Abstract Rewriting
Christian Sternagel and René Thiemann
September 13, 2023

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-informatikuibk.ac.at/software/ceta
5 Carrier 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 nat ⇒ 'a.

type-synonym 'a seq = nat ⇒ 'a

1.1 Operations on Infinite Sequences

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

abbreviation (input) chainp :: ('a ⇒ 'a ⇒ bool) ⇒ 'a seq ⇒ bool where
chainp P S ≡ ∀i. P (S i) (S (Suc i))

Special version for relations.

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

Extending a chain at the front.

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

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

Special version for relations.

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

A chain admits arbitrary transitive steps.

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

lemma chain-imp-relpow:
assumes chain r S shows (S i, S (i + j)) ∈ r^j
proof (induct i + j arbitrary: j)
case (Suc n) thus ?case using assms by (cases j) auto
qed simp

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

A chain admits arbitrary reflexive and transitive steps.

lemma chain-imp-rtrancl:
assumes chain r S and i ≤ j shows (S i, S j) ∈ r^*
proof from less-imp-Suc-add[OF assms(2)] obtain n where j = i + Suc n by auto
with chain-imp-relpow[of P S, OF assms(1), of i Suc n]
  show ?thesis unfolding trancl-power by force
qed

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

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

lemma stepfun-imp-chainp':
assumes ∀i≥n::nat. f i ≥ i ∧ P (S i) (S (f i))
sows chainp P (λi. S ((f ^^ i) n)) (is chainp P ?T)
proof
fix i
from assms have \((f \sim i)\) \(n \geq n\) by (induct i) auto
with assms[THEN spec[of -(f \sim i) n]]
show \(P \ (\ ?T i) \ ?T (\ Suc i)\) by simp
qed

lemma stepfun-imp-chainp:
assumes \(\forall i \geq n::nat. \ f \ i \ > \ i \ \land \ P \ (\ S \ i) \ (S \ (f \ i))\)
shows \(\text{chainp} \ P \ (\lambda i. \ S \ ((f \sim i) \ n))\) \(\text{is chainp} \ P ?T\)
using stepfun-imp-chainp[of n f S] and assms by force

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

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

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

lemma steps-imp-chainp:
assumes \(\forall i \geq n::nat. \ \exists j > i. \ P \ (S \ i) \ (S \ j)\)
shows \(\exists T. \ \text{chainp} \ P \ T\)
using steps-imp-chainp'[of n f 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 \Rightarrow bool
assumes infinite: INFM \ j. p j

begin

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

fun index :: nat seq where
index 0 = (LEAST n. p n)
| index (Suc n) = (LEAST k. p k \land k > index n)

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

lemma index-ordered: index n < index (Suc n)
proof
from inf obtain k where k \geq Suc (index n) \land p k by auto
with LeastI[of \\lambda k. p k \land 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 \neg p i
proof
from not-less-Least[of i2[simplified]] i1 show ?thesis by auto
qed

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

lemma index-surj:
assumes $k \geq \text{index } l$
shows $\exists i. j. \ k = \text{index } i + j \land \text{index } i + j < \text{index } (\text{Suc } i)$
proof –
from assms have $k = \text{index } l + (k - \text{index } l)$ by auto
then obtain $u$ where $k; \ k = \text{index } l + u$ 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 $i: \ i < \text{index } 0$ shows $\neg p \ i$
proof –
from $i[/simplified index.simps]$ have $i < \text{Least } p$.
from not-less-Least[of this] show ?thesis.
qed

end
1.3 Assembling Infinite Words from Finite Words

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

fun inf-concat-simple :: (nat ⇒ 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 ?thesis 
    proof (cases f i) 
    case 0 
    show ?thesis 
      by (rule exI[of - Suc k], simp add: ck 0)
next 
    case (Suc n) 
    hence 0 + n < f i by auto 
    from inf-concat-simple-add[OF ck, OF this] Suc 
    show ?thesis 
      by (intro exI[of - k + Suc n], auto)
qed
qed

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

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

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

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

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

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

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

proof (intro allI impI, elim conjE)

fix \(j\)

assume \(j < \text{Suc} \, l \) and \(n j \geq 0 < n j\)

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

proof (cases \(j < l\))

case True

from \(\text{Suc}[\text{THEN} \, \text{spec[of - j]] \, True \, nj \, show \, \text{thesis} \, by \, auto}\)

next

case False

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

show \(\text{thesis}\)

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

case False

have \(l: (\text{LEAST} \, i. \, 0 < n \, i) = l\)

proof (rule Least-equality, rule \(nj[\text{unfolded} \, j]\))

fix \(l'\)

assume \(0 < n \, l'\)

with False have \(\neg \, l' < l\) by auto

thus \(l \leq l'\) by auto

qed

show \(\text{thesis}\)

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

next

case True

then obtain \(lll\) where \(lll: lll < l \) and \(nlll: 0 < n \, lll\) by auto

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

from \(ll\) have \(lll: ll = l - (ll - lll)\) by auto

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

have \(nl': 0 < n \, (ll - ?l')\)

proof (rule LeastI)

show \(0 < n \, (ll - (ll - lll))\) using \(lll \, nlll\) by auto

qed

with \(\text{Suc}[\text{THEN} \, \text{spec[of - lll - ?l']}] \, obtain \, k \, where \, k:\)

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

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

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

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

have \(ll: \quad l = l\) unfolding \(l\)

proof (rule Least-equality)

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

next

fix \(l'\)

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

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

proof (rule ccontr)

assume not: \(\neg \, \text{thesis}\)

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

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

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

thus \?thesis by arith
qed

lemma inf-concat-Suc:
assumes inf: INFM $i$. $n i > 0$
and $f$: $\land i. f i (n i) = f (Suc i) \theta$
and resk: inf-concat $n k = (i', j')$
shows $f i' j' = f i (Suc j)$

proof –
note bounds = inf-concat-bounds[OF inf]
from bounds[OF resk] have $j$: $j < n i$.
show \?thesis
proof (cases Suc $j < n i$)
case True
with ressk
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[unfolded INFM-nat] obtain $k$ where $?p k$ by auto
from LeastI[of $?p$, OF this] have $?p ?i'$.
hence $?i' = \text{Suc } i + (\text{Suc } i - \text{Suc } i)$ by simp
then obtain $d$ where $?i'$: $?i' = \text{Suc } i + d$ by auto
from not-less-Least[of - $?p$, unfolded $?i'$] have $d'$: $\land d'. d' < d \Rightarrow n (\text{Suc } i + d') = 0$ by auto

12
have \( f(Suc\,i)\,0 = f\,i'\,0 \) unfolding \( ii' \) using \( d' \)
proof (induct \( d \))
case 0
  show \( ?\text{case} \) by simp
next
case \( (Suc\,d) \)
hence \( f(Suc\,i)\,0 = f(Suc\,(i + d))\,0 \) by auto
also have ... = \( f(Suc\,(Suc\,(i + d)))\,0 \)
  unfolding \( f[symmetric] \)
  using Suc(2)[of \( d \)] by simp
finally show \( ?\text{case} \) by simp
qed
thus \( ?\text{thesis} \) unfolding \( i'\,j'\,j\,f \) by simp
qed
qed

2 Abstract Rewrite Systems

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

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

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

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

2.1 Definitions

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

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

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

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

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

**Abbreviation** symcl :: \( 'a rel \Rightarrow 'a rel \) where  
\[ A^{**} = A \cup A^{-1} \]

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

**Definition**  
conversion :: \( 'a rel \Rightarrow 'a rel \) where  
\[ A^{**} = (A^{**})^* \]

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

**Definition** NF :: \( 'a rel \Rightarrow 'a set \) where  
\[ NF A = \{ a. A \quad {'}\{ a \}\} = \{\} \]

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

**Notation** (ASCII)  
symcl (\((-\leftrightarrow-)\) [1000] 999) and  
conversion (\((-\leftrightarrow\ast\ast)\) [1000] 999) and  
normalizability (\((-\uparrow)\) [1000] 999)

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

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

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

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

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

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

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

14
using assms by (auto simp: join-def rtrancl-converse)

lemma joinD:
\[(a, b) \in A^+ \Rightarrow \exists c. (a, c) \in A^* \land (b, c) \in A^*\]
by (blast elim: joinE)

lemma meetI:
\[(a, b) \in A^* \Rightarrow (a, c) \in A^* \rightarrow (b, c) \in A^+\]
by (auto simp: meet-def rtrancl-converse)

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

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

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

lemma conversion-refl [simp]: \((a, a) \in A^{***}\)
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:
\(r^* O r = r^+\) by regexp

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

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

Later declarations are tried first for ‘proof’ and ‘rule,’ then have the “main” introduction / elimination rules for constants should be declared last.
declare joinI-left [intro]
declare joinI-right [intro]
declare joinI [intro]
declare joinD [dest]
declare joinE [elim]

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

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

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

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

lemma conversion-inv:
(x, y) ∈ R*** ↔ (y, x) ∈ R***
by (auto simp: conversion-def)
  (metis (full-types) rtrancl-converseD symcl-converse)
lemma conversion-converse [simp]:
\[(A^{**})^{-1} = A^{**}\]
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 \((a, b) \in A^{*} \text{ and } (b, c) \in A\downarrow\)
sows \((a, c) \in A\downarrow\)
proof -
from \((b, c) \in A\downarrow\) obtain \(b'\) where \((b, b') \in A^{*} \text{ and } (b', c) \in (A^{-1})^{*}\)
unfolding join-def by blast
with \((a, b) \in A^{*}\) have \((a, b') \in A^{*}\) by simp
with \((b', c) \in (A^{-1})^{*}\) show \(?thesis\) unfolding join-def by blast
qed

lemma join-rtrancl-join:
assumes \((a, b) \in A\downarrow \text{ and } (c, b) \in A^{*}\)
sows \((a, c) \in A\downarrow\)
proof -
from \((c, b) \in A^{*}\) have \((b, c) \in (A^{-1})^{*}\)
unfolding join-def by simp
from \((a, b) \in A^{*}\) obtain \(a'\) where \((a, a') \in A^{*} \text{ and } (a', b) \in (A^{-1})^{*}\)
unfolding join-def by best
with \((b, c) \in (A^{-1})^{*}\) have \((a', c) \in (A^{-1})^{*}\) by simp
with \((a', a) \in A^{*}\) 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: \(a \in NF A \implies ((a, b) \notin A \implies P) \implies P\) by (auto simp: NF-def)

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

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

lemma NF-anti-mono:
assumes \(A \subseteq B\) shows \(NF B \subseteq NF A\)
using assms by auto

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

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

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

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

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

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

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

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

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

2.2 Properties of ARSs

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

definition CR-on :: \( 'a \) rel \( \Rightarrow \) 'a set \( \Rightarrow \) bool
where
\( CR-on\ r\ A \leftrightarrow (\forall a \in A. \forall b. c. (a, b) \in r^* \land (a, c) \in r^* \rightarrow (b, c) \in join\ r)\)

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

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

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

\( SN\) is defined as \( SN-on\) restricted to \( UNIV\).

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

definition UNF-on :: \( 'a \) rel \( \Rightarrow \) 'a set \( \Rightarrow \) bool
where
\( UNF-on\ r\ A \leftrightarrow (\forall a \in A. \forall b. c. (a, b) \in r^! \land (a, c) \in r^! \rightarrow b = c)\)

18
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

19
\( r A \)

by (simp add: CR-defs)

**lemma** CR-on-singletonI:
\[
\left( \forall b. c. \, (a, b) \in r^* \Rightarrow (a, c) \in r^* \Rightarrow (b, c) \in \text{join } r \right) \Rightarrow \text{CR-on } r \{a\}
\]

by (simp add: CR-defs)

**lemma** CR-onE:
\[
\text{CR-on } r A \Rightarrow a \in A \Rightarrow ((b, c) \in \text{join } r \Rightarrow P) \Rightarrow ((a, b) \notin r^* \Rightarrow P) \Rightarrow ((a, c) \notin r^* \Rightarrow P) \Rightarrow P
\]

unfolding CR-defs by blast

**lemma** CR-onD:
\[
\text{CR-on } r A \Rightarrow a \in A \Rightarrow (a, b) \in r^* \Rightarrow (a, c) \in r^* \Rightarrow (b, c) \in \text{join } r
\]

by (blast elim: CR-onE)

**lemma** semi-complete-onI:
\[
\text{WN-on } r A \Rightarrow \text{CR-on } r A \Rightarrow \text{semi-complete-on } r A
\]

by (simp add: semi-complete-defs)

**lemma** semi-complete-onE:
\[
\text{semi-complete-on } r A \Rightarrow (\text{WN-on } r A \Rightarrow \text{CR-on } r A \Rightarrow P) \Rightarrow P
\]

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:
\[
\left( \forall a. b. \, a \in \text{NF } A \Rightarrow b \in \text{NF } A \Rightarrow (a, b) \in A^{**} \Rightarrow a = b \right) \Rightarrow \text{UNC } A
\]

by (simp add: UNC-def)

**lemma** UNC-E:
\[
\begin{align*}
\text{UNC } A; \, a = b \Rightarrow P; \, a \notin \text{NF } A \Rightarrow P; \, b \notin \text{NF } A \Rightarrow P; \, (a, b) \notin A^{**} \Rightarrow P
\end{align*}
\]

unfolding UNC-def by blast

**lemma** UNF-onI:
\[
\left( \forall a. b. c. \, a \in A \Rightarrow (a, b) \in r^i \Rightarrow (a, c) \in r^i \Rightarrow b = c \right) \Rightarrow \text{UNF-on } r A
\]

by (simp add: UNF-defs)

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

unfolding UNF-on-def by blast

lemma \( UNF-onD \):
\[ UNF-on \ r \ A \Rightarrow a \in A \Rightarrow (a, b) \in r \Rightarrow (a, c) \in r^l \Rightarrow b = c \]

by (blast elim: UNF-onE)

declare \( UNF-onI \) [intro]
declare \( UNF-onD \) [dest]
declare \( UNF-onE \) [elim]

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

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

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

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

lemma \( not-SN-onE \):
assumes \( \neg SN-on \ r \ A \)
  and \( \forall f. [f \ 0 \in A ; \ chain \ r \ f] \Rightarrow P \)
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: \( (x, x) \in R \Rightarrow \neg SN \ R \)
unfolding \( SN-defs \) by force
lemma SN-on-irrefl:
  assumes SN-on r A
  shows \( \forall a \in A. \ (a, a) \notin r \)
proof (intro ballI notI)
  fix a assume a \in A and (a, a) \in r
  with assms show False unfolding SN-defs by auto
qed

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

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

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

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

lemma WN-onI:
  \( (\forall a \in A \Rightarrow \exists b. \ (a, b) \in r') \Rightarrow \text{WN-on } r A \)
  by (auto simp: WN-defs)

lemma WN-onE:
  \( \text{WN-on } r A \Rightarrow a \in A \Rightarrow (\forall b. \ (a, b) \in r' \Rightarrow P) \Rightarrow P \)
  unfolding WN-defs by blast
lemma WN-onD: WN-on r A \iff a \in A \implies \exists b. (a, b) \in r'
by (blast elim: WN-onE)

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

Restricting a relation \( r \) to those elements that are strongly normalizing with respect to a relation \( s \).
definition restrict-SN :: 'a rel \Rightarrow 'a rel \Rightarrow 'a rel where
 restrict-SN r s = \{ (a, b) | a b. (a, b) \in r \land SN-on s \{a\} \}

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

lemma SN-on-Image:
assumes SN-on r A
shows SN-on r (r '' A)
proof
fix f
assume f 0 \in r '' A and chain: chain r f
then obtain a where a \in A and 1: (a, f 0) \in r by auto
let ?g = case-nat a f
from cons-chain [OF 1 chain] have chain r ?g .
moreover have ?g 0 \in A by (simp add: \{a \in A\})
ultimately have \neg SN-on r A unfolding SN-defs by best
with assms show False by simp
qed

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

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

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

lemma relpow-seq:
assumes \((x, y) \in r \rightleftharpoons n\)
shows \(\exists f. f 0 = x \land f n = y \land (\forall i < n. (f i, f (Suc i)) \in r)\)
using assms
proof (induct \(n\) arbitrary: \(y\))
case 0 then show \(\text{?case by auto}\)
next
case (Suc \(n\))
then obtain \(z\) where \((x, z) \in r \rightleftharpoons n\) and \((z, y) \in r\) by auto
  obtain \(f\) where \(f 0 = x\) and \(f n = z\) and \(\forall i < n. (f i, f (Suc i)) \in r\) by auto
  let \(?n = \text{Suc } n\)
  let \(?f = \lambda i. \text{if } i = ?n \text{ then } y \text{ else } f i\)
have \(?f ?n = y\) by simp
from \(\text{Suc (Suc 1)}\) [OF \((x, z) \in r \rightleftharpoons n\)]
  obtain \(g\) where \(g 0 = a\) and \(g n = f 0\)
  and \(\forall i < n. (g i, g (Suc i)) \in r\)
  let \(?f = \lambda i. \text{if } i = ?n \text{ then } y \text{ else } f i\)
  have \(?f ?n = y\) by simp
from \(\text{Suc (Suc 1)}\) [OF \((x, z) \in r \rightleftharpoons n\)]
  obtain \(n\) where \((a, f 0) \in r \rightleftharpoons n\) unfolding rtrancl-power by auto
show \(\text{False}\)
next
case (Suc \(n\))
from rtrancl-power [OF \((a, f 0) \in r \rightleftharpoons n\)]
  obtain \(g\) where \(g 0 = a\) and \(g n = f 0\)
  and \(\forall i < n. (g i, g (Suc i)) \in r\)
  let \(?f = \lambda i. \text{if } i < n \text{ then } g i \text{ else } f (i - n)\)
  have \(\text{chain } r\) \(?f\)
  proof
    fix \(i\)
\[
\begin{align*}
\{ & \text{assume } \text{Suc } i < n \\
& \text{then have } (\text{if } i, \text{if } (\text{Suc } i)) \in r \text{ by } (\text{simp add: gseq}) \\
\} \\
\text{moreover} \\
\{ & \text{assume } \text{Suc } i > n \\
& \text{then have } \text{eq: } \text{Suc } (i - n) = \text{Suc } i - n \text{ by arith} \\
& \text{from chain have } (f (i - n), f (\text{Suc } (i - n))) \in r \text{ by } \text{simp} \\
& \text{then have } (f (i - n), f (\text{Suc } i - n)) \in r \text{ by } (\text{simp add: } \text{eq}) \\
& \text{with } (\text{Suc } i > n) \text{ have } (\text{if } i, \text{if } (\text{Suc } i)) \in r \text{ by } \text{simp} \\
\} \\
\text{moreover} \\
\{ & \text{assume } \text{Suc } i = n \\
& \text{then have } \text{eq: } f (\text{Suc } i - n) = g n \text{ by } (\text{simp add: } g n = f 0) \\
& \text{from } (\text{Suc } i = n) \text{ have eq': } i = n - 1 \text{ by arith} \\
& \text{from gseq have } (g i, f (\text{Suc } i - n)) \in r \text{ unfolding eq by } (\text{simp add: Suc eq'}) \\
& \text{then have } (\text{if } i, \text{if } (\text{Suc } i)) \in r \text{ using } (\text{Suc } i = n) \text{ by } \text{simp} \\
\} \\
\text{ultimately show } (\text{if } i, \text{if } (\text{Suc } i)) \in r \text{ by } \text{simp} \\
\text{qed} \\
\text{moreover have } \text{if } 0 \in A \\
\text{proof } (\text{cases } n) \\
\text{case } 0 \\
& \text{with } (a, f 0) \in r^{\sim n} \text{ have eq: } a = f 0 \text{ by } \text{simp} \\
& \text{from } a \text{ show } \text{thesis by } (\text{simp add: eq 0}) \\
\text{next} \\
\text{case } (\text{Suc } m) \\
& \text{then show } \text{thesis by } (\text{simp add: a g0}) \\
\text{qed} \\
\text{ultimately have } \text{¬ } \text{SN-on } r A \text{ unfolding } \text{SN-defs by } \text{best} \\
& \text{with } \text{assms show } \text{False by } \text{simp} \\
\text{qed} \\
\text{qed} \\
\text{declare } \text{subrelI } [\text{Pure.intro}] \\
\text{lemma restrict-SN-trancl-simp } [\text{simp}]: (\text{restrict-SN } A A)^+ = \text{restrict-SN } (A^+) A \\
(\text{is } ?lhs = ?rhs) \\
\text{proof} \\
\text{show } ?lhs \subseteq ?rhs \\
\text{proof} \\
\text{fix } a b \text{ assume } (a, b) \in ?lhs \text{ then show } (a, b) \in ?rhs \\
& \text{unfolding } \text{restrict-SN-def by } (\text{induct rule: trancl.induct}) \text{ auto} \\
\text{qed} \\
\text{next} \\
\end{align*}
\]
show \( \text{rhs} \subseteq \text{lhs} \)
proof
  fix \( a \ b \) assume \( (a, b) \in \text{rhs} \)
  then have \( (a, b) \in A^+ \) and \( \text{SN-on} \ A \{a\} \) unfolding \( \text{restrict-SN-def} \) by \( \text{auto} \)
  then show \( (a, b) \in \text{lhs} \)
proof (induct rule: trancl.induct)
  case \( (\text{r-into-trancl} \ x \ y) \) then show \( \text{case} \) unfolding \( \text{restrict-SN-def} \) by \( \text{auto} \)
next
  case \( (\text{transl-into-trancl} \ a \ b \ c) \)
  then have \( \text{IH} \) : \( (a, b) \in \text{lhs} \) by \( \text{auto} \)
  from \( \text{transl-into-trancl} \) have \( (a, b) \in A^+ \) by \( \text{auto} \)
  from this and \( \langle \text{SN-on} \ A \{a\} \rangle \) have \( \text{SN-on} \ A \{b\} \) by (rule \( \text{steps-preserve-SN-on} \))
  with \( \langle (b, c) \in A \rangle \) have \( (b, c) \in \text{lhs} \)
  unfolding \( \text{restrict-SN-def} \) by \( \text{auto} \)
  with \( \text{IH} \) show \( \text{case} \) by \( \text{simp} \)
qed
qed

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

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

lemma \( \text{join-NF-imp-eq} \):\nassumes \( (x, y) \in r^1 \) and \( x \in \text{NF} \ r \) and \( y \in \text{NF} \ r \)
shows \( x = y \)
proof –

26
from \((x, y) \in r^+\) obtain \(z\) where \((x, z) \in r^*\) and \((z, y) \in (r^{-1})^*\) unfolding join-def by auto
then have \((y, z) \in r^*\) unfolding rtrancl-converse by simp
from \((x \in NF r)\) have \((x, z) \notin r^+\) using NF-no-trancl-step by best
then have \(x = z\) using rtranclD [OF \((x, z) \in r^*\)] by auto
from \((y \in NF r)\) have \((y, z) \notin r^+\) using NF-no-trancl-step by best
then have \(y = z\) using rtranclD [OF \((y, z) \in r^*\)] by auto
with \((x = z)\) show ?thesis by simp
qed

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

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

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

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

lemma join-imp-conversion:
\(r^\downarrow \subseteq r^{\leftrightarrow}\)
proof
fix \(x, z\) assume \((x, z) \in r^\downarrow\)
then obtain \(y\) where \((x, y) \in r^*\) and \((z, y) \in r^*\) by auto
then have \((x, y) \in r^{\leftrightarrow}\) and \((z, y) \in r^{\leftrightarrow}\) by auto
from \((z, y) \in r^{\leftrightarrow}\) have \((y, z) \in r^{\leftrightarrow}\) using conversion-sym unfolding sym-def by best
}
with \( (x, y) \in r^{+++} \) show \((x, z) \in r^{+++} \) using conversion-trans unfolding

trans-def by best

qed

lemma meet-imp-conversion: \( r^\uparrow \subseteq r^{+++} \)
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 \((x, y) \in r^\uparrow \) and \((x, z) \in r^\uparrow \)
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 \((y, z) \in r^\uparrow \)
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 \( \text{thesis} \) by auto

qed

lemma CR-iff-conversion-imp-join: \( \text{CR} \ r = (r^{+++} \subseteq r^\uparrow) \)
proof (intro iffI subrelI)
fix \( x \) \( y \) assume \( \text{CR} \ r \) and \((x, y) \in r^{+++} \)
then obtain \( n \) where \((x, y) \in (r^*)^n \) unfolding conversion-def rtrancl-is-UN-relpow
by auto
then show \((x, y) \in r^{\uparrow}\)
proof (induct \( n \) arbitrary: \( x \))
case \( 0 \)
assume \((x, y) \in r^{\leftrightarrow} \) \( ^{\sim} \) \( 0 \) then have \( x = y \) by simp
show \( \text{thesis} \) unfolding \( cx = y \) by auto
next
case \( (Suc \ n) \)
from \((x, y) \in r^{\leftrightarrow} \) \( ^{\sim} \) \( Suc \ n \) obtain \( z \) where \((x, z) \in r^{\leftrightarrow} \) and \((z, y) \in r^{\leftrightarrow} \)
\( ^{\sim} \) \( n \)
using relpow-Suc-D2 by best
with \( Suc \) have \((z, y) \in r^\uparrow \) by simp
from \((x, z) \in r^{+++} \) show \( \text{thesis} \) by (case)
proof
assume \((x, z) \in r \) with \((z, y) \in r^\downarrow \) show \( \text{thesis} \) by (auto intr: rtrancl-join-join)
next
assume \((x, z) \in r^{\sim-1} \)
then have \((z, x) \in r^* \) by simp
from $(z, y) \in r^1$ obtain $z'$ where $(z, z') \in r^*$ and $(y, z') \in r^*$ by auto
from $\langle CR \rangle r$ and $\langle (z, z') \in r^* \rangle$ have $(x, z') \in r^*$
  by (rule CR-divergence-imp-join)
then obtain $x'$ where $(x, z') \in r^*$ and $(z', x') \in r^*$ by auto
with $\langle (y, z') \in r^* \rangle$ show $\text{thesis}$ by auto
qed

next

proof

lemma $\text{CR-imp-conversionIff-join}$:
  assumes $CR \ r$ shows $r^{***} = r^1$
proof
  show $r^{***} \subseteq r^1$ using $\text{CR-iff-conversion-imp-join \ assms}$ by auto
next

next

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

lemma $\text{join-sym}$: $(s, t) \in A^1 \Longrightarrow (t, s) \in A^1$ by auto

lemma $\text{CR-join-left-I}$:
  assumes $CR \ r$ and $(x, y) \in r^*$ and $(x, z) \in r^1$ shows $(y, z) \in r^1$
proof
  from $\langle (x, z) \in r^1 \rangle$ obtain $x'$ where $(x, x') \in r^*$ and $(z, x') \in r^1$ by auto
  from $\langle CR \ r \rangle$ and $\langle (x, x') \in r^* \rangle$ and $\langle (x, y) \in r^* \rangle$ have $(x, y) \in r^1$ by auto
  then have $(y, x) \in r^1$ using $\text{join-sym}$ by best

lemma $\text{CR-join-right-I}$:
  assumes $CR \ r$ and $(x, y) \in r^1$ and $(y, z) \in r^*$ shows $(x, z) \in r^1$
proof
  have $r^{***} = r^1$ by (rule $\text{CR-imp-conversionIff-join} \ \langle OF \ CR \ r1 \rangle$)
  from $\langle (y, z) \in r^* \rangle$ have $(y, z) \in r^{***}$ by auto
  with $\langle (x, y) \in r^1 \rangle$ show $\text{thesis}$ unfolding $r^{***} = r^1$ [symmetric] using $\text{conversion-trans}$
  unfolding $\text{trans-def}$ by fast
qed

lemma $\text{NF-not-suc}$:
  assumes $(x, y) \in r^*$ and $x \in NF \ r$ shows $x = y$
proof
  from $\langle x \in NF \ r \rangle$ have $\forall y. \ (x, y) \notin r$ using $\text{NF-no-step}$ by auto
then have \( x \notin \text{Domain } r \) unfolding Domain-unfold by simp
from \( (x, y) \in r^* \) show \(?thesis\) unfolding Not-Domain-rtrancl [OF \( x \notin \text{Domain } r^* \)] by simp
qed

lemma semi-complete-imp-conversionIff-same-NF:
assumes semi-complete \( r \)
shows \( ((x, y) \in r^{***}) = ((\forall \: u \: v. \: (x, u) \in r^i \land (y, v) \in r^i \rightarrow u = v) \) proof
from assms have WN \( r \) and CR \( r \) unfolding semi-complete-defs by auto
then have \( r^{***} = r^i \) using CR-imp-conversionIff-join by auto
show \(?thesis\)
proof
assume \( (x, y) \in r^{***} \)
from \( (x, y) \in r^{***} \) have \( (x, y) \in r^i \) unfolding \( r^{***} = r^i \) .
show \( \forall \: u \: v. \: (x, u) \in r^i \land (y, v) \in r^i \rightarrow u = v \)
proof (intro allI impl, elim conjE)
fix \( u \: v \) assume \( (x, u) \in r^i \) and \( (y, v) \in r^i \)
then have \( (x, u) \in r^* \) and \( (y, v) \in r^* \) and \( u \in \text{NF } r \) and \( v \in \text{NF } r \) by auto
from CR \( r \) and \( (x, u) \in r^* \) and \( (x, y) \in r^i \) have \( (u, y) \in r^i \)
by (auto intro: CR-join-left-I)
then have \( (y, u) \in r^i \) unfolding join-sym by best
with \( (x, y) \in r^i \) have \( (x, u) \in r^i \) unfolding \( r^{***} = r^i \) [symmetric]
using conversion-trans unfolding trans-def by best
from CR \( r \) and \( (x, y) \in r^i \) and \( (y, v) \in r^* \) have \( (x, v) \in r^i \)
by (auto intro: CR-join-right-I)
then have \( (v, x) \in r^i \) unfolding join-sym unfolding sym-def by best
with \( (x, u) \in r^i \) have \( (v, u) \in r^i \) unfolding \( r^{***} = r^i \) [symmetric]
using conversion-trans unfolding trans-def by best
then obtain \( v' \) where \( (v, v') \in r^* \) and \( (u, v') \in r^* \) by auto
from \( (u, v') \in r^* \) and \( u \in \text{NF } r \) have \( u = v' \) by (rule NF-not-suc)
from \( (v, v') \in r^* \) and \( v \in \text{NF } r \) have \( v = v' \) by (rule NF-not-suc)
then show \( u = v \) unfolding \( u = v' \) by simp
qed
next
assume equal-NF: \( \forall \: u \: v. \: (x, u) \in r^i \land (y, v) \in r^i \rightarrow u = v \)
from WN \( r \) obtain \( u \) where \( (x, u) \in r^i \) by auto
from WN \( r \) obtain \( v \) where \( (y, v) \in r^i \) by auto
from \( (x, u) \in r^i \) and \( (y, v) \in r^i \) have \( u = v \) unfolding equal-NF by simp
from \( (x, u) \in r^i \) and \( (y, v) \in r^i \) have \( (x, v) \in r^* \) and \( (y, v) \in r^* \)
unfolding \( u = v \) by auto
then have \( (x, v) \in r^{***} \) and \( (y, v) \in r^{***} \) by auto
from \( (y, v) \in r^{***} \) have \( (v, y) \in r^{***} \) using conversion-sym unfolding sym-def by best
with \( (x, v) \in r^{***} \) show \( (x, y) \in r^{***} \) using conversion-trans unfolding trans-def by best
qed
qed
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** = r¹ by (rule CR-imp-conversioniff-join [OF assms])
  from ⟨(x, y) ∈ r**⟩ have (x, y) ∈ r¹ unfolding ⟨r** = r¹⟩ by simp
  then obtain x' where (x, x') ∈ r¹ and (y, x') ∈ r* by best
  from ⟨(x, x') ∈ r¹⟩ and ⟨x ∈ NF r⟩ have x = x¹ by (rule NF-not-suc)
  from ⟨(y, x') ∈ r¹⟩ and ⟨y ∈ NF r⟩ have y = x¹ by (rule NF-not-suc)
  then have x = y unfolding ⟨x = x¹ ⟩ by simp
} then show ?thesis by (auto simp: UNC-def)
qed

lemma WN-UNF-imp-CR:
  assumes WN r and UNF r shows CR r
proof - {
  fix x y z assume ⟨x, y⟩ ∈ r¹ and ⟨x, z⟩ ∈ r¹
  from assms obtain y' where ⟨y, y'⟩ ∈ r¹ unfolding WN-defs by best
  with ⟨(x, y) ∈ r¹⟩ have ⟨x, y'⟩ ∈ r¹ by auto
  from assms obtain z' where ⟨z, z'⟩ ∈ r¹ unfolding WN-defs by best
  with ⟨(x, z) ∈ r¹⟩ have ⟨x, z'⟩ ∈ r¹ by auto
  with ⟨(y, y') ∈ r¹⟩ have y' = z' using ⟨UNF r¹⟩ unfolding UNF-defs by auto
  from ⟨(y, y') ∈ r¹⟩ and ⟨(z, z') ∈ r¹⟩ have ⟨y, z⟩ ∈ r¹ unfolding ⟨y' = z'⟩ by auto
} then show ?thesis by auto
qed

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

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

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

lemma diamond-imp-semi-confluence:
  assumes ✐ r shows (r⁻¹ O r¹) ⊆ r¹
proof (rule subrell)
  fix y z assume (y, z) ∈ r⁻¹ O r¹
  then obtain x where (x, y) ∈ r and (x, z) ∈ r¹ by best
  then obtain n where (x, z) ∈ r⁻ⁿ using rtrancl-imp-UN-relpow by best
  with ⟨(x, y) ∈ r¹⟩ show (y, z) ∈ r¹
  proof (induct n arbitrary: x y)
    case 0 then show ?case by auto
  next
    case (Suc n)
    from ⟨(x, z) ∈ r⁻ⁿ Suc n⟩ obtain x' where (x, x') ∈ r and (x', z) ∈ r⁻ⁿ
      using relpow-Suc-D2 by best

31
with \((x, y) \in r\) have \((y, x') \in (r^{-1} O r)\) by auto

with \(\Diamond r\) have \((y, x') \in (r O r^{-1})\) by auto

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

with \(\text{Suc} and \langle x', z \rangle \in r^{-\infty} n\) have \((y', z) \in r^\perp\) by auto

with \(\langle y, y' \rangle \in r\) show \(?\text{thesis}\) by (auto intro: rtrancl-join-join)

qed

lemma \textit{semi-confluence-imp-CR}:

assumes \((r^{-1} O r^*) \subseteq r^\perp\) shows \(\text{CR } r\)

proof -

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

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

with \(\langle x, y \rangle \in r^*\) have \((y, z) \in r^\perp\) by simp

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

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

next

\begin{itemize}
  \item case \((\text{Suc } n)\)
  \begin{itemize}
    \item from \((x, z) \in r^{-\infty} \text{Suc } n\) obtain \(x'\) where \((x, x') \in r\) and \((x', z) \in r^{-\infty} n\)
      using relpow-Suc-D2 by best

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

    \item with assms have \((x', y) \in r^\perp\) by auto

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

    \item with \(\text{Suc} and \langle x', z \rangle \in r^{-\infty} n\) have \((y', z) \in r^\perp\) by simp

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

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

    \item with \(\langle z, u \rangle \in r^*\) show \(?\text{thesis}\) by best
  \end{itemize}
\end{itemize}

qed

lemma \textit{diamond-imp-CR}:

assumes \(\Diamond s\) shows \(\text{CR } r\)

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

lemma \textit{diamond-imp-CR'}:

assumes \(\Diamond s\) and \(r \subseteq s\) and \(s \subseteq r^*\) shows \(\text{CR } r\)

unfolding \(\text{CR-iff-meet-subset-join}\)

proof -

from \(\Diamond s\) have \(\text{CR } s\) by (rule diamond-imp-CR)

then have \(s^1 \subseteq s^\perp\) unfolding \(\text{CR-iff-meet-subset-join}\) by simp

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

from \(s \subseteq r^*\) have \(s^* \subseteq (r^*)^\perp\) by (rule rtrancl-mono)

then have \(s^\perp \subseteq r^*\) by simp

with \(r^* \subseteq s^*\) have \(r^* = s^*\) by simp

show \(r^\perp \subseteq r^*\) unfolding meet-def join-def rtrancl-converse \(r^* = s^*\)

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

join-def [symmetric] by (rule \(s^1 \subseteq s^\perp\))

qed

32
lemma SN-imp-minimal:
assumes SN A
shows ∀ Q x. x ∈ Q → (∃ z∈Q. ∀ y. (z, y) ∈ A → y ∉ Q)
proof (rule ccontr)
assume ∃ (∀ Q x. x ∈ Q → (∃ z∈Q. ∀ y. (z, y) ∈ A → y ∉ Q))
then obtain Q x where x ∈ Q and ∀ z∈Q. ∃ y. (z, y) ∈ A ∧ y ∈ Q by auto
then have ∀ x. x ∈ Q → (∃ y. (z, y) ∈ A ∧ y ∈ Q) by auto
then have ∃ f. ∀ x. x ∈ Q → (x, f x) ∈ A ∧ f x ∈ Q by (rule ccontr)
then obtain f where a:∀ x. x ∈ Q → (x, f x) ∈ A ∧ f x ∈ Q (is ∀x. P x)
by best
let ?S = λi. (f ^^ i) x
have ?S 0 = x by simp
have ∀ i. (?!S i, ?S (Suc i)) ∈ A ∧ ?S (Suc i) ∈ Q
proof
fix i show (?S i, ?S (Suc i)) ∈ A ∧ ?S (Suc i) ∈ Q
  by (induct i) (auto simp: x ∈ Q, a)
qed
with (?S 0 = x) have ∃ S. S 0 = x ∧ chain A S by fast
with assms show False by auto
qed

lemma SN-on-imp-on-minimal:
assumes SN-on r {x}
shows ∀ Q. x ∈ Q → (∃ z∈Q. ∀ y. (z, y) ∈ r → y ∉ Q)
proof (rule ccontr)
assume ∃ (∀ Q. x ∈ Q → (∃ z∈Q. ∀ y. (z, y) ∈ r → y ∉ Q))
then obtain Q where x ∈ Q and ∀ z∈Q. ∃ y. (z, y) ∈ r ∧ y ∈ Q by auto
then have ∀ z. ∃ y. z ∈ Q → (z, y) ∈ r ∧ y ∈ Q by auto
then have ∃ f. ∀ x. x ∈ Q → (x, f x) ∈ r ∧ f x ∈ Q by (rule ccontr)
then obtain f where a:∀ x. x ∈ Q → (x, f x) ∈ r ∧ f x ∈ Q (is ∀x. P x)
by best
let ?S = λi. (f ^^ i) x
have ?S 0 = x by simp
have ∀ i. (?S i, ?S (Suc i)) ∈ r ∧ ?S (Suc i) ∈ Q
proof
fix i show (?S i, ?S (Suc i)) ∈ r ∧ ?S (Suc i) ∈ Q by (induct i) (auto simp: x ∈ Q, a)
qed
with (?S 0 = x) have ∃ S. S 0 = x ∧ chain r S by fast
with assms show False by auto
qed

lemma minimal-imp-wf:
assumes ∀ Q x. x ∈ Q → (∃ z∈Q. ∀ y. (z, y) ∈ r → y ∉ Q)
shows wf(r⁻¹)
proof (rule ccontr)
assume ∃ P. (∀ y. (∀ x. (y, x) ∈ r → P y) → P x) ∧ (∃ x. ¬ P x) unfolding
...
\[\text{wf-def by simp}
\]

then obtain \(P\) where \(\forall x. (\forall y. (x, y) \in r \rightarrow P y) \rightarrow P x\) and \(\neg P x\) by auto

let \(?Q = \{x. \neg P x\}\)

from \(\neg P x\) have \(x \in ?Q\) by simp

from assms have \(\forall x. x \in ?Q \rightarrow (\exists z \in ?Q. \forall y. (z, y) \in r \rightarrow y \notin ?Q)\) by (rule allE [where \(x = ?Q\)])

with \(x \in ?Q\), obtain \(z\) where \(z \in ?Q\) and \(\forall y. (z, y) \in r \rightarrow y \notin ?Q\) by best

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

**Lemma**: \(\text{wf-imp-SN}\):

assumes \(\text{wf} (A^{-1})\) shows \(\text{SN} A\)

proof - { \(\text{fix} a\)

let \(?P = \lambda a. \neg(\exists S. S 0 = a \land \text{chain} A S)\)

from \(\text{wf} (A^{-1})\) have \(?P a\)

proof induct

case (less \(a\))

then have \(IH: \forall b. (a, b) \in A \Rightarrow ?P b\) 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 \(IH\) have \(?P (S 1)\) unfolding \(S 0 = a\) by auto

with \(\text{chain} A S\) show \(\text{False}\) by auto

qed

then have \(\text{SN-on} A \{a\}\) unfolding \(\text{SN-defs}\) by auto

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

\hline

**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 this] show \(?\text{thesis}\).

\hline

**Lemma**: \(\text{SN-iff-wf}: \text{SN} A = \text{wf} (A^{-1})\) by (auto simp: \(\text{SN-imp-wf} \ \text{wf-imp-SN}\))

\hline

34
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. SN-on 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
      assume ind: \forall b. (a, b) \in ?R \implies SN-on R \{b\} \implies P b
      show SN-on R \{a\} \implies P a
        proof
          assume SN: SN-on R \{a\}
          show P a
            proof (rule imp)
              fix b
              assume (a, b) \in R
              with SN step-preserves-SN-on [OF this SN]
                show P b using ind [of b] unfolding restrict-SN-def by auto
            qed
          qed
        qed
      with SN show P s using \langle s \in A \rangle unfolding SN-on-def by blast
    qed
  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 - {
  fix x assume x \in A
  from assms [OF this]
have $SN$-on $\{(y, z), g z y\} \{x\}$

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

then show $?thesis$ unfolding $SN$-on-def by blast
qed

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

proof
  fix $x$ assume $x \in A$
  with assms show Wellfounded.accp $g x$
  proof (induct rule: $SN$-on-induct)
    case (IH $x$)
    show $?case$
    proof
      fix $y$
      assume $g y x$
      with IH show Wellfounded.accp $g y$ by simp
    qed
  qed
qed

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

using $SN$-imp-accp [of $g \{x\}$]
  accp-imp-$SN$-on [of $\{x\} g$]
by auto

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

unfolding $SN$-conv-accp accp-acc-eq ..

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

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

lemma relpow-Suc-E2' [elim]:
assumes \((x, z) \in A^\sim Suc\ n\) obtains \(y\) where \((x, y) \in A\) and \((y, z) \in A^*\)
proof -
  assume \(\text{assm}: \forall y. (x, y) \in A \Longrightarrow (y, z) \in A^* \Longrightarrow \text{thesis}\)
  from relpow-Suc-E2 [OF assms] obtain \(y\) where \((x, y) \in A\) and \((y, z) \in A^\sim n\)
by auto
  then have \((y, z) \in A^*\) using relpow-imp-rtrancl by auto
from \(\text{assm} [OF \langle x, y \rangle \in A\; \text{this}]\) show \(\text{thesis}\).
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 \(r\) \(\{x\}\)
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 \((x, y) \in r^\sim m\) ..
      from \((x, z) \in r^*\) obtain \(n\) where \((x, z) \in r^\sim n\) ..
      show \((y, z) \in r^*\)
      proof (cases \(n\))
        case 0
        from \((x, z) \in r^\sim n\) have eq: \(x = z\) by (simp add: 0)
        from \((x, y) \in r^*\) show \(?thesis\) unfolding eq ..
      next
      case (Suc \(n')\)
      from \((x, z) \in r^\sim n\) [unfolded Suc] obtain \(t\) where \((x, t) \in r\) and \((t, z) \in r^*\) ..
      show \(?thesis\)
      proof (cases \(m\))
        case 0
        from \((x, y) \in r^\sim m\) have eq: \(x = y\) by (simp add: 0)
        from \((x, z) \in r^*\) show \(?thesis\) unfolding eq ..
      next
      case (Suc \(m')\)
      from \((x, y) \in r^\sim m\) [unfolded Suc] obtain \(s\) where \((x, s) \in r\) and \((s,
\( y \in r^* .. \)

from \( \text{WCR IH}(2) \) have \( \text{WCR-on } r \{x\} \) unfolding \( \text{WCR-on-def} \) by auto
with \( \langle x, s \rangle \in r, \text{ and } \langle x, t \rangle \in r \) have \( s, t \in r^* \) by auto
then obtain \( u \) where \( (s, u) \in r^* \) and \( (t, u) \in r^* .. \)
from \( \langle x, s \rangle \in r, \text{ IH}(2) \) have \( \text{SN-on } r \{s\} \) by \( \text{(rule step-preserves-SN-on)} \)
from \( \text{IH}(1)[OF \langle x, s \rangle \in r \} \) this\) have \( \text{CR-on } r \{s\} \).
from \( \text{this and } (s, u) \in r^* \) and \( \langle s, y \rangle \in r^* \) have \( u, y \in r^* \) by auto
then obtain \( v \) where \( (u, v) \in r^* \) and \( (y, v) \in r^* .. \)
from \( \langle x, t \rangle \in r, \text{ IH}(2) \) have \( \text{SN-on } r \{t\} \) by \( \text{(rule step-preserves-SN-on)} \)
from \( \text{IH}(1)[OF \langle x, t \rangle \in r \} \) this\) have \( \text{CR-on } r \{t\} \).
moreover from \( \langle t, u \rangle \in r^* \) and \( \langle u, v \rangle \in r^* \) have \( t, v \in r^* \) by auto
ultimately have \( z, v \in r^* \) using \( \langle (t, z) \in r^* \) by auto
then obtain \( w \) where \( (z, w) \in r^* \) and \( (v, w) \in r^* .. \)
from \( \langle (y, v) \in r^* \) and \( \langle (v, w) \in r^* \) have \( (y, w) \in r^* \) by auto
with \( \langle (z, w) \in r^* \) show \( \text{thesis by auto} \)
qed
qed
qed
qed
}\)
then show \( \text{thesis unfolding \text{CR-on-def by blast]} \)
qed

\begin{lemma}
\text{Newman: SN } r \implies \text{WCR } r \implies \text{CR } r
\end{lemma}

\begin{proof}
fix \( f \)
assume \( f \) 0 \in \( A \) and \( \text{chain: chain } r f \)
then have \( f (\text{Suc } 0) \in r \implies A \) by auto
with \( \text{assms have SN-on } r \{f (\text{Suc } 0)\} \) by \( \text{auto simp add: } f \) 0 \in \( A \} \) \text{SN-defs}
moreover have \( \neg \text{SN-on } r \{f (\text{Suc } 0)\} \)
\\begin{proof}
have \( f (\text{Suc } 0) \in \{f (\text{Suc } 0)\} \) by \( \text{simp} \)
moreover from \( \text{chain have chain } r (f \circ \text{Suc }) \) by auto
ultimately show \( \text{thesis by auto} \)
qed
ultimately show \( \text{False by simp} \)
qed
\end{proof}

\begin{lemma}
\text{SN-on-Image-conv: SN-on } r \implies (r \implies A) = \text{SN-on } r A
\end{lemma}

\begin{proof}
If all successors are terminating, then the current element is also terminating.
\end{proof}
lemma step-reflects-SN-on:
assumes \( (\forall b. (a, b) \in r \Rightarrow SN \text{-} on \ r \ {b}) \)
shows \( SN \text{-} on \ r \ {a} \)
using assms and Image-SN-on \([of \ r \ {a}]\) by (auto simp: SN-defs)

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

lemma SN-imp-SN-trancl:
\[ SN \ R = \Rightarrow SN (R^+) \]
unfolding SN-ifw by (rule wf-converse-trancl)

lemma SN-trancl-imp-SN:
assumes \( SN (R^+) \)
shows \( SN \ R \)
using assms by (rule SN-on-trancl-imp-SN-on)

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

lemma SN-inv-image:
\[ SN \ R = \Rightarrow SN (inv\text{-}image \ R \ f) \]
unfolding SN-iff-wf by simp

lemma SN-subset:
\[ SN \ R = \Rightarrow R' \subseteq R \Rightarrow SN \ R' \]
unfolding SN-defs by blast

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

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

lemma SN-imp-SN-pow:
assumes \( SN \ R \) shows \( SN (R^{\sim} Suc \ n) \)
using SN-subset \([where \ R=R^+\), OF SN-imp-SN-trancl \([OF \ assms]\) pow-Suc-subset-trancl]\)
by simp
lemma  \( SN\text{-pow}: SN R \iff SN (R \imp Succ\ n) \)
by (rule iffI, rule SN-imp-SN-pow, assumption, rule SN-pow-imp-SN, assumption)

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

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

Restrict an ARS to elements of a given set.

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

lemma  \( SN\text{-on-restrict}: \)
assumes  \( SN\text{-on } r A \)
shows  \( SN\text{-on } (\text{restrict } r S) A \) (is  \( SN\text{-on } \tilde{r} A \))
proof  (rule ccontr)
assumption ¬SN-on ?r A
then have ∃f. f 0 ∈ A ∧ chain ?rf 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 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) ∈ r^n and x ∈ ?S and y ∈ ?S
  using rtrancl-imp-UN-relpow by best
then have (x, y) ∈ ?r^* by auto
proof (induct n arbitrary: x y)
case 0 then show \(?case\) by simp

next

case \(\text{Suc } n\)

from \(\langle x, y \rangle \in r{\sim} \text{Suc } n\) obtain \(x'\) where \(x, x' \in r\) and \((x', y) \in r{\sim} n\)
using relpow-Suc-D2 by best
then have \((x, x') \in r^*\) by simp

with \(\langle x, y \rangle \in ?S\) have \((x', y) \in ?S\) by (rule rtrancl-Image-step)

with \(\text{Suc}\) and \(\langle x', y \rangle \in r{\sim} n\) have \((x', y) \in r^*\) by simp
from \((x, x') \in r\) and \(\langle x \in ?S\rangle\) and \((x' \in ?S)\) have \((x, x') \in r\)
unfolding restrict-def by simp

with \(\langle x', y \rangle \in ?r^*\) show \(?case\) by simp

qed

\}
then have \(\forall x\ y\ z\ \forall x, y \in ?S\land y \in ?S\rightarrow (x, y) \in ?r^*\) by simp

\fix \(x'\ y\ z\ \forall x, y \in ?S\land y \in ?S\rightarrow (x, y) \in ?r^*\) by simp

then have \(x' \in ?S\land y \in ?S\land z \in ?S\land (x', y) \in r\land (x', z) \in r\)

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

\fix \(x\ y\ z\ \forall x, y \in ?S\land z \in ?S\rightarrow (x, y) \in r^*\) by auto
then have \(y \in ?S\land z \in ?S\) by auto
have \(x \in ?S\) using \(\langle x \in A\rangle\) by auto
from \(\langle (x, y) \in r^*\rangle\) and \(\langle x \in ?S\rangle\) and \(\langle y \in ?S\rangle\) have \((x, y) \in ?r^*\) by simp
from \(\langle (x, z) \in r^*\rangle\) and \(\langle x \in ?S\rangle\) and \(\langle z \in ?S\rangle\) have \((x, z) \in ?r^*\) by simp

\with \(\langle \text{CR } ?r \rangle\) and \((x, y) \in ?r^*\) have \((y, z) \in r^+\) by auto
then obtain \(u\) where \((y, u) \in ?r^*\) and \((z, u) \in ?r^*\) by best
then have \((y, u) \in r^*\land (z, u) \in r^*\) using restrict-rtrancl by auto
then have \((y, z) \in r^*\) by auto

\then show \(?thesis\) by auto

qed

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

2.4 Commutation

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

lemma \text{CR-iff-self-commute}: \ CR r = \text{commute } r \;\;\text{r}
unfolding \text{commute-def} \;\;\text{CR-iff-meet-subset-join \; meet-def \; join-def}
by simp

lemma \text{rtrancl-imp-rtrancl-UN}: 
assumes \ (x, y) \in r^\ast \;\;\text{and } r \in \text{I}
shows \ (x, y) \in (\bigcup r \in \text{I}. \; r^\ast) \;\;\text{(is } (x, y) \in \;\;r^\ast)
using assms proof induct
  case base then show ?case by simp
next
  case (step y z)
  then show (x, y) ∈ ?r∗ by simp
from (y, z) ∈ r, and (r ∈ I) have (y, z) ∈ ?r∗ by auto
with (x, y) ∈ ?r∗ show ?case by auto
qed

definition quasi-commute :: 'a rel ⇒ 'a rel ⇒ bool where
  quasi-commute r s ←→ (s O r) ⊆ r O (r ∪ s)∗

lemma rtrancl-union-subset-rtrancl-union-trancl: (r ∪ s)+∗ = (r ∪ s)∗
proof
  show (r ∪ s)+∗ ⊆ (r ∪ s)∗
    proof (rule subrelI)
      fix x y assume (x, y) ∈ (r ∪ s)+∗
      then show (x, y) ∈ (r ∪ s)∗
        proof (induct)
          case base then show ?thesis by auto
        next
          case (step y z)
          then have (y, z) ∈ r ∨ (y, z) ∈ s+ by auto
          then have (y, z) ∈ (r ∪ s)+ by auto
          proof
            assume (y, z) ∈ r then show ?thesis by auto
          next
            assume (y, z) ∈ s+ then have (y, z) ∈ s∗ by auto
            then have (y, z) ∈ r∗ ∪ s∗ by auto
            then show ?thesis using rtrancl-Un-subset by auto
          qed
          with (x, y) ∈ (r ∪ s)+∗ show ?thesis by simp
          qed
          qed
        qed
      qed
  next
    show (r ∪ s)∗ ⊆ (r ∪ s)+∗
      proof (rule subrelI)
        fix x y assume (x, y) ∈ (r ∪ s)∗
        then show (x, y) ∈ (r ∪ s)+∗
          proof (induct)
            case base then show ?case by auto
          next
            case (step y z)
            then have (y, z) ∈ (r ∪ s)+∗ by auto
            with (x, y) ∈ (r ∪ s)+∗ show ?case by auto
            qed
            qed
          qed
        qed
      qed
  qed
qed
lemma qc-imp-qc-trancl:
  assumes quasi-commute r s shows quasi-commute r (s+)
unfolding quasi-commute-def
proof (rule subrelI)
  fix x z assume (x, z) ∈ s+ O r
  then obtain y where (x, y) ∈ s+ and (y, z) ∈ r by best
  then show (x, z) ∈ r O (r ∪ s+)*
proof (induct arbitrary: z)
  case (base y)
    then have (x, z) ∈ (s O r) by auto
    with assms have (x, z) ∈ r O (r ∪ s+)* unfolding quasi-commute-def by auto
    then show ?thesis using rtrancl-union-subset-rtrancl-union-trancl by auto
  next
    case (step a b)
    then have (a, z) ∈ (s O r) by auto
    with assms have (a, z) ∈ r O (r ∪ s+)* unfolding quasi-commute-def by auto
    then obtain u where (a, u) ∈ r and (u, z) ∈ (r ∪ s+)* by best
    then have (u, z) ∈ (r ∪ s+)* using rtrancl-union-subset-rtrancl-union-trancl by auto
    from ‹(a, u) ∈ r› and step have (x, u) ∈ r O (r ∪ s+)* by auto
    then obtain v where (x, v) ∈ r and (v, u) ∈ (r ∪ s+)* by best
    with ‹(x, v) ∈ r› show ?case by auto
qed
qed

lemma steps-reflect-SN-on:
  assumes ¬SN-on r {b} and (a, b) ∈ r*
  shows ¬SN-on r {a}
using SN-on-Image-rtrancl [of r {a}]
and assms and SN-on-subset2 [of (b) r* "{a} r] by blast

lemma chain-imp-not-SN-on:
  assumes chain r f
  shows ¬SN-on r {f i}
proof –
  let ?f = λj. f (i + j)
  have ?f 0 ∈ {f i} by simp
  moreover have chain r ?f using assms by auto
  ultimately have (?f 0 ∈ {f i} ∧ chain r ?f) by blast
  then have ∃g. g 0 ∈ {f i} ∧ chain r g by (rule exI [of - ?f])
  then show ?thesis unfolding SN-defs by auto
qed

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

46
then obtain \( j \) where \(?i = Suc \ j\) unfolding gr0-conv-Suc by best
with \( \text{ini} \) have \( (S (?i ∼ Suc 0), S (Suc (?i ∼ Suc 0))) \in s \) by auto
with \( \text{pref} \) have \( (b, S (Suc \ j)) \in s^+ \) unfolding \( (?i = Suc \ j) \) by auto
then have \( (b, S ?i) \in s^+ \) unfolding \( (?i = Suc \ j) \) by auto
with \( (S ?i, S (Suc ?i)) \in r \) have \( (b, S (Suc \ j)) \in (s^+ O r) \) by auto
with \( \text{quasi-commute} \ r \ (s^+) \) have \( (b, S (Suc ?i)) \in r O (r \cup s^+) \)

unfolding quasi-commute-def by auto
then obtain \( c \) where \( (b, c) \in r \) and \( (c, S (Suc ?i)) \in (r \cup s^+) \) by best
from \( (b, c) \in r \) have \( (b, c) \in (r \cup s^+) \) by auto
from \( \text{chain-imp-not-SN-on} \ [of \ S r \cup s] \)
and \( \text{chain} \) have \( \neg \text{SN-on} \ (r \cup s) \ \{S (Suc ?i)\} \) by auto
from \( (c, S (Suc ?i)) \in (r \cup s^+) \) have \( (c, S (Suc ?i)) \in (r \cup s) \)
unfolding rtrancl-union-subset-rtrancl-union-trancl by auto
with \( \text{steps-reflect-SN-on} \ [of \ r \cup s] \)
and \( \neg \text{SN-on} \ (r \cup s) \ \{S (Suc ?i)\} \) have \( \neg \text{SN-on} \ (r \cup s) \ \{c\} \) by auto
then have \( c \in yB \) by simp
with \( (b, c) \in r \) and \( \text{min} \) have \( \text{False} \) by auto
\}
then show \( ?\text{thesis} \) by auto
qed

2.5 Strong Normalization

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

lemma comp-trancl:
assumes \( R O S \subseteq S \) shows \( R O S^+ \subseteq S^+ \)
proof (rule subrelI)
fix \( w \) \( z \) assume \( (w, z) \in R O S^+ \)
then obtain \( x \) where \( \text{R-step} \ (w, x) \in R \) and \( \text{S-seq} \ (x, z) \in S^+ \) by best
from tranclD \( [OF \ S-seq] \) obtain \( y \) where \( \text{S-step} \ (x, y) \in S \) and \( \text{S-seq} : (y, z) \in S^+ \) by auto
from \( \text{R-step} \) and \( \text{S-step} \) have \( (w, y) \in R O S \) by auto
with \( \text{assms} \) have \( (w, y) \in S \) by auto

47
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 $(b, c) \in S$ by simp
  with rtrancl-into-rtrancl show ?thesis by simp
next
  assume $(b, c) \in R$ by simp
  with comp-trancl [OF comp] rtrancl-into-rtrancl
  show ?thesis by auto
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

lemma restrict-SN-subset: restrict-SN $R S \subseteq R$
proof (rule subrelI)
  fix $a b$ assume $(a, b) \in restrict-SN R S$ then show $(a, b) \in R$
  unfolding restrict-SN-def by simp
qed
lemma \textit{chain-Un-SN-on-imp-first-step}:
\begin{itemize}
\item \textbf{assumes} \textit{chain} \((R \cup S) t \text{ and } SN-on S \{t \ 0\}\)
\item \textbf{shows} \(\exists i. \ (t \ i, t \ (Suc \ i)) \in R \land (\forall j<i. \ (t \ j, t \ (Suc \ j)) \in S \land (t \ j, t \ (Suc \ j)) \notin R)\)
\end{itemize}
\textbf{proof} –
\begin{itemize}
\item from \(\langle SN-on S \{t \ 0\}\rangle\) \textbf{obtain} \(i\) \textbf{where} \((t \ i, t \ (Suc \ i)) \notin S\) \textbf{by} blast
\item with \textbf{assms} \textbf{have} \((t \ i, t \ (Suc \ i)) \in R \ (is \ ?P \ i)\) \textbf{by} auto
\item let \(?i = \text{Least} \ ?P\) from \(\langle ?P \ i\rangle\) \textbf{have} \(?P \ ?i\) \textbf{by} (rule \text{LeastI})
\item have \((\forall j<i. \ (t \ j, t \ (Suc \ j)) \notin R)\) \textbf{using} \text{not-less-Least} \textbf{by} auto
\end{itemize}
\textbf{qed}

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

lemma \textit{first-step-O}:
\begin{itemize}
\item \textbf{assumes} \(C: \ C = A \cup B\) \textbf{and} \textit{steps}: \((x, y) \in C^* \text{ O B}\)
\item \textbf{shows} \(\exists y. \ (x, y) \in A^* \text{ O B}\)
\item \textbf{proof} –
\item from \textit{steps} \textbf{obtain} \(z\) \textbf{where} \((x, z) \in C^* \textbf{ and} \ (z, y) \in B\) \textbf{by} auto
\item from \textit{first-step} [OF \(C\ \text{this}\)] \textbf{show} \(\textit{thesis}\).
\item \textbf{qed}
\item \textbf{qed}

lemma \textit{firstStep}:
\begin{itemize}
\item \textbf{assumes} \(LSR: \ L = S \cup R\) \textbf{and} \(xyL: \ (x, y) \in L^*\)
\end{itemize}
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 \(x y L\) and \(LS R\) have \((x, y) \in ?SR^*\) by simp
  from this and False have \((x, y) \in R^* O S O ?SR^*\)
  proof (induct rule: rtrancl-induct)
    case base then show ?case by simp
  
  next
    case (step y z)
    then show ?case
  
  proof (cases \((x, y) \in R^*\))
    case False
    with \(\text{step}\) have \((x, y) \in R^* O S O ?SR^*\) by simp
    from this obtain \(u\) where \(xu\): \((x, u) \in R^* O S\) and \(uy\): \((u, y) \in ?SR^*\) by force
    from \((y, z) \in ?SR\) have \((y, z) \in R^*\) 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 \(\neg ?thesis\)
      with \(\text{chain}\) have \(\forall i\geq j. \ (t i, t (Suc i)) \in R - S\)
      proof (rule contr)
        assume \(\neg ?thesis\)
        with \(\text{chain}\) 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: \(\forall i. i \leq f i \land (t (f i), t (Suc (f i))) \in S^+\)
        let \(\theta = \lambda i. \ (Suc \circ f) \ (\_\_\_\_ i)\)
        have \(S\)-chain: \(\forall i. (t i, t (Suc (f i))) \in S^+\)
        proof

lemma non-strict-ending:
  assumes chain: chain \((R \cup S)\) \(t\)
  and comp: \(R O S \subseteq S\)
  and SN: \(\text{SN-on } S \{t 0\}\)
  shows \(\exists j. \forall i\geq j. \ (t i, t (Suc i)) \in R - S\)
  proof (rule contr)
    assume \(\neg ?thesis\)
    with \(\text{chain}\) 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: \(\forall i. i \leq f i \land (t (f i), t (Suc (f i))) \in S^+\)
    proof
fix \( i \)

from S-steps have \( \leq f \) and step: \((t(f i), t(Suc(f i))) \in S\) by auto

then have \((t i, t(Suc(f i))) \in (R \cup S)^* \).

qed

then have chain \((S^+)\) ?by simp

moreover have \(SN-on(S^+)\) { ?t 0 } using SN-on-trancl [OF SN] by simp

ultimately show False unfolding SN-defs by best

qed

lemma \( SN-on-subset1 \):

assumes \( SN-on r A \) and \( s \subseteq r \)

shows \( SN-on s A \)

using assms unfolding SN-defs by blast

lemmas \( SN-on-mono = SN-on-subset1 \)

lemma \( rtrancl-fun-conv \):

\( ((s, t) \in R^*) = (\exists f \ n. f \ 0 = s \wedge f \ n = t \wedge (\forall i < n. (f \ i, f \ (Suc \ i)) \in R)) \)

unfolding rtrancl-is-UN-relpow using relpow-fun-conv [where \( R = R \)] by auto

lemma \( compat-tr-compat \):

assumes \( NS \ O S \subseteq S \) shows \( NS^* \ O S \subseteq S \)

using non-strict-into-strict [where \( S = S \) and \( NS = NS \)] assms by blast

lemma \( right-comp-S \) [simp]:

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

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

proof –

from assms have \((x, y) \in (S \ O S \ O S^* \ O NS^*) \cup (S \ O NS^*) \) by auto

then have \(xy:(x, y) \in (S \ O (S \ O S^*) \ O NS^*) \cup (S \ O NS^*) \) by auto

have \( S \ O S^* \subseteq S^* \) by auto

with \(xy\) have \((x, y) \in (S \ O S^* \ O NS^*) \cup (S \ O NS^*) \) by auto

then show \((x, y) \in (S \ O S^* \ O NS^*) \) by auto

qed

lemma \( compatible-SN \):

assumes \( SN: SN S \)

and \( compat: NS \ O S \subseteq S \)

shows \( SN \ (S \ O S^* \ O NS^*) \) (is \( SN ?A \))

proof

fix \( F \) assume chain: chain ?A \( F \)

from \( compat-tr-compat \) have \( tr-compat: NS^* \ O S \subseteq S \) by blast

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

proof

fix \( i \)

from chain have \((F \ i, F \ (Suc \ i)) \in (S \ O S^* \ O NS^*) \) by auto
then show $\exists \ y \ z. \ (F \ i, \ y) \in S \land (y, z) \in S^* \land (z, F (Suc \ i)) \in NS^*$

unfolding relcomp-def using mem-Collect-eq by auto

qed

then have $\exists \ f. \ (\forall \ i. \ (\exists \ z. \ (F \ i, \ f \ i) \in S \land ((f \ i, \ z) \in S^*) \land (z, F (Suc \ i)) \in NS^*))$

by (rule choice)

then obtain $f$

where $\forall \ i. \ (\exists \ z. \ (F \ i, \ f \ i) \in S \land ((f \ i, \ z) \in S^*) \land (z, F (Suc \ i)) \in NS^*)$ ..

then have $\exists \ g. \ \forall \ i. \ (F \ i, \ f \ i) \in S \land (f \ i, \ g \ i) \in S^* \land (g \ i, F (Suc \ i)) \in NS^*$

by (rule choice)

then obtain $g$ where $\forall \ i. \ (F \ i, \ f \ i) \in S \land (f \ i, \ g \ i) \in S^* \land (g \ i, F (Suc \ i)) \in NS^*$ ..

then have $\forall \ i. \ (f \ i, \ g \ i) \in S^* \land (g \ i, F (Suc \ i)) \in NS^* \land (F (Suc \ i), f (Suc \ i)) \in S$

by auto

then have $\forall \ i. \ (f \ i, \ g \ i) \in S^* \land (g \ i, F (Suc \ i)) \in S$ unfolding relcomp-def

using tr-compat by auto

then have $\forall \ i. \ (f \ i, \ g \ i) \in S^* \land (g \ i, F (Suc \ i)) \in S^+$ 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 $\exists \ n. \ (x, \ y) \in (NS \cup S)^\sim n$ using rtrancl-imp-relpow $[where \ \{r=\} = NS \cup S]$ by auto

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

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

proof (induct $n$ arbitrary: $x$, simp)

case $(Suc \ m)$

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

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

using relpow-Suc-D2 $[where \ \{r=\} = NS \cup S]$ by auto

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

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

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

53
lemma \texttt{rtrancl-diff-decomp}:
assumes \((x, y) \in A^* - B^*
shows \((x, y) \in A^* O (A - B) O A^*
proof
from \text{assms} \text{have } A: (x, y) \in A^* \text{ and } B: (x, y) \notin B^* \text{ by auto}
from A \text{ have } \exists k. (x, y) \in A^{-k} \text{ by (rule } \text{rtrancl-imp-relpow)}
then obtain k where Ak: (x, y) \in A^{-k} \text{ by auto}
from Ak B \text{ show } (x, y) \in A^* O (A - B) O A^*
proof (induct k arbitrary: x)
  case 0
  with \(((x, y) \notin B^*) \text{ 0 show } \text{case using ccontr by auto}
next
  case (Suc i)
  then have B: (x, y) \notin B^* \text{ and } ASk: (x, y) \in A ^{\sim i} \text{ by auto}
from ASk \text{ have } \exists z. (x, z) \in A \land (z, y) \in A ^{\sim i} \text{ using relpow-Suc-D2 [where } \text{?R=A]} \text{ by auto}
then obtain z where xz: (x, z) \in A \text{ and } (z, y) \in A ^{\sim i} \text{ by auto}
then have zy: (z, y) \in A^* \text{ using relpow-imp-rtrancl by auto}
from zy \text{ show } (x, y) \in A^* O (A - B) O A^*
proof (cases (x, z) \in B)
  case False
  with xz zy \text{ show } (x, y) \in A^* O (A - B) O A^* \text{ by auto}
next
  case True
  then have (x, z) \in B^* \text{ by auto}
  have \(((x, z) \in B^*; (z, y) \in B^*) \implies (x, y) \in B^* \text{ using rtrancl-trans [of } x z \text{ B]} \text{ by auto}
  with \(((x, z) \in B^*; (x, y) \notin B^*; \text{ have } (z, y) \notin B^* \text{ by auto}
  with Suc \text{ ([z, y) } A ^{\sim i} \text{ have } (z, y) \in A^* O (A - B) O A^* \text{ by auto}
  with xz \text{ have } zy: (x, y) \in A O A^* O (A - B) O A^* \text{ by auto}
  have A O A^* O (A - B) O A^* \subseteq A^* O (A - B) O A^* \text{ by regexp}
  from this xz \text{ show } (x, y) \in A^* O (A - B) O A^*
  using subsetD [where } \text{?A=A O A^* O (A - B) O A^*} \text{ by auto}
qed
qed

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

lemma \texttt{SN-on-weakening}:
assumes \texttt{SN-on R1 A}
shows \texttt{SN-on (R1 \cap R2) A}
proof
\{ 
  assume \exists S. S 0 \in A \land chain (R1 \cap R2) S
  then obtain S where
  S0: S 0 \in A \text{ and }
  SN: chain (R1 \cap R2) S
\}
by auto
from SN have SN': chain R1 S by simp
with S0 and assms have False by auto
}
then show ?thesis by force
qed

definition ideriv :: 'a rel ⇒ 'a rel ⇒ (nat ⇒ 'a) ⇒ bool where
ideriv R S as ⇔ (∀ i. (as i, as (Suc i)) ∈ R ∪ S) ∧ (INF i. (as i, as (Suc i)) ∈ R)

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

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

lemma ideriv-split:
assumes ideriv: ideriv R S as
and nideriv: ∼ ideriv (D ∩ (R ∪ S)) (R ∪ S − D) as
shows ⋁ i. ideriv (R − D) (S − D) (shift as i)
proof −
have RS: R − D ∪ (S − D) = R ∪ S − D by auto
from ideriv [unfolded ideriv-def]
have as: ⋀ i. (as i, as (Suc i)) ∈ R ∪ S
and inf: INF i. (as i, as (Suc i)) ∈ R by auto
show ?thesis
proof (cases INF i. (as i, as (Suc i)) ∈ D ∩ (R ∪ S))
  case True
  have ideriv (D ∩ (R ∪ S)) (R ∪ S − D) as
    unfolding ideriv-def
    using as True by auto
  with nideriv show ?thesis ..
next
  case False
  from False [unfolded INFM-nat]
  obtain i where Dn: ⋀ j. i < j ⇒ (as j, as (Suc j)) ∉ D ∩ (R ∪ S)
    by auto
  from Dn as have as: ⋀ j. i < j ⇒ (as j, as (Suc j)) ∈ R ∪ S − D by auto
  show ?thesis
    proof (rule exI [of - Suc i], unfold ideriv-def RS, insert as, intro conjI, simp,
      unfold INFM-nat, intro allI)
      fix m
      from inf [unfolded INFM-nat] obtain j where j: j > Suc i + m
        and R: (as j, as (Suc j)) ∈ R by auto
      with as [of j] have RD: (as j, as (Suc j)) ∈ R − D by auto
    55
show \( \exists j > m. (\text{shift as } (\text{Suc } i) j, \text{shift as } (\text{Suc } i) (\text{Suc } j)) \in R - D \)

by (rule exI [of - j - Suc i], insert j RD, auto)

qed

lemma ideriv-SN:
assumes SN: \( SN S \)
and compat: \( NS O S \subseteq S \)
and R: \( R \subseteq NS \cup S \)
shows \( \neg \text{ideriv } (S \cap R) (R - S) \) as

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

qed

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

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

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

qed

lemma rtrancl-list-conv:
\((s, t) \in R^* \iff \)
\( (\exists \ ts. \ last \ (s \# \ ts) = t \land (\forall i < \text{length} \ ts. \ ((s \# \ ts) ! i, (s \# \ ts) ! \text{Suc} i) \in R)) \)

(is \ ?l = ?r)

proof

assume \ ?r

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

then show \ ?l

proof (induct \ ts \ arbitrary; \ s, simp)

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

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

by \ auto

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

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

with \ rec \ show \ ?case \ by \ auto

qed

next

assume \ ?l

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

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

show \ ?r

proof (rule \exI \ [of \ - \ ?ts], intro \ conjI, \\
cases \ n, \ simp \ add: \ s \ [symmetric] \ t \ [symmetric], \ simp \ add: \ t \ [symmetric])

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

proof (intro \ allI \ impI)

fix \ i

assume \ i: \ i < \text{length} \ ?ts

then show \ ((s \# \ ?ts) ! i, (s \# \ ?ts) ! \text{Suc} i) \in R

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

case (\text{Suc} \ j)

with \ i \ steps \ show \ ?thesis \ by \ simp

qed

qed

qed

lemma \ SN-reaches-NF:

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

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

using \ assms

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

case (IH \ x)

show \ ?case

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

case \ True

then show \ ?thesis \ by \ auto

next

case \ False

57
then obtain $y$ where step: $(x, y) \in r$ by auto
from IH [OF this] obtain $z$ where steps: $(y, z) \in r^*$ and $\text{NF}$: $z \in \text{NF } r$ by auto

show ?thesis by (intro exI, rule conjI [OF - NF], insert step steps, auto)

qed

lemma SN-WCR-reaches-NF:
assumes SN: SN-on $r$ \{ $x$ \}
and WCR: WCR-on $r$ \{ $x$. SN-on $r$ \{ $x$ \} \}
shows $\exists !\ y. (x, y) \in r^* \land y \in \text{NF } r$
proof
from SN-reaches-NF [OF SN] obtain $y$ where steps: $(x, y) \in r^*$ and NF: $y \in \text{NF } r$ by auto
show ?thesis

from SN-WCR-reaches-NF [OF SN WCR] have CR-on $r$ \{ $x$ \} by auto
from CR-onD [OF this - steps] steps\' have $(y, z) \in r^\downarrow$ by simp
from join-NF-imp-eq [OF this NF] steps\' show $z = y$ by simp

qed

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

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

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

let $?p = \lambda y. (x, y) \in r^* \land y \in \text{NF } r$
from SN-WCR-reaches-NF [OF SN WCR]
have one: $\exists !\ y. ?p y$.
from steps NF have $y$: $?p y$.
from some-NF [OF SN] have some: $?p (\text{some-NF } r x)$.
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 \( ?p = \lambda y. (x, y) \in r^* \wedge y \in NF r \)
from steps NF have py: ?p y by simp
then have pNF: ?p (some-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^! \implies c = b for c
    using CR and ab by (meson CR-divergence-imp-join join-NF-imp-eq normalizability-E)
  ultimately have \( \exists! b. (a, b) ∈ A^! \) 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) ∈ A^{***}
shows the-NF A a = the-NF A b
using assms
by (meson CR WN-on-def the-NF semi-complete-imp-conversionIff-same-NF semi-complete-onI)

end

definition weak-diamond :: 'a rel ⇒ bool (w♦) where
w♦ r ⇔ (r⁻¹ O r) - 1d ⊆ (r O r⁻¹)

lemma weak-diamond-imp-CR:
  assumes wd:
  shows CR r
proof (rule semi-confluence-imp-CR, rule)
  fix x y
  assume (x, y) ∈ r⁻¹ O r*
  then obtain z where step: (z, x) ∈ r and steps: (z, y) ∈ r* by auto
  from steps
  have ∃ u. (x, u) ∈ r* ∧ (y, u) ∈ r=
  proof (induct)
    case base
    by (rule exI [of - x], insert step, auto)
  next
    case (step y' y)
    from step(3) obtain u where xu: (x, u) ∈ r* and y'u: (y', u) ∈ r= by auto
    from y'u have (y', u) ∈ r ∨ y' = u by auto
    then show ?case
    proof
      assume y'u: y' = u
      with xu step(2) have xy: (x, y) ∈ r* by auto
      show ?thesis
      by (intro exI conjI, rule xy, simp)
    next
    assume (y', u) ∈ r
    with step(2) have uy: (u, y) ∈ 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
    obtain u' where uu': (u, u') ∈ r
    and yu': (y, u') ∈ r by auto
    from xu uu' have xu: (x, u') ∈ r* by auto
    show ?thesis
    by (intro exI conjI, rule xu, insert yu', auto)
then show \((x, y) \in r^+\) by auto

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

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

\[\text{lemma steps-map:}\]
\[
\begin{array}{l}
\text{assumes } fg: \bigwedge u R . P t \implies Q \implies (t, u) \in R \implies P u \land (f t, f u) \in g R \\
\text{and } t: P t \\
\text{and } R: Q R \\
\text{and } S: Q S \\
\text{shows } ((t, u) \in R^* \implies (f t, f u) \in (g R)^*) \\
\land ((t, u) \in R^* \implies O S O R^* \implies (f t, f u) \in (g R)^* O (g S) O (g R)^*) \\
\end{array}
\]
\[
\begin{array}{l}
\text{proof} - \\
\text{fix } t u \\
\text{assume } (t, u) \in R^* \text{ and } P t \\
\text{then have } P u \land (f t, f u) \in (g R)^* \\
\text{proof (induct)} \\
\text{case (step } u v\text{)} \\
\text{from step(3)[OF step(4)]] have } Pu: P u \text{ and steps: } (f t, f u) \in (g R)^* \text{ by auto} \\
\text{from } fg [OF Pu R step(2)] \text{ have } P v \text{ and } step: (f u, f v) \in g R \text{ by auto} \\
\text{with steps have } (f t, f v) \in (g R)^* \text{ by auto} \\
\end{array}
\]

\[61\]
with \( P_v \) show \(?case by simp \\
qed simp \}

\}

note main = this
note maint = main [OF - t]
from maint [of u] have one: \((t, u) \in R^* \implies (f \ t, f \ u) \in (g \ R)^* \) by simp
show \(?thesis
proof (\(rule conjI [OF \ one \ impI]\))
assume \((t, u) \in R^* \implies O \ S \ O \ R^*\)
then obtain \(s \ v\) where \(ts\): \((t, s) \in R^* \) and \(sv\): \((s, v) \in S\) and \(vu\): \((v, u) \in R^*\) by auto

from maint [OF ts] have \(Ps\): \(P \ s\) and \(ts\): \((f \ t, f \ s) \in (g \ R)^*\) by auto
from \(fg\) [OF \ Ps S sv] have \(Pv\): \(P \ v\) and \(sv\): \((f \ s, f \ v) \in g \ S\) by auto
from main [OF vu \ Pv] have \(vu\): \((f \ v, f \ u) \in (g \ R)^*\) by auto

from \(ts \ sv \ vu\) show \((f \ t, f \ u) \in (g \ R)^* \implies O \ g \ S \ O \ (g \ R)^*\) by auto

qed

qed

2.6 Terminating part of a relation

inductive-set
\( SN-part :: \ a \ rel \Rightarrow \ a \ set \)
for \( r :: \ a \ rel \)
where
\( SN-partI\): \(\\forall y. (x, y) \in r \implies y \in SN-part \ r \implies x \in SN-part \ r \)

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

Characterization of \(SN-on\) via terminating part.

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

Special case for “full” termination.

lemma \(SN-SN-part-UNIV-conv\):
\(SN \ r \iff SN-part \ r \ = \ UNIV\)
using \(SN-on-SN-part-conv \ [of \ r \ UNIV]\) by \(auto\)

lemma \(closed-imp-rtrancl-closed\): assumes \(L: L \subseteq A\)
and \(R: R \ " A \subseteq A\)

62
shows \{ t \mid s, s \in L \land (s, t) \in R^\ast \} \subseteq A

proof -
{ fix s t
  assume (s, t) \in R^\ast and s \in L
  hence t \in A
  by (induct, insert L R, auto)
}

thus thesis by auto

qed

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

proof (induct n arbitrary: y)

next

case (Suc n z)

from Suc(2) obtain y where xy: \( (x, y) \in a^\sim n \) and yz: \( (y, z) \in a \) by auto

from Suc(1)(OF xy) obtain m where m: \( m \geq n \) and xy: \( (x, y) \in b^\sim m \) by auto

from yz assms have \( (y, z) \in b^\sim + \) by auto

with k m show \( k > 0 \) unfolding relpow-add by auto

qed auto

lemma relpow-image: assumes \( \forall s t. \ (s, t) \in r \Longrightarrow (f s, f t) \in r' \)
shows \( (s, t) \in r^\sim n \Longrightarrow (f s, f t) \in r'^\sim n \)

proof (induct n arbitrary: t)

next

case (Suc n u)

from Suc(2) obtain t where st: \( (s, t) \in r^\sim n \) and tu: \( (t, u) \in r \) by auto

from Suc(1)(OF st)(OF tu) show thesis by auto

qed auto

lemma relpow-refl-mono:
assumes refl: \( \forall x. \ (x,x) \in \text{Rel} \)
shows \( m \leq n \Longrightarrow (a,b) \in \text{Rel}^\sim m \Longrightarrow (a,b) \in \text{Rel}^\sim n \)

proof (induct rule:dec-induct)

next

case (step i)

hence abi: \( (a, b) \in \text{Rel}^\sim i \) by auto

from refl[of b] abi relpowp-Suc-I[of i \( \lambda x y \) \( (x,y) \in \text{Rel} \)] show \( (a, b) \in \text{Rel}^\sim \) Suc i by auto

qed

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

and IH: \( \forall x. \ \text{SN-on} \ R \ \{x\} \Longrightarrow [\forall y. \ (x, y) \in R \Longrightarrow P y] \Longrightarrow P x \)

shows \( P a \)
proof
from sn SN-on-conv-acc [of R^{-1} a] have a: a ∈ termi R by auto
show ?thesis

proof (rule Wellfounded.acc.induct [OF a, of P], rule IH)
fix x
assume ∃y. (y, x) ∈ R^{-1} ⇒ y ∈ termi R
from this [folded SN-on-conv-acc]
show SN-on R {x} by simp fast
qed auto
qed

lemma partially-localize-CR:
CR r ←→ (∀ x y z. (x, y) ∈ r ∧ (x, z) ∈ r* → (y, z) ∈ join r)

proof
assume CR r
thus ∀ x y z. (x, y) ∈ r ∧ (x, z) ∈ r* → (y, z) ∈ join r by auto

next
assume 1: ∀ x y z. (x, y) ∈ r ∧ (x, z) ∈ r* → (y, z) ∈ join r
show CR r

proof
fix a b c
assume 2: a ∈ UNIV and 3: (a, b) ∈ r* and 4: (a, c) ∈ r*
then obtain n where (a, c) ∈ r^n using rtrancl-is-UNrelpow by fast
with 2 3 show (b, c) ∈ join r
proof (induct n arbitrary: a b c)
case 0 thus ?case by auto
next
case (Suc m)
from Suc(4) obtain d where ad: (a, d) ∈ r^~m and dc: (d, c) ∈ r by auto
from Suc(1) [OF Suc(2) Suc(3)] have (b, d) ∈ join r .
with 1 dc joinE joinI [of b - r c] join-rtrancl-join show ?case by metis
qed
qed

definition strongly-confluent-on :: 'a rel ⇒ 'a set ⇒ bool
where
strongly-confluent-on r A ≡
(∀ x ∈ A. ∀ y z. (x, y) ∈ r ∧ (x, z) ∈ r → (∃ u. (y, u) ∈ r* ∧ (z, u) ∈ r=))

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

lemma strongly-confluent-on-E11:
strongly-confluent-on r A ⇒ x ∈ A ⇒ (x, y) ∈ r ⇒ (x, z) ∈ r ⇒
∃ u. (y, u) ∈ r* ∧ (z, u) ∈ r=

unfolding strongly-confluent-on-def by blast

64
lemma strongly-confluentI [intro]:
\[ \land x\ y\ z. (x, y) \in r \Longrightarrow (x, z) \in r \Longrightarrow \exists u. (y, u) \in r^* \land (z, u) \in r^= \] \Longrightarrow strongly-confluent r

unfolding strongly-confluent-on-def by auto

lemma strongly-confluent-E1n:

assumes scr: strongly-confluent r

shows (x, y) \in r^= \Longrightarrow (x, z) \in r^\sim n \Longrightarrow \exists u. (y, u) \in r^* \land (z, u) \in r^=

proof (induct n arbitrary: x y z)

\begin{itemize}
  \item case (Suc m)
    \begin{itemize}
      \item from Suc(3) obtain w where xw: (x, w) \in r^\sim m and wz: (w, z) \in r by auto
      \item from Suc(1) [OF Suc(2) xw] obtain u where yu: (y, u) \in r^* and uw: (w, u) \in r^= by auto
      \item from strongly-confluent-on-E11 [OF scr, of w] wz yu uw show ?case
      \end{itemize}
    \end{itemize}

  \item { fix x y z
      have (x, y) \in r \Longrightarrow (x, z) \in r^* \Longrightarrow (y, z) \in join r
      \begin{itemize}
        \item by (cases x = y, insert strongly-confluent-E1n [OF assms], blast+)
      \end{itemize}
      \item then show CR r using partially-localize-CR by blast
  }

\end{itemize}

qed auto

lemma strong-confluence-imp-CR:

assumes strongly-confluent r

shows CR r

proof -

\{ fix x y z
  have (x, y) \in r \Longrightarrow (x, z) \in r^* \Longrightarrow (y, z) \in join r
  \begin{itemize}
    \item by (cases x = y, insert strongly-confluent-E1n [OF assms], blast+)
  \end{itemize}
  \item then show CR r using partially-localize-CR by blast
\}

qed

lemma WCR-alt-def: WCR A \iff A^{-1} O A \subseteq A^4 by (auto simp: WCR-defs)

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

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

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

lemma CR-NF-conv:

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

shows (u, t) \in r^!

using assms

unfolding CR-imp-conversionIff-join [OF \langle CR r \rangle]
by (auto simp: NF-iff-no-step normalizability-def)
  (metis (mono-tags) converse-rtranclE joinE)

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

lemma relpow-Suc: $r ^^ Suc n = r O r ^^ n$
using relpow-add[of 1 n r] by auto

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

lemma conversion-mono: $A \subseteq B \implies A^{**} \subseteq B^{**}$
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 $\neg \text{SN-on } R \{s\}$
by (meson SN-on-imp-on-minimal assms)

lemma SN-on-Image-rtrancl-iff [simp]: $\text{SN-on } R (R^* \cdot 'X) \iff \text{SN-on } R X$ (is $?l = ?r$)
proof (intro iffI)
  assume $?l$ show $?r$ by (rule SN-on-subset2[of - ?l], auto)
qed (fact SN-on-Image-rtrancl)

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

lemma rtrancl-O-shift: $(S O R)^* O S = S O (R O S)^*$
proof (intro equalityI subrelI)
  fix $x y$
  assume $(x, y) \in (S O R)^* O S$
  then obtain $n$ where $(x, y) \in (S O R)^{~~n} O S$ by blast
  then show $(x, y) \in S O (R O S)^*$
proof (induct $n$ arbitrary: $y$)
case IH: (Suc n)
then obtain z where xx: (x, z) ∈ (S O R) ∼^n O S and yy: (z, y) ∈ R O S by auto
from IH.hyps[OF xx] yy have (x, y) ∈ S O (R O S)^* O R O S by auto
then show ?case by (fold trancl-unfold-right, auto)
qed auto

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

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

lemma SN-on-subset-SN-terms:
assumes SN: SN-on R X shows X ⊆ {x. SN-on R {x}}
proof (intro subsetI, unfold mem-Collect-eq)
fix x assume x: x ∈ 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 ∪ Y)
using assms by fast

lemma SN-on-UN:
assumes ∀x. SN-on R (X x) shows SN-on R (∪ x. X x)
using assms by fast

lemma Image-subsetI: R ⊆ R' ⇒ R " X ⊆ R' " X by auto

lemma SN-on-O-comm:
assumes SN: SN-on ((R :: ('a × 'b) set) O (S :: ('b × 'a) set)) (S " X)
shows SN-on (S O R) X
proof
fix seq :: nat ⇒ 'b assume seq0: seq 0 ∈ X and chain: chain (S O R) seq
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

67
have seq \( i \in (S \circ R)^* \circ X \)
proof (induct \( i \))
  case 0 from seq0 show \(?case by auto\)
next
  case (Suc \( i \)) with chain have seq (Suc \( i \)) \( \in ((S \circ O \circ R)^* O S O R) \circ X \) by blast
also have ... \( \subseteq (S \circ O \circ R)^* \circ X \) by (fold trancl-unfold-right, auto)
finally show \(?case by auto\).
qed with ia have a \( \in ((S \circ O \circ R)^* O S) \circ X \) by auto
then have a : a \( \in ((R \circ O \circ S)^* S \circ X \) by (auto simp: rtrancl-O-shift)
with ia aSi have False
proof (induct a arbitrary: i rule: SN-on-induct[OF SN])
  case 1 show \(?case by (fact a)\)
next
  case IH: (2 a)
  from chain obtain b
  where \( \ast \): (seq (Suc \( i \)), b) \( \in S (b, \text{seq (Suc} \( i \))) \) \( \in R \) by auto
  with IH have ab \( : (a,b) \in R O S \) by auto
  with \( \text{a} \in (R O S)^* S \circ X \) : have b \( \in ((R O S)^* O R O S) \circ S \circ X \) by auto
  then have b \( \in (R O S)^* S \circ X \)
    by (rule rev-subsetD, intro Image-subsetI, fold trancl-unfold-right, auto)
  from IH.hyps[OF ab * this] IH.prems ab show False by auto
qed

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

lemma chain-mono: assumes R' \( \subseteq \) R chain R' seq shows chain R seq
using assms by auto

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

lemma rtrancl-O-push: S^* O R \( \subseteq \) R O S^*
proof
\{ fix n
  have \( \\land s \ t.\ (s,t) \in S \implies n O R \implies (s,t) \in R O S^* \)
  proof (induct n)
  case (Suc \( n \))
  then obtain \( u \) where \( (s,u) \in S (u,t) \in R O S^* \) unfolding relpow-Suc by blast
  then have \( (s,t) \in S O R O S^* \) by auto
\}

68
also have \( \subseteq R O S^* O S^* \) using push by blast

also have \( \subseteq R O S^* \) by auto

finally show \(?case.\)

qed auto

}\)

thus \(?thesis\) by blast

qed

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

proof (intro equalityI subrelI)

fix \( x \) \( y \)

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

also have \( \subseteq (S^* O R)^* O S^* \) by regexp

finally obtain \( z \) where \( xx: (x,z) \in (S^* O R)^* \) and \( yy: (z,y) \in S^* \) by auto

from \( xx \) have \( (x,z) \in R^* O S^* \) by auto

proof (induct rule: rtrancl-induct)

case (step \( z \ w \))

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

also have \( \subseteq R^* O S^* O R \) by regexp

also have \( \subseteq R^* O R O S^* \) using rtrancl-O-push by auto

also have \( \subseteq R^* O S^* \) by regexp

finally show \(?case.\)

qed auto

with \( yy \) show \( (x,y) \in R^* O S^* \) by auto

qed regexp

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

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

proof

fix \( \text{seq} \)

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

moreover assume \(\text{seq (0::nat)} \in X\)

then have \(\text{seq 0} \in R^* \cdot X\) by auto

ultimately show \(\text{chain (R O S^*) seq \Rightarrow False}\)

proof (induct \(\text{seq 0}\) arbitrary: \(\text{seq}\) rule: \(\text{SN-on-induct}\))

case IH

then have \(01: (\text{seq 0}, \text{seq 1}) \in R O S^*\)

and \(12: (\text{seq 1}, \text{seq 2}) \in R O S^*\)

and \(23: (\text{seq 2}, \text{seq 3}) \in R O S^*\) by (auto simp: eval-nat-numeral)

then obtain \(s t\)

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

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

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

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

from \(t2\) \(23\) have \((t, \text{seq 3}) \in S^* O R O S^*\) by auto

also from \(\text{rtrancl-O-push}\) have \(\subseteq R O S^* O S^*\) by blast

finally have \(t3: (t, \text{seq 3}) \in R O S^*\) by regexp

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

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

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

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

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

lemma relpow-union-cases: \(((a,d) \in (A \cup B) \leftarrow ^n \Longrightarrow (a,d) \in B \leftarrow ^n \vee \exists b c k m. (a,b) \in B \leftarrow ^k \land (b,c) \in A \land (c,d) \in (A \cup B) \leftarrow ^m \land n = \text{Suc } (k + m))\)
proof (induct \(n\) arbitrary: \(a\) \(d\))
  case (Suc \(n\) \(a\) \(e\))
let \( \mathcal{AB} = A \cup B \)
from \( \text{Suc}(2) \) obtain \( b \) where \( ab : (a,b) \in \mathcal{AB} \) and \( be : (b,e) \in \mathcal{AB}^\sim n \) by (rule \( \text{relopow-Suc-E2} \))
from \( ab \)
show \( ?\text{case} \)
proof
  assume \( (a,b) \in A \)
  show \( ?\text{thesis} \)
  proof
    (rule \text{disjI2}, \text{intro exI conjI})
    show \( \text{Suc } n = \text{Suc } (0 + n) \) by \( \text{simp} \)
    show \( (a,b) \in A \) by \( \text{fact} \)
  qed
next
  assume \( ab : (a,b) \in B \)
  from \( \text{Suc}(1)[OF be] \)
  show \( ?\text{thesis} \)
  proof
    assume \( (b,e) \in B ^\sim n \)
    with \( ab \) show \( ?\text{thesis} \)
      by (\text{intro disjI1 relopow-Suc-I2})
  next
  assume \( \exists c \ d \ k \ m. (b, c) \in B ^\sim k \land (c, d) \in A \land (d, e) \in \mathcal{AB} ^\sim m \land n \)
  \( = \text{Suc } (k + m) \)
  then obtain \( c \ d \ k \ m \) where \( (b, c) \in B ^\sim k \) and \( * : (c, d) \in A \) (\( d, e) \in \mathcal{AB} ^\sim m \land n \)
  \( = \text{Suc } (k + m) \) by \( \text{blast} \)
  with \( ab \) have \( ac : (a,c) \in B ^\sim (\text{Suc } k) \) by (intro \( \text{relopow-Suc-I2} \))
  show \( ?\text{thesis} \)
    by (\text{intro disjI2 exI conjI, rule ac, (rule *)}, \text{simp add: *)}
  qed
qed
qed \( \text{simp} \)

\text{lemma } \text{trans-refl-imp-rtrancl-id}:
assumes \( \text{trans } r \ \text{refl } r \)
shows \( r^* = r \)
proof
  show \( r^* \subseteq r \)
  proof
    fix \( x \ y \)
    assume \( (x,y) \in r^* \)
    thus \( (x,y) \in r \)
      by (\text{induct, insert assms, unfold refl-on-def trans-def, blast+})
  qed
qed \( \text{regexp} \)

\text{lemma } \text{trans-refl-imp-O-id}:
assumes \( \text{trans } r \ \text{refl } r \)
shows \( r \ O \ r = r \)
proof (intro \text{equalityI})
show $r \circ O \circ r \subseteq r$ by (fact trans-O-subset[OF assms(1)])
have $r \subseteq r \circ O \circ Id$ by auto
moreover have $Id \subseteq r$ by (fact assms(2)[unfolded refl-O-iff])
ultimately show $r \subseteq r \circ O \circ r$ by auto
qed

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

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

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

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

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

lemma symcl-idemp [simp]: $(r^*)^{++} = r^{++}$ 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 ≡ S^* O R O S^*`

72
definition SN-rel-on :: 'a rel ⇒ 'a rel ⇒ 'a set ⇒ bool where
   SN-rel-on R S T ≡ SN-on (relto R S)

abbreviation SN-rel :: 'a rel ⇒ 'a rel ⇒ bool where
   SN-rel R S ≡ SN-rel-on R S UNIV

abbreviation SN-rel-alt :: 'a rel ⇒ 'a rel ⇒ bool where
   SN-rel-alt R S ≡ SN-rel-on-alt R S UNIV

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

lemma steps-preserve-SN-on-relto:
   assumes steps: ((a, b) ∈ (R ∪ S)∗)
   and SN: SN-on (relto R S) {a}
   shows SN-on (relto R S) {b}
proof
  let ?RS = relto R S
  have (R ∪ S)∗ ⊆ S∗ ∪ ?RS∗ by regexp
  with steps have (a, b) ∈ S∗ ∨ (a, b) ∈ ?RS∗ by auto
  thus ?thesis
  proof
    assume (a, b) ∈ ?RS∗
    from steps-preserve-SN-on[OF this SN] show ?thesis .
  next
    assume Ssteps: (a, b) ∈ S∗
    show ?thesis
    proof
      fix f
      assume f 0 ∈ {b} and chain ?RS f
      hence [f 0 = b and steps: !i. (f i, f (Suc i)) ∈ ?RS] by auto
      let ?g = λ i. if i = 0 then a else f i
      have ¬ 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)) ∈ ?RS
        proof (cases i)
          case (Suc j)
          show ?thesis using steps[of i] unfolding Suc by simp
        next
          case 0
          from steps[of 0, unfolded f0] Ssteps have steps: (a,f (Suc 0)) ∈ S∗ O
          ?RS by blast
    end
  end

73
have \((a, f (SUC 0)) \in RS\) by (rule subsetD[OF - steps], regexp)

thus \(?thesis\) unfolding \(?\) by simp

qed

with \(SN\) show \(False\) by simp

qed

lemma step-preserves-SN-on-relto: assumes \(st: (s, t) \in R \cup E\)

and \(SN: SN\-on (relto R E) \{s\}\)

shows \(SN\-on (relto R E) \{t\}\)

by (rule steps-preserve-SN-on-relto[OF - SN], insert \(st\), auto)

lemma \(SN\-rel-on-imp-SN-rel-on-alt\): \(SN\-rel-on R S T = \Rightarrow SN\-rel-on-alt R S T\)

proof (unfold \(SN\-rel-on-def\))

assume \(SN: SN\-on (relto R S) T\)

show \(?thesis\)

proof (unfold \(SN\-rel-on-alt-def\), intro allI impI)

fix \(f\)

assume steps: chain \((R \cup S) f \land f 0 \in T\)

with \(SN\) have \(SN\: SN\-on (relto R S) \{f 0\}\)

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 \(\neg (INFM j. (f j, f (SUC j)) \in R)\)

proof (rule ccontr)

assume \(\neg \?thesis\)

hence ih: infinitely-many \(r\) unfolding infinitely-many-def \(r\) by blast

obtain \(r\)-index where \(r\)-index = infinitely-many-index \(r\) by simp

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

have \(r\)-index: \(\land i. r (r\-index i) \land r\-index i < r\-index (SUC i) \land (\lor j. r\-index i < j \land j < r\-index (SUC i) \Rightarrow \neg r j)\) by auto

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

{ fix \(i\)

let \(?ri\) = \(r\-index i\)

let \(?rsi\) = \(r\-index (SUC i)\)

from \(r\-index\) have isi: \(?ri < ?rsi\) by auto

obtain \(ri\) \(\text{r\ where}\ ri = ?ri\ and\ rsi = ?rsi\) by auto

with \(r\-index[of i]\) 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 risi: \(ri < rsi\) by simp

{ fix \(n\)

assume Suc \(n \leq rsi - ri\)

hence \((f (\text{SUC ri}), f (\text{SUC (n + ri)})) \in S\) * 

proof (induct \(n\), simp)

qed

qed

qed

qed
Suc j

proof
lemma SN-rel-on-alt-imp-SN-rel-on
qed
auto
proof
show SN-on
assume
qed
let
?prop
from
steps
from
choice
choice
from
hence
∀
}
{
from
this[of rsi – ri – 1] risi have
(f (Suc ri), f rsi) ∈ S∗ by simp
with rsi have ssteps: (f (Suc ?rri), f ?rsi) ∈ S∗ by simp
with r-index[of i] have (f ?ri, f ?rsi) ∈ R O S∗ unfolding r by auto
hence (g i, g (Suc i)) ∈ S∗ O R O S∗ using rtrancl-refl unfolding g by auto
qed
}

hence nSN: ¬ SN-on (S∗ O R O S∗) {g 0} unfolding SN-defs by blast
have SN: SN-on (S∗ O R O S∗) {f (r-index 0)}
proof (rule steps-preserve-SN-on-reltol[OF - SN])
show (f 0, f (r-index 0)) ∈ (R ∪ S)∗
  unfolding rtrancl-fan-conv
   by (rule exI[of - f], rule exI[of - r-index 0], insert steps, auto)
qed
with nSN show False unfolding g ..
qed
qed

lemma SN-rel-on-alt-imp-SN-rel-on: SN-rel-on-alt R S T → SN-rel-on R S T
proof (unfold SN-rel-on-def)
assume SN: SN-rel-on-alt R S T
show SN-on (relto R S) T
proof
fix f
assume start: f 0 ∈ T and chain (relto R S) f
hence steps: ∨ i. (f i, f (Suc i)) ∈ S∗ O R O S∗ by auto
let ?prop = λ i ai bi. (f i, bi) ∈ S∗ ∧ (bi, ai) ∈ R ∧ (af, f (Suc (i))) ∈ S∗
{
  fix i
  from steps obtain bi ai where ?prop i ai bi by blast
  hence ∃ ai bi. ?prop i ai bi by blast
}

hence ∀ i. ∃ ai bi. ?prop i ai bi by blast
from choice[OF this] obtain b where ∀ i. ∃ ai. ?prop i ai (b i) by blast
from choice[OF this] obtain a where steps: ∨ i. ?prop i (a i) (b i) by blast
from steps[of 0] have fa0: (f 0, a 0) ∈ S∗ O R by auto
let ?prop = λ i li. (b i, a i) ∈ R ∧ (∀ j < last li. ((a i ≠ li) ! j, (a i ≠ li) ! Suc j) ∈ S) ∧ last (a i ≠ li) = b (Suc i)
{
  fix i
  from steps[of i] steps[of Suc i] have (a i, f (Suc i)) ∈ S∗ and (f (Suc i), b

75
\[(\text{Suc } i) \in S^* \text{ by } \text{auto} \]

from \text{rtrancl-trans[OF this]} steps[of i] have \( R: (b \ i, a \ i) \in R \text{ and } S: (a \ i, b \text{Suc } i) \in S^* \text{ by } \text{blast+} \)

from \text{S[unfolded rtrancl-list-conv]} obtain \( li \) where \( \text{last (a \ i \# li)} = b \text{ (Suc } i) \land \forall j < \text{length li. (a \ i \# li) ! j, (a \ i \# li) ! (Suc } j) \in S \) ..

with \( R \) have \( \text{?prop } li \) by \text{blast}

hence \( \exists li. \ ?prop li \) ..

hence \( \forall i. \exists li. \ ?prop i li \) ..

from \text{choice[OF this]} obtain \( l \) where \( \text{steps: } \forall i. \ ?prop (l i) \) by \text{auto}

let \( ?p = \lambda i. \ ?prop i (l i) \) from \text{steps have steps: } \forall i. \ ?p i \) by \text{blast}

let \( ?l = \lambda i. \ a i \# l i \)

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

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

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

have \( g0: g \ 0 = a \ 0 \text{ unfolding } g \text{ Let-def by simp} \)

with \( fa0 \) have \( fg0: (f \ 0, g \ 0) \in S^* O R \text{ by auto} \)

have \( fg0: (f \ 0, g \ 0) \in (R \cup S)^* \text{ by } (\text{rule subsetD[OF - fg0], regexp}) \)

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

proof –

fix \( i \ j \ n \)

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

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

proof (cases \( n \))

case \( \emptyset \)

with \( n \) have \( j = 0 \) by \text{auto}

thus \( \text{thesis by simp} \)

next

case \( (\text{Suc } nn) \)

obtain \( ii jj \) where \( \text{nn: } ?g \ nn = (ii, jj) \) by (cases \( ?g \ nn, \text{auto} \))

show \( \text{thesis} \)

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

case \( \text{True} \)

with \( \text{nn } Suc \) have \( ?g \ nn = (ii, \text{Suc } jj) \) by \text{auto}

with \( \text{nn } \text{True } show \ ?\text{thesis by simp} \)

next

case \( \text{False} \)

with \( \text{nn } Suc \) have \( ?g \ nn = (\text{Suc } ii, 0) \) by \text{auto}

with \( \text{nn } show \ ?\text{thesis by simp} \)

qed

qed

have \( g\text{steps: } \forall i. (g i, g (\text{Suc } i)) \in R \cup S \)

proof –

fix \( n \)

obtain \( i j \) where \( ?g n = (i, j) \) by (cases \( ?g \ n, \text{auto} \))

show \( (g n, g (\text{Suc } n)) \in R \cup S \)
proof (cases Suc \( j \) < length \( (?l \ i) \))

  case True
  with \( n \) have \( ?g \ (Suc \ n) = (i, Suc \ j) \) by auto
  with \( n \) have \( gn: g \ n = {?l \ i} \ {j} \) and \( gsn: g \ (Suc \ n) = {?l \ i} \ (Suc \ j) \) unfolding
  \( g \) by auto
  thus \( ?thesis \) using \( \text{steps[of}\ ?i} \) True by auto
next
  case False
  with \( n \) have \( ?g \ (Suc \ n) = (Suc \ i, 0) \) by auto
  with \( n \) have \( gn: g \ n = last \ ( {?l \ i}) \) using \( \text{last-conv-nth[of}\ {?l \ i}} \) by auto
  from \( gn \ gsn \) show \( ?thesis \) using \( \text{steps[of}\ ?i} \) \( \text{steps[of Suc} \ ?i} \) by auto
qed

have \( \text{infR: INFM} \ j. \ (g \ j, g \ (Suc \ j)) \in R \) unfolding \( \text{INFM-nat-le} \)
proof
fix \( n \)
  obtain \( i \ j \) where \( n: ?g \ n = (i,j) \) by (cases \( ?g \ n, \) auto)
  from \( \text{len[OF} \ n \) have \( j: j < {?l \ i} \).
  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 \( \text{inf-concat-simple-add[OF} \ n, \ of \ ?k, \ OF \ k3 \)
  have \( gnk: ?g \ (n + {?k}) = (i, k) \) by (simp only: \( k) \)
  hence \( gnk2: g \ (n + {?k}) = last \ ( {?l} \ i) \) using \( \text{last-conv-nth[of}\ {?l} \ i} \)
  \( k2 \) by auto
  hence \( gsnk2: g \ (Suc \ (n+?k)) = (Suc \ i, 0) \) by auto
  from \( k2 \ gnk \) have \( ?g \ (Suc \ (n+?k)) = (Suc \ i, 0) \) by auto
  hence \( gsnk2: g \ (Suc \ (n+?k)) = a \ (Suc \ i) \) unfolding \( g \) by auto
  from \( \text{steps[of}\ ?i} \) \( \text{steps[of Suc} \ ?i} \) have main: \( g \ (n+?k), g \ (Suc \ (n+?k)) \) \( R \)
  by (simp only: \( gnk2 \ gsnk2 \)
  show \( \exists \ j \geq n. \ (g \ j, g \ (Suc \ j)) \in R \)
  by (rule \( exI[of- \ n + {?k}, \ auto \ simp: \text{main[simplified]}] \)
qed

from \( fg\!\![\text{unfolded rtrancl-fan-conv}] \) obtain \( gg \ n \) where \( \text{start:} gg \ 0 = f \ 0 \)
and \( n: gg \ n = g \ 0 \) and \( \text{steps:} \ \bigwedge \ i. \ i < n \implies (gg \ i, gg \ (Suc \ i)) \in R \cup S \)
by auto

let \( ?h = \lambda i. \text{if} \ i < n \text{ then} \ gg \ i \text{ else} \ g \ (i - n) \)
obtain \( h \) where \( h: ?h \) by auto

\{ fix \( i \)
  assume \( i: i \leq n \)
  have \( h = gg \ i \) using \( i \) unfolding \( h \)
  by (cases \( i < n, \auto \ simp: \ n) \)
\} note \( gg = \text{this} \)
from \( gg[\text{of} \ 0] \): \( f \) \( 0 \in T \) have \( h0: h \ 0 \in T \) unfolding \( \text{start by auto} \)
\{ fix \( i \)

77
have \((h, h \text{ Suc } i)) \in R \cup S\)

proof (cases \(i < n\))

  case True
  from steps[of \(i\)] gg[of \(i\)] gg[of \(\text{Suc } i\)] True show thesis by auto

next

  case False
  hence \(i = n + (i - n)\) by auto
  then obtain \(k\) where \(i \cdot i = n + k\) 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]

have \(\neg (\text{INFM } j. (h, h \text{ Suc } j)) \in R \cdot \)

moreover have \(\text{INFM } j. (h, h \text{ Suc } j)) \in R\) unfolding \(\text{INFM-nat-le}\)

proof (rule)

  fix \(m\)
  from infR[unfolded \(\text{INFM-nat-le}, \text{rule-format}, \text{of m}\)]
  obtain \(i\) where \(i \cdot i \geq m\) and \(g : (g, g \text{ Suc } i)) \in R\) by auto
  show \(\exists n \geq m. (h, h \text{ Suc } n)) \in R\)
    by (rule exI[of \(- i + n\)], unfold \(h\), insert \(g\), auto)

qed

ultimately show False ..

qed

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

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 \(R : R \subseteq R'\) and \(S : S \subseteq R' \cup S'\) and \(\text{SN}: \text{SN-rel-on } R' S' T\)
  shows \(\text{SN-rel-on } R S T\)

proof -

  note conv = SN-rel-on-conv SN-rel-on-alt-def INFM-nat-le
  show thesis unfolding conv

  proof(intro allI impI)
    fix \(f\)
    assume chain \((R \cup S)\) \(f \wedge f 0 \in T\)
    with \(R S\) have chain \((R' \cup S')\) \(f \wedge f 0 \in T\) by auto
    from SN[unfolded conv, rule-format, OF this]
show \(\neg (\forall m. \exists n \geq m. (f n, f (Suc n)) \in R)\) using \(R\) by auto
qed

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

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

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

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

hence \(\forall i. (f i, f (Suc i)) \in \text{relto } R S\) by blast
thus \(\text{False}\) using \(\text{assms } f0\)
unfolding \(\text{SN-rel-on-def SN-defs by blast}\)
qed

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

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

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

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

from \(\text{?L[unfolded SN-rel-on-cone SN-rel-on-alt-def, THEN spec[of - as]]}\)
\(\text{id T obtain i where i: } i \land j. j \geq i \implies (\text{as } j, \text{ as } (\text{Suc } j)) \notin R\) by auto
with \(\text{id[unfolded INFM-nat, THEN conjunct2, THEN spec[of - Suc i]} show}\)

79
False by auto

qed

next

assume ?R

show ?L

proof (intro allI impI)

fix as

assume chain (R ∪ S) as ∧ as 0 ∈ T

with (\R: unfolded ideriv-def) have \neg (INFM i. (as i, as (Suc i)) ∈ R) by auto

from this[unfolded INFM-nat] obtain i where i: (∀ j. i < j ⇒ (as j, as (Suc j)) /∈ R) 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 \thetaesis

proof (unfold SN-rel-on-alt-def, intro allI impI)

fix f

presume steps: chain (R ∪ S) f

obtain r where r: (∀ j. r j ≡ (f j, f (Suc j)) ∈ R) by auto

show \neg (INFM j. (as j, as (Suc j)) ∈ R) unfolding INFM-nat using i by blast

qed

qed
proof (induct n, simp)
case (Suc n)
hence stepps: (f (Suc ri), f (Suc (n+ri))) ∈ S^∗ by simp
have (f (Suc (n+ri)), f (Suc (Suc n + ri))) ∈ S
  using inter[of Suc n + ri] Suc(2) by auto
with stepps show ?case by simp
qed

\{ from this[of rsi - ri - 1] risi have
  (f (Suc ri), f rsi) ∈ S^∗ by simp
with ri rsi have ssteps: (f (Suc ?ri), f ?rsi) ∈ S^∗ by simp
with r-index[of i] have (f ?ri, f ?rsi) ∈ R O S^∗ unfolding r by auto
hence (g i, g (Suc i)) ∈ S^∗ O R O S^∗ using rtrancl-refl unfolding g by auto
\}

hence ¬ SN (S^∗ O R O S^∗) unfolding SN-defs by blast
with SN show False by simp
qed
qed simp

lemma SN-rel-alt-to-SN-rel : SN-rel-alt R S \implies SN-rel R S
proof (unfold SN-rel-on-def)
assume SN : SN-rel-alt R S
show SN (relto R S)
proof
fix f
assume chain (relto R S) f
hence steps: \(\forall i. (f i, f (Suc i)) \in S^∗ O R O S^∗\) by auto
let ?prop = \(\lambda i ai bi. (f i, bi) \in S^∗ \land (ai, f (Suc i)) \in S^∗\)
\{ fix i
  from steps obtain bi ai where ?prop i ai bi by blast
  hence \(\exists ai bi. ?prop i ai bi\) by blast
\}

hence \(\forall i. \exists ai bi. ?prop i ai bi\) by blast
from choice[of this] obtain b where \(\forall i. \exists ai. ?prop i ai (b i)\) by blast
from choice[of this] obtain a where steps: \(\land i. ?prop i (a i) (b i)\) by blast
let ?prop = \(\lambda i li. (b i, a i) \in R \land (\forall j < length li. ((a i \# li) \land j, (a i \# li) \land Suc j) \in S) \land (Suc i)\)
Suc j) \in S) \land Suc i)
\{ fix i
  from steps[of i] steps[of Suc i] have (a i, f (Suc i)) ∈ S^∗ and (f (Suc i), b (Suc i)) ∈ S^∗ by auto
  from rtrancl-trans[of this] steps[of i] have R: (b i, a i) ∈ R and S: (a i, b (Suc i)) ∈ S^∗ by blast+
  from S[unfolded rtrancl-list-conv] obtain li where last (a i \# li) = b (Suc i) \land (\forall j < length li. ((a i \# li) \land j, (a i \# li) \land Suc j) \in S) ..
  \{ with R have ?prop i li by blast

81
hence $\exists l. \ ?prop i \ l$ ..

\[
\begin{align*}
\text{hence } & \forall \ i. \ ?prop \ i \ l \ .. \\
\text{from } & \text{choice[OF this] obtain } l \text{ where steps: } \land \ i. \ ?prop \ i \ (l \ i) \text{ by auto} \\
\text{let } & ?p = \lambda i. \ ?prop \ i \ (l \ i) \\
\text{from } & \text{steps have steps: } \land \ i. \ ?p \ i \text{ by blast} \\
\text{let } & ?l' = \lambda i. \ \text{length } (l \ i) \\
\text{let } & ?g = \lambda i. \ \text{inf-concat-simple } l' \ i \\
\end{align*}
\]

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

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

proof –

fix $i j n$
assume $n: \ ?g \ n = (i, j)$
show $j < \text{length } (l \ i)$
proof (cases $n$)
  case $0$
  with $n$ have $j = 0$ by auto
  thus $\text{thesis by simp}$
next
  case $(\text{Suc } nn)$
  obtain $ii jj$ where $nn: \ ?g \ nn = (ii, jj)$ by (cases $\ ?g \ nn$, auto)
  show $\text{thesis}$
  proof (cases Suc $jj < \text{length } (l \ ii)$)
    case True
    with $nn \ \text{Suc}$ have $?g \ n = (ii, \ \text{Suc } jj)$ by auto
    with $n \ \text{True}$ show $\text{thesis by simp}$
  next
    case False
    with $nn \ \text{Suc}$ have $?g \ n = (\text{Suc } ii, 0)$ by auto
    with $n$ show $\text{thesis by simp}$
  qed
qed

have $g$steps: $\land i. \ (g \ i, g \ (\text{Suc } i)) \in R \cup S$
proof –

fix $n$

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

show $(g \ n, g \ (\text{Suc } n)) \in R \cup S$
proof (cases Suc $j < \text{length } (l \ i)$)
  case True
  with $n$ have $?g \ (\text{Suc } n) = (i, \ \text{Suc } j)$ by auto
  with $n$ have $gn: \ ?l \ i ! j \text{ and } gsn: \ g \ (\text{Suc } n) = ?l \ i ! (\text{Suc } j)$ unfolding $g$ by auto
  thus $\text{thesis using steps[of i] True by auto}$
next
  case False
  with $n$ have $?g \ (\text{Suc } n) = (\text{Suc } i, 0)$ by auto
  with $n$ have $gn: \ ?l \ i ! j \text{ and } gsn: \ g \ (\text{Suc } n) = ?l \ i ! (\text{Suc } i)$ unfolding $g$ by auto

82
g by auto

from gn len[OF \n] False have \n = length (?l \i) - 1 by auto

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

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

qed

qed

have infR: \thesis j. (g \j, g (Suc \j)) \in R unfolding \thesis-nat-le

proof

fix \n

obtain \i \j \n where \i \j = \thesis \n by (cases \thesis \n, auto)

from len[OF \n] have \n: \j < \thesis \i .

let \k = \thesis \i' - 1 - \j

obtain \k where \k = \j + \?k by auto

from \j \k have \k: \k = \thesis \i' - 1 and \k3: \j + \?k < \thesis \i' \i by auto

from inf-concat-simple-add[OF \n, of \?k, OF \k3]

have gnk: \thesis (\n + \?k) = (\i, \j) by (simp only: \k)

hence \thesis (\n + \?k) = \i \j ! \k unfolding g by auto

hence gnk2: \thesis (\n + \?k) = last (?l \i) using last-conv-nth[of \?
 \i] \k2 by auto

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

hence gnsk2: \thesis (Suc (\n + \?k)) = a (Suc \i) unfolding g by auto

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

by (simp only: gnk2 gnsk2)

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

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

qed

from SN[unfolded SN-rel-on-alt-def] gsteps infR show False by blast

qed

lemma SN-rel-alt-r-empty : SN-rel {} S

unfolding SN-rel-defs by auto

lemma SN-rel-alt-s-empty : SN-rel-alt R {} = SN R

unfolding SN-rel-defs SN-defs by auto

lemma SN-rel-mono':

R \subseteq R' \implies S \subseteq R' \cup S' \implies SN-rel R' S' \implies SN-rel R S

unfolding SN-rel-on-conv SN-rel-defs INFM-nat-le

by (metis contra-subsetD sup.left-idem sup.mono)

lemma SN-rel-mono:

assumes R: R \subseteq R' and S: S \subseteq S' and SN: SN-rel R' S'

shows SN R S

using SN unfolding SN-rel-defs using SN-subset[OF relto-mono[OF R S]] by blast

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

lemma SN-rel-imp-SN : assumes SN-rel R S shows SN R
proof
  fix f
  assume ∀ i. (f i, f (Suc i)) ∈ R
  hence ∀ i. (f i, f (Suc i)) ∈ relto R S by blast
  thus False using assms unfolding SN-rel-defs SN-defs by fast
qed

lemma relto-trancl-conv : (relto R S)˘+ = ((R ∪ S))˘ O R O ((R ∪ S))˘ by regexp

lemma SN-rel-Id:
  shows SN-rel R (S ∪ Id) = SN-rel R S
  unfolding SN-rel-defs by (simp only: relto-Id)

lemma relto-rtrancl: relto R (S˘) = relto R S by regexp

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

lemma SN-rel-ideriv: SN-rel R S = (¬ (∃ as. ideriv R S as)) (is ?L = ?R)
proof
  assume ?L
  show ?R
proof
    assume ∃ as. ideriv R S as
    then obtain as where id = id[unfolded ideriv-def]
    from ?L[unfolded relto-on-conv SN-rel-defs, THEN spec[of - as]]
      id obtain i where i: j. j ≥ i ⇒ (as j, as (Suc j)) /∈ 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 relto-on-conv SN-rel-defs
  proof (intro allI impI)
    fix as
    presume chain (R ∪ S) as
    with ?R[unfolded ideriv-def] have ¬ (INFM i. (as i, as (Suc i)) ∈ R) by auto
    from this[unfolded INFM-nat] obtain i where i: j. j < i ⇒ (as j, as (Suc j)) /∈ R by auto
    show ¬ (INFM j. (as j, as (Suc j)) ∈ R) unfolding INFM-nat using i by blast
    qed simp
  qed

lemma SN-rel-map:
fixes $R \mathrel{\r W} R' \mathrel{\r W'}$ :: 'a rel
defines $A : A \equiv R' \cup Rw'$
assumes $SN: SN-\mathrel{rel} R' \mathrel{\r W'}$
and $R: \forall s \mathrel{\r t}. \mathrel{(s,t)} \in R \implies (f s, f t) \in A^* O R' O A^*$
and $Rw: \forall s \mathrel{\r t}. \mathrel{(s,t)} \in Rw \implies (f s, f t) \in A^*$
shows $SN-\mathrel{rel} R \mathrel{\r W}$

unfolding $SN-\mathrel{rel}-defs$

proof
fix $g$
assume steps: chain $(\mathrel{relto} R \mathrel{\r W}) g$
let $\lambda = \lambda i. (f (g i))$
obtain $h$ where $h = \lambda$ by auto
{
fix $i$
let $\lambda m = \lambda (x,y). (f x, f y)$
{
fix $s\ t$
assume $(s,t) \in Rw^*$
hence $\lambda m (s,t) \in A^*$
proof (induct)
  case base show $\lambda$ by simp
next
  case (step $t\ u$)
  from $Rw[OF\ step(2)]\ step(3)$
  show $\lambda$ by auto
qed
}

note $Rw = this$
from steps have $(g\ i, g (Suc\ i)) \in \mathrel{relto} R \mathrel{\r W} ..$
from this
obtain $s\ t$ where $gs: (g\ i, s) \in Rw^*$ and $st: (s,t) \in R$ and $tg: (t, g (Suc\ i)) \in Rw^*$

by auto
from $Rw[OF\ gs]\ R[OF\ st]\ Rw[OF\ tg]$
have step: $(\lambda f i, \lambda f (Suc\ i)) \in A^* O (A^* O (A^* O R' O A^*)) O A^*$
  by fast
have $(\lambda f i, \lambda f (Suc\ i)) \in A^* O R' O A^*$
  by (rule subsetD[OF - step], regexp)
hence $(h\ i, h (Suc\ i)) \in (\mathrel{relto} R' \mathrel{\r W'})^+$
  unfolding $A h\ relto-trancl-conv$ .
}
hence $SN ((\mathrel{relto} R' \mathrel{\r W'})^+)$ by auto
with $SN-imp-SN-trancl[OF\ SN[unfolded\ SN-\mathrel{rel-on-}def]]$
show False by simp
qed

datatype $SN-\mathrel{rel-ext-type} = top-s | top-ns | normal-s | normal-ns$

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

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

lemma SN-rel-ext-mono: assumes subset: P ⊆ P' Pw ⊆ Pw' R ⊆ R' Rw ⊆ Rw' and
SN: SN-rel-ext P' Pw' R' Rw' M shows SN-rel-ext P Pw R Rw M
using SN-rel-ext-step-mono[OF subset] SN unfolding SN-rel-ext-def by blast

lemma SN-rel-ext-trans:
  fixes P Pw R Rw :: 'a rel and M :: 'a ⇒ bool
  defines M' : M' ≡ {(s,t). M t}
  defines A: A ≡ (P ∪ Pw ∪ R ∪ Rw) ∩ M'
  assumes SN-rel-ext P Pw R Rw M
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}
  let ?R' = ?UnM {top-s, normal-s}
  let ?Rw' = ?UnM {top-s, top-ns, normal-s, normal-ns}
  have A: A = ?A unfolding A by auto

86
have \( P: (P \cap M') = ?P' \) by auto
have \( Pw: (P \cup Pw) \cap M' = ?Pw' \) by auto
have \( R: (P \cup R) \cap M' = ?R' \) by auto
have \( Rw: A = ?Rw' \) unfolding \( A \).

\{  
  fix \ s \ t tt  
  assume \( m: M s \) and \( st: (s, t) \in ?UnM tt \)  
  hence \( \exists \ typ \in tt. (s, t) \in ?rel ty \land M s \land M t \) unfolding \( M' \) by auto  
\} note one-step = this

let \( \bar{seq} = \lambda s t n ty. s = g 0 \land t = g n \land (\forall \ i < n. (g \ i, g (Suc \ i)) \in ?rel (ty \ i)) \land (\forall \ i \leq n. M (g \ i)) \)

\{  
  fix \ s \ t  
  assume \( m: M s \) and \( st: (s, t) \in A^* \)  
  from \( st[unfolded \ rtrancl-fun-conv] \) obtain \( g n \) where \( g0: g 0 = s \) and \( gn: g n = t \) and \( steps: \bigwedge \ i. i < n \implies (g \ i, g (Suc \ i)) \in ?A \) unfolding \( A \) by auto  
\} fix \ i

assume \( i < n \) hence \( i': i \leq n \) by auto

from \( i' \) one-step[\( OF \ min \ steps[OF \ i'] \)] have \( \exists \ ty. (g \ i, g (Suc \ i)) \in ?rel ty \) by blast

\{  
  hence \( \forall \ i. (\exists ty. i < n \implies (g \ i, g (Suc \ i)) \in ?rel ty) \) by auto  
  from \( choice[OF \ this] \) obtain \( tt \) where \( steps: \bigwedge \ i. i < n \implies (g \ i, g (Suc \ i)) \in ?rel (tt \ i) \) by auto  
  from \( g0 \) \( gn \) \( steps \) \( min \) have \( ?seq s t n tt \) by auto  
  hence \( \exists \ g n tt. ?seq s t g n tt \) by blast  
\} note A-steps = this

let \( ?seqtt = \lambda s t g n ty. s = g 0 \land t = g n \land n > 0 \land (\forall \ i < n. (g \ i, g (Suc \ i)) \in ?rel (ty \ i)) \land (\forall \ i \leq n. M (g \ i)) \land (\exists \ i < n. ty \ i \in tt) \)

\{  
  fix \ s \ t tt  
  assume \( m: M s \) and \( st: (s, t) \in A^* \)  
  then obtain \( u v \) where \( su: (s, u) \in A^* \) and \( uv: (u, v) \in ?UnM tt \) and \( vt: 

87
\[(v, t) \in A^*\]

by auto

from \textit{A-steps} \(\text{OF} m \text{ su}\) obtain \(g1 \ n1 \ ty1\) where \(seq1\): \(?seq\ s \ u \ g1 \ n1 \ ty1\) by auto

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

from \textit{A-steps} \(\text{OF this vt}\) obtain \(g2 \ n2 \ ty2\) where \(seq2\): \(?seq\ v \ t \ g2 \ n2 \ ty2\) by auto

from \(seq1\) have \(M\ u\) by auto

from one-step \(\text{OF this uv}\) obtain \(ty\) where \(ty\): \(ty \in tt\) and \(uv\): \((u, v) \in \text{?rel}\)

let \(?g\) = \(\lambda i.\) if \(i \leq n1\) then \(g1\ i\) else \(g2\ (i - (Suc\ n1))\)

let \(?ty\) = \(\lambda i.\) if \(i < n1\) then \(ty1\ i\) else if \(i = n1\) then \(ty\) else \(ty2\ (i - (Suc\ n1))\)

let \(?n\) = \(Suc\ (n1 + n2)\)

have \(ex\): \(\exists i < ?n. \ ?ty\ i \in tt\)

by \((\text{rule exI} [of - n1], \text{simp add}: ty)\)

have \(\text{steps}\): \(\forall i < ?n. \ (?g\ i, \ ?g\ (Suc\ i)) \in \text{?rel} \ (?ty\ i)\)

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

fix \(i\)

assume \(i < ?n\)

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

proof \((\text{cases } i \leq n1)\)

\begin{cases}
  \text{case True} & \\
  \text{case False} & \\
\end{cases}

hence \(i = Suc\ n1 + (i - Suc\ n1)\) by auto

then obtain \(k\) where \(i = Suc\ n1 + k\) by auto

with \((i < ?n)\) have \(k < n2\) by auto

thus \(?thesis\) using \(seq2\) unfolding \(i\) by auto

qed

qed

from \textit{steps seq1 seq2 ex}

have \(seq\): \(?seqtt\ s \ t \ tt \ ?g \ ?n \ ?ty\) by auto

have \(\exists n ty. \ ?seqtt\ s \ t \ tt \ g \ n \ ty\)

by \((\text{intro exI, rule seq})\)

\{ note \(A-tt-A = this\)

let \(?tycon = \lambda ty1 ty2 tt ty'\ n. \ ty1 = ty2 \rightarrow (\exists i < n. \ ty' i \in tt)\)

let \(?seqt = \lambda i ty g n ty'. \ f i = g 0 \land f (Suc\ i) = g n \land (\forall j < n. \ (g j, g (Suc\ j))) \in \text{?rel} \ (?ty'\ j)\) \land (\forall j \leq n. \ M\ (g j))\)

\begin{align*}
  \wedge (\text{?tycon (ty i) top-s {top-s} ty' n}) \\
  \wedge (\text{?tycon (ty i) top-ns {top-s,top-ns} ty' n}) \\
  \wedge (\text{?lycon (ty i) normal-s {top-s,normal-s} ty' n})
\end{align*}

\}

\{ fix \(i\)

have \(\exists n ty'. \ ?seqt\ i ty g n ty'\)

proof \((\text{cases } ty\ i)\)

\begin{cases}
  \text{case top-s} & \\
  \text{case top-s} & \\
\end{cases}

from \textit{steps[of i, unfolded top-s]}

have \((f i, f (Suc\ i)) \in \text{?P}\) by auto
from A-tt-A[OF min this[unfolded P]]
show ?thesis unfolding top-s by auto
next
case top-ns
from steps[of i, unfolded top-ns]
have (f i, f (Suc i)) ∈ ?Pw by auto
from A-tt-A[OF min this[unfolded Pw]]
show ?thesis unfolding top-ns by auto
next
case normal-s
from steps[of i, unfolded normal-s]
have (f i, f (Suc i)) ∈ ?R by auto
from A-tt-A[OF min this[unfolded R]]
show ?thesis unfolding normal-s by auto
next
case normal-ns
from steps[of i, unfolded normal-ns]
have (f i, f (Suc i)) ∈ ?Rw by auto
from A-steps[OF min this]
show ?thesis unfolding normal-ns by auto
qed

hence ∀ i. ∃ g n ty′. ?seqt i ty g n ty′ by auto
from choice[OF this] obtain g where ∀ i. ∃ n ty′. ?seqt i ty (g i) n ty′ by auto
from choice[OF this] obtain n where ∀ i. ∃ ty′ i. ?seqt i ty (g i) (n i) ty′ by auto
from choice[OF this] obtain ty′ where ∀ i. ?seqt i ty (g i) (n i) (ty′ i) by auto
hence partial: ∀ i. ∀ n ty′. ?seqt i ty (g i) (n i) (ty′ i) ..

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

note bounds = inf-concat-bounds[OF inf]
note inf-Suc = inf-concat-Suc[OF inf]
note inf-mono = inf-concat-mono[OF inf]
have ¬ SN-rel-ext P Pw R Rw M
  unfolding SN-rel-ext-def simp-thms
proof (rule exI[of - ?g], rule exI[of - ?ty], intro conjI allI)
fix k
obtain i j where ik: ?ind k = (i,j) by force
from bounds[OF this] have j: j < n i by auto
show M (?g k) unfolding ik using partial[of i] j by auto
next
fix k
obtain i j where ik: ?ind k = (i,j) by force
from bounds[OF this] have j: j < n i by auto
from partial[of i] j have step: (g i j, g i (Suc j)) ∈ ?rel (ty' i j) by auto
obtain i' j' where isk: ?ind (Suc k) = (i',j') by force
have i'j': g i' j' = g i (Suc j)
proof (rule inf-Suc[OF - ik isk])
fix i
from partial[of i]
have g i (n i) = f (Suc i) by simp
also have ... = g (Suc i) 0 using partial[of Suc i] by simp
finally show g i (n i) = g (Suc i) 0 .
qed
show (?g k, ?g (Suc k)) ∈ ?rel (?ty k)
unfolding ik isk split i'j'
by (rule step)
next
show INFM i. ?ty i ∈ {top-s, top-ns}
unfolding INFM-nat-le
proof (intro allI)
fix k
obtain i j where ik: ?ind k = (i,j) by force
from inf1[unfolded INFM-nat] obtain i' 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 ik' i'] have k: k ≤ k' by simp
show ∃ k' ≥ k. ?ty k' ∈ {top-s, top-ns}
  by (intro exI conjI, rule k, unfold ik' split, rule ty')
qed
next
show INFM i. ?ty i ∈ {top-s, normal-s}
unfolding INFM-nat-le
proof (intro allI)
fix k
obtain i j where ik: ?ind k = (i,j) by force
from inf2[unfolded INFM-nat] obtain i' 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
from inf-concat-surj[of - n, OF j'] obtain k' where ik': ?ind k' = (i'j') ..
from inf-mono[OF ik ik' i'] have k: k ≤ k' by simp
\begin{verbatim}
show \( \exists k'. k \geq k. \forall ty k'. \{\text{top-s, normal-s}\} \)
  by (intro exI conjI, rule k, unfold ik' split, rule ty')
qued
qed
with \text{assms show} \ False \ by \ auto
qed

\textbf{lemma} \ SN-rel-ext-map: \text{fixes} \ P Pw R Rw P' Pw' R' Rw' :: \ 'a \ rel \ and \ M M' :: \ 'a \Rightarrow \ bool
\text{defines} \ Ms \:: \ \text{Ms} \equiv \{(s,t). M'(t)\}
\text{defines} \ A :: A \equiv (P' \cup Pw' \cup R' \cup Rw') \cap \text{Ms}
\text{assumes} \ SN :: \text{SN-rel-ext} P Pw R Rw M'
\and \ P :: \bigwedge_{s t} M s \Rightarrow M t \Rightarrow (s, t) \in P \Rightarrow (f s, f t) \in (A^\sim \circ \text{O} (P' \cap \text{Ms}) \circ O (A^\sim)) \cap I t
\and \ Pw :: \bigwedge_{s t} M s \Rightarrow M t \Rightarrow (s, t) \in Pw \Rightarrow (f s, f t) \in (A^\sim \circ \text{O} ((P' \cup Pw') \cap \text{Ms}) \circ O (A^\sim)) \cap I t
\and \ R :: \bigwedge_{s t} M s \Rightarrow M t \Rightarrow (s, t) \in R \Rightarrow (f s, f t) \in (A^\sim \circ \text{O} ((P' \cup R') \cap \text{Ms}) \circ O (A^\sim)) \cap I t
\and \ Rw :: \bigwedge_{s t} M s \Rightarrow M t \Rightarrow (s, t) \in Rw \Rightarrow (f s, f t) \in (A^\sim \cap I t)
\text{shows} \ SN-rel-ext P Pw R Rw M
\text{proof} –
\text{note} \ SN \equiv \text{SN-rel-ext-trans[OF \ SN]}
\let \?P = (A^\sim \circ \text{O} (P' \cap \text{Ms}) \circ O (A^\sim))
\let \?Pw = (A^\sim \circ \text{O} ((P' \cup Pw') \cap \text{Ms}) \circ O (A^\sim))
\let \?R = (A^\sim \circ \text{O} ((P' \cup R') \cap \text{Ms}) \circ O (A^\sim))
\let \?Rw = (A^\sim \cap I t)
\let \?relt = \text{SN-rel-ext-step} ?P ?Pw ?R ?Rw
\let \?rel = \text{SN-rel-ext-step} P Pw R Rw
\text{show} \ \?thesis
\text{proof} \ (\text{rule ccontr})
\text{assume} \ \neg \ ?thesis
\text{from} \ this[unfolded \ SN-rel-ext-def]
\text{obtain} \ g ty
\text{where} \ \text{steps}: \bigwedge i. (g i, g (Suc i)) \in ?rel (ty i)
\text{and} \ \text{min}: \bigwedge i. M (g i)
\text{and} \ \text{inf1}: \text{INFM i. ty i} \in \{\text{top-s, top-ns}\}
\text{and} \ \text{inf2}: \text{INFM i. ty i} \in \{\text{top-s, normal-s}\}
\text{by} \ auto
\text{from} \ \text{inf1[unfolded \ INFM-nat]} \ \text{obtain} \ k \ \text{where} \ k: ty k \in \{\text{top-s, top-ns}\} \ \text{by} \ auto
\let \?k = \text{Suc} k
\let \?i = \text{shift} id \ ?k
\let \?f = \lambda i. f (\text{shift} g \ ?k i)
\let \?ty = \text{shift} ty \ ?k
{\text{fix} i
\text{assume} ty: ty i \in \{\text{top-s, top-ns}\}
\text{note} m = \text{min[of} \ i\]}
\end{verbatim}
\textbf{note} \( ms = \min[\text{of } Suc \ i] \)

from \( P[\text{OF } m \ ms] \)

\( Pw[\text{OF } m \ ms] \)

\( \text{steps}[\text{of } i] \)

\( ty \)

have \((f \ (g \ i), f \ (g \ (Suc \ i))) \in \ richTextBox\ (ty \ i) \land I \ (g \ (Suc \ i)) \)

by (cases \( ty \ i, \ auto) \)

\} \textbf{note} \( stepsP = \ this \)

\{ 

fix \( i \)

assume \( I: I \ (g \ i) \)

\textbf{note} \( m = \min[\text{of } i] \)

\textbf{note} \( ms = \min[\text{of } Suc \ i] \)

from \( P[\text{OF } m \ ms] \)

\( Pw[\text{OF } m \ ms] \)

\( R[\text{OF } I \ m \ ms] \)

\( Rw[\text{OF } I \ m \ ms] \)

\( \text{steps}[\text{of } i] \)

have \((f \ (g \ i), f \ (g \ (Suc \ i))) \in \ richTextBox\ (ty \ i) \land I \ (g \ (Suc \ i)) \)

by (cases \( ty \ i, \ auto) \)

\} \textbf{note} \( stepsI = \ this \)

\{ 

fix \( i \)

have \( I \ (g \ (\tilde{\iota} \ i)) \)

\textbf{proof} (induct \( i \))

\texttt{case } \( 0 \)

\texttt{show } ?\texttt{case using stepsP[OF } k] \texttt{ by simp}

next

\texttt{case } (Suc \( i))

\texttt{from stepsI[OF } Suc \texttt{] show } ?\texttt{case by simp}

\texttt{qed}

\} \textbf{note} \( I = \ this \)

have \( \sim \) \( \text{SN-rel-ext} \ {?P \ ?Pw \ ?R \ ?Rw \ M'} \)

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

\textbf{proof} (rule exI[of - ?f], rule exI[of - ?ty], intro allI conjI)

fix \( i \)

\texttt{show } \{?f \ i, ?f \ (Suc \ i)\} \in \ richTextBox\ (\forall ty \ i)

\texttt{using stepsI[OF } I[\text{of } i]\texttt{] by auto}

\texttt{next}

\texttt{show INFM } i \ . \ ty \ i \in \{top-s, top-ns\}

\texttt{unfolding } Infm-shift[of \ \lambda i. i \in \{top-s,top-ns\} \ ty \ k]

\texttt{by (rule inf1)}

\texttt{next}

\texttt{show INFM } i \ . \ ty \ i \in \{top-s, normal-s\}

\texttt{unfolding } Infm-shift[of \ \lambda i. i \in \{top-s,normal-s\} \ ty \ k]

\texttt{by (rule inf2)}

\texttt{next}

fix \( i \)

have \( A: A \subseteq Ms \) \textbf{unfolding} \( A \) \textbf{by auto}

92
from \texttt{rtrancl-mono[\textbar this]} have \(A^* \subseteq M^*\) by \texttt{auto}

have \(PM: \ ?P \subseteq M^* \ O \ M^* \) using \(As\) by \texttt{auto}

have \(PwM: \ ?Pw \subseteq M^* \ O \ M^* \) using \(As\) by \texttt{auto}

have \(RM: \ ?R \subseteq M^* \ O \ M^* \) using \(As\) by \texttt{auto}

have \(RwM: \ ?Rw \subseteq M^* \) using \(As\) by \texttt{auto}

from \(PM, PwM, RM\) have \(\ ?P \cup \ ?Pw \cup \ ?R \subseteq M^* \ O \ M^* \) by \texttt{auto}

also have \(... \subseteq M^+ \) by \texttt{regexp}

also have \(... = M^\) by \texttt{regexp}

proof

have \(M^+ \subseteq M^* \ O \ M^* \) by \texttt{regexp}

also have \(... \subseteq M\) unfolding \(M^\) by \texttt{auto}

finally show \(M^+ \subseteq M\) by \texttt{auto}

qed \(\texttt{regexp}\)

finally have \(PPR: \ ?PPR \subseteq M^\) show \(M^\) by \texttt{simp blast}

proof \((\texttt{induct } i)\)

case \(0\)

from \texttt{stepsP[of } k\texttt{]} have \(f(\ g\ k)\), \(f(\ g\ (\texttt{Suc}\ k))\) \(\in \ ?PPR\) by \((\texttt{cases } ty\ k, \texttt{auto})\)

with \(PPR\) show \(?case\ unfolding\ M^\) by \texttt{simp blast}

next
case \(\texttt{Suc}\ i\)

show \(?case\)

proof \((\texttt{cases } ?ty\ i = \texttt{normal-ns})\)

case \(\texttt{False}\)

hence \(?ty\ i \in \{\texttt{top-s, top-ns, normal-s}\}\)

by \((\texttt{cases } ?ty\ i, \texttt{auto})\)

with \texttt{stepsI[of } i\texttt{]} have \(\ ?f\ i, \ ?f\ (\texttt{Suc}\ i)\) \(\in \ ?PPR\) by \texttt{auto}

from \texttt{subsetD[of } PPR\ this\texttt{]} have \(\ ?f\ i, \ ?f\ (\texttt{Suc}\ i)\) \(\in M^\) .

thus \(?thesis\ unfolding\ M^\) by \texttt{auto}

next
case \(\texttt{True}\)

with \texttt{stepsI[of } i\texttt{]} have \(\ ?f\ i, \ ?f\ (\texttt{Suc}\ i)\) \(\in \ ?Rw\) by \texttt{auto}

with \(RwM\) have \(\texttt{mem}: \ ?f\ i, \ ?f\ (\texttt{Suc}\ i)\) \(\in M^*\) by \texttt{auto}

thus \(?thesis\)

proof \((\texttt{cases})\)

case \(\texttt{base}\)

with \(\texttt{Suc}\ show\ ?thesis\ by \texttt{simp}\)

next
case \(\texttt{step}\)

thus \(?thesis\ unfolding\ M^\) by \texttt{simp}\n
qed

qed

qed

with \(SN\)

show \(\texttt{False}\ unfolding\ A\ M^\) by \texttt{simp}\n
93
lemma SN-rel-ext-map-min: fixes P Pw R Rw P' Pw' R' Rw' :: 'a rel and M M' :: 'a ⇒ bool

defines Ms: Ms ≡ {(s,t). M' t}
defines A: A ≡ P' ∩ Ms ∪ Pw' ∩ Ms ∪ R' ∪ Rw'
assumes SN: SN-rel-ext P' Pw' R' Rw' M'
and M: ∩ t. M t ⇒ M' (f t)
and M': ∩ s t. M' s ⇒ (s,t) ∈ R' ∪ Rw' ⇒ M' t
and P: ∩ s t. M s ⇒ M t ⇒ M' (f s) ⇒ M' (f t) ⇒ (s,t) ∈ P ⇒ (f s, f t) ∈ (A~* O (P' ∩ Ms) O A~*) ∧ I t
and Pw: ∩ s t. M s ⇒ M t ⇒ M' (f s) ⇒ M' (f t) ⇒ (s,t) ∈ Pw ⇒ (f s, f t) ∈ (A~* O (P' ∩ Pw' ∪ Ms) O A~*) ∧ I t
and R: ∩ s t. I s ⇒ M s ⇒ M t ⇒ M' (f s) ⇒ M' (f t) ⇒ (s,t) ∈ R ⇒ (f s, f t) ∈ (A~* O (P' ∩ Pw' ∪ R') O A~*) ∧ I t
and Rw: ∩ s t. I s ⇒ M s ⇒ M t ⇒ M' (f s) ⇒ M' (f t) ⇒ (s,t) ∈ Rw ⇒ (f s, f t) ∈ A~* ∧ I t
shows SN-rel-ext P Pw R Rw M

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

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

{ fix s t
assume s: M' s and steps: (s,t) ∈ A~*
from steps have (s,t) ∈ ?A~* ∧ M' t
proof (induct)
case base from s show ?case by simp
next
case (step t u)
  note one = Aone[OF step(3)[THEN conjunct2] step(2)]
  from step(3) one
  have steps: (s,u) ∈ ?A~* O ?A by blast
  have (s,u) ∈ ?A~*
    by (rule subsetD[OF steps, regexp])
  with one show ?case by simp
qed
} note Amany = this
let ?P = (A~* O (P' ∩ Ms) O A~*)
let ?Pw = (A~* O (P' ∩ Ms ∪ Pw' ∩ Ms) O A~*)
let ?R = (A~* O (P' ∩ Ms ∪ R') O A~*)
let ?Rw = A~*
let \( Pw' = (\exists \bar{A}^* \cup \bar{P} \cup \bar{Pw}) \cap \bar{Ms} \cup \bar{A^*} \)
let \( R' = (\exists \bar{A}^* \cup \bar{P} \cup \bar{R}) \cap \bar{Ms} \cup \bar{A^*} \)
let \( Rw' = \bar{A^*} \)

show \( ?\text{thesis} \)

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

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

next

fix \( s \) \( t \)
assume \( s: M s \) \( t: M t \) \( \text{and step:} (s, t) \in Pw \)
from \( Pw[\text{OF} s t M[\text{OF} s] M[\text{OF} t] \text{ step}] \)
have \( (f, s, f t) \in \bar{Pw} \) \( \text{and} \) \( I: I t \) \text{ by auto} \)
then obtain \( u v \) \( \text{where} \) \( su: (f, s, u) \in \bar{A^*} \) \( \text{and} \) \( vw: (u, v) \in \bar{P} \cap \bar{Ms} \cup \bar{Pw} \cap \bar{Ms} \)
and \( vt: (v, f t) \in \bar{A^*} \) \text{ by auto} \)
from \( \text{Amany[} \text{OF} M[\text{OF} s] \text{ su] have su:} (f, s, u) \in \bar{A^*} \) \( \text{and} \) \( u: M' u \) \text{ by auto} \)
from \( \text{uv have uv:} \bar{u} \in \bar{P} \cap Ms \) \( \text{and} \) \( v: \bar{M'} \) \( \text{v unfolding} \) \( \text{Ms by auto} \)
from \( \text{Amany[} \text{OF} v \text{ vt] have vt:} (v, f t) \in \bar{A^*} \) \text{ by auto} \)
from \( \text{su uv vt I} \) show \( (f, s, f t) \in \bar{Pw} \cap I t \) \text{ by auto} \)

next

fix \( s \) \( t \)
assume \( I: I s \) \( s: M s \) \( t: M t \) \( \text{and step:} (s, t) \in R \)
from \( R[\text{OF} I s t M[\text{OF} s] M[\text{OF} t] \text{ step}] \)
have \( (f, s, f t) \in \bar{R} \) \( \text{and} \) \( I: I t \) \text{ by auto} \)
then obtain \( u v \) \( \text{where} \) \( su: (f, s, u) \in \bar{A^*} \) \( \text{and} \) \( vw: (u, v) \in \bar{P} \cap \bar{Ms} \cup \bar{R} \)
and \( vt: (v, f t) \in \bar{A^*} \) \text{ by auto} \)
from \( \text{Amany[} \text{OF} M[\text{OF} s] \text{ su] have su:} (f, s, u) \in \bar{A^*} \) \( \text{and} \) \( u: M' u \) \text{ by auto} \)
from \( \text{uv have uv:} \bar{u} \in \bar{P} \cap Ms \) \( \text{and} \) \( v: \bar{M'} \) \( \text{v unfolding} \) \( \text{Ms by auto} \)
from \( \text{uv have uv:} \bar{u} \in \bar{P} \cap Ms \) \( \text{and} \) \( v: \bar{M'} \) \( \text{v unfolding} \) \( \text{Ms by auto} \)

by \text{ auto} \)
from \( \text{Amany[} \text{OF} v \text{ vt] have vt:} (v, f t) \in \bar{A^*} \) \text{ by auto} \)
from \( \text{su uv vt I} \) show \( (f, s, f t) \in \bar{R} \cap I t \) \text{ by auto} \)

next

fix \( s \) \( t \)
assume \( I: I s \) \( s: M s \) \( t: M t \) \( \text{and step:} (s, t) \in Rw \)
from \( Rw[\text{OF} I s t M[\text{OF} s] M[\text{OF} t] \text{ step}] \)
have \( \text{steps:} (f, s, f t) \in \bar{Rw} \) \( \text{and} \) \( I: I t \) \text{ by auto} \)
from Amany[OF M[OF s] steps] I
show \((f \, s, f \, t) \in ?RW \land I \, t\) by auto
qed
qed

lemma SN-relto-imp-SN-rel: \(SN (\text{relto} \, R \, S) \implies SN-\text{rel} \, R \, S\)
proof
  assume SN: \(SN (\text{relto} \, R \, S)\)
  show \(?\text{thesis}\)
  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 \(r \equiv \bigwedge j. r \, j \equiv (f \, j, f \, (\text{Suc} \, j)) \in R\) by auto
    show \(\neg (\text{INFM} \, j. (f \, j, f \, (\text{Suc} \, j)) \in R)\)
    proof (rule ccontr)
      assume \(\neg ?\text{thesis}\)
      hence \(ih: \text{infinitely-many} \, r\) unfolding infinitely-many-def r INFM-nat-le by blast
      obtain \(r\) where \(r \equiv \text{infinitely-many-def} \, r\) by simp
      with \(\text{infinitely-many-def} \, r\) have \(\text{inter}: \bigwedge j. r \, j \land j < r \, \text{Suc} \, j \implies (f \, j, f \, (\text{Suc} \, j)) \in S^*\) by auto
      obtain \(g\) where \(g \equiv f \, (\text{r-index} \, i)\)
      { fix \(i\)
        let \(\text{ri} = \text{r-index} \, i\)
        let \(\text{rsi} = \text{r-index} \, (\text{Suc} \, i)\)
        from r-index have isi: \(\text{ri} \leq \text{rsi}\) by auto
        obtain \(\text{ri} \, \text{rsi}\) where \(\text{ri} = \text{ri} \text{and} \, \text{rsi} = \text{rsi} \text{by auto}\)
        with r-index[of i] steps have inter: \(\bigwedge j. r \, i \land j < \text{rsi} \implies (f \, j, f \, (\text{Suc} \, j)) \in S^*\) by auto
        from \(\text{ri} \, \text{isi}\) have \(\text{risi}: \text{ri} < \text{rsi}\) by simp
        { fix \(n\)
          assume \(\text{Suc} \, n \leq \text{rsi} \, r\, \text{ri}\)
          hence \((f \, (\text{Suc} \, \text{ri}), f \, (\text{Suc} \, (n + \text{ri}))) \in S^*\) by simp
          proof (induct n, simp)
            case (Suc \(n\))
            hence steps: \((f \, (\text{Suc} \, \text{ri}), f \, (\text{Suc} \, (n+\text{ri}))) \in S^*\) by simp
            have \((f \, (\text{Suc} \, n+\text{ri})), f \, (\text{Suc} \, (\text{Suc} \, n + \text{ri}))) \in S\) by auto
            using \text{inter}[of Suc  \, n + \text{ri}] Suc(2) by auto
            with steps show \(?\text{case}\) by simp
          qed
        from \(\text{this}[\text{of} \, \text{rsi} - \text{ri} - 1]\) \(\text{risi}\) have \((f \, (\text{Suc} \, \text{ri}), f \, \text{rsi}) \in S^*\) by simp
        with \(\text{ri} \, \text{rsi}\) have ssteps: \((f \, (\text{Suc} \, ?\text{ri})), f \, \text{rsi}) \in S^*\) by simp
      qed
    qed
  qed
qed
with r-index[of i] have \((f \circ r_i, f \circ rsi) \in R \times S^*\) unfolding \(r\) by auto  

hence \((g \circ i, g \circ (Suc i)) \in S^* \times R \times O \times S^*\) using \(\text{rtrancl-refl}\) unfolding \(g\) by auto

\}

hence \(\neg SN (S^* \times O \times R \times S^*)\) unfolding \(\text{SN-defs}\) by blast  
with \(\text{SN}\) show False by simp

qed

qed

lemma \(\text{rtrancl-list-conv}\):
\[(s,t) \in R^* = \]
\[(\exists \text{list}. \ 	ext{last}(s \# \text{list}) = t \land (\forall i. \ i < \text{length list} \rightarrow ((s \# \text{list}) ! i, (s \# \text{list}) ! Suc i) \in R)) \ 	ext{(is ?l = ?r)}\]

proof  
assume ?r  
then obtain \(\text{list}\) where \(\text{last}(s \# \text{list}) = t \land (\forall i. \ i < \text{length list} \rightarrow ((s \# \text{list}) ! i, (s \# \text{list}) ! Suc i) \in R)\) \ ..

thus \(?l\)  
proof (induct list arbitrary: \(s\), simp)

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

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

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

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

with \(\text{rec}\) show ?case by auto

qed

next

assume ?l

from \(\text{rtrancl-imp-seq}[OF this]\)

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

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

show ?r

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

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

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

proof (intro \(\text{allI}\ \text{impl}\)

fix \(i\)

assume \(i: i < \text{length } ?list\)

thus \((s \# ?list) ! i, (s \# ?list) ! Suc i) \in R\)

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

case \((Suc \ j)\)

with \(i\) \text{steps show } ?\text{thesis} \ by \ simp

qed

qed

qed

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

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

have \( gsteps: \forall i. (g i, g (Suc i)) \in R \cup S \)
proof
  fix \( n \)
  obtain \( i \) \( j \) where \( n: ?g n = (i, j) \) by (cases \( ?g n \), auto)
  from len \( \text{OF } n \) False have \( j = \text{length } (?l i) - 1 \) by auto
  with \( gn \) have \( g n = \text{last } (?l i) \) using last-conv-nth[of \( ?l i \)] by auto
  from \( gn \) \( gsn \) show ?thesis using steps[of \( i \)] steps[of Suc \( i \)] by auto
qed

have \( \text{infR: } \forall n. \exists j \geq n. (g j, g (Suc j)) \in R \)
proof
  fix \( n \)
  obtain \( i \) \( j \) where \( n: ?g n = (i,j) \) by (cases \( ?g n \), auto)
  from \( \text{len } \text{OF } n \) have \( j: j \leq \text{length } (?l i) - 1 \) by simp
  let \( ?k = \text{length } (?l i) - 1 - j \)
  obtain \( k \) where \( k = j + ?k \) by auto
  from \( j \) \( k \) have \( k2: k = \text{length } (?l i) - 1 \) and \( k3: j + ?k < \text{length } (?l i) \) by
{ fix \ n \ i \ j \ k \ l \\
  assume \ n: \ choice \ l \ n = (i, j) \ and \ j + k < \ length (l \ i) \\
  hence \ choice \ l \ (n + k) = (i, j + k) \\
  by \ (induct \ k \ arbitrary: \ j, \ simp, \ auto) \\
} \\
from \ this[\ OF \ n, \ of ?k, \ OF \ k3] \\
have \ gnk: ?g (n + ?k) = (i, k) \ by \ (simp \ only: \ k) \\
hence \ gnk2: ?g (n + ?k) = last (?(l \ i) \ k2 \ by \ auto \\
from \ k2 \ gnk \ have \ main: (g (n+?k), g (Suc (n+?k))) \in \ R \\
  by \ (simp \ only: \ gnk2 \ gnsk2) \\
show \ \exists \ j \geq \ n. \ (g \ j, \ g (Suc \ j)) \in \ R \\
  by \ (rule \ exI[of \ - \ n + \ ?k], \ auto \ simp: \ main[simplified]) \\
qed \\
from \ SN[\ simplified \ SN-rel-on-conv \ SN-rel-defs] \ gsteps \ infR \ show \ False \\
  unfolding \ INFM-nat-le \ by \ fast \\
qed \\
hide-const \ choice
lemma $\text{SN-relto-Id}$:

$$\text{SN} \ (\text{relto} \ R \ (S \cup \text{Id})) = \text{SN} \ (\text{relto} \ R \ S)$$

by (simp only: relto-Id)

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

lemma $\text{trans-subset-SN}$:

assumes $\text{trans} \ R$ and $R \subseteq (r \cup s)$ and $\text{SN} \ r$ and $\text{SN} \ s$

shows $\text{SN} \ R$

proof

fix $f :: \text{nat} \Rightarrow 'a$

assume $f \ 0 \in \text{UNIV}$

and chain: chain $R \ f$

have $\ast: \\forall i \ j. \ i < j \implies (f \ i, f \ j) \in r \cup s$

using assms and chain-imp-trancl [OF chain] by auto

let $?M = \{i. \ \forall j>i. \ (f \ i, f \ j) \not\in r\}$

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 $\text{INFM} \ j. \ j \in ?M$ by (simp add: Inf-many-def)

then interpret infinitely-many $\lambda i. \ i \in ?M$ by (unfold-locales) assumption

define $g$ where $[\text{simp}]: \ g = \text{index}$

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 $i \ \text{Suc} \ i$] by simp

have $g \ i \in ?M$ using index-p by simp

then have $(f \ (g \ i), f \ (g \ (\text{Suc} \ i))) \not\in r$ using less by simp

moreover have $(f \ (g \ i), f \ (g \ (\text{Suc} \ i))) \in r \cup s$ using $\ast$ [OF less] by simp

ultimately have $(f \ (g \ i), f \ (g \ (\text{Suc} \ i))) \in s$ by blast

qed

with $\text{SN} \ s$ show False by (auto simp: $\text{SN-defs}$)

qed

qed

lemma $\text{SN-Un-conv}$:

assumes $\text{trans} \ (r \cup s)$

shows $\text{SN} \ (r \cup s) \iff \text{SN} \ r \wedge \text{SN} \ s$

(is $\text{SN} \ ?r \iff ?rhs$)

proof

assume $\text{SN} \ (r \cup s)$ thus $\text{SN} \ r \wedge \text{SN} \ s$

using $\text{SN-subset[of} \ ?r]$ by blast

next

101
assume SN r \land SN s
with trans-subset-SN[OF assms subset-refl] show SN ?r by simp
qed

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

lemma SN-relto-split:
assumes SN (relto r (s \cup q2) \cup relto q1 (s \cup q2)) (is SN ?a)
and SN (relto s q2) (is SN ?b)
shows SN (relto r (q1 \cup q2) \cup relto s q1 \cup q2)) (is SN ?c)
proof
  have '?c^+ \subseteq ?a^+ \cup ?b^+' by regexp
  from trans-subset-SN[OF - this], unfolded SN-trancl-SN-conv, OF - assms
  show ?thesis by simp
qed

lemma relto-trancl-subset:
assumes a \subseteq c and b \subseteq c shows relto a b \subseteq c^+
proof
  have relto a b \subseteq (a \cup b)^+ by regexp
  also have \ldots \subseteq c^+
    by (rule trancl-mono-set, insert assms, auto)
  finally show ?thesis.
qed

An explicit version of relto which mentions all intermediate terms

inductive relto-fun :: 'a rel \Rightarrow 'a rel \Rightarrow nat \Rightarrow ('a \Rightarrow bool) \Rightarrow nat
\Rightarrow 'a \times 'a \Rightarrow bool where
relto-fun: as 0 = a \Rightarrow as m = b \Rightarrow
  \lambda 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 \mid i < m \land sel i \}
  \Rightarrow (n = 0 \iff m = 0) \Rightarrow relto-fun A B n as sel m (a,b)

lemma relto-funD: assumes relto-fun A B n as sel m (a,b)
shows as 0 = a as m = b
\lambda i. i < m \Rightarrow sel i \rightarrow (as i, as (Suc i)) \in A
\lambda i. i < m \Rightarrow \neg sel i \rightarrow (as i, as (Suc i)) \in B
n = card \{ i \mid 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 - λ a], rule exI, rule relto-fun, auto)

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

lemma relto-fun-trans: assumes \(ab: \text{relto-fun} ~ A ~ B ~ n1 ~ \text{as1} ~ s\ell1 ~ m1 \ (a,b)\)
and \(bc: \text{relto-fun} ~ A ~ B ~ n2 ~ \text{as2} ~ s\ell2 ~ m2 \ (b,c)\)
shows \(\exists ~ \text{as sel} \ \text{relto-fun} ~ A ~ B ~ (n1 + n2) ~ \text{as sel} \ (m1 + m2) \ (a,c)\)
proof –
  from \(\text{relto-funD}(\text{OF} \ ab)\)
  have \(1: \text{as1} ~ 0 = a \) \(\text{as1} \ m1 = b\)
  \(\land i. ~ i < m1 \Longrightarrow (s\ell1 ~ i \longrightarrow (\text{as1} ~ i, \text{as1} (\text{Suc} ~ i)) \in A) \land (\neg s\ell1 ~ i \longrightarrow (\text{as1} ~ i, \text{as1} (\text{Suc} ~ i)) \in B)\)
  qed
as1 (Suc i) \in B)
\begin{align*}
n1 &= 0 \iff m1 = 0 \quad \text{and card1: } n1 = \text{card } \{ i. \; i < m1 \land \text{sel1 } i \} \quad \text{by blast+} \\
\text{from relto-funD[OF bc]} \\
\text{have } 2: \; \alpha \in \beta \quad \text{and card2: } n2 = \text{card } \{ i. \; i < m2 \land \text{sel2 } i \} \quad \text{by blast+} \\
\text{let } \alpha = \lambda i. \; \text{as1 } i \quad \text{and } \beta = \lambda i. \; \text{as2 } (\text{Suc } i) \\
\text{let } \gamma = m1 + m2 \\
\text{let } \eta = n1 + n2 \\
\text{show } \eta \text{thesis} \\
\text{proof } \text{(rule exI[of } \alpha \text{], rule exI[of } \beta \text{], rule relto-fun)} \\
\text{have id: } \{ i. \; i < m1 \land \text{sel1 } i \} = \{ i. \; i < m1 \land \text{sel1 } i \} \cup ((+ \text{ m1}) \cdot \{ i. \; i < m2 \land \text{sel2 } i \}) \\
\text{by force} \\
\text{have } \text{card } (\alpha \cup \beta) = \text{card } \alpha + \text{card } (\beta) \\
\text{by (rule card-Un-disjoint, auto)} \\
\text{also have } \text{card } (\beta) = \text{card } \beta \\
\text{by (rule card-image, auto simp inj-on-def)} \\
\text{finally show } \eta = \text{card } \{ i. \; i < m1 \land \text{sel1 } i \} \quad \text{unfolding card1 card2 id by simp} \\
\text{next} \\
\text{fix } i \\
\text{assume } i: \; i < m1 \\
\text{show } (i= \text{as1 } i \quad \text{as2 } (\text{Suc } i) \in A) \land (\neg \; \text{sel1 } i \quad \text{as2 } (\text{Suc } i) \in B) \\
\text{proof } \text{(cases } i < m1) \\
\text{case True} \\
\text{from 1 2 have } \text{as2 } 0 = \text{as1 } m1 \quad \text{by simp} \\
\text{from True 1(3)[of i] 1(2) show } \text{thesis} \quad \text{by (cases Suc } i = m1, \text{ auto)} \\
\text{next} \\
\text{case False} \\
\text{define } j \text{ where } j = i - m1 \\
\text{have i: } i = m1 + j \quad \text{and } j < m2 \quad \text{using i False unfolding j-def by auto} \\
\text{thus } \text{thesis} \quad \text{using False 2(3)[of j] by auto} \\
\text{qed} \\
\text{qed} \quad \text{(insert 1 2, auto)} \\
\text{qed} \\
\text{lemma } \text{reltos-into-relto-fun: assumes } (a,b) \in (\text{relto } A B)^{\sim n} \\
\text{shows } \exists \; \text{as sel m, relto-fun } A B \; n \quad \text{as sel m } (a,b) \\
\text{using assms} \\
\text{proof } \text{(induct } n \text{ arbitrary; b)} \\
\text{case } (\emptyset b) \\
\text{hence } b: b = a \quad \text{by auto} \\
\text{show } \text{case unfolding b using relto-fun-refl[of } A B a \text{] by blast} \\
\text{next} \\
\text{qed}
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: \( \text{relto-fun} \ A \ B \ n \) as sel m \( (a, b) \) by auto
from relto-into-relto-fun[OF bc] obtain as sel m where \( \text{relto-fun} \ A \ B \ (\text{Suc} \ 0) \) as sel m \( (b, c) \) by blast
from relto-fun-trans[OF IH this] show \( \text{?case} \) by auto
qed

lemma relto-fun-into-reltos; assumes \( \text{relto-fun} \ A \ B \ n \) as sel m \( (a, b) \)
shows \( (a, b) \in (\text{relto} \ A \ B) \sim n \)
proof –
  note \( \ast = \text{relto-funD}[OF assms] \)
  \{ \begin{align*}
  &\text{fix } m' \\
  &\text{let } \text{?c} = \lambda m'. \text{card } \{ \text{i. } i < m' \land \text{sel } i \} \\
  &\text{assume } m' \leq m \\
  &\text{hence } (\text{?c } m' > 0 \implies (\text{as } 0, \text{ as } m') \in (\text{relto} \ A \ B) \sim \text{?c } m') \land (\text{?c } m' = 0 \implies (\text{as } 0, \text{ as } m') \in B^*) \\
  &\text{proof (induct } m') \\
  &\text{case (Suc } m') \\
  &\text{let } \text{?x} = \text{as } 0 \\
  &\text{let } \text{?y} = \text{as } m' \\
  &\text{let } \text{?z} = \text{(Suc } m') \\
  &\text{let } \text{?C} = \text{?c } (\text{Suc } m') \\
  &\text{have } C: \ (\text{?C} = \text{?c } m' + (\text{if } (\text{sel } m') \text{ then } 1 \text{ else } 0) \\
  &\text{proof -} \\
  &\text{have id: } \{ \text{i. } i < \text{Suc } m' \land \text{sel } i \} = \{ \text{i. } i < m' \land \text{sel } i \} \cup (\text{if } \text{sel } m' \text{ then } \{ m' \} \text{ else } \{} \} \\
  &\text{by (cases sel } m', \text{ auto, case-tac } x = m', \text{ auto) \\
  &\text{show } \text{thesis unfolding id by auto} \\
  qed \\
  &\text{from Suc(2) have } m': m' \leq m \text{ and } \text{lt}: m' < m \text{ by auto} \\
  &\text{from Suc(1)[OF m'] have } \text{IH}: \ (\text{?c } m' > 0 \implies (\text{?x, } \text{?y}) \in (\text{relto} \ A \ B) \sim \text{?c } m') \\
  &\ (\text{?c } m' = 0 \implies (\text{?x, } \text{?y}) \in B^*) \text{ by auto} \\
  &\text{from } \ast(3-4)[OF \lt]\text{ have } yz: \text{ sel } m' \implies (\text{?y, } \text{?z}) \in A \sim \text{ sel } m' \implies (\text{?y, } \text{?z}) \\
  \in B \text{ by auto} \\
  &\text{show } \text{?case} \\
  &\text{proof (cases } \text{?c } m' = 0) \\
  &\text{case True note } c = \text{this} \\
  &\text{from } \text{IH}(2)[OF this] \text{ have } xy: (\text{?x, } \text{?y}) \in B^* \text{ by auto} \\
  &\text{show } \text{thesis} \\
  &\text{proof (cases sel } m') \\
  &\text{case False} \\
  &\text{from } xy yz(2)[OF False] \text{ have } xz: (\text{?x, } \text{?z}) \in B^* \text{ by auto} \\
  &\text{from False c have } C: \ (\text{?C} = 0 \text{ unfolding } C \text{ by simp} \\
  &\text{from } xz \text{ show } \text{thesis unfolding } C \text{ by auto} \\
\end{align*} \}
\)
\)
\)
\)

105
next
case True
from \( xy \text{ yz(1)} \text{[OF True]} \) have \( xx : (\?x, \?z) \in \text{ relto } A \text{ B by auto} \)
from True c have C: \(?C = 1\) unfolding C by simp
from \( xx \) show \(?\text{thesis unfolding } C \) by auto
qed
next
case False
hence c: \(?c m' > 0\) (\(?c m' = 0\)) = False by arith+
from \( IH(1) \text{[OF c(1)]} \) have \( xy: (\?x, \?y) \in (\text{ relto } A \text{ B}) \sim \ ?c m' \).
show ?thesis
proof (cases \( \text{ sel m' } \))
  case False
  from c obtain \( k \) where \( \text{ ck: } \?c m' = \text{ Suc } \) \( k \) by (cases \( \?c m' \), auto)
  from relpow-Suc-E [OF \( xy \) unfolded this] obtain
    \( u \) where \( xu: (\?x, \?u) \in (\text{ relto } A \text{ B}) \sim k \) \text{ and } \( uy: (\?u, \?y) \in \text{ relto } A \text{ B by auto} \)
  from uy yz [OF False] have \( uz: (\?u, \?z) \in \text{ relto } A \text{ B by force} \)
  with \( xu \) have \( \text{zx: } (\?x, \?z) \in (\text{ relto } A \text{ B}) \sim \text{ Suc } \?c m' \).
  qed
  qed
  qed simp
}\)
from \( \text{ this[of } m \} \ast \text{ show } \?\text{thesis by auto} \)
qed

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

lemma \( \text{ relto-fun-intermediate: assumes } A \subseteq C \text{ and } B \subseteq C \)
  and \( \text{ rf: relto-fun A B n as sel m (a,b) } \)
  shows \( i \leq m \Rightarrow (a, \text{as } i) \in C \sim \ast \)
proof (induct \( i \))
  case 0
  from \( \text{ relto-funD[OF rf]} \) show ?case by simp
next
  case (Suc \( i \))
  hence IH: \((a, \text{as } i) \in C \sim \ast \text{ and im: } i < m \text{ by auto} \)
  from relto-funD(3-4)[OF rf im] assms have \((a, \text{as } (\text{Suc } i)) \in C \text{ by auto} \)
with IH show ?case by auto
qed

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

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

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

lemma relative-ending:
  assumes chain: \(\text{chain} (R \cup S) \ t\)
  and \(0\): \(t \ 0 \in X\)
  and \(\text{SN}\): \(\text{SN-on (relto R S)} \ X\)
  shows \(\exists j. \ \forall i \geq j. (t \ i, t (\text{Suc} \ i)) \in S - R\)
proof \(\text{(rule ccontr)}\)
  assume \(\neg \text{thesis}\)
  with \(\text{chain}\) have \(\forall i. \ \exists j. j \geq i \land (t \ j, t (\text{Suc} \ j)) \in R\) by blast
  from choice \(\text{[OF this]}\) obtain \(f\) where \(R\text{-steps}: \forall i. \ i \leq f i \land (t \ (f \ i), t \ (\text{Suc} \ (f \ i))) \in R\ .\)
  let \(\forall i. t \ (((\text{Suc} \circ f) \ \sim \ i) \ 0)\)
  have \(\forall i. (t \ i, t (\text{Suc} \ (f \ i))) \in (\text{relto R S})^+\)
proof
  fix \(i\)
  from \(R\text{-steps}\) have \(\text{leq}: i \leq f i\) and \(\text{step}: (t (f \ i), t (\text{Suc} (f \ i))) \in R\) by auto
  from chain-imp-rtrancl \([[\text{OF chain leq}}\)] have \(t \ i, t (\text{Suc} (f \ i)) \in (R \cup S)^+\).
  with \(\text{step}\) have \((t \ i, t (\text{Suc} (f \ i))) \in (R \cup S)^+ \ O \ R\) by auto
  then show \((t \ i, t (\text{Suc} (f \ i))) \in (\text{relto R S})^+\) by regexp
qed
then have \(\text{chain} ((\text{relto R S})^+)\) \(\forall t\) by simp
with \(0\) have \(\neg \text{SN-on ((relto R S)^+)} \ X\) by (unfold SN-on-def, auto intro: exI[of - \(\forall t\)])
with \(\text{SN-on-trancl}[OF SN]\) show False by auto
qed
from Geser’s thesis \([p.32, \text{Corollary-1}], \text{generalized for SN-on}\).

lemma \(\text{SN-on-relto-Un}\):
  assumes closure: \(\text{relto} \ (R \cup R') \ S \ X \subseteq X\)
  shows \(\text{SN-on (relto} \ (R \cup R') \ S) \ X \leftrightarrow \text{SN-on(\text{relto} R (R' \cup S))} \ X \land \text{SN-on (relto} R' \ S) \ X\)
(is \(?c\) \(\leftrightarrow \ ?a \land \ ?b\))
proof\(\text{(safe)}\)
  assume \(\text{SN}: \ ?a\) and \(\text{SN}: \ ?b\)
from SN have SN: SN-on (relto (relto R S) (relto R' S)) X by (rule SN-on-subset1)

proof
  show ?c
  proof
    fix f
    assume f0: f 0 ∈ X and chain: chain (relto (R ∪ R') S) f
    then have chain (relto R S ∪ relto R' S) f by auto
    from relative-ending[OF this f0 SN] have ∃ j. ∀ i ≥ j. (f i, f (Suc i)) ∈ relto R' S − relto R S by auto
    then obtain j where ∀ i ≥ j. (f i, f (Suc i)) ∈ relto R' S by auto
    then have chain (relto R' S) (shift f j) by auto
    moreover have f j ∈ X
    proof (induct j)
      case 0 from f0 show ?case by simp
      next
      case (Suc j)
      let ?s = (f j, f (Suc j))
      from chain have ?s ∈ relto (R ∪ R') S by auto
      with Image-closed-trancl[OF closure Suc] show f (Suc j) ∈ X by blast
      qed
      then have shift f j 0 ∈ X by auto
      ultimately have = SN-on (relto R' S) X by (intro not-SN-onI)
      with SN' show False by auto
    qed
  qed
next
  assume SN: ?c
  then show ?b by (rule SN-on-subset1, auto)
moreover
  from SN have SN-on ((relto (R ∪ R') S)⁺) X by (unfold SN-on-trancl-SN-on-conv)
  then show ?a by (rule SN-on-subset1) regexp
qed

lemma SN-on-Un: (R ∪ R')⁺ X ⊆ X =⇒ SN-on (R ∪ R') X ⇐ SN-on (relto R R') X ∧ SN-on R' X
  using SN-on-relto-Un[of {}] by simp

end

4 Strongly Normalizing Orders

theory SN-Orders
imports Abstract-Rewriting
begin

  We define several classes of orders which are used to build ordered semirings. Note that we do not use Isabelle’s preorders since the condition x > y = x ≥ y ∧ y ≤ x is sometimes not applicable. E.g., for δ-orders over the rationals we have 0.2 ≥ 0.1 ∧ 0.1 ≤ 0.2, but 0.2 >_δ 0.1 does not hold if δ is larger than 0.1.
class non-strict-order = ord +
  assumes ge-refl: \( x \geq (x :: 'a) \)
  and ge-trans[trans]: \( [x \geq y; (y :: 'a) \geq z] \implies x \geq z \)
  and max-comm: \( \text{max } x y = \text{max } y x \)
  and max-ge-x[intro]: \( \text{max } x y \geq x \)
  and max-id: \( x \geq y \implies \text{max } x y = x \)
  and max-mono: \( x \geq y \implies \text{max } z x \geq \text{max } z y \)

begin
  lemma max-ge-y[intro]: \( \text{max } x y \geq y \)
  unfolding max-comm[of x y] ..

lemma max-mono2: \( x \geq y \implies \text{max } x z \geq \text{max } y z \)
  unfolding max-comm[of - z] by (rule max-mono)
end

class ordered-ab-semigroup = non-strict-order + ab-semigroup-add + monoid-add +
  assumes plus-left-mono: \( x \geq y \implies x + z \geq y + z \)

lemma plus-right-mono: \( y \geq (z :: 'a :: ordered-ab-semigroup) \implies x + y \geq x + z \)
  by (simp add: add.commute[of x], rule plus-left-mono, auto)

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

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

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

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

begin
  lemma gt-trans[trans]: \( [x \succ y; y \succ z] \implies x \succ z \)
    by (rule compat[OF gt-imp-ge])
end

locale one-mono-ordered-semiring-1 = order-pair gt
  for gt :: 'a :: ordered-semiring-1 \Rightarrow 'a \Rightarrow bool (infix \succ 50) +
  assumes plus-gt-left-mono: \( x \succ y \implies x + z \succ y + z \)
  and default-gt-zero: default \( \succ 0 \)
begin
lemma plus-gt-right-mono: x > 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; x ≥ 0 ⟹ x * y ≥ x * z
locale both-mono-ordered-semiring-1 = order-pair gt
  for gt :: 'a :: ordered-semiring-1 ⇒ bool
  by (rule zero.leastI)
  and arc-pos-zero: ¬ arc-pos 0
  and arc-pos-plus: arc-pos x ⟹ arc-pos (x + y)
  and arc-pos-mult: arc-pos x; arc-pos y ⟹ arc-pos (x * y)
  and not-all-ge: ∀ c d. arc-pos d ⟹ ∃ e. e ≥ 0 ∧ arc-pos e ∧ ¬ (c ≥ d * e)
begin
lemma max0-id: max 0 (x :: 'a) = x
  unfolding max_comp[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 ⇒ bool
  and default :: 'a
  and mono :: 'a ⇒ bool
  assumes weak-gt-mono: ∀ x y. (x, y) ∈ set xys ⟹ weak-gt x y ⟹ ∃ gt. SN-strict-mono-ordered-semiring-1 default gt mono ∧ (∀ x y. (x, y) ∈ set xys ⟹ gt x y)
locale weak-SN-both-mono-ordered-semiring-1 = 
  fixes weak-gt :: 'a :: ordered-semiring-1 ⇒ 'a ⇒ bool 
  and default :: 'a 
  and arc-pos :: 'a ⇒ bool 
  assumes weak-gt-both-mono: ∀ x y. (x,y) ∈ set xys ⇒ weak-gt x y ⇒ ∃ gt. 
SN-both-mono-ordered-semiring-1 default gt arc-pos ∧ (∀ x y. (x,y) ∈ set xys ⇒ 
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
  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 
  also have ... ≥ b * a ^ n unfolding power-Suc by simp 
  using 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 
proof (induct n)
  case 0
  finally show ?case by simp
next 
  case (Suc n)
show ?case using times-left-mono[of Suc a] by simp
qed
end

lemma of-nat-ge-zero[intro,simp]: of-nat \( n \geq (0 :: 'a :: ordered-semiring-1) \)
proof (induct n)
case 0
  show ?case by (simp add: ge-refl)
next
case (Suc n)
  from plus-right-mono[of Suc, of 1] have of-nat (Suc n) \( \geq (1 :: 'a) \) by simp
  also have \( (1 :: 'a) \geq 0 \) using one-ge-zero .
  finally show ?case .
qed

lemma mult-ge-zero[intro]: \( (a :: 'a :: ordered-semiring-1) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a \ast b \geq 0 \)
using times-left-mono[of b 0 a] by auto

lemma pow-mono-one: assumes \( a :: 'a :: ordered-semiring-1 \)
shows \( a \geq (1 :: 'a) \)
proof (induct n)
case (Suc n)
  show ?case unfolding power-Suc
    using ge-trans[of times-right-mono[of ge-trans[of a one-ge-zero] Suc], of 1]
    by (auto simp: field-simps)
qed (auto simp: ge-refl)

lemma pow-mono-exp: assumes \( a :: 'a :: ordered-semiring-1 \)
shows \( n \geq m \Longrightarrow a \ast n \geq a \ast m \)
proof (induct m arbitrary: n)
case 0
  show ?case using pow-mono-one[of a] by auto
next
case (Suc m nn)
  then obtain n where nn: \( nn = Suc\ n \) by (cases nn, auto)
  note Suc = Suc[unfolded nn]
  hence rec: \( a \ast n \geq a \ast m \) by auto
  show ?case unfolding nn power-Suc
    by (rule times-right-mono[of ge-trans[of a one-ge-zero] rec])
qed

lemma mult-ge-one[intro]: assumes \( a :: 'a :: ordered-semiring-1 \) \( \geq 1 \)
  and \( b : b \geq 1 \)
  shows \( a \ast b \geq b \)
proof
  from ge-trans[of b one-ge-zero] have b0: \( b \geq 0 \).
  from times-left-mono[of b0 a] have \( a \ast b \geq b \) by simp
from ge-trans[ OF this b] show ?thesis.

qed

lemma sum-list-ge-mono: fixes as :: ('a :: ordered-semiring-0) list
assumes length as = length bs
and \( \forall i. i < \text{length } bs \Rightarrow \text{as }! i \geq \text{bs }! i \)
shows sum-list as \( \geq \) sum-list bs
using assms
proof (induct as arbitrary: bs)
case (Nil bs)
from Nil(1) show ?case by (simp add: ge-refl)
next
case (Cons a as bbs)
from Cons(2) obtain b bs where bbs = b \# bs and len: length as = length bs by (cases bbs, auto)

note ge = Cons(3)[unfolded bbs]
{
  fix i
  assume i < length bs
  hence Suc i < length (b \# bs) by simp
  from ge[OF this] have as ! i \geq \text{bs }! i by simp
}

from Cons(1)[OF len this] have IH: sum-list as \( \geq \) sum-list bs .
from ge[of \theta] have ab: a \geq b by simp
from ge-trans[OF plus-left-mono[OF ab] plus-right-mono[OF IH]]
show ?case unfolding bbs by simp
qed

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

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

lemma foldr-max: a \in set as \( \Rightarrow \) foldr max as b \( \geq \) (a :: 'a :: ordered-ab-semigroup)
proof (induct as arbitrary: b)
case Nil thus ?case by simp
next
case (Cons c as)
show \( ?case \)
proof (cases \( a = c \))
  case True
  show \( ?thesis \) unfolding True by auto
next
  case False
  with Cons have foldr max as \( b \geq a \) by auto
  from ge-trans[OF - this] show \( ?thesis \) by auto
qed

lemma of-nat-mono[intro]: assumes \( n \geq m \) shows \((\text{of-nat } n :: 'a :: \text{ordered-semiring-1}) \geq \text{of-nat } m\)
proof –
  let \( ?n = \text{of-nat} :: \text{nat} \Rightarrow 'a \)
  from assms show \( ?thesis \)
  proof (induct \( m \) arbitrary: \( n \))
    case 0
    show \( ?case \) by auto
  next
    case (Suc m \( nn \))
    then obtain \( n \) where \( nn = \text{Suc } n \) by (cases \( nn \), auto)
    note Suc = Suc[unfolded \( nn \)]
    hence rec: \( ?n \geq ?n m \) by simp
    show \( ?case \) unfolding \( nn \) of-nat-Suc
    by (rule plus-right-mono[OF rec])
  qed
qed

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

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

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

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

lemma non-inf-image: assumes \( ni: \text{non-inf } r \) and image: \( \land a b. \ (a,b) \in s \Longrightarrow (f \ a, f \ b) \in r \)
  shows non-inf \( s \)
proof
  fix \( a \) \( g \)
assume \( s : \bigwedge i. (g i, g (\text{Suc } i)) \in s \)
define \( h \) where \( h = f \circ g \)

from image[OF \( s \)] have \( h : \bigwedge i. (h i, h (\text{Suc } i)) \in r \) unfolding \( h \)-def comp-def .
from non-infE[OF \( ni \), of \( h \)] have \( \bigwedge a. \exists i. (h i, a) \notin r \) using \( h \) by blast
thus \( \exists i. (g i, a) \notin s \) using image unfolding \( h \)-def comp-def by blast

end

5 Carriers of Strongly Normalizing Orders

theory \texttt{SN-Order-Carrier}

imports
\texttt{SN-Orders}  
\texttt{HOL.Rat}

begin

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

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

5.1 The standard semiring over the naturals

instantiation \texttt{nat :: large-ordered-semiring-1}

begin

instance by (intro-classes, auto)

end

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

interpretation \texttt{nat-SN: SN-strict-mono-ordered-semiring-1 (>) :: nat \Rightarrow bool nat-mono}

by (unfold-locales, insert \texttt{SN-nat-gt}, auto simp: \texttt{nat-mono-def})

interpretation \texttt{nat-poly: poly-order-carrier 1 (>) :: nat \Rightarrow bool True discrete}

proof (unfold-locales)

fix \( x, y :: \texttt{nat} \)

assume \( \texttt{ge} : x \geq y \)

obtain \( k \) where \( k : x - y = k \) by auto

qed
show \( \exists \, k. \, x = ((+ \ 1 \ ^\ k) \ y) \)

proof (rule exI[of - \ k])
  from \( ge \ k \) have \( x = k + y \) by simp
  also have \( \ldots = ((+ \ 1 \ ^\ k) \ y) \)
    by (induct \( k \), auto)
  finally show \( x = ((+ \ 1 \ ^\ k) \ y) \).
qed

qed (auto simp: field-simps power-strict-mono)

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

lemma non-inf-delta-gt: assumes \( \delta \): \( \delta > 0 \)
  shows \( \{ (a, b): delta-gt \delta a b \} (is\, \non-inf \ ?r) \)
proof
  let \( ?gt = \) delta-gt \( \delta \)
  fix \( a :: 'a \) and \( f \)
  assume \( \bigwedge \ i. \, \, (f \ i, f (Suc \ i)) \in \ ?r \)
  hence \( gt: \bigwedge \ i. \, ?gt (f \ i) (f (Suc \ i)) \) by simp
  { fix \( i \)
    have \( f \ i ≤ f 0 - \delta * of-nat \ i \)
      proof (induct \( i \))
        case (Suc \( i \))
        thus \( \exists \)case using \( gt[of \ i, \ unfolded \ delta-gt-def] \)
          by (auto simp: field-simps)
      qed simp
    } note \( \hat{f} = this \)
  { fix \( r :: 'a \)
    have \( of-nat (nat (ceiling (f 0)) ≥ r \)
      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 \( i = nat (ceiling ((f 0 - a) / \delta)) \)
  from \( \hat{f}[of \ i] \) have \( f \ i - f 0 ≤ -\delta * of-nat (nat (ceiling ((f 0 - a) / \delta))) \)
    unfolding i-def by simp
  also have \( \ldots ≤ -\delta * ((f 0 - a) / \delta) \) using ceil-elim[of \( f 0 - a \) / \( \delta \)] \( \) delta
    by (metis le-imp-neg-le minus-mult-commute mult-le-cancel-left-pos)
  also have \( \ldots = - f 0 + a \) using delta by auto
  also have \( \ldots < - f 0 + a + \delta \) using delta by auto
  finally have \( \neg \, ?gt (f \ i) \) a unfolding delta-gt-def by arith
  thus \( \exists \ i. \, (f \ i, a) \notin \ ?r \) by blast
qed

lemma delta-gt-SN: assumes \( dpos: \delta > 0 \) shows \( SN \ \{(x, y). \, 0 \leq y \land delta-gt \delta x y\} \)

117
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: z ≤ int (nat z) by auto
  with x have x ≤ of-int (int (nat z))
    by (metis (full-types) le-cases of-int-0-le-iff of-int-of-nat-eq of-nat-0-le-iff of-nat-nat order-trans)
  also have . . . = of-nat (nat z) unfolding of-int-of-nat-eq ..
  finally show ∃ y. x ≤ of-nat y by blast
qed (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg max-def)

lemma delta-interpretation: assumes dpos: δ > 0 and default: δ ≤ def
  shows SN-strict-mono-ordered-semiring-1 def (delta-gt δ) delta-mono
proof –
  from dpos default have defz: 0 ≤ def by auto
  show ?thesis
  proof (unfold-locales)
    show SN {(x,y). y ≥ 0 ∧ delta-gt δ x y} by (rule delta-gt-SN[OF dpos])
  next
    fix x y z :: 'a
    assume delta-mono x and yz: delta-gt δ y z
    hence x: 1 ≤ x unfolding delta-mono-def by simp
    have ∃ d > 0. delta-gt δ = (λ x y. d ≤ x − y)
      by (rule exI[of _], 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 yz0: 0 ≤ y − z by auto
        have 0 ≤ ?p
          by (rule mult-nonneg-nonneg[OF x1 yz0!])
        have x * y − x * z = x * (y − z) using right-diff-distrib[of x y z] by auto
        also have . . . = ((x − 1) + 1) * (y − z) by auto
        also have . . . = ?p + 1 * (y − z) by (rule ring-distrib(2))
        also have . . . ≥ (0 + d) using yzd (0 ≤ ?p) by auto
      end
finally
  show \( d \leq x * y - x * z \) by auto
qed
qed (insert dpos, auto simp: delta-gt-def default defz)
qed

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

next
  fix \( x \ y :: 'a \)
  assume False
  thus \( \exists \ k \cdot x = ((+ \ 1) \ ^ \ k \) \( y \) \) by simp
qed (rule times-gt-mono, auto)
lemma delta-minimal-delta: assumes \( \forall x, y. (x, y) \in \text{set } xys \rightarrow x > y \)
shows \( \exists \delta > 0. \forall x, y. (x, y) \in \text{set } xys \rightarrow \delta \text{-gt } \delta \text{ x y} \)
using assms
proof (induct xys)
case Nil
show \( ?\text{case} \) by (rule exI[of - 1], auto)
next
case (Cons xy xys)
show \( ?\text{case} \)
proof (cases xy)
case (Pair x y)
with Cons have \( x > y \) by auto
then obtain \( d_1 \) where \( d_1 = x - y \) and \( d_1 > 0 \) and \( d_1 \leq x - y \) by auto
hence \( \ldots \) unfolding \( \delta \text{-gt-def} \) by auto
from Cons obtain \( d_2 \) where \( d_2 > 0 \) and \( \ldots \) unfolding \( \delta \text{-gt-def} \) by auto
with \( \text{dpos} \) have \( d > 0 \) and \( \ldots \) unfolding \( \delta \text{-gt-def} \) by force
with \( \text{dpos} \) show \( ?\text{thesis} \) by auto
qed
qed

interpretation weak-delta-SN: weak-SN-strict-mono-ordered-semiring-1 (\( > \)) \( 1 \) delta-mono
proof
fix \( xysp :: ('a \times 'a) \text{ list} \)
assume orient: \( \forall x, y. (x, y) \in \text{set } xysp \rightarrow x > y \)
obtain \( \text{xys where } xys: xys = (1, 0) \neq \text{xsy by auto} \)
with orient have \( \ldots \) by auto
with \( \delta \text{-minimal-delta} \) have \( \exists \delta > 0. \forall x, y. (x, y) \in \text{set } xys \rightarrow \delta \text{-gt } \delta \text{ x y} \) by auto
then obtain \( \delta \) where \( \text{dpos: } \delta > 0 \) and orient: \( \ldots \) unfolding \( \delta \text{-gt-def} \) by auto
from orient have orient1: \( \forall x, y. (x, y) \in \text{set } xysp \rightarrow \delta \text{-gt } \delta \text{ x y} \) and orient2:
\( \delta \text{-gt-def} \) by auto
from orient2 have oned: \( \delta \leq 1 \) unfolding \( \delta \text{-gt-def} \) by auto
show \( \exists g. \text{SN-strict-mono-ordered-semiring-1 } 1 \text{ gt delta-mono } \land \ldots \) unfolding \( \delta \text{-gt-def} \) by auto
\( \ldots \) by (intro exI conjI, rule delta-interpretation[OF \( \text{dpos} \) oned], rule orient1)
qed
5.3 The standard semiring over the integers

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

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

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

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

interpretation int-poly: poly-order-carrier 1 (>) :: int ⇒ int ⇒ bool True discrete
proof (unfold-locales)
  fix x y :: int
  assume ge: x ≥ y
  then obtain k where k: x − y = k and kp: 0 ≤ k by auto
  then obtain nk where nk: nk = nat k and k: x − y = int nk by auto
  show ∃ k. x = ((+) 1 ^^ 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
begin
fun less-eq-arctic :: arctic ⇒ arctic ⇒ bool where
less-eq-arctic MinInfty x = True
| less-eq-arctic (Num-arc -) MinInfty = False
| less-eq-arctic (Num-arc y) (Num-arc x) = (y ≤ x)

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

instance ..

end

instantiation arctic :: ordered-semiring-1
begin

fun plus-arctic :: arctic ⇒ arctic ⇒ arctic where
plus-arctic MinInfty y = y
| plus-arctic x MinInfty = x
| plus-arctic (Num-arc x) (Num-arc y) = (Num-arc (max x y))

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

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

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

instance

proof
  fix x y z :: arctic
  show x + y = y + x
    by (cases x, cases y, auto, cases y, auto)
  show (x + y) + z = x + (y + z)
    by (cases x, auto, cases y, auto, cases z, auto)
  show (x * y) * z = x * (y * z)
    by (cases x, auto, cases y, auto, cases z, auto)
  show x * 0 = 0
    by (cases x, auto simp: zero-arctic-def)
  show x * (y + z) = x * y + x * z
    by (cases x, auto, cases y, auto, cases z, auto)
  show (x + y) * z = x * z + y * z
    by (cases x, auto, cases y, auto, cases z, auto)
  show 1 * x = x
    by (cases x, simp-all add: one-arctic-def)
  show x * 1 = x
    by (cases x, simp-all add: one-arctic-def)
show \( 0 + x = x \)
by (simp add: zero-arctic-def)

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

show \( (0 :: \text{arctic}) \neq 1 \)
by (simp add: zero-arctic-def one-arctic-def)

show \( x + 0 = x \) by (cases x, auto simp: zero-arctic-def)

show \( x \geq x \)
by (cases x, auto)

show \( (1 :: \text{arctic}) \geq 0 \)
by (simp add: zero-arctic-def one-arctic-def)

show \( \max x y = \max y x \) unfolding max-def
by (cases x, (cases y, auto)+)

show \( \max x y \geq x \) unfolding max-def
by (cases x, (cases y, auto)+)

assume ge: \( x \geq y \)
from ge show \( x + z \geq y + z \)
by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)

from ge show \( x \times z \geq y \times z \)
by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)

from ge show \( \max x y = x \) unfolding max-def
by (cases x, (cases y, auto)+)

from ge show \( \max z x \geq \max z y \) unfolding max-def
by (cases z, cases x, auto, cases x, (cases y, auto)+)

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

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

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

qed

end

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

fun pos-arctic :: \( \text{arctic} \Rightarrow \text{bool} \)
where pos-arctic MinInfty = False
| pos-arctic (Num-arc n) = (0 <= n)
interpretation arctic-SN: SN-both-mono-ordered-semiring-1 1 (>({>) pos-arctic
proof
fix x y z :: arctic
assume x ≥ y and y > z
thus x > z
  by (cases z, simp, cases y, simp, cases x, auto)
next
fix x y z :: arctic
assume x > y and y ≥ z
thus x > z
  by (cases z, simp, cases y, simp, cases x, auto)
next
fix x y z :: arctic
assume x > y
thus x ≥ y
  by (cases x, (cases y, auto)+)
next
fix x y z u :: arctic
assume x > y and z > u
thus x + z > y + u
  by (cases y, cases z, simp, cases u, auto, cases x, auto, cases u, auto, cases z,
  auto, cases z, auto, cases x, auto, cases z, auto, cases x, auto)
next
fix x y z :: arctic
assume x > y
thus x * z > y * z
  by (cases y, simp, cases z, simp, cases x, auto)
next
fix x :: arctic
assume 0 > x
thus x = 0
  by (cases x, auto simp: zero-arctic-def)
next
fix x :: arctic
show pos-arctic 1 unfolding one-arctic-def by simp
show x > 0 unfolding zero-arctic-def by simp
show (1 :: arctic) ≥ 0 unfolding zero-arctic-def by simp
show x ≥ 0 unfolding zero-arctic-def by simp
show ¬ pos-arctic 0 unfolding zero-arctic-def by simp
next
fix x y
assume pos-arctic x
thus pos-arctic (x + y) by (cases x, simp, cases y, auto)
next
fix x y
assume pos-arctic x and pos-arctic y
thus pos-arctic (x * y) by (cases x, simp, cases y, auto)
next
show $SN \{ (x, y). \text{pos-artic} y \wedge x > y \}$ (is $SN \ ?rel$)
proof ~ {
  fix $x$
  assume $\exists f . f \ 0 = x \wedge (\forall i . (f \ i, f \ (Suc \ i)) \in \ ?rel)$
  from this obtain $f$ where $f \ 0 = x$ and seq: $\forall i . (f \ i, f \ (Suc \ i)) \in \ ?rel$ by auto
  from seq have steps: $\forall i . f \ i > f \ (Suc \ i) \wedge \text{pos-artic} (f \ (Suc \ i))$ by auto
  let $?g = \lambda i . \text{get-artic-num} \ (f \ i)$
  have $\forall i . ?g \ (Suc \ i) \geq 0 \wedge ?g \ i > ?g \ (Suc \ i)$ by auto
  proof
    fix $i$
    from steps have $i$: $f \ i > f \ (Suc \ i) \wedge \text{pos-artic} (f \ (Suc \ i))$ by auto
    from $i$ obtain $n$ where $f i$: $f \ i = \text{Num-arc} \ n$ by cases $f \ (Suc \ i)$, simp, cases $f i$, auto
    with $i$ have $gz$: $0 \leq m$ by simp
    from $i$ $f i$ $fsi$ have $n$: $n > m$ by auto
    with $fi$ $fsi$ $gz$ show $\forall i . ?g \ (Suc \ i) \geq 0 \wedge ?g \ i > ?g \ (Suc \ i)$ by auto
  qed
  from this obtain $g$ where $\forall i . g \ (Suc \ i) \geq 0 \wedge (\forall i . (f \ i, f \ (Suc \ i)) \in \ {(x, y). y \geq 0 \wedge x > y})$ by auto
  with $int-SN.SN$ have $False$ unfolding $SN-defs$ by auto
} thus $?thesis$ unfolding $SN-defs$ by auto
  qed
next
  fix $y$ $z$ $x$ :: arctic
  assume $y > z$
  thus $x * y > x * z$
    by (cases $x$, simp, cases $z$, simp, cases $y$, auto)
next
  fix $c$ $d$
  assume $\text{pos-artic} \ d$
  then obtain $n$ where $d = \text{Num-arc} \ n$ and $n$: $0 \leq n$
    by (cases $d$, auto)
  show $\exists e . e \geq 0 \wedge \text{pos-artic} \ e \wedge \neg e \geq d * e$
    proof (cases $e$)
      case $\text{MinInfty}$
      show $?thesis$
        by (rule exI[of - $\text{Num-arc} \ 0$],
            unfold $d$ MinInfty $\text{zero-artic-def}$, simp)
    next
      case ($\text{Num-arc} \ m$)
      show $?thesis$
        by (rule exI[of - $\text{Num-arc} \ (abs \ m \ + \ 1)$], insert $n$,
5.5 The arctic semiring over an arbitrary archimedean field

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

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

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

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

instance ..
end

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

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

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

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

instance
proof
  fix x y z :: 'a arctic-delta
  show x + y = y + x
  by (cases x, cases y, auto, cases y, auto)
show \((x + y) + z = x + (y + z)\)
   by (cases x, auto, cases y, auto, cases z, auto)
show \((x * y) * z = x * (y * z)\)
   by (cases x, auto, cases y, auto, cases z, auto)
show \(x * 0 = 0\)
   by (cases x, auto simp: zero-arctic-delta-def)
show \(x * (y + z) = x * y + x * z\)
   by (cases x, auto, cases y, auto, cases z, auto)
show \((x + y) * z = x * z + y * z\)
   by (cases x, auto, cases y, cases z, auto, cases z, auto)
show \(1 * x = x\)
   by (cases x, simp-all add: one-arctic-delta-def)
show \(x * 1 = x\)
   by (simp add: zero-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 \(\text{ge}\) \(x \geq y\)
from \(\text{ge}\) show \(x + z \geq y + z\)
   by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
from \(\text{ge}\) show \(x * z \geq y * z\)
   by (cases x, cases y, cases z, auto, cases y, cases z, 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 \(x y z :: 'a arctic-delta\)
assume \(x \geq y\) and \(y \geq z\)
thus \(x \geq z\)
   by (cases x, cases y, auto, cases y, cases z, auto, cases z, auto)
next
fix \(x y z :: 'a arctic-delta\)
assume \(y \geq z\)
thus \(x * y \geq x * z\)
   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, simp add: delta-gt-def, cases z, cases x, simp, cases u, simp, simp add: delta-gt-def)
  qed
next
  fix x y z
  assume gt-arctic-delta \( \delta \) x y
  thus gt-arctic-delta \( \delta \) (x * z) (y * z)
  by (cases y, simp, cases z, simp, cases x, simp, simp add: plus-gt-left-mono)

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

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

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

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

hence \(\exists\ f.\ f\ 0 = g\ 0 \land (\forall\ i.\ (f\ i,\ f\ (\text{Suc } i)) \in \{(x,y).\ y \geq 0 \land \text{delta-gt } \delta\ x\ y\})\) by auto

with SN have False unfolding SN-defs by auto

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

qed

next

fix \(c\ d::\ 'a\ \text{arctic-delta}\)
assume pos-arctic-delta \(d\)
then obtain \(n\) where \(d = \text{Num-arc-delta } n\) and \(n\: 0 \leq n\)
by (cases \(d\), auto)
show \(\exists\ e.\ e \geq 0 \land \text{pos-arctic-delta } e \land \neg\ e \geq d \ast e\)
proof (cases \(c\))
case MinInfty-delta
show \(?\)thesis
by (rule exI[of - \text{Num-arc-delta } 0], unfold \(d\) MinInfty-delta zero-arctic-delta-def, simp)

next
case (\text{Num-arc-delta } m)
show \(?\)thesis
by (rule exI[of - \text{Num-arc-delta } (\text{abs } m + 1)], insert \(n\), unfold \(d\) Num-arc-delta zero-arctic-delta-def, simp)

qed

next

fix \(x\ y\ z\)
assume \(\text{gt}:\ \text{gt-arctic-delta } \delta\ y\ z\)
{
fix \(x\ y\ z\)
assume \(\text{gt}:\ \text{delta-gt } \delta\ y\ z\)
have \(\text{delta-gt } \delta\ (x + y)\ (x + z)\)
using plus-gt-left-mono[OF \text{gt}] by (auto simp: field-simps)
}

with \(\text{gt}\) show \(\text{gt-arctic-delta } \delta\ (x \ast y)\ (x \ast z)\)
by (cases \(x\), simp, cases \(z\), simp, cases \(y\), simp-all)

qed

fun weak-gt-arctic-delta :: ('a::floor-ceiling) arctic-delta ⇒ 'a arctic-delta ⇒ bool
where weak-gt-arctic-delta - MinInfty-delta = True
    | weak-gt-arctic-delta MinInfty-delta (\text{Num-arc-delta } -) = False
    | weak-gt-arctic-delta (\text{Num-arc-delta } x) (\text{Num-arc-delta } y) = (x \geq y)

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

proof
fix \(xys\)
assume orient: \(\forall\ x\ y.\ (x,y) \in \text{set } xys \rightarrow \text{weak-gt-arctic-delta } x\ y\)

130
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 s = 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 ax ay where (ax, ay) ∈ set xys and ax ≠ MinInfty-delta and ay ≠ MinInfty-delta
  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 azay: (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, simp)
    case (Num-arc-delta y) note ay = this
    show ?thesis
    proof (cases ax)
      case MinInfty-delta
      with ay orient ?thesis by auto
    next
      case (Num-arc-delta x) note ax = this
      from azay 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 ezl conjI, rule arctic-delta-interpretation[OF dpos], rule orient)
qed

end

References
