obtain \(z'\) where \((z, z') \in r^*\) and \((y, z') \in r^+\) by auto
from \((CR r)\) and \((z, x) \in r^+\) and \((z, z') \in r^*\) have \((x, z') \in r^+\) by (rule CR-divergence-imp-join)

then obtain \(x'\) where \((x, x') \in r^+\) and \((z', x') \in r^*\) by auto
with \((y, z') \in r^+\) show ?thesis by auto
qed

next

proof

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

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

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

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

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

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

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

lemma semi-complete-imp-conversionIff-same-NF:
assumes semi-complete $r$
shows $\langle (x, y) \in r^{***} \rangle = (\forall u v. (x, u) \in r^1 \land (y, v) \in r^1 \rightarrow u = v)$
proof –
from assms have $\text{WN } r$ and $\text{CR } r$ unfolding semi-complete-defs by auto
then have $r^{***} = r^1$ using CR-imp-conversionIff-join by auto
show $\theta\text{thesis}$
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 allI implE, elim conjE)
fix $u v$ assume $(x, u) \in r^1$ and $(y, v) \in r^1$
then have $(x, u) \in r^* \land (y, v) \in r^*$ and $u \in \text{NF } r$ and $v \in \text{NF } r$ by auto
from $(\text{CR } r)$ and $(x, u) \in r^*$ and $(x, y) \in r^1; \text{ have } (u, y) \in r^1$
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 $(\text{CR } r)$ and $(x, y) \in r^1; \text{ and } (y, v) \in r^*$; 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 $(u, v') \in r^*; \text{ and } (u \in \text{NF } r) \text{ have } u = v'$ by (rule NF-not-suc)
from $(v, v') \in r^*$ and $(v \in \text{NF } r) \text{ have } v = v'$ 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 $(\text{WN } r)$ obtain $u$ where $(x, u) \in r^1$ by auto
from $(\text{WN } r)$ 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 $(y, v) \in r^{***}; \text{ have } (v, y) \in r^{***}$ 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

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^1 by (rule CR-imp-conversionIff-join [OF assms]) 
  from (x, y) ∈ r^{+++} have (x, y) ∈ r^1 unfolding ⟨r^{+++} = r^1⟩ by simp 
  then obtain x' where ⟨x, x'⟩ ∈ r^* and ⟨y, x'⟩ ∈ r^* by best 
  from ⟨x, x'⟩ ∈ r^* and (x ∈ NF r) have x = x' by (rule NF-not-suc) 
  from ⟨y, x'⟩ ∈ r^* and (y ∈ NF r) have y = x' by (rule NF-not-suc) 
  then have x = y unfolding ⟨x = x'⟩ by simp 
} then show ⟨thesis by (auto simp: UNF-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^1 unfolding WN-defs by best 
  with ⟨x, y⟩ ∈ r^1 have ⟨x, y'⟩ ∈ r^1 by auto 
  from assms obtain z' where ⟨z, z'⟩ ∈ r^1 unfolding WN-defs by best 
  with ⟨x, z⟩ ∈ r^1 have ⟨x, z'⟩ ∈ r^1 by auto 
  with ⟨x, y⟩ ∈ r^1 have y' = z' using ⟨UNF r⟩ unfolding UNF-defs by auto 
  from ⟨y, y'⟩ ∈ r^1 and ⟨z, z'⟩ ∈ r^1 have ⟨y, z⟩ ∈ r^1 unfolding ⟨y' = z'⟩ by auto 
} then show ⟨thesis by auto qed

definition diamond :: {a rel ⇒ bool} where
  oval r ≡ (r^{-1} O r) ⊆ (r O r^{-1})

lemma diamond-I [intro]: ⟨r^{-1} O r⟩ ⊆ ⟨r O r^{-1}⟩  ⟹ oval r unfolding diamond-def by simp

lemma diamond-E [elim]: oval r  ⟹ ⟨(r^{-1} O r) ⊆ (r O r^{-1}) ⟹ P ⟹ P⟩ unfolding diamond-def by simp

lemma diamond-imp-semi-confluence:
  assumes oval r shows (r^{-1} O r^*) ⊆ r^1
proof (rule subrelI) 
  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^1 
  proof (induct n arbitrary: x z y) 
    case 0 then show ⟨case by auto 
  next 
    case (Suc n)

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

lemma diamond-imp-CR':
  assumes \(\Diamond \circ s \text{ and } r \subseteq s \text{ and } s \subseteq r^* \text{ shows } \text{CR } r\)
  unfolding \(\text{CR-iff-meet-subset-join}\)
  proof
  \[
  \text{from } (\Diamond \circ s \text{ have } \text{CR } s \text{ by (rule diamond-imp-CR)} \\
  \text{then have } s^* \subseteq s^4 \text{ unfolding } \text{CR-iff-meet-subset-join by simp} \\
  \text{from } \text{r} \subseteq s \text{ have } r^* \subseteq s^* \text{ by (rule rtrancl-mono)} \\
  \text{from } s \subseteq \text{r}^* \text{ have } s^* \subseteq \text{(r}^*)^* \text{ by (rule rtrancl-mono)} \\
  \text{then have } s^* \subseteq \text{r}^* \text{ by simp} \\
  \text{with } \text{r}^* \subseteq s^* \text{ have } r^* = s^* \text{ by simp} \\
  \text{show } r^* \subseteq s^* \text{ unfolding meet-def join-def rtrancl-converse } (r^* = s^*) \\
  \text{unfolding rtrancl-converse [symmetric] meet-def [symmetric]}
  \]
proof \(\vdash\) (rule \(s^1 \subseteq s^1\))

qed

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

qed

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

lemma \(\text{minimal-imp-uf} \): \[\begin{align*}
\text{assumes } & \forall Q. x. x \in Q \rightarrow (\exists z \in Q. \forall y. (z, y) \in r \rightarrow y \notin Q) \\
\text{shows } & uf(r^{-1}) \\
\text{proof (rule ccontr)} & \\
\end{align*} \]

33
assume \( \neg \text{wf}(r^{-1}) \)
then have \( \exists P. (\forall y. (x, y) \in r \rightarrow P y) \rightarrow P x \) \& \( \exists x. \neg P x \) unfolding \text{wf-def} by simp
then obtain \( P x \) where suc:\( \forall x. (\forall y. (x, y) \in r \rightarrow P y) \rightarrow P x \) \& \( \neg P x \)
by auto
let \( ?Q = \{ x. \neg P x \} \)
from \( \langle \neg P x \rangle \) have \( x \in ?Q \) by simp
from \text{assms} have \( \forall y. (z, y) \in r \rightarrow y \notin ?Q \) by (rule allE [where \( x = ?Q \)])
with \( (z, y) \in r \) obtain \( z \) where \( z \in ?Q \) and \( \text{min show False} \) by simp
qed
lemmas \text{SN-imp-wf} = \text{SN-imp-minimal} [THEN \text{minimal-imp-wf}]

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

34
lemma SN-iff-wf: SN A = wf (A^{-1}) by (auto simp: SN-imp-wf wf-imp-SN)

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

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

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

lemma SN-on-induct [consumes 2, case-names IH, induct pred: SN-on]:
    assumes SN: SN-on R A  
    and s \in A  
    and imp: \\( \forall t. (\\forall u. (t, u) \in R \implies P u) \implies P t \)  
    shows P s
    proof
        let ?R = restrict-SN R R
        let ?P = \\( \lambda t. (\\forall R \{ t \} \implies P t) \)  
        have SN-on R \{ s \} \implies P s
        proof (rule SN-induct [OF SN-restrict-SN-idemp [of R], of ?P])
            fix a
            have ind: \\( \forall b. (a, b) \in ?R \implies SN-on R \{ b \} \implies P b \)  
                using SN step-preserves-SN-on [of this SN]
            proof
                fix b
                have (a, b) \in R
                    with SN step-preserves-SN-on [OF this SN]
                    show P b using ind [of b] unfolding restrict-SN-def by auto
            qed
        qed
        qed
        with SN show P s using (s \in A) unfolding SN-on-def by blast
        qed

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

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

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

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

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

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

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

using assms unfolding SN-on-conv-acc by simp

2.3 Newman’s Lemma

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

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

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

lemma Newman-local:
assumes SN-on r X and WCR: WCR-on r \{ x. SN-on r \{ x \} \}
shows CR-on r X
proof -
fix x
assume x \in X
with assms have SN-on r \{ x \} unfolding SN-on-def by auto
with this have CR-on r \{ x \}
proof (induct rule: SN-on-induct')
  case (IH x) show ?case
  proof
    fix y z assume (x, y) \in r^* and (x, z) \in r^*
    from (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 (simp add: 0)
      from (x, y) \in r^* show ?thesis unfolding eq ..
      next
      case (Suc n')
      from (x, z) \in r^{-n} [unfolded Suc] obtain t where (x, t) \in r and (t, z) \in r^+
      show ?thesis
      proof (cases m)
        case 0
        from (x, y) \in r^{-m} have eq: x = y by (simp add: 0)
from \((x, z) \in r^*\) show ?thesis unfolding eq ..
next
case (Suc m')
  from \(\langle x, y \rangle \in r^{\ast\ast m'}\) [unfolded Suc] obtain s where \((x, s) \in r\) and \((s, y) \in r^*\) ..
  from \(\langle x, x \rangle \in r\) \(\langle x, t \rangle \in r\) have \((s, t) \in r^4\) by auto
  then obtain u where \((s, u) \in r^*\) and \((t, u) \in r^*\) ..
  from \(\langle x, s \rangle \in r\) \(\langle x, t \rangle \in r\) have \(\langle s, y \rangle \in r^*\) by (rule step-preserves-SN-on)
  from \(IH(1)\) OF \(\langle x, s \rangle \in r\) \(\langle x, t \rangle \in r\) have \(\langle s, y \rangle \in r^*\) by auto
  then obtain v where \((u, v) \in r^*\) and \((y, v) \in r^*\) ..
  from \(\langle x, t \rangle \in r\) \(\langle x, t \rangle \in r\) have \(\langle t, v \rangle \in r^*\) by (rule step-preserves-SN-on)
  from \(IH(1)\) OF \(\langle x, t \rangle \in r\) \(\langle x, t \rangle \in r\) have \(\langle t, v \rangle \in r^*\) by auto
  ultimately have \((z, v) \in r^4\) using \((t, z) \in r^*\) by auto
  then obtain w where \((z, w) \in r^*\) and \((v, w) \in r^*\) ..
  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
qed
qed
qed

then show ?thesis unfolding CR-on-def by blast
qed

lemma Newman: \(SN r \Longrightarrow WCR r \Longrightarrow CR r\)
  using Newman-local [of r UNIV]
  unfolding WCR-on-def by auto

lemma Image-SN-on:
  assumes \(SN-on r (r \circ A)\)
  shows \(SN-on r A\)
proof
  fix f
  assume \(f \circ 0 \in A\) and chain: chain r f
  then have \(f (Suc 0) \in r \circ A\) by auto
  with assms have \(SN-on r \{ f (Suc 0) \}\) by (auto simp add: \(f \circ 0 \in A\) SN-defs)
  moreover have \(-SN-on r \{ f (Suc 0) \}\)
  proof
    have \(f (Suc 0) \in \{ f (Suc 0) \}\) by simp
    moreover from chain have \(chain r (f \circ Suc)\) by auto
    ultimately show ?thesis by auto
    qed
    ultimately show \(False\) by simp
    qed

lemma SN-on-Image-conv: \(SN-on r (r \circ A) = SN-on r A\)

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

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

Assumes: \( \forall b. (a, b) \in r \implies SN-on r \{b\} \)

Shows: \( SN-on r \{a\} \)

Using assms and Image-SN-on [of r \{a\}] by (auto simp: SN-defs)

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

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

Using SN-on-Image-conv [of r \{a\}] by (auto simp: SN-defs)

**Lemma SN-inv-image:**

\( SN R = \implies SN (inv-image R f) \)

Unfolding SN-iff-wf by simp

**Lemma SN-subset:**

\( SN R = \implies R' \subseteq R \implies SN R' \)

Unfolding SN-defs by blast

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

\( R^{\ast\ast}(Suc n) \subseteq R^+ \)

Using trancl-power [of - R] by blast
**Lemma** SN-imp-SN-pow:
- **Assumes** SN R shows SN (R ^^ Suc n)
- **By** simp

**Lemma** SN-pow: SN R ⊢→ SN (R ^^ Suc n)
- by (rule iffI, rule SN-imp-SN-pow, assumption, rule SN-pow-imp-SN, assumption)

**Lemma** SN-on-trancl:
- **Assumes** SN-on r A shows SN-on (r^+) A
- **Using** assms
- **Proof** (rule contrapos-pp)
  - let ?r = restrict-SN r r
  - assume ¬ SN-on (r^+) A
  - then obtain f where f 0 ∈ 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 ∀ i. (f 0, f i) ∈ r^+
  - proof
    - fix i show (f 0, f i) ∈ r^+
    - proof (induct i)
      - case 0 show ?case ..
    - next
      - case (Suc i)
        - from chain have (f i, f (Suc i)) ∈ r^+ ..
        - with Suc show ?case by auto
  - qed
  - with assms have ∀ i. SN-on r {f i}
    - using steps-preserve-SN-on [of f 0 - r]
    - and (f 0 ∈ 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 ¬ SN-on (?r^+) {f 0} by auto
    - with ?SN (?r^+): have False by (simp add: SN-defs)
    - then show ¬ SN-on r A by simp
  - qed

**Lemma** SN-on-trancl-SN-on-conv: 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 ⇒ 'a set ⇒ 'a rel where
- restrict r S = \{(x, y). x ∈ S ∧ y ∈ S ∧ (x, y) ∈ r\}
lemma SN-on-restrict:
assumes SN-on r A
shows SN-on (restrict r S) A (is SN-on ?r A)
proof (rule ccontr)
  assume ¬ SN-on ?r A
  then have ∃ f. f 0 ∈ A ∧ chain ?r f by auto
  then have ∃ f. f 0 ∈ A ∧ chain r f unfolding restrict-def by auto
  with ⟨SN-on r A⟩ show False by auto
qed

lemma restrict-rtrancl: (restrict r S)* ⊆ r* (is ?r* ⊆ r*)
proof − {
  fix x y assume (x, y) ∈ ?r* then have (x, y) ∈ r* unfolding restrict-def by induct auto
} then show ?thesis by auto
qed

lemma rtrancl-Image-step:
assumes a ∈ r* " A
  and (a, b) ∈ r*
shows b ∈ r* " A
proof −
  from assms(1) obtain c where c ∈ A and (c, a) ∈ r* by auto
  with assms have (c, b) ∈ r* by auto
  with ⟨c ∈ A⟩ show ?thesis by auto
qed

lemma WCR-SN-on-imp-CR-on:
assumes WCR r and SN-on r A shows CR-on r A
proof −
  let ?S = r* " A
  let ?r = restrict r ?S
  have ∀ x. SN-on ?r {x}
  proof
    fix y have y ∉ ?S ∨ y ∈ ?S by simp
    then show SN-on ?r {y}
    proof
      assume y ∉ ?S then show ?thesis unfolding restrict-def by auto
    next
      assume y ∈ ?S
      then have y ∈ r* " A by simp
      with SN-on-Image-rtrancl [OF SN-on r A]
      have SN-on r {y} using SN-on-subset2 [of {y} r* " A] by blast
      then show ?thesis by (rule SN-on-restrict)
    qed
  qed
  then have SN ?r unfolding SN-defs by auto
  { fix x y assume (x, y) ∈ r* and x ∈ ?S and y ∈ ?S
then obtain $n$ where $(x, y) \in r^{-n}$ and $x \in ?S$ and $y \in ?S$
using rtrancl_imp_UN-relpow by best
then have $(x, y) \in ?r^*$
proof (induct $n$ arbitrary: $x$ $y$)
  case 0 then show ?case by simp
next
case $(\text{Suc } n)$
from $(x, y) \in r^{-\text{Suc } n}$ obtain $x'$ where $(x, x') \in r$ and $(x', y) \in r^{-n}$
using relpow-Suc-D2 by best
then have $(x, x') \in r^*$ by simp
with $(x \in ?S)$ have $x' \in ?S$ by (rule rtrancl-Image-step)
with Suc and $(x', y) \in r^{-n}$ have $(x', y) \in ?r^*$ by simp
from $(x, x') \in r^*$ and $(x \in ?S)$ and $(x' \in ?S)$ have $(x, x') \in ?r$
unfolding restrict-def by simp
with $(x', y) \in ?r^*$ show ?case by simp
qed

} then have $\forall x \cdot y \cdot (x, y) \in r^* \land x \in ?S \land y \in ?S \then (x, y) \in ?r^*$ by simp
{ fix $x' \cdot y \cdot z$ assume $(x', y, z) \in ?r$ and $(x', z) \in ?r$
then have $x' \in ?S$ and $y \in ?S$ and $z \in ?S$ and $(x', y) \in r$ and $(x', z) \in r$
unfolding restrict-def by auto
with $(WCR \cdot y)$ have $(y, z) \in r^+$ by auto
then obtain $u$ where $(y, u) \in r^*$ and $(z, u) \in r^*$ by auto
from $(x' \in ?S)$ obtain $x$ where $x \in A$ and $(x, x') \in r^*$ by auto
from $(x', y) \in r^*$ have $(x', y) \in r^*$ by auto
with $(\langle y, u \rangle \in r^*)$ have $(x', u) \in r^*$ by auto
with $(\langle x, x' \rangle \in r^*)$ have $(x, u) \in r^*$ by simp
then have $u \in ?S$ using $(x \in A)$ by auto
from $(y \in ?S)$ and $(u \in ?S)$ and $(\langle y, u \rangle \in r^*)$ have $(y, u) \in ?r^*$ using $a$ by auto
from $(z \in ?S)$ and $(u \in ?S)$ and $(\langle z, u \rangle \in r^*)$ have $(z, u) \in ?r^*$ using $a$ by auto
with $(\langle y, u \rangle \in ?r^*)$ have $(y, z) \in ?r^+$ by auto
} then have $WCR \cdot ?r$ by auto
have $\text{CR } ?r$ using Newman [OF $\langle SN \cdot ?r \rangle \cdot (WCR \cdot ?r)$] by simp
{ fix $x \cdot y \cdot z$ assume $x \in A$ and $(x, y) \in r^*$ and $(x, z) \in r^*$
then have $y \in ?S$ and $z \in ?S$ by auto
have $x \in ?S$ using $(x \in A)$ by auto
from $a$ and $(\langle x, y \rangle \in r^*)$ and $(x \in ?S)$ and $(y \in ?S)$ have $(x, y) \in ?r^*$ by simp
from $a$ and $(\langle x, z \rangle \in r^*)$ and $(x \in ?S)$ and $(z \in ?S)$ have $(x, z) \in ?r^*$ by simp
with $(\text{CR } ?r)$ and $(\langle x, y \rangle \in ?r^*)$ have $(y, z) \in ?r^+$ by auto
then obtain $u$ where $(y, u) \in ?r^*$ and $(z, u) \in ?r^*$ by best
then have $(y, u) \in r^*$ and $(z, u) \in r^*$ using restrict-rtrancl by auto
then have $(y, z) \in r^+$ by auto

42
lemma SN-on-Image-normalizable:
assumes SN-on r A
shows ∀ a∈A. ∃ b. b ∈ r' " A
proof
  fix a assume a; a ∈ A
  show ∃ b. b ∈ r' " A
  proof
    (rule ccontr)
    assume ¬ (∃ b. b ∈ r' " A)
    then have A: ∀ b. (a, b) ∈ r* → b ∉ NF r using a by auto
    then have a ∉ NF r by auto
    let ?Q = { c. (a, c) ∈ r* ∧ c ∉ NF r }
    have a ∈ ?Q using (a ∉ NF r) by simp
    have ∀ c∈?Q. ∃ b. (c, b) ∈ r ∧ b ∈ ?Q
    proof
      fix c
      assume c ∈ ?Q
      then have (a, c) ∈ r* and c ∉ NF r by auto
      then obtain d where (c, d) ∈ r by auto
      with (a, c) ∈ r* have (a, d) ∈ r* by simp
      with A have d ∉ NF r by simp
      with (c, d) ∈ r* and (a, c) ∈ r*
      show ∃ b. (c, b) ∈ r ∧ b ∈ ?Q by auto
    qed
    with ⋄ a ∈ ?Q! have a ∈ ?Q ∧ (∀ c∈?Q. ∃ b. (c, b) ∈ r ∧ b ∈ ?Q) by auto
    then have ∃ Q, a ∈ Q ∧ (∀ c∈Q. ∃ b. (c, b) ∈ r ∧ b ∈ Q) by (rule exI [of - ?Q])
    then have ¬ (∀ Q. a ∈ Q → (∃ c∈Q. ∃ b. (c, b) ∈ r → b ∉ Q) ) by simp
    with SN-on-imp-on-minimal [of r a] have ¬ SN-on r {a} by blast
    with assms and (a ∈ A; and SN-on-subset2 [of {a} A r] show False by simp
  qed
qed

lemma SN-on-imp-normalizability:
assumes SN-on r {a} shows ∃ b. (a, b) ∈ r'
using SN-on-Image-normalizable [OF assms] by auto

2.4 Commutation

definition commute :: 'a rel ⇒ 'a rel ⇒ bool where
  commute r s ←→ ((r⁻¹)* O s*) ⊆ (s* O (r⁻¹)*)

lemma CR-iff-self-commute: CR r = commute r r
  unfolding commute-def CR-iff-meet-subset-join meet-def join-def
  by simp
lemma rtrancl-imp-rtrancl-UN:
  assumes \((x, y) \in r^* \text{ and } r \in I\)
  shows \((x, y) \in (\bigcup_{r \in I} r)^* \text{ (is } (x, y) \in ?r^*)\)
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 ⇒ ‘a rel ⇒ bool where
  quasi-commute r s ⇐⇒ \((s O r) \subseteq r O (r \cup s)^*\)

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 \((y, z) \in r\) \(\lor\) \((y, z) \in s\) by auto
      then have \((y, z) \in r^* \cup s^*\) by auto
      then show ?thesis using rtrancl-Un-subset by auto
    qed
    with \((x, y) \in (r \cup s)^*\) 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 \( (x, y) \in (r \cup s)^* \), 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 \( (x, z) \in s^+ \circ r \)
then obtain \( y \) where \( (x, y) \in s^+ \circ r \)
\( y, z \in r \) by best
then show \( (x, z) \in r \circ (r \cup s)^* \)
proof (induct arbitrary: \( z \))
case \( \text{base } y \)
then have \( (x, z) \in (s \circ r) \) by auto
with assms have \( (x, z) \in r \circ (r \cup s)^* \)
unfolding quasi-commute-def by auto
then show \( ? \) case using rtrancl-union-subset-rtrancl-union-trancl by auto
next
case \( \text{step } a \ b \)
then have \( (a, z) \in (s \circ r) \) by auto
with assms have \( (a, z) \in r \circ (r \cup s)^* \)
unfolding quasi-commute-def by auto
then obtain \( u \) where \( (a, u) \in r \) and \( (u, z) \in (r \cup s)^* \) by best
then have \( (u, z) \in (r \cup s)^+ \) using rtrancl-union-subset-rtrancl-union-trancl by auto
from \( (a, u) \in r \) and \( \text{step } (x, u) \in r \circ (r \cup s)^* \) by auto
then obtain \( v \) where \( (x, v) \in r \) and \( (v, u) \in (r \cup s)^* \) by best
with \( (u, z) \in (r \cup s)^* \) have \( (v, z) \in (r \cup s)^* \) by auto
with \( (x, v) \in r \) show \( ? \) case by auto
qed
qed

lemma steps-reflect-SN-on:
assumes \( \neg \text{SN-on } r \{ b \} \) and \( (a, b) \in r^* \)
shows \( \neg \text{SN-on } r \{ a \} \)
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 \( \neg \text{SN-on } r \{ f \ i \} \)
proof
let \( f = \lambda j. f (i + j) \)

have \( \neg f \in \{ f \ i \} \) by simp
moreover have chain r \( ? f \) using assms by auto
ultimately have \( \neg f \in \{ f \ i \} \land \text{chain } r \ ? f \) by blast
then have \( \exists g. g 0 \in \{ f \ i \} \land \text{chain } r g \) 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 ∪ s)

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

46
proof using steps lemma comp-trancl qed

then obtain \( x \) where \( \exists i. S(Suc\ ?i) \notin x \) by (rule LeastI-ex)
with \( \langle S\ ?i, S(Suc\ ?i) \rangle \in x \) have False unfolding \( \langle \exists i. S(Suc\ ?i) \rangle \notin x \) by simp 

then have \( 0 < ?i \) by auto
then obtain \( j \) where \( ?i = Suc\ j \) unfolding gr0-conv-Suc by best
with \( \langle S\ ?i, S(Suc\ ?i) \rangle \in S \) have \( ?i = Suc\ j \) by auto

next

assume \( \langle x, u, z \rangle \)
from \( \exists i. S(Suc\ ?i) \notin x \) have False unfolding \( \langle \exists i. S(Suc\ ?i) \rangle \notin x \) by simp 

then show \(?thesis\) by auto

\qed

2.5 Strong Normalization

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

using steps proof

fix \( x u z \)
assume \( (s, t) = (x, z) \) and \( (x, u) \in NS^* \) and \( (u, z) \in S \)
then have \( (s, u) \in NS^* \) and \( (u, t) \in S \) by auto
then show \(?thesis\) 

proof (induct rule:trancl.induct)

case (trancl-refl \( x \)) then show \(?case\) .

next

case (trancl-into-trancl \( a \ b \ c \))

with compat show \(?case\) by auto

\qed

\qed

lemma comp-trancl:
assumes \( R\ O\ S \subseteq S \) shows \( R\ O\ S^+ \subseteq S^+ \)

proof (rule subrell)

fix \( w z \) assume \( (w, z) \in R\ O\ S^+ \)
then obtain \( x \) where \( R\text{-step}: (w, x) \in R \) and \( S\text{-seq}: (x, z) \in S^+ \) by best
from tranclD [OF S-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 \((w, y) \in R \circ S\) by auto
with assms have \((w, y) \in S\) by auto
with S-seq' show \((w, z) \in S^+\) by simp
qed

lemma comp-rtrancl-trancl:
assumes comp: \(R \circ S \subseteq S\)
and seq: \((s, t) \in (R \cup S)^* \circ S\)
shows \((s, t) \in S^+\)
using seq proof
fix x u z
assume \((s, t) = (x, z)\) and \((x, u) \in (R \cup S)^*\) and \((u, z) \in S\)
then have \((s, u) \in (R \cup S)^*\) and \((u, t) \in S^+\) by auto
then show \(?thesis\)
proof (induct rule: rtrancl.induct)
case (rtrancl-refl x) then show \(?case .\)
  next
case (rtrancl-into-rtrancl a b c)
then have \((b, c) \in R \cup S\) by simp
then show \(?case\)
  proof
    assume \((b, c) \in S\)
    with rtrancl-into-rtrancl
    have \((b, t) \in S^+\) by simp
    with rtrancl-into-rtrancl show \(?thesis\) by simp
  next
    assume \((b, c) \in R\)
    with comp-trancl [OF comp] rtrancl-into-rtrancl
    show \(?thesis\) by auto
  qed
  qed
qed

lemma trancl-union-right: \(r^+ \subseteq (s \cup r)^+\)
proof (rule subrelI)
fix x y assume \((x, y) \in r^+\) then show \((x, y) \in (s \cup r)^+\)
proof (induct)
case base then show \(?case\) by auto
next
case (step a b)
then have \((a, b) \in (s \cup r)^+\) by auto
with \((x, a) \in (s \cup r)^+\) show \(?case\) by auto
qed
qed

lemma restrict-SN-subset: restrict-SN R S \(\subseteq R\)
proof (rule subrelI)
fix a b assume \((a, b) \in \text{restrict-SN } R S\) then show \((a, b) \in R\) unfolding \text{restrict-SN-def} by simp

qed

**lemma** chain-Un-SN-on-imp-first-step:
assumes chain \((R \cup S) t\) and SN-on \(S\) \(\{t 0\}\)
shows \(\exists i. (t i, t (Suc i)) \in R \land (\forall j < i. (t j, t (Suc j)) \in S \land (t j, t (Suc j)) \notin R)\)
proof
from \(\langle \text{SN-on } S \{t 0\} \rangle\) obtain \(i\) where \((t i, t (Suc i)) \notin S\) by blast with assms have \((t i, t (Suc i)) \in R\) (is \(?P i) by auto
let \(?i = \text{Least } ?P\)
from \(\langle ?P i \rangle\) have \(?P ?i\) by (rule LeastI)
moreover with assms have \(\forall j < ?i. (t j, t (Suc j)) \in S\) by best ultimately have \(\forall j < ?i. (t j, t (Suc j)) \in S \land (t j, t (Suc j)) \notin R\) by best with \(\langle ?P ?i \rangle\) show ?thesis by best
qed

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

qed

**lemma** first-step-O:
assumes \(C: C = A \cup B\) and \(\text{steps}: (x, y) \in C^* O B\)
shows \(\exists y. (x, y) \in A^* O B\)
proof
from steps obtain \(z\) where \((x, z) \in C^*\) and \((z, y) \in B\) by auto
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 ?thesis by simp
next
  case False
  let ?SR = \( S \cup R \)
  from xyL and LSR have \( (x, y) \in ?SR^* \) by simp
  from this and False have \( (x, y) \in R^* O S O ?SR^* \)
proof (induct rule: rtrancl-induct)
  case base then show ?case by simp
next
  case (step y z)
  then show ?case by simp
proof (cases \( (x, y) \in R^* \))
  case False
  with step have \( (x, y) \in R^* O S O ?SR^* \) by simp
  from this obtain u where xu: \( (x, u) \in R^* O S \) and uy: \( (u, y) \in ?SR^* \)
  force
  from \( (y, z) \in ?SR \) have \( (y, z) \in ?SR^* \) by auto
  with uy have \( (u, z) \in ?SR^* \) by (rule rtrancl-trans)
  with xu show ?thesis by auto
next
  case True
  have \( (y, z) \in S \)
  proof (rule ccontr)
    assume \((y, z) \notin S \) with \((y, z) \in ?SR \) have \( (y, z) \in R \) by auto
    with True have \( (x, z) \in R^* \) by auto
    with \((x, z) \notin R^* \) show False ..
  qed
  with True show ?thesis by auto
  qed
  qed
  with LSR show ?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 \( \exists j. \forall i \geq j. (t \ i, t (Suc \ i)) \in R - S \)
proof (rule ccontr)
  assume \( \neg ?thesis \)
  with chain have \( \forall i. \exists j. \ j \geq i \land (t \ j, t (Suc \ j)) \in S \) by blast
  from choice [OF this] obtain f where \( S\)-steps: \( \forall i. \ i \leq f \ i \land (t \ f \ i), t (Suc \ f \)
\[
\begin{align*}
&i)) \in S. \\
\text{let } \ ?t = \lambda i. ( ((\text{Suc } f) ^\circ i) 0) \\
&\text{have } S\text{-chain: } \forall i. (t \circ (\text{Suc } (f i))) \in S^+ \\
&\text{proof} \\
&\quad \text{fix } i \\
&\quad \text{from } S\text{-steps have leg: } i \leq f i \text{ and step: } (t(f i), t((\text{Suc } (f i))) \in S \text{ by auto} \\
&\quad \text{from chain-imp-rtrancl [OF chain leg] have } (t i, t(f i)) \in (R \cup S)^* \text{ by auto} \\
&\quad \text{with step have } (t i, t((\text{Suc } (f i))) \in (R \cup S)^* O S \text{ by auto} \\
&\quad \text{from comp-rtrancl-trancl [OF comp this] show } (t i, t((\text{Suc } (f i))) \in S^+. \text{ qed} \\
&\text{then have } \text{chain } (S^+) \ ?t \text{ by simp} \\
&\text{moreover have } \text{SN-on } (S^+) \ (\forall 0) \text{ using } \text{SN-on-trancl [OF SN]} \text{ by simp} \\
&\text{ultimately show } False \text{ unfolding SN-defs by best} \text{ qed} \\
&\text{lemmas } \text{SN-on-mono } = \text{SN-on-subset1} \\
&\text{lemma } \text{rtrancl-fun-conv:} \\
&\quad ((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)) \\
&\text{unfolding } \text{rtrancl-is-UN-relpow using relpow-fun-conv [where } R = R] \text{ by auto} \\
&\text{lemma } \text{compat-tr-compat:} \\
&\quad \text{assumes } NS O S \subseteq S \text{ shows } NS^* O S \subseteq S \\
&\text{using non-strict-into-strict [where } S = S \text{ and } NS = NS] \text{ assms by blast} \\
&\text{lemma } \text{right-comp-S [simp]:} \\
&\quad \text{assumes } (x, y) \in S O (S O S^* O NS^* \cup NS^*) \\
&\text{shows } (x, y) \in (S O S^* O NS^*) \\
&\text{proof} \\
&\quad \text{from assms have } (x, y) \in (S O S O S^* O NS^*) \cup (S O NS^*) \text{ by auto} \\
&\quad \text{then have } xy(x, y) \in (S O (S O S^*) O NS^*) \cup (S O NS^*) \text{ by auto} \\
&\quad \text{have } S O S^* \subseteq S^* \text{ by auto} \\
&\quad \text{with } xy \text{ have } (x, y) \in (S O S^* O NS^*) \cup (S O NS^*) \text{ by auto} \\
&\quad \text{then show } (x, y) \in (S O S^* O NS^*) \text{ by auto} \text{ qed} \\
&\text{lemma } \text{compatible-SN:} \\
&\quad \text{assumes } SN: SN S \\
&\quad \text{and compat: } NS O S \subseteq S \\
&\text{shows } SN (S O S^* O NS^*) \ (is } SN \ ?{A} \text{) \\
&\text{proof} \\
&\quad \text{fix } F \text{ assume chain: chain } ?{A} F \\
&\quad \text{from compat compat-tr-compat have } tr-compat: NS^* O S \subseteq S \text{ by blast} \\
\end{align*}
\]
have }i. (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 }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 }f. (\forall i. (y 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 }f. }y 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 have }g where }f 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 }\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^+) f by auto
then obtain x where }f 0 = x \land chain (S^+) f by auto
then have }\exists f. f 0 = x \land chain (S^+) f by auto
then have }\neg SN-on (S^+) \{x\} by auto
then have }\neg SN (S^+) unfolding SN-defs by auto
then have wfSconv: }\neg wf ((S^+)^{-1}) using SN-iff-wf by auto
from SN have wf (S^{-1}) using SN-imp-wf [where ?r=S] by simp
with wf-converse-trancl wfSconv show False by auto
qed

lemma compatible-rtrancl-split:
  assumes compat: NS O S \subseteq S
  and steps: (x, y) \in (NS \cup S)^*
  shows (x, y) \in S O S^* O NS^* \cup NS^*
proof-
  from steps have }\exists n. (x, y) \in (NS \cup S)^-n using rtrancl-imp-relpow [where ?R=NS \cup S] 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 m)
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 [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 \(True\)
from compat compat-tr-compat have \(trCompat: NS^* O S \subseteq S\) by blast
from \(zy\) True have \((x, y) \in (NS O S O S^* O NS^*) \cup (NS O NS^*)\) by auto
then have \((x, y) \in ((NS O S) O S^* O NS^*) \cup (NS O NS^*)\) by auto
then have \((x, y) \in ((NS^* O S) O S^* O NS^*) \cup (NS O NS^*)\) by auto
with \(trCompat\) have \(xy: (x, y) \in (S O S^* O NS^*) \cup (NS O NS^*)\) by auto
have \(NS O NS^* \subseteq NS^*\) by auto
with \(xy\) show \((x, y) \in (S O S^* O NS^*) \cup NS^*\) by auto
next
case \(False\)
with \(xz\) have \(xz: (x, z) \in S\) by auto
with \(zy\) have \((x, y) \in S O (S O S^* O NS^* \cup NS^*)\) by auto
then show \((x, y) \in (S O S^* O NS^*) \cup NS^*\) using right-comp-S by simp
qed
qed

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

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

lemma compatible-NS):

53
assumes compat: NS ∩ S ⊆ S and SN: SN S
shows SN((NS ∪ S)* ∩ O ∩ (NS ∪ S))∗
using compatible-conv [where S = S and NS = NS]
compatible-SN [where S = S and NS = NS] assms by force

lemma rtrancl-diff-decomp:
assumes (x, y) ∈ A* − B*
shows (x, y) ∈ A* O (A − B) O A∗
proof –
from assms have A: (x, y) ∈ A* and B: (x, y) ∉ B* by auto
from A have ∃ k. (x, y) ∈ A ″ k by (rule rtrancl-imp-relpow)
then obtain k where Ak: (x, y) ∈ A ″ k by auto
from Ak B show (x, y) ∈ A* O (A − B) O A∗
proof (induct k arbitrary: x)
case 0
with ⟨(x, y) ∉ B∗⟩ 0 show ?case using ccontr by auto
next
case (Suc i)
then have B: (x, y) ∉ B* and ASk:(x, y) ∈ A ″ i by auto
from ASk have ∃ z. (x, z) ∈ A ∧ (z, y) ∈ A ″ i using relpow-Suc-D2 [where ?R=A] by auto
then obtain z where xx: (x, z) ∈ A and (z, y) ∈ A ″ i by auto
then have zy: (z, y) ∈ A* using relpow-imp-rtrancl by auto
from xx show (x, y) ∈ A* O (A − B) O A∗
proof (cases (x, z) ∈ B)
case False
with xx zy show (x, y) ∈ A* O (A − B) O A∗ by auto
next
case True
then have (x, z) ∈ B* by auto
have [(x, z) ∈ B*; (z, y) ∈ B*] → (x, y) ∈ B* using rtrancl-trans [of x z]
proof (auto)
qed
qed

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

lemma SN-on-weakening:
assumes SN-on R1 A
shows SN-on (R1 ∩ R2) A
proof –
{-------------------
assume $\exists S. S \ 0 \in A \land \text{chain } (R1 \cap R2) \ 0$
then obtain $S$ where
\begin{itemize}
  \item $S^0: S \ 0 \in A$ and
  \item $SN$: chain $R1 \cap R2) \ S$
by auto
\end{itemize}
from $SN$ have $SN'$: chain $R1 \ S$ by simp
with $S^0$ and assms have $False$ by auto
}
then show ?thesis by force
qed

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

lemma ideriv-mono: $R \subseteq R' \Rightarrow S \subseteq S' \Rightarrow ideriv \ R \ S \ as \Rightarrow ideriv \ R' \ S' \ as$
unfolding ideriv-def INFM-nat by blast

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

lemma ideriv-split:
assumes ideriv: ideriv $R \ S$ as
and nideriv: $\neg ideriv (D \cap (R \cup S)) (R \cup S - D)$ as
shows $\exists \ i. \ ideriv (R - D) (S - D) (shift \ as \ i)$
proof -
  have $RS: R - D \cup (S - D) = R \cup S - D$ by auto
  from ideriv [unfolded ideriv-def]
  have as: $\bigwedge \ i. (as \ i, as (Suc \ i)) \in R \cup S$
  and inf: INFM i. (as i, as (Suc i)) \in R by auto
  show ?thesis
  proof (cases INFM i. (as i, as (Suc i)) \in D \cap (R \cup S))
    case True
    have ideriv (D \cap (R \cup S)) (R \cup S - D) as
      unfolding ideriv-def
    using as True by auto
    with nideriv show ?thesis ..
  next
  case False
  from False [unfolded INFM-nat]
  obtain i where $Dn: \bigwedge \ j. \ i < j \implies (as \ j, as (Suc \ j)) \notin D \cap (R \cup S)$
    by auto
  from $Dn$ as have as: $\bigwedge \ j. \ i < j \implies (as \ j, as (Suc \ j)) \in R \cup S - D$ by auto
  show ?thesis
    proof (rule exI [of - Suc \ i], unfold ideriv-def RS, insert as, intro conjI, simp, unfold INFM-nat, intro allI)
}

55
fix m
from inf [unfolded INFM-nat] obtain j where 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 ≥ 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

lemma Infm-shift: (INFM i. P (shift f n i)) = (INFM i. P (f i)) (is ?S = ?O)
proof
assume ?S
show ?O
unfolding INFM-nat-le
proof
fix m
from (?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:
\[(s, t) \in R^* \iff \exists ts \cdot \text{last} (s \# ts) = t \land (\forall i < \text{length} ts. ((s \# ts) ! i, (s \# ts) ! \text{Suc} i) \in R))\]
(is \(\equiv \equiv_{r})
proof
assume \(?r\)
then obtain ts where \(\text{last} (s \# ts) = t \land (\forall i < \text{length} ts. ((s \# ts) ! i, (s \# ts) ! \text{Suc} i) \in R))\) by auto
from Cons(1)[OF this] have rec: \((u, t) \in R^*\).
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 \(\text{steps}: \forall i < n. (S i, S (\text{Suc} i)) \in R\) by auto
let \(?ts = \text{map} (\lambda i. S (\text{Suc} i)) [0..< n]\)
show \(?r\)
proof (rule exI[of - ?ts], intro conjI,
cases n, simp add: t [symmetric], simp add: t [symmetric])
show \(\forall i < \text{length} \ ?ts. ((s \# \ ?ts) ! i, (s \# \ ?ts) ! \text{Suc} i) \in R\)
proof (intro allI impI)
fix i
assume i: \(i < \text{length} \ ?ts\)
then show \((s \# \ ?ts) ! i, (s \# \ ?ts) ! \text{Suc} i) \in R\)
proof (cases i, simp add: t [symmetric] steps)
  case (Suc j)
  with i steps show \(?thesis by simp\)
qed
qed
qed

lemma SN-reaches-NF:
assumes \(\text{SN-on } r \{x\}\)
shows \(\exists y. (x, y) \in r^* \land y \in \text{NF } r\)
using assms
proof (induct rule: \text{SN-on-induct}')
case (IH x)
show \(?case\)
proof (cases x \(\in \text{NF } r\) )
```plaintext
case True
  then show \( ?\text{thesis} \) by auto

next

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

qed

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

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

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

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

lemma some-NF-WCR:
  assumes SN: SN-on \( r \) \{x\}
    and WCR: WCR-on \( r \) \{x. SN-on r \{x\}\}
  shows \( y = some-NF r \ x \)

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

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 = λy. (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!)

context
  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 \(\text{the-NF} A a = \text{the-NF} A b\)
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 ⇒ bool) where
\(w\Diamond r \iff (r^{-1} O r) - \text{Id} \subseteq (r O r^{-1})\)

lemma weak-diamond-imp-CR:
assumes \(wd\):
\(w\Diamond r\)
shows \(CR r\)
proof (rule semi-confluence-imp-CR, rule)
fix \(x y\)
assume \((x, y) \in r^{-1} O r^*\)
then obtain \(z\) where \(\text{step}: (z, x) \in r\) and \(\text{steps}: (z, y) \in r^*\) by auto
from \(\text{steps}\)
have \(\exists u. (x, u) \in r^* \land (y, u) \in r^=\)
proof (induct)
case base
by (rule exI [of - x], insert \(\text{step}\), auto)
next
case \((\text{step} y' y)\)
from \(\text{step}(3)\) 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 \(\text{xu}\) 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 \(\text{step}(2)\) 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: \( \forall x. (t x, t (f x)) \in R \)
shows \( \neg SN-on R \{t x\} \)
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: \( \forall y. (tz, y) \in R \Longrightarrow y \notin range t \)
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: \( \forall x. (t x, t (f x)) \in R \)
shows \( \neg SN R \)
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: \( \forall t u R . P t \Longrightarrow Q R \Longrightarrow (t, u) \in R \Longrightarrow P u \land (f t, f u) \in g R \)
and t: P t
and R: Q R
and S: Q S
shows \( \forall (t, u) \in R^* \Longrightarrow (f t, f u) \in (g R)^* \)
  \land \( \forall (t, u) \in R^* \Longrightarrow (f t, f u) \in (g R)^* \land (g S) \land (g R)^* \)
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
auto

from \texttt{fg} \{OF \texttt{Pu R} step(2)\} have \texttt{Pv}: \texttt{P v} and \texttt{step}: (f u, f v) \in g R by auto
with \texttt{steps} have (f t, f v) \in (g R)^* by auto
with \texttt{Pv} show ?case by simp
qed simp

} note \texttt{main} = this

from \texttt{maint} \{OF - t\} have one: (t, u) \in R^* \rightarrow (f t, f u) \in (g R)^* by simp
show ?thesis by \texttt{simp}
proof (rule \texttt{conjI} \{OF \texttt{impI}\})
assume (t, u) \in R^* O S O R^*
then obtain s v where \texttt{ts}: (t, s) \in R^* and \texttt{sv}: (s, v) \in S and \texttt{vu}: (v, u) \in R^* by auto
from \texttt{maint} \{OF \texttt{ts}\} have \texttt{Ps}: \texttt{P s} and \texttt{ts}: (f t, f s) \in (g R)^* by simp
from \texttt{fg} \{OF \texttt{Ps S sv}\} have \texttt{Pv}: \texttt{P v} and \texttt{sv}: (f s, f v) \in g S by auto
from \texttt{maint} \{OF \texttt{vu \texttt{Pv}}\} have \texttt{vu}: (f v, f u) \in (g R)^* O g S O (g R)^* by auto
qed

\section{2.6 Terminating part of a relation}

\texttt{inductive-set}
\begin{align*}
\texttt{SN-part} :: & \quad 'a rel \Rightarrow 'a set \\
\texttt{for} & \quad 'a rel \\
\texttt{where} & \\
\texttt{SN-partI}: & \quad \forall y . (x, y) \in r \implies y \in \texttt{SN-part r} \implies x \in \texttt{SN-part r}
\end{align*}

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

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

\texttt{lemma \texttt{SN-on-SN-part-conv}:}
\begin{align*}
\texttt{SN-on r A} & \iff A \subseteq \texttt{SN-part r}
\end{align*}

\texttt{proof}
\begin{enumerate}
\item \texttt{fix x assume \texttt{SN-on r A} and \texttt{x \in A}}
\texttt{then have \texttt{x \in SN-part r} by (induct) (auto intro: \texttt{SN-partI})}
\item \texttt{moreover}
\item \texttt{fix \texttt{x assume} \texttt{A \subseteq SN-part r}}
\texttt{then have \texttt{x \in SN-part r} by \texttt{auto}}
\item \texttt{ultimately show \texttt{?thesis} by (force simp: \texttt{SN-defs})}
\end{enumerate}

\texttt{qed}

Special case for “full” termination.

\texttt{lemma \texttt{SN-SN-part-UNIV-conv}:}
\begin{align*}
\texttt{SN r} & \iff \texttt{SN-part r} = \texttt{UNIV}
\end{align*}

\texttt{using \texttt{SN-on-SN-part-conv [of r UNIV]} by auto}

62
lemma closed-imp-rtrancl-closed: assumes $L \subseteq A$
and $R``A \subseteq A$
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 $(x,y) \in a^\cdot n \Longrightarrow \exists m. m \geq n \land (x,y) \in b^{\cdot m}$
proof (induct $n$ arbitrary: $y$)
case $0$ thus ?case by (intro exI[of - 0], auto) 
next 
case $(Succ n \ z)$
from Succ(2) obtain $y$ where $xy$: $(x,y) \in a^{\cdot n}$ and $yz$: $(y,z) \in a$ by auto 
from Succ(1)[OF $xy$] obtain $m$ where $m: m \geq n$ and $xy$: $(x,y) \in b^{\cdot m}$ by auto 
from $yz$ assms have $(y,z) \in b^\cdot m$ unfolding relpow-add by auto 
with $k \ m$ show ?case by (intro exI[of - $m + k$], auto) 
qed 

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

lemma relpow-refl-mono: 
assumes refl:$(x,x) \in Rel$
shows $m \leq n \Longrightarrow (a,b) \in Rel^{\cdot m} \Longrightarrow (a,b) \in Rel^{\cdot n}$
proof (induct rule:dec-induct) 
case $(step \ i)$
hence $abi(a, b) \in Rel^{\cdot i}$ by auto 
from refl[of $b$] $abi$ relpowp-Succ-I[of $i \ x \ y$. $(x,y) \in Rel]$ show $(a, b) \in Rel^{\cdot i}$ 
Suc $i$ by auto 
qed 

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

shows \( P a \)

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

show \( \vdash \text{thesis} \)

proof (rule Wellfounded.acc.induct [OF \( a \), of \( P \)], rule IH)

fix \( x \)
assume \( \forall y. (y, x) \in R^{-1} \implies y \in \text{termi} \ R \)
from this [folded SN-on-cone-acc]
show \( \text{SN-on} \ R \{x\} \) by simp fast

qed auto

qed

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

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

next
assume 1: \( \forall x y z. (x, y) \in R \land (x, z) \in r^* \implies (y, z) \in \text{join} \ R \)
show \( \text{CR} \ R \)
proof
fix \( a b c \)
assume 2: \( a \in \text{UNIV} \) and 3: \( (a, b) \in r^* \) and 4: \( (a, c) \in r^* \)
then obtain \( n \) where \( (a, c) \in r^* \backslash n \) using rtrancl-in-UN-relpow by fast
with 2 3 show \( (b, c) \in \text{join} \ R \)
proof (induct \( n \) arbitrary: \( a b c \))

case 0 thus \( \vdash \) by auto

next

case \( \text{Suc} \ m \)
from \( \text{Suc}(4) \) obtain \( d \) where \( \text{ad:} (a, d) \in r^* \backslash \text{m} \) and \( \text{dc:} (d, c) \in r \) by auto
from \( \text{Suc}(1) \) [OF \( \text{Suc}(2) \) \( \text{Suc}(3) \) \( \text{ad} \)] have \( (b, d) \in \text{join} \ R \).
with 1 \text{dc \ joinI \ [of \ b - r \ c]} \text{ join-rtrancl-join show} \( \vdash \) by metis

qed

qed

definition strongly-confluent-on :: 'a rel => 'a set => bool
where
strongly-confluent-on \( r \ A \) \iff
\((\forall x \in A. \forall y z. (x, y) \in r \land (x, z) \in r \rightarrow (\exists u. (y, u) \in r^* \land (z, u) \in r^-))\)

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

lemma strongly-confluent-on-E11:
\[ \text{strongly-confluent-on } r A \implies x \in A \implies (x, y) \in r \implies (x, z) \in r \implies \exists u. (y, u) \in r^* \land (z, u) \in r^= \]

unfolding strongly-confluent-on-def by blast

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

unfolding strongly-confluent-on-def by auto

lemma strongly-confluent-E1n:
assumes scr: strongly-confluent r
shows \((x, y) \in r^= \implies (x, z) \in r^* \implies \exists u. (y, u) \in r^* \land (z, u) \in r^= \)
proof (induct n arbitrary: \(x \, y \, z\))
\begin{align*}
\text{case } \text{Suc } m & : \\
\text{from } \text{Suc}(3) & \text{ obtain } w \text{ where } xw : (x, w) \in r^m \text{ and } wz : (w, z) \in r \text{ by auto }
\end{align*}
\begin{align*}
\text{from } \text{Suc}(1) \ [\text{OF } \text{Suc}(2) \ xw] \text{ obtain } u \text{ where } yu : (y, u) \in r^* \text{ and } wu : (w, u) \in r^= \text{ by auto }
\end{align*}
\begin{align*}
\text{from } \text{strongly-confluent-on-E11 } [\text{OF scr, of w}] \ wz \ yu \ wu \text{ show } \text{case by (metis UnE converse-rtrancl-into-rtrancl iso-tuple-UNIV-I pair-in-Id-conv rtrancl-trans)}
\end{align*}
qed auto

lemma strong-confluence-imp-CR:
assumes strongly-confluent r
shows CR r
proof –
\begin{align*}
\{ \text{fix } x \, y \, z \\
\text{have } (x, y) \in r \implies (x, z) \in r^* \implies (y, z) \in \text{join } r \\
\text{by (cases } x = y, \text{ insert strongly-confluent-E1n } [\text{OF assms, blast+}]) \}
\end{align*}
\begin{align*}
\text{then show } CR r \text{ using partially-localize-CR by blast }
\end{align*}
qed

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

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

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

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

lemma CR-NF-conv:
assumes CR r and \(t \in NF r\) and \((u, t) \in r^{**}\)
shows \((a, t) \in r^\dagger\)
using assms
unfolding CR-imp-conversionIff-join [OF CR r] by (auto simp: NF-iff-no-step normalizability-def)
(metis (mono-tags) converse-rtranclE joinE)

lemma NF-join-imp-reach:
  assumes $y \in NF A$ and $(x, y) \in A^i$
  shows $(x, y) \in A^*$
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 $A \iff A O A \subseteq A$ unfolding trans-def by auto
lemma refl-O-iff: refl $A \iff Id \subseteq A$ unfolding refl-on-def by auto

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

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

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

lemma conversion-mono: $A \subseteq B \Rightarrow A^{**} \subseteq B^{**}$
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 $s \in X \forall t \in X. \exists u \in X. (t, u) \in R$
  shows $\neg SN-on R \{s\}$
by (meson SN-on-imp-on-minimal assms)

lemma SN-on-Image-rtrancl-iff[simp]: $SN-on R (R^* " X) \iff SN-on R X$ (is $?l = ?r$)
proof (intro iffI)
  assume $?l$ show $?r$ by (rule SN-on-subset2[OF - (?l)], auto)
qed (fact SN-on-imp-image-rtrancl)

lemma O-mono1: $R \subseteq R' \Rightarrow S O R \subseteq S O R'$ by auto
lemma O-mono2: $R \subseteq R' \Rightarrow R O T \subseteq R' O T$ by auto

lemma rtrancl-O-shift: $(S O R)^* O S = S O (R O S)^*$
proof (intro equalityI subrelI)
  fix $x y$
  assume $(x, y) \in (S O R)^* O S$
  by auto

66
then obtain $n$ where $(x, y) \in (S \circ R)^n$ $O S$ by blast

then show $(x, y) \in S \circ (R \circ S)^*$

proof (induct $n$ arbitrary: $y$)

case IH: $(\text{Suc } n)$

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

from IH.hyps[OF $xz$] $zy$ have $(x, y) \in S \circ (R \circ S)^*$ $O R \circ S$ by auto

then show ?case by (fold trancl-unfold-right, auto)

qed auto

next

fix $x$ $y$

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

then obtain $n$ where $(x, y) \in S \circ (R \circ S)^n$ $O S$

proof (induct $n$ arbitrary: $y$)

case IH: $(\text{Suc } n)$

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

from IH.hyps[OF $xz$] $zy$ have $(x, y) \in ((S \circ R)^* O S \circ R)$ $O S$ by auto

from this[folded trancl-unfold-right]

show ?case by (rule rev-subsetD[OF - O-mono2], auto simp: O-assoc)

qed auto

qed

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

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 $x$: $x \in X$

show SN-on $R \ \{x\}$ by (rule SN-on-subset2[OF - SN], insert $x$, auto)

qed

lemma SN-on-Un2:

assumes SN-on $R \ X$ and SN-on $R \ Y$ shows SN-on $R \ (X \cup Y)$

using assms by fast

lemma SN-on-UN:

assumes $\forall x. \ SN-on \ R \ (X \ x)$ shows SN-on $R \ (\bigcup x. \ X \ x)$

using assms by fast

lemma Image-subsetI: $R \subseteq R' \Longrightarrow R' \circ X \subseteq R' \circ X$ by auto

lemma SN-on-O-comm:

assumes SN: SN-on ($(R :: ('a \times 'b) set) \ O \ (S :: ('b \times 'a) set)) \ (S \circ X)$

shows SN-on $(S \circ R) \ X$

proof

fix seq :: nat $\Rightarrow \ 'b$ assume seq0: seq 0 $\in X$ and chain: chain $(S \circ R) \ seq$

qed
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 chain show False by auto  
qed

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

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

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

lemma rtrancl-O-push: S* O R ⊆ R O S*  
proof  
{ fix n  
  have ∃ s t. (s,t) ∈ S " " n O R → (s,t) ∈ R O S*  
  proof (induct n)  
    case (Suc n)  

  qed

68
then obtain \( u \) where \((s,u) \in S (u,t) \in R O S^*\) unfolding \texttt{relpow-Suc} by \texttt{blast}

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

\}
thus \(?thesis\) by \texttt{blast}
\texttt{qed}

\texttt{lemma} \texttt{rtrancl-U-push}: \((S \cup R)^* = R^* O S^*\)
\texttt{proof(intro equalityI subrelI)}
\texttt{fix x y}
\texttt{assume }\((x,y) \in (S \cup R)^*\)
\texttt{also have }\(...) \subseteq (S^* O R)^* O S^* by \texttt{regexp}
\texttt{finally obtain} \(z\) \texttt{where }\(xz:\ ((x,z) \in (S^* O R)^*\) \texttt{and }\(zy:\ (z,y) \in S^*\) by \texttt{auto}
\texttt{from }\(xz\) \texttt{have }\((x,z) \in R^* O S^*\)
\texttt{proof (induct rule: rtrancl-induct)}
\texttt{case }\((step \ z \ w)\)
then have \((x,w) \in R^* O S^* O S^* O R\) by \texttt{auto}
also have \(...) \subseteq R^* O S^* O R by \texttt{regexp}
also have \(...) \subseteq R^* O R O S^* using \texttt{rtrancl-O-push} by \texttt{auto}
also have \(...) \subseteq R^* O S^* by \texttt{regexp}
finally show \(?case.\)
\texttt{qed} \texttt{auto}
\texttt{with }\(zy\) \texttt{show }\((x,y) \in R^* O S^*\) by \texttt{auto}
\texttt{qed \texttt{regexp}}

\texttt{lemma} \texttt{SN-on-O-push}:
\texttt{assumes }\texttt{SN}: \texttt{SN-on R X shows SN-on (R O S^*) X}
\texttt{proof}
\texttt{fix seq}
\texttt{have }\texttt{SN}: \texttt{SN-on R (R^* " X) using SN-on-Image-rtrancl[OF SN].}
\texttt{moreover assume }\texttt{seq (\theta::nat) \in X}
then have \texttt{seq 0 \in R^* " X by auto}
ultimately show \texttt{chain (R O S^*) seq \Rightarrow False}
\texttt{proof(induct seq 0 arbitrary: seq rule: SN-on-induct)}
\texttt{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 \(s:\ (seq 0, s) \in R\) \texttt{and }\(s1:\ (s, seq 1) \in S^*
and \(t:\ (seq 1, t) \in R\) \texttt{and }\(t2:\ (t, seq 2) \in S^*\) by \texttt{auto}
\texttt{from s1} \texttt{t have }\((s,t) \in S^* O R\) by \texttt{auto}
\texttt{with }\texttt{rtrancl-O-push have }\texttt{st:\ (s,t) \in R O S^*\ by auto}
\texttt{from t2} \texttt{23 have }\((t, seq 3) \in S^* O R O S^*\) by \texttt{auto}
\texttt{also from }\texttt{rtrancl-O-push have }\(...) \subseteq R O S^* O S^* by \texttt{blast}
finally have \( t3 : (t, \text{seq } 3) \in R \circ O \circ S^* \) by regexp

let \( ?\text{seq} = \lambda i. \text{case } i \Rightarrow s | \text{Suc } 0 \Rightarrow t | i \Rightarrow \text{seq} (\text{Suc } i) \)

show \(?\text{case}\)

proof (rule IH)

from \( s \) show \((\text{seq } 0, ?\text{seq } 0) \in R\) by auto

show chain \((R \circ O \circ S^*) \ ?\text{seq}\)

proof (intro allI)

fix \( i \) show \((?\text{seq } i, ?\text{seq} (\text{Suc } i)) \in R \circ O \circ S^*\)

proof (cases \( i \))

  case 0 with \( \text{st} \) show \(?\text{thesis}\) by auto

next

  case (Suc \( i \)) with \( t3 \) IH show \(?\text{thesis}\) by (cases \( i \), auto simp: eval-nat-numeral)

qed

qed

qed

lemma \( \text{SN-on-Image-push}\):

assumes \( \text{SN} : \text{SN-on } R \circ X \) shows \( \text{SN-on } R \circ (S^* \ " X)\)

proof –

\{ fix \( n \)

  have \( \text{SN-on } R \ ((S^{"}n) \ " X)\)

  proof (induct \( n \))

    case 0 from \( \text{SN} \) show \(?\text{case}\) by auto

    case (Suc \( n \))

      from \( \text{SN-on-O-push}[OF this]\) have \( \text{SN-on } (R \circ O \circ S^*) \ ((S^{"}n) \ " X)\).

      from \( \text{SN-on-Image}[OF this]\) have \( \text{SN-on } (R \circ O \circ S^*) \ ((R \circ O \circ S^*) \ ((S^{"}n) \ " X))\).

      then have \( \text{SN-on } R \ ((R \circ O \circ S^*) \ ((S^{"}n) \ " X) \) by (rule \( \text{SN-on-mono} \),

                           auto)

      from \( \text{SN-on-subset2}[OF \text{Image-monO} [OF \text{push subset-refl]} \ this]\)

      have \( \text{SN-on } R \ ((R \ " (S^{"} \text{Suc } n) \ " X) \) by (auto simp: relcomp-Image)

      then show \(?\text{case}\) by fast

    qed

  \}

then show \(?\text{thesis}\) by fast

qed

end

lemma \( \text{not-SN-onI}[\text{intro}]\): \( f 0 \in X \implies \text{chain } R \circ f \implies \neg \text{SN-on } R \circ X\)

by (unfold \( \text{SN-on-def} \ \text{not-not}\), intro \( \text{exI} \ \text{conjI}\))

lemma \( \text{shift-comp}[\text{simp}]\): \( \text{shift } (f \circ \text{seq} ) \ n = f \circ (\text{shift } \text{seq} \ n) \) by auto

lemma \( \text{Id-on-union}\): \( \text{Id-on } (A \cup B) = \text{Id-on } A \cup \text{Id-on } B\) unfolding \( \text{Id-on-def}\)

by auto

lemma \( \text{relpow-union-cases}\): \( (a, d) \in (A \cup B)^{*} \implies (a, d) \in B^{*} \ \forall (\exists \ b \ c \ k \ m)\).
\[(a, b) \in B^\sim k \land (b, c) \in A \land (c, d) \in (A \cup B)^\sim m \land n = Suc (k + m)\]

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

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

  - let \(?AB = A \cup B\)

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

  - from \(ab\) show \(?case\)

  - proof

  - assume \((a, b) \in A\)

  - show \(?thesis\)

  - proof

  - (rule disjI2, intro exI conjI)

  - show \(Suc n = Suc (0 + n)\) by simp

  - show \((a, b) \in A\) by fact

  - qed (insert \(be\), auto)

  - next

  - assume \(ab: (a, b) \in B\)

  - from \(Suc(1)[OF be]\)

  - show \(?thesis\)

  - proof

  - assume \((b, c) \in B^\sim n\)

  - with \(ab\) show \(?thesis\)

  - by (intro disjII relpow-Suc-I2)

  - next

  - assume \(\exists c\ d\ k\ m. (b, c) \in B^\sim k \land (c, d) \in A \land (d, e) \in ?AB^\sim m \land n = Suc (k + m)\)

  - then obtain \(c\ d\ m\ n = Suc (k + m)\) by blast

  - with \(ab\) have \(ac: (a,c) \in B^\sim (Suc k)\) by (intro relpow-Suc-I2)

  - show \(?thesis\)

  - by (intro disjII exI conjI, rule \(ac\), (rule *)+, simp add: *)

  - qed

  - qed

  - qed simp

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

- assumes \(trans\ \(r\)\ refl \(r\)

- shows \(r^* = r\)

**proof**

- show \(r^* \subseteq r\)

- 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

- regexp

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

-
assumes trans r refl r
shows r O r = r
proof (intro equalityI)
  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
corverse-converse converse-Id

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) relto :: 'a rel ⇒ 'a rel ⇒ 'a rel where
relto $R S \equiv S^* O R O S^*$

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

**definition** SN-rel-on-alt :: 'a rel ⇒ 'a set ⇒ bool where
$SN-rel-on-alt R S T = (\forall f. chain (R \cup S) f \land f 0 \in T \rightarrow \neg (INFM j. (f j, f (Suc j)) \in R))$

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

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

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

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

proof –
let $?RS = relto R S$
have $(R \cup S)^* \subseteq S^* \cup ?RS^*$ by regexp
with steps have $(a,b) \in S^* \setminus (a,b) \in ?RS^*$ 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: $\\land i. (f i, f (Suc i)) \in ?RS$ by auto

let $?g = \lambda i. \ if i = 0 then a else f i$

have $\neg SN-on \ ?RS \{a\}$ unfolding SN-on-def not-not
proof (rule exI[of - $?g$], intro conjI allI)

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

proof (cases $i$)
case $(Suc j)$
show $?thesis$ using steps[of $i$] unfolding Suc by simp
next

73
case 0  
from steps[of 0, unfolded f0] Ssteps have steps: (a, f (Suc 0)) ∈ S^* O  
\( ?RS \) by blast  
\[ \text{have } (a, f (Suc 0)) \in ?RS \]
\[ \text{by (rule set-mp[OF - steps], regexp)} \]
thus \( \neg \text{thesis unfolding } 0 \) by simp  
qed  
qed simp  
with SN show False by simp  
qed  
qed  
qed

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

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

74
assume \( \text{Suc } n \leq rsi - ri \)
hence \((f \text{ (Suc } ri), f \text{ (Suc } (n + ri))) \in S^*\)
proof (induct \( n \), simp)
case (Suc \( n \))
hence \( \text{stepps: (} f \text{ (Suc } ri), f \text{ (Suc } (n + ri))\) \in S^* \) by simp
have \((f \text{ (Suc } (n + ri)), f \text{ (Suc } (n + ri))) \in S\)
using \( \text{inter [of } Suc \text{ } n \text{ } + \text{ } ri \text{ ] } Suc(2) \) by auto
with \text{stepps show} \ ?prop (\text{by simp})
qed

}\)
from this[of \( rsi - ri - 1 \) \( rsi \) have
\((f \text{ (Suc } ri), f \text{ (} rsi \text{ )}) \in S^* \) by simp
with \( ri \) \( rsi \) have \( \text{stepps: (} f \text{ (} rsi \text{ )}, f \text{ (} rsi \text{ )}) \in S^* \) by simp
with \( r-index[\text{of } i] \) have \((f \text{ (} rsi \text{ )}, f \text{ (} rsi \text{ )}) \in R O S^* \) unfolding \( \text{r} \) by auto
hence \((g i, g \text{ (Suc } i)) \in S^* \) \( \text{O } R \) \( \text{O } S^* \) using \( \text{rtrancl-refl} \) unfolding \( \text{g} \) by auto
hence \( nSN: \neg \text{SN-on (} S^* \) \( \text{O } R \) \( \text{O } S^* \) \) \{\( g \) 0\} unfolding \( \text{SN-defs} \) by blast
have \( \text{SN: SN-on (} S^* \) \( \text{O } R \) \( \text{O } S^* \) \} \( \{f \) \( \text{ (r-index 0)}\)\)
proof (rule steps-preserve-\text{SN-on-re}lto[\text{OF } SN])
show \((f 0, f \text{ (r-index 0)}) \in (R \cup S)^*\)
unfolding \( \text{rtrancl-fan-conv} \)
by (rule \( \text{exI[of } f\text{ ]} \), rule \( \text{exI[of } r-index \text{ 0]} \), insert \text{steps, auto})
qed
with \( nSN \) show \( \text{False unfolding } g \) ..
qed
qed

lemma \( \text{SN-rel-on-alt-imp-SN-rel-on: SN-rel-on-alt R S T } \Rightarrow \text{SN-rel-on R S T} \)
proof (unfold \( \text{SN-rel-on-def} \))
assume \( \text{SN: SN-rel-on-alt R S T} \)
show \( \text{SN-on (relto R S) } T \)
proof
fix \( f \)
assume start: \( f 0 \in T \) and \( \text{chain (relto R S) } f \)
hence \( \text{steps: } \bigwedge i. \ (f i, f \text{ (Suc } i)) \in S^* \) \( \text{O } R \) \( \text{O } S^* \) by auto
let \( \text{?prop = } \lambda i \ ai. \ (f i, bi) \in S^* \) \( \text{A } (bi, ai) \in R \) \( \land \ (ai, f \text{ (Suc } i)) \in S^* \)
\{ fix \( i \)
from \( \text{steps obtain } bi ai \text{ where } \text{?prop i ai bi by blast} \)
hence \( \forall i. \ ai. \ \text{?prop i ai bi by blast} \)
hence \( \forall i. \ \exists bi ai. \ \text{?prop i ai bi by blast} \)
from \( \text{choice[OF this]} \) obtain \( b \) \text{ where } \( \forall i. \ \exists ai. \ \text{?prop i ai (b i) by blast} \)
from \( \text{choice[OF this]} \) obtain \( a \) \text{ where steps: } \bigwedge i. \ \text{?prop i (a i) (b i) by blast} \)
from \( \text{steps[of 0]} \) have \( \text{fa0: (} f 0, a 0 \text{ ) } \in S^* \) \( \text{O } R \) by auto
let \( \text{?prop = } \lambda i. \ ai. \ (b i, ai) \in R \) \( \land \ (\forall j < \text{length } ai. \ ((a i \neq li) \land (a i \neq li) \land Suc j) \in S) \) \( \land \text{last (a i \# li)} = b \) \( (\text{Suc } i) \)

75
\{ 
fix i
from steps[of i] steps[of Suc i] have \((a i, f (Suc i)) \in S^\ast\) and \((f (Suc i), b (Suc i)) \in S^\ast\) by auto
from rtrancl-trans[OF this] steps[of i] have \(R: (b i, a i) \in R\) and \(S: (a i, b (Suc i)) \in S^\ast\) by blast

hence \((\forall j < length li. (((a i \# li) \# j, (a i \# li) \# Suc j) \in S^\ast) \# b Suc i) \# last (a i \# li) = b (Suc i)\) 
with \(R\) have \(?prop i li\) by blast

hence \(\exists li. \ ?prop i li\) ..
from choice[OF this] obtain l where steps: \(\forall i. \ ?prop i (l i)\) by auto
let \(?p\) = \(\lambda i. \ ?prop i (l i)\)
from steps have steps: \(\forall i. \ ?p i\) by blast

let \(?l\) = \(\lambda i. \ a i \# l i\)
let \(?l'\) = \(\lambda i. \ length (\?l i)\)
let \(?g\) = \(\lambda i. \ inf-concat-simple ?l' i\)
obtain g where \(g: \?prop i \# ?l l i = (\?g i in \?l ii \# jj)\) by auto
have \(g0: \ g 0 = a 0\) unfolding g Let-def by simp
with \(fa0\) have \(fg0: \ (f 0, g 0) \in S^\ast\) by auto
have \(fg0: \ (f 0, g 0) \in (R \cup S)^\ast\) by (rule set-mp[OF - \(fg0\)], regexp)

have \(len: \?g n = (i, j) \Rightarrow j < length (\?l i)\) 
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 \(gsteps: \?prop i \# (g i, g (Suc i)) \in R \cup S\)
proof
-
fix \( n \)

obtain \( i \, j \) where \( n : ?g \, n = (i, \, j) \) by (cases ?g \, n, auto)

show \((g \, n, \, g \, (Suc \, n)) \in R \cup S\)

proof (cases Suc \( j < \) length \((\forall \, i)\))

  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 \((\forall \, i) \) \( - \) \( 1 \) by auto

  with \( gn \) have \( gn: \, g \, n = last \,(\forall \, i) \) using last-conv-nth[of \( \forall \, i \)] by auto

  from \( gn \) \( gsn \) show \( ?thesis \) using steps[of \( i \)] steps[of Suc \( i \)] by auto

qed

qed

have inf\( R \): INFM \( j. \) \((g \, j, \, g \, (Suc \, j)) \in R \) 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 \((\forall \, i)\) using last-conv-nth[of \( \forall \, i \)] \( k2 \) by auto

from \( k2 \) \( gnk \) have \( ?g \, (Suc \, (n + \, ?k)) = (Suc \, i, \, 0) \) by auto

hence \( gnk3: \, g \, (Suc \, (n + \, ?k)) = a \, (Suc \, i) \) unfolding \(?g\) by auto

from steps[of \( i \)] steps[of Suc \( i \)] have \( main: \, (g \, (n + \, ?k), \, g \, (Suc \, (n + \, ?k))) \in R \)

  by (simp only: \( gnk2 \, gnk3 \))

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

  by (rule exI[of - \, n + \, ?k], auto simp: main[simplified])

qed

from \( fg0[unfolded \, rtrancl-fan-conv] \) obtain \( gg \, n \) where \( start: \, gg \, 0 = f \, 0 \)

and \( n: \, gg \, n = g \, 0 \) and \( steps: \, \bigwedge \, i. \, i < \, \mathbb{N} \, \Rightarrow \, (gg \, i, \, gg \, (Suc \, i)) \in R \cup S \) by auto

let \( \lambda \, ?h = \lambda \, i. \, if \, i < \, n \, then \, gg \, i \, else \, g \, (i - \, n) \)

obtain \( h \) where \( h: \, ?h \) by auto

\{ 

  fix \( i \)

  assume \( i \leq \, n \)

  have \( h \, i = gg \, i \) using \( i \) unfolding \( h \)

    by (cases \( i < \, n \), auto simp: \( n \))

  \}

note \( gg = \) this
from gg[of 0] ⟨f 0 ∈ T⟩ have h0: h 0 ∈ T unfolding start by auto
{
  fix i
  have (h i, h (Suc i)) ∈ R ∪ S
  proof (cases i < n)
    case True
    from steps[of i] gg[of i] gg[of Suc i] True show ?thesis by auto
  next
    case False
    hence i = n + (i - n) by auto
    then obtain k where i = n + k by auto
    from gsteps[of k] show ?thesis unfolding h i by simp
  qed
} note hsteps = this
from SN[unfolded SN-rel-on-alt-def, rule-format, OF conjI[OF allI[OF hsteps] h0]]
  have ¬ (INFM j. (h j, h (Suc j)) ∈ R).
moreover have INFM j. (h j, h (Suc j)) ∈ R unfolding INFM-nat-le
proof (rule)
  fix m
  from infR[unfolded INFM-nat-le, rule-format, of m]
  obtain i where i ≥ m and g: (g i, g (Suc i)) ∈ R by auto
  show ∃ n ≥ m. (h n, h (Suc n)) ∈ R
    by (rule exI[of - i + n], unfold h, insert g i, auto)
  qed
  ultimately show False ..
  qed
qed

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

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

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

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

lemma SN-rel-on-mono':
  assumes R: R ⊆ R' and S: S ⊆ R' ∪ S' and SN: SN-rel-on R' S' T
  shows SN-rel-on R S T
proof
  note conv = SN-rel-on-conv SN-rel-on-alt-def INFM-nat-le
  show ?thesis unfolding conv
  proof (intro allI impI)
    fix f
\textbf{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 \(\text{SN[unfolded conv, rule-format, OF this]}\) show \(\neg (\forall m. \exists n \geq m. (f n, f (\text{Suc} n)) \in R)\) using \(R\) by auto qed

\textbf{lemma} relto-mono:
\textbf{assumes} \(R \subseteq R'\) and \(S \subseteq S'\) shows relto \(R S \subseteq\) relto \(R' S'\) using \(\text{assms rtrancl-mono}\) by blast

\textbf{lemma} SN-rel-on-mono:
\textbf{assumes} \(R: R \subseteq R'\) and \(S: S \subseteq S'\) and \(\text{SN}: \text{SN-rel-on R' S' T}\) shows \(\text{SN-rel-on R S T}\) using \(\text{SN}\) unfolding \(\text{SN-rel-on-def}\) using \(\text{SN-on-mono[OF relto-mono]}\) by blast

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

\textbf{lemma} SN-rel-on-imp-SN-on:
\textbf{assumes} \(\text{SN-rel-on R S T}\) shows \(\text{SN-on R T}\) proof
\textbf{fix} \(f\)
\textbf{assume} chain \(R f\) and \(f 0: f 0 \in T\)
\textbf{hence} \(\land i. (f i, f (\text{Suc} i)) \in \text{relto R S}\) by blast
thus False using \(\text{assms f0 unfolding SN-rel-on-def SN-defs}\) by blast qed

\textbf{lemma} relto-Id: \(\text{relto R (S \cup \text{Id}) = relto R S}\) by simp

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

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

\textbf{lemma} SN-rel-on-ideriv: \(\text{SN-rel-on R S T =} \neg (\exists a. \text{ideriv R S as} \land as 0 \in T)\) (is \(?L = ?R)\)
\textbf{proof}
\textbf{assume} \(?L\)
\textbf{show} \(?R\)
\textbf{proof}
\textbf{assume} \(\exists a. \text{ideriv R S as} \land as 0 \in T\)
\textbf{then obtain} \text{where id: ideriv R S as and T: as 0 \in T}\) by auto
\textbf{note} id = id[unfolded ideriv-def]

79
from ⟨L⟩\[unfolded SN-rel-on-conv SN-rel-on-alt-def, THEN spec[of - as]]
  id T obtain i where i: \(\forall j. j \geq i \Rightarrow (as_j, as_{Suc j}) \not\in R\) by auto
  with id\[unfolded INFM-nat, THEN conjunct2, THEN spec[of - Suc i]] show False by auto
  qed
next
  assume ?R
  show ?L unfolding SN-rel-on-conv SN-rel-on-alt-def
    proof (intro allI impI)
      fix f
      presume steps: chain \((R \cup S) \cup f\) 
      obtain r where r: \(\forall j. r \equiv (f j, f_{Suc j}) \in R\) by auto
      show \((INFM j. (f j, f_{Suc j}) \in R)\) unfolding INFM-nat using i by blast
      qed
    qed

lemma SN-rel-to-SN-rel-alt: SN-rel R S =⇒ SN-rel-alt R S
  proof (unfold SN-rel-on-def)
    assume SN: SN (relto R S)
    show \(?\)thesis
      proof (unfold SN-rel-on-alt-def, intro allI impI)
        fix f
        presume steps: chain \((R \cup S) \cup f\)
        obtain r where r: \(\forall j. r \equiv (f j, f_{Suc j}) \in R\) by auto
        show \((INFM j. (f j, f_{Suc j}) \in R)\) unfolding INFM-nat using i by blast
      qed
    qed

80
fix \( n \)
assume \( Suc \ n \leq rsi - ri \)
hence \( (f \ Suc ri, f \ Suc \ (n + ri)) \in S^* \)
proof (induct \( n \), simp)
case \( Suc \ n \)
hence \( \text{steps:} \ (f \ Suc ri, f \ Suc \ (n + ri)) \in S^* \) by simp
have \( (f \ Suc \ (n + ri)), f \ Suc \ (Suc n + ri)) \in S \)
  using \( \text{inter[of} \ Suc \ n + ri \ Suc(2) \ \text{by} \ \text{auto} \)
with \( \text{steps show} \ ?\text{case by simp} \)
qed

\[
\text{lemma} \quad \text{SN-rel-alt-to-SN-rel} : \ \text{SN-rel-alt} \ R \ S \implies \text{SN-rel} \ R \ S
\]
proof (unfold \( \text{SN-rel-on-def} \))
assume \( \text{SN:} \ \text{SN-rel-alt} \ R \ S \)
show \( \text{SN} \ (\text{relto} \ R \ S) \)
proof
  fix \( f \)
  assume \( \text{chain} \ (\text{relto} \ R \ S) \ f \)
hence \( \text{steps:} \ \forall i. \ (f \ i, f \ Suc \ i) \in S^* \ R \ O \ S^* \) by auto
let \( ?\text{prop} = \lambda i \ ai \ bi. \ (f \ i, bi) \in S^* \land (bi, ai) \in R \land (ai, f \ Suc \ i) \in S^* \)
  
\[
\{
  \text{fix} \ i \\
  \text{from} \ \text{steps obtain} \ bi \ ai \ \text{where} \ ?\text{prop} \ i \ ai \ bi \ \text{by} \ \text{blast} \\
  \text{hence} \ \exists ai \ bi. \ ?\text{prop} \ i \ ai \ bi \ \text{by} \ \text{blast} \\
\}
\]
hence \( \forall i, \exists ai \ bi. \ ?\text{prop} \ i \ ai \ bi \ \text{by} \ \text{blast} \)
from choice[OF this] obtain \( b \ \text{where} \ \forall i, \exists ai. \ ?\text{prop} \ i \ ai \ (bi) \ \text{by} \ \text{blast} \)
from choice[OF this] obtain \( a \ \text{where} \ \text{steps:} \ \forall i. \ ?\text{prop} \ i \ (a \ i) \ (bi) \ \text{by} \ \text{blast} \)
  let \( ?\text{prop} = \lambda i \ li. \ (b i, a i) \in R \land (\forall j < \text{length} li. \ ((a i \ # \ li) \land j \land (a i \ # \ li)) \land Suc \ j) \in S) \land \text{last} (a i \ # \ li) = b \ Suc \ i) \)
  
\[
\{
  \text{fix} \ i \\
  \text{from} \ \text{steps[of} \ i \ \text{steps[of} \ Suc \ i \ \text{have} \ (a i, f \ Suc \ i) \in S^* \ \text{and} \ (f \ Suc \ i), \ b \ Suc \ i) \in S^* \ \text{by} \ \text{auto} \\
  \text{from} \ \text{rtrancl-trans[OF this] steps[of} \ i \ \text{have} \ R : \ (b \ i, a i) \in R \land S : \ (a i, b \ Suc \ Suc \ i) \in S^* \ \text{by} \ \text{blast}+ 
\}
\]
81
from $S$[unfolded rtrancl-list-conv] obtain $li$ where last ($a_i \# li$) = $b$ (Suc $i$) \land (\forall j < \text{length } li \cdot ((a_i \# li) \# j, (a_i \# li) \# \text{Suc } j) \in S)$ ..

with $R$ have ?prop $i$ $li$ by blast

hence $\exists li. ?prop i li$ ..

} hence $\forall i. \exists li. ?prop i li$ ..

from choice[OF this] obtain $l$ where steps: $\bigwedge i. ?prop i (l i)$ by auto

let $?p = \lambda i. ?prop i (l i)$

from steps have steps: $\bigwedge i. ?p i$ by blast

let $?l' = \lambda i. \text{length } (?l i)$

let $?g = \lambda i. \text{inf-concat-simple } ?l' i$

obtain $g$ where $g: \bigwedge i. g i = (\text{let } (ii, jj) = ?g i \in {?l ii} \# jj) \text{ by auto}$

have len: $\bigwedge i j n. ?g n = (i, j) \implies j < \text{length } (?l i)$

proof –

fix $i j n$

assume $n$: $?g n = (i, j)$

show $j < \text{length } (?l i)$

proof (cases $n$)

case $0$

with $n$ have $j = 0$ by auto

thus ?thesis by simp

next
case (Suc $nn$)

obtain $ii jj$ where $nn$: $?g nn = (ii, jj)$ by (cases $?g nn, auto)

show ?thesis

proof (cases Suc $jj < \text{length } (?l ii)$)

case $True$

with $nn$ Suc have $?g n = (ii, Suc jj)$ by auto

with $n$ True show ?thesis by simp

next
case $False$

with $nn$ Suc have $?g n = (Suc ii, 0)$ by auto

with $n$ show ?thesis by simp

qed

qed

qed

have $gsteps: \bigwedge i. (g i, g (\text{Suc } i)) \in R \cup S$

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 Suc $j < \text{length } (?l i)$)

case $True$

with $n$ have $?g (\text{Suc } n) = (i, \text{Suc } j)$ by auto

with $n$ have $gn: g n = {?l i} \# j$ and $gsn: g (\text{Suc } n) = {?l i} \# (\text{Suc } j)$ unfolding $g$ by auto

thus ?thesis using steps[of $i$] $True$ by auto

next

82
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 gm len[OF n] False have j = length (?l i) - 1 by auto
  with gn have gn: g n = last (?l i) using last-cone-rth[of ?l i] by auto
  from gm 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 SN[unfolded SN-rel-on-alt-def] gsteps infR show False by blast
  qed
  qed

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

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

lemma SN-rel-mono':
  R ⊆ R' ⇒ S ⊆ R' ∪ S' ⇒ SN-rel R' S' ⇒ SN-rel R S
  unfolding SN-rel-on-conv SN-rel-defs INFM-nat-le
  by (metis contr_subsetD sup.left_idem sup_mono)

lemma SN-rel-mono:
  assumes R: R ⊆ R' and S: S ⊆ S' and SN: SN-rel R' S'
  shows SN-rel R S
  using SN unfolding SN-rel-defs using SN-subset[OF relto-mono[OF R S]] by blast

83
lemmas $\text{SN-rel-alt-mono} = \text{SN-rel-mono[unfolded SN-rel-on-conv]}$

lemma $\text{SN-rel-imp-SN}$: assumes $\text{SN-rel R S}$ 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 $\text{SN-rel-defs SN-defs}$ by fast qed

lemma $\text{relto-trancl-conv}$: $(\text{relto } R S)^{\ast +} = ((R \cup S)^{\ast})^{\ast} O R O ((R \cup S)^{\ast})^{\ast}$ by regexp

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

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

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

lemma $\text{SN-rel-ideriv}$: $\text{SN-rel R S} = (\neg (\exists \text{ as. ideriv } R S \text{ as}))$ (is $\text{?L } = \text{?R}$) proof
  assume $\text{?L}$
  show $\text{?R}$ proof
    assume $\exists \text{ as. ideriv } R S \text{ as}$
    then obtain $\text{as}$ where $\text{id: ideriv } R S \text{ as}$ by auto
    note $\text{id } = \text{id[unfolded ideriv-def]}$
    from $?L[unfolded SN-rel-on-conv SN-rel-defs, THEN spec[of - as]]$
    id obtain $i$ where $i: \forall j. j \geq i \implies (\text{as } j, \text{ as } (\text{Suc } j)) \notin R$ by auto
    with $\text{id[unfolded INFM-nat, THEN conjunct2, THEN spec[of - Suc i]}]}$ show False by auto
  qed
next
  assume $\text{?R}$
  show $\text{?L}$ unfolding $\text{SN-rel-on-conv SN-rel-defs}$ proof (intro allI impI)
    fix $\text{as}$
    presume $\text{chain } (R \cup S)$ as
    with $?R[unfolded ideriv-def]$ have $\neg (\text{INFM } i. (\text{as } i, \text{ as } (\text{Suc } i))) \in R)$ by auto
    from this[unfolded INFM-nat] obtain $i$ where $i: \forall j. i < j \implies (\text{as } j, \text{ as } (\text{Suc } j)) \notin R$ by auto
    show $\neg (\text{INFM } j. (\text{as } j, \text{ as } (\text{Suc } j))) \in R)$ unfolding INFM-nat using $i$ by blast
  qed simp
lemma SN-rel-map:
  fixes R Rw R' Rw' :: 'a rel
  defines A :: 'a set ≡ R' ∪ Rw'
  assumes SN: SN-rel R Rw' and R: ∀ t. (s,t) ∈ R ⟹ (f s, f t) ∈ A'* O R' O A'*
  shows SN-rel Rw
  unfolding SN-rel-defs
proof
  fix g
  assume steps: chain (relto R Rw) g
  let f = λ i. (f (g i))
  obtain h where h = f by auto
  { fix i
    let m = λ (x,y). (f x, f y)
  }
  fix s t
  assume (s,t) ∈ Rw'*
  hence m (s,t) ∈ A'*
  proof (induct)
    case base show ?case by simp
    next
    case (step t u)
    from Rw (OF step(2)) step(3)
    show ?case by auto
  } note Rw = this
from steps have (g i, g (Suc i)) ∈ relto R Rw ..
from this
  obtain s t where gs: (g i,s) ∈ Rw'* and st: (s,t) ∈ R and tg: (t, g (Suc i)) ∈ Rw'*
  by auto
  from Rw (OF gs) R (OF st) Rw (OF tg)
  have step: (?f i, ?f (Suc i)) ∈ A'* O (A'* O R' O A') O A'*
  by fast
  have (?f i, ?f (Suc i)) ∈ A'* O R' O A' by (rule set-mp[OF - step], regexp)
  hence (h i, h (Suc i)) ∈ (relto R' Rw')`+
  unfolding A h relto-trancl-conv .
  } hence ¬ SN ((relto R' Rw')`+) 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 ⇒ 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'∗) (A'∗ O ((P ∪ Pw) ∩ M') O A'∗)
proof (rule ccontr)
let ?rel = SN-rel-ext-step P Pw R Rw
assume ¬ ?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}

86
let \( R' = \{\text{top}-s, \text{normal}-s\} \)
let \( Rw' = \{\text{top}-s, \text{top}-ns, \text{normal}-s, \text{normal}-ns\} \)

have \( A : A = A \) unfolding \( A \) by auto

have \( P : (P \cap M') = P' \) by auto

have \( Pw : (P \cup Pw) \cap M' = Pw' \) by auto

have \( R : (P \cup R) \cap M' = R' \) by auto

have \( Rw : A = Rw' \) unfolding \( A \) ...

\[
\begin{align*}
\text{let } & \text{seq} = \lambda s t g n ty. \quad s = g_0 \land t = g_n \land (\forall i < n. (g_i, g(Suc\ i)) \in \text{rel } (ty\ i)) \\
& \quad \land (\forall i \leq n. M(g_i)) \\
\end{align*}
\]

\[
\begin{align*}
\text{fix } & s t \text{ tt} \\
\text{assume } & m: M s \text{ and st: } (s, t) \in \text{UnM tt} \\
\text{hence } & \exists \text{ typ } \in \text{tt}. (s, t) \in \text{rel typ } \land M s \land M t \text{ unfolding } M' \text{ by auto} \\
\text{note } & \text{one-step } = \text{this} \\
\text{let } & \text{seq } = \lambda s t g n ty. s = g_0 \land t = g_n \land (\forall i < n. (g_i, g(Suc\ i)) \in \text{rel } (ty\ i)) \\
& \quad \land (\forall i \leq n. M(g_i)) \\
\end{align*}
\]

\[
\begin{align*}
\text{fix } & i \\
\text{assume } & i \leq n \\
\text{have } & M(g_i) \\
\text{proof } & (\text{cases } i) \\
\text{case } & 0 \\
\text{show } & \text{?thesis unfolding } 0 \text{ g0 by (rule m)} \\
\text{next} \\
\text{case } & (Suc\ j) \\
\text{with } & i \leq n \text{ have j < n by auto} \\
\text{from } & \text{steps}[OF this] \text{ show } \text{?thesis unfolding } \text{Suc M'} \text{ by auto} \\
\text{qed} \\
\end{align*}
\]

\[
\begin{align*}
\text{fix } & i \\
\text{assume } & i < n \text{ hence i': i < n by auto} \\
\text{from } & i' \text{ one-step}[OF min steps[OF i]] \\
\text{have } & \exists \text{ ty}. (g_i, g(Suc\ i)) \in \text{rel ty by blast} \\
\text{hence } & \forall i. (\exists \text{ ty}. i < n \rightarrow (g_i, g(Suc\ i)) \in \text{rel ty}) \text{ by auto} \\
\text{from } & \text{choice}[OF this] \\
\text{obtain } & tt \text{ where steps: } \land i. i < n \rightarrow (g_i, g(Suc\ i)) \in \text{rel (tt i)} \text{ by auto} \\
\text{from } & g0 gn steps min \\
\text{have } & \text{?seq s t g n tt by auto} \\
\text{hence } & \exists g n tt. \text{?seq s t g n tt by blast} \\
\text{note } & \text{A-steps } = \text{this} \\
\text{let } & \text{seqtt } = \lambda s t tt g n ty. \quad s = g_0 \land t = g_n \land n > 0 \land (\forall i < n. (g_i, g(Suc\ i)) \in \text{rel } (ty\ i)) \land (\forall i < n. t ty i \in tt) \\
\end{align*}
\]
\[
\text{fix } s \text{ } t \text{ } tt \\
\text{assume m: } M \text{ } s \text{ } \text{and st: } (s,t) \in A^* \text{ O } \text{?UnM tt O A}^* \\
\text{then obtain u v where su: } (s,u) \in A^* \text{ and uv: } (u,v) \in ?UnM tt \text{ and vt: } \\
(v,t) \in A^* \\
\text{by auto} \\
\text{from A-steps[OF m su] obtain g1 n1 ty1 where seq1: } ?seq s u g1 n1 ty1 \text{ by auto} \\
\text{from uv have M v unfolding M' by auto} \\
\text{from A-steps[OF this vt] obtain g2 n2 ty2 where seq2: } ?seq v t g2 n2 ty2 \text{ by auto} \\
\text{from seq1 have M u by auto} \\
\text{from one-step[OF this uv] obtain ty where ty: } ty \in tt \text{ and uv: } (u,v) \in \text{?rel ty by auto} \\
\text{let } ?g = \lambda i. \text{if } i \leq n1 \text{ then } g1 i \text{ else } g2 (i - (Suc n1)) \\
\text{let } ?ty = \lambda i. \text{if } i < n1 \text{ then ty1 } i \text{ else if } i = n1 \text{ then ty else ty2 } (i - (Suc n1)) \\
\text{let } ?n = Suc (n1 + n2) \\
\text{have ex: } \exists i < ?n. ?ty i \in tt \\
\text{by (rule exI[of - n1], simp add: ty) \\
\text{have steps: } \forall i < ?n. (?g i, ?g (Suc i)) \in ?rel (?ty i) \\
\text{proof (intro allI impI) \\
\text{fix i \\
\text{assume i < ?n \\
\text{show } (?g i, ?g (Suc i)) \in ?rel (?ty i) \\
\text{proof (cases i \leq n1) \\
\text{case True \\
\text{with seq1 seq2 uv show ?thesis by auto \\
\text{next \\
\text{case False \\
\text{hence i = Suc n1 + (i - Suc n1) by auto \\
\text{then obtain k where i = Suc n1 + k by auto} \\
\text{with i < ?n: have k < n2 by auto \\
\text{thus ?thesis using seq2 unfolding i by auto \\
\text{qed \\
\text{qed \\
\text{from steps seq1 seq2 ex \\
\text{have seq: } ?seqt s t tt ?g ?n ?ty by auto \\
\text{have } \exists g n ty. ?seqt s t tt g n ty \\
\text{by (intro exI, rule seq) \\
\} } \\
\text{note A-tt-A = this} \\
\text{let } ?tycon = \lambda ty1 ty2 tt ty'n. ty1 = ty2 \rightarrow (\exists i < n. ty' \ i \in tt) \\
\text{let } ?seqt = \lambda i ty g n ty'. f i = g 0 \wedge f (Suc i) = g n \wedge (\forall j < n. (g j, g (Suc j)) \in ?rel (ty' j)) \wedge (\forall j \leq n. M (g j)) \\
\wedge (?tycon (ty i) top-s {top-s} ty'n) \\
\wedge (?tycon (ty i) top-ns {top-s, top-ns} ty'n) \\
\wedge (?tycon (ty i) normal-s {top-s, normal-s} ty'n) \\
\{ \\
\text{fix i \\
\text{have } \exists g n ty'. ?seqt i ty g n ty' \\
\text{proof (cases ty i) \\
\} 
\}
case top-s
  from steps[of i, unfolded top-s]
  have (f i, f (Suc i)) ∈ ?P by auto
  from A-tt-A[OF min this[unfolded P]]
  show ?thesis unfolding top-s by auto
next
  case top-ns
  from steps[of i, unfolded top-ns]
  have (f i, f (Suc i)) ∈ ?Pw by auto
  from A-tt-A[OF min this[unfolded Pw]]
  show ?thesis unfolding top-ns by auto
next
  case normal-s
  from steps[of i, unfolded normal-s]
  have (f i, f (Suc i)) ∈ ?R by auto
  from A-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: \( \forall i. ?seqt i ty (g i) (n i) (ty' i) \).

let ?ind = inf-concat n
let ?g = \( \lambda k. (\lambda (i,j). g i j) (?ind k) \)
let ?ty = \( \lambda k. (\lambda (i,j). ty' i j) (?ind k) \)
have inf: INFM i. 0 < n i
  unfolding INFM-nat-le
proof (intro allI)
  fix m
  from inf1[unfolded INFM-nat-le]
  obtain k where k: k ≥ m and ty: ty k ∈ {top-s, top-ns} by auto
  show \( \exists k ≥ m. 0 < n k \)
  proof (intro exI conjI, rule k)
    from partial[of k] ty show \( \exists k < n k \) by (cases n k, auto)
  qed
  qed

note bounds = inf-concat-bounds[OF inf]
note inf-Suc = inf-concat-Suc[OF inf]
note inf-mono = inf-concat-mono[OF inf]
have ¬ SN-rel-ext P Pw R Rw M

unfolding SN-rel-ext-def simp-thms

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

  fix k

  obtain i j where ik: ?ind k = (i, j) by force

  from bounds[OF this] have: j < n i by auto

  show M (?g k) unfolding ik using partial[of i] j by auto

next

  fix k

  obtain i j where ik: ?ind k = (i, j) by force

  from bounds[OF this] have: j < n i by auto

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

  obtain i' j' where isk: ?ind (Suc k) = (i', j') by force

  have i'j': g i' j' = g i (Suc j)

  proof (rule inf-Suc[OF - ik isk])

    fix i

    from partial[of i]

    have: g i (n i) = f (Suc i) by simp

    also have: ... = g (Suc i) ∅ using partial[of Suc i] by simp

    finally show: g i (n i) = g (Suc i) ∅ .

  qed

  show (?g k, ?g (Suc k)) ∈ ?rel (?ty k)

  unfolding ik isk split i'j'

  by (rule step)

next

  show INFM i. ?ty i ∈ {top-s, top-ns}

  unfolding INFM-nat-le

  proof (intro allI)

    fix k

    obtain i j where ik: ?ind k = (i, j) by force

    from infI[unfolded INFM-nat] obtain i' where i': i' > i and ty: ty i' ∈ {top-s, top-ns} by auto

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

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

    from inf-mono[OF ik' i'] have: k ≤ k' by simp

    show ∃ k' ≥ k. ?ty k' ∈ {top-s, top-ns}

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

  qed

next

  show INFM i. ?ty i ∈ {top-s, normal-s}

  unfolding INFM-nat-le

  proof (intro allI)

    fix k

    obtain i j where ik: ?ind k = (i, j) by force

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

    from partial[of i'] ty obtain j' where j': j' < n i' and ty': ty' i' j' ∈ {top-s,
normal-s} by autorom inf-concat-surj[of - n, OF j]\ obtain k' where ik': ?ind k' = (i',j') ..rom inf-mono[OF ik ik' i'] have k: k ≤ k' by simp
show ∃ k' ≥ k. ?ty k' ∈ {top-s, normal-s}
  by (intro extI conjI, rule k, unfold ik' split, rule ty')
qed
qed
with assms show False by auto
qed

\textbf{lemma SN-rel-ext-map}: fixes \( P Pw R Rw P' Pw' R' Rw' :: 'a rel \) and \( M M' :: \) 'a ⇒ bool
\begin{itemize}
deﬁnes \( Ms: Ms \equiv \{(s,t), M' t\} \)
deﬁnes \( A: A \equiv (P' ∪ Pw' ∪ R' ∪ Rw') ∩ Ms \)
assumes \( SN: SN-rel-ext P Pw' R' Rw' M' \)
and \( P: \bigwedge s t. M s \implies M t \implies (s,t) ∈ P \implies (f s, f t) ∈ (A^* ∩ O (P' ∩ Ms) O A^*) \land I t \)
and \( Pw: \bigwedge s t. M s \implies M t \implies (s,t) ∈ Pw \implies (f s, f t) ∈ (A^* ∩ O ((P' ∪ Pw') ∩ Ms) O A^*) \land I t \)
and \( R: \bigwedge s t. I s \implies M s \implies M t \implies (s,t) ∈ R \implies (f s, f t) ∈ (A^* ∩ O ((P' ∪ R') ∩ Ms) O A^*) \land I t \)
and \( Rw: \bigwedge s t. I s \implies M s \implies M t \implies (s,t) ∈ Rw \implies (f s, f t) ∈ A^* \land I t \)
shows \( SN-rel-ext P Pw R Rw M \)
\end{itemize}
\textbf{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 \( ?rel = SN-rel-ext-step P Pw R Rw \)
show \( ?thesis \)
\textbf{proof} (rule ccontr)
assume \( ¬ ?thesis \)
from \textit{this}[unfolded SN-rel-ext-def]
\begin{itemize}
  obtain g ty
    where steps: \( \bigwedge i. (g i, g (Suc i)) \in ?rel (ty i) \)
    and min: \( \bigwedge i. M (g i) \)
  and \textit{inf1}: INFM i. ty i ∈ \{top-s, top-ns\}
  and \textit{inf2}: INFM i. ty i ∈ \{top-s, normal-s\}
  by auto
\end{itemize}
from \textit{inf1}[unfolded INFM-nat] \begin{itemize}
  obtain k where k: ty k ∈ \{top-s, top-ns\} by auto
\end{itemize}
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))) ∈ ?relty 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))) ∈ ?relty ty i ∧ I (g (Suc i))
    by (cases ty i, auto)
} note stepsI = this
{
fix i
have I (g (λ 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)) ∈ ?relty ty i)
    using stepsI[OF I[of i]] by auto
next
show INFM i. ty i ∈ {top-s, top-ns}
    unfolding Infn-shift[of λ i. i ∈ {top-s, top-ns} ty k]
    by (rule inf1)
next
show INFM i. ty i ∈ {top-s, normal-s}
    unfolding Infn-shift[of λ i. i ∈ {top-s, normal-s} ty k]
    by (rule inf2)

next

fix i

have A: A ⊆ Ms unfolding A by auto

from rtrancl-mono[OF this] have As: 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 ?case

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 (?f i, ?f (Suc i)) ∈ ?PPR

  by auto

  from set-mp[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: (?f i, ?f (Suc i)) ∈ Ms* by auto

thus ?thesis

proof (cases)

  case base

  with Suc show ?thesis by simp

next

case step

thus ?thesis unfolding Ms by simp

qed

qed
lemma SN-rel-ext-map-min: fixes \( P \) \( Pw \) \( R \) \( Rw \) \( P' \) \( Pw' \) \( R' \) \( Rw' \) :: 'a rel and \( M \) \( M' \)

with \( SN \)

show \( False \) unfolding \( A \) \( Ms \) by simp

qed

qed

\textbf{lemma} SN-rel-ext-map-min: fixes \( P \) \( Pw \) \( R \) \( Rw \) \( P' \) \( Pw' \) \( R' \) \( Rw' \) :: 'a rel and \( M \) \( M' \)

\textbf{defines} \( Ms \): \( Ms \equiv \{(s,t), M' t\} \)

\textbf{defines} \( A \): \( A \equiv P' \cap Ms \cup Pw' \cap Ms \cup R' \cup Rw' \)

\textbf{assumes} \( SN \): SN-rel-ext \( P' \) \( Pw' \) \( R' \) \( Rw' \) \( M' \)

\textbf{and} \( M \): \( s, t \mapsto M'(f t) \)

\textbf{and} \( M' \): \( s, t \mapsto M'(f s) \mapsto M'(f t) \mapsto (s,t) \in P \mapsto (f s, f t) \in (A^* O (P' \cap Ms) O A^*) \cap I t \)

\textbf{and} \( Pw \): \( s, t \mapsto M t \mapsto M'(f s) \mapsto M'(f t) \mapsto (s,t) \in Pw \mapsto (f s, f t) \in (A^* O (P' \cap Ms \cup Pw' \cap Ms) O A^*) \cap I t \)

\textbf{and} \( R \): \( s, t \mapsto M s \mapsto M t \mapsto M'(f s) \mapsto M'(f t) \mapsto (s,t) \in R \mapsto (f s, f t) \in (A^* O (P' \cap Ms \cup R') O A^*) \cap I t \)

\textbf{and} \( Rw \): \( s, t \mapsto M s \mapsto M t \mapsto M'(f s) \mapsto M'(f t) \mapsto (s,t) \in Rw \mapsto (f s, f t) \in A^* \cap I t \)

\textbf{shows} SN-rel-ext \( P \) \( Pw \) \( R \) \( Rw \) \( M \)

\textbf{proof} --

let \( ?Ms = \{(s,t), M' t\} \)

let \( ?A = (P' \cup Pw' \cup R' \cup Rw') \cap ?Ms \)

\{ 
  fix \( s \) \( t \)
  assume \( s, M' s \) and \( (s,t) \in A \)
  with \( M'(s, t) \in ?A \cap M' t \)
  unfolding \( Ms \) \( A \) by auto
\}

\} note Aone = this

\}

\} fix \( s \) \( t \)

assume \( s, M' s \) and \( \text{steps}: (s,t) \in A^* \)

from \( \text{steps} \) have \( (s,t) \in A^* \cap M' t \)

\textbf{proof} (induct)

\textbf{case} base from \( s \) show \( ?\text{case} \) by simp

\textbf{next}

\textbf{case} (step \( t \) \( u \))

\textbf{note} one = Aone[OF step(3)[THEN conjunct2] step(2)]

from \( \text{step(3)} \) one

have \( \text{steps}: (s,u) \in A^* O ?A \) by blast

have \( (s,u) \in ?A^* \)

by (rule set-mp[OF - steps], regexp)

with \( \text{one} \) show \( ?\text{case} \) by simp

\textbf{qed}

\} note Amany = this

let \( ?P = (A^* O (P' \cap Ms) O A^*) \)

let \( ?Pw = (A^* O (P' \cap Ms \cup Pw' \cap Ms) O A^*) \)

94
let \( \bar{R} = (A^* O (P' \cap Ms \cup R') O A^*) \)

let \( \bar{R}_w = A^* \)

let \( \bar{P}' = (\bar{A}^* O (P' \cap ?Ms) O \bar{A}^*) \)

let \( \bar{P}_w' = (\bar{A}^* O ((P' \cup Pw') \cap ?Ms) O \bar{A}^*) \)

let \( \bar{R}' = (\bar{A}^* O ((P' \cup R') \cap ?Ms) O \bar{A}^*) \)

let \( \bar{R}_w' = \bar{A}^* \)

show \?thesis

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

\[
\text{fix } s t
\]

assume \( s: M s \) and \( t: M t \) and step: \( (s,t) \in P \)

from \( P[\bar{O} f s t M[\bar{O} s] M[\bar{O} t] \text{ step}] \)

have \( (f,s,t) \in \bar{?P} \) and \( I: I t \) by auto

then obtain \( u v \) where \( su: (f,s,u) \in A^* \) and \( uv: (u,v) \in P' \cap Ms \)

and \( vt: (v,f,t) \in A^* \) by auto

from Amany[\bar{O} M[\bar{O} s] su] have \( su: (f,s,u) \in \bar{A}^* \) and \( u: M' u \) by auto

from \( uv \) have \( v: M' v \) unfolding \( Ms \) by auto

from Amany[\bar{O} v vt] have \( vt: (v,f,t) \in \bar{A}^* \) by auto

from su \( uv \) \( vt \) I

show \( (f,s,f,t) \in \bar{P}' \cap I t \) unfolding \( Ms \) by auto

next

\[
\text{fix } s t
\]

assume \( s: M s \) and \( t: M t \) and step: \( (s,t) \in Pw \)

from \( Pw[\bar{O} f s t M[\bar{O} s] M[\bar{O} t] \text{ step}] \)

have \( (f,s,t) \in \bar{?Pw} \) and \( I: I t \) by auto

then obtain \( u v \) where \( su: (f,s,u) \in A^* \) and \( uv: (u,v) \in P' \cap Ms \cup Pw' \cap Ms \)

and \( vt: (v,f,t) \in A^* \) by auto

from Amany[\bar{O} M[\bar{O} s] su] have \( su: (f,s,u) \in \bar{A}^* \) and \( u: M' u \) by auto

from \( uv \) have \( v: (u,v) \in (P' \cup Pw') \cap ?Ms \) and \( v: M' v \) unfolding \( Ms \)

by auto

from Amany[\bar{O} v vt] have \( vt: (v,f,t) \in \bar{A}^* \) by auto

from su \( uv \) \( vt \) I

show \( (f,s,f,t) \in \bar{P}_w' \cap I t \) by auto

next

\[
\text{fix } s t
\]

assume \( I: I s t s: M s \) and \( t: M t \) and step: \( (s,t) \in R \)

from \( R[\bar{O} f s t M[\bar{O} s] M[\bar{O} t] \text{ step}] \)

have \( (f,s,t) \in \bar{?R} \) and \( I: I t \) by auto

then obtain \( u v \) where \( su: (f,s,u) \in A^* \) and \( uv: (u,v) \in P' \cap Ms \cup R' \)

and \( vt: (v,f,t) \in A^* \) by auto

from Amany[\bar{O} M[\bar{O} s] su] have \( su: (f,s,u) \in \bar{A}^* \) and \( u: M' u \) by auto

from \( uv \) have \( M'[\bar{O} u, of v] \) have \( uv: (u,v) \in (P' \cup R') \cap ?Ms \) and \( v: M' v \)

unfolding \( Ms \)

by auto

from Amany[\bar{O} v vt] have \( vt: (v,f,t) \in \bar{A}^* \) by auto

from su \( uv \) \( vt \) I

show \( (f,s,f,t) \in \bar{R}' \cap I t \) by auto

next

\[
\text{fix } s t
\]
assume $I : I \ s$ and $s : M \ t$ and $t : M \ t$ and step: $(s,t) \in Rw$

from $Rw[OF I \ s \ t M[OF s] M[OF t]$ step]

have steps: $(f s, f t) \in ?Rw \ and \ I : I \ t$ by auto

from $Amany[OF M[OF s] steps] I$

show $(f s, f t) \in ?Rw' \ and \ I \ t$ by auto

qed

qed

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

proof –

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

show $?thesis$

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

fix $f$

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

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

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

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

{ fix $i$

  let $?ri = r-index \ i$

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

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

  obtain $ri \ rsi$ where $ri \ rsi = r-index \ Suc \ i$ by auto

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

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

  { fix $n$

    assume $Suc \ n \leq rsi - ri$

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

    proof (induct $n$, simp)

      case $Suc \ n$

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

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

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

      with stepps show $?case$ by simp

    qed

  }

}
\[ \text{from this[of rsi – ri – 1] risi have} \]
\[ (f (\text{Suc ri}), f rsi) \in S^\ast \text{ by simp} \]

\[ \text{with ri rsi have ssteps: (f (\text{Suc } ?ri), f ?rsi) \in S^\ast \text{ by simp}} \]

\[ \text{with r-index[of i] have (f ?ri, f ?rsi) \in R O S^\ast \text{ unfolding r by auto}} \]

\[ \text{hence (g i, g (\text{Suc } i)) \in S^\ast O R O S^\ast \text{ using rtrancl-refl unfolding g}} \]

\[ \text{by auto} \]

\[ \text{hence } \neg \text{SN} (S^\ast O R O S^\ast) \text{ unfolding SN-defs by blast} \]

\[ \text{with SN show False by simp} \]

\[ \text{qed} \]

\[ \text{simp} \]

\[ \text{qed} \]

\textbf{lemma rtrancl-list-conv:}

\[(s,t) \in R^\ast = \]

\[ (\exists \text{list. last } (s \# \text{ list}) = t \land (\forall i. i < \text{ length } \text{ list} \rightarrow ((s \# \text{ list}) \# i, (s \# \text{ list}) \# i) \in R)) \text{ (is } ?l = ?r) \]

\[ \text{proof} \]

\[ \text{assume } ?r \]

\[ \text{then obtain list where last } (s \# \text{ list}) = t \land (\forall i. i < \text{ length } \text{ list} \rightarrow ((s \# \text{ list}) \# i, (s \# \text{ list}) \# i) \in R) \ldots \]

\[ \text{thus } ?l \]

\[ \text{proof (induct list arbitrary: s, simp)} \]

\[ \text{case (Cons u ll) } \]

\[ \text{hence last } (u \# \text{ ll}) = t \land (\forall i. i < \text{ length } \text{ ll} \rightarrow ((u \# \text{ ll}) \# i, (u \# \text{ ll}) \# i) \in R) \text{ by auto} \]

\[ \text{from Cons(1)[OF this] have rec: } (u,t) \in R^\ast . \]

\[ \text{from Cons have } (s, u) \in R \text{ by auto} \]

\[ \text{with rec show } ?\text{case by auto} \]

\[ \text{qed} \]

\[ \text{next} \]

\[ \text{assume } ?l \]

\[ \text{from rtrancl-imp-seq[OF this] obtain S n where s: } S 0 = s \text{ and t: } S n = t \text{ and steps: } \forall i<n. (S i, S (Suc i)) \in R \text{ by auto} \]

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

\[ \text{show } ?r \]

\[ \text{proof (rule exI[of - } ?\text{list}], intro conjI,} \]

\[ \text{cases n, simp add: s[symmetric] t[symmetric], simp add: t[symmetric]} \]

\[ \text{show } \forall i < \text{ length } ?\text{list}. ((s \# ?\text{list}) \# i, (s \# ?\text{list}) \# i) \in R \]

\[ \text{proof (intro allI impI)} \]

\[ \text{fix i} \]

\[ \text{assume i: } i < \text{ length } ?\text{list} \]

\[ \text{thus } ((s \# ?\text{list}) \# i, (s \# ?\text{list}) \# i) \in R \]

\[ \text{proof (cases i, simp add: s[symmetric] steps)} \]

\[ \text{case (Suc j)} \]

\[ \text{with i steps show } ?\text{thesis by simp} \]

\[ \text{qed} \]

\textbf{97}
fun choice :: (nat ⇒ 'a list) ⇒ nat ⇒ (nat × nat) where

choice f 0 = (0,0)
| choice f (Suc n) = (let (i, j) = choice f n in
  if Suc j < length (f i)
    then (i, Suc j)
    else (Suc i, 0))

lemma SN-rel-imp-SN-rel : SN-rel R S → SN (relto R S)
proof –
  assume SN: SN-rel R S
  show SN (relto R S)
proof
  fix f
  assume ∀ i. (f i, f (Suc i)) ∈ relto R S
  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
  let ?prop = λ i. li. (b i, a i) ∈ R ∧ (∀ j < length li. ((a i ≠ li) ! j, (a i ≠ li) ! Suc j) ∈ S) ∧ length li = Suc i
  { fix i
    from steps[of i] have (a i, f (Suc i)) ∈ S* and (f (Suc i), b (Suc i)) ∈ S* by auto
    from rtrancl-trans[OF this] have R: (b i, a i) ∈ R and S: (a i, b (Suc i)) ∈ S* by blast+
    from S[unfolded rtrancl-list-conv] obtain li where last (a i ≠ li) = b (Suc i) ∧ (∀ j < length li. ((a i ≠ li) ! j, (a i ≠ li) ! Suc j) ∈ S)
      with R have ?prop i li by blast
    hence ∃ li. ?prop i li ..
  }
  hence ∀ i. ∃ li. ?prop i li ..
  from choice[OF this] obtain l where steps: ∀ i. ?prop i (l i) by auto
  let ?p = λ i. ?prop i (l i)
  from steps have steps: ∀ i. ?p i by blast
  let ?l = λ i. a i ≠ l i
  let ?g = λ i. choice (λ j. ?l j) i
  obtain g where g: ∀ i. g i = (let (ii, jj) = ?p i in ?l ii ! jj) by auto
  have len: ∀ i j. ?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 have ?g (Suc n) = (ii, Suc jj) by auto
    with n True show ?thesis by simp

  next
    case False
    with nn have ?g (Suc n) = (Suc ii, 0) by auto
    with n show ?thesis by simp
  
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 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

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] False have j = length (?l i) − 1 by auto

with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto

from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto

qed

qed

have infR: ∀ 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 \( \mathcal{A} = \text{length} \, (?l \, i) - 1 - j \)

obtain \( k \) where \( k = j + \mathcal{A} \) by auto

from \( j \, k \) have \( \mathcal{A}2: \, k = \text{length} \, (?l \, i) - 1 \) and \( \mathcal{A}3: \, j + \mathcal{A} < \text{length} \, (?l \, i) \) by auto

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

from this

\[
\begin{align*}
\text{OF } n, \text{ of } \mathcal{A} \, k \\
\text{have } \mathcal{A}k: \, g \, (n + \mathcal{A}) = (i, k) \text{ by } (\text{simp only: } k) \\
\text{hence } g \, (n + \mathcal{A}) = !l \, i \, ! k \text{ unfolding } g \text{ by auto} \\
\text{hence } \mathcal{A}k2: \, g \, (n + \mathcal{A}) = \text{last } (?l \, i) \text{ using } \text{last-conv-nth[of } ?l \, i \text{] } k2 \text{ by auto} \\
\text{from } k2 \, gk \text{ have } \mathcal{A}g: \, g \, (\text{Suc } (n + \mathcal{A})) = (\text{Suc } i, 0) \text{ by auto} \\
\text{hence } \mathcal{A}gk2: \, g \, (\text{Suc } (n + \mathcal{A})) = a \, (\text{Suc } i) \text{ unfolding } g \text{ by auto} \\
\text{from } \text{steps[of } i \text{] steps[of } \text{Suc } i \text{] have } \text{main}: \, (g \, (n + \mathcal{A}), g \, (\text{Suc } (n + \mathcal{A}))) \in R \\
\text{by } (\text{simp only: } \mathcal{A}gk2 \, \mathcal{A}gsk2) \\
\text{show } \exists \, j \geq n. \, (g \, j, g \, (\text{Suc } j)) \in R \\
\text{by } (\text{rule } \text{exI[of } - n + \mathcal{A}]k, \text{ auto simp: main[simplified]})
\end{align*}
\]

qed

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

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 (\text{blast intro: 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 \( \text{SN SN-subset[OF - relto-mono[OF } R \, S]] by blast} \)

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

lemma SN-relto-Id:
\[
SN (\text{relto } R (S \cup Id)) = SN (\text{relto } R S)
\]
by (simp only: relto-Id)

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

lemma trans-subset-SN:
assumes trans R and R \subseteq (r \cup s) and SN r and SN s
shows SN R

proof
fix f :: nat\Rightarrow' a
assume f 0 \in UNIV
and chain: chain R f
have \*: \(\forall i. i < j \implies (f i, f j) \in r \cup s\)
using assms and chain-imp-trancl [OF chain] by auto
let ?M = \{i. \(\forall j > i. (f i, f j) \notin r\}\}
show False
proof (cases finite ?M)
let \(?n = \text{Max } ?M\)
assume finite ?M
with Max-ge have \(\forall i \in ?M. i \leq ?n\) by simp
then have \(\forall k \geq \text{Suc } ?n. \exists k' > k. (f k, f k') \in r\) by auto
with steps-imp-chainp [of Suc ?n \(\lambda x y. (x, y) \in r\)] and assms
show False by auto
next
assume infinite ?M
then have INFM j. j \in ?M by (simp add: Inf-many-def)
then interpret infinitely-many \(\lambda i. i \in ?M\) by (unfold-locales) assumption
define g where \([simp]: g = index\)
have \(\forall i. (f (g i), f (g (\text{Suc } i))) \in s\)
proof
fix i
have less: \(g i < g (\text{Suc } i)\) using index-ordered-less [of i Suc i] by simp
have g i \in ?M using index-p by simp
then have \((f (g i), f (g (\text{Suc } i))) \notin r\) using less by simp
moreover have \((f (g i), f (g (\text{Suc } i))) \in r \cup s\) using s [OF less] by simp
ultimately show \((f (g i), f (g (\text{Suc } i))) \in s\) by blast
qed
with \(\langle SN s\rangle\) show False by (auto simp: SN-defs)
qed

lemma SN-Un-conv:
assumes trans (r \cup s)
shows SN (r \cup s) \longleftrightarrow SN r \land SN s
(is SN ?r \longleftrightarrow ?rhs)

proof
assume \( SN \ (r \cup s) \) thus \( SN \ r \land SN \ s \)
using \( SN\text{-subset[of ?r]} \) by blast
next
assume \( SN \ r \land SN \ s \)
with \( trans\text{-subset-SN[OF assms subset-refl]} \) show \( SN \ ?r \)
by simp
qed

lemma \( SN\text{-relto-Un} \):
\[ SN \ (relto \ (R \cup S) \ Q) \longleftrightarrow SN \ (relto \ R \ (S \cup Q)) \land SN \ (relto S \ Q) \]
(is \( SN \ ?a \longleftrightarrow SN \ ?b \land SN \ ?c \))
proof
have \( eq : ?a^+ = ?b^+ \cup ?c^+ \) by regexp
from \( SN\text{-Un-conv[of ?b^+ \ ?c^+, unfolded eq[symmetric]]} \)
show \( ?thesis \) unfolding \( SN\text{-trancl-SN-conv} \) by simp
qed

lemma \( SN\text{-relto-split} \):
assumes \( SN \ (relto \ r \ (s \cup q_2) \cup relto \ q_1 \ (s \cup q_2)) \) (is \( SN \ ?a \))
and \( SN \ (relto \ s \ q_2) \) (is \( SN \ ?b \))
shows \( SN \ (relto \ r \ (q_1 \cup q_2) \cup relto \ s \ (q_1 \cup q_2)) \) (is \( SN \ ?c \))
proof
have \( ?c^+ \subseteq ?a^+ \cup ?b^+ \) by regexp
from \( trans\text{-subset-SN[OF - this, unfolded SN\text{-trancl-SN-conv, OF - assms]}} \)
show \( ?thesis \) by simp
qed

lemma \( relto\text{-trancl-subset} \):
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 \( relto\text{-fun} :: \ 'a \ rel \Rightarrow \ 'a \ rel \Rightarrow \ nat \Rightarrow \ (nat \Rightarrow \ 'a) \Rightarrow \ (nat \Rightarrow \ bool) \Rightarrow \ nat \Rightarrow \ 'a \times \ 'a \Rightarrow \ bool \) where
\( relto\text{-fun}: \) as \( 0 = a \Rightarrow as \ m = b \Rightarrow \)
\( \land i . \ i < m \Rightarrow \)
\( (sel \ i \Rightarrow (as \ i, as \ (Suc \ i)) \in A) \land \ (\neg \ sel \ i \Rightarrow (as \ i, as \ (Suc \ i)) \in B)) \)
\( \Rightarrow n = card \ { i . \ i < m \land sel \ i} \)
\( \Rightarrow (n = 0 \iff m = 0) \Rightarrow relto\text{-fun} \ A \ B \ n \ as \ sel \ m \ (a,b) \)

lemma \( relto\text{-funD} \):
assumes \( relto\text{-fun} \ A \ B \ n \ as \ sel \ m \ (a,b) \)
sows as \( 0 = a \Rightarrow as \ m = b \)
\( \land i . \ i < m \Rightarrow sel \ i \Rightarrow (as \ i, as \ (Suc \ i)) \in A \)
\( \land i . \ i < m \Rightarrow \neg \ sel \ i \Rightarrow (as \ i, as \ (Suc \ i)) \in B \)
\( n = card \ { i . \ i < m \land sel \ i} \)
\( n = 0 \iff m = 0 \)
using assms[unfolded relto-fun.simps] by blast+

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

lemma relto-into-relto-fun: assumes (a,b) \in relto A B
shows \exists as sel m. relto-fun A B (Suc 0) as sel m (a,b)
proof –
from assms obtain a' b' where aa: (a,a') \in B^* and ab: (a',b') \in A
and bb: (b',b) \in B^* by auto
from aa[unfolded rtrancl-fun-conv] obtain f1 n1 where
f1: f1 0 = a f1 n1 = a' \land i < n1 \implies (f1 i, f1 (Suc i)) \in B by auto
from bb[unfolded rtrancl-fun-conv] obtain f2 n2 where
f2: f2 0 = b' f2 n2 = b \land i < n2 \implies (f2 i, f2 (Suc i)) \in B by auto
let ?gen = \lambda aa ab bb i. if i < n1 then aa i else if i = n1 then ab else bb (i - Suc n1)
let ?f = ?gen f1 a' f2
let ?sel = ?gen (\lambda -. False) True (\lambda -. False)
let ?m = 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 (Suc i)) \in A) \land (\neg ?sel i \longrightarrow (?f i, ?f (Suc i)) \in B)
proof (cases i < n1)
case True
with f1(3)[OF this] f1(2) show ?thesis by (cases Suc i = n1, auto)
next
case False
note nle = 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 - Suc n1
have i: i = Suc n1 + j and j: j < n2 using i True unfolding j-def by auto
thus ?thesis using f2 by auto
qed
qed (insert f1 f2, auto)
qed

lemma relto-fun-trans: assumes ab: relto-fun A B n1 as1 sel1 m1 (a,b)
and bc: relto-fun A B n2 as2 sel2 m2 (b,c)
shows \exists as sel. relto-fun A B (n1 + n2) as sel (m1 + m2) (a,c)
proof
from relto-funD[OF ab]
have 1: as1 0 = a as1 m1 = b
\( \land i. i < m1 \Rightarrow (\text{sel1 } i \rightarrow (as1 \ i, as1 \ (Suc \ i)) \in A) \land (\neg \text{sel1 } i \rightarrow (as1 \ i, as1 \ (Suc \ i)) \in B) \)
\( \n1 = 0 \iff m1 = 0 \) and card1: \( n1 = \text{card} \{ i. i < m1 \land \text{sel1 } i \} \) by blast+
from relto-funD[OF bc]
have 2: as2 0 = b as2 m2 = c
\( \land i. i < m2 \Rightarrow (\text{sel2 } i \rightarrow (as2 \ i, as2 \ (Suc \ i)) \in A) \land (\neg \text{sel2 } i \rightarrow (as2 \ i, as2 \ (Suc \ i)) \in B) \)
\( n2 = 0 \iff m2 = 0 \) and card2: \( n2 = \text{card} \{ i. i < m2 \land \text{sel2 } i \} \) by blast+
let ?as = \( \lambda \ i. \text{if } i < m1 \text{ then } as1 \ i \text{ else } as2 \ (i - m1) \)
let ?sel = \( \lambda \ i. \text{if } i < m1 \text{ then } \text{sel1 } i \text{ else } \text{sel2 } (i - m1) \)
let ?m = m1 + m2
let ?n = n1 + n2
show \(?\text{thesis}\)
proof
(rule exI[of \ ?as], rule exI[of \ ?sel], rule relto-fun)
have id: \( \{ i. i < ?m \land \text{sel1 } i \} = \{ i. i < m1 \land \text{sel1 } i \} \cup (\text{(+)} \ m1) \ ' \{ i. i < m2 \land \text{sel2 } i \} \)
(is - = \ ?A \cup \ ?f ' \ ?B)
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 \( ?n = \text{card} \{ i. i < ?m \land \text{sel1 } i \} \) unfolding card1 card2 id by simp
next
fix i
assume i: \( i < ?m \)
show (?sel i \rightarrow (\text{as1 } i, \text{as1 } (Suc \ i)) \in A) \land (\neg \text{sel1 } i \rightarrow (\text{as1 } i, \text{as1 } (Suc \ i)) \in B)
proof (cases i < m1)
case True
from 1 2 have \( [\text{simp}]: \text{as2 } 0 = \text{as1 } m1 \) 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 \( (a,b) \in (\text{relto } A \ B) \)\ "n
shows \( \exists \text{ as sel } m. \text{ relto-fun } A \ B \ n \) as \( \text{sel } m \ (a,b) \)
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 (Suc n c)
from relpow-Suc-E[OF Suc(2)]
obtain \(b\) where \(ab : (a, b) \in (\text{relto A B})^{\sim n}\) and \(bc : (b, c) \in \text{relto A B}\) by auto
from Suc(1)[OF ab] obtain as sel m where
IH: relto-fun A B n as sel m (a, b) by auto
from relto-into-relto-fun[OF bc] obtain as sel m where relto-fun A B (Suc 0) as sel m (b, c) by blast
from relto-fun-trans[OF IH this] show ?case by auto
qed

lemma relto-fun-into-reltos: assumes relto-fun A B n as sel m (a,b)
shows \((a,b) \in (\text{relto A B})^{\sim n}\)
proof -
  note * = relto-funD[OF assms]
  {
    fix \(m'\)
    let \(?c = \lambda m'. \text{card}\ \{i. i < m' \land sel i\}\)
    assume \(m' \leq m\)
    hence \((?c m' > 0 \rightarrow (as 0, as m') \in (\text{relto A B})^{\sim ?c m'} \land (?c m' = 0 \rightarrow (as 0, as m') \in B^\ast)\)
    proof (induct \(m'\))
      case (Suc \(m')\)
      let \(?x = as 0\)
      let \(?y = as m'\)
      let \(?z = (\text{Suc } m')\)
      let \(?C = ?c (\text{Suc } m')\)
      have \(?C = ?c m' + (\text{if } (\text{sel } m') \text{ then } 1 \text{ else } 0)\)
      proof -
        have id: \(\{i. i \in \text{Suc } m' \land sel i\} = \{i. i < m' \land sel i\} \cup (\text{if sel } m' \text{ then } \{\} \text{ else } \{\})\)
        by (cases sel m', auto, case-tac \(x = m', auto\))
        show \(?thesis unfolding id by auto\)
      qed
      from Suc(2) have \(m' : m' \leq m \text{ and } lt: m' < m \text{ by auto}\)
      from Suc(1)[OF lt] have IH: \(?c m' > 0 \rightarrow (\?x, \?y) \in (\text{relto A B})^{\sim ?c m'}\)
      proof
        \(?c m' = 0 \rightarrow (\?x, \?y) \in B^\ast\) by auto
      from *(3-4)[OF lt] have \(yz: sel m' \rightarrow (\?y, \?z) \in A \land sel m' \rightarrow (\?y, \?z) \in B^\ast\) by auto
      show ?thesis
      qed
    proof (cases \(?c m' = 0\) )
      case True note \(c = this\)
      from IH(2)[OF this] have \(xy: (\?x, \?y) \in B^\ast\) by auto
      show ?thesis
      proof (cases sel m')
    qed
  
  105
case False
  from \( x y z (2) | \text{OF False} \) have \( x z: (\alpha x, \alpha z) \in B^* \) by auto
  from False \( c \) have \( C: \alpha C = 0 \) unfolding \( C \) by simp
  from \( x z \) show \( ?\text{thesis unfolding } C \) by auto
next
  case True
  from \( x y z (1) | \text{OF True} \) have \( x z: (\alpha x, \alpha z) \in \text{relto } A B \) by auto
  from True \( c \) have \( C: \alpha C = 1 \) unfolding \( C \) by simp
  from \( x z \) show \( ?\text{thesis unfolding } C \) by auto
qed
next
case False
  hence \( c: \alpha c m' > 0 \) (\( \alpha c m' = 0 \)) = False by arith+
  from \( IH (1) | \text{OF c(1)} \) have \( x y: (\alpha x, \alpha y) \in (\text{relto } A B) \) "" \( \alpha c m' \).
  show \( ?\text{thesis} \)
  proof (cases \( \text{sel } m' \))
    case False
    from \( c \) obtain \( k \) where \( ck: \alpha c m' = \text{Suc } k \) by (cases \( \alpha c m' \), auto)
    from relpow-Suc-E [\text{OF } xy]\[\text{unfolded this}] obtain \( u \) where \( xu: (\alpha x, u) \in (\text{relto } A B) \) "" \( k \) and \( uy: (u, \alpha y) \in \text{relto } A B \) by auto
    from \( uy z (2) | \text{OF False} \) have \( uz: (u, \alpha z) \in \text{relto } A B \) by force
    with \( xu \) have \( x z: (\alpha x, \alpha z) \in (\text{relto } A B) \) "" \( \alpha c m' \) unfolding \( ck \) by auto
    from False \( c \) have \( C: \alpha C = \alpha c m' \) unfolding \( C \) by simp
    from \( x z \) show \( ?\text{thesis unfolding } C \) by auto
  next
    case True
    from \( x y z (1) | \text{OF True} \) have \( x z: (\alpha x, \alpha z) \in (\text{relto } A B) \) "" (\( \text{Suc } (\alpha c m') \))
    by auto
    from \( c \) True have \( C: \alpha C = \text{Suc } (\alpha c m') \) unfolding \( C \) by simp
    from \( x z \) show \( ?\text{thesis unfolding } C \) by auto
  qed
  qed
  qed simp
}
from this[of m] \ast show \( ?\text{thesis by auto} \)
qed

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

lemma relto-fun-intermediate: assumes \( A \subseteq C \) and \( B \subseteq C \)
and \( rf: \text{relto-fun } A B \text{ as sel } m (\alpha a, \alpha b) \)
shows \( i \leq m \Rightarrow (\alpha a, \alpha i) \in C^* \)
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 ∃ 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 u : (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

qed

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

\textbf{qed}

\textbf{lemma} relative-ending:
\begin{itemize}
  \item \textbf{assumes} chain: chain \((R \cup S)\) \(t\)
  \item \(t 0: t 0 \in X\)
  \item and \(SN: SN-on (relto R S)\) \(X\)
  \item \textbf{shows} \(\exists j. \forall i \geq j. (t i, Suc (t i)) \in S - R\)
\end{itemize}
\textbf{proof} (rule ccontr)
\begin{itemize}
  \item assume \(\neg \exists t = \lambda i. t \ ((((Suc o f)^{\circ i}) t)) 0\)
  \item have \(\forall i. (t i, Suc (t i)) \in (relto R S)^+\)
  \item \textbf{proof}
    \begin{itemize}
      \item fix \(i\)
      \item from R-steps \textbf{have} \(leg: i \leq f i\) and \textbf{step}: \((t(f i), Suc (f i)) \in R\) by auto
      \item from chain-imp-rtrancl \(\{OF chain leg\}\) \textbf{have} \((t i, Suc (f i)) \in (R \cup S)^+\).
      \item with \textbf{step} \textbf{have} \((t i, Suc (f i)) \in (R \cup S)^+ O R\) by auto
      \item then \textbf{show} \((t i, Suc (f i)) \in (relto R S)^+\) by regexp
    \end{itemize}
\end{itemize}

\textbf{qed}

\item then have \(chain ((relto R S)^+)\) \(\forall t\) by simp
\item with \(t 0\) \textbf{have} \(\neg SN-on ((relto R S)^+)\) \(X\) by (unfold SN-on-def, auto intro: exI[of - \(?t\)])
\item with \(SN-on-trancl[\{OF SN\}\] \textbf{show} \textbf{False} by auto
\end{itemize}
\textbf{qed}

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

\textbf{lemma} \(SN-on-relto-Un:\)
\begin{itemize}
  \item \textbf{assumes} closure: relto \((R \cup R')\) \(S \subseteq X\)
  \item \textbf{shows} \(SN-on (relto (R \cup R') S)\) \(X \leftarrow SN-on (relto R (R' \cup S))\) \(X \land SN-on\)
\end{itemize}
\[(\text{relto } R' S) \ X\]
\[(\text{is } ?c \iff ?a \land ?b)\]

**proof (safe)**

- Assume \(SN: ?a\) and \(SN': ?b\)
- From \(SN\) have \(SN: \text{SN-on (relto (relto R S) (relto R’ S))} X\) by (rule SN-on-subset1)

**regexp**

- Show \(?c\)

**proof**

- Fix \(f\)
- Assume \(f0: f \ 0 \in X\) and \(\text{chain: chain (relto (R \cup R') S) f}\)
- Then have \(\text{chain (relto R S \cup relto R' S) f}\) by auto
- From relative-ending \(\{\text{OF this}\ f0\ \text{SN}\}\)
- Have \(\exists j. \forall i \geq j. (f i, f (\text{Suc } i)) \in \text{relto R' S} - \text{relto R S}\) by auto
- Then obtain \(j\) where \(\forall i \geq j. (f i, f (\text{Suc } i)) \in \text{relto R' S}\) by auto
- Then have \(\text{chain (relto R' S) (shift f j)}\) by auto
- Moreover have \(f j \in X\)

**proof (induct j)**

- Case 0 from \(f0\) show \(?case\) by simp

**next**

- Case (Suc \(j\))
- Let \(?s = (f j, f (\text{Suc } j))\)
- From \(\text{chain}\) have \(?s \in \text{relto (R \cup R') S}\) by auto
- With \(\text{Image-closed-trancl[OF closure]}\) Suc show \(f (\text{Suc } j) \in X\) by blast
- Qed
- Then have \(\text{shift f j 0} \in X\) by auto
- Ultimately have \(\neg \text{SN-on (relto R' S)} X\) by (intro not-SN-onI)
- With \(SN'\) show \(\text{False}\) by auto
- Qed

**next**

- Assume \(SN: ?c\)
- Then show \(?b\) by (rule SN-on-subset1, auto)
- Moreover
- From \(SN\) have \(SN-on ((\text{relto (R \cup R') S})^+) X\) by (unfold SN-on-trancl-SN-on-conv)
- Then show \(?a\) by (rule SN-on-subset1) regexp
- Qed

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

- Using \(\text{SN-on-relto-Un[of \{\}\]}\) by simp

**end**

### 4 Strongly Normalizing Orders

**theory SN-Orders**

**imports** Abstract-Rewriting

**begin**

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

```plaintext
class non-strict-order = ord +
  assumes ge-refl: \( x \geq (x :: 'a) \)
  and ge-trans[trans]: \[ x \geq y; (y :: 'a) \geq z \] \( \Longrightarrow x \geq z \)
  and max-comm: max \( x \ y = max \ y \ x \)
  and max-ge-x[intro]: max \( x \ y \geq x \)
  and max-id[trans]: \[ x \geq y \] \( \Longrightarrow \) max \( x \ y = x \)
  and max-mono: \[ x \geq y \] \( \Longrightarrow \) max \( z \ x \geq max \ z \ y \)
begin
lemma max-ge-y[trans]: max \( x \ y \geq y \)
  unfolding max-comm[of \( x \ y \)] ..

lemma max-mono2: \( x \geq y \) \( \Longrightarrow \) 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: \( x \geq y \) \( \Longrightarrow \) \( x + z \geq y + z \)

lemma plus-right-mono: \( y \geq (z :: 'a :: ordered-ab-semigroup) \) \( \Longrightarrow \) \( x + y \geq x + z \)
  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: \( z \geq 0 \) \( \Longrightarrow \) \( x \geq y \) \( \Longrightarrow \) \( x \cdot z \geq y \cdot z \)
  and times-right-mono: \( x \geq 0 \) \( \Longrightarrow \) \( y \geq z \) \( \Longrightarrow \) \( x \cdot y \geq x \cdot z \)
  and times-left-anti-mono: \( x \geq y \) \( \Longrightarrow \) \( 0 \geq z \) \( \Longrightarrow \) \( y \cdot z \geq x \cdot z \)

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

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

locale order-pair =
  fixes gt :: 'a :: \{ non-strict-order,zero \} \( \Rightarrow \) 'a \( \Rightarrow \) bool (infix \( \succ \) 50)
  and default :: 'a
  assumes compat[trans]: \[ x \geq y; y \succ z \] \( \Longrightarrow x \succ z \)
  and compat2[trans]: \[ x \succ y; y \geq z \] \( \Longrightarrow x \succ z \)
  and gt-imp-ge: \( x \succ y \) \( \Longrightarrow \) \( x \geq y \)
  and default-ge-zero: default \( \geq 0 \)
begin
lemma gt-trans[trans]: \[ x \succ y; y \succ z \] \( \Longrightarrow x \succ z \)
  by (rule compat[OF gt-imp-ge])
end
```
```
locale one-mono-ordered-semiring-1 = order-pair gt
for gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool (infix 50) +
assumes plus-gt-left-mono: x ≻ y ⇒ x + z ≻ y + z
and default-gt-zero: default ≻ 0
begin
lemma plus-gt-right-mono: x ≻ y ⇒ a + x ≻ a + y
  unfolding add.commute[of a] by (rule plus-gt-left-mono)
lemma plus-gt-both-mono: x ≻ y =⇒ a ≻ b =⇒ 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 + order-pair
+ 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 ≻ u] =⇒ x * y ≻ x * z
locale both-mono-ordered-semiring-1 = order-pair gt
for gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool (infix 50) +
fixes arc-pos :: 'a ⇒ bool
assumes plus-gt-both-mono: [x ≻ y; z ≻ u] =⇒ x + z ≻ y + u
and times-gt-left-mono: x ≻ y =⇒ x * z ≻ y * z
and times-gt-right-mono: y ≻ z =⇒ 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: \( \forall c d. \ \text{arc-pos} \ d = \exists e. e \geq 0 \land \text{arc-pos} \ e \land \neg (c \geq d * e) \)
begin
lemma max0-id: max 0 (x :: 'a) = x
  unfolding max-comm[of 0]
  by (rule max-id[OF zero-leastIII])
end
locale SN-both-mono-ordered-semiring-1 = both-mono-ordered-semiring-1 +
assumes SN: SN {(x,y). arc-pos y ∧ x ≻ y}
locale weak-SN-strict-mono-ordered-semiring-1 =
fixes weak-gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool
and default :: 'a

111
locale weak-SN-both-mono-ordered-semiring-1 =
  fixes weak-gt :: 'a::ordered-semiring-1 ⇒ 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 −→
  gt x y)

class poly-carrier = ordered-semiring-1 + comm-semiring-1
locale poly-order-carrier =
  SN-one-mono-ordered-semiring-1 default gt
for default :: 'a::poly-carrier and gt (infix ≻ 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) 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 bn: b ^ n ≥ 0 by auto
  have bsn: b * Suc n ≥ 0 unfolding power-Suc
    using times-left-mono[OF bn b] 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 bn] 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

end
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

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 0
  show ?case using pow-mono-one[OF a] by auto
next
  case (Suc n)
  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] Suc], of 1]
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 \( \text{ge-trans[OF } b \text{ one-ge-zero] have } b0 : b \geq 0 . \)
from \( \text{times-left-mono[OF } b0 a] \text{ have } a \ast b \geq b \) by simp
from \( \text{ge-trans[OF this } b] \) show \( ?\text{thesis} \).
qed

lemma \( \text{sum-list-ge-mono: fixes } as :: ('}a :: \text{ordered-semiring-0}) \text{ list} \)
assumes \( \text{length as } = \text{ length bs} \)
and \( \bigwedge i. i < \text{ length bs } \Longrightarrow \text{ as } ! i \geq \text{ bs } ! i \)
shows \( \text{sum-list as } \geq \text{ sum-list bs} \)
using \( \text{assms} \)
proof (induct as arbitrary: \( bs \))
next
from \( \text{Cons(2) obtain } b \text{ bs where } bs : bs = b \# bs \text{ and } \text{len: length as } = \text{ length bs by (cases bs, auto)} \)
note \( \text{ge } = \text{ Cons(3)[unfolded bs]} \)
\begin{small}
\{ 
fix \( i \)
assume \( i < \text{ length bs} \)
hence \( \text{Suc } i < \text{ length (} b \# bs ) \) by simp
from \( \text{ge[OF this] have } \text{ as } ! i \geq \text{ bs } ! i \) by simp
\}
\end{small}
from \( \text{Cons(1)[OF len this] have IH: sum-list as } \geq \text{ sum-list bs} \).
from \( \text{ge[of 0] have } ab : a \geq b \) by simp
from \( \text{ge-trans[OF plus-left-mono[OF } ab \text{] plus-right-mono[OF IH]]} \)
show \( ?\text{case unfolding bs by simp} \)
qed

lemma \( \text{sum-list-ge-0-nth: fixes } xs :: ('}a :: \text{ordered-semiring-0}) \text{ list} \)
assumes \( \text{ge: } \bigwedge x. x \in \text{ set xs } \Longrightarrow x \geq 0 \)
shows \( \text{sum-list xs } \geq 0 \)
proof –
let \( ?l = \text{ replicate (length xs) (0 :: } '{a}) \)
have \( \text{length xs } = \text{ length } ?l \) by simp
from \( \text{sum-list-ge-mono[OF this] } \text{ge have } \text{sum-list xs } \geq \text{ sum-list } ?l \) by simp
also have \( \text{sum-list } ?l = 0 \) using \( \text{sum-list-0[of } ?l] \) by auto
finally show \( ?\text{thesis} \).
qed

lemma \( \text{sum-list-ge-0: fixes } xs :: ('}a :: \text{ordered-semiring-0}) \text{ list} \)
assumes \( \text{ge: } \bigwedge x. x \in \text{ set xs } \Longrightarrow x \geq 0 \)
shows \( \text{sum-list xs } \geq 0 \)
by (rule \( \text{sum-list-ge-0-nth, insert } \text{ge[unfolded set-conv-nth], auto}) \)
\end{document}
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 $\forall n \in \text{nat} \Rightarrow 'a$

from assms show ?thesis

proof
(induct m arbitrary: n)

case 0

show ?case by auto

next

case (Suc m)$

then obtain $n$ where $nn$:

$nn = \text{Suc } n$ by (cases nn, auto)

note $\text{Suc } = \text{Suc[unfolded } nn]\$

hence rec: $\forall n \geq ?n m$ 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 $\text{non-inf } r \equiv \forall a f. \exists i. (f i, f (\text{Suc } i)) \notin r \lor (f i, a) \notin r$

lemma non-infl[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 $\land i. (f i, f (\text{Suc } i)) \notin r \lor (f i, a) \notin r \Rightarrow P$

shows $P$

using assms unfolding non-inf-def by blast
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 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 Archemedian 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 (>) :: 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 ⇒ 'a ⇒ bool where
delta-gt δ ≡ (λ x y. x − y ≥ δ)
lemma non-inf-delta-gt: assumes delta: δ > 0
  shows 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 simp
    }
    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 ?i[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

117
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 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 yzd: 0 ≤ y − z by auto
          have 0 ≤ ?p
            by (rule mult-nonneg-nonneg[OF x1 yzd])
          have x * y − x * z = x * (y − z) using right-diff-distrib[of x y z] by auto
also have \((x - 1) + 1\) \(*\) \((y - z)\) by auto
also have \(\leq p + 1\) \(*\) \((y - z)\) by \(\text{rule ring-distrib(2)}\)
also have \(\leq (\theta + d)\) using \(\text{gcd (\(\theta \leq p\))}\) by auto
finally
show \(d \leq x \ast y - x \ast z\) by auto
qed

\text{qed (insert dpos, auto simp: delta-gt-def default defz)}

\text{qed}

\text{lemma delta-poly: assumes dpos: } \delta > 0 \text{ and default: } \delta \leq \text{def}
\text{shows poly-order-carrier def } (\text{delta-gt } \delta) (1 \leq \delta) \text{ False}
proof –
from delta-interpretation[\(\text{OF dpos default}\)]
interpret SN-strict-mono-ordered-semiring-1 def delta-gt \delta \text{ delta-mono .}
interpret poly-order-carrier def delta-gt \delta \text{ False False}
proof(\text{unfold-locales})
fix \(y\) \(z\) \(x\) :: \('a\)
assume gt: delta-gt \delta \(y\) \(z\) and ge: \(x \geq 1\)
from ge have ge: \(x \geq 0\) and m: delta-mono \(x\) unfolding delta-mono-def by auto
show delta-gt \delta \((y \ast x)\) \((z \ast x)\)
using mono[\(\text{OF m gt ge}\)] by (auto simp: field-simps)
next
fix \(x\) \(y\) :: \('a\) and \(n\) :: nat
assume False thus delta-gt \delta \((x \ast n)\) \((y \ast n)\) ..
next
fix \(x\) \(y\) :: \('a\)
assume False
thus \(\exists k.\) \(x = ((+) 1 \ast^k y)\) by simp
qed
show \(\theta\)theory
proof(\text{unfold-locales})
fix \(x\) \(y\) :: \('a\) and \(n\) :: nat
assume one: \(1 \leq \delta\) and gt: delta-gt \delta \(x\) \(y\) and y: \(y \geq 0\) and n: \(1 \leq n\)
then obtain p where n: \(n = \text{Suc p}\) and x: \(x \geq 1\) and y2: \(0 \leq y\) and xy: \(x \geq y\) by (cases n, auto simp: delta-gt-def)
show delta-gt \delta \((x \ast n)\) \((y \ast n)\)
proof (simp only: n, induct p, simp add: gt)
case (Suc p)
from times-gt-mono[\(\text{OF this x}\)]
have one: delta-gt \delta \((x \ast \text{Suc (Suc p)})\) \((x \ast y \ast \text{Suc p})\) by (auto simp: field-simps)
also have \(\geq y \ast y \ast \text{Suc p}\)
by (rule times-left-mono[\(\text{OF - xy}\)], auto simp: zero-le-power[\(\text{OF y2, of Suc p, simplified}\)])
finally show \(\theta\)case by auto
qed
next
fix $x\ y :: 'a$
assume $\exists k. x = (+) (1 ^^ k) y$ by simp
qed (rule times-gt-mono, auto)

lemma delta-minimal-delta: assumes $\bigwedge x\ y. (x,y) \in \text{set xys} \implies x > y$
shows $\exists \delta > 0. \forall x\ y. (x,y) \in \text{set xys} \implies \delta\gt x\ y$
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 $d1\pos: d1 > 0$ and $d1 \leq x - y$ by auto
hence $xy\ : \delta\gt d1$ unfolding $\delta\gt-def$ by auto
from Cons obtain $d2$ where $d2\pos: d2 > 0$ and $xys\ : \delta\\gt d x\ y$ by auto
obtain $d$ where $d = \min d1\ d2$ by auto
with $d\pos\ d2\pos\ xy$ have $dpos: d > 0$ and $\delta\\gt d x\ y$ unfolding $\delta\gt-def$ by auto
with $xys\ d\ Pair\ have\ \forall x\ y. (x,y) \in \text{set (xy \# xys)} \implies \delta\\gt d x\ y$ unfolding $\delta\gt-def$ by force
with $dpos$ show $?thesis$ by auto
qed

interpretation weak-delta-SN: weak-SN-strict-mono-ordered-semiring-1 ($>$) 1 delta-mono
proof
fix $xysp :: (\prime a \times 'a)\ list$
assume orient: $\forall x\ y. (x,y) \in \text{set xysp} \implies x > y$
obtain $xys\ where\ xsy: xys = (1,0) \# xysp$ by auto
with orient have $\bigwedge x\ y. (x,y) \in \text{set xys} \implies x > y$ by auto
with delta-minimal-delta have $\exists \delta > 0. \forall x\ y. (x,y) \in \text{set xys} \implies \delta\\gt x\ y$ by auto
then obtain $\delta$ where $dpos: \delta > 0$ and orient: $\bigwedge x\ y. (x,y) \in \text{set xys} \implies \delta\\gt x\ y$ by auto
from orient have orient1: $\forall x\ y. (x,y) \in \text{set xysp} \implies \delta\\gt x\ y$ and orient2: $\delta\\gt 1$ 0 unfolding $\times y$ by auto
from orient2 have oned: $\delta \leq 1$ unfolding $\delta\gt-def$ by auto
show $\exists gt. SN-strict-mono-ordered-semiring-1\ 1\ gt\ \delta\\gt x\ y$ unfolding $\gt\ x\ y$
  by (intro exI conjI, rule delta-interpretation[OF $dpos\ oned$, rule orient1]}

120
5.3 The standard semiring over the integers

**definition** int-mono :: int ⇒ bool where int-mono x ≡ x ≥ 1

**instantiation** int :: large-ordered-semiring-1
begin
instance
proof
fix y :: int
show ∃ x. of-nat x ≥ y
by (rule exI[of - nat y], simp)
qed (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg)
end

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

**interpretation** int-SN: SN-strict-mono-ordered-semiring-1 1 (> ) :: int ⇒ int ⇒ bool int-mono
proof (unfold-locales)
have [simp]: ∀ x :: int. (−1 < x) = (0 ≤ x) by auto
show SN {(x,y). y ≥ 0 ∧ (y :: int) < x}
using non-inf-imp-SN-bound[OF non-inf-int-gt, of −1] by auto
qed (auto simp: mult-strict-left-mono int-mono-def)

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

5.4 The arctic semiring over the integers

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

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

**instantiation** arctic :: ord
fun less-eq-arctic :: arctic ⇒ arctic ⇒ bool where
  less-eq-arctic MinInfty x = True
| less-eq-arctic (Num-arc -) MinInfty = False
| less-eq-arctic (Num-arc y) (Num-arc x) = (y ≤ x)

fun less-arctic :: arctic ⇒ arctic ⇒ bool where
  less-arctic MinInfty x = True
| less-arctic (Num-arc -) MinInfty = False
| less-arctic (Num-arc y) (Num-arc x) = (y < x)

instance ..
end

instantiation arctic :: ordered-semiring-1 begin
fun plus-arctic :: arctic ⇒ arctic ⇒ arctic where
  plus-arctic MinInfty y = y
| plus-arctic x MinInfty = x
| plus-arctic (Num-arc x) (Num-arc y) = (Num-arc (max x y))

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

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

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

instance proof
fix x y z :: arctic
show x + y = y + x
  by (cases x, cases y, auto, cases y, auto)
show (x + y) + z = x + (y + z)
  by (cases x, auto, cases y, auto, cases z, auto)
show (x * y) * z = x * (y * z)
  by (cases x, auto, cases y, auto, cases z, auto)
show x * 0 = 0
  by (cases x, auto simp: zero-arctic-def)
show x * (y + z) = x * y + x * z
  by (cases x, auto, cases y, auto, cases z, auto)
show (x + y) * z = x * z + y * z
  by (cases x, auto, cases y, cases z, auto, cases z, auto)
show 1 * x = x
  by (cases x, simp-all add: one-arctic-def)
show \( x \cdot 1 = x \)
  by (cases \( x \), simp-all add: one-arctic-def)
show \( 0 + x = x \)
  by (simp add: zero-arctic-def)
show \( 0 \cdot x = 0 \)
  by (simp add: zero-arctic-def)
show \((\theta :: \text{arctic}) \neq 1\)
  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 \( \text{ge} : x \geq y \)
from \( \text{ge} \) show \( x + z \geq y + z \)
  by (cases \( x \), cases \( y \), cases \( z \), auto, cases \( y \), auto, cases \( z \), auto)
from \( \text{ge} \) show \( x \cdot z \geq y \cdot z \)
  by (cases \( x \), cases \( y \), cases \( z \), auto, cases \( y \), auto, cases \( z \), auto)
from \( \text{ge} \) show \( \max x y = x \) unfolding max-def
  by (cases \( x \), (cases \( y \), auto)+)
from \( \text{ge} \) 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 \( y \), auto, cases \( y \), cases \( z \), auto, cases \( z \), auto)

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

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

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

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

interpretation arctic-SN: SN-both-mono-ordered-semiring-1 1 (>)
proof
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 and y ≥ z
thus x > z
  by (cases z, simp, cases y, simp, cases x, 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 u, auto, cases z, auto, 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 * 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 (cases \(x\), simp, cases \(y\), auto)

next

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

proof = {

  fix \(x\)

  assume \(\exists f.\ f\ 0 = x \land (\forall i.\ (f\ i, f\ (Suc\ i)) \in \ ?rel)\)

  from this obtain \(f\) where \(f\ 0 = x\) and \(\forall i.\ (f\ i, f\ (Suc\ i)) \in \ ?rel\)

  by auto

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

  let \(g = \lambda i.\ get-arctic-num\ (f\ i)\)

  have \(\forall i.\ ?g\ (Suc\ i) \geq 0\ \land \ ?g\ i > ?g\ (Suc\ i)\)

  proof

    fix \(i\)

    from steps have \(i: f\ i > f\ (Suc\ i)\) \land pos-arctic\ (f\ (Suc\ i)) \by auto

    from \(i\) obtain \(n\) where \(f\ i = \text{Num-arc}\ n\)

    by (cases \(f\ (Suc\ i)\), simp, cases \(f\ i\), auto)

  qed

  with \(int-SN\).\ \(SN\) have False unfolding \(SN\)-defs \by auto

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

qed

next

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

assume \(y > z\)

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

by (cases \(x\), simp, cases \(z\), simp, cases \(y\), auto)

next

fix \(c\ d\)

assume pos-arctic \(d\)

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

by (cases \(d\), auto)

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

proof (cases \(c\))

  case MinInfty

  show \(?thesis\)

  by (rule exI[of - Num-arc 0],

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

next

  case (Num-arc \(m\))

125
show \( ?\)thesis
by (rule exI[of - Num-arc (abs m + 1)], insert n, unfold d Num-arc zero-arctic-def, simp)
qed

5.5 The arctic semiring over an arbitrary archimedean field
completely analogous to the integers, where one has to use delta-orderings

datatype \( 'a \) arctic-delta = MinInfty-delta | Num-arc-delta \( 'a \)

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

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

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

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

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

instance
proof
  fix x y z :: 'a arctic-delta
show \(x + y = y + x\)
  by (cases x, cases y, auto, cases y, auto)
show \((x + y) + z = x + (y + z)\)
  by (cases x, auto, cases y, auto, cases z, auto)
show \((x * y) * z = x * (y * z)\)
  by (cases x, auto, cases y, auto, cases z, auto)
show \(x * 0 = 0\)
  by (cases x, auto simp: zero-arctic-delta-def)
show \(x * (y + z) = x * y + x * z\)
  by (cases x, auto, cases y, auto, cases z, auto)
show \((x + y) * z = x * z + y * z\)
  by (cases x, auto, cases y, auto, cases z, auto)
show \(1 * x = x\)
  by (cases x, simp-all add: one-arctic-delta-def)
show \(x * 1 = x\)
  by (cases x, simp-all add: one-arctic-delta-def)
show \(0 + x = x\)
  by (simp add: zero-arctic-delta-def)
show \(0 * x = 0\)
  by (simp add: zero-arctic-delta-def)
show \((0 :: 'a arctic-delta) \neq 1\)
  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 ge: \(x \geq y\)
from ge show \(x + z \geq y + z\)
  by (cases x, cases y, cases z, auto, cases y, auto, cases z, auto)
from ge show \(x * z \geq y * z\)
  by (cases x, cases y, cases z, auto, cases y, auto, cases z, auto)
from ge show \(\max x y = x\) unfolding max-def
  by (cases x, (cases y, auto)+)
from ge show \(\max x z \geq \max z y\) 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, auto, cases z, auto)
next
fix \(x y z :: 'a arctic-delta\)
assume \(y \geq z\)
thus \(x \geq y\) \(\geq x * z\)
by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)

next
fix x y z :: 'a arctic-delta
show x ≥ y ⇒ 0 ≥ z ⇒ y * z ≥ x * z
  by (cases z, cases x, auto simp: zero-arctic-delta-def)
qed
end

x ⪯d y is interpreted as y = -inf or (x,y #= -inf and x ⪯d y)

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

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

fun pos-arctic-delta :: ('a :: floor-ceiling) arctic-delta ⇒ bool
where pos-arctic-delta MinInfty-delta = False |
  pos-arctic-delta (Num-arc-delta n) = (0 ≤ n)

lemma arctic-delta-interpretation: assumes dpos: δ > 0 shows SN-both-mono-ordered-semiring-1 1 (gt-arctic-delta δ) pos-arctic-delta
proof –
  from delta-interpretation[OF dpos] interpret SN-strict-mono-ordered-semiring-1 δ delta-gt δ delta-mono by simp
  show ?thesis
  proof
    fix x y z :: 'a arctic-delta
    assume x ≥ y and gt-arctic-delta δ y z
    thus gt-arctic-delta δ x z
      by (cases z, simp, cases y, simp, cases x, simp, simp add: compat)
  next
    fix x y z :: 'a arctic-delta
    assume gt-arctic-delta δ x y and y ≥ z
    thus gt-arctic-delta δ x z
      by (cases z, simp, cases y, simp, cases x, simp, simp add: compat2)
  next
    fix x y :: 'a arctic-delta
    assume gt-arctic-delta δ x y
    thus x ≥ y
      by (cases x, insert dpos, (cases y, auto simp: delta-gt-def)+)
  next
    fix x y z u
    assume gt-arctic-delta δ x y and gt-arctic-delta δ z u
    thus gt-arctic-delta δ (x + z) (y + u)
      by (cases y, cases u, simp, cases z, simp, cases x, simp, simp add: delta-gt-def)
next
  fix \( x \) \( y \) \( z \)
  assume \( \text{gt-arctic-delta} \ \delta \ x \ y \)
  thus \( \text{gt-arctic-delta} \ (x * z) (y * z) \)
    by (cases \( y \), simp, cases \( z \), simp, cases \( x \), simp, simp add: plus-gt-left-mono)

next
  fix \( x \)
  assume \( \text{gt-arctic-delta} \ \delta \ 0 \ x \)
  thus \( x = 0 \)
    by (cases \( x \), auto simp: zero-arctic-delta-def)

next
  fix \( x \) \( y \)
    with \( x \)
      show \( \text{pos-arctic-delta} \ 1 \)
       unfolding one-arctic-delta-def by simp
    next
      show \( \text{pos-arctic-delta} \ x \)
       unfolding zero-arctic-delta-def by simp
    next
      show \( (1 :: 'a \text{ arctic-delta}) \geq 0 \)
       unfolding zero-arctic-delta-def by simp
      show \( \neg \text{pos-arctic-delta} \ 0 \)
       unfolding zero-arctic-delta-def by simp

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

next
  fix \( x \) \( y \) :: 'a \text{ arctic-delta}
  assume \( \text{pos-arctic-delta} \ x \) and \( \text{pos-arctic-delta} \ y \)
  thus \( \text{pos-arctic-delta} \ (x * y) \)
    by (cases \( x \), simp, cases \( y \), auto)

next
  show \( \text{SN} \ \{(x,y), \text{pos-arctic-delta} \ y \land \text{gt-arctic-delta} \ \delta \ x \ y\} \) (is \( \text{SN} \ ?rel)\)
  proof -
  fix \( x \)
    assume \( \exists \ f . \ f 0 = x \land (\forall \ i . (f \ i, f \ (\text{Suc} \ i))) \in \ ?rel)\)
    from this obtain \( f \) where \( f 0 = x \) and \( \text{seq} \ (\forall \ i . (f \ i, f \ (\text{Suc} \ i))) \in \ ?rel)\)
      by auto
    from \( \text{seq} \) have \( \text{steps} : (\forall \ i . \text{gt-arctic-delta} \ \delta \ (f \ i) (f \ (\text{Suc} \ i)) \land \text{pos-arctic-delta} \)
    (f (\text{Suc} \ i))\) by auto
      let \( ?g = \lambda \ i . \text{get-arctic-delta-num} \ f \ i\)
    have \( (\forall \ i . ?g \ (\text{Suc} \ i)) \geq 0 \land \text{delta-gt} \ \delta \ (?g \ i) (?g \ (\text{Suc} \ i))\) by auto
      fix \( i \)
        from \( \text{steps} \) have \( \text{i: gt-arctic-delta} \ \delta \ (f \ i) (f \ (\text{Suc} \ i)) \land \text{pos-arctic-delta} \ (f \ (\text{Suc} \ i))\) by auto
          from \( \text{i} \) obtain \( n \) where \( fi: f \ i = \text{Num-arc-delta} \ n \) by (cases \( f \ (\text{Suc} \ i)\), simp, cases \( f \ i\), auto)
            from \( \text{i} \) obtain \( m \) where \( fsi: f \ (\text{Suc} \ i) = \text{Num-arc-delta} \ m \) by (cases \( f \ (\text{Suc} \ i)\), auto)
              with \( i \) have \( gz: 0 \leq m \) by simp
              from \( \text{i fi fsi} \) have \( \text{delta-gt} \ \delta \ n \ m \) by auto
                with \( fi fsi \) gz

129
show \( g (Suc i) \geq 0 \land \delta (g i) (g (Suc i)) \) by auto
qed
from this obtain \( g \) where \( \forall i. g (Suc i) \geq 0 \land \delta (g i) (g (Suc i)) \)
by auto
hence \( \exists f. f 0 = g 0 \land (\forall i. (f i, f (Suc i)) \in \{(x,y). y \geq 0 \land \delta x y}) \)
by auto
with SN have False unfolding SN-defs by auto
}
thus \?thesis unfolding SN-defs by auto
qed
next
fix \( c d :: \text{'}a \text{ arctic-delta} \)
assume pos-arctic-delta \( d \)
then obtain \( n \) where \( d = Num-arctic-delta n \) and \( n: 0 \leq n \)
by (cases \( d \), auto)
show \( \exists e. e \geq 0 \land pos-arctic-delta e \land \neg c \geq d * e \)
proof (cases \( c \))
case MinInfty-delta
show \?thesis
by (rule exI[of - Num-arctic-def 0], unfold d MinInfty-delta zero-arctic-def, simp)
next
case (Num-arctic-delta \( m \))
show \?thesis
by (rule exI[of - Num-arctic-delta (abs m + 1)], insert \( n \), unfold d Num-arctic-delta zero-arctic-def, simp)
qed
next
fix \( x y z \)
assume gt: \( \text{gt-arctic-delta} \delta y z \)
{
fix \( x y z \)
assume gt: \( \text{delta-gt} \delta y z \)
have \( \text{delta-gt} (x + y) (x + z) \)
using plus-gt-left-mono[\text{OF 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 :: \( \text{'}a :: \text{floor-ceiling} \) \text{arctic-delta} \rightarrow \text{'}a \text{ arctic-delta} \rightarrow bool
where weak-gt-arctic-delta - MinInfty-delta = True
| weak-gt-arctic-delta MinInfty-delta (Num-arctic-delta -) = False
| weak-gt-arctic-delta (Num-arctic-delta \( x \)) (Num-arctic-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)) (filter (λ (ax,ay). ax ≠ MinInfty-delta ∧ ay ≠ MinInfty-delta) xys)
(is - = map ?f -)
by auto
have ∀ x y. (x,y) ∈ set xysp → x > y
proof (intro allI impI)
  fix x y
  assume (x,y) ∈ set xysp
  with xysp obtain az ay where (az,ay) ∈ set xys and ax ≠ MinInfty-delta and ay ≠ MinInfty-delta and (x,y) = ?f (ax,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: ∀ x y. (x,y) ∈ set xysp → delta-gt δ x y by auto
have orient: ∀ x y. (x,y) ∈ set xys → gt-arctic-delta δ x y
proof (intro allI impI)
  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 δ ax ay
  proof (cases ay)
    case (Num-arc-delta y) note ay = this
    show ?thesis
    proof (cases ax)
      case MinInfty-delta
      with ay orient show ?thesis by auto
    next
      case (Num-arc-delta x) note ax = this
      from ax ay axay have (x,y) ∈ set xys unfolding xysp by force
      from ax ay orient2[of this] show ?thesis by simp
    qed
  qed
  qed
  qed
  show ∃ gt. SN-both-mono-ordered-semiring-1 1 gt pos-arctic-delta ∧ (∀ x y. (x, y) ∈ set xys → gt x y)
  by (intro exI conjI, rule arctic-delta-interpretation[of dpos], rule orient)
qed
end
References

