# 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

[^0]5 Carriers of Strongly Normalizing Orders ..... 116
5.1 The standard semiring over the naturals ..... 116
5.2 The standard semiring over the Archimedean fields using delta- orderings ..... 117
5.3 The standard semiring over the integers ..... 121
5.4 The arctic semiring over the integers ..... 121
5.5 The arctic semiring over an arbitrary archimedean field ..... 126
A description of this formalization will be available in [2].

## 1 Infinite Sequences

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

Infinite sequences are represented by functions of type nat $\Rightarrow$ ' $a$.
type-synonym 'a seq $=n a t \Rightarrow{ }^{\prime} a$

### 1.1 Operations on Infinite Sequences

An infinite sequence is linked by a binary predicate $P$ if every two consecutive elements satisfy it. Such a sequence is called a $P$-chain.
abbreviation (input) chainp :: ('a $\Rightarrow^{\prime} a \Rightarrow$ bool $) \Rightarrow^{\prime} a$ seq $\Rightarrow$ bool where chainp $P S \equiv \forall$. $P\left(\begin{array}{l}\text { Si) })(S \text { (Suc i) }) ~\end{array}\right.$ Special version for relations.
abbreviation (input) chain :: 'a rel $\Rightarrow$ 'a seq $\Rightarrow$ bool where chain $r S \equiv$ chainp $(\lambda x y .(x, y) \in r) S$
Extending a chain at the front.
lemma cons-chainp:
assumes $P x(S 0)$ and chainp $P S$
shows chainp $P$ (case-nat $x$ S) (is chainp $P$ ? $S$ )
proof
fix $i$ show $P$ (?S i) (?S (Suc i)) using assms by (cases i) simp-all
qed
Special version for relations.
lemma cons-chain:
assumes $(x, S 0) \in r$ and chain $r S$ shows chain $r$ (case-nat $x S$ )
using cons-chainp[of $\lambda x y .(x, y) \in r$, OF assms].
A chain admits arbitrary transitive steps.
lemma chainp-imp-relpowp:
assumes chainp $P S$ shows $\left(P^{\wedge} j\right)(S i)(S(i+j))$

```
proof (induct i + j arbitrary: j)
    case (Suc n) thus ?case using assms by (cases j) auto
qed simp
lemma chain-imp-relpow:
    assumes chain r S shows (S i,S (i+j)) \inr~~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 PS and i<j shows P}\mp@subsup{P}{}{`}++(Si)(Sj
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
lemma chain-imp-trancl:
    assumes chain r S and i<j shows (Si,Sj) \in r^+
proof -
    from less-imp-Suc-add[OF assms(2)] obtain n where j=i+Suc n by auto
    with chain-imp-relpow[OF assms(1), of i Suc n]
        show ?thesis unfolding trancl-power by force
qed
```

A chain admits arbitrary reflexive and transitive steps.
lemma chainp-imp-rtranclp:
assumes chainp $P S$ and $i \leq j$ shows $P^{\wedge} * *(S i)(S j)$
proof -
from $\operatorname{assms}(2)$ obtain $n$ where $j=i+n$ by (induct $j-i$ arbitrary: $j$ ) force +
with chainp-imp-relpowp [of PS,OF assms(1), of $n i]$ show ?thesis
by (simp add: relpow-imp-rtrancl $[$ of $(S i, S(i+n))$, to-pred $])$
qed
lemma chain-imp-rtrancl:
assumes chain $r S$ and $i \leq j$ shows $(S i, S j) \in r^{*}$
proof -
from assms(2) obtain $n$ where $j=i+n$ by (induct $j-i$ arbitrary: $j$ ) force +
with chain-imp-relpow[OF assms(1), of $i n]$ show?thesis by (simp add: relpow-imp-rtrancl)
qed

If for every $i$ there is a later index $f i$ such that the corresponding elements satisfy the predicate $P$, then there is a $P$-chain.
lemma stepfun-imp-chainp':
assumes $\forall i \geq n:: n a t . f i \geq i \wedge P(S i)(S(f i))$
shows chainp $P\left(\lambda i . S\left(\left(f \sim_{i}\right) n\right)\right)$ (is chainp $P$ ? T $)$

```
proof
    fix }
    from assms have (f~ i) n\geqn by (induct i) auto
    with assms[THEN spec[of - (f^^ i) n]]
        show P(?T i)(?T (Suc i)) by simp
qed
lemma stepfun-imp-chainp:
    assumes \foralli\geqn::nat. fi> i^P(Si) (S (fi))
    shows chainp P (\lambdai.S ((f^~ i) n)) (is chainp P ?T)
    using stepfun-imp-chainp'[of nf PS] and assms by force
lemma subchain:
    assumes }\foralli::nat>n. \existsj>i.P(fi)(fj
    shows \exists\varphi.(\forallij.i<j\longrightarrow\varphii<\varphij)\wedge(\foralli.P(f(\varphii))(f(\varphi(Suci))))
proof -
    from assms have \foralli\in{i.i>n}. \existsj>i.P(fi)(fj) by simp
    from bchoice [OF this] obtain g
        where *: }\foralli>n.gi>
        and **:}\foralli>n.P(fi)(f(gi)) by aut
    define }\varphi\mathrm{ where [simp]: }\varphi=(g\mp@subsup{)}{}{~}i)(Suc n) for 
    from * have ***: \bigwedgei.\varphi i>n by (induct-tac i) auto
    then have \i. \varphi i<\varphi (Suc i) using * by (induct-tac i) auto
    then have }\ij.i<j\Longrightarrow\varphii<\varphij by (rule lift-Suc-mono-less
    moreover have \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 }\foralli\geqn::nat. \existsj\geqi. P(Si)(Sj) shows \existsT. chainp P 
proof -
    from assms have }\foralli\in{i.i\geqn}.\existsj\geqi.P(Si)(Sj) by aut
    from bchoice [OF this]
        obtain f}\mathrm{ where }\foralli\geqn.fi\geqi\wedgeP(Si)(S(fi)) by aut
    from stepfun-imp-chainp'[of nfPS,OF this] show ?thesis by fast
qed
lemma steps-imp-chainp:
    assumes }\foralli\geqn::nat. \existsj>i.P (S i) (S j) shows \exists T. chainp P T
    using steps-imp-chainp}\mp@subsup{}{}{\prime}[of n P S] and assms by forc
```


### 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 }=>\mathrm{ bool
```

```
    assumes infinite: INFM j. p j
begin
lemma inf: \existsj\geqi.pj using infinite[unfolded INFM-nat-le] by auto
fun index :: nat seq where
    index 0 = (LEAST n.p n)
| index (Suc n)=(LEAST k.pk^k> index n)
lemma index-p: p (index n)
proof (induct n)
    case 0
    from inf obtain j where pj by auto
    with LeastI[of p j] show ?case by auto
next
    case (Suc n)
    from inf obtain k where k\geqSuc (index n) \wedge pk by auto
    with LeastI[of \lambdak.pk\wedgek> index n k] show ?case by auto
qed
lemma index-ordered: index n < index (Suc n)
proof -
    from inf obtain k where k\geqSuc (index n) ^ pk by auto
    with LeastI[of \lambdak.pk\wedgek> 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 }\negp
proof -
    from not-less-Least[OF i2[simplified]] i1 show ?thesis by auto
qed
lemma index-ordered-le:
    assumes i\leqj 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\mathrm{ index l
    shows \existsij.k = index i+j^index i+j<index (Suc i)
proof -
    from assms have k= index l + (k-index l) by auto
    then obtain u where k: k= index l+u by auto
    show ?thesis unfolding }
    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 ij
            where lu: index l+u= index i+j and lt: index i+j< index (Suc i) by
auto
    hence index l + u< index (Suc i) by auto
    show ?case
    proof (cases index l + (Suc u)= index (Suc i))
        case False
        show ?thesis
            by (rule exI[of - i], rule exI[of - Suc j], insert lu lt False, auto)
    next
        case True
        show ?thesis
            by (rule exI[of-Suc i], rule exI[of-0], insert True index-ordered[of Suc i],
auto)
    qed
    qed
qed
lemma index-ordered-less:
    assumes }i<j\mathrm{ shows index i< index j
proof -
    from assms have Suc i\leqj by auto
    from index-ordered-le[OF this]
    have index (Suc i)\leqindex j .
    with index-ordered[of i] show ?thesis by auto
qed
lemma index-not-p-start: assumes i:i<index 0 shows }\negp
proof -
    from i[simplified index.simps] have i< Least p.
    from not-less-Least[OF this] show ?thesis .
qed
end
```


### 1.3 Assembling Infinite Words from Finite Words

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

```
fun inf-concat-simple \(::(\) nat \(\Rightarrow\) nat \() \Rightarrow\) nat \(\Rightarrow\) (nat \(\times\) nat \()\) where
    inf-concat-simple f \(0=(0,0)\)
\(\mid\) 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 \(j l: j+l<f i\)
    shows inf-concat-simple \(f(k+l)=(i, j+l)\)
using \(j l\)
proof (induct \(l\) )
    case 0
    thus ?case using ck by simp
next
    case (Suc l)
    hence \(c\) : inf-concat-simple \(f(k+l)=(i, j+l)\) by auto
    show ?case
        by (simp add: c, insert Suc(2), auto)
qed
lemma inf-concat-simple-surj-zero: \(\exists k\). inf-concat-simple \(f k=(i, 0)\)
proof (induct \(i\) )
    case 0
    show ?case
        by (rule exI[of-0], simp)
next
    case (Suc i)
    then obtain \(k\) where \(c k\) : inf-concat-simple \(f k=(i, 0)\) by auto
    show ?case
    proof (cases \(f i\) )
        case 0
        show ?thesis
            by (rule exI [of - Suc k], simp add: ck 0)
    next
        case (Suc n)
        hence \(0+n<f i\) by auto
        from inf-concat-simple-add[OF ck, OF this] Suc
        show ?thesis
            by (intro exI \([\) of \(-k+\) Suc \(n]\), auto)
    qed
qed
lemma inf-concat-simple-surj:
    assumes \(j<f i\)
```

```
    shows \existsk. inf-concat-simple f k=(i,j)
proof -
    from assms have j: 0+j<fi by auto
    from inf-concat-simple-surj-zero obtain k where inf-concat-simple f k=(i,0)
by auto
    from inf-concat-simple-add[OF this, OF j] show ?thesis by auto
qed
lemma inf-concat-simple-mono:
    assumes k}\leq\mp@subsup{k}{}{\prime}\mathrm{ shows fst (inf-concat-simple f k)}\leqfst(inf-concat-simple f k'
proof -
    from assms have }\mp@subsup{k}{}{\prime}=k+(\mp@subsup{k}{}{\prime}-k)\mathrm{ by auto
    then obtain l where }\mp@subsup{k}{}{\prime}:\mp@subsup{k}{}{\prime}=k+l\mathrm{ by auto
    show ?thesis unfolding k'
    proof (induct l)
        case (Suc l)
    obtain ij where ckl: inf-concat-simple f (k+l)=(i,j) by (cases inf-concat-simple
f(k+l),auto)
    with Suc have fst (inf-concat-simple fk)\leqi by auto
    also have .. \leqfst (inf-concat-simple f (k+Suc l))
        by (simp add: ckl)
    finally show ?case .
    qed simp
qed
```

fun inf-concat $::($ nat $\Rightarrow$ nat $) \Rightarrow$ nat $\Rightarrow$ nat $\times$ nat where
inf-concat $n 0=($ LEAST $j . n j>0,0)$
$\mid$ inf-concat $n($ Suc $k)=($ let $(i, j)=$ inf-concat $n k$ in (if Suc $j<n i$ then $(i, S u c$
j) else (LEAST $\left.\left.i^{\prime} . i^{\prime}>i \wedge n i^{\prime}>0,0\right)\right)$ )
lemma inf-concat-bounds:
assumes inf:INFM i. n $i>0$
and res: inf-concat $n k=(i, j)$
shows $j<n i$
proof (cases $k$ )
case 0
with res have $i: i=(L E A S T i . n i>0)$ and $j: j=0$ by auto
from inf[unfolded INFM-nat-le] obtain $i^{\prime}$ where $i^{\prime}: 0<n i^{\prime}$ by auto
have $0<n(L E A S T$ i. $n i>0)$
by (rule LeastI, rule $i^{\prime}$ )
with $i j$ show ?thesis by auto
next
case (Suc $k^{\prime}$ )
obtain $i^{\prime} j^{\prime}$ where res': inf-concat $n k^{\prime}=\left(i^{\prime}, j^{\prime}\right)$ by force
note res $=$ res[unfolded Suc inf-concat.simps res' Let-def split]
show ?thesis
proof (cases Suc $j^{\prime}<n i^{\prime}$ )

```
        case True
        with res show ?thesis by auto
    next
        case False
        with res have i:i=(LEAST f. i'< f^0<nf) and j:j=0 by auto
        from inf[unfolded INFM-nat] obtain f}\mathrm{ where f: i'< f^0<nf by auto
        have 0<n(LEASTf. i'<f^0<nf)
        using LeastI[of \lambdaf. i' }<f\^0<nf,OFf
        by auto
    with ij show ?thesis by auto
    qed
qed
lemma inf-concat-add:
    assumes res: inf-concat n k=(i,j)
        and j:j+m<ni
    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)=ni
    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<ni
    shows }\existsk.\mathrm{ inf-concat n k}=(i,0
proof -
    {
        fix l
        have }\forallj.j<l\wedge0<nj\longrightarrow(\existsk. inf-concat n k=(j,0)
        proof (induct l)
            case 0
            thus ?case by auto
        next
            case (Suc l)
```

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

```
                    hence ll
                    hence ll= l'}+(ll-\mp@subsup{l}{}{\prime})\mathrm{ by auto
                    then obtain k where ll: ll= l'}+k\mathrm{ by auto
                    from ass have l'}+k-?\mp@subsup{l}{}{\prime}<\mp@subsup{l}{}{\prime}\mathrm{ unfolding ll by auto
                    hence }k\mp@subsup{l}{}{\prime}:k<?\mp@subsup{l}{}{\prime}\mathrm{ by auto
                    have 0<n (ll - k) using ass unfolding ll by simp
                    from Least-le[of \lambdak. 0<n(ll-k), OF this] kl'
                    show False by auto
                    qed
                    qed
                    show ?thesis unfolding j
                    by (rule exI[of - k + Suc off], unfold id ll, simp)
            qed
                qed
        qed
    qed
}
with assms show ?thesis by auto
qed
lemma inf-concat-surj:
    assumes j: j<ni
    shows \existsk.inf-concat n k= (i,j)
proof -
    from j have 0<n i by auto
    from inf-concat-surj-zero[of n,OF this]
    obtain k where inf-concat n k=(i,0) by auto
    from inf-concat-add[OF this, of j] j
    show ?thesis by auto
qed
lemma inf-concat-mono:
    assumes inf:INFM i.n i>0
    and resk:inf-concat n k}=(i,j
    and reskp: inf-concat n k
    and lt:i< i'
    shows }k<\mp@subsup{k}{}{\prime
proof -
    note bounds= inf-concat-bounds[OF inf]
    {
    assume k}\mp@subsup{k}{}{\prime}\leq
    hence }k=\mp@subsup{k}{}{\prime}+(k-\mp@subsup{k}{}{\prime})\mathrm{ by auto
    then obtain l where k: k= k'}+l\mathrm{ by auto
    have i'
    proof (induct l)
            case 0
            with reskp show ?case by auto
    next
        case (Suc l)
```

```
        obtain }\mp@subsup{i}{}{\prime\prime}\mp@subsup{j}{}{\prime\prime}\mathrm{ where l: inf-concat n ( }\mp@subsup{k}{}{\prime}+l)=(\mp@subsup{i}{}{\prime\prime},\mp@subsup{j}{}{\prime\prime})\mathrm{ by force
        with Suc have one: i' }\mp@subsup{i}{}{\prime\prime
        from bounds[OF l] have }\mp@subsup{j}{}{\prime\prime}:\mp@subsup{j}{}{\prime\prime}<n\mp@subsup{i}{}{\prime\prime}\mathrm{ 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 = LEAST }\mp@subsup{i}{}{\prime}.\mp@subsup{i}{}{\prime\prime}<\mp@subsup{i}{}{\prime}\wedge0<n\mp@subsup{i}{}{\prime
            from inf[unfolded INFM-nat] obtain k where }\mp@subsup{i}{}{\prime\prime}<k\wedge0<nk by aut
            from LeastI[of \lambdak. i'\prime}<k\wedge0<nk,OF this
            have i" < ?i by auto
            with one show ?thesis by (simp add: l False)
        qed
    qed
    with resk klt have False by auto
}
thus ?thesis by arith
qed
lemma inf-concat-Suc:
    assumes inf:INFM i.n i>0
        and f:\bigwedgei.fi(ni)=f(Suc i) 0
        and resk: inf-concat n k=(i,j)
        and ressk: inf-concat n (Suc k) = (i', j')
    shows fi' j' = fi (Suc j)
proof -
    note bounds= inf-concat-bounds[OF inf]
    from bounds[OF resk] have j:j<ni.
    show ?thesis
    proof (cases Suc j<ni)
    case True
    with ressk resk
    show ?thesis by simp
    next
    case False
    let ?p=\lambda i'. i< i'^0<n i'
    let ? }\mp@subsup{i}{}{\prime}=LEAST i'. ?p i
    from False j have id:Suc (j+0)=ni by auto
    from inf-concat-step[OF resk, OF id] ressk
    have }\mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}=?\mp@subsup{i}{}{\prime}\mathrm{ and }\mp@subsup{j}{}{\prime}:\mp@subsup{j}{}{\prime}=0\mathrm{ 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 ? }\mp@subsup{i}{}{\prime}=S\mathrm{ Suc i+(? ? ' - Suc i) by simp
    then obtain d}\mathrm{ where }i\mp@subsup{i}{}{\prime}:? ?\mp@subsup{i}{}{\prime}=Suc i+d by aut
    from not-less-Least[of - ?p, unfolded ii`] have d': \ d'. d' < d C n (Suc i+
d')}=0\mathrm{ by auto
```

```
    have f(Suc i) 0 = f ? i' 0 unfolding ii' using d'
    proof (induct d)
        case 0
        show ?case by simp
    next
        case (Suc d)
        hence f(Suc i) 0=f(Suci+d)0 by auto
        also have ... =f(Suc (Suc i +d))0
        unfolding f[symmetric]
        using Suc(2)[of d] by simp
    finally show ?case by simp
    qed
    thus ?thesis unfolding }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime}jf\mathrm{ by simp
    qed
qed
end
```


## 2 Abstract Rewrite Systems

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

lemma trancl-mono-set:
$r \subseteq s \Longrightarrow r^{+} \subseteq s^{+}$
by (blast intro: trancl-mono)
lemma relpow-mono:
fixes $r$ :: 'a rel
assumes $r \subseteq r^{\prime}$ shows $r{ }^{\wedge} n \subseteq r^{\prime} \sim_{n}$
using assms by (induct $n$ ) auto
lemma refl-inv-image:
refl $R \Longrightarrow$ refl (inv-image $R f$ )
by (simp add: inv-image-def refl-on-def)

### 2.1 Definitions

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

```
definition join :: 'a rel # 'a rel ((-\downarrow) [1000] 999) where
    A\downarrow}=\mp@subsup{A}{}{*}O(\mp@subsup{A}{}{-1}\mp@subsup{)}{}{*
```

Two elements are meetable (and then have in the meetability relation)
w.r.t. $A$, iff they have a common ancestor.
definition meet :: 'a rel $\Rightarrow$ 'a rel ( $(-\uparrow)$ [1000] 999) where

$$
A^{\uparrow}=\left(A^{-1}\right)^{*} O A^{*}
$$

The symmetric closure of a relation allows steps in both directions.
abbreviation symcl :: 'a rel $\Rightarrow$ 'a rel $((-\leftrightarrow)$ [1000] 999) where $A^{\leftrightarrow} \equiv A \cup A^{-1}$

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

```
definition conversion :: 'a rel m 'a rel ((-↔*) [1000] 999) where
```

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

The set of normal forms of an ARS constitutes all the elements that do not have any successors.
definition $N F::$ 'a rel $\Rightarrow$ ' a set where $N F A=\{a . A "\{a\}=\{ \}\}$
definition normalizability :: 'a rel $\Rightarrow{ }^{\prime}$ 'a rel ( $\left(-{ }^{\prime}\right)$ [1000] 999) where $A^{!}=\left\{(a, b) .(a, b) \in A^{*} \wedge b \in N F A\right\}$

## notation (ASCII)

$$
\text { symcl }\left(\left(-{ }^{\wedge}<->\right)[1000]\right. \text { 999) and }
$$

$$
\text { conversion }((-<->*)[1000] 999) \text { and }
$$

$$
\text { normalizability }((-\uparrow)[1000] 999)
$$

lemma symcl-converse:

$$
\left(A^{\leftrightarrow}\right)^{-1}=A^{\leftrightarrow} \text { by auto }
$$

lemma symcl-Un: $(A \cup B)^{\leftrightarrow}=A^{\leftrightarrow} \cup B^{\leftrightarrow}$ by auto
lemma no-step:
assumes $A$ " $\{a\}=\{ \}$ shows $a \in N F A$
using assms by (auto simp: NF-def)

## lemma joinI:

$(a, c) \in A^{*} \Longrightarrow(b, c) \in A^{*} \Longrightarrow(a, b) \in A^{\downarrow}$ by (auto simp: join-def rtrancl-converse)
lemma joinI-left:
$(a, b) \in A^{*} \Longrightarrow(a, b) \in A^{\downarrow}$ by (auto simp: join-def)
lemma joinI-right: $(b, a) \in A^{*} \Longrightarrow(a, b) \in A^{\downarrow}$
by (rule joinI) auto
lemma joinE:
assumes $(a, b) \in A^{\downarrow}$
obtains $c$ where $(a, c) \in A^{*}$ and $(b, c) \in A^{*}$

```
    using assms by (auto simp: join-def rtrancl-converse)
lemma joinD:
    \((a, b) \in A^{\downarrow} \Longrightarrow \exists c .(a, c) \in A^{*} \wedge(b, c) \in A^{*}\)
    by (blast elim: joinE)
lemma meetI:
    \((a, b) \in A^{*} \Longrightarrow(a, c) \in A^{*} \Longrightarrow(b, c) \in A^{\uparrow}\)
    by (auto simp: meet-def rtrancl-converse)
lemma meetE:
    assumes \((b, c) \in A^{\uparrow}\)
    obtains \(a\) where \((a, b) \in A^{*}\) and \((a, c) \in A^{*}\)
    using assms by (auto simp: meet-def rtrancl-converse)
lemma meetD: \((b, c) \in A^{\uparrow} \Longrightarrow \exists a .(a, b) \in A^{*} \wedge(a, c) \in A^{*}\)
    by (blast elim: meetE)
lemma conversion \(:(a, b) \in\left(A^{\leftrightarrow}\right)^{*} \Longrightarrow(a, b) \in A^{\leftrightarrow *}\)
    by (simp add: conversion-def)
lemma conversion-refl \([\) simp \(]:(a, a) \in A^{\leftrightarrow *}\)
    by (simp add: conversion-def)
lemma conversionI':
    assumes \((a, b) \in A^{*}\) shows \((a, b) \in A^{\leftrightarrow *}\)
using assms
proof (induct)
    case base then show? case by simp
next
    case (step b c)
    then have \((b, c) \in A^{\leftrightarrow}\) by \(\operatorname{simp}\)
    with \(\left\langle(a, b) \in A^{\leftrightarrow *}\right\rangle\) 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:
    \((a, b) \in A^{\leftrightarrow *} \Longrightarrow\left((a, b) \in\left(A^{\leftrightarrow}\right)^{*} \Longrightarrow P\right) \Longrightarrow P\)
    by (simp add: conversion-def)
Later declarations are tried first for 'proof' and 'rule,' then have the "main" introduction / elimination rules for constants should be declared last.
declare joinI-left [intro]
```

```
declare joinI-right [intro]
declare joinI [intro]
declare joinD [dest]
declare joinE [elim]
declare meetI [intro]
declare meetD [dest]
declare meetE [elim]
declare conversionI' [intro]
declare conversionI [intro]
declare conversionE [elim]
lemma conversion-trans:
    trans ( }\mp@subsup{A}{}{\leftrightarrow*}\mathrm{ )
    unfolding trans-def
proof (intro allI impI)
    fix abc assume (a,b)\inA\leftrightarrow* and (b,c)\inA⿱**
    then show (a,c)\inA\leftrightarrow* unfolding conversion-def
    proof (induct)
        case base then show ?case by simp
    next
        case (step b c')
        from }\langle(b,\mp@subsup{c}{}{\prime})\in\mp@subsup{A}{}{\leftrightarrow}\rangle\mathrm{ and }\langle(\mp@subsup{c}{}{\prime},c)\in(\mp@subsup{A}{}{\leftrightarrow}\mp@subsup{)}{}{*}
            have (b,c)\in(A\leftrightarrow)* by (rule converse-rtrancl-into-rtrancl)
        with step show ?case by simp
    qed
qed
lemma conversion-sym:
    sym ( }A⿱\leftrightarrow**
    unfolding sym-def
proof (intro allI impI)
    fix ab assume ( }a,b)\in\mp@subsup{A}{}{\leftrightarrow**}\mathrm{ then show ( }b,a)\in\mp@subsup{A}{}{\leftrightarrow**}\mathrm{ unfolding conversion-def
    proof (induct)
        case base then show ?case by simp
    next
        case (step b c)
        then have (c,b) \in A}\leftrightarrow\mathrm{ by blast
        from < (c,b) \inA\leftrightarrow\rangle}\mathrm{ and }\langle(b,a)\in(\mp@subsup{A}{}{\leftrightarrow}\mp@subsup{)}{}{*}
            show ?case by (rule converse-rtrancl-into-rtrancl)
    qed
qed
lemma conversion-inv:
    (x,y)\in\mp@subsup{R}{}{\leftrightarrow*}\longleftrightarrow(y,x)\in\mp@subsup{R}{}{\leftrightarrow*}
    by (auto simp: conversion-def)
        (metis (full-types) rtrancl-converseD symcl-converse)+
```

```
lemma conversion-converse [simp]:
    \(\left(A^{\leftrightarrow *}\right)^{-1}=A^{\leftrightarrow *}\)
    by (metis conversion-sym sym-conv-converse-eq)
lemma conversion-rtrancl [simp]:
    \(\left(A^{\leftrightarrow *}\right)^{*}=A^{\leftrightarrow *}\)
    by (metis conversion-def rtrancl-idemp)
lemma rtrancl-join-join:
    assumes \((a, b) \in A^{*}\) and \((b, c) \in A^{\downarrow}\) shows \((a, c) \in A^{\downarrow}\)
proof -
    from \(\left\langle(b, c) \in A^{\downarrow}\right\rangle\) obtain \(b^{\prime}\) where \(\left(b, b^{\prime}\right) \in A^{*}\) and \(\left(b^{\prime}, c\right) \in\left(A^{-1}\right)^{*}\)
        unfolding join-def by blast
    with \(\left\langle(a, b) \in A^{*}\right\rangle\) have \(\left(a, b^{\prime}\right) \in A^{*}\) by \(\operatorname{simp}\)
    with \(\left\langle\left(b^{\prime}, c\right) \in\left(A^{-1}\right)^{*}\right\rangle\) show ?thesis unfolding join-def by blast
qed
lemma join-rtrancl-join:
    assumes \((a, b) \in A^{\downarrow}\) and \((c, b) \in A^{*}\) shows \((a, c) \in A^{\downarrow}\)
proof -
    from \(\left\langle(c, b) \in A^{*}\right\rangle\) have \((b, c) \in\left(A^{-1}\right)^{*}\) unfolding rtrancl-converse by simp
    from \(\left\langle(a, b) \in A^{\downarrow}\right\rangle\) obtain \(a^{\prime}\) where \(\left(a, a^{\prime}\right) \in A^{*}\) and \(\left(a^{\prime}, b\right) \in\left(A^{-1}\right)^{*}\)
        unfolding join-def by best
    with \(\left\langle(b, c) \in\left(A^{-1}\right)^{*}\right\rangle\) have \(\left(a^{\prime}, c\right) \in\left(A^{-1}\right)^{*}\) by simp
    with \(\left\langle\left(a, a^{\prime}\right) \in A^{*}\right\rangle\) show ?thesis unfolding join-def by blast
qed
lemma NF-I: \((\bigwedge b .(a, b) \notin A) \Longrightarrow a \in N F A\) by (auto intro: no-step)
lemma \(N F-E: a \in N F A \Longrightarrow((a, b) \notin A \Longrightarrow P) \Longrightarrow P\) by (auto simp: NF-def)
declare NF-I [intro]
declare NF-E [elim]
lemma NF-no-step: \(a \in N F A \Longrightarrow \forall b .(a, b) \notin A\) by auto
lemma NF-anti-mono:
    assumes \(A \subseteq B\) shows \(N F B \subseteq N F A\)
    using assms by auto
lemma NF-iff-no-step: \(a \in N F A=(\forall b .(a, b) \notin A)\) by auto
lemma NF-no-trancl-step:
    assumes \(a \in N F A\) shows \(\forall b\). \((a, b) \notin A^{+}\)
proof -
    from assms have \(\forall b .(a, b) \notin A\) by auto
    show ?thesis
    proof (intro allI notI)
```

```
    fix b assume (a,b) \in A+
    then show False by (induct) (auto simp: \langle\forallb. (a,b)\not\inA\rangle)
    qed
qed
```

lemma $N F$-Id-on-fst-image $[$ simp $]$ : $N F(I d-o n(f s t ‘ A))=N F A$ by force
lemma fst-image-NF-Id-on [simp]: fst' $R=Q \Longrightarrow N F(I d-o n ~ Q)=N F R$ by
force
lemma $N F$-empty $[$ simp $]: N F\{ \}=$ UNIV by auto
lemma normalizability- $I:(a, b) \in A^{*} \Longrightarrow b \in N F A \Longrightarrow(a, b) \in A^{!}$ by (simp add: normalizability-def)
lemma normalizability- $I^{\prime}:(a, b) \in A^{*} \Longrightarrow(b, c) \in A^{!} \Longrightarrow(a, c) \in A^{!}$
by (auto simp add: normalizability-def)
lemma normalizability- $E:(a, b) \in A^{!} \Longrightarrow\left((a, b) \in A^{*} \Longrightarrow b \in N F A \Longrightarrow P\right) \Longrightarrow$ $P$ by (simp add: normalizability-def)
declare normalizability- $I^{\prime}$ [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 $C R$-on :: 'a rel $\Rightarrow$ 'a set $\Rightarrow$ bool where

$$
C R \text {-on } r A \longleftrightarrow\left(\forall a \in A . \forall b c .(a, b) \in r^{*} \wedge(a, c) \in r^{*} \longrightarrow(b, c) \in \text { join } r\right)
$$

abbreviation $C R$ :: 'a rel $\Rightarrow$ bool where
$C R r \equiv C R$-on $r$ UNIV
definition $S N$-on :: 'a rel $\Rightarrow{ }^{\prime}$ 'a set $\Rightarrow$ bool where
$S N$-on $r A \longleftrightarrow \neg(\exists f . f 0 \in A \wedge$ chain $r f)$
abbreviation $S N$ :: 'a rel $\Rightarrow$ bool where SN $r \equiv S N$-on $r$ UNIV

Alternative definition of $S N$.
lemma $S N$-def: $S N r=(\forall x . S N$-on $r\{x\})$
unfolding $S N$-on-def by blast
definition $U N F$-on :: 'a rel $\Rightarrow$ 'a set $\Rightarrow$ bool where
UNF-on $r A \longleftrightarrow\left(\forall a \in A . \forall b c .(a, b) \in r^{!} \wedge(a, c) \in r^{!} \longrightarrow b=c\right)$

```
abbreviation UNF :: 'a rel => bool where UNF r \equivUNF-on r UNIV
definition WCR-on :: 'a rel # 'a set }=>\mathrm{ bool where
    WCR-on r A}\longleftrightarrow(\foralla\inA.\forallbc. (a,b)\inr^(a,c)\inr\longrightarrow(b,c)\injoin r
abbreviation WCR :: 'a rel => bool where WCR r \equivWCR-on r UNIV
definition WN-on :: 'a rel # 'a set }=>\mathrm{ bool where
    WN-on r A \longleftrightarrow(\foralla\inA.\existsb. (a,b)\in\mp@subsup{r}{}{!})
abbreviation WN :: 'a rel }=>\mathrm{ bool where
    WNr\equivWN-on r UNIV
lemmas CR-defs = CR-on-def
lemmas }SN\mathrm{ -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 }=>\mathrm{ bool where
    complete-on r A \longleftrightarrowSN-on r A ^CR-on r A
abbreviation complete :: 'a rel => bool where
    complete r \equivcomplete-on r UNIV
definition semi-complete-on :: 'a rel }=>\mathrm{ ' 'a set }=>\mathrm{ bool where
    semi-complete-on r }A\longleftrightarrowWN-on r A ^CR-on r A
abbreviation semi-complete :: 'a rel }=>\mathrm{ bool where
    semi-complete r\equiv 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 }=>\mathrm{ bool where
    UNCA\longleftrightarrow(\forallab.a\inNFA\wedgeb\inNFA\wedge(a,b)\inA\leftrightarrow* \longrightarrowa=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 \Longrightarrow(SN-on r A \LongrightarrowCR-on r A \LongrightarrowP)\LongrightarrowP
    by (simp add: complete-defs)
lemma CR-onI:
    (\bigwedgeabc.a }\inA\Longrightarrow(a,b)\in\mp@subsup{r}{}{*}\Longrightarrow(a,c)\in\mp@subsup{r}{}{*}\Longrightarrow(b,c)\injoin r)\LongrightarrowCR-o
```

```
r A
    by (simp add: CR-defs)
lemma CR-on-singletonI:
    (\bigwedgebc. (a,b) \in r* \Longrightarrow(a,c)\in r* \Longrightarrow(b,c)\injoin r)\LongrightarrowCR-on r {a}
    by (simp add: CR-defs)
lemma CR-onE:
    CR-on r A\Longrightarrowa\inA\Longrightarrow((b,c) f join r\LongrightarrowP)\Longrightarrow((a,b)\not\inr* \LongrightarrowP)\Longrightarrow
((a,c)\not\in\mp@subsup{r}{}{*}\LongrightarrowP)\LongrightarrowP
    unfolding CR-defs by blast
lemma CR-onD:
    CR-on r A \Longrightarrowa\inA \Longrightarrow(a,b) \in r* \Longrightarrow(a,c)\in r* \Longrightarrow(b,c)\in join r
    by (blast elim:CR-onE)
lemma semi-complete-onI:WN-on r A \LongrightarrowCR-on r A \Longrightarrow semi-complete-on r A
    by (simp add: semi-complete-defs)
lemma semi-complete-onE:
    semi-complete-on r A \Longrightarrow(WN-on r A C CR-on r A \LongrightarrowP)\LongrightarrowP
    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:
    (\bigwedgeab. a \inNFA\Longrightarrowb\inNFA\Longrightarrow (a,b) \in A⿱* \Longrightarrow \Longrightarrowa=b)\LongrightarrowUNC A
    by (simp add:UNC-def)
lemma UNC-E:
    \llbracketUNCA;a=b\LongrightarrowP;a\not\inNFA\LongrightarrowP;b\not\inNFA\LongrightarrowP;(a,b)\not\inA\leftrightarrow* \Longrightarrow
P\\LongrightarrowP
    unfolding UNC-def by blast
lemma UNF-onI:(\bigwedgeabc. a }\inA\Longrightarrow(a,b)\in\mp@subsup{r}{}{!}\Longrightarrow(a,c)\in\mp@subsup{r}{}{!}\Longrightarrowb=c)
UNF-on r A
    by (simp add:UNF-defs)
lemma UNF-onE:
```

```
    UNF-on r A \Longrightarrowa\inA \Longrightarrow (b=c\LongrightarrowP)\Longrightarrow((a,b)\not\in r \ \LongrightarrowP)\Longrightarrow((a,c)
\notin r ^ { ! } \Longrightarrow P ) \Longrightarrow P
    unfolding UNF-on-def by blast
lemma UNF-onD:
    UNF-on r A \Longrightarrowa\inA \Longrightarrow (a,b) \inr! \Longrightarrow(a,c)\inr! \Longrightarrowb=c
    by (blast elim: UNF-onE)
declare UNF-onI [intro]
declare UNF-onD [dest]
declare UNF-onE [elim]
lemma SN-onI:
    assumes \f.\llbracketf 0 \in A; chain r f\rrbracket\Longrightarrow False
    shows SN-on r A
    using assms unfolding SN-defs by blast
lemma SN-I:(\bigwedgea.SN-on A {a})\LongrightarrowSN A
    unfolding SN-on-def by blast
lemma SN-on-trancl-imp-SN-on:
    assumes SN-on ( }\mp@subsup{R}{}{+}\mathrm{ ) T shows SN-on R T
proof (rule ccontr)
    assume }\negSN\mathrm{ -on R T
    then obtain s where s 0 G T and chain R s unfolding SN-defs by auto
    then have chain ( }\mp@subsup{R}{}{+}\mathrm{ ) s by auto
    with «s 0 \inT` have }\neg\mathrm{ SN-on ( }\mp@subsup{R}{}{+}\mathrm{ )T unfolding SN-defs by auto
    with assms show False by simp
qed
lemma SN-onE:
    assumes SN-on r A
        and }\neg(\existsf.f0\inA\wedge chain rf)\Longrightarrow
    shows P
    using assms unfolding SN-defs by simp
lemma not-SN-onE:
    assumes }\neg\textrm{SN}\mathrm{ -on r A
        and }\bigwedgef.\llbracketf0\inA;chain rf\rrbracket\Longrightarrow
    shows P
    using assms unfolding SN-defs by blast
declare SN-onI [intro]
declare SN-onE [elim]
declare not-SN-onE [Pure.elim, elim]
lemma refl-not-SN: (x,x) \inR\Longrightarrow\negSNR
    unfolding SN-defs by force
```

```
lemma \(S N\)-on-irrefl:
    assumes \(S N\)-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 \(S N\)-defs by auto
qed
lemma WCR-onI: (\abc. \(a \in A \Longrightarrow(a, b) \in r \Longrightarrow(a, c) \in r \Longrightarrow(b, c) \in j\) join
\(r) \Longrightarrow W C R\)-on \(r A\)
    by (simp add: WCR-defs)
lemma WCR-onE:
    WCR-on \(r A \Longrightarrow a \in A \Longrightarrow((b, c) \in\) join \(r \Longrightarrow P) \Longrightarrow((a, b) \notin r \Longrightarrow P) \Longrightarrow\)
\(((a, c) \notin r \Longrightarrow P) \Longrightarrow P\)
    unfolding WCR-on-def by blast
lemma \(S N\)-nat-bounded: \(S N\{(x, y:: n a t) . x<y \wedge y \leq b\}\) (is \(S N\) ?R)
proof
    fix \(f\)
    assume chain ?R f
    then have steps: \(\bigwedge i .(f i, f(S u c i)) \in ? R .\).
    \{
        fix \(i\)
        have inc: f0+isfi
        proof (induct i)
            case 0 then show ? case by auto
        next
            case (Suc i)
            have \(f 0+\) Suc \(i \leq f i+\) Suc 0 using Suc by simp
            also have \(\ldots \leq f\) (Suc \(i\) ) using steps \([\) of \(i]\)
                by auto
            finally show? ?ase by simp
        qed
    \}
    from this [of Suc b] steps [of b]
    show False by simp
qed
lemma \(W C R\)-onD:
    WCR-on \(r A \Longrightarrow a \in A \Longrightarrow(a, b) \in r \Longrightarrow(a, c) \in r \Longrightarrow(b, c) \in\) join \(r\)
    by (blast elim: WCR-onE)
lemma \(W N\)-onI: \(\left(\bigwedge a . a \in A \Longrightarrow \exists b .(a, b) \in r^{!}\right) \Longrightarrow W N\)-on \(r A\)
    by (auto simp: WN-defs)
lemma \(W N\)-onE: \(W N\)-on \(r A \Longrightarrow a \in A \Longrightarrow\left(\bigwedge b .(a, b) \in r^{!} \Longrightarrow P\right) \Longrightarrow P\)
    unfolding \(W N\)-defs by blast
```

```
lemma WN-onD:WN-on r A \Longrightarrowa\inA\Longrightarrow\existsb. (a,b)\inr!
    by (blast elim: WN-onE)
declare WCR-onI [intro]
declare WCR-onD [dest]
declare WCR-onE [elim]
declare WN-onI [intro]
declare WN-onD [dest]
declare WN-onE [elim]
Restricting a relation \(r\) to those elements that are strongly normalizing with respect to a relation \(s\).
```

```
definition restrict-SN :: 'a rel # 'a rel # 'a rel where
```

definition restrict-SN :: 'a rel \# 'a rel \# 'a rel where
restrict-SNrs={(a,b)|ab. (a,b)\inr\wedgeSN-on s {a}}
restrict-SNrs={(a,b)|ab. (a,b)\inr\wedgeSN-on s {a}}
lemma SN-restrict-SN-idemp [simp]: SN (restrict-SN A A)
lemma SN-restrict-SN-idemp [simp]: SN (restrict-SN A A)
by (auto simp: restrict-SN-def SN-defs)
by (auto simp: restrict-SN-def SN-defs)
lemma SN-on-Image:
lemma SN-on-Image:
assumes SN-on r A
assumes SN-on r A
shows SN-on r (r"A)
shows SN-on r (r"A)
proof
proof
fix f
fix f
assume f 0 \inr" A and chain: chain r f
assume f 0 \inr" A and chain: chain r f
then obtain a where a\inA and 1:(a,f0)\inr by auto
then obtain a where a\inA and 1:(a,f0)\inr by auto
let ?g = case-nat a f
let ?g = case-nat a f
from cons-chain [OF 1 chain] have chain r ?g.
from cons-chain [OF 1 chain] have chain r ?g.
moreover have ?g 0 \in A by (simp add: <a \inA`)     moreover have ?g 0 \in A by (simp add: <a \inA`)
ultimately have }\neg\mathrm{ SN-on r A unfolding SN-defs by best
ultimately have }\neg\mathrm{ SN-on r A unfolding SN-defs by best
with assms show False by simp
with assms show False by simp
qed
qed
lemma SN-on-subset2:
lemma SN-on-subset2:
assumes A\subseteqB and SN-on r B
assumes A\subseteqB and SN-on r B
shows SN-on r A
shows SN-on r A
using assms unfolding SN-on-def by blast
using assms unfolding SN-on-def by blast
lemma step-preserves-SN-on:
lemma step-preserves-SN-on:
assumes 1: (a,b)\inr
assumes 1: (a,b)\inr
and 2:SN-on r {a}
and 2:SN-on r {a}
shows SN-on r {b}
shows SN-on r {b}
using 1 and SN-on-Image [OF 2] and SN-on-subset2 [of {b}r"{a}] by auto
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*\LongrightarrowSN-on A {a}\LongrightarrowSN-on A {b}
lemma steps-preserve-SN-on: (a,b)\in A*\LongrightarrowSN-on A {a}\LongrightarrowSN-on A {b}
by (induct rule: rtrancl.induct) (auto simp: step-preserves-SN-on)

```
    by (induct rule: rtrancl.induct) (auto simp: step-preserves-SN-on)
```

lemma relpow-seq:

```
    assumes (x,y)\inr^n
    shows \existsf.f0=x^fn=y^(\foralli<n.(fi,f(Suc i))\inr)
using assms
proof (induct n arbitrary: y)
    case 0 then show ?case by auto
next
    case (Suc n)
    then obtain z}\mathrm{ where (x,z) & r^n and (z,y) fr by auto
    from Suc(1)[OF〈(x,z)\in r^n}n>
        obtain f}\mathrm{ where f 0=x and fn=z and seq: }\foralli<n.(fi,f(Suc i))\inr by
auto
    let ?n = Suc n
    let ?f = \lambdai. if i=?n then y else fi
    have ?f ?n = y by simp
    from <f 0 = x have ?f 0 =x by simp
    from seq have seq': }\foralli<n\mathrm{ . (?f i, ?f (Suc i)) &r by auto
    with }\langlefn=z\rangle\mathrm{ and }\langle(z,y)\inr\rangle\mathrm{ have }\foralli<?n\mathrm{ . (?f i, ?f (Suc i)) }\inr\mathrm{ by auto
    with〈?f 0 = x〉 and <?f ? n = y〉 show ?case by best
qed
lemma rtrancl-imp-seq:
    assumes (x,y)\in r*
    shows \existsfn.f0=x^fn=y^(\foralli<n. (fi,f(Suc i))\inr)
    using assms [unfolded rtrancl-power] and relpow-seq [of x y-r] by blast
lemma SN-on-Image-rtrancl:
    assumes SN-on r A
    shows SN-on r (r* " A)
proof
    fix f
    assume f0: f 0\inr* " A and chain: chain r f
    then obtain }a\mathrm{ where a:a 
    then obtain n where (a,f 0) \inr^~n unfolding rtrancl-power by auto
    show False
    proof (cases n)
        case 0
        with}\langle(a,f0)\in\mp@subsup{r}{}{~}n\rangle\mathrm{ have f 0=a by simp
    then have f0\inA by (simp add: a)
    with chain have }\negSN\mathrm{ -on r A by auto
    with assms show False by simp
    next
    case (Suc m)
    from relpow-seq [OF<<(a,f 0) \in r^^n>]
        obtain g}\mathrm{ where g0:g 0=a and g n=f0
        and gseq: }\foralli<n.(gi,g(Suc i))\inr by aut
    let ?f = \lambdai. if i<n then g i else f (i-n)
    have chain r?f
    proof
        fix }
```

```
    {
        assume Suc i<n
        then have (?f i, ?f (Suc i)) \inr by (simp add: gseq)
    }
    moreover
    {
        assume Suc i>n
        then have eq: Suc (i-n)=Suc i-n by arith
        from chain have (f (i-n),f(Suc (i-n))) \inr by simp
        then have (f(i-n),f(Suc i-n))\inr by (simp add: eq)
        with «Suc i>n\rangle have (?f i, ?f (Suc i)) \inr by simp
    }
    moreover
    {
        assume Suc i=n
        then have eq: f(Suc i - n)=gn by (simp add: <g n = f0`)
        from <Suci=n` have eq':i=n-1 by arith
        from gseq have (g i,f(Suc i-n))\inr unfolding eq by (simp add: Suc
eq')
            then have (?f i, ?f (Suc i)) \inr using \langleSuc i=n> by simp
        }
        ultimately show (?f i, ?f (Suc i)) \inr by simp
    qed
    moreover have ?f 0}\in
    proof (cases n)
        case 0
        with}\langle(a,f0)\inr^~n\rangle have eq: a = f 0 by simp
        from a show ?thesis by (simp add: eq 0)
    next
        case (Suc m)
        then show ?thesis by (simp add: a g0)
    qed
    ultimately have }\neg\mathrm{ SN-on r A unfolding SN-defs by best
    with assms show False by simp
    qed
qed
declare subrelI [Pure.intro]
lemma restrict-SN-trancl-simp [simp]:(restrict-SN A A)+
(is ?lhs = ?rhs)
proof
    show ?lhs \subseteq?rhs
    proof
        fix ab assume (a,b) \in?lhs then show (a,b)\in?rhs
        unfolding restrict-SN-def by (induct rule: trancl.induct) auto
    qed
next
```

```
    show ?rhs \subseteq?lhs
    proof
        fix a b assume (a,b) \in?rhs
```



```
        then show (a,b)\in?lhs
        proof (induct rule: trancl.induct)
            case (r-into-trancl x y) then show ?case unfolding restrict-SN-def by auto
    next
        case (trancl-into-trancl a b c)
        then have IH:(a,b)\in?lhs by auto
        from trancl-into-trancl have (a,b) \in A* by auto
    from this and «SN-on A {a}〉 have SN-on A {b} by (rule steps-preserve-SN-on)
        with}\langle(b,c)\inA\rangle\mathrm{ have (b,c) {?lhs unfolding restrict-SN-def by auto
        with IH show ?case by simp
    qed
    qed
qed
lemma SN-imp-WN:
    assumes SN A shows WN A
proof -
    from \langleSN A〉 have wf ( }\mp@subsup{A}{}{-1})\mathrm{ by (simp add: SN-defs wf-iff-no-infinite-down-chain)
    show WN A
    proof
        fix }
        show }\existsb.(a,b)\in\mp@subsup{A}{}{!}\mathrm{ unfolding normalizability-def NF-def Image-def
            by (rule wfE-min [OF \langlewf ( }\mp@subsup{A}{}{-1})\rangle\mathrm{ , of a A* " {a}, simplified])
                (auto intro: rtrancl-into-rtrancl)
    qed
qed
lemma UNC-imp-UNF:
    assumes UNC r shows UNF r
proof - {
    fix x y z assume (x,y) \inr! and (x,z) \inr!
    then have (x,y)\in\mp@subsup{r}{}{*}\mathrm{ and (x,z) & r* and y fNFr and z A NFr by auto}
    then have (x,y)\in\mp@subsup{r}{}{\leftrightarrow*}\mathrm{ and (x,z) & r r** by auto}
    then have (z,x)\in r }\mp@subsup{}{}{\leftrightarrow*}\mathrm{ using conversion-sym unfolding sym-def by best
    with «(x,y) \in r ↔*` have (z,y) \in r ↔* using conversion-trans unfolding
trans-def by best
    from assms and this and }\langlez\inNFr\rangle\mathrm{ and }\langley\inNFr\rangle have z=y unfoldin
UNC-def by auto
} then show ?thesis by auto
qed
lemma join-NF-imp-eq:
    assumes (x,y)\in r文 and x\inNFr and y\inNFr
    shows }x=
proof -
```

from $\left\langle(x, y) \in r^{\downarrow}\right\rangle$ obtain $z$ where $(x, z) \in r^{*}$ and $(z, y) \in\left(r^{-1}\right)^{*}$ unfolding join-def by auto
then have $(y, z) \in r^{*}$ unfolding rtrancl-converse by simp
from $\langle x \in N F r\rangle$ have $(x, z) \notin r^{+}$using NF-no-trancl-step by best
then have $x=z$ using rtranclD $\left.\left[O F «(x, z) \in r^{*}\right\rangle\right]$ by auto
from $\langle y \in N F r\rangle$ have $(y, z) \notin r^{+}$using NF-no-trancl-step by best then have $y=z$ using rtranclD $\left.\left[O F \prec(y, z) \in r^{*}\right\rangle\right]$ by auto with $\langle x=z\rangle$ show ?thesis by simp
qed
lemma rtrancl-Restr:
assumes $(x, y) \in(\text { Restr } r A)^{*}$
shows $(x, y) \in r^{*}$
using assms by induct auto
lemma join-mono:
assumes $r \subseteq s$
shows $r^{\downarrow} \subseteq s^{\downarrow}$
using rtrancl-mono [OF assms] by (auto simp: join-def rtrancl-converse)

```
lemma CR-iff-meet-subset-join: \(C R \quad r=\left(r^{\uparrow} \subseteq r^{\downarrow}\right)\)
proof
assume \(C R r\) 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 meet \(D\) by best
    with \(\left\langle C R\right.\) r show \((x, y) \in r^{\downarrow}\) by (auto simp: CR-defs)
qed
next
assume \(r^{\uparrow} \subseteq r^{\downarrow}\{\)
    fix \(x y z\) assume \((x, y) \in r^{*}\) and \((x, z) \in r^{*}\)
    then have \((y, z) \in r^{\uparrow}\) unfolding meet-def rtrancl-converse by auto
    with \(\left\langle r^{\uparrow} \subseteq r^{\downarrow}\right\rangle\) have \((y, z) \in r^{\downarrow}\) by auto
    \} then show \(C R r\) by (auto simp: \(C R\)-defs)
qed
lemma \(C R\)-divergence-imp-join:
    assumes \(C R r\) 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 \(\left\langle(z, y) \in r^{\leftrightarrow *}\right\rangle\) have \((y, z) \in r^{\leftrightarrow *}\) using conversion-sym unfolding sym-def
by best
```

with $\left\langle(x, y) \in r^{\leftrightarrow *}\right\rangle$ show $(x, z) \in r^{\leftrightarrow *}$ using conversion-trans unfolding trans-def by best
qed
lemma meet-imp-conversion: $r^{\uparrow} \subseteq r^{\leftrightarrow *}$
proof (rule subrelI)
fix $y z$ assume $(y, z) \in r^{\uparrow}$
then obtain $x$ where $(x, y) \in r^{*}$ and $(x, z) \in r^{*}$ by auto
then have $(x, y) \in r^{\leftrightarrow *}$ and $(x, z) \in r^{\leftrightarrow *}$ by auto
from $\left\langle(x, y) \in r^{\leftrightarrow *}\right\rangle$ have $(y, x) \in r^{\leftrightarrow *}$ using conversion-sym unfolding sym-def by best
with $\left\langle(x, z) \in r^{\leftrightarrow *}\right\rangle$ show $(y, z) \in r^{\leftrightarrow *}$ using conversion-trans unfolding trans-def by best
qed
lemma CR-imp-UNF:
assumes $C R r$ shows $U N F r$
proof - \{
fix $x y z$ assume $(x, y) \in r^{!}$and $(x, z) \in r^{!}$
then have $(x, y) \in r^{*}$ and $y \in N F r$ and $(x, z) \in r^{*}$ and $z \in N F r$
unfolding normalizability-def by auto
from assms and $\left\langle(x, y) \in r^{*}\right\rangle$ and $\left\langle(x, z) \in r^{*}\right\rangle$ have $(y, z) \in r^{\downarrow}$
by (rule CR-divergence-imp-join)
from this and $\langle y \in N F r\rangle$ and $\langle z \in N F r\rangle$ have $y=z$ by (rule join-NF-imp-eq)
\} then show ?thesis by auto
qed
lemma CR-iff-conversion-imp-join: $C R \quad r=\left(r^{\leftrightarrow *} \subseteq r^{\downarrow}\right)$
proof (intro iffI subrelI)
fix $x y$ assume $C R r$ and $(x, y) \in r^{\leftrightarrow *}$
then obtain $n$ where $(x, y) \in\left(r^{\leftrightarrow}\right)^{\wedge} n$ unfolding conversion-def rtrancl-is-UN-relpow
by auto
then show $(x, y) \in r^{\downarrow}$
proof (induct $n$ arbitrary: $x$ )
case 0
assume $(x, y) \in r^{\leftrightarrow} \leadsto 0$ then have $x=y$ by $\operatorname{simp}$
show ?case unfolding $\langle x=y\rangle$ by auto
next
case (Suc n)
from $\left\langle(x, y) \in r^{\leftrightarrow} \sim\right.$ Suc $\left.n\right\rangle$ obtain $z$ where $(x, z) \in r^{\leftrightarrow}$ and $(z, y) \in r^{\leftrightarrow}$
~n
using relpow-Suc-D2 by best
with Suc have $(z, y) \in r^{\downarrow}$ by simp
from $\left\langle(x, z) \in r^{\leftrightarrow}\right\rangle$ show ?case
proof
assume $(x, z) \in r$ with $\left\langle(z, y) \in r^{\downarrow}\right\rangle$ show ?thesis by (auto intro: rtrancl-join-join)
next
assume $(x, z) \in r^{-1}$
then have $(z, x) \in r^{*}$ by $\operatorname{simp}$

```
        from }\langle(z,y)\in\mp@subsup{r}{}{\downarrow}\rangle\mathrm{ obtain }\mp@subsup{z}{}{\prime}\mathrm{ where (z, z') G r* and (y, z') G r* by auto
        from \langleCR r\rangle and }\langle(z,x)\in\mp@subsup{r}{}{*}\rangle\mathrm{ and }\langle(z,\mp@subsup{z}{}{\prime})\in\mp@subsup{r}{}{*}\rangle\mathrm{ have }(x,\mp@subsup{z}{}{\prime})\in\mp@subsup{r}{}{\downarrow
            by (rule CR-divergence-imp-join)
            then obtain \mp@subsup{x}{}{\prime}}\mathrm{ where (x, x') & r *}\mathrm{ and ( }\mp@subsup{z}{}{\prime},\mp@subsup{x}{}{\prime})\in\mp@subsup{r}{}{*}\mathrm{ by auto
            with «(y, z')\in r*` show ?thesis by auto
        qed
    qed
next
    assume rr** \subseteqr r}\downarrow\mathrm{ then show CRr unfolding CR-iff-meet-subset-join
        using meet-imp-conversion by auto
qed
lemma CR-imp-conversionIff-join:
    assumes CR r shows rr* = r r
proof
    show rr*}\subseteq\mp@subsup{r}{}{\downarrow}\mathrm{ using CR-iff-conversion-imp-join assms by auto
next
    show }\mp@subsup{r}{}{\downarrow}\subseteq\mp@subsup{r}{}{\leftrightarrow*}\mathrm{ by (rule join-imp-conversion)
qed
lemma sym-join: sym (join r) by (auto simp: sym-def)
lemma join-sym: (s,t)\in A }\\Longrightarrow(t,s)\in\mp@subsup{A}{}{\downarrow}\mathrm{ by auto
lemma CR-join-left-I:
    assumes CR r and (x,y)\in r* and (x,z)\in r ' shows (y,z) \in r r
proof -
    from }\langle(x,z)\in\mp@subsup{r}{}{\downarrow}\rangle\mathrm{ obtain }\mp@subsup{x}{}{\prime}\mathrm{ where (x, x') & r r
    from \langleCR r\rangle and }\langle(x,\mp@subsup{x}{}{\prime})\in\mp@subsup{r}{}{*}\rangle\mathrm{ and }\langle(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have (x,y) & r r
    then have (y,x)\in\mp@subsup{r}{}{\downarrow}\mathrm{ using join-sym by best}\0\mathrm{ jon}
    from 〈CR r〉 have r r** = r '\downarrow by (rule CR-imp-conversionIff-join)
    from }\langle(y,x)\in\mp@subsup{r}{}{\downarrow}\rangle\mathrm{ and }\langle(x,z)\in\mp@subsup{r}{}{\downarrow}\rangle\mathrm{ show ?thesis using conversion-trans
        unfolding trans-def \langler** = r r}\downarrow>>[symmetric] by bes
qed
lemma CR-join-right-I:
    assumes CR r and (x,y)\in r}\downarrow\mathrm{ and (y,z) & r* shows (x,z) & r r
proof -
    have }\mp@subsup{r}{}{\leftrightarrow*}=\mp@subsup{r}{}{\downarrow}\mathrm{ by (rule CR-imp-conversionIff-join [OF<CR r〉])
    from «(y,z)\in r*}>\mathrm{ have (y,z) & r ↔* by auto
        with }\langle(x,y)\in\mp@subsup{r}{}{\downarrow}\rangle\mathrm{ show ?thesis unfolding <r }\mp@subsup{r}{}{\leftrightarrow*}=\mp@subsup{r}{}{\downarrow}\rangle[\mathrm{ [symmetric] using
conversion-trans
        unfolding trans-def by fast
qed
lemma NF-not-suc:
    assumes (x,y)\in r* and x\inNFr shows }x=
proof -
    from <x \inNF r> have }\forally.(x,y)\not\inr\mathrm{ using NF-no-step by auto
```

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

```
lemma semi-complete-imp-conversionIff-same-NF:
    assumes semi-complete \(r\)
    shows \(\left((x, y) \in r^{\leftrightarrow *}\right)=\left(\forall u v .(x, u) \in r^{!} \wedge(y, v) \in r^{!} \longrightarrow u=v\right)\)
proof -
    from assms have \(W N r\) and \(C R r\) unfolding semi-complete-defs by auto
    then have \(r^{\leftrightarrow *}=r^{\downarrow}\) using CR-imp-conversionIff-join by auto
    show ?thesis
    proof
    assume \((x, y) \in r^{\leftrightarrow *}\)
    from \(\left\langle(x, y) \in r^{\leftrightarrow *}\right\rangle\) have \((x, y) \in r^{\downarrow}\) unfolding \(\left\langle r^{\leftrightarrow *}=r^{\downarrow}\right\rangle\).
    show \(\forall u v .(x, u) \in r^{!} \wedge(y, v) \in r^{!} \longrightarrow u=v\)
    proof (intro allI impI, elim conjE)
            fix \(u v\) assume \((x, u) \in r^{!}\)and \((y, v) \in r^{!}\)
            then have \((x, u) \in r^{*}\) and \((y, v) \in r^{*}\) and \(u \in N F r\) and \(v \in N F r\) by auto
            from \(\langle C R r\rangle\) and \(\left\langle(x, u) \in r^{*}\right\rangle\) and \(\left\langle(x, y) \in r^{\downarrow}\right\rangle\) have \((u, y) \in r^{\downarrow}\)
                by (auto intro: CR-join-left-I)
            then have \((y, u) \in r^{\downarrow}\) using join-sym by best
            with \(\left\langle(x, y) \in r^{\downarrow}\right\rangle\) have \((x, u) \in r^{\downarrow}\) unfolding \(\left\langle r^{\leftrightarrow *}=r^{\downarrow}\right\rangle\) [symmetric]
                using conversion-trans unfolding trans-def by best
            from \(\langle C R r\rangle\) and \(\left\langle(x, y) \in r^{\downarrow}\right\rangle\) and \(\left\langle(y, v) \in r^{*}\right\rangle\) have \((x, v) \in r^{\downarrow}\)
                by (auto intro: CR-join-right-I)
            then have \((v, x) \in r^{\downarrow}\) using join-sym unfolding sym-def by best
            with \(\left\langle(x, u) \in r^{\downarrow}\right\rangle\) have \((v, u) \in r^{\downarrow}\) unfolding \(\left\langle r^{\leftrightarrow *}=r^{\downarrow}\right\rangle\) [symmetric]
                using conversion-trans unfolding trans-def by best
            then obtain \(v^{\prime}\) where \(\left(v, v^{\prime}\right) \in r^{*}\) and \(\left(u, v^{\prime}\right) \in r^{*}\) by auto
            from \(\left\langle\left(u, v^{\prime}\right) \in r^{*}\right\rangle\) and \(\langle u \in N F r\rangle\) have \(u=v^{\prime}\) by (rule NF-not-suc)
            from \(\left\langle\left(v, v^{\prime}\right) \in r^{*}\right\rangle\) and \(\langle v \in N F r\rangle\) have \(v=v^{\prime}\) by (rule NF-not-suc)
            then show \(u=v\) unfolding \(\left\langle u=v^{\prime}\right\rangle\) by simp
    qed
    next
    assume equal-NF: \(\forall u v .(x, u) \in r^{!} \wedge(y, v) \in r^{!} \longrightarrow u=v\)
    from \(\langle W N r\rangle\) obtain \(u\) where \((x, u) \in r^{!}\)by auto
    from \(\langle W N r\rangle\) obtain \(v\) where \((y, v) \in r^{!}\)by auto
    from \(\left\langle(x, u) \in r!\right.\) and \(\left\langle(y, v) \in r^{!}\right\rangle\)have \(u=v\) using equal-NF by simp
    from \(\left\langle(x, u) \in r^{!}\right\rangle\)and \(\left\langle(y, v) \in r^{!}\right\rangle\)have \((x, v) \in r^{*}\) and \((y, v) \in r^{*}\)
        unfolding \(\langle u=v\rangle\) by auto
    then have \((x, v) \in r^{\leftrightarrow *}\) and \((y, v) \in r^{\leftrightarrow *}\) by auto
            from \(\left\langle(y, v) \in r^{\leftrightarrow *}\right\rangle\) have \((v, y) \in r^{\leftrightarrow *}\) using conversion-sym unfolding
sym-def by best
            with \(\left\langle(x, v) \in r^{\leftrightarrow *}\right.\) show \((x, y) \in r^{\leftrightarrow *}\) using conversion-trans unfolding
trans-def by best
    qed
qed
```

```
lemma CR-imp-UNC:
    assumes CR r shows UNCr
proof - {
    fix x y assume x\inNF r and y\inNFr and (x,y)\in r ↔*
    have }\mp@subsup{r}{}{\leftrightarrow*}=\mp@subsup{r}{}{\downarrow}\mathrm{ by (rule CR-imp-conversionIff-join [OF assms])
```



```
    then obtain }\mp@subsup{x}{}{\prime}\mathrm{ where (x, x) & r** and ( }y,\mp@subsup{x}{}{\prime})\in\mp@subsup{r}{}{*}\mathrm{ by best
    from }\langle(x,\mp@subsup{x}{}{\prime})\in\mp@subsup{r}{}{*}\rangle\mathrm{ and }\langlex\inNFr> have x=\mp@subsup{x}{}{\prime}\mathrm{ by (rule NF-not-suc)
    from «(y, x) \in r'* and }\langley\inNFr> have y=\mp@subsup{x}{}{\prime}\mathrm{ by (rule NF-not-suc)
    then have }x=y\mathrm{ unfolding }\langlex=\mp@subsup{x}{}{\prime}\rangle\mathrm{ by simp
} then show ?thesis by (auto simp: UNC-def)
qed
lemma WN-UNF-imp-CR:
    assumes WNr and UNF r shows CR r
proof - {
    fix }xyz\mathrm{ assume (x,y) & r* and (x,z) & r*
    from assms obtain }\mp@subsup{y}{}{\prime}\mathrm{ where (y, y') & r! unfolding WN-defs by best
    with }\langle(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have ( }x,\mp@subsup{y}{}{\prime})\in\mp@subsup{r}{}{!}\mathrm{ by auto
    from assms obtain }\mp@subsup{z}{}{\prime}\mathrm{ where (z,z') & !! unfolding WN-defs by best
    with }{(x,z)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have (x, z') & r! by auto
    with «(x, y')\in r'` have }\mp@subsup{y}{}{\prime}=\mp@subsup{z}{}{\prime}\mathrm{ using <UNF r〉 unfolding UNF-defs by auto
    from }\langle(y,\mp@subsup{y}{}{\prime})\in\mp@subsup{r}{}{!}\rangle\mathrm{ and }\langle(z,\mp@subsup{z}{}{\prime})\in\mp@subsup{r}{}{!}\rangle\mathrm{ have }(y,z)\in\mp@subsup{r}{}{\downarrow}\mathrm{ unfolding }\langle\mp@subsup{y}{}{\prime}=\mp@subsup{z}{}{\prime}\rangle\mathrm{ by
auto
} then show ?thesis by auto
qed
definition diamond :: 'a rel }=>\mathrm{ bool ( }\diamond\mathrm{ ) where
    \diamond r \longleftrightarrow ( r ^ { - 1 } O r ) \subseteq ( r O r ^ { - 1 } )
```

lemma diamond- $I$ [intro]: $\left(r^{-1} O r\right) \subseteq\left(r O r^{-1}\right) \Longrightarrow \diamond r$ unfolding diamond-def
by $\operatorname{simp}$
lemma diamond- $E\left[\right.$ elim] $: \diamond r \Longrightarrow\left(\left(r^{-1} O r\right) \subseteq\left(r O r^{-1}\right) \Longrightarrow P\right) \Longrightarrow P$
unfolding diamond-def by simp
lemma diamond-imp-semi-confluence:
assumes $\diamond r$ shows $\left(r^{-1} O r^{*}\right) \subseteq r^{\downarrow}$
proof (rule subrelI)
fix $y z$ assume $(y, z) \in r^{-1} O r^{*}$
then obtain $x$ where $(x, y) \in r$ and $(x, z) \in r^{*}$ by best
then obtain $n$ where $(x, z) \in r^{\wedge} n$ using rtrancl-imp-UN-relpow by best
with $\langle(x, y) \in r\rangle$ show $(y, z) \in r^{\downarrow}$
proof (induct $n$ arbitrary: $x z y$ )
case 0 then show ?case by auto
next
case (Suc n)
from $\left\langle(x, z) \in r^{\sim}\right.$ Suc $\left.n\right\rangle$ obtain $x^{\prime}$ where $\left(x, x^{\prime}\right) \in r$ and $\left(x^{\prime}, z\right) \in r^{\wedge} n$
using relpow-Suc-D2 by best

```
    with }\langle(x,y)\inr\rangle\mathrm{ have (y, x') &( (r - \ Orr) by auto
    with \langle\diamond r\rangle have (y, x) \in(rO r-1) by auto
    then obtain }\mp@subsup{y}{}{\prime}\mathrm{ where ( }\mp@subsup{x}{}{\prime},\mp@subsup{y}{}{\prime})\inr\mathrm{ and (y, y') fr by best
    with Suc and }\langle(\mp@subsup{x}{}{\prime},z)\in\mp@subsup{r}{}{~}n\rangle have ( (\mp@subsup{y}{}{\prime},z)\in\mp@subsup{r}{}{\downarrow}\mathrm{ by auto
    with }\langle(y,\mp@subsup{y}{}{\prime})\inr\rangle\mathrm{ show ?case by (auto intro: rtrancl-join-join)
    qed
qed
lemma semi-confluence-imp-CR:
    assumes (r}\mp@subsup{r}{}{-1}O\mp@subsup{r}{}{*})\subseteq\mp@subsup{r}{}{\downarrow}\mathrm{ shows CR r
proof - {
    fix x y z assume (x,y)\in r* and (x,z) \in r*
    then obtain n where (x,z)\inr^n
    with }<(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have ( }y,z)\in\mp@subsup{r}{}{\downarrow
    proof (induct n arbitrary: x y z)
        case 0 then show ?case by auto
    next
        case (Suc n)
```



```
            using relpow-Suc-D2 by best
        from }\langle(x,\mp@subsup{x}{}{\prime})\inr\rangle\mathrm{ and }\langle(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have ( }\mp@subsup{x}{}{\prime},y)\in(\mp@subsup{r}{}{-1}O\mp@subsup{r}{}{*})\mathrm{ by auto
        with assms have ( }\mp@subsup{x}{}{\prime},y)\in\mp@subsup{r}{}{\downarrow}\mathrm{ by auto
        then obtain }\mp@subsup{y}{}{\prime}\mathrm{ where ( }\mp@subsup{x}{}{\prime},\mp@subsup{y}{}{\prime})\in\mp@subsup{r}{}{*}\mathrm{ and ( }y,\mp@subsup{y}{}{\prime})\in\mp@subsup{r}{}{*}\mathrm{ by best
        with Suc and «( (x,z) \in r^~
        then obtain u where (z,u)\in\mp@subsup{r}{}{*}\mathrm{ and ( }\mp@subsup{y}{}{\prime},u)\in\mp@subsup{r}{}{*}\mathrm{ by best}
        from }\langle(y,\mp@subsup{y}{}{\prime})\in\mp@subsup{r}{}{*}\rangle\mathrm{ and }\langle(\mp@subsup{y}{}{\prime},u)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have }(y,u)\in\mp@subsup{r}{}{*}\mathrm{ by auto
        with }\langle(z,u)\in\mp@subsup{r}{}{*}\rangle\mathrm{ show ?case by best
    qed
} then show ?thesis by auto
qed
lemma diamond-imp-CR:
    assumes }\diamondr\mathrm{ shows CR r
    using assms by (rule diamond-imp-semi-confluence [THEN semi-confluence-imp-CR])
lemma diamond-imp-CR':
    assumes }\diamonds\mathrm{ and }r\subseteqs\mathrm{ and s}\subseteq\mp@subsup{r}{}{*}\mathrm{ shows CR r
    unfolding CR-iff-meet-subset-join
proof -
    from }\langle\diamonds\rangle\mathrm{ have CR s by (rule diamond-imp-CR)
    then have }\mp@subsup{s}{}{\uparrow}\subseteq\mp@subsup{s}{}{\downarrow}\mathrm{ unfolding CR-iff-meet-subset-join by simp
    from <r\subseteqs\rangle}\mathrm{ have r*}\subseteq\mp@subsup{s}{}{*}\mathrm{ by (rule rtrancl-mono)
    from <s\subseteq r ** have s*}\subseteq(\mp@subsup{r}{}{*}\mp@subsup{)}{}{*}\mathrm{ by (rule rtrancl-mono)
    then have s*}\subseteq\mp@subsup{s}{}{*
    with \langler*}\subseteq\mp@subsup{s}{}{*}\rangle\mathrm{ have }\mp@subsup{r}{}{*}=\mp@subsup{s}{}{*}\mathrm{ by simp
    show }\mp@subsup{r}{}{\uparrow}\subseteq\mp@subsup{r}{}{\downarrow}\mathrm{ unfolding meet-def join-def rtrancl-converse 〈r*}=\mp@subsup{r}{}{*}
        unfolding rtrancl-converse [symmetric] meet-def [symmetric]
        join-def [symmetric] by (rule «s`}\subseteq\mp@subsup{s}{}{\downarrow}\rangle
qed
```

```
lemma \(S N\)-imp-minimal:
    assumes \(S N A\)
    shows \(\forall Q x . x \in Q \longrightarrow(\exists z \in Q . \forall y .(z, y) \in A \longrightarrow y \notin Q)\)
proof (rule ccontr)
    assume \(\neg(\forall Q x . x \in Q \longrightarrow(\exists z \in Q . \forall y .(z, y) \in A \longrightarrow y \notin Q))\)
    then obtain \(Q x\) where \(x \in Q\) and \(\forall z \in Q . \exists y .(z, y) \in A \wedge y \in Q\) by auto
    then have \(\forall z . \exists y . z \in Q \longrightarrow(z, y) \in A \wedge y \in Q\) by auto
    then have \(\exists f . \forall x . x \in Q \longrightarrow(x, f x) \in A \wedge f x \in Q\) by (rule choice)
    then obtain \(f\) where \(a: \forall x . x \in Q \longrightarrow(x, f x) \in A \wedge f x \in Q\) (is \(\forall x\). ?P \(x\) )
by best
    let ? \(S=\lambda i .(f \sim i) x\)
    have ?S \(0=x\) by simp
    have \(\forall i\). \((\) ?S \(i\), ?S \((\) Suc \(i)) \in A \wedge\) ?S \((\) Suc \(i) \in Q\)
    proof
        fix \(i\) show \((? S i, ? S(\) Suc \(i)) \in A \wedge ? S(\) Suc \(i) \in Q\)
            by (induct \(i\) ) (auto simp: \(\langle x \in Q\rangle a\) )
    qed
    with \(\langle ? S 0=x\rangle\) have \(\exists S . S 0=x \wedge\) chain \(A S\) by fast
    with assms show False by auto
qed
lemma SN-on-imp-on-minimal:
    assumes \(S N\)-on \(r\{x\}\)
    shows \(\forall Q . x \in Q \longrightarrow(\exists z \in Q . \forall y .(z, y) \in r \longrightarrow y \notin Q)\)
proof (rule ccontr)
    assume \(\neg(\forall Q . x \in Q \longrightarrow(\exists z \in Q . \forall y .(z, y) \in r \longrightarrow y \notin Q))\)
    then obtain \(Q\) where \(x \in Q\) and \(\forall z \in Q . \exists y .(z, y) \in r \wedge y \in Q\) by auto
    then have \(\forall z . \exists y . z \in Q \longrightarrow(z, y) \in r \wedge y \in Q\) by auto
    then have \(\exists f . \forall x . x \in Q \longrightarrow(x, f x) \in r \wedge f x \in Q\) by (rule choice)
    then obtain \(f\) where \(a: \forall x . x \in Q \longrightarrow(x, f x) \in r \wedge f x \in Q\) (is \(\forall x\). ? \(P x\) )
by best
    let ? \(S=\lambda i .\left(f \sim_{i} i\right) x\)
    have ?S \(0=x\) by simp
    have \(\forall i .(? S i, ? S(\) Suc \(i)) \in r \wedge ? S(\) Suc \(i) \in Q\)
    proof
        fix \(i\) show \((? S i, ? S(\) Suc \(i)) \in r \wedge ? S(\) Suc \(i) \in Q\) by (induct \(i\) ) (auto simp: \(\langle x\)
\(\in Q>a)\)
    qed
    with \(\langle ? S 0=x\rangle\) have \(\exists S . S 0=x \wedge\) chain \(r S\) by fast
    with assms show False by auto
qed
lemma minimal-imp-wf:
    assumes \(\forall Q x . x \in Q \longrightarrow(\exists z \in Q . \forall y .(z, y) \in r \longrightarrow y \notin Q)\)
    shows \(w f\left(r^{-1}\right)\)
proof (rule ccontr)
    assume \(\neg w f\left(r^{-1}\right)\)
    then have \(\exists P .(\forall x .(\forall y .(x, y) \in r \longrightarrow P y) \longrightarrow P x) \wedge(\exists x . \neg P x)\) unfolding
```

```
wf-def by simp
    then obtain Px}\mathrm{ where suc: }\forallx.(\forally.(x,y)\inr\longrightarrowPy)\longrightarrowPx\mathrm{ and }\negP
by auto
    let ?Q = {x.\negP x}
    from }\langle\negPx\rangle\mathrm{ have }x\in\mathrm{ ?Q by simp
    from assms have }\forallx.x\in?Q\longrightarrow(\existsz\in?Q.\forally.(z,y)\inr\longrightarrowy\not\in?Q) by (rul
allE [where x=?Q])
    with }\langlex\in?Q\rangle\mathrm{ obtain z where z ? ?Q and min: }\forally.(z,y)\inr\longrightarrowy\not\in?
by best
    from}\langlez\in?Q\rangle\mathrm{ have }\negPz\mathrm{ by simp
    with suc obtain }y\mathrm{ where (z,y) fr and }\negPy\mathrm{ by best
    then have }y\in?QQ\mathrm{ by simp
    with }\langle(z,y)\inr\rangle\mathrm{ and min show False by simp
qed
lemmas SN-imp-wf =SN-imp-minimal [THEN minimal-imp-wf]
lemma wf-imp-SN:
    assumes wf (A}\mp@subsup{A}{}{-1})\mathrm{ shows SN A
proof - {
    fix a
    let ?P = \lambdaa.\neg(\existsS.S 0 = a^ chain A S)
    from <wf (A-1)〉 have ?P a
    proof induct
        case (less a)
        then have IH: \b. (a,b) \inA\Longrightarrow?P b by auto
        show ?P a
        proof (rule ccontr)
            assume \neg?P a
            then obtain S where S O=a and chain A S by auto
            then have (S0,S1)\inA by auto
            with IH have ?P (S 1) unfolding <S 0 = a〉 by auto
            with <chain A S> show False by auto
                qed
    qed
    then have SN-on A {a} unfolding SN-defs by auto
} then show ?thesis by fast
qed
lemma SN-nat-gt: SN {(a,b :: nat) . a>b}
proof -
    from wf-less have wf ({(x,y).(x:: nat)>y}
by auto
    from wf-imp-SN [OF this] show ?thesis
qed
```

lemma $S N$-iff-wf: $S N A=w f\left(A^{-1}\right)$ by (auto simp: $S N$-imp-wf wf-imp-SN)
lemma $S N$-imp-acyclic: $S N R \Longrightarrow$ acyclic $R$
using wf-acyclic [of $R^{-1}$, unfolded $S N$-iff-wf [symmetric]] by auto
lemma $S N$-induct:
assumes sn: $S N r$ and step: $\bigwedge a .(\bigwedge b .(a, b) \in r \Longrightarrow P b) \Longrightarrow P a$
shows $P a$
using sn unfolding $S N$-iff-wf proof induct
case (less a)
with step show ?case by best
qed
lemmas $S N$-induct-rule $=S N$-induct [consumes 1, case-names $I H$, induct pred: SN]
lemma SN-on-induct [consumes 2, case-names IH, induct pred: SN-on]:
assumes $S N$ : $S N$-on $R A$
and $s \in A$
and $i m p: \wedge t .(\bigwedge u .(t, u) \in R \Longrightarrow P u) \Longrightarrow P t$
shows $P s$
proof -
let $? R=$ restrict-SN $R R$
let $? P=\lambda t$. $S N$-on $R\{t\} \longrightarrow P t$
have $S N$-on $R\{s\} \longrightarrow P s$
proof (rule $S N$-induct [OF SN-restrict-SN-idemp $[$ of $R]$, of ?P])
fix $a$
assume ind: $\wedge b .(a, b) \in ? R \Longrightarrow S N$-on $R\{b\} \longrightarrow P b$
show $S N$-on $R\{a\} \longrightarrow P a$
proof
assume $S N$ : $S N$-on $R\{a\}$
show $P a$ proof (rule imp)
fix $b$
assume $(a, b) \in R$
with $S N$ step-preserves-SN-on [OF this $S N$ ]
show $P b$ using ind $[o f b]$ unfolding restrict-SN-def by auto qed
qed
qed
with $S N$ show $P s$ using $\langle s \in A\rangle$ unfolding $S N$-on-def by blast qed
lemma accp-imp-SN-on:
assumes $\bigwedge x . x \in A \Longrightarrow$ Wellfounded.accp $g x$
shows $S N$-on $\{(y, z), g z y\} A$
proof - \{
fix $x$ assume $x \in A$
from assms [OF this]

```
    have SN-on {(y,z).gz y} {x}
    proof (induct rule: accp.induct)
        case (accI x)
        show ?case
        proof
            fix f
            assume x: f0\in{x} and steps: }\foralli.(fi,f(Suci))\in{a.(\lambda(y,z).gzy)a
            then have g}(f1)x\mathrm{ by auto
            from accI(2)[OF this] steps x show False unfolding SN-on-def by auto
        qed
    qed
    }
    then show ?thesis unfolding SN-on-def by blast
qed
lemma SN-on-imp-accp:
    assumes }SN\mathrm{ -on {(y,z).gz y} A
    shows }\forallx\inA\mathrm{ . Wellfounded.accp g x
proof
    fix }x\mathrm{ assume }x\in
    with assms show Wellfounded.accp g x
    proof (induct rule: SN-on-induct)
        case (IH x)
        show ?case
        proof
            fix }
            assume g y x
            with IH show Wellfounded.accp g y by simp
        qed
    qed
qed
lemma SN-on-conv-accp:
    SN-on {(y,z).gzy}{x}=Wellfounded.accp g x
    using SN-on-imp-accp [of g{x}]
            accp-imp-SN-on [of {x}g]
    by auto
lemma SN-on-conv-acc: SN-on {(y,z). (z,y)\inr}{x}\longleftrightarrowx\in Wellfounded.acc
r
    unfolding SN-on-conv-accp accp-acc-eq ..
lemma acc-imp-SN-on:
    assumes x G Wellfounded.acc r shows SN-on {(y,z). (z,y)\inr} {x}
    using assms unfolding SN-on-conv-acc by simp
lemma SN-on-imp-acc:
    assumes SN-on {(y,z). (z,y)\inr}{x} shows x \in Wellfounded.acc r
    using assms unfolding SN-on-conv-acc by simp
```


### 2.3 Newman's Lemma

```
lemma rtrancl-len-E [elim]:
    assumes (x,y)\in r* obtains n where (x,y)\in r^n
    using rtrancl-imp-UN-relpow [OF assms] by best
lemma relpow-Suc-E2' [elim]:
```



```
proof -
    assume assm: }\y.(x,y)\inA\Longrightarrow(y,z)\in\mp@subsup{A}{}{*}\Longrightarrow\mathrm{ thesis
    from relpow-Suc-E2 [OF assms] obtain y where (x,y) \inA and (y,z)\inA~n
by auto
    then have (y,z)\in\mp@subsup{A}{}{*}\mathrm{ using relpow-imp-rtrancl by auto}
    from assm [OF<<(x,y)\inA> this] show thesis .
qed
lemmas SN-on-induct' [consumes 1, case-names IH] = SN-on-induct [OF - sin-
gletonI]
lemma Newman-local:
    assumes SN-on r X and WCR:WCR-on r {x.SN-on r {x}}
    shows CR-on r X
proof - {
    fix }
    assume }x\in
    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 yz assume (x,y)\in r* and (x,z)\in r*
            from}\langle(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ obtain m where (x,y) f r^`m ..
            from }\langle(x,z)\in\mp@subsup{r}{}{*}\rangle\mathrm{ obtain }n\mathrm{ where (x,z) fr^^n ..
            show (y,z)\in\mp@subsup{r}{}{\downarrow}
            proof (cases n)
                case 0
                from }\langle(x,z)\in\mp@subsup{r}{}{~}~n\rangle\mathrm{ have eq: }x=z\mathrm{ by (simp add: 0)
                from }\langle(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ show ?thesis unfolding eq ..
            next
                case (Suc n')
                from}\langle(x,z)\in\mp@code{`^n\rangle [unfolded Suc] obtain t where (x,t)\inr and (t,z)
\in r* ..
                show ?thesis
                proof (cases m)
                    case 0
                    from }\langle(x,y)\inr~m>> have eq: x=y by (simp add: 0
                    from}\langle(x,z)\in\mp@subsup{r}{}{*}\rangle\mathrm{ show ?thesis unfolding eq ..
                next
                    case (Suc m')
                    from}\langle(x,y)\inr~m>[unfolded Suc] obtain s where (x,s)\inr and (s
```

$y) \in r^{*} .$.
from $W C R I H(2)$ have $W C R$-on $r\{x\}$ unfolding $W C R$-on-def by auto with $\langle(x, s) \in r\rangle$ and $\langle(x, t) \in r\rangle$ have $(s, t) \in r^{\downarrow}$ by auto then obtain $u$ where $(s, u) \in r^{*}$ and $(t, u) \in r^{*}$.. from $\langle(x, s) \in r\rangle I H(2)$ have $S N$-on $r\{s\}$ by (rule step-preserves-SN-on) from $I H(1)[O F\langle(x, s) \in r\rangle$ this $]$ have CR-on $r\{s\}$.
from this and $\left\langle(s, u) \in r^{*}\right\rangle$ and $\left\langle(s, y) \in r^{*}\right\rangle$ have $(u, y) \in r^{\downarrow}$ by auto then obtain $v$ where $(u, v) \in r^{*}$ and $(y, v) \in r^{*}$.. from $\langle(x, t) \in r\rangle I H(2)$ have $S N$-on $r\{t\}$ by (rule step-preserves-SN-on) from $I H(1)[O F\langle(x, t) \in r\rangle$ this $]$ have CR-on $r\{t\}$.
moreover from $\left\langle(t, u) \in r^{*}\right\rangle$ and $\left\langle(u, v) \in r^{*}\right\rangle$ have $(t, v) \in r^{*}$ by auto ultimately have $(z, v) \in r^{\downarrow}$ using $\left\langle(t, z) \in r^{*}\right\rangle$ by auto
then obtain $w$ where $(z, w) \in r^{*}$ and $(v, w) \in r^{*}$..
from $\left\langle(y, v) \in r^{*}\right\rangle$ and $\left\langle(v, w) \in r^{*}\right\rangle$ have $(y, w) \in r^{*}$ by auto with $\left\langle(z, w) \in r^{*}\right\rangle$ show ?thesis by auto
qed
qed
qed
qed
\}
then show ?thesis unfolding $C R$-on-def by blast
qed
lemma Newman: SN $r \Longrightarrow W C R r \Longrightarrow C R r$
using Newman-local [of r UNIV]
unfolding WCR-on-def by auto
lemma Image-SN-on:
assumes $S N$-on $r(r$ " $A$ )
shows $S N$-on r $A$
proof
fix $f$
assume $f 0 \in A$ and chain: chain $r f$
then have $f$ (Suc 0) $\in r$ " $A$ by auto
with assms have $S N$-on $r\{f$ (Suc 0) \} by (auto simp add: $\langle f 0 \in A\rangle S N$-defs)
moreover have $\neg S N$-on $r\{f($ Suc 0$)\}$
proof -
have $f($ Suc 0) $\in\{f($ Suc 0) $\}$ by simp
moreover from chain have chain $r(f \circ$ Suc) by auto
ultimately show ?thesis by auto
qed
ultimately show False by simp
qed
lemma $S N$-on-Image-conv: $S N$-on $r(r$ " $A)=S N$-on $r A$
using $S N$-on-Image and Image-SN-on by blast

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

```
lemma step-reflects-SN-on:
    assumes (\b. (a,b) \inr\LongrightarrowSN-on r {b})
    shows SN-on r {a}
    using assms and Image-SN-on [of r {a}] by (auto simp: SN-defs)
lemma SN-on-all-reducts-SN-on-conv:
    SN-on r {a} = (\forallb. (a,b) \inr\longrightarrowSN-on r {b})
    using SN-on-Image-conv [of r {a}] by (auto simp:SN-defs)
lemma SN-imp-SN-trancl: SN R\LongrightarrowSN ( }\mp@subsup{R}{}{+
    unfolding SN-iff-wf by (rule wf-converse-trancl)
lemma SN-trancl-imp-SN:
    assumes }SN(\mp@subsup{R}{}{+})\mathrm{ shows }SN
    using assms by (rule SN-on-trancl-imp-SN-on)
lemma SN-trancl-SN-conv: SN ( }\mp@subsup{R}{}{+}\mathrm{ ) = SN R
    using SN-trancl-imp-SN [of R] SN-imp-SN-trancl [of R] by blast
lemma SN-inv-image: SN R\LongrightarrowSN (inv-image R f) unfolding SN-iff-wf by
simp
lemma SN-subset: SN R\Longrightarrow 盾\subseteqR\LongrightarrowSN R'unfolding SN-defs by blast
lemma SN-pow-imp-SN:
    assumes SN (A^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: ^i. (Si,S (i+(Suc n))) \in A^^Suc n .
    let ?T = \lambdai.S (i*(Suc n))
    have chain (A~Suc n) ?T
    proof
        fix i show (?T i,?T (Suc i)) \in A^Suc n unfolding mult-Suc
        using step [of i*Suc n] by (simp only: add.commute)
    qed
    then have }\negSN(A~Suc n) unfolding SN-defs by fas
    with assms show False by simp
qed
lemma pow-Suc-subset-trancl: R}\mp@subsup{R}{}{~}(\mathrm{ Suc n) }\subseteq\mp@subsup{R}{}{+
    using trancl-power [of - R] by blast
lemma SN-imp-SN-pow:
    assumes SN R shows SN ( R^~Suc n)
    using SN-subset [where R=R ', OF SN-imp-SN-trancl [OF assms] pow-Suc-subset-trancl]
by simp
```

```
lemma \(S N\)-pow: \(S N R \longleftrightarrow S N(R \leadsto\) Suc \(n)\)
    by (rule iffI, rule \(S N\)-imp-SN-pow, assumption, rule \(S N-p o w-i m p-S N\), assump-
tion)
lemma \(S N\)-on-trancl:
    assumes \(S N\)-on \(r A\) shows \(S N\)-on \(\left(r^{+}\right) A\)
using assms
proof (rule contrapos-pp)
    let \(? r=\) restrict-SN r r
    assume \(\neg S N\)-on \(\left(r^{+}\right) A\)
    then obtain \(f\) where \(f 0 \in A\) and chain: chain \(\left(r^{+}\right) f\) by auto
    have \(S N\) ?r by (rule \(S N\)-restrict-SN-idemp)
    then have \(S N\left(? r^{+}\right)\)by (rule \(S N\)-imp-SN-trancl)
    have \(\forall i .(f 0, f i) \in r^{*}\)
    proof
        fix \(i\) show \((f 0, f i) \in r^{*}\)
        proof (induct i)
            case 0 show ?case ..
        next
                case (Suc i)
                from chain have \((f i, f(S u c i)) \in r^{+}\)..
                with Suc show? case by auto
        qed
    qed
    with assms have \(\forall i\). SN-on \(r\{f i\}\)
        using steps-preserve-SN-on [of f0-r]
        and \(\langle f 0 \in A\rangle\)
        and \(S N\)-on-subset2 \([\) of \(\{f 0\} A]\) by auto
    with chain have chain \(\left(? r^{+}\right) f\)
        unfolding restrict-SN-trancl-simp
        unfolding restrict-SN-def by auto
    then have \(\neg S N\)-on \(\left(? r^{+}\right)\left\{f_{0} 0\right\}\) by auto
    with \(\left\langle S N\left(? r^{+}\right)\right\rangle\)have False by (simp add: \(S N-d e f s\) )
    then show \(\neg S N\)-on \(r A\) by simp
qed
lemma \(S N\)-on-trancl-SN-on-conv: \(S N\)-on \(\left(R^{+}\right) T=S N\)-on \(R T\)
    using \(S N\)-on-trancl-imp-SN-on \([\) of \(R] S N\)-on-trancl \([\) of \(R]\) by blast
        Restrict an ARS to elements of a given set.
definition restrict :: 'a rel \(\Rightarrow{ }^{\prime}\) 'a set \(\Rightarrow\) 'a rel where
    restrict \(r S=\{(x, y) . x \in S \wedge y \in S \wedge(x, y) \in r\}\)
lemma \(S N\)-on-restrict:
    assumes \(S N\)-on r \(A\)
    shows \(S N\)-on (restrict \(r S\) ) \(A\) (is \(S N\)-on ?r \(A\) )
proof (rule ccontr)
```

```
    assume }\negSN-on ?r 
    then have }\exists\textrm{f}.f0\inA\wedge\mathrm{ chain ?r f by auto
    then have \existsf.f |\inA ^ chain rf unfolding restrict-def by auto
    with 〈SN-on r A show False by auto
qed
lemma restrict-rtrancl: (restrict rS S)}\subseteq\mp@subsup{r}{}{*}(\mathbf{is}?\mp@subsup{r}{}{*}\subseteq\mp@subsup{r}{}{*}
proof - {
    fix }xy\mathrm{ assume (x,y) &?r* then have (x,y) & r* unfolding restrict-def by
induct auto
} then show ?thesis by auto
qed
lemma rtrancl-Image-step:
    assumes }a\in\mp@subsup{r}{}{*}\mathrm{ " A
        and (a,b) \in r*
    shows b\inr* " A
proof -
    from assms(1) obtain c where c\inA and (c,a)\in r* by auto
    with assms have (c,b)\in\mp@subsup{r}{}{*}\mathrm{ by auto}
    with }\langlec\inA\rangle\mathrm{ 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 }\forallx.SN\mathrm{ -on ?r {x}
    proof
        fix y have y &?S \vee y\in?S by simp
        then show SN-on ?r {y}
        proof
            assume y &?S then show ?thesis unfolding restrict-def by auto
        next
            assume y \in?S
            then have }y\in\mp@subsup{r}{}{*}\mathrm{ " 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 }xy\mathrm{ assume (x,y) & r* and x & ?S and y f?S
        then obtain n where (x,y)\inr~}n\mathrm{ and }x\in?S\mathrm{ and }y\in?
            using rtrancl-imp-UN-relpow by best
        then have (x,y)\in?r*
        proof (induct n arbitrary: x y)
```

```
        case 0 then show ?case by simp
    next
        case (Suc n)
        from < (x,y) \in r^^Suc n> obtain \mp@subsup{x}{}{\prime}}\mathrm{ where (x, x') &r and ( }\mp@subsup{x}{}{\prime},y)\in\mp@subsup{r}{}{~}
            using relpow-Suc-D2 by best
        then have ( }x,\mp@subsup{x}{}{\prime})\in\mp@subsup{r}{}{*}\mathrm{ by simp
        with <x \in?S` have }\mp@subsup{x}{}{\prime}\in\mathrm{ ?S by (rule rtrancl-Image-step)
        with Suc and }\langle(\mp@subsup{x}{}{\prime},y)\in\mp@subsup{r}{}{~}n\rangle\mathrm{ have ( (x', y) &? ? * by simp
        from }\langle(x,\mp@subsup{x}{}{\prime})\inr\rangle\mathrm{ and }\langlex\in??S\rangle\mathrm{ and }\langle\mp@subsup{x}{}{\prime}\in\mathrm{ ?S> have (x, x') < ?r
            unfolding restrict-def by simp
        with }\langle(\mp@subsup{x}{}{\prime},y)\in?\mp@subsup{r}{}{*}\rangle\mathrm{ show ?case by simp
    qed
}
then have a:\forallx y. (x,y)\in\mp@subsup{r}{}{*}\wedgex\in?S\wedge y\in?S\longrightarrow(x,y)\in?r* by simp
{
    fix \mp@subsup{x}{}{\prime}yz\mathrm{ assume ( }\mp@subsup{x}{}{\prime},y)\in?r and ( (x',z)\in?r
```



```
        unfolding restrict-def by auto
    with \langleWCR r> have (y,z)\in r}\downarrow\mathrm{ by auto
    then obtain u}\mathrm{ where (y,u) & r* and (z,u) & r* by auto
    from }\langle\mp@subsup{x}{}{\prime}\in?S\rangle\mathrm{ obtain }x\mathrm{ where }x\inA\mathrm{ and (x, x) ) 䄪* by auto
    from }\langle(\mp@subsup{x}{}{\prime},y)\inr\rangle\mathrm{ have ( }\mp@subsup{x}{}{\prime},y)\in\mp@subsup{r}{}{*}\mathrm{ by auto
    with }\langle(y,u)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have ( }\mp@subsup{x}{}{\prime},u)\in\mp@subsup{r}{}{*}\mathrm{ by auto
    with }\langle(x,\mp@subsup{x}{}{\prime})\in\mp@subsup{r}{}{*}\rangle\mathrm{ have (x,u) G r* by simp
    then have }u\in?S\mathrm{ using }\langlex\inA\rangle\mathrm{ by auto
    from }\langley\in?S\rangle\mathrm{ and }\langleu\in?S\rangle\mathrm{ and }\langle(y,u)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have }(y,u)\in?\mp@subsup{r}{}{*}\mathrm{ using a by
auto
    from }\langlez\in?S\rangle\mathrm{ and }\langleu\in?S\rangle\mathrm{ and }\langle(z,u)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have (z,u) Є?r* using a by
auto
    with}\langle(y,u)\in?\mp@subsup{r}{}{*}\rangle\mathrm{ have }(y,z)\in?\mp@subsup{r}{}{\downarrow}\mathrm{ by auto
    }
    then have WCR ?r by auto
    have CR ?r using Newman [OF\langleSN ?r\rangle\langleWCR ?r\rangle] by simp
    {
        fix x y z assume }x\inA\mathrm{ and (x,y) fr* and (x,z) fr*
        then have }y\in?S\mathrm{ and z}\in?S\mathrm{ by auto
        have }x\in?S\mathrm{ using }\langlex\inA\rangle\mathrm{ by auto
        from }a\mathrm{ and }\langle(x,y)\in\mp@subsup{r}{}{*}\rangle\mathrm{ and }\langlex\in?S\rangle\mathrm{ and }\langley\in?S\rangle\mathrm{ have }(x,y)\in?\mp@subsup{r}{}{*}\mathrm{ by
simp
    from }a\mathrm{ and }\langle(x,z)\in\mp@subsup{r}{}{*}\rangle\mathrm{ and }\langlex\in?S\rangle\mathrm{ and }\langlez\in?S\rangle\mathrm{ have }(x,z)\in?\mp@subsup{r}{}{*}\mathrm{ by
simp
            with 〈CR ?r\rangle and }\langle(x,y)\in??\mp@subsup{r}{}{*}\rangle\mathrm{ have ( }y,z)\in?\mp@subsup{?}{}{\downarrow}\downarrow by aut
            then obtain u where (y,u)\in? ?r* and (z,u)\in? ?r* by best
            then have (y,u)\in\mp@subsup{r}{}{*}\mathrm{ and (z,u) & r* using restrict-rtrancl by auto}
            then have (y,z)\in r}\downarrow\mathrm{ by auto
    }
    then show ?thesis by auto
qed
```

```
lemma SN-on-Image-normalizable:
    assumes SN-on r A
    shows }\foralla\inA.\existsb.b\inr!" "
proof
    fix a assume a: a}\in
    show }\existsb.b\inr! " A
    proof (rule ccontr)
        assume }\neg(\existsb.b\in\mp@subsup{r}{}{!}"A
    then have A: \forallb. (a,b)\in r* \longrightarrowb \ NFr using a by auto
    then have a &NFr by auto
    let ?Q }={c.(a,c)\in\mp@subsup{r}{}{*}\wedgec\not\inNFr
    have a\in??Q using <a\not\inNF r> by simp
    have }\forallc\in?Q.\existsb.(c,b)\inr\wedgeb\in?
    proof
        fix c
        assume c \in ?Q
        then have (a,c)\in\mp@subsup{r}{}{*}\mathrm{ and c&NFr by auto}
        then obtain d}\mathrm{ where (c,d) fr by auto
        with }\langle(a,c)\in\mp@subsup{r}{}{*}\rangle\mathrm{ have (a,d) fr* by simp
        with A have d}\not=NFr by sim
        with }\langle(c,d)\inr\rangle\mathrm{ and }\langle(a,c)\in\mp@subsup{r}{}{*}
            show }\existsb.(c,b)\inr\wedgeb\in?Q by aut
    qed
    with }\langlea\in?Q\rangle\mathrm{ have }a\in?Q\wedge(\forallc\in?Q.\existsb.(c,b)\inr\wedgeb\in?Q) by aut
    then have }\existsQ.a\inQ\wedge(\forallc\inQ.\existsb.(c,b)\inr\wedgeb\inQ) by (rule exI [of -
?Q])
    then have }\neg(\forallQ.a\inQ\longrightarrow(\existsc\inQ.\forallb.(c,b)\inr\longrightarrowb\not\inQ))\mathrm{ by simp
    with SN-on-imp-on-minimal [of r a] have }\neg\mathrm{ SN-on r {a} by blast
    with assms and }\langlea\inA\rangle\mathrm{ and SN-on-subset2 [of {a} A r] show False by simp
    qed
qed
lemma \(S N\)-on-imp-normalizability:
assumes \(S N\)-on \(r\{a\}\) shows \(\exists b .(a, b) \in r^{!}\)
using \(S N\)-on-Image-normalizable [OF assms] by auto
```


### 2.4 Commutation

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

$$
\text { commute } r s \longleftrightarrow\left(\left(r^{-1}\right)^{*} O s^{*}\right) \subseteq\left(s^{*} O\left(r^{-1}\right)^{*}\right)
$$

lemma CR-iff-self-commute: $C R \quad r=$ commute $r$ r unfolding commute-def CR-iff-meet-subset-join meet-def join-def by $\operatorname{simp}$
lemma rtrancl-imp-rtrancl-UN:
assumes $(x, y) \in r^{*}$ and $r \in I$
shows $(x, y) \in(\bigcup r \in I . r)^{*}\left(\right.$ is $\left.(x, y) \in ? r^{*}\right)$

```
using assms proof induct
    case base then show ?case by simp
next
    case (step y z)
    then have (x,y)\in?r* by simp
    from }\langle(y,z)\inr\rangle\mathrm{ and }\langler\inI\rangle\mathrm{ have (y,z) €? ? r* by auto
    with }\langle(x,y)\in??\mp@subsup{r}{}{*}\rangle\mathrm{ show ?case by auto
qed
definition quasi-commute :: 'a rel }=>\mathrm{ 'a rel }=>\mathrm{ bool where
    quasi-commute rs \longleftrightarrow(sOr)\subseteqrO(r\cups)*
lemma rtrancl-union-subset-rtrancl-union-trancl: (r\cups+)*}=(r\cups\mp@subsup{)}{}{*
proof
    show (r\cups+}\mp@subsup{)}{}{*}\subseteq(r\cups\mp@subsup{)}{}{*
    proof (rule subrelI)
        fix x y assume (x,y)\in(r\cups s}\mp@subsup{)}{}{*
        then show (x,y)\in(r\cups)*
        proof (induct)
            case base then show ?case by auto
        next
            case (step y z)
            then have (y,z)\inr\vee (y,z)\in s+}\mathrm{ by auto
            then have (y,z)\in(r\cups)*
            proof
                assume (y,z)\inr then show ?thesis by auto
            next
                assume (y,z)\in s+
                then have (y,z)\in s* by auto
                then have (y,z)\in\mp@subsup{r}{}{*}\cup\mp@subsup{s}{}{*}\mathrm{ by auto}
                then show ?thesis using rtrancl-Un-subset by auto
            qed
            with}\langle(x,y)\in(r\cups\mp@subsup{)}{}{*}\rangle\mathrm{ show ?case by simp
        qed
    qed
next
    show (r\cups)*}\subseteq(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*
    proof (rule subrelI)
        fix x y assume (x,y)\in(r\cups)*
        then show (x,y)\in(r\cups+)*
        proof (induct)
            case base then show ?case by auto
        next
            case (step y z)
            then have (y,z)\in(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}\mathrm{ by auto}
            with}\langle(x,y)\in(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}\rangle\mathrm{ show ?case by auto
        qed
    qed
qed
```

```
lemma qc-imp-qc-trancl:
    assumes quasi-commute r s shows quasi-commute r ( }\mp@subsup{s}{}{+}\mathrm{ )
unfolding quasi-commute-def
proof (rule subrelI)
    fix xz assume (x,z)\in s+}O
    then obtain }y\mathrm{ where (x,y) & s+
    then show (x,z)\inrO(r\cups+)*
    proof (induct arbitrary: z)
        case (base y)
        then have }(x,z)\in(sOr)\mathrm{ by auto
        with assms have (x,z)\inrO(r\cups\mp@subsup{)}{}{*}\mathrm{ unfolding quasi-commute-def by auto}
        then show ?case using rtrancl-union-subset-rtrancl-union-trancl by auto
    next
        case (step a b)
        then have (a,z)\in(sOr) by auto
        with assms have (a,z)\inrO(r\cups\mp@subsup{)}{}{*}\mathrm{ unfolding quasi-commute-def by auto}
        then obtain u where (a,u)\inr and (u,z)\in(r\cups)* by best
        then have (u,z)\in(r\cups+}\mp@subsup{)}{}{*}\mathrm{ using rtrancl-union-subset-rtrancl-union-trancl
by auto
    from}\langle(a,u)\inr\rangle and step have (x,u)\inrO(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}\mathrm{ by auto
    then obtain v where (x,v)\inr and (v,u)\in(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}\mathrm{ by best}
    with}\langle(u,z)\in(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}\rangle\mathrm{ have }(v,z)\in(r\cup\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}\mathrm{ by auto
    with }\langle(x,v)\inr\rangle\mathrm{ show ?case by auto
    qed
qed
lemma steps-reflect-SN-on:
    assumes }\negSN\mathrm{ -on r {b} and (a,b) fr*
    shows }\negSN\mathrm{ -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 rf
    shows \negSN-on r {fi}
proof -
    let ?f = \lambdaj. f(i+j)
    have ?f 0 \in{f i} by simp
    moreover have chain r ?f using assms by auto
    ultimately have ?f 0}\in{fi}\wedge chain r ?f by blas
    then have \existsg.g 0\in{fi}^chain rg 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\cups)
proof -
```

```
have quasi-commute r (s+) by (rule qc-imp-qc-trancl [OF <quasi-commute r s`])
let ?B ={a.\negSN-on (r\cups){a}}
{
    assume }\negSN(r\cups
    then obtain a where a ? ?B unfolding SN-defs by fast
    from \langleSN r\rangle have }\forallQx.x\inQ\longrightarrow(\existsz\inQ.\forally.(z,y)\inr\longrightarrowy\not\inQ
        by (rule SN-imp-minimal)
    then have }\forallx.x\in?B\longrightarrow(\existsz\in?B.\forally.(z,y)\inr\longrightarrowy\not\in?B)\mathrm{ by (rule spec
[where x = ? B])
    with }\langlea\in?B\rangle\mathrm{ obtain }b\mathrm{ where }b\in?B\mathrm{ and min: }\forally.(b,y)\inr\longrightarrowy\not\in?
by auto
    from}\langleb\in?B\rangle\mathrm{ obtain S where S 0 = b and
        chain: chain (r\cups)S unfolding SN-on-def by auto
    let ?S = \lambdai. S(Suc i)
    have ?S 0 = S 1 by simp
    from chain have chain (r\cups) ?S by auto
    with <?S 0 = S 1` have \negSN-on (r\cups) {S 1} unfolding SN-on-def by auto
    from }\langleS0=b\rangle\mathrm{ and chain have (b,S 1) GrUs by auto
    with min and «\neg SN-on (r\cups){S1}` have (b,S 1) \ins by auto
    let ?i=LEAST i. (Si,S(Suc i))}\not=
    {
        assume chain s S
        with }\langleS0=b\rangle\mathrm{ have }\negSN\mathrm{ -on s {b} unfolding SN-on-def by auto
        with \langleSN s\rangle have False unfolding SN-defs by auto
    }
    then have ex: \existsi.(Si,S(Suc i)) \not\ins by auto
    then have (S ?i,S(Suc ?i)) &s by (rule LeastI-ex)
    with chain have (S ?i,S(Suc ?i)) \inr by auto
    have ini: }\foralli<?\mathrm{ ?. (S i,S(Suc i)) }\ins\mathrm{ using not-less-Least by auto
    {
        fix i assume i<?i then have (b,S(Suc i))\in s+
        proof (induct i)
            case 0 then show ?case using <(b,S 1)\ins\rangle and \langleS 0 = b\rangle by auto
        next
            case (Suc k)
        then have (b,S(Suck))\ins\mp@subsup{s}{}{+}\mathrm{ and Suc k< ?i by auto}
        with <\foralli<?i. (S i,S(Suc i)) \in s> have (S(Suc k), S(Suc(Suc k))) \ins by
fast
        with «(b,S(Suc k)) \in s+` show ?case by auto
    qed
    }
    then have pref: }\foralli<?i.(b,S(Suc i))\in\mp@subsup{s}{}{+}\mathrm{ by auto
    from}\langle(b,S 1)\ins\rangle\mathrm{ and }\langleS0=b\rangle\mathrm{ have (S 0, S(Suc 0)) G s by auto
    {
        assume ? i = 0
        from ex have (S?i, S(Suc ?i)) }\not=s\mathrm{ by (rule LeastI-ex)
        with «(S 0, S(Suc 0)) \in s` have False unfolding <? i = 0` by simp
    }
    then have 0 < ?i by auto
```

```
    then obtain j where ?i = Suc j unfolding grO-conv-Suc by best
    with ini have (S(?i-Suc 0),S(Suc(?i-Suc 0))) \ins by auto
    with 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}\langle(S?i,S(Suc ?i))\inr\rangle\mathrm{ have (b,S(Suc ?i)) }\in(\mp@subsup{s}{}{+}Or)\mathrm{ by auto
    with <quasi-commute r (s+)> have (b,S(Suc ?i)) \inrO(r\cups+)*
        unfolding quasi-commute-def by auto
    then obtain c where (b,c)\inr and (c,S(Suc ?i)) \in(r\cups+)* by best
    from }\langle(b,c)\inr\rangle\mathrm{ have (b,c) G(rUs)* by auto
    from chain-imp-not-SN-on [of Sr\cups]
        and chain have }\negSN-on (r\cups){S(Suc ?i)} by aut
    from <(c,S(Suc ?i)) \in(r\cups\mp@subsup{s}{}{+}\mp@subsup{)}{}{*}>\mathrm{ have (c,S(Suc ?i)) }\in(r\cups\mp@subsup{)}{}{*}
        unfolding rtrancl-union-subset-rtrancl-union-trancl by auto
    with steps-reflect-SN-on [of r \cups]
    and «\negSN-on (r\cups){S(Suc ?i)}` have }\neg\mathrm{ SN-on (r Us) {c} by auto
    then have c\in??B by simp
    with }\langle(b,c)\inr\rangle\mathrm{ and min have False by auto
    }
    then show ?thesis by auto
qed
```


### 2.5 Strong Normalization

lemma non-strict-into-strict:
assumes compat: $N S O S \subseteq S$ and steps: $(s, t) \in\left(N S^{*}\right) O S$
shows $(s, t) \in S$
using steps proof
fix $x u z$
assume $(s, t)=(x, z)$ and $(x, u) \in N S^{*}$ and $(u, z) \in S$
then have $(s, u) \in N S^{*}$ 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 abc)
with compat show ?case by auto
qed
qed
lemma comp-trancl:
assumes $R O S \subseteq S$ shows $R O S^{+} \subseteq S^{+}$
proof (rule subrelI)
fix $w z$ assume $(w, z) \in R O S^{+}$
then obtain $x$ where $R$-step: $(w, x) \in R$ and $S$-seq: $(x, z) \in S^{+}$by best
from tranclD [OF S-seq] obtain $y$ where $S$-step: $(x, y) \in S$ and $S$-seq': $(y, z)$
$\in S^{*}$ by auto
from $R$-step and $S$-step have $(w, y) \in R O S$ by auto
with assms have $(w, y) \in S$ by auto

```
    with }S\mathrm{ -seq' show (w,z) G S' by simp
qed
lemma comp-rtrancl-trancl:
    assumes comp: R OS\subseteqS
        and seq: (s,t)\in(R\cupS)*}O
    shows (s,t)\inS S
using seq proof
    fix x uz
```



```
    then have (s,u)\in(R\cupS\mp@subsup{)}{}{*}\mathrm{ and (u,t) G 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)\inR\cupS by simp
        then show ?case
        proof
            assume (b,c)\inS
            with rtrancl-into-rtrancl
            have (b,t)\in\mp@subsup{S}{}{+}}\mathrm{ by simp
            with rtrancl-into-rtrancl show ?thesis by simp
    next
            assume (b,c)\inR
            with comp-trancl [OF comp] rtrancl-into-rtrancl
            show ?thesis by auto
        qed
    qed
qed
lemma trancl-union-right: }\mp@subsup{r}{}{+}\subseteq(s\cupr\mp@subsup{)}{}{+
proof (rule subrelI)
    fix x y assume (x,y)\in r' then show (x,y)\in(s\cupr)+
    proof (induct)
    case base then show ?case by auto
    next
        case (step a b)
        then have (a,b) \in(s\cupr\mp@subsup{)}{}{+}}\mathrm{ by auto
        with}\langle(x,a)\in(s\cupr\mp@subsup{)}{}{+}\rangle\mathrm{ show ?case by auto
    qed
qed
lemma restrict-SN-subset: restrict-SN R S\subseteqR
proof (rule subrelI)
    fix a b assume (a,b) \in restrict-SN R S then show (a,b) \inR unfolding
restrict-SN-def by simp
qed
```

```
lemma chain-Un-SN-on-imp-first-step:
    assumes chain (R\cupS)t and SN-on S{t 0}
    shows \existsi.(ti,t(Suc i)) \inR\wedge(\forallj<i.(t j,t (Suc j)) \inS\wedge (t j,t (Suc j))\not\in
R)
proof -
    from 〈SN-on S {t 0}> obtain i where (t i, t(Suc i)) & S by blast
    with assms have (t i,t (Suc i)) \inR (is ?P i) by auto
    let ?i = Least ?P
    from <?P i` have ?P ?i by (rule LeastI)
    have }\forallj<??.(tj,t(Sucj))\not\inR\mathrm{ using not-less-Least by auto
    moreover with assms have }\forallj<?i.(tj,t(Suc j))\inS by bes
    ultimately have }\forallj<??.(tj,t(Suc j))\inS\wedge(tj,t(Suc j))\not\inR\mathrm{ by best
    with〈?P ?i〉 show ?thesis by best
qed
lemma first-step:
    assumes C:C=A\cupB and steps: (x,y)\inC * and Bstep: (y,z)\inB
    shows }\existsy.(x,y)\in\mp@subsup{A}{}{*}O
    using steps
proof (induct rule: converse-rtrancl-induct)
    case base
    show ?case using Bstep by auto
next
    case (step u x)
    from step(1)[unfolded C]
    show ?case
    proof
        assume (u,x)\inB
        then show ?thesis by auto
    next
        assume ux: (u,x)\inA
        from step(3) obtain y where (x,y)\in A* O B by auto
        then obtain z where (x,z)\in\mp@subsup{A}{}{*}\mathrm{ and step: (z,y) & B by auto}\0
        with ux have (u,z)\in\mp@subsup{A}{}{*}\mathrm{ by auto}
        with step have (u,y)\in\mp@subsup{A}{}{*}OB\mathrm{ by auto}
        then show ?thesis by auto
    qed
qed
lemma first-step-O:
    assumes C:C=A\cupB and steps: }(x,y)\in\mp@subsup{C}{}{*}O
    shows }\existsy.(x,y)\in\mp@subsup{A}{}{*}O
proof -
    from steps obtain z where (x,z)\in\mp@subsup{C}{}{*}\mathrm{ and (z,y) & B by auto}
    from first-step [OF C this] show ?thesis.
qed
lemma firstStep:
    assumes LSR:L=S\cupR and xyL: (x,y)\in L*
```

```
    shows }(x,y)\in\mp@subsup{R}{}{*}\vee(x,y)\in\mp@subsup{R}{}{*}OSO\mp@subsup{L}{}{*
proof (cases (x,y)\in R*)
    case True
    then show?thesis by simp
next
    case False
    let ?SR=S\cupR
    from xyL and LSR have ( }x,y)\in?S\mp@subsup{R}{}{*}\mathrm{ by simp
    from this and False have (x,y)\in R* OSO ?SR*
    proof (induct rule: rtrancl-induct)
    case base then show ?case by simp
    next
    case (step y z)
    then show ?case
    proof (cases (x,y)\in R*)
        case False with step have (x,y)\in R* OS O?SR* by simp
        from this obtain u where xu: (x,u)\in R* OS and uy: (u,y)\in?SR* by
force
        from }\langle(y,z)\in?SR\rangle have (y,z)\in?SR* by aut
        with uy have (u,z)\in?SR* by (rule rtrancl-trans)
        with }xu\mathrm{ show ?thesis by auto
    next
        case True
        have (y,z)\inS
        proof (rule ccontr)
            assume (y,z)\not\inS with «(y,z)\in?SR` have (y,z)\inR by auto
            with True have (x,z)\in R* by auto
            with <(x,z)\not\in R*> show False ..
        qed
        with True show ?thesis by auto
    qed
    qed
    with LSR show ?thesis by simp
qed
lemma non-strict-ending:
    assumes chain: chain (R\cupS)t
        and comp: R OS\subseteqS
        and SN:SN-on S {t 0}
    shows }\existsj.\foralli\geqj.(ti,t(Suci))\inR-
proof (rule ccontr)
    assume ᄀ?thesis
    with chain have }\foralli.\existsj.j\geqi\wedge(tj,t(Suc j))\inS\mathrm{ by blast
    from choice [OF this] obtain f}\mathrm{ where S-steps: }\foralli.i\leqfi\wedge(t(fi),t(Suc(
i))) \inS ..
    let ?t = \lambdai.t (((Suc\circf) ~ i) 0)
    have S-chain: \foralli.(t i,t (Suc (f i))) \in S+
    proof
```

fix $i$
from $S$-steps have leq: $i \leq f i$ and step: $(t(f i), t(S u c(f i))) \in S$ by auto from chain-imp-rtrancl $\left[O F\right.$ chain leq] have $(t i, t(f i)) \in(R \cup S)^{*}$. with step have $(t i, t(S u c(f i))) \in(R \cup S)^{*} O S$ by auto from comp-rtrancl-trancl [OF comp this] show ( $t i, t(S u c(f i))) \in S^{+}$. qed
then have chain $\left(S^{+}\right)$?tby simp
moreover have $S N$-on $\left(S^{+}\right)\{$?t 0$\}$ using $S N$-on-trancl $[O F S N]$ by simp ultimately show False unfolding $S N$-defs by best
qed
lemma $S N$-on-subset1:
assumes $S N$-on $r A$ and $s \subseteq r$
shows $S N$-on s $A$
using assms unfolding $S N$-defs by blast
lemmas $S N$-on-mono $=S N$-on-subset1
lemma rtrancl-fun-conv:
$\left((s, t) \in R^{*}\right)=(\exists f n . f 0=s \wedge f n=t \wedge(\forall i<n .(f i, f($ Suc $i)) \in R))$ unfolding rtrancl-is-UN-relpow using relpow-fun-conv [where $R=R]$ by auto
lemma compat-tr-compat:
assumes $N S O S \subseteq S$ shows $N S^{*} O S \subseteq S$
using non-strict-into-strict [where $S=S$ and $N S=N S$ ] assms by blast
lemma right-comp-S [simp]:
assumes $(x, y) \in S O\left(S O S^{*} O N S^{*} \cup N S^{*}\right)$
shows $(x, y) \in\left(S O S^{*} O N S^{*}\right)$
proof-
from assms have $(x, y) \in\left(S O S O S^{*} O N S^{*}\right) \cup\left(S O N S^{*}\right)$ by auto then have $x y:(x, y) \in\left(S O\left(S O S^{*}\right) O N S^{*}\right) \cup\left(S O N S^{*}\right)$ by auto have $S O S^{*} \subseteq S^{*}$ by auto with $x y$ have $(x, y) \in\left(S O S^{*} O N S^{*}\right) \cup\left(S O N S^{*}\right)$ by auto then show $(x, y) \in\left(S O S^{*} O N S^{*}\right)$ by auto
qed
lemma compatible-SN:
assumes $S N$ : $S N S$
and compat: $N S O S \subseteq S$
shows $S N\left(S O S^{*} O N S^{*}\right)($ is $S N ? A)$
proof
fix $F$ assume chain: chain ?A $F$
from compat compat-tr-compat have tr-compat: $N S^{*} O S \subseteq S$ by blast
have $\forall i .\left(\exists y z .(F i, y) \in S \wedge(y, z) \in S^{*} \wedge(z, F(S u c i)) \in N S^{*}\right)$
proof
fix $i$
from chain have $(F i, F(S u c i)) \in\left(S O S^{*} O N S^{*}\right)$ by auto
then show $\exists y z .(F i, y) \in S \wedge(y, z) \in S^{*} \wedge(z, F(S u c i)) \in N S^{*}$ unfolding relcomp-def using mem-Collect-eq by auto
qed
then have $\exists f .\left(\forall i .\left(\exists z .(F i, f i) \in S \wedge\left((f i, z) \in S^{*}\right) \wedge(z, F(S u c i)) \in\right.\right.$ $\left.N S^{*}\right)$ )
by (rule choice)
then obtain $f$
where $\forall i .\left(\exists z .(F i, f i) \in S \wedge\left((f i, z) \in S^{*}\right) \wedge(z, F(S u c i)) \in N S^{*}\right) .$.
then have $\exists g . \forall i .(F i, f i) \in S \wedge(f i, g i) \in S^{*} \wedge(g i, F(S u c i)) \in N S^{*}$ by (rule choice)
then obtain $g$ where $\forall i .(F i, f i) \in S \wedge(f i, g i) \in S^{*} \wedge(g i, F(S u c i))$ $\in N S^{*}$..
then have $\forall i .(f i, g i) \in S^{*} \wedge(g i, F(S u c i)) \in N S^{*} \wedge(F($ Suc $i), f($ Suc
i)) $\in S$ by auto
then have $\forall i .(f i, g i) \in S^{*} \wedge(g i, f($ Suc $i)) \in S$ unfolding relcomp-def using tr-compat by auto
then have all: $\forall$ i. $(f i, g i) \in S^{*} \wedge(g i, f(S u c i)) \in S^{+}$by auto
have $\forall i .(f i, f($ Suc $i)) \in S^{+}$
proof
fix $i$
from all have $(f i, g i) \in S^{*} \wedge(g i, f($ Suc $i)) \in S^{+} .$.
then show $(f i, f(S u c i)) \in S^{+}$using transitive-closure-trans by auto
qed
then have $\exists x . f 0=x \wedge$ chain $\left(S^{+}\right) f$ by auto
then obtain $x$ where $f 0=x \wedge$ chain $\left(S^{+}\right) f$ by auto
then have $\exists f . f 0=x \wedge$ chain $\left(S^{+}\right) f$ by auto
then have $\neg S N$-on $\left(S^{+}\right)\{x\}$ by auto
then have $\neg S N\left(S^{+}\right)$unfolding $S N$-defs by auto
then have wfSconv: $\neg$ wf $\left(\left(S^{+}\right)^{-1}\right)$ using $S N$-iff-wf by auto
from $S N$ have $w f\left(S^{-1}\right)$ using $S N-i m p-w f[$ where $? r=S$ ] by simp
with $w f$-converse-trancl wfSconv show False by auto
qed
lemma compatible-rtrancl-split:
assumes compat: $N S O S \subseteq S$
and steps: $(x, y) \in(N S \cup S)^{*}$
shows $(x, y) \in S O S^{*} O N S^{*} \cup N S^{*}$
proof-
from steps have $\exists n .(x, y) \in(N S \cup S) \uparrow n$ using rtrancl-imp-relpow [where $? R=N S \cup S]$ by auto
then obtain $n$ where $(x, y) \in(N S \cup S)^{\wedge} n$ by auto
then show $(x, y) \in S O S^{*} O N S^{*} \cup N S^{*}$
proof (induct $n$ arbitrary: $x$, simp)
case (Suc m)
assume $(x, y) \in(N S \cup S) \leadsto(S u c m)$
then have $\exists z .(x, z) \in(N S \cup S) \wedge(z, y) \in(N S \cup S){ }^{\wedge} m$
using relpow-Suc-D2 [where ? $R=N S \cup S$ ] by auto
then obtain $z$ where $x z:(x, z) \in(N S \cup S)$ and $z y:(z, y) \in(N S \cup S) \leadsto m$ by

```
auto
    with Suc have zy:(z,y)\inSOS* ONS* \cupNS* by auto
    then show (x,y)\inSOS S* ONS*}\cupN\mp@subsup{S}{}{*
    proof (cases (x,z)\inNS)
        case True
        from compat compat-tr-compat have trCompat: NS* OS\subseteqS by blast
        from zy True have (x,y)\in(NSOSOS* ONS*)\cup(NSO NS*) by auto
        then have }(x,y)\in((NSOS)O\mp@subsup{S}{}{*}ON\mp@subsup{S}{}{*})\cup(NSONS*) by aut
        then have }(x,y)\in((N\mp@subsup{S}{}{*}OS)O\mp@subsup{S}{}{*}ON\mp@subsup{S}{}{*})\cup(NSON\mp@subsup{S}{}{*})\mathrm{ by auto
        with trCompat have xy:(x,y) \in(SOS S* ONS*)\cup (NSONS*) by auto
        have NSONS*}\subseteqNS** by aut
        with }xy\mathrm{ show (x,y) ( SO S* ONS*) UNS* by auto
    next
        case False
        with }xz\mathrm{ have }xz:(x,z)\inS\mathrm{ by auto
        with zy have (x,y) \inSO(SO S* O NS* \cupNS*) by auto
        then show (x,y)\in(SOS S* ONS*)\cupNS* using right-comp-S by simp
    qed
    qed
qed
lemma compatible-conv:
    assumes compat: NS OS\subseteqS
    shows (NS\cupS)* OSO(NS\cupS)*}=SO\mp@subsup{S}{}{*}ON\mp@subsup{S}{}{*
proof -
    let ?NSuS = NS \cupS
    let ?NSS =SO S* O NS*
    let ?midS = ?NSuS* O S O ?NSuS*
    have one: ?NSS \subseteq?midS by regexp
    have ?NSuS* OS\subseteq(?NSS\cupNS*)OS
    using compatible-rtrancl-split [where S=S and NS =NS] compat by blast
    also have \ldots\subseteq? NSSS OS\cupNS* OS by auto
    also have \ldots\subseteq?NSSS OS\cupS using compat compat-tr-compat [where S=S
and NS = NS] by auto
    also have ...\subseteqSO ?NSuS* by regexp
    finally have ?midS\subseteqSO ?NSuS* O ?NSuS** by blast
    also have \ldots\subseteqSO ?NSuS* by regexp
    also have \ldots\subseteqSO(?NSS \cupNS*)
    using compatible-rtrancl-split [where S=S and NS = NS] compat by blast
    also have ...\subseteq?NSS by regexp
    finally have two: ?midS \subseteq? NSS .
    from one two show ?thesis by auto
qed
lemma compatible-SN':
    assumes compat: NS OS\subseteqS and SN:SNS
    shows SN((NS\cupS)* OSO (NS\cupS)*)
using compatible-conv [where S=S and NS=NS]
    compatible-SN [where S=S and NS=NS] assms by force
```

```
lemma rtrancl-diff-decomp:
    assumes (x,y)\in\mp@subsup{A}{}{*}-\mp@subsup{B}{}{*}
    shows (x,y)\in A* O (A-B)O A*
proof -
    from assms have A: (x,y)\in\mp@subsup{A}{}{*}\mathrm{ and B:(x,y) & B* by auto}
    from A have }\existsk.(x,y)\in\mp@subsup{A}{}{~}k\mathrm{ by (rule rtrancl-imp-relpow)
    then obtain }k\mathrm{ where Ak:(x,y) & A ` k by auto
    from Ak B show (x,y) \in A* O (A-B)O A*
    proof (induct k arbitrary: x)
        case 0
        with 〈(x,y)\not\in B*〉0 show ?case using ccontr by auto
    next
        case (Suc i)
    then have B:(x,y)\not\in\mp@subsup{B}{}{*}\mathrm{ and ASk:(x,y) &A ~ Suc i by auto}
    from ASk have \existsz. (x,z)\inA\wedge(z,y)\inA^^i using relpow-Suc-D2 [where
?R=A] by auto
    then obtain z where xz:(x,z)\inA and (z,y)\inA ~ i by auto
    then have zy:(z,y)\in\mp@subsup{A}{}{*}\mathrm{ using relpow-imp-rtrancl by auto}
    from xz show (x,y)\in\mp@subsup{A}{}{*}O(A-B)O\mp@subsup{A}{}{*}
    proof (cases (x,z)\inB)
        case False
            with xz zy show (x,y)\in A* O(A-B)O A* by auto
    next
            case True
            then have }(x,z)\in\mp@subsup{B}{}{*}\mathrm{ by auto
            have}\llbracket(x,z)\in\mp@subsup{B}{}{*};(z,y)\in\mp@subsup{B}{}{*}\rrbracket\Longrightarrow(x,y)\in\mp@subsup{B}{}{*}\mathrm{ using rtrancl-trans [of x z
B] by auto
            with }\langle(x,z)\in\mp@subsup{B}{}{*}\rangle\langle(x,y)\not\in\mp@subsup{B}{}{*}>\mathrm{ have (z,y) & B* by auto
            with Suc«(z,y)\inA ~ i〉 have (z,y)\in\mp@subsup{A}{}{*}O(A-B)O\mp@subsup{A}{}{*}\mathrm{ by auto}
            with xz have xy:(x,y) \inAO O* O (A-B)O A* by auto
            have AO A* O(A-B)O\mp@subsup{A}{}{*}\subseteq\mp@subsup{A}{}{*}O(A-B)O\mp@subsup{A}{}{*}\mathrm{ by regexp}
            from this xy show (x,y)\in\mp@subsup{A}{}{*}O(A-B)O\mp@subsup{A}{}{*}
                using subsetD [where ?A=A O A* O (A-B)O A*] by auto
    qed
    qed
qed
lemma \(S N\)－empty［simp］：\(S N\}\) by auto
lemma \(S N\)－on－weakening：
    assumes SN-on R1 A
    shows SN-on (R1\capR2)A
proof -
    {
        assume }\existsS.S0\inA\wedge\operatorname{chain}(R1\capR2)
        then obtain S where
            SO:S0\inA and
            SN:chain (R1\capR2) S
```


## qed

definition ideriv :: 'a rel $\Rightarrow{ }^{\prime}$ a rel $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow$ bool where
ideriv $R S$ as $\longleftrightarrow(\forall$ i. (as i, as $($ Suc $i)) \in R \cup S) \wedge(I N F M i .($ as $i$, as (Suc $i)$ )
$\in R$ )
lemma ideriv-mono: $R \subseteq R^{\prime} \Longrightarrow S \subseteq S^{\prime} \Longrightarrow$ ideriv $R S$ as $\Longrightarrow$ ideriv $R^{\prime} S^{\prime}$ as unfolding ideriv-def INFM-nat by blast
fun
shift $::\left(n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow n a t \Rightarrow n a t \Rightarrow{ }^{\prime} a$
where

$$
\text { shift } f j=(\lambda i . f(i+j))
$$

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

```
            show \existsj>m.(shift as (Suc i) j, shift as (Suc i) (Suc j)) \inR-D
            by (rule exI [of-j - Suc i], insert j RD, auto)
        qed
    qed
qed
lemma ideriv-SN:
    assumes SN: SN S
        and compat: NS OS\subseteqS
        and R:R\subseteqNS\cupS
    shows \neg ideriv (S\capR) (R-S) as
proof
    assume ideriv}(S\capR)(R-S) a
    with R have steps: }\forall\mathrm{ i. (as i, as (Suc i)) }\inNS\cup
        and inf:INFM i. (as i, as (Suc i)) \inS \capR unfolding ideriv-def by auto
    from non-strict-ending [OF steps compat] SN
    obtain i where i: \j.j\geqi\Longrightarrow(as j, as (Suc j)) \inNS - S by fast
    from inf [unfolded INFM-nat] obtain j where j>i and (as j, as (Suc j)) \inS
by auto
    with i [of j] show False by auto
qed
lemma Infm-shift: (INFM i.P (shift f n i)) =(INFM i.P (f i)) (is ?S = ?O)
proof
    assume ?S
    show ?O
            unfolding INFM-nat-le
    proof
        fix m
        from <?S〉 [unfolded INFM-nat-le]
        obtain k where k:k\geqm}\mathrm{ and p:P(shift fnk) by auto
        show }\existsk\geqm.P(fk
            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\geqm+n and p: P (fk) by auto
            show \exists k\geqm.P(shift f n k)
                by (rule exI [of - k-n], insert k p,auto)
    qed
qed
lemma rtrancl-list-conv:
    (s,t) \in R* \longleftrightarrow
```

```
    (\exists ts.last (s#ts) = t ^(\forall i<length ts. ((s#ts)!i,(s# ts)!Suc i)\inR))
(is ?l=?r)
proof
    assume ?r
    then obtain ts where last (s#ts)=t\wedge(\foralli<length ts. ((s#ts)!i,(s#ts)
!Suc i) \in R) ..
    then show ?l
    proof (induct ts arbitrary: s, simp)
    case (Cons u ll)
    then have last (u# ll) = t\wedge (\forall i<length ll. ((u# ll)!i,(u# ll)!Suc i)\in
R) by auto
    from Cons(1)[OF this] have rec: (u,t)\in R* .
    from Cons have (s,u)\inR by auto
    with rec show ?case by auto
    qed
next
assume ?l
from rtrancl-imp-seq [OF this]
obtain Sn where s:S 0=s and t:S n=t and steps: }\foralli<n.(Si,S (Su
i)) }\inR\mathrm{ by auto
    let ?ts = map (\lambda i.S (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 }\foralli<length ?ts. ((s# ?ts)!i,(s# ?ts)!Suc i)\in
        proof (intro allI impI)
            fix }
            assume i: i< length ?ts
            then show ((s# ?ts)!i, (s#?ts)!Suc i)\inR
            proof (cases i, simp add:s [symmetric] steps)
                case (Suc j)
                with i steps show ?thesis by simp
            qed
        qed
    qed
qed
lemma SN-reaches-NF:
assumes SN-on r {x}
shows }\existsy.(x,y)\in\mp@subsup{r}{}{*}\wedgey\inNF
using assms
proof (induct rule: SN-on-induct')
case (IH x)
show ?case
proof (cases x N NF r)
    case True
    then show ?thesis by auto
next
    case False
```

```
    then obtain }y\mathrm{ where step: }(x,y)\inr\mathrm{ by auto
    from IH [OF this] obtain z where steps: (y,z)\in\mp@subsup{r}{}{*}\mathrm{ and NF:z }|NFr\mathrm{ by}
auto
    show ?thesis
        by (intro exI, rule conjI [OF - NF], insert step steps, auto)
    qed
qed
lemma SN-WCR-reaches-NF:
    assumes SN: SN-on r {x}
    and WCR:WCR-on r {x.SN-on r {x}}
    shows }\exists!y.(x,y)\in\mp@subsup{r}{}{*}\wedgey\inNF
proof -
    from SN-reaches-NF [OF SN] obtain y where steps: (x,y)\in r* and NF:y\in
NF r by auto
    show ?thesis
    proof(rule, rule conjI [OF steps NF])
        fix z
        assume steps':}(x,z)\in\mp@subsup{r}{}{*}\wedgez\inNF
        from Newman-local [OF SN WCR] have CR-on r {x} by auto
        from CR-onD [OF this - steps] steps' have (y,z)\in r ' by simp
        from join-NF-imp-eq [OF this NF] steps' show z = y by simp
    qed
qed
definition some-NF :: 'a rel }=>\mp@subsup{}{}{\prime}a=>\mp@subsup{}{}{\prime}a\mathrm{ where
    some-NF r x = (SOME y. (x,y) \in r* ^ y f NFr)
lemma some-NF:
    assumes SN: SN-on r {x}
    shows (x, some-NF r x) \in r* ^ some-NF r x E NF r
    using someI-ex [OF SN-reaches-NF [OF SN]]
    unfolding some-NF-def .
lemma some-NF-WCR:
    assumes SN:SN-on r {x}
        and WCR:WCR-on r {x.SN-on r {x}}
        and steps:}(x,y)\in\mp@subsup{r}{}{*
        and NF:y GNFr
    shows }y=\mathrm{ some-NF r x
proof -
    let ?p = \lambda y. (x,y) \in r* ^ y \inNFr
    from SN-WCR-reaches-NF [OF SN WCR]
    have one: \exists! y. ?p y .
    from steps NF have y: ?p y ..
    from some-NF [OF SN] have some: ?p (some-NF r x).
    from one some y show ?thesis by auto
qed
```

```
lemma some-NF-UNF:
    assumes UNF:UNF r
        and steps:}(x,y)\in\mp@subsup{r}{}{*
        and NF: y GNFr
    shows }y=\mathrm{ some-NF r x
proof -
    let ?p = \lambda y. (x,y) \in r* ^ y \inNFr
    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)\in\mp@subsup{r}{}{!}\mathrm{ by auto
    from pNF have nf:(x, some-NF r x) \in r! by auto
    from UNF [unfolded UNF-on-def] y nf show ?thesis by auto
qed
definition the-NF A a = (THE b. (a,b) \in A')
context
    fixes }
    assumes SN:SN A and CR:CR A
begin
lemma the-NF:(a, the-NF A a) \in A!
proof -
    obtain b}\mathrm{ where ab: (a,b) & A! using SN by (meson SN-imp-WN UNIV-I
WN-onE)
    moreover have (a,c)\in A
        using CR and ab by (meson CR-divergence-imp-join join-NF-imp-eq normal-
izability-E)
    ultimately have }\exists!b.(a,b)\in\mp@subsup{A}{}{!}\mathrm{ by blast
    then show ?thesis unfolding the-NF-def by (rule theI')
qed
lemma the-NF-NF: the-NF A a \inNF A
    using the-NF by (auto simp: normalizability-def)
lemma the-NF-step:
    assumes (a,b)\inA
    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) \in A*
    shows the-NF A a = the-NF A b
    using assms by (induct) (auto dest: the-NF-step)
lemma the-NF-conv:
    assumes (a,b)\in A⿱艹*
```

shows the- NF A $a=$ the-NF $A b$
using assms
by (meson CR WN-on-def the-NF semi-complete-imp-conversionIff-same-NF semi-complete-onI)
end
definition weak-diamond :: 'a rel $\Rightarrow$ bool $(w \diamond)$ where $w \diamond r \longleftrightarrow\left(r^{-1} O r\right)-I d \subseteq\left(r O r^{-1}\right)$
lemma weak-diamond-imp-CR:
assumes $w d: w \diamond r$
shows $C R r$
proof (rule semi-confluence-imp-CR, rule)
fix $x y$
assume $(x, y) \in r^{-1} O r^{*}$
then obtain $z$ where step: $(z, x) \in r$ and steps: $(z, y) \in r^{*}$ by auto
from steps
have $\exists u .(x, u) \in r^{*} \wedge(y, u) \in r^{=}$
proof (induct)
case base
show ?case
by (rule exI $[o f-x]$, insert step, auto)

## next

case (step $y^{\prime} y$ )
from step (3) obtain $u$ where $x u:(x, u) \in r^{*}$ and $y^{\prime} u:\left(y^{\prime}, u\right) \in r^{=}$by auto
from $y^{\prime} u$ have $\left(y^{\prime}, u\right) \in r \vee y^{\prime}=u$ by auto
then show? ?case
proof
assume $y^{\prime} u: y^{\prime}=u$
with $x u$ step (2) have $x y:(x, y) \in r^{*}$ by auto
show ?thesis
by (intro exI conjI, rule xy, simp)
next
assume $\left(y^{\prime}, u\right) \in r$
with step(2) have uy: $(u, y) \in r^{-1} O r$ by auto
show ?thesis
proof (cases $u=y$ )
case True
show ?thesis
by (intro exI conjI, rule xu, unfold True, simp)
next
case False
with $u y$
$w d$ [unfolded weak-diamond-def] obtain $u^{\prime}$ where $u u^{\prime}:\left(u, u^{\prime}\right) \in r$
and $y u^{\prime}:\left(y, u^{\prime}\right) \in r$ by auto
from $x u u u^{\prime}$ have $x u:\left(x, u^{\prime}\right) \in r^{*}$ by auto show ?thesis
by (intro exI conjI, rule $x u$, insert $y u^{\prime}$, auto)

```
        qed
        qed
    qed
    then show (x,y)\in r b}\mathrm{ by auto
qed
lemma steps-imp-not-SN-on:
    fixes }t:: ' a=>'
        and }R\mathrm{ :: 'b rel
    assumes steps: \ x. (t x,t (f x)) \inR
    shows }\neg\mathrm{ SN-on R {t x}
proof
    let ?U = range t
    assume SN-on R {t x}
    from SN-on-imp-on-minimal [OF this, rule-format, of ?U]
    obtain tz where tz: tz\in range t and min: \bigwedge y. (tz,y) \inR\Longrightarrowy\not\in range t
by auto
    from tz obtain z where tz: tz=tz by auto
    from steps [of z] min [of t (fz)] show False unfolding tz by auto
qed
lemma steps-imp-not-SN:
    fixes }t:: 'a=>'
        and }R:: 'b re
    assumes steps: \ x. (t x,t (fx)) \inR
    shows \negSN R
proof -
    from steps-imp-not-SN-on [of t f R,OF steps]
    show ?thesis unfolding SN-def by blast
qed
lemma steps-map:
    assumes fg: \tu R.Pt\LongrightarrowQ R\Longrightarrow(t,u)\inR\LongrightarrowPu^(ft,fu)\ingR
    and t:Pt
    and R:QR
    and S:QS
    shows ((t,u)\in\mp@subsup{R}{}{*}\longrightarrow(ft,fu)\in(gR\mp@subsup{)}{}{*})
    \wedge((t,u) \in R*}OSO\mp@subsup{R}{}{*}\longrightarrow(ft,fu)\in(gR\mp@subsup{)}{}{*}O(gS)O(gR\mp@subsup{)}{}{*}
proof -
    {
        fix }t
        assume (t,u)\in R* and Pt
        then have Pu}\wedge(ft,fu)\in(gR\mp@subsup{)}{}{*
        proof (induct)
            case (step u v)
            from step(3)[OF step(4)] have Pu: Pu and steps: (ft,fu)\in(g R)* by
auto
            from fg [OF Pu R step(2)] have Pv: P v and step: (fu,fv)\ingR by auto
            with steps have (ft,fv) \in(gR)* by auto
```

```
        with Pv show ?case by simp
    qed simp
    } note main = this
    note maint = main [OF - t]
    from maint [of u] have one: }(t,u)\in\mp@subsup{R}{}{*}\longrightarrow\longrightarrow(ft,fu)\in(gR\mp@subsup{)}{}{*}\mathrm{ by simp
    show ?thesis
    proof (rule conjI [OF one impI])
    assume (t,u)\in\mp@subsup{R}{}{*}OSO\mp@subsup{R}{}{*}
    then obtain sv where ts: (t,s)\in R* and sv: (s,v)\inS and vu:(v,u)\in
R* by auto
    from maint [OF ts] have Ps: P s and ts: (ft,fs)\in(gR)* by auto
    from fg[OF Ps S sv] have Pv: Pv and sv: (fs,fv)\ingS by auto
    from main [OF vu Pv] have vu: (fv,fu)\in(gR)* by auto
    from ts sv vu show (ft,fu)\in(gR)*}OgSO(gR\mp@subsup{)}{}{*}\mathrm{ 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: $(\bigwedge y .(x, y) \in r \Longrightarrow y \in S N$-part $r) \Longrightarrow x \in S N$-part $r$
The accessible part of a relation is the same as the terminating part (just two names for the same definition - modulo argument order). See ( $\bigwedge y .(y$, $? x) \in$ ? $r \Longrightarrow y \in$ Wellfounded.acc ? $r) \Longrightarrow$ ? $x \in$ Wellfounded.acc ? $r$.

Characterization of SN -on via terminating part.

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

    Special case for "full" termination.
    lemma $S N$-SN-part-UNIV-conv:
$S N r \longleftrightarrow S N$-part $r=U N I V$
using SN-on-SN-part-conv [of r UNIV] by auto
lemma closed-imp-rtrancl-closed: assumes $L: L \subseteq A$
and $R$ : $R$ " $A \subseteq A$

```
    shows {t| s. s\inL^(s,t)\inR`*}\subseteqA
proof -
    {
        fix st
        assume (s,t)\inR`* and s\inL
        hence }t\in
            by (induct, insert L R, auto)
    }
    thus ?thesis by auto
qed
lemma trancl-steps-relpow: assumes a\subseteqb^+
    shows }(x,y)\in\mp@subsup{a}{}{\wedge}n\Longrightarrow\Longrightarrow\existsm.m\geqn^^(x,y)\in b^
proof (induct n arbitrary: y)
    case 0 thus ?case by (intro exI[of-0],auto)
next
    case (Suc n z)
    from Suc(2) obtain y where xy: (x,y)\ina^~n and yz:(y,z)\ina by auto
    from Suc(1)[OF xy] obtain m where m:m\geqn and xy: (x,y)\inb~~m}\mathrm{ by
auto
    from yz assms have ( }y,z)\in\mp@subsup{b}{}{`}+\mathrm{ by auto
    from this[unfolded trancl-power] obtain k where k: k>0 and yz: (y,z)\inb^^
k by auto
    from xy yz have (x,z) \in b~~}(m+k) unfolding relpow-add by aut
    with }km\mathrm{ show ?case by (intro exI[of-m+k], auto)
qed
lemma relpow-image: assumes f: \bigwedgest. (s,t)\inr\Longrightarrow(fs,ft)\in\mp@subsup{r}{}{\prime}
    shows}(s,t)\inr``n\Longrightarrow(fs,ft)\in\mp@subsup{r}{}{\prime}\leadsto~
proof (induct n arbitrary: t)
    case (Suc n u)
    from Suc(2) obtain t where st: (s,t) \inr^^n and tu: (t,u) \inr by auto
    from Suc(1)[OF st] f[OF tu] show ?case by auto
qed auto
lemma relpow-refl-mono:
    assumes refl:\}\x.(x,x)\in\mathrm{ Rel
    shows m\leqn\Longrightarrow(a,b) \inRel^^m\Longrightarrow(a,b)\inRel^n}
proof (induct rule:dec-induct)
    case (step i)
    hence abi:(a,b)\inRel ~ i by auto
    from refl[of b] abi relpowp-Suc-I[of i \lambda x y. (x,y) \in Rel] show (a,b) \in Rel ^~
Suc i by auto
qed
lemma SN-on-induct-acc-style [consumes 1, case-names IH]:
    assumes sn:SN-on R {a}
        and IH:\x.SN-on R {x}\Longrightarrow\llbracket\y. (x,y)\inR\LongrightarrowPy\rrbracket\LongrightarrowPx
    shows Pa
```

```
proof -
    from sn SN-on-conv-acc [of R R }\mp@subsup{R}{}{-1}a]\mathrm{ have a: a f termi R by auto
    show ?thesis
    proof (rule Wellfounded.acc.induct [OF a, of P], rule IH)
        fix }
        assume }\bigwedgey.(y,x)\in\mp@subsup{R}{}{-1}\Longrightarrowy\in\mathrm{ termi }
        from this [folded SN-on-conv-acc]
            show SN-on R {x} by simp fast
        qed auto
qed
```

lemma partially-localize- $C R$ :
$C R r \longleftrightarrow\left(\forall x y z .(x, y) \in r \wedge(x, z) \in r^{*} \longrightarrow(y, z) \in j o i n r\right)$
proof
assume $C R r$
thus $\forall x y z .(x, y) \in r \wedge(x, z) \in r^{*} \longrightarrow(y, z) \in$ join $r$ by auto
next
assume $1: \forall x$ y $z .(x, y) \in r \wedge(x, z) \in r^{*} \longrightarrow(y, z) \in$ join $r$
show $C R r$
proof
fix $a b c$
assume 2: $a \in U N I V$ and 3: $(a, b) \in r^{*}$ and $4:(a, c) \in r^{*}$
then obtain $n$ where $(a, c) \in r^{\wedge} n$ using rtrancl-is-UN-relpow by fast
with 23 show $(b, c) \in$ join $r$
proof (induct $n$ arbitrary: $a b c$ )
case 0 thus ?case by auto
next
case (Suc m)
from $\operatorname{Suc}(4)$ obtain $d$ where $a d:(a, d) \in r^{\sim} m$ and $d c:(d, c) \in r$ by auto
from $\operatorname{Suc}(1)[O F \operatorname{Suc}(2) \operatorname{Suc}(3) a d]$ have $(b, d) \in j o i n r$.
with 1 dc joinE joinI [of b-rc] join-rtrancl-join show ?case by metis
qed
qed
qed
definition strongly-confluent-on :: 'a rel $\Rightarrow$ 'a set $\Rightarrow$ bool
where
strongly-confluent-on $r A \longleftrightarrow$
$\left(\forall x \in A . \forall y z .(x, y) \in r \wedge(x, z) \in r \longrightarrow\left(\exists u .(y, u) \in r^{*} \wedge(z, u) \in r^{=}\right)\right)$
abbreviation strongly-confluent :: 'a rel $\Rightarrow$ bool
where
strongly-confluent $r \equiv$ strongly-confluent-on r UNIV
lemma strongly-confluent-on-E11:
strongly-confluent-on $r A \Longrightarrow x \in A \Longrightarrow(x, y) \in r \Longrightarrow(x, z) \in r \Longrightarrow$
$\exists u .(y, u) \in r^{*} \wedge(z, u) \in r^{=}$
unfolding strongly-confluent-on-def by blast

## lemma strongly-confluentI [intro]:

$\llbracket \bigwedge x y z .(x, y) \in r \Longrightarrow(x, z) \in r \Longrightarrow \exists u .(y, u) \in r^{*} \wedge(z, u) \in r^{=} \rrbracket \Longrightarrow$ strongly-confluent $r$
unfolding strongly-confluent-on-def by auto
lemma strongly-confluent-E1n: assumes scr: strongly-confluent $r$
shows $(x, y) \in r^{=} \Longrightarrow(x, z) \in r \leadsto n \Longrightarrow \exists u .(y, u) \in r^{*} \wedge(z, u) \in r^{=}$
proof (induct $n$ arbitrary: $x y z$ )
case (Suc m)
from $\operatorname{Suc}(3)$ obtain $w$ where $x w:(x, w) \in r^{\wedge} m$ and $w z:(w, z) \in r$ by auto from $\operatorname{Suc}(1)[O F S u c(2) x w]$ obtain $u$ where $y u:(y, u) \in r^{*}$ and $w u:(w, u)$ $\in r^{=}$by auto
from strongly-confluent-on-E11 [OF scr, of w] wz yu wu show ?case
by (metis UnE converse-rtrancl-into-rtrancl iso-tuple-UNIV-I pair-in-Id-conv rtrancl-trans)
qed auto
lemma strong-confluence-imp-CR:
assumes strongly-confluent $r$
shows $C R r$
proof -
$\{$ fix $x y z$
have $(x, y) \in r \Longrightarrow(x, z) \in r^{*} \Longrightarrow(y, z) \in$ join $r$
by (cases $x=y$, insert strongly-confluent-E1n [OF assms], blast+) $\}$
then show $C R r$ using partially-localize- $C R$ by blast
qed
lemma $W C R$-alt-def: $W C R A \longleftrightarrow A^{-1} O A \subseteq A^{\downarrow}$ by (auto simp: WCR-defs)
lemma NF-imp-SN-on: $a \in N F R \Longrightarrow S N$-on $R\{a\}$ unfolding $S N$-on-def NF-def by blast
lemma Union-sym: $(s, t) \in\left(\bigcup i \leq n .(S i)^{\leftrightarrow}\right) \longleftrightarrow(t, s) \in\left(\bigcup i \leq n .(S i)^{\leftrightarrow}\right)$ by auto
lemma peak-iff: $(x, y) \in A^{-1} O B \longleftrightarrow(\exists u .(u, x) \in A \wedge(u, y) \in B)$ by auto
lemma $C R-N F-c o n v$ :
assumes $C R r$ and $t \in N F r$ and $(u, t) \in r^{\leftrightarrow *}$
shows $(u, t) \in r^{!}$
using assms
unfolding $C R$-imp-conversionIff-join $[O F\langle C R \quad r\rangle]$
by (auto simp: NF-iff-no-step normalizability-def)
(metis (mono-tags) converse-rtranclE joinE)
lemma NF-join-imp-reach:

```
    assumes \(y \in N F A\) and \((x, y) \in A^{\downarrow}\)
    shows \((x, y) \in A^{*}\)
using assms by (auto simp: join-def) (metis NF-not-suc rtrancl-converseD)
lemma conversion-O-conversion [simp]:
\(A^{\leftrightarrow *} O A^{\leftrightarrow *}=A^{\leftrightarrow *}\)
by (force simp: converse-def)
lemma trans- \(O\)-iff: trans \(A \longleftrightarrow A O A \subseteq A\) unfolding trans-def by auto lemma refl-O-iff: refl \(A \longleftrightarrow I d \subseteq A\) unfolding refl-on-def by auto
lemma relpow-Suc: \(r\) ~Suc \(n=r O r \leadsto n\) using relpow-add \([\) of \(1 n r]\) by auto
lemma converse-power: fixes \(r::\) ' \(a\) rel shows \(\left(r^{-1}\right)^{\wedge} n=\left(r^{\wedge} n\right)^{-1}\)
proof (induct \(n\) )
case (Suc n)
show ?case unfolding relpow.simps(2)[of - \(\left.r^{-1}\right]\) relpow-Suc \([o f-r]\)
by (simp add: Suc converse-relcomp)
qed \(\operatorname{simp}\)
lemma conversion-mono: \(A \subseteq B \Longrightarrow A^{\leftrightarrow *} \subseteq B^{\leftrightarrow *}\)
by (auto simp: conversion-def intro!: rtrancl-mono)
lemma conversion-conversion-idemp \([\) simp \(]:\left(A^{\leftrightarrow *}\right)^{\leftrightarrow *}=A^{\leftrightarrow *}\) by auto
lemma lower-set-imp-not-SN-on:
assumes \(s \in X \forall t \in X . \exists u \in X .(t, u) \in R\) shows \(\neg S N\)-on \(R\{s\}\)
by (meson \(S N\)-on-imp-on-minimal assms)
```

lemma $S N$-on-Image-rtrancl-iff[simp]: SN-on $R\left(R^{*} " X\right) \longleftrightarrow S N$-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: $R \subseteq R^{\prime} \Longrightarrow S O R \subseteq S O R^{\prime}$ by auto
lemma O-mono2: $R \subseteq R^{\prime} \Longrightarrow R O T \subseteq R^{\prime} O T$ by auto
lemma rtrancl-O-shift: $\left(\begin{array}{ll}S O R\end{array}\right)^{*} O S=S O(R O S)^{*}$
proof (intro equalityI subrelI)
fix $x y$
assume $(x, y) \in(S O R)^{*} O S$
then obtain $n$ where $(x, y) \in(S O R)^{\wedge} n O S$ by blast
then show $(x, y) \in S O(R O S)^{*}$
proof (induct $n$ arbitrary: $y$ )

```
    case IH:(Suc n)
    then obtain z where xz: (x,z)\in(SOR)^^n OS and zy:(z,y)\inROS by
auto
    from IH.hyps[OF xz] zy have (x,y)\inSO(ROS)*}OROS by aut
    then show ?case by(fold trancl-unfold-right, auto)
    qed auto
next
    fix }x
    assume (x,y) \inSO(ROS)*
    then obtain n where (x,y) \inSO(ROS)^n by blast
    then show (x,y)\in(SOR)*}O
    proof(induct n arbitrary: y)
        case IH:(Suc n)
        then obtain z where xz: (x,z) \inSO(ROS)^n and zy: (z,y)\inROS by
auto
        from IH.hyps[OF xz] zy have (x,y)\in((SO R)* OS O R)OS 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: RO(SOR)*}OS=(ROS\mp@subsup{)}{}{+
    by (unfold rtrancl-O-shift trancl-unfold-left, auto)
lemma SN-on-subset-SN-terms:
    assumes SN:SN-on R X shows X\subseteq{x.SN-on R {x}}
proof(intro subsetI, unfold mem-Collect-eq)
    fix }x\mathrm{ assume }x:x\in
    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\cupY)
    using assms by fast
lemma SN-on-UN:
    assumes }\x.SN-on R(Xx) shows SN-on R(Ux. X x
    using assms by fast
lemma Image-subsetI: R\subseteqR'\LongrightarrowR" 'X\subseteqR'"X by auto
lemma SN-on-O-comm:
    assumes SN:SN-on ((R::('a\times'b) set) O (S::('b\times'a) set)) (S " X)
    shows SN-on (SOR)X
proof
    fix seq :: nat = 'b assume seq0: seq 0 \inX and chain: chain (SO R) seq
    from SN have SN:SN-on (ROS) ((ROS)* " S"X) by simp
    { fix ia
    assume ia: (seq i,a)\inS and aSi:(a,seq (Suc i)) \inR
```

```
    have seq i\in(SOR)* " X
    proof (induct i)
    case 0 from seq0 show ?case by auto
    next
        case (Suc i) with chain have seq (Suc i) \in((SOR)* OSOR)" X by
blast
    also have \ldots\subseteq(SOR)* "X by (fold trancl-unfold-right, auto)
    finally show ?case.
    qed
    with ia have a\in((SOR)* OS) " X by auto
    then have a: a\in((ROS\mp@subsup{)}{}{*}) "S "X by (auto simp: rtrancl-O-shift)
    with ia aSi have False
    proof(induct a arbitrary: i rule: SN-on-induct[OF SN])
        case 1 show ?case by (fact a)
    next
        case IH:(2 a)
        from chain obtain b
        where *: (seq (Suc i),b)\inS(b, seq (Suc (Suc i))) \inR by auto
        with IH have ab: (a,b)\inROS by auto
        with}\langlea\in(ROS\mp@subsup{)}{}{*}"S\mathrm{ " X> have b f((ROS)* OROS) " S"X by
auto
        then have b\in(ROS)* " S"X
                by (rule rev-subsetD, intro Image-subsetI, fold trancl-unfold-right, auto)
            from IH.hyps[OF ab*this] IH.prems ab show False by auto
        qed
    }
    with chain show False by auto
qed
lemma SN-O-comm: SN (RO S)\longleftrightarrowSN(SOR)
    by (intro iffI; rule SN-on-O-comm[OF SN-on-subset2], auto)
lemma chain-mono: assumes R'\subseteqR chain R' seq shows chain R seq
    using assms by auto
context
    fixes SR
    assumes push: SOR\subseteqROS*
begin
lemma rtrancl-O-push: S* OR\subseteqROS*
proof-
    { fix n
        have \st. (s,t)\inS^^nOR\Longrightarrow(s,t)\inROS*
        proof(induct n)
            case (Suc n)
                    then obtain u where (s,u)\inS(u,t)\inROS* unfolding relpow-Suc by
blast
            then have (s,t)\inSOROS* by auto
```

```
            also have ...\subseteqRO S* OS using push by blast
            also have \ldots\subseteqROS* by auto
            finally show ?case.
        qed auto
    }
    thus ?thesis by blast
qed
lemma rtrancl-U-push: (S\cupR)*}=\mp@subsup{R}{}{*}O\mp@subsup{S}{}{*
proof(intro equalityI subrelI)
    fix }x
    assume (x,y) \in(S\cupR)*
    also have ...\subseteq(S* OR * O S* by regexp
    finally obtain z where xz: (x,z) \in(S*OR)* and zy:(z,y)\inS* by auto
    from xz have (x,z)\in R* O S*
    proof (induct rule: rtrancl-induct)
    case (step z w)
        then have (x,w)\in R* OS S* OS S* OR by auto
        also have ...\subseteq 盾OS S* OR by regexp
        also have \ldots}\subseteq\subseteq\mp@subsup{R}{}{*}ORO\mp@subsup{S}{}{*}\mathrm{ using rtrancl-O-push by auto
        also have \ldots}\subseteq\subseteq\mp@subsup{R}{}{*}O\mp@subsup{S}{}{*}\mathrm{ by regexp
        finally show ?case.
    qed auto
    with zy show (x,y)\in R* O S* by auto
qed regexp
lemma SN-on-O-push:
    assumes SN: SN-on R X shows SN-on (R O S*)X
proof
    fix seq
    have SN:SN-on R (R* "X) using SN-on-Image-rtrancl[OF SN].
    moreover assume seq (0::nat) \inX
    then have seq 0 \in R* " }X\mathrm{ by auto
    ultimately show chain (ROS S ) seq \Longrightarrow False
    proof(induct seq 0 arbitrary: seq rule: SN-on-induct)
    case IH
    then have 01:(seq 0, seq 1) \inRO S*
            and 12:(seq 1, seq 2) }\inROS\mp@subsup{S}{}{*
            and 23:(seq 2, seq 3) \inRO S* by (auto simp: eval-nat-numeral)
    then obtain st
    where s:(seq 0, s)\inR and s1: (s, seq 1) \in S*
        and t:(seq 1, t)\inR and t2: (t, seq 2) }\in\mp@subsup{S}{}{*}\mathrm{ by auto
    from s1t have (s,t) \in S* OR by auto
    with rtrancl-O-push have st: (s,t)\inRO S* by auto
    from t2 23 have (t, seq 3) \in S* ORO S* by auto
    also from rtrancl-O-push have \ldots}\subseteqROS S* O S* by blas
    finally have t3: (t, seq 3) \inROS by regexp
    let ?seq = \lambdai.case i of 0=>s|Suc 0=>t|i=>seq (Suc i)
    show ?case
```

```
    proof(rule IH)
            from s show (seq 0, ?seq 0) \inR by auto
            show chain (ROS S*)?seq
            proof (intro allI)
                fix i show (?seq i, ?seq (Suc i)) \inRO S*
            proof (cases i)
                    case 0 with st show ?thesis by auto
            next
            case (Suc i) with t3 IH show ?thesis by (cases i, auto simp: eval-nat-numeral)
                qed
            qed
        qed
    qed
qed
lemma SN-on-Image-push:
    assumes SN: SN-on R X shows SN-on R (S* " X)
proof-
    {fix n
        have SN-on R ((S~n) " X)
        proof(induct n)
            case 0 from SN show ?case by auto
            case (Suc n)
            from SN-on-O-push[OF this] have SN-on (RO S*) ((S~~n)"X).
            from SN-on-Image[OF this]
            have SN-on (R O S*) ((ROS S*)" (S^~n) " X).
                    then have SN-on R ((ROS*)" (S~n) "X) by (rule SN-on-mono,
auto)
            from SN-on-subset2[OF Image-mono[OF push subset-refl] this]
                    have SN-on R (R " (S~ Suc n) " X) by (auto simp: relcomp-Image)
            then show ?case by fast
        qed
    }
    then show ?thesis by fast
qed
end
lemma not-SN-onI[intro]: f 0 \in X\Longrightarrow chain R f \Longrightarrow 
    by (unfold SN-on-def not-not, intro exI conjI)
lemma shift-comp[simp]: shift (f\circseq) n=f\circ(shift seq n) by auto
lemma Id-on-union: Id-on (A\cupB) = Id-on A UId-on B unfolding Id-on-def
by auto
lemma relpow-union-cases: }(a,d)\in(A\cupB)^~n\Longrightarrow(a,d)\in\mp@subsup{B}{}{~}n\vee (\existsbckm
(a,b)\inB^k}\wedge(b,c)\inA\wedge(c,d)\in(A\cupB)^^m^n=Suc (k+m)
proof (induct n arbitrary: a d)
    case (Suc n a e)
```

```
    let ?AB=A\cupB
    from Suc(2) obtain b where ab: (a,b) \in?AB and be: (b,e) \in?AB^n by (rule
relpow-Suc-E2)
    from ab
    show ?case
    proof
        assume (a,b) \inA
        show ?thesis
    proof (rule disjI2, intro exI conjI)
            show Suc n = Suc (0 + n) by simp
            show (a,b) \inA by fact
    qed (insert be, auto)
    next
    assume ab: (a,b)\inB
    from Suc(1)[OF be]
    show ?thesis
    proof
        assume (b,e)\inB^^n
        with ab show ?thesis
            by (intro disjI1 relpow-Suc-I2)
    next
        assume \existscdk m. (b,c)\inB^^k\wedge (c,d)\inA\wedge(d,e) \in?AB^~m^n
=Suc (k+m)
        then obtain cdkm where (b,c)\inB\leadsto~}k\mathrm{ and *: (c,d) & A (d,e) &?AB
    ~mn=Suc (k+m) by blast
        with ab have ac: (a,c)\in B^ (Suc k) by (intro relpow-Suc-I2)
        show ?thesis
            by (intro disjI2 exI conjI, rule ac, (rule *)+, simp add: *)
        qed
    qed
qed simp
lemma trans-refl-imp-rtrancl-id:
    assumes trans r refl r
    shows r* =r
proof
    show r*}\subseteq
    proof
        fix }x
        assume (x,y)\in r*
        thus (x,y)\inr
            by (induct, insert assms, unfold refl-on-def trans-def, blast+)
    qed
qed regexp
lemma trans-refl-imp-O-id:
    assumes trans r refl r
    shows r Or =r
proof(intro equalityI)
```

```
    show r Or \subseteqr by(fact trans-O-subset[OF assms(1)])
    have r\subseteqrO Id by auto
    moreover have Id \subseteqr by(fact assms(2)[unfolded refl-O-iff])
    ultimately show r\subseteqrO r by auto
qed
lemma relcomp3-I:
    assumes (t,u)\inA and (s,t)\inB and (u,v)\inB
    shows }(s,v)\inBOAO
    using assms by blast
lemma relcomp3-transI:
    assumes trans B and (t,u)\inBOAOB and (s,t)\inB and (u,v)\inB
    shows (s,v)\inBOAOB
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 rOs\subseteqsO(s\cupr)*
    shows SN (r* Os O r*)}=SN
proof -
    from converse-mono [THEN iffD2 , OF assms]
    have *: s
    have (r*}OsO\mp@subsup{r}{}{*}\mp@subsup{)}{}{-1}=(\mp@subsup{r}{}{-1}\mp@subsup{)}{}{*}O\mp@subsup{s}{}{-1}O(\mp@subsup{r}{}{-1}\mp@subsup{)}{}{*
    by (simp only: converse-relcomp O-assoc rtrancl-converse)
    with qc-wf-relto-iff [OF *]
    show ?thesis by (simp add: SN-iff-wf)
qed
lemma conversion-empty [simp]: conversion {} = Id
    by (auto simp: conversion-def)
lemma symcl-idemp [simp]:(r}\leftrightarrow)\leftrightarrow<<=r\leftrightarrow by aut
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 $::$ ' $a \mathrm{rel} \Rightarrow{ }^{\prime}$ 'a rel $\Rightarrow$ ' $a$ rel where
relto $R S \equiv S^{\star} * O R O S^{*}$

```
definition \(S N\)-rel-on :: 'a rel \(\Rightarrow{ }^{\prime}\) 'a rel \(\Rightarrow\) 'a set \(\Rightarrow\) bool where
    \(S N\)-rel-on \(R S \equiv S N\)-on (relto \(R S\) )
definition \(S N\)-rel-on-alt :: 'a rel \(\Rightarrow{ }^{\prime}\) a rel \(\Rightarrow\) 'a set \(\Rightarrow\) bool where
    SN-rel-on-alt \(R S T=(\forall f\). chain \((R \cup S) f \wedge f 0 \in T \longrightarrow \neg(I N F M j .(f j, f\)
\((S u c j)) \in R)\) )
abbreviation \(S N\)-rel :: 'a rel \(\Rightarrow\) 'a rel \(\Rightarrow\) bool where
    \(S N\)-rel \(R S \equiv S N\)-rel-on \(R\) S UNIV
abbreviation \(S N\)-rel-alt :: 'a rel \(\Rightarrow{ }^{\prime}\) 'a rel \(\Rightarrow\) bool where
    SN-rel-alt \(R S \equiv S N\)-rel-on-alt \(R S\) UNIV
lemma relto-absorb [simp]: relto \(R E O E^{*}=\) relto \(R E E^{*} O\) relto \(R E=\) relto \(R\)
E
    using \(O\)-assoc and rtrancl-idemp-self-comp by (metis) +
lemma steps-preserve-SN-on-relto:
    assumes steps: \((a, b) \in(R \cup S)^{*} *\)
        and \(S N\) : \(S N\)-on (relto \(R S\) ) \(\{a\}\)
    shows \(S N\)-on (relto \(R S\) ) \(\{b\}\)
proof -
    let ? \(R S=\) relto \(R S\)
    have \((R \cup S)^{\wedge} * \subseteq S^{\wedge} * \cup ? R S^{\wedge} *\) by regexp
    with steps have \((a, b) \in S^{\wedge} * \vee(a, b) \in ? R S^{\wedge} *\) by auto
    thus ?thesis
    proof
        assume \((a, b) \in ? R S^{\wedge} *\)
        from steps-preserve-SN-on[OF this \(S N]\) show ?thesis .
    next
        assume Ssteps: \((a, b) \in S^{*}\)
        show ?thesis
        proof
            fix \(f\)
            assume \(f 0 \in\{b\}\) and chain ? RS \(f\)
            hence f0: \(f 0=b\) and steps: \(\bigwedge i .(f i, f(S u c i)) \in ? R S\) by auto
            let \(? g=\lambda\) i. if \(i=0\) then a else \(f i\)
            have \(\neg S N\)-on ? \(R S\{a\}\) unfolding \(S N\)-on-def not-not
            proof (rule exI[of - ?g], intro conjI allI)
                fix \(i\)
                show \((? g i, ? g(\) Suc \(i)) \in ? R S\)
                    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: \((a, f(S u c ~ 0)) \in S^{\wedge} O\)
```

?RS by blast

```
                have (a,f (Suc 0)) \in?RS
                    by (rule subsetD[OF - steps], regexp)
                thus ?thesis unfolding 0 by simp
            qed
            qed simp
            with SN show False by simp
    qed
    qed
qed
lemma step-preserves-SN-on-relto: assumes st: (s,t)\inR\cupE
    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\LongrightarrowSN-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\cupS) f^f0\inT
    with SN have SN: SN-on (relto R S) {f 0}
            and steps: \ i. (f i,f(Suc i)) \inR\cupS unfolding SN-defs by auto
    obtain r where r:\bigwedge j.rj\equiv (fj,f(Suc j)) \inR by auto
    show }\neg(INFM j.(fj,f(Suc j))\inR
    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] in-
finitely-many.index-not-p-between[OF ih]
            have r-index: ^i.r (r-index i)^r-index i<r-index (Suc i) ^(\forall j.r-index
i<j^j<r-index (Suc i)\longrightarrow\negrj) by auto
            obtain g where g: \bigwedgei.gi\equivf(r-index i)..
            {
            fix }
            let ?ri = r-index i
            let ?rsi = r-index (Suc i)
            from r-index have isi:?ri < ?rsi by auto
            obtain ri rsi where ri:ri=?ri and rsi: rsi=?rsi by auto
            with r-index[of i] steps have inter: \ j.ri<j^j<rsi\Longrightarrow(fj,f (Suc
j)) \inS unfolding r by auto
            from ri isi rsi have risi: ri<rsi by simp
            {
                fix n
                assume Suc n \leqrsi - ri
                hence (f (Suc ri), f(Suc ( n + ri))) \in S`*
                proof (induct n, simp)
```

```
                case (Suc n)
                    hence stepps: (f (Suc ri),f(Suc (n+ri))) \in S`* by simp
                    have (f (Suc (n+ri)), f(Suc (Suc n + ri))) \inS
                    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) \in S`* by simp
        with ri rsi have ssteps: (f (Suc ?ri), f ?rsi) \in S`* by simp
        with r-index[of i] have ( f ?ri,f ?rsi) \inR O S * unfolding r by auto
        hence (gi,g(Suc i)) \in S`* OROS`* using rtrancl-refl unfolding g by
auto
    }
    hence nSN: \negSN-on (S`* O R O S`*) {g 0} unfolding SN-defs by blast
    have SN: SN-on (S**OROS*) {f (r-index 0)}
    proof (rule steps-preserve-SN-on-relto[OF - SN])
            show (f 0, f (r-index 0)) \in(R\cupS) **
            unfolding rtrancl-fun-conv
            by (rule exI[of-f], rule exI[of-r-index 0], insert steps, auto)
        qed
        with nSN show False unfolding g ..
        qed
    qed
qed
lemma SN-rel-on-alt-imp-SN-rel-on: SN-rel-on-alt R ST\LongrightarrowSN-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: f0\inT and chain (relto R S) f
    hence steps: \ i. (f i,f(Suc i)) \in S`* OR O S`* by auto
    let ?prop = \lambda i ai bi. (f i,bi) \in S`* ^ (bi, ai) \inR^(ai,f(Suc (i))) \in S`*
    {
        fix }
        from steps obtain bi ai where ?prop i ai bi by blast
        hence }\exists\mathrm{ ai bi. ?prop i ai bi by blast
    }
    hence }\foralli.\exists\mathrm{ bi ai. ?prop i ai bi by blast
    from choice[OF this] obtain b where }\foralli.\exists\mathrm{ 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) \inS`*O R by auto
    let ?prop = \lambda ili. (b i, a i) \inR\wedge(\forallj< length li.((a i# li)!j, (a i# # li)!
Suc j) \inS) ^ last (a i# li) =b (Suc i)
    {
        fix }
        from steps[of i] steps[of Suc i] have (a i,f(Suc i)) \in S`* and (f (Suc i),b
```

$($ Suc $i)) \in S^{\wedge}$ * by auto
from rtrancl-trans $[$ OF this $]$ steps $[o f i]$ have $R:(b i, a i) \in R$ and $S:(a i, b$ $(S u c i)) \in S^{\wedge} *$ by blast +
from $S$ [unfolded rtrancl-list-conv] obtain $l i$ where last ( $a i \# l i$ ) $=b$ (Suc
i) $\wedge(\forall j<$ length li. $((a i \# l i)!j,(a i \# l i)!S u c j) \in S) .$.
with $R$ have ?prop $i l i$ by blast
hence $\exists$ li. ?prop ili..
\}
hence $\forall i$. $\exists$ li. ?prop $i l i$..
from choice $[O F$ this] obtain $l$ where steps: $\bigwedge i$. ?prop $i(l i)$ by auto
let ?p $=\lambda i$. ?prop $i(l i)$
from steps have steps: $\bigwedge i$. ?p $i$ by blast
let $? l=\lambda i . a i \# l i$
let ? $l^{\prime}=\lambda i$. length $(? l i)$
let ? $g=\lambda i$. inf-concat-simple ? $l^{\prime} i$
obtain $g$ where $g: \bigwedge i . g i=($ let $(i i, j j)=? g i$ in ?l $i i!j j)$ by auto
have $g 0: g 0=a 0$ unfolding $g$ Let-def by simp
with $f a 0$ have $f g 0:(f 0, g 0) \in S^{\wedge} O R$ by auto
have $f g 0:(f 0, g 0) \in(R \cup S){ }^{*}$
by (rule subsetD[OF - fg0], regexp)
have len: $\wedge i j n$. ?g $n=(i, j) \Longrightarrow j<$ length (?l $i)$
proof -
fix $i j n$
assume $n$ : ? $g n=(i, j)$
show $j<$ length (?l $i$ )
proof (cases $n$ )
case 0
with $n$ have $j=0$ by auto
thus ?thesis by simp
next case (Suc nn)
obtain $i i j j$ where $n n$ : ?g $n n=(i i, j j)$ by (cases ?g nn, auto)
show ?thesis
proof (cases Suc jj < length (?l ii))
case True
with $n n$ Suc have ? $g n=(i i, S u c j j)$ by auto
with $n$ True show ?thesis by simp
next
case False
with $n n$ Suc have ?g $n=(S u c i i, 0)$ by auto
with $n$ show ?thesis by simp
qed
qed
qed
have gsteps: $\bigwedge i .(g i, g($ Suc $i)) \in R \cup S$
proof -
fix $n$
obtain $i j$ where $n$ : ? $g=(i, j)$ by (cases ? $g n$, auto)
show $(g n, g(S u c n)) \in R \cup S$

```
    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 \mp@code { b y ~ a u t o }
    thus ?thesis using steps[of i] True by auto
    next
        case False
        with n have ?g (Suc n)=(Suc i, 0) by auto
        with n have gn:g n = ?l i ! j and gsn: g (Suc n)=a (Suc i) unfolding
g \mp@code { b y ~ a u t o }
    from gn len[OF n] False have j = length (?l i) - 1 by auto
    with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
    from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
    qed
    qed
    have infR: INFM j. (g j,g (Suc j)) \inR unfolding INFM-nat-le
    proof
        fix n
        obtain ij where n: ?g n=(i,j) by (cases ?g n, auto)
        from len[OF n] have j:j<?l' i .
        let ?k = ?l' i - 1 - j
        obtain k}\mathrm{ where k: k=j+? ? by auto
        from jk have k2: k=?l' i-1 and k3: j + ?k< < l' i by auto
        from inf-concat-simple-add[OF n, of ?k, OF k3]
        have gnk:?g ( n + ?k) = (i,k) by (simp only:k)
        hence g(n+?k)=?l i!k unfolding g by auto
        hence gnk2: g ( n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
        from k2 gnk have ?g (Suc (n+?k)) = (Suc i,0) by auto
    hence gnsk2: g (Suc (n+?k))=a (Suc i) unfolding g by auto
    from steps[of i] steps[of Suc i] have main: (g (n+?k),g(Suc (n+?k)))\inR
        by (simp only: gnk2 gnsk2)
    show \exists j\geqn.(gj,g(Suc j)) \inR
        by (rule exI[of-n+?k], auto simp: main[simplified])
    qed
    from fg0[unfolded rtrancl-fun-conv] obtain gg n where start: gg 0 = f0
    and n:gg n=g0 and steps: \bigwedge i. i<n\Longrightarrow(gg i,gg (Suc i)) \inR\cupS by
auto
    let ?h=\lambda i. if i<n then gg i else g (i-n)
    obtain h}\mathrm{ where h: h=?h by auto
    {
        fix }
        assume i:i\leqn
        have hi=ggi using i unfolding h
            by (cases i<n, auto simp: n)
    } note gg= this
    from gg[of 0] <f 0\inT\rangle have h0:h 0\inT unfolding start by auto
    {
        fix }
```

```
        have (h i,h(Suc i)) \inR\cupS
        proof (cases i<n)
            case True
            from steps[of i] gg[of i] gg[of Suc i] True show ?thesis by auto
            next
            case False
            hence i=n+(i-n) by auto
            then obtain }k\mathrm{ where }i:i=n+k\mathrm{ by auto
            from gsteps[of k] show ?thesis unfolding hi by simp
        qed
    } note hsteps=this
    from SN[unfolded SN-rel-on-alt-def, rule-format, OF conjI[OF allI[OF hsteps]
h0]]
    have }\neg(INFM j.(hj,h(Suc j))\inR)
    moreover have INFM j.(hj,h (Suc j)) \inR unfolding INFM-nat-le
    proof (rule)
        fix m
        from infR[unfolded INFM-nat-le, rule-format, of m]
        obtain i where i:i\geqm and g:(gi,g(Suc i)) \inR by auto
        show \exists n\geqm.(hn,h(Suc n)) \inR
        by (rule exI[of - i + n], unfold h, insert g i, auto)
    qed
    ultimately show False ..
    qed
qed
lemma SN-rel-on-conv: SN-rel-on =SN-rel-on-alt
    by (intro ext) (blast intro: SN-rel-on-imp-SN-rel-on-alt SN-rel-on-alt-imp-SN-rel-on)
lemmas SN-rel-defs = SN-rel-on-def SN-rel-on-alt-def
lemma SN-rel-on-alt-r-empty : SN-rel-on-alt {} S T
    unfolding SN-rel-defs by auto
lemma SN-rel-on-alt-s-empty : SN-rel-on-alt R {} =SN-on R
    by (intro ext, unfold SN-rel-defs SN-defs, auto)
lemma SN-rel-on-mono':
    assumes R:R\subseteq\mp@subsup{R}{}{\prime}\mathrm{ and S:S}\subseteq\mp@subsup{R}{}{\prime}\cup\mp@subsup{S}{}{\prime}\mathrm{ and SN:SN-rel-on R R' S'T}
    shows SN-rel-on R ST
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\cupS)f^f0\inT
    with RS have chain ( }\mp@subsup{R}{}{\prime}\cup\mp@subsup{S}{}{\prime})f\wedgef0\inT\mathrm{ by auto
    from SN[unfolded conv, rule-format, OF this]
```

```
        show }\neg(\forallm.\existsn\geqm.(fn,f(Suc n))\inR) using R by aut
    qed
qed
lemma relto-mono:
    assumes R\subseteq\mp@subsup{R}{}{\prime}\mathrm{ and S}\subseteq\mp@subsup{S}{}{\prime}
    shows relto R S\subseteqrelto R' S'
    using assms rtrancl-mono by blast
lemma SN-rel-on-mono:
    assumes R:R\subseteq\mp@subsup{R}{}{\prime}\mathrm{ and S:S}\subseteq\mp@subsup{S}{}{\prime}
    and SN: SN-rel-on R' S'T
    shows SN-rel-on R S T
    using SN
    unfolding SN-rel-on-def using SN-on-mono[OF - relto-mono[OF R S]] by blast
lemmas SN-rel-on-alt-mono = SN-rel-on-mono[unfolded SN-rel-on-conv]
lemma SN-rel-on-imp-SN-on:
    assumes SN-rel-on R S T shows SN-on R T
proof
    fix f
    assume chain Rf
    and f0: f0\inT
    hence }\i.(fi,f(Suc i))\in relto R S by blas
    thus False using assms f0 unfolding SN-rel-on-def SN-defs by blast
qed
lemma relto-Id: relto R(S\cupId)= relto R S by simp
lemma SN-rel-on-Id:
    shows SN-rel-on R (S\cupId) T=SN-rel-on R ST
    unfolding SN-rel-on-def by (simp only: relto-Id)
lemma SN-rel-on-empty[simp]:SN-rel-on R {} T = SN-on R T
    unfolding SN-rel-on-def by auto
lemma SN-rel-on-ideriv: SN-rel-on R S T = (\neg(\exists as.ideriv R S as ^ as 0 \inT))
(is ?L = ?R)
proof
    assume ?L
    show ?R
    proof
        assume \exists as. ideriv R S as ^ as 0 \inT
        then obtain as where id: ideriv R S as and T: as 0 \inT by auto
        note id = id[unfolded ideriv-def]
        from〈?L〉[unfolded SN-rel-on-conv SN-rel-on-alt-def,THEN spec[of - as]]
            id T obtain i where i: \bigwedgej.j\geqi\Longrightarrow(as j, as (Suc j))\not\inR 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 $S N$-rel-on-conv $S N$-rel-on-alt-def
proof (intro allI impI)
fix $a s$
assume chain $(R \cup S)$ as $\wedge$ as $0 \in T$
with $\langle ? R\rangle[$ unfolded ideriv-def] have $\neg($ INFM $i$. (as i, as $($ Suc $i)) \in R)$ by
auto
from this[unfolded INFM-nat] obtain $i$ where $i: \bigwedge j . i<j \Longrightarrow$ (as $j$, as (Suc
j)) $\notin R$ by auto show $\neg(I N F M j$. (as $j$, as $(S u c j)) \in R)$ unfolding INFM-nat using $i$ by blast
qed
qed
lemma $S N$-rel-to-SN-rel-alt: $S N$-rel $R S \Longrightarrow S N$-rel-alt $R S$
proof (unfold $S N$-rel-on-def)
assume $S N$ : $S N$ (relto $R S$ )
show ?thesis
proof (unfold $S N$-rel-on-alt-def, intro allI impI) fix $f$
presume steps: chain $(R \cup S) f$
obtain $r$ where $r: \bigwedge j . r j \equiv(f j, f(S u c j)) \in R$ by auto
show $\neg(I N F M j$. $(f j, f(S u c j)) \in R)$
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] in-finitely-many.index-not-p-between[OF ih]
have r-index: $\wedge i . r(r$-index $i) \wedge r$-index $i<r$-index $(S u c i) \wedge(\forall j . r$-index $i<j \wedge j<r$-index (Suc $i) \longrightarrow \neg r j$ ) by auto
obtain $g$ where $g: \wedge i . g i \equiv f(r$-index $i) .$.
\{
fix $i$
let ?ri $=r$-index $i$
let ?rsi $=r$-index $($ Suc $i)$
from $r$-index have isi: ? ri $<$ ? rsi by auto
obtain ri rsi where $r i: r i=$ ? $r i$ and $r s i: r s i=$ ? $r s i$ by auto
with r-index[of $i]$ steps have inter: $\wedge j . r i<j \wedge j<r s i \Longrightarrow(f j, f$ (Suc
$j)) \in S$ unfolding $r$ by auto
from ri isi rsi have risi: ri<rsi by simp
\{
fix $n$
assume Suc $n \leq r s i-r i$
hence $(f$ (Suc ri), $f($ Suc $(n+r i))) \in S^{*}$

```
            proof (induct n, simp)
                    case (Suc n)
                    hence stepps:(f(Suc ri),f(Suc (n+ri))) \inS`* by simp
                    have (f(Suc (n+ri)), f(Suc (Suc n + ri))) \inS
                    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) \in S`* by simp
        with ri rsi have ssteps: (f (Suc ?ri), f ?rsi) \in S** by simp
        with r-index[of i] have ( f ?ri, f ?rsi) \inR O S * unfolding r by auto
        hence (gi,g(Suc i)) \in S *}OROS`* using rtrancl-refl unfolding g by
auto
        }
        hence }\negSN(S`*OROS`*) unfolding SN-defs by blas
        with SN show False by simp
        qed
    qed simp
qed
lemma SN-rel-alt-to-SN-rel : SN-rel-alt R S \LongrightarrowSN-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: \i. (fi,f(Suc i)) \inS`* OR O S`* by auto
    let ?prop = \lambda i ai bi. (f i,bi) \in S`* ^(bi,ai) \inR^(ai,f(Suc (i))) \inS`*
    {
        fix }
        from steps obtain bi ai where ?prop i ai bi by blast
        hence \exists ai bi. ?prop i ai bi by blast
    }
    hence }\forall i. \exists bi ai. ?prop i ai bi by blas
    from choice[OF this] obtain b where \forall i. \exists ai. ?prop i ai (b i) by blast
    from choice[OF this] obtain a where steps: \bigwedge i. ?prop i (a i) (b i) by blast
    let ?prop = \lambda ili. (b i, a i) \inR\wedge(\forallj< length li. ((a i# # li)! j, (ai# # li)!
Suc j) \inS)^last (a i#li)=b(Suc i)
    {
        fix }
        from steps[of i] steps[of Suc i] have (a i,f(Suc i)) \in S** and (f (Suc i),b
(Suc i)) \in S * * by auto
    from rtrancl-trans[OF this] steps[of i] have R:(b i,a i)\inR and S:(a i,b
(Suc i)) \inS S* by blast+
    from S[unfolded rtrancl-list-conv] obtain li where last (a i# li) =b (Suc
i)}\wedge(\forallj<length li. ((a i # li)!j, (a i # li)!Suc j) \inS) ..
    with R have ?prop i li by blast
```

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

```
g \mp@code { b y ~ a u t o }
            from gn len[OF n] False have j= length (?l i) - 1 by auto
            with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
            from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
        qed
    qed
    have infR: INFM j. (g j,g (Suc j)) \inR unfolding INFM-nat-le
    proof
        fix n
        obtain ij where n: ?g n = (i,j) by (cases ?g n, auto)
        from len[OF n] have j:j<?l' }i\mathrm{ .
        let ?k = ?l' i - 1 - j
        obtain k}\mathrm{ where k: k=j+? }k\mathrm{ by auto
        from jk have k2: k=?l' i-1 and k3: j+? k < ?l' i by auto
        from inf-concat-simple-add[OF n, of ?k, OF k3]
        have gnk:?g ( n + ?k) = (i,k) by (simp only:k)
        hence g(n+?k)=?l i!k unfolding g by auto
        hence gnk2: g ( n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
        from k2 gnk have ?g (Suc (n+?k))=(Suc i,0) by auto
        hence gnsk2: g (Suc (n+?k)) =a (Suc i) unfolding g by auto
        from steps[of i] steps[of Suc i] have main: (g (n+?k),g(Suc (n+?k)))\inR
            by (simp only: gnk2 gnsk2)
        show }\existsj\geqn.(gj,g(Suc j))\in
            by (rule exI[of-n+?k], auto simp: main[simplified])
        qed
        from SN[unfolded SN-rel-on-alt-def] gsteps infR show False by blast
    qed
qed
lemma SN-rel-alt-r-empty : SN-rel-alt {} S
    unfolding SN-rel-defs by auto
lemma SN-rel-alt-s-empty : SN-rel-alt R {} = SN R
    unfolding SN-rel-defs SN-defs by auto
lemma SN-rel-mono':
    R\subseteq\mp@subsup{R}{}{\prime}\LongrightarrowS\subseteq\mp@subsup{R}{}{\prime}\cup\mp@subsup{S}{}{\prime}\LongrightarrowSN-rel R' S'\LongrightarrowSN-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\mp@subsup{R}{}{\prime}\mathrm{ and S:S}\subseteq\mp@subsup{S}{}{\prime}\mathrm{ and SN:SN-rel R' S'}
    shows SN-rel R S
    using SN unfolding SN-rel-defs using SN-subset[OF - relto-mono[OF R S]] by
blast
lemmas SN-rel-alt-mono =SN-rel-mono[unfolded SN-rel-on-conv]
lemma SN-rel-imp-SN:assumes SN-rel R S shows SN R
```


## proof

fix $f$
assume $\forall i .(f i, f($ Suc $i)) \in R$
hence $\wedge i$ ．$(f i, f(S u c i)) \in$ relto $R S$ by blast
thus False using assms unfolding $S N$－rel－defs $S N$－defs by fast
qed
lemma relto－trancl－conv ：（relto $R S) \uparrow+=((R \cup S)) \uparrow O R O((R \cup S)) \widehat{*}$ by regexp
lemma $S N$－rel－Id：
shows $S N$－rel $R(S \cup I d)=S N$－rel $R S$
unfolding $S N$－rel－defs by（simp only：relto－Id）
lemma relto－rtrancl：relto $R\left(S^{*}\right)=$ relto $R S$ by regexp
lemma $S N$－rel－empty［simp］：SN－rel $R\}=S N R$
unfolding $S N$－rel－defs by auto
lemma $S N$－rel－ideriv：$S N$－rel $R S=(\neg(\exists$ as．ideriv $R S$ as $))($ is ？$L=? R)$
proof
assume ？$L$
show？$R$
proof
assume $\exists$ as．ideriv $R S$ as
then obtain as where id：ideriv $R S$ as by auto
note $i d=i d[$ unfolded $i d e r i v-d e f]$
from 〈？L〉［unfolded SN－rel－on－conv SN－rel－defs，THEN spec［of－as］］
id obtain $i$ where $i: \bigwedge j . j \geq i \Longrightarrow($ as $j$ ，as $(S u c j)) \notin R$ by auto with id［unfolded INFM－nat，THEN conjunct2，THEN spec $[$ of－Suc i $i]$ show
False by auto
qed
next
assume ？$R$
show ？L
unfolding $S N$－rel－on－conv $S N$－rel－defs
proof（intro allI impI）
fix as
presume chain $(R \cup S)$ as
with «？R〉［unfolded ideriv－def］have $\neg($ INFM i．（as i，as $($ Suc $i)) \in R)$ by auto
from this［unfolded INFM－nat］obtain $i$ where $i: \bigwedge j . i<j \Longrightarrow$（as $j$ ，as（Suc j））$\notin R$ by auto
show $\neg(I N F M j$ ．（as $j$ ，as（Suc $j)) \in R$ ）unfolding INFM－nat using $i$ by blast
qed $\operatorname{simp}$
qed
lemma $S N$－rel－map：

```
    fixes R Rw R' Rw' :: 'a rel
    defines A:A\equivR'\cupR\mp@subsup{w}{}{\prime}
    assumes SN:SN-rel R'Rw'
    and R: \bigwedgest. (s,t) \inR\Longrightarrow(fs,ft)\inA`*O R'O A`*
    and Rw: \st. (s,t)\inRw\Longrightarrow(fs,ft) \in A`*
    shows SN-rel R Rw
    unfolding SN-rel-defs
proof
    fix g
    assume steps: chain (relto R Rw) g
    let ?f = \lambdai.(f (g i))
    obtain h where h:h=?f by auto
    {
    fix }
    let ?m = \lambda (x,y). (f x, f y)
    {
        fix st
        assume (s,t) \inRw`*
        hence ?m}(s,t)\in\widehat{A*
        proof (induct)
            case base show ?case by simp
        next
                case (step t u)
                from Rw[OF step(2)] step(3)
                show ?case by auto
        qed
    } note Rw = this
    from steps have (g i,g (Suc i)) \in relto R Rw ..
    from this
    obtain st where gs: (gi,s)\inRw`* and st: (s,t) \inR and tg:(t,g (Suc i))
\inRw`* by auto
    from Rw[OF gs] R[OF st] Rw[OF tg]
    have step:(?f i, ?f (Suc i)) \in A`* O(A`*O R'OA`*)OA`*
        by fast
    have (?f i, ?f (Suc i)) \in A`* O R'OA`*
        by (rule subsetD[OF - step], regexp)
    hence (h i,h(Suc i)) \in(relto R'Rw')^+
        unfolding A h relto-trancl-conv.
    }
    hence }\negSN((\mathrm{ relto R}\mp@subsup{R}{}{\prime}R\mp@subsup{w}{}{\prime}\mp@subsup{)}{}{`}+) by aut
    with SN-imp-SN-trancl[OF SN[unfolded SN-rel-on-def]]
    show False by simp
qed
datatype SN-rel-ext-type =top-s |top-ns | normal-s | normal-ns
fun SN-rel-ext-step :: 'a rel }=>\mp@subsup{|}{}{\prime}a\textrm{rel}=>\mp@subsup{}{}{\prime}'a rel => ''a rel => SN-rel-ext-type = 'a rel
where
    SN-rel-ext-step P Pw R Rw top-s = P
```

definition $S N$-rel-ext :: 'a rel $\Rightarrow$ 'a rel $\Rightarrow{ }^{\prime}$ 'a rel $\Rightarrow$ 'a rel $\Rightarrow\left({ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow$ bool where
$S N$-rel-ext $P$ Pw R Rw $M \equiv(\neg(\exists f t$.
$(\forall i .(f i, f(S u c i)) \in S N$-rel-ext-step $P P w R R w(t i))$
$\wedge(\forall i . M(f i))$
$\wedge($ INFM i. t $i \in\{$ top-s,top-ns $\})$
$\wedge($ INFM i. $t i \in\{$ top-s,normal-s $\})))$
lemma $S N$-rel-ext-step-mono: assumes $P \subseteq P^{\prime} P w \subseteq P w^{\prime} R \subseteq R^{\prime} R w \subseteq R w^{\prime}$ shows $S N$-rel-ext-step $P$ Pw R $R w t \subseteq S N$-rel-ext-step $P^{\prime} P w^{\prime} R^{\prime} R w^{\prime} t$
using assms
by (cases t, auto)
lemma $S N$-rel-ext-mono: assumes subset: $P \subseteq P^{\prime} P w \subseteq P w^{\prime} R \subseteq R^{\prime} R w \subseteq R w^{\prime}$ and
$S N$ : SN-rel-ext $P^{\prime} P w^{\prime} R^{\prime} R w^{\prime} M$ shows $S N$-rel-ext $P$ Pw R Rw M using $S N$-rel-ext-step-mono[OF subset] $S N$ unfolding $S N$-rel-ext-def by blast
lemma $S N$-rel-ext-trans:
fixes $P P w R R w::$ 'a rel and $M$ :: ' $a \Rightarrow$ bool
defines $M^{\prime}: M^{\prime} \equiv\{(s, t) . M t\}$
defines $A: A \equiv(P \cup P w \cup R \cup R w) \cap M^{\prime}$
assumes $S N$-rel-ext $P P w R$ Rw $M$
shows $S N$-rel-ext $\left(A \mathcal{*}^{*} O\left(P \cap M^{\prime}\right) O A^{\wedge}\right)\left(A^{*} O\left((P \cup P w) \cap M^{\prime}\right) O A^{*}\right)$
$\left(A{ }^{*} O\left((P \cup R) \cap M^{\prime}\right) O A^{*}\right)\left(A^{*}\right) M$ (is $S N$-rel-ext ?P ?Pw? ? ?Rw M)
proof (rule ccontr)
let ?relt $=S N$-rel-ext-step ?P ?Pw ?R ?Rw
let ?rel $=S N$-rel-ext-step $P P w R R w$
assume $\neg$ ?thesis
from this[unfolded $S N$-rel-ext-def]
obtain $f$ ty
where steps: $\bigwedge i .(f i, f($ Suc $i)) \in$ ?relt $(t y i)$
and $\min : \bigwedge i . M(f i)$
and inf1: INFM i. ty $i \in\{$ top-s, top-ns $\}$
and inf2: INFM i. ty $i \in\{$ top-s, normal-s $\}$
by auto
let ?Un $=\lambda t t$. U (?rel'tt)
let ? $U n M=\lambda t t$. $(U($ ?rel ' $t t)) \cap M^{\prime}$
let ? $A=$ ?UnM $\{$ top-s,top-ns,normal-s,normal-ns $\}$
let $? P^{\prime}=? U n M\{t o p-s\}$
let $? P w^{\prime}=$ ? UnM $\{$ top-s,top-ns $\}$
let ? $R^{\prime}=$ ? UnM $\{$ top- $s$, normal-s $\}$
let $? R w^{\prime}=?$ UnM $\{$ top-s,top-ns,normal-s,normal-ns $\}$
have $A: A=$ ? $A$ unfolding $A$ by auto

```
have \(P:\left(P \cap M^{\prime}\right)=? P^{\prime}\) by auto
have \(P w:(P \cup P w) \cap M^{\prime}=? P w^{\prime}\) by auto
have \(R:(P \cup R) \cap M^{\prime}=? R^{\prime}\) by auto
have \(R w: A=\) ? \(R w^{\prime}\) unfolding \(A\)..
\{
    fix \(s t t\)
    assume \(m: M s\) and \(s t:(s, t) \in ? U n M t t\)
    hence \(\exists\) typ \(\in t t\). \((s, t) \in\) ? rel typ \(\wedge M s \wedge M t\) unfolding \(M^{\prime}\) by auto
    \} note one-step \(=\) this
    let ? seq \(=\lambda\) stgnty. \(s=g 0 \wedge t=g n \wedge(\forall i<n .(g i, g(\) Suc \(i)) \in\) ? rel \((t y\)
i)) \(\wedge(\forall i \leq n . M(g i))\)
    \{
    fix \(s t\)
    assume \(m: M s\) and \(s t:(s, t) \in A^{*}\)
    from st[unfolded rtrancl-fun-conv]
    obtain \(g n\) where \(g 0: g 0=s\) and \(g n: g n=t\) and steps: \(\bigwedge i . i<n \Longrightarrow(g\)
\(i, g(\) Suc \(i)) \in\) ? \(A\) unfolding \(A\) by auto
    \{
        fix \(i\)
        assume \(i \leq n\)
        have \(M(g i)\)
        proof (cases i)
            case 0
            show ?thesis unfolding 0 g 0 by (rule \(m\) )
        next
            case (Suc j)
            with \(\langle i \leq n\rangle\) have \(j<n\) by auto
            from steps \(\left[O F\right.\) this show ?thesis unfolding Suc \(M^{\prime}\) by auto
        qed
    \(\}\) note \(\min =t h i s\)
    \{
        fix \(i\)
        assume \(i: i<n\) hence \(i^{\prime}: i \leq n\) by auto
        from \(i^{\prime}\) one-step \([\) OF min steps \([\) OF \(i]]\)
        have \(\exists\) ty. \((g i, g(S u c i)) \in\) ? rel ty by blast
    \}
    hence \(\forall i .(\exists\) ty. \(i<n \longrightarrow(g i, g(S u c i)) \in\) ? rel ty \()\) by auto
    from choice [OF this]
    obtain \(t t\) where steps: \(\bigwedge i . i<n \Longrightarrow(g i, g(S u c i)) \in ?\) rel \((t t i)\) by auto
    from \(g 0\) gn steps min
    have ? seq stg n tt by auto
    hence \(\exists g n t t\). ?seq st \(g n t t\) by blast
    \(\}\) note \(A\)-steps \(=\) this
    let ?seqtt \(=\lambda s t\) tt \(g n\) ty. \(s=g 0 \wedge t=g n \wedge n>0 \wedge(\forall i<n .(g i, g\) (Suc
\(i)) \in\) ? rel \((t y i)) \wedge(\forall i \leq n . M(g i)) \wedge(\exists i<n . t y i \in t t)\)
    \{
        fix \(s t t\)
        assume \(m: M s\) and \(s t:(s, t) \in A^{*} * O\) ? UnM tt \(O A^{*}\)
        then obtain \(u v\) where \(s u:(s, u) \in A^{*}\) and \(u v:(u, v) \in ? U n M\) tt and \(v t\) :
```

```
(v,t)\inA`*
        by auto
    from A-steps[OF m su] obtain g1 n1 ty1 where seq1: ?seq s u g1 n1 ty1 by
auto
    from uv have Mv unfolding M' by auto
    from A-steps[OF this vt] obtain g2 n2 ty2 where seq2: ?seq v t g2 n2 ty2 by
auto
    from seq1 have Mu by auto
    from one-step[OF this uv] obtain ty where ty: ty ftt and uv: (u,v)\in?rel
ty by auto
    let ?g=\lambda i. if i\leqn1 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-(\mathrm{ Suc n1))
    let ? n = Suc (n1 + n2)
    have ex:\existsi<?n. ?ty i\intt
        by (rule exI[of - n1], simp add: ty)
    have steps: }\foralli<?n.(?g i, ?g (Suc i)) \in ?rel (?ty i
    proof (intro allI impI)
            fix }
            assume i<?n
            show (?g i,?g (Suc i)) \in?rel (?ty i)
            proof (cases i\leqn1)
                case True
                with seq1 seq2 uv show ?thesis by auto
            next
                case False
                hence i=Suc n1 + (i-Suc n1) by auto
                then obtain k where i:i=Suc n1 + k by auto
                with \langlei< ?n\rangle have k< n2 by auto
                thus ?thesis using seq2 unfolding i by auto
            qed
    qed
    from steps seq1 seq2 ex
    have seq: ?seqtt st tt ?g ?n ?ty by auto
    have }\exists\textrm{g}n\mathrm{ ty. ?seqtt s t tt g n ty
            by (intro exI, rule seq)
    } note }A-tt-A=thi
    let ?tycon = \lambda ty1 ty2 tt ty' n. ty1 = ty2 \longrightarrow( }\mp@subsup{\exists}{}{\prime
    let ?seqt = \lambda ity g nty'.fi=g 0^f(Suc i)=gn\wedge (\forallj<n.(gj,g (Suc
j)) \in? ?rel (ty'j))^(\forallj\leqn.M (g j))
                    \wedge(?tycon (ty i) top-s {top-s} ty' n)
                    \wedge(?tycon (ty i) top-ns {top-s,top-ns} ty' n)
                        ^(?tycon (ty i) normal-s {top-s,normal-s} ty' n)
{
fix }
    have \existsg n ty'. ?seqt i ty g n ty'
    proof (cases ty i)
        case top-s
        from steps[of i, unfolded top-s]
        have (f i,f (Suc i)) \in?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 (fi,f(Suc i)) \in?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 (fi,f(Suc i)) \in?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 (fi,f(Suc i)) \in?Rw by auto
        from A-steps[OF min this]
        show ?thesis unfolding normal-ns by auto
    qed
}
hence }\foralli.\existsgnty'. ?seqt ity gnty' by aut
from choice[OF this] obtain g}\mathrm{ where }\foralli.\existsn ty'. ?seqt ity (g i) n ty' by aut
from choice[OF this] obtain n where }\foralli.\existst\mp@subsup{y}{}{\prime}\mathrm{ . ?seqt ity (gi) (n i)ty' by
auto
    from choice[OF this] obtain ty' where }\forall\mathrm{ i. ?seqt ity (g i) (n i) (ty' i) by auto
    hence partial: \bigwedge i. ?seqt i ty (gi) (ni) (ty' i) ..
    let ?ind = inf-concat n
    let ?g=\lambdak. (\lambda (i,j).gij)(?ind k)
    let ?ty=\lambdak. (\lambda (i,j).ty' i j)(?ind k)
    have inf:INFM i.0<ni
        unfolding INFM-nat-le
    proof (intro allI)
    fix m
    from inf1[unfolded INFM-nat-le]
    obtain k where k:k\geqm and ty: ty k\in{top-s, top-ns} by auto
    show \exists k\geqm.0<nk
    proof (intro exI conjI, rule k)
        from partial[of k] ty show 0<nk by (cases n k,auto)
    qed
qed
note bounds = inf-concat-bounds[OF inf]
note inf-Suc = inf-concat-Suc[OF inf]
note inf-mono = inf-concat-mono[OF inf]
have }\neg\mathrm{ SN-rel-ext P Pw R RwM
    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 $i k$ : ? ind $k=(i, j)$ by force
from bounds [OF this] have $j: j<n i$ by auto
show $M(? g k)$ unfolding $i k$ using partial $[o f i] j$ by auto
next
fix $k$
obtain $i j$ where $i k$ : ?ind $k=(i, j)$ by force
from bounds $[O F$ this] have $j: j<n i$ by auto
from partial[of $i] j$ have step: $(g i j, g i(S u c j)) \in$ ? rel $\left(t y^{\prime} i j\right)$ by auto
obtain $i^{\prime} j^{\prime}$ where isk: ?ind (Suc $k$ ) $=\left(i^{\prime}, j^{\prime}\right)$ by force
have $i^{\prime} j^{\prime}: g i^{\prime} j^{\prime}=g i(S u c j)$
proof (rule inf-Suc[OF -ik isk])
fix $i$
from partial $[$ of $i]$
have $g i(n i)=f(S u c i)$ by simp
also have $\ldots=g$ (Suc $i) 0$ using partial[ of Suc $i]$ by simp
finally show $g i(n i)=g(S u c i) 0$.
qed
show $(? g k, ? g($ Suc $k)) \in ?$ rel (?ty $k)$
unfolding $i k$ isk split $i^{\prime} j^{\prime}$
by (rule step)
next
show INFM i. ?ty $i \in\{$ top-s, top-ns $\}$
unfolding INFM-nat-le
proof (intro allI)
fix $k$
obtain $i j$ where $i k$ : ? ind $k=(i, j)$ by force
from inf1[unfolded INFM-nat] obtain $i^{\prime}$ where $i^{\prime}: i^{\prime}>i$ and ty: ty $i^{\prime} \in$ \{top-s, top-ns $\}$ by auto
from partial $\left[\right.$ of $i \boldsymbol{\eta}$ ty obtain $j^{\prime}$ where $j^{\prime}: j^{\prime}<n i^{\prime}$ and $t y^{\prime}: t y^{\prime} i^{\prime} j^{\prime} \in\{$ top-s, top-ns $\}$ by auto
from inf-concat-surj[of - n, OF $\left.j^{\prime}\right]$ obtain $k^{\prime}$ where $i k^{\prime}$ : ? ind $k^{\prime}=\left(i^{\prime}, j^{\prime}\right) .$.
from inf-mono[OF ik $i k^{\prime} i^{\eta}$ have $k$ : $k \leq k^{\prime}$ by simp
show $\exists k^{\prime} \geq k$. ?ty $k^{\prime} \in\{$ top-s, top-ns $\}$
by (intro exI conjI, rule $k$, unfold $i k^{\prime}$ split, rule ty')
qed
next
show INFM i. ?ty $i \in\{$ top-s, normal-s $\}$
unfolding INFM-nat-le
proof (intro allI)
fix $k$
obtain $i j$ where $i k$ : ? ind $k=(i, j)$ by force
from inf2[unfolded INFM-nat] obtain $i^{\prime}$ where $i^{\prime}: i^{\prime}>i$ and ty: ty $i^{\prime} \in$
\{top-s, normal-s $\}$ by auto
from partial[of $\left.i^{\prime}\right]$ ty obtain $j^{\prime}$ where $j^{\prime}: j^{\prime}<n i^{\prime}$ and $t y^{\prime}: t y^{\prime} i^{\prime} j^{\prime} \in\{t o p-s$, normal-s $\}$ by auto
from inf-concat-surj[of-n, OF $\left.j^{\prime}\right]$ obtain $k^{\prime}$ where $i k^{\prime}$ : ?ind $k^{\prime}=\left(i^{\prime}, j^{\prime}\right) .$.
from inf-mono[OF ik $\left.i k^{\prime} i^{\prime}\right]$ have $k$ : $k \leq k^{\prime}$ by simp

```
        show }\exists\mp@subsup{k}{}{\prime}\geqk.?.?ty k'\in{top-s, normal-s
        by (intro exI conjI, rule k, unfold ik' split, rule ty')
    qed
    qed
    with assms show False by auto
qed
```

lemma $S N$-rel-ext-map: fixes $P P w R R w P^{\prime} P w^{\prime} R^{\prime} R w^{\prime}::{ }^{\prime} a$ rel and $M M^{\prime}$ :: 'a
$\Rightarrow$ bool
defines $M s: M s \equiv\left\{(s, t) . M^{\prime} t\right\}$
defines $A: A \equiv\left(P^{\prime} \cup P w^{\prime} \cup R^{\prime} \cup R w^{\prime}\right) \cap M s$
assumes $S N$ : $S N$-rel-ext $P^{\prime} P w^{\prime} R^{\prime} R w^{\prime} M^{\prime}$
and $P: \bigwedge s t . M s \Longrightarrow M t \Longrightarrow(s, t) \in P \Longrightarrow(f s, f t) \in\left(A^{*} * O\left(P^{\prime} \cap M s\right) O\right.$
$\left.A^{\wedge} *\right) \wedge I t$
and Pw: $\wedge s t . M s \Longrightarrow M t \Longrightarrow(s, t) \in P w \Longrightarrow(f s, f t) \in\left(A^{\wedge} * O\left(\left(P^{\prime} \cup P w^{\prime}\right)\right.\right.$
$\left.\cap M s) O A{ }^{*}\right) \wedge I t$
and $R: \wedge$ st. Is $\Longrightarrow M s \Longrightarrow M t \Longrightarrow(s, t) \in R \Longrightarrow(f s, f t) \in\left(A \mathcal{*} O\left(\left(P^{\prime}\right.\right.\right.$
$\left.\left.\left.\cup R^{\prime}\right) \cap M s\right) O A A^{*}\right) \wedge I t$
and $R w: \wedge s t . I s \Longrightarrow M s \Longrightarrow M t \Longrightarrow(s, t) \in R w \Longrightarrow(f s, f t) \in A^{\wedge} * \wedge I t$
shows $S N$-rel-ext P Pw R Rw M
proof -
note $S N=S N$-rel-ext-trans $[O F S N]$
let $? P=\left(A{ }^{*} O\left(P^{\prime} \cap M s\right) O A^{*}\right)$
let ? $P \mathrm{Pw}=\left(A^{*} * O\left(\left(P^{\prime} \cup P w^{\prime}\right) \cap M s\right) O A{ }^{*}\right)$
let $? R=\left(A^{*} * O\left(\left(P^{\prime} \cup R^{\prime}\right) \cap M s\right) O A{ }^{*}\right)$
let $? R w=A{ }^{*}$
let ?relt $=S N$-rel-ext-step ?P ?Pw ?R ?Rw
let ?rel $=S N$-rel-ext-step $P P w R R w$
show ?thesis
proof (rule ccontr)
assume $\neg$ ?thesis
from this[unfolded $S N$-rel-ext-def]
obtain $g$ ty
where steps: $\bigwedge i .(g i, g($ Suc $i)) \in ?$ rel (ty $i)$
and $\min : \bigwedge i . M(g i)$
and inf1: INFM i. ty $i \in\{$ top-s, top-ns $\}$
and inf2: INFM i. ty $i \in\{$ top-s, normal-s $\}$
by auto
from inf1[unfolded INFM-nat] obtain $k$ where $k:$ ty $k \in\{t o p-s$, top-ns $\}$ by
auto
let $? k=$ Suc $k$
let $? i=$ shift $i d ? k$
let ?f $=\lambda i . f($ shift $g ? k i)$
let ? ${ }^{\text {ty } y=}$ shift ty $? k$
\{
fix $i$
assume ty: ty $i \in\{$ top-s,top-ns $\}$
note $m=\min [o f i]$

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

```
    from rtrancl-mono[OF this] have As:A`*\subseteqMs`* by auto
    have PM: ?P\subseteqMs^* O Ms O Ms`* using As by auto
    have PwM: ?Pw\subseteqMs`* O Ms O Ms`* using As by auto
    have RM: ?R\subseteqMs`* O Ms O Ms`* using As by auto
    have RwM: ?Rw\subseteqMs`* using As by auto
    from PM PwM RM have ?P \cup?Pw\cup?R\subseteqMs`* OMs OMs`* (is ?PPR
\subseteq - ) ~ b y ~ a u t o
    also have ...\subseteqMs^+ by regexp
    also have ... = Ms
proof
    have Ms`+}\subseteq\subseteqMs`* OMs by regex
    also have ...\subseteqMs unfolding Ms by auto
    finally show Ms^+}\subseteqMs
qed regexp
finally have PPR: ?PPR\subseteqMs .
show M'(?f i)
proof (induct i)
    case 0
    from stepsP[OF k] k
    have (f(gk),f(g(Suc k)))\in?PPR by (cases ty k, auto)
    with PPR show ?case unfolding Ms by simp blast
    next
    case (Suc i)
    show ?case
    proof (cases ?ty i= normal-ns)
        case False
        hence ?ty i }\in{\mathrm{ top-s,top-ns,normal-s}
            by (cases ?ty i, auto)
        with stepsI[OF I[of i]] have (?f i, ?f (Suc i)) \in ?PPR
            by auto
        from subsetD[OF PPR this] have (?f i, ?f (Suc i)) \in Ms .
        thus ?thesis unfolding Ms by auto
    next
        case True
        with stepsI[OF I[of i]] have (?f i, ?f (Suc i)) \in ?Rw by auto
        with RwM have mem: (?f i, ?f (Suc i)) \inMs`* by auto
        thus ?thesis
        proof (cases)
            case base
            with Suc show ?thesis by simp
        next
            case step
            thus ?thesis unfolding Ms by simp
        qed
    qed
    qed
qed
with SN
show False unfolding A Ms by simp
```

qed
qed
lemma $S N$-rel-ext-map-min: fixes $P P w R R w P^{\prime} P w^{\prime} R^{\prime} R w^{\prime}::$ 'a rel and $M M^{\prime}$ :: ' $a \Rightarrow$ bool
defines $M s: M s \equiv\left\{(s, t) . M^{\prime} t\right\}$
defines $A: A \equiv P^{\prime} \cap M s \cup P w^{\prime} \cap M s \cup R^{\prime} \cup R w^{\prime}$
assumes $S N$ : $S N$-rel-ext $P^{\prime} P w^{\prime} R^{\prime} R w^{\prime} M^{\prime}$
and $M: \wedge t . M t \Longrightarrow M^{\prime}(f t)$
and $M^{\prime}: \bigwedge s t . M^{\prime} s \Longrightarrow(s, t) \in R^{\prime} \cup R w^{\prime} \Longrightarrow M^{\prime} t$
and $P: \bigwedge s t . M s \Longrightarrow M t \Longrightarrow M^{\prime}(f s) \Longrightarrow M^{\prime}(f t) \Longrightarrow(s, t) \in P \Longrightarrow(f s, f$
$t) \in\left(A^{*} O\left(P^{\prime} \cap M s\right) O A^{*}\right) \wedge I t$
and $P w: \bigwedge s t . M s \Longrightarrow M t \Longrightarrow M^{\prime}(f s) \Longrightarrow M^{\prime}(f t) \Longrightarrow(s, t) \in P w \Longrightarrow(f$ $s, f t) \in\left(A^{*} O\left(P^{\prime} \cap M s \cup P w^{\prime} \cap M s\right) O A^{*}\right) \wedge I t$
and $R: \bigwedge s t$. $I s \Longrightarrow M s \Longrightarrow M t \Longrightarrow M^{\prime}(f s) \Longrightarrow M^{\prime}(f t) \Longrightarrow(s, t) \in R \Longrightarrow$
$(f s, f t) \in\left(A^{*} * O\left(P^{\prime} \cap M s \cup R^{\prime}\right) O A{ }^{*}\right) \wedge I t$
and $R w: \bigwedge s t . I s \Longrightarrow M s \Longrightarrow M t \Longrightarrow M^{\prime}(f s) \Longrightarrow M^{\prime}(f t) \Longrightarrow(s, t) \in R w$ $\Longrightarrow(f s, f t) \in A^{\wedge} * \wedge I t$
shows $S N$-rel-ext $P$ Pw R Rw M
proof -
let $? M s=\left\{(s, t) . M^{\prime} t\right\}$
let ? $A=\left(P^{\prime} \cup P w^{\prime} \cup R^{\prime} \cup R w^{\prime}\right) \cap$ ?Ms
\{
fix $s t$
assume $s: M^{\prime} s$ and $(s, t) \in A$
with $M^{\prime}[O F s$, of $t]$ have $(s, t) \in ? A \wedge M^{\prime} t$ unfolding $M s A$ by auto
\} note Aone $=$ this
\{
fix $s t$
assume $s: M^{\prime} s$ and steps: $(s, t) \in A{ }^{*}$
from steps have $(s, t) \in ? A^{*} * \wedge M^{\prime} t$
proof (induct)
case base from $s$ show? case by simp
next
case (step $t u$ )
note one $=$ Aone $[$ OF step(3)[THEN conjunct2] $\operatorname{step}$ (2)]
from step (3) one
have steps: $(s, u) \in ? A^{`} * O ? A$ by blast
have $(s, u) \in ? A^{*}$
by (rule subset $D[O F-$ steps $]$, regexp)
with one show? case by simp
qed
$\}$ note Amany $=$ this
let $? P=\left(A^{*} * O\left(P^{\prime} \cap M s\right) O A{ }^{*}\right)$
let ? $P \mathrm{P}=\left(A^{*} * O\left(P^{\prime} \cap M s \cup P w^{\prime} \cap M s\right) O A \wedge\right)$
let ? $R=\left(A^{*} O\left(P^{\prime} \cap M s \cup R^{\prime}\right) O A{ }^{*}\right)$
let $? R w=A^{*} *$
let $? P^{\prime}=\left(? A \widehat{*} O\left(P^{\prime} \cap ? M s\right) O ? A^{*}\right)$

```
let \(? P w^{\prime}=\left(? A{ }^{*} *\left(\left(P^{\prime} \cup P w^{\prime}\right) \cap ? M s\right) O ? A{ }^{*}\right)\)
let \(? R^{\prime}=\left(? A^{*} O\left(\left(P^{\prime} \cup R^{\prime}\right) \cap ? M s\right) O ? A^{*}\right)\)
let \(? R w^{\prime}=? A^{-}{ }^{*}\)
show ?thesis
proof (rule SN-rel-ext-map[OF SN])
    fix \(s t\)
    assume \(s: M s\) and \(t: M t\) and step: \((s, t) \in P\)
    from \(P[O F s t M[O F s] M[O F t]\) step \(]\)
    have \((f s, f t) \in ? P\) and \(I: I t\) by auto
    then obtain \(u v\) where \(s u:(f s, u) \in A^{*}\) and \(u v:(u, v) \in P^{\prime} \cap M s\)
        and \(v t:(v, f t) \in A^{*} *\) by auto
    from Amany[OF \(M[O F s] s u]\) have \(s u:(f s, u) \in ? A^{*} *\) and \(u: M^{\prime} u\) by auto
    from \(u v\) have \(v: M^{\prime} v\) unfolding \(M s\) by auto
    from Amany \([O F v v t]\) have \(v t:(v, f t) \in ? A{ }^{*} *\) by auto
    from su uv vt I
    show \((f s, f t) \in ? P^{\prime} \wedge I t\) unfolding \(M s\) by auto
next
    fix \(s t\)
    assume \(s: M s\) and \(t: M t\) and step: \((s, t) \in P w\)
    from \(P w[O F\) s \(t M[O F s] M[O F t]\) step \(]\)
    have \((f s, f t) \in ? P w\) and \(I: I t\) by auto
    then obtain \(u v\) where \(s u:(f s, u) \in A^{*} *\) and \(u v:(u, v) \in P^{\prime} \cap M s \cup P w^{\prime} \cap\)
Ms
        and \(v t:(v, f t) \in A^{\wedge} *\) by auto
    from Amany \(\left[O F M[O F s]\right.\) su] have su: \((f s, u) \in ? A^{*} *\) and \(u: M^{\prime} u\) by auto
    from \(u v\) have \(u v:(u, v) \in\left(P^{\prime} \cup P w^{\prime}\right) \cap\) ?Ms and \(v: M^{\prime} v\) unfolding \(M s\)
        by auto
    from Amany[OF v vt] have vt: \((v, f t) \in ? A^{\circ} *\) by auto
    from su uv vt I
    show \((f s, f t) \in ? P w^{\prime} \wedge I t\) by auto
next
    fix \(s t\)
    assume \(I: I s\) and \(s: M s\) and \(t: M t\) and step: \((s, t) \in R\)
    from \(R[O F I s t M[O F s] M[O F t]\) step \(]\)
    have \((f s, f t) \in ?\) ? and \(I: I t\) by auto
    then obtain \(u v\) where su: \((f s, u) \in A^{*}\) and \(u v:(u, v) \in P^{\prime} \cap M s \cup R^{\prime}\)
        and \(v t:(v, f t) \in A^{*} *\) by auto
    from Amany[OF \(M[O F s] s u]\) have \(s u:(f s, u) \in ? A^{*}\) and \(u: M^{\prime} u\) by auto
        from \(u v M^{\prime}[O F u\), of \(v]\) have \(u v:(u, v) \in\left(P^{\prime} \cup R^{\prime}\right) \cap ? M s\) and \(v: M^{\prime} v\)
unfolding \(M s\)
        by auto
    from Amany \([O F v v t]\) have \(v t:(v, f t) \in ? A^{*} *\) by auto
    from su uv vt \(I\)
    show \((f s, f t) \in ? R^{\prime} \wedge I t\) by auto
next
    fix \(s t\)
    assume \(I: I s\) and \(s: M s\) and \(t: M t\) and step: \((s, t) \in R w\)
    from \(R w\left[\right.\) OF Is \(t M\left[\begin{array}{lll}\text { OF } s] & M[O F t] & \text { step }]\end{array}\right.\)
    have steps: \((f s, f t) \in ? R w\) and \(I: I t\) by auto
```

```
    from Amany[OF M[OF s] steps] I
    show (fs,ft)\in?Rw'^It by auto
    qed
qed
```

lemma $S N$-relto-imp-SN-rel: $S N($ relto $R S) \Longrightarrow S N$-rel $R S$
proof -
assume $S N$ : $S N$ (relto $R S$ )
show ?thesis
proof (simp only: SN-rel-on-conv $S N$-rel-defs, intro allI impI)
fix $f$
presume steps: chain $(R \cup S) f$
obtain $r$ where $r: \bigwedge j . r j \equiv(f j, f(S u c j)) \in R$ by auto
show $\neg(I N F M j .(f j, f(S u c j)) \in R)$
proof (rule ccontr)
assume $\neg$ ?thesis
hence ih: infinitely-many $r$ unfolding infinitely-many-def $r$ INFM-nat-le by
blast
obtain $r$-index where $r$-index $=$ infinitely-many.index $r$ by simp
with infinitely-many.index-p[OF ih] infinitely-many.index-ordered $[O F$ ih] in-
finitely-many.index-not-p-between[OF ih]
have r-index: $\wedge i . r(r$-index $i) \wedge r$-index $i<r$-index $($ Suc $i) \wedge(\forall j$. r-index
$i<j \wedge j<r$-index (Suc $i) \longrightarrow \neg r j$ ) by auto
obtain $g$ where $g: \wedge i . g i \equiv f(r$-index $i) .$.
\{
fix $i$
let ? $r i=r$-index $i$
let ? $\mathrm{rsi}=r$-index $(S u c i)$
from $r$-index have isi: ? ri $<$ ? rsi by auto
obtain ri rsi where $r i: r i=$ ? ri and rsi: rsi $=$ ? rsi by auto
with $r$-index[of $i]$ steps have inter: $\wedge j . r i<j \wedge j<r s i \Longrightarrow(f j, f(S u c$
$j)) \in S$ unfolding $r$ by auto
from ri isi rsi have risi: ri<rsi by simp
\{
fix $n$
assume Suc $n \leq r s i-r i$
hence $(f$ (Suc ri), $f($ Suc $(n+r i))) \in S^{*}$
proof (induct n, simp)
case (Suc n)
hence stepps: $(f$ (Suc ri), $f(S u c(n+r i))) \in S^{\wedge}$ * by simp
have $(f(S u c(n+r i)), f(S u c(S u c n+r i))) \in S$
using inter[of Suc $n+r i]$ Suc(2) by auto
with stepps show ?case by simp
qed
\}
from this[of rsi - ri-1] risi have
$(f$ (Suc ri), $f$ rsi $) \in S^{\wedge} *$ by simp
with ri rsi have ssteps: $(f$ (Suc ?ri),$f$ ? rsi $) \in S^{\wedge}$ * by simp

```
            with r-index[of i] have (f ?ri, f ?rsi) \inR O S** unfolding r by auto
            hence (gi,g(Suc i)) \in S *}OROS`* using rtrancl-refl unfolding g by
auto
            }
            hence }\negSN(S`*OROS`*) unfolding SN-defs by blas
            with SN show False by simp
        qed
    qed simp
qed
lemma rtrancl-list-conv:
    ((s,t) \in R`*) =
    (\existslist.last (s # list) = t ^ ( }\forall\textrm{i}.i<length list \longrightarrow ((s # list)! i, (s # list)!
Suc i)\inR))(is ?l = ?r)
proof
    assume ?r
    then obtain list where last (s# list) = t\wedge(\forall i. i< length list \longrightarrow((s# list)
! i, (s # list)! Suc i) \inR)..
    thus?l
    proof (induct list arbitrary: s, simp)
        case (Cons u ll)
    hence last (u# #l) =t\wedge(\forall i. i< length ll \longrightarrow((u# ll)!i,(u# ll)!Suc
i) }\inR\mathrm{ ) by auto
    from Cons(1)[OF this] have rec: (u,t)\in R`*.
    from Cons have (s,u)\inR 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:Sn=t and steps: }\foralli<n.(Si,S (Su
i)) }\inR\mathrm{ by auto
    let ?list = map (\lambda i.S (Suc i)) [0 ..< n]
    show ?r
    proof (rule exI[of - ?list], intro conjI,
        cases n, simp add: s[symmetric] t[symmetric], simp add: t[symmetric])
        show }\foralli<length ?list. ((s # ?list)!i,(s# ?list)!Suc i)\in
        proof (intro allI impI)
            fix }
                assume i:i< length ?list
                thus ((s # ?list)!i,(s # ?list)!Suc i) \inR
            proof (cases i, simp add: s[symmetric] steps)
            case (Suc j)
            with i steps show ?thesis by simp
        qed
        qed
    qed
qed
```

```
fun choice :: (nat \(\Rightarrow\) 'a list \() \Rightarrow\) nat \(\Rightarrow\) (nat \(\times\) nat \()\) where
    choice f \(0=(0,0)\)
    \(\mid\) choice \(f(\) Suc \(n)=(\) let \((i, j)=\) choice \(f n\) in
        if Suc \(j<\) length \((f i)\)
        then ( \(i\), Suc \(j\) )
        else (Suc i, O))
lemma \(S N\)-rel-imp-SN-relto : \(S N\)-rel \(R S \Longrightarrow S N(\) relto \(R S\) )
proof -
    assume \(S N\) : \(S N\)-rel \(R S\)
    show \(S N\) (relto \(R S\) )
    proof
        fix \(f\)
        assume \(\forall i .(f i, f(\) Suc \(i)) \in\) relto \(R S\)
    hence steps: \(\wedge i .(f i, f(\) Suc \(i)) \in S^{\wedge} * O R O S{ }^{*}\) by auto
    let ?prop \(=\lambda\) i ai bi. \((f i, b i) \in S^{\wedge}{ }^{*} \wedge(b i, a i) \in R \wedge(a i, f(S u c(i))) \in S^{\wedge}\)
    \{
        fix \(i\)
        from steps obtain bi ai where ?prop \(i\) ai bi by blast
        hence \(\exists\) ai bi. ?prop \(i\) ai bi by blast
    \}
    hence \(\forall i\). \(\exists\) bi ai. ?prop \(i\) ai bi by blast
    from choice \([O F\) this \(]\) obtain \(b\) where \(\forall i\). \(\exists\) ai. ?prop \(i\) ai \((b i)\) by blast
    from choice \([O F\) this] obtain \(a\) where steps: \(\bigwedge i\). ?prop \(i(a i)(b i)\) by blast
    let ?prop \(=\lambda i l i .(b i, a i) \in R \wedge(\forall j<\) length li. \(((a i \# l i)!j,(a i \# l i)\) !
Suc \(j) \in S) \wedge\) last \((a i \# l i)=b(\) Suc \(i)\)
    \{
        fix \(i\)
        from steps [of i] steps \([\) of Suc \(i]\) have \((a i, f(S u c i)) \in S^{\wedge} *\) and \((f(S u c i), b\)
(Suc \(i)) \in S^{\wedge}\) by auto
        from rtrancl-trans \([\) OF this] steps \([o f i]\) have \(R:(b i, a i) \in R\) and \(S:(a i, b\)
(Suc i)) \(\in S^{\wedge}\) * by blast +
        from \(S\) [unfolded rtrancl-list-conv] obtain \(l i\) where last \((a i \# l i)=b(S u c\)
i) \(\wedge(\forall j<\) length li. \(((a i \# l i)!j,(a i \# l i)!S u c j) \in S)\)..
        with \(R\) have ?prop ili by blast
        hence \(\exists\) li. ?prop ili ..
    \}
    hence \(\forall\) i. \(\exists\) li. ?prop ili ..
    from choice \([O F\) this \(]\) obtain \(l\) where steps: \(\bigwedge i\). ?prop \(i(l i)\) by auto
    let ? \(p=\lambda i\). ?prop \(i(l i)\)
    from steps have steps: \(\bigwedge i\). ?p \(i\) by blast
    let \(? l=\lambda i\). a \(i \# l i\)
    let ? \(g=\lambda\) i. choice \((\lambda j\). ?l \(j) i\)
    obtain \(g\) where \(g: \bigwedge i . g i=(\operatorname{let}(i i, j j)=? g i\) in ?l \(i i!j j)\) by auto
    have len: \(\bigwedge i j n\).?g \(n=(i, j) \Longrightarrow j<\) length (?l \(i)\)
    proof -
        fix \(i j n\)
        assume \(n\) : ?g \(n=(i, j)\)
```

```
    show j< length (?l i)
    proof (cases n)
        case 0
        with n have j=0 by auto
        thus ?thesis by simp
    next
        case (Suc nn)
        obtain ii jj where nn: ?g nn = (ii,jj) by (cases ?g nn, auto)
        show ?thesis
        proof (cases Suc jj < length (?l ii))
            case True
            with nn Suc have ?g n = (ii,Suc jj) by auto
            with n True show ?thesis by simp
        next
            case False
            with nn Suc have ?g n=(Suc ii,0) by auto
            with n show ?thesis by simp
        qed
    qed
qed
have gsteps: \bigwedgei.(gi,g(Suc i))\inR\cupS
proof -
    fix n
    obtain i j where n: ?g n = (i,j) by (cases ?g n,auto)
    show (gn,g(Suc n)) \inR\cupS
    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 \mp@code { b y ~ a u t o }
        thus ?thesis using steps[of i] True by auto
    next
        case False
        with n have ?g (Suc n)=(Suc i, 0) by auto
        with n have gn: g n =?l i! j and gsn: g (Suc n)=a(Suc i) unfolding
g \mp@code { b y ~ a u t o }
    from gn len[OF n] False have j=length (?l i) - 1 by auto
    with gn have gn: g n = last (?l i) using last-conv-nth[of ?l i] by auto
    from gn gsn show ?thesis using steps[of i] steps[of Suc i] by auto
    qed
qed
have infR: }\foralln.\existsj\geqn.(gj,g(Suc j))\in
proof
    fix n
    obtain i j where n: ?g n = (i,j) by (cases ?g n, auto)
    from len[OF n] have j:j\leq length (?l i) - 1 by simp
    let ?k = length (?l i) - 1-j
    obtain k where k: k=j+?k by auto
    from jk have k2: k = length (?l i) - 1 and k3: j + ?k<length (?l i) by
```

```
auto
            fix nijkl
            assume n: choice l n = (i,j) and j + k<length (l i)
            hence choice l ( n+k) = (i,j+k)
            by (induct k arbitrary: j, simp, auto)
        }
        from this[OF n, of ?k, OF k3]
        have gnk:?g (n+?k)=(i,k) by (simp only: k)
        hence g}(n+?k)=?l i!k unfolding g by aut
        hence gnk2: g ( n + ?k) = last (?l i) using last-conv-nth[of ?l i] k2 by auto
        from k2 gnk have ?g (Suc (n+?k)) = (Suc i,0) by auto
        hence gnsk2: g (Suc (n+?k))=a (Suc i) unfolding g by auto
        from steps[of i] steps[of Suc i] have main: (g (n+?k),g(Suc (n+?k)))\inR
        by (simp only: gnk2 gnsk2)
        show }\existsj\geqn.(gj,g(Suc j))\in
        by (rule exI[of-n + ?k], auto simp: main[simplified])
    qed
    from SN[simplified SN-rel-on-conv SN-rel-defs] gsteps infR show False
        unfolding INFM-nat-le by fast
    qed
qed
hide-const choice
lemma SN-relto-SN-rel-conv: SN (relto R S) = SN-rel R S
    by (blast intro: SN-relto-imp-SN-rel SN-rel-imp-SN-relto)
lemma SN-rel-empty1: SN-rel {} S
    unfolding SN-rel-defs by auto
lemma SN-rel-empty2: SN-rel R {}=SN R
    unfolding SN-rel-defs SN-defs by auto
lemma SN-relto-mono:
    assumes R:R\subseteq\mp@subsup{R}{}{\prime}\mathrm{ and S:S}\subseteq\mp@subsup{S}{}{\prime}
    and SN:SN (relto R' S')
    shows }SN\mathrm{ (relto R S)
    using SN SN-subset[OF - relto-mono[OF R S]] by blast
lemma SN-relto-imp-SN:
    assumes SN (relto R S) shows SN R
proof
    fix f
    assume }\foralli.(fi,f(Suc i)) \in
    hence }\i.(fi,f(Suci))\in\mathrm{ relto R S by blast
    thus False using assms unfolding SN-defs by blast
qed
```

```
lemma SN-relto-Id:
    SN (relto R (S\cupId)) =SN (relto R S)
    by (simp only: relto-Id)
    Termination inheritance by transitivity (see, e.g., Geser's thesis).
lemma trans-subset-SN:
    assumes trans R and R\subseteq(r\cups) and SNr and SNs
    shows SN R
proof
    fix f :: nat }=>\mp@subsup{}{}{\prime}
    assume f 0 \in UNIV
        and chain: chain R f
    have *: \ij.i<j\Longrightarrow(fi,fj) \inr\cups
        using assms and chain-imp-trancl [OF chain] by auto
    let ?M = {i.}\forallj>i.(fi,fj)\not\inr
    show False
    proof (cases finite ?M)
        let ? n = Max ?M
        assume finite?M
        with Max-ge have }\foralli\in?M.i\leq?n by sim
        then have }\forallk\geqSuc ?n. \exists\mp@subsup{k}{}{\prime}>k.(fk,f\mp@subsup{k}{}{\prime})\inr\mathrm{ by auto
        with steps-imp-chainp [of Suc ?n \lambdax y. (x,y)\inr] and assms
            show False by auto
    next
        assume infinite ?M
        then have INFM j. j\in?M by (simp add: Inf-many-def)
        then interpret infinitely-many \lambdai.i\in?M by (unfold-locales) assumption
        define g}\mathrm{ where [simp]: g= index
        have \foralli.(f(gi),f(g(Suc i))) \ins
        proof
            fix }
            have less: g i < g (Suc i) using index-ordered-less [of i Suc i] by simp
            have gi\in?M using index-p by simp
            then have (f(g i),f(g(Suc i))) &r using less by simp
            moreover have (f(gi),f(g(Suc i)))\inr\cups using * [OF less] by simp
            ultimately show (f (gi),f(g(Suc i))) \ins by blast
        qed
        with «SN s` show False by (auto simp: SN-defs)
    qed
qed
lemma SN-Un-conv:
    assumes trans (r\cups)
    shows SN (r\cups)\longleftrightarrowSNr^SNs
        (is SN ?r \longleftrightarrow ?rhs)
proof
    assume SN (r\cups) thus SNr^\SNs
        using SN-subset[of ?r] by blast
next
```

```
    assume SN r ^ SN s
    with trans-subset-SN[OF assms subset-refl] show SN ?r by simp
qed
lemma SN-relto-Un:
    SN(relto }(R\cupS)Q)\longleftrightarrowSN(\mathrm{ relto }R(S\cupQ))\wedgeSN(relto S Q
    (is SN ?a \longleftrightarrowSN ?b ^SN?c)
proof -
    have eq: ?a^}+=?\mp@subsup{b}{}{\wedge}+\cup?\mp@subsup{?}{}{\wedge}+\mathrm{ by regexp
    from SN-Un-conv[of ?b^+? ?c}+\mathrm{ , unfolded eq[symmetric]]
        show ?thesis unfolding SN-trancl-SN-conv by simp
qed
lemma SN-relto-split:
    assumes SN (relto r (s\cupq2)\cup relto q1 (s\cupq2)) (is SN ?a)
        and SN (relto s q2) (is SN ?b)
    shows SN (relto r (q1 \cupq2) \cup relto s (q1 \cupq2)) (is SN ?c)
proof -
    have }?\mp@subsup{c}{}{\wedge}+\subseteq?\a`+\cup?b^+ by regex
    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 $a b \subseteq c \widehat{ }+$
proof -
have relto $a b \subseteq(a \cup b)^{\wedge}+$ by regexp
also have $\ldots \subseteq c^{\wedge}+$
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 $\Rightarrow{ }^{\prime}$ a rel $\Rightarrow$ nat $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow(n a t \Rightarrow$ bool $) \Rightarrow$ nat $\Rightarrow{ }^{\prime} a \times{ }^{\prime} a \Rightarrow$ bool where
relto-fun: as $0=a \Longrightarrow$ as $m=b \Longrightarrow$
( $\bigwedge i . i<m \Longrightarrow$
$($ sel $i \longrightarrow($ as $i$, as $($ Suc $i)) \in A) \wedge(\neg$ sel $i \longrightarrow($ as $i$, as $($ Suc $i)) \in B))$
$\Longrightarrow n=$ card $\{i . i<m \wedge$ sel $i\}$
$\Longrightarrow(n=0 \longleftrightarrow m=0) \Longrightarrow$ relto-fun $A B n$ as sel $m(a, b)$
lemma relto-funD: assumes relto-fun $A B n$ as sel $m(a, b)$
shows as $0=a$ as $m=b$
$\bigwedge i . i<m \Longrightarrow$ sel $i \Longrightarrow($ as $i$, as $($ Suc $i)) \in A$
$\bigwedge i . i<m \Longrightarrow \neg$ sel $i \Longrightarrow($ as $i$, as $($ Suc $i)) \in B$
$n=\operatorname{card}\{i . i<m \wedge$ sel $i\}$
$n=0 \longleftrightarrow m=0$
using assms[unfolded relto-fun.simps] by blast+
lemma relto-fun-refl: $\exists$ as sel. relto-fun AB as sel $0(a, a)$

```
    by (rule exI[of - \lambda -. a], rule exI, rule relto-fun, auto)
lemma relto-into-relto-fun: assumes (a,b) \in relto A B
    shows \exists as sel m. relto-fun A B (Suc 0) as sel m (a,b)
proof -
    from assms obtain }\mp@subsup{a}{}{\prime}\mp@subsup{b}{}{\prime}\mathrm{ where }aa:(a,\mp@subsup{a}{}{\prime})\in\mp@subsup{B}{}{`}*\mathrm{ and ab: ( a', b})\in
    and bb: ( }\mp@subsup{b}{}{\prime},b)\in\mp@subsup{B}{}{`*}\mathrm{ * by auto
    from aa[unfolded rtrancl-fun-conv] obtain f1 n1 where
        f1: f1 0 = a f1 n1 = a' \ i. i<n1\Longrightarrow(f1 i,f1 (Suc i)) \in B by auto
    from bb[unfolded rtrancl-fun-conv] obtain f2 n2 where
        f2: f2 0 = b' f2 n2 = b \ i.i<n2 \Longrightarrow(f2 i, f2 (Suc i)) \in B by auto
    let ?gen = \lambda aa ab bb i. if i<n1 then aa i else if i=n1 then ab else bb ( }i
Suc n1)
    let ?f = ?gen f1 a' f2
    let ?sel =?gen ( }\lambda\mathrm{ -. False) True ( }\lambda\mathrm{ -. False)
    let ?m = Suc (n1 + n2)
    show ?thesis
    proof (rule exI[of - ?f], rule exI[of - ?sel], rule exI[of - ?m], rule relto-fun)
        fix }
        assume i:i<?m
        show (?sel i\longrightarrow(?f i, ?f (Suc i)) \inA)\wedge(\neg?sel i}\longrightarrow(?f i,?f (Suc i))\inB
        proof (cases i<n1)
            case True
            with f1(3)[OF this] f1(2) show ?thesis by (cases Suc i=n1, auto)
        next
            case False note nle = this
            show ?thesis
            proof (cases i>n1)
                case False
                with nle have i=n1 by auto
                thus ?thesis using f1 f2 ab by auto
                next
                case True
                define j where j=i - Suc n1
                have i: i=Suc n1 + j and j:j<n2 using i True unfolding j-def by
auto
                thus ?thesis using f2 by auto
                qed
            qed
    qed (insert f1 f2, auto)
qed
lemma relto-fun-trans: assumes ab: relto-fun A B n1 as1 sel1 m1 (a,b)
    and bc: relto-fun A B n2 as2 sel2 m2 (b,c)
    shows \exists as sel. relto-fun A B (n1 + n2) as sel (m1 +m2) (a,c)
proof -
    from relto-funD[[OF ab]
    have 1: as1 0 =a as1 m1 = b
        \i.i<m1\Longrightarrow(sel1 i\longrightarrow(as1 i, as1 (Suc i)) \inA)\wedge(\neg sel1 i \longrightarrow (as1 i,
```

```
as1 (Suc i)) \in B)
    n1=0\longleftrightarrowm1=0 and card1:n1 = card {i. i<m1^ sel1 i} by blast+
    from relto-funD[[OF bc]
    have 2: as2 0 = b as2 m2 = c
    ^i.i<m2\Longrightarrow(sel2 i \longrightarrow(as2 i, as2 (Suc i)) \inA)^(\neg sel2 i \longrightarrow(as2 i,
as2 (Suc i)) \in B)
    n2 = 0 \longleftrightarrowm2 = 0 and card2: n2 = card {i. i<m2 ^ sel2 i} by blast+
    let ?as = \lambda i. if i<m1 then as1 i else as2 ( i - m1)
    let ?sel = \lambda i. if i<m1 then sel1 i else sel2 ( }i-m1
    let ?m = m1 + m2
    let ? n = n1 + n2
    show ?thesis
    proof (rule exI[of - ?as], rule exI[of - ?sel], rule relto-fun)
    have id: {i.i<?m^ ?sel i} ={ i. i<m1^ sel1 i} U((+)m1)'{ i. i
< m2 ^ sel2 i}
        (is - = ?A U ?f' ?B)
        by force
    have card (?A\cup?f'?B) = card ?A + card (?f'?B)
        by (rule card-Un-disjoint, auto)
    also have card (?f' ?B) = card ?B
        by (rule card-image, auto simp: inj-on-def)
    finally show ? n = card { i. i< ?m ^ ?sel i} unfolding card1 card2 id by
simp
    next
        fix }
        assume i:i<?m
        show (?sel i\longrightarrow(?as i,?as (Suc i)) \inA)\wedge(\neg ?sel i \longrightarrow (?as i, ?as (Suc i))
\inB)
    proof (cases i<m1)
            case True
            from 12 have [simp]: as2 0 = as1 m1 by simp
            from True 1(3)[of i] 1(2) show ?thesis by (cases Suc i=m1,auto)
    next
            case False
            define j where j=i-m1
            have i: i=m1 + j and j:j<m2 using i False unfolding j-def by auto
            thus ?thesis using False 2(3)[of j] by auto
        qed
    qed (insert 1 2, auto)
qed
lemma reltos-into-relto-fun: assumes (a,b)\in(relto A B )^n
    shows \exists as sel m. relto-fun A B n as sel m (a,b)
    using assms
proof (induct n arbitrary: b)
    case (0 b)
    hence b:b=a by auto
    show ?case unfolding b using relto-fun-refl[of A B a] by blast
next
```

```
    case (Suc n c)
    from relpow-Suc-E[OF Suc(2)]
    obtain b where ab: (a,b)\in(relto A B)^n and bc: (b,c)\in relto A B by auto
    from Suc(1)[OFab] obtain as sel m where
    IH: relto-fun A B n as sel m (a,b) by auto
    from relto-into-relto-fun[OF bc] obtain as sel m where relto-fun A B (Suc 0)
as sel m (b,c) by blast
    from relto-fun-trans[OF IH this] show ?case by auto
qed
lemma relto-fun-into-reltos: assumes relto-fun A B n as sel m (a,b)
    shows (a,b) \in(relto A B)^^n
proof -
    note * = relto-funD[OF assms]
    {
    fix m'
    let ?c = \lambda m'. card {i.i< m'^ sel i}
    assume m'\leqm
    hence (?c m'>
(as 0, as m') \in B`*)
    proof (induct m')
            case (Suc m')
            let ?}x=\mathrm{ as 0
            let ?y = as m'
            let ?z=as(Suc m')
            let ?C = ?c (Suc m')
            have C: ?C = ?c m' + (if (sel m') then 1 else 0)
            proof -
                have id: {i.i<Suc m'^ sel i} ={i.i< m'^ sel i} \cup(if sel m}\mp@subsup{m}{}{\prime}\mathrm{ then
{m'} else {})
            by (cases sel m', auto, case-tac x= m', auto)
                show ?thesis unfolding id by auto
    qed
    from Suc(2) have m': m'sm and lt: m'< m by auto
    from Suc(1)[OF m'] have IH: ?c m'>0\Longrightarrow(?x,?y) \in(relto A B)^~ ?c
m'
                ?c m' = 0 \Longrightarrow(?x, ?y) \in B`* by auto
    from *(3-4)[OF lt] have yz: sel m' \Longrightarrow(?y,?z)\inA\neg sel m' \Longrightarrow(?y,?z)
E by auto
    show ?case
    proof (cases ?c m'=0)
        case True note c= this
        from IH(2)[OF this] have xy:(?x, ?y) \in B`* by auto
        show ?thesis
        proof (cases sel m')
            case False
            from xy yz(2)[OF False] have xz: (?x,?z) \in B`* by auto
            from False c have C: ?C = 0 unfolding C by simp
            from }xz\mathrm{ show ?thesis unfolding C by auto
```

```
            next
                    case True
            from xy yz(1)[OF True] have xz:(?x,?z) \in relto A B by auto
            from True c have C: ?C = 1 unfolding C by simp
            from xz show ?thesis unfolding C by auto
        qed
        next
            case False
            hence c:?c m'>0(?c m'=0) = False by arith+
            from }IH(1)[OFc(1)] have xy:(?x,?y)\in(relto A B)^^ ?c m'
            show ?thesis
            proof (cases sel m')
                case False
            from c obtain k where ck: ?c m'=Suc k by (cases ?c m', auto)
            from relpow-Suc-E[OF xy[unfolded this]] obtain
                    u}\mathrm{ where xu: (?x,u) ( (relto A B)^ }~\mathrm{ and uy: (u,?y) f relto A B by
auto
            from uy yz(2)[OF False] have uz: (u,?z) \in relto A B by force
            with }xu\mathrm{ have }xz:(?x,?z)\in(\mathrm{ relto }AB)~~?c m'unfolding ck by aut
            from False c have C: ?C = ?c m' unfolding C by simp
            from xz show ?thesis unfolding C c by auto
            next
                case True
                from xy yz(1)[OF True] have xz: (?x,?z)\in(relto A B)~ (Suc (?c m'))
by auto
            from c True have C:?C = Suc (?c m') unfolding C by simp
            from xz show ?thesis unfolding C by auto
            qed
        qed
    qed simp
    }
    from this[of m]* show ?thesis by auto
qed
lemma relto-relto-fun-conv: ((a,b)\in(relto A B )^n})=(\exists\mathrm{ as sel m. relto-fun A
B n as sel m (a,b))
    using relto-fun-into-reltos[of A B n-- a b] reltos-into-relto-fun[of a b n B A]
by blast
```

```
lemma relto-fun-intermediate: assumes \(A \subseteq C\) and \(B \subseteq C\)
```

lemma relto-fun-intermediate: assumes $A \subseteq C$ and $B \subseteq C$
and rf: relto-fun $A B n$ as sel $m(a, b)$
and rf: relto-fun $A B n$ as sel $m(a, b)$
shows $i \leq m \Longrightarrow(a$, as $i) \in C^{\wedge} *$
shows $i \leq m \Longrightarrow(a$, as $i) \in C^{\wedge} *$
proof (induct $i$ )
proof (induct $i$ )
case 0
case 0
from relto-funD $[O F r f]$ show ?case by simp
from relto-funD $[O F r f]$ show ?case by simp
next
next
case (Suc i)
case (Suc i)
hence $I H:($ a, as $i) \in C^{*}$ and $i m: i<m$ by auto
hence $I H:($ a, as $i) \in C^{*}$ and $i m: i<m$ by auto
from relto-funD (3-4)[OF rf im] assms have (as i, as (Suc i)) $\in C$ by auto

```
    from relto-funD (3-4)[OF rf im] assms have (as i, as (Suc i)) \(\in C\) by auto
```

with $I H$ show ?case by auto
qed
lemma not-SN-on-rel-succ:
assumes $\neg S N$-on (relto $R E$ ) $\{s\}$
shows $\exists t u$. $(s, t) \in E^{*} \wedge(t, u) \in R \wedge \neg S N$-on (relto $\left.R E\right)\{u\}$
proof -
obtain $v$ where $(s, v) \in$ relto $R E$ and $v: \neg S N$-on (relto $R E)\{v\}$
using assms by fast
moreover then obtain $t$ and $u$
where $(s, t) \in E^{*}$ and $(t, u) \in R$ and $u v:(u, v) \in E^{*}$ by auto
moreover from $u v$ have $u v:(u, v) \in(R \cup E)^{*} *$ by regexp
moreover have $\neg S N$-on (relto $R E$ ) $\{u\}$ using
$v$ steps-preserve-SN-on-relto[OF uv] by auto
ultimately show ?thesis by auto
qed
lemma $S N$-on-relto-relcomp: $S N$-on (relto $R S$ ) $T=S N$-on $\left(S^{*} O R\right.$ ) $T$ (is ? $L T$
$=? R T)$
proof
assume $L$ : ? $L T$
$\{$ fix $t$ assume $t \in T$ hence ? $L\{t\}$ using $L$ by fast $\}$
thus ? $R T$ by fast
next
\{ fix $s$
have $S N$-on (relto $R S$ ) $\{s\}=S N$-on $\left(S^{*} O R\right)\{s\}$
proof
let ? $X=\{s . \neg S N$-on (relto $R S$ ) $\{s\}\}$
$\{$ assume $\neg$ ? $L\{s\}$
hence $s \in$ ? $X$ by auto
hence $\neg$ ? $R\{s\}$
proof (rule lower-set-imp-not-SN-on, intro ballI)
fix $s$ assume $s \in$ ? $X$
then obtain $t u$ where $(s, t) \in S^{*}(t, u) \in R$ and $u: u \in ? X$
unfolding mem-Collect-eq by (metis not-SN-on-rel-succ)
hence $(s, u) \in S^{*} O R$ by auto
with $u$ show $\exists u \in ? X .(s, u) \in S^{*} O R$ by auto qed
\}
thus $? R\{s\} \Longrightarrow$ ? $L\{s\}$ by auto
assume ? $L\{s\}$ thus ? $R\{s\}$ by (rule $S N$-on-mono, auto)
qed
$\}$ note main $=$ this
assume $R$ : ? $R$ T
\{ fix $t$ assume $t \in T$ hence ? $L\{t\}$ unfolding main using $R$ by fast \}
thus ? $L T$ by fast
qed
lemma trans-relto:

```
    assumes trans: trans R and SOR\subseteqROS
    shows trans (relto R S)
proof
    fix abc
    assume ab: (a,b)\in\mp@subsup{S}{}{*}ORO\mp@subsup{S}{}{*}\mathrm{ and bc: (b,c) & S* OROS*}
    from rtrancl-O-push [of S R] assms(2) have comm: S* OR\subseteqRO S* by blast
    from ab obtain d e where de: (a,d)\in S* (d,e)\inR (e,b)\inS* by auto
    from bc obtain fg}\mathrm{ where fg:(b,f) & S** (f,g) &R (g,c) & S* by auto
    from de(3) fg(1) have (e,f) \in S* by auto
    with fg(2) comm have (e,g)\inROS 的 by blast
    then obtain h where h: (e,h)\inR(h,g)\inS* by auto
    with de(2) trans have dh: (d,h) \inR unfolding trans-def by blast
    from fg(3) h(2) have (h,c)\in\mp@subsup{S}{}{*}\mathrm{ by auto}
    with de(1) dh(1) show (a,c)\in S* ORO S* by auto
qed
lemma relative-ending:
    assumes chain: chain (R\cupS)t
        and t0:t 0\inX
    and SN:SN-on (relto R S) X
    shows }\existsj.\foralli\geqj.(ti,t(Suc i))\inS-
proof (rule ccontr)
    assume \neg?thesis
    with chain have }\foralli.\existsj.j\geqi\wedge(tj,t(Suc j))\inR by blas
    from choice [OF this] obtain f where R-steps: \foralli.i\leqfi\wedge(t (fi),t (Suc(f
i))) }\inR
    let ?t = \lambdai.t (((Suc\circf) ~ i) 0)
    have }\foralli.(t i,t(Suc (fi)))\in(\mathrm{ relto R S)}\mp@subsup{)}{}{+
    proof
        fix }
        from R-steps have leq: i\leqfi and step: (t(fi),t(Suc(fi))) \inR by auto
        from chain-imp-rtrancl [OF chain leq] have (t i,t(fi)) \in(R\cupS)*.
        with step have (t i,t(Suc(fi))) \in(R\cupS)* O R by auto
        then show (ti,t(Suc(fi))) \in(relto R S)+}\mathrm{ by regexp
    qed
    then have chain ((relto R S)+}) ?t by sim
    with t0 have }\neg\mathrm{ SN-on ((relto R S)+) X by (unfold SN-on-def, auto intro: exI[of
- ?t])
    with SN-on-trancl[OF SN] show False by auto
qed
from Geser's thesis [p.32, Corollary-1], generalized for \(S N\)-on.
lemma \(S N\)-on-relto-Un:
    assumes closure: relto (R\cupR')S"X\subseteqX
    shows SN-on (relto (R\cup\mp@subsup{R}{}{\prime})S)X\longleftrightarrowSN-on (relto R (R'\cupS)) X ^SN-on
(relto R'S) X
    (is ?c}\longleftrightarrow\longleftrightarrow?a\wedge?b
proof(safe)
    assume SN:?a and SN':?b
```

```
from SN have SN: SN-on (relto (relto R S) (relto R'S))X by (rule SN-on-subset1)
regexp
    show ?c
    proof
        fix f
        assume f0: f 0 \in X and chain: chain (relto (R\cup R')S) f
        then have chain (relto RS\cup relto R'S) f by auto
        from relative-ending[OF this fO SN]
        have }\existsj.\foralli\geqj.(fi,f(Suc i))\in relto R'S - relto R S by aut
        then obtain j where }\foralli\geqj.(fi,f(Suc i))\in\mathrm{ relto R'S by auto
        then have chain (relto R'S) (shift f j) by auto
        moreover have fj\inX
        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 f relto ( }R\cup\mp@subsup{R}{}{\prime})S\mathrm{ by auto
            with Image-closed-trancl[OF closure] Suc show f(Suc j) \inX by blast
    qed
    then have shift fj0\inX by auto
    ultimately have }\negSN\mathrm{ -on (relto }\mp@subsup{R}{}{\prime}S\mathrm{ ) X by (intro not-SN-onI)
    with }S\mp@subsup{N}{}{\prime}\mathrm{ show False by auto
    qed
next
    assume SN: ?c
    then show ?b by (rule SN-on-subset1, auto)
    moreover
        from SN have SN-on ((relto (R\cupR')S\mp@subsup{)}{}{+})X X by (unfold SN-on-trancl-SN-on-conv)
        then show ?a by (rule SN-on-subset1) regexp
qed
lemma SN-on-Un: (R\cupR')"X\subseteqX\LongrightarrowSN-on (R\cupR')X\longleftrightarrowSN-on (relto R
R') X ^SN-on R'X
    using SN-on-relto-Un[of {}] by simp
end
```


## 4 Strongly Normalizing Orders

theory $S N$-Orders
imports Abstract-Rewriting
begin
We define several classes of orders which are used to build ordered semirings. Note that we do not use Isabelle's preorders since the condition $x>y=x \geq y \wedge y \nsupseteq x$ is sometimes not applicable. E.g., for $\delta$-orders over the rationals we have $0.2 \geq 0.1 \wedge 0.1 \nsupseteq 0.2$, but $0.2>_{\delta} 0.1$ does not hold if $\delta$ is larger than 0.1.

```
class non-strict-order = ord +
    assumes ge-refl: x \geq (x :: 'a)
    and ge-trans[trans]:\llbracketx\geqy;(y::' }a)\geqz\rrbracket\Longrightarrowx\geq
    and max-comm: max x y = max y x
    and max-ge-x[intro]: max x y \geqx
    and max-id: x \geq y \Longrightarrow max x y = x
    and max-mono: }x\geqy\Longrightarrow\operatorname{max}zx\geq\operatorname{max}z
begin
lemma max-ge-y[intro]: max x y \geqy
    unfolding max-comm[of x y] ..
lemma max-mono2: x \geq y \Longrightarrow max }xz\geq\operatorname{max}y
    unfolding max-comm[of-z] by (rule max-mono)
end
class ordered-ab-semigroup = non-strict-order }+\mathrm{ ab-semigroup-add }+\mathrm{ monoid-add
+
    assumes plus-left-mono: }x\geqy\Longrightarrowx+z\geqy+
lemma plus-right-mono: y \geq(z :: 'a :: ordered-ab-semigroup) \Longrightarrowx+y\geqx+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\geq0\Longrightarrowx\geqy\Longrightarrowx*z\geqy*z
    and times-right-mono: }x\geq0\Longrightarrowy\geqz\Longrightarrowx*y\geqx*
    and times-left-anti-mono: }x\geqy\Longrightarrow0\geqz\Longrightarrowy*z\geqx*
class ordered-semiring-1 = ordered-semiring-0 + semiring-1 +
    assumes one-ge-zero: 1 \geq0
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 \(=\)
```

locale order-pair $=$
fixes gt :: ' $a$ :: \{non-strict-order,zero $\} \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infix $\succ$ 50)
fixes gt :: ' $a$ :: \{non-strict-order,zero $\} \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infix $\succ$ 50)
and default :: 'a
and default :: 'a
assumes compat[trans] $\llbracket x \geq y ; y \succ z \rrbracket \Longrightarrow x \succ z$
assumes compat[trans] $\llbracket x \geq y ; y \succ z \rrbracket \Longrightarrow x \succ z$
and compat2[trans]: $\llbracket x \succ y ; y \geq z \rrbracket \Longrightarrow x \succ z$
and compat2[trans]: $\llbracket x \succ y ; y \geq z \rrbracket \Longrightarrow x \succ z$
and gt-imp-ge: $x \succ y \Longrightarrow x \geq y$
and gt-imp-ge: $x \succ y \Longrightarrow x \geq y$
and default-ge-zero: default $\geq 0$
and default-ge-zero: default $\geq 0$
begin
begin
lemma gt-trans[trans]: $\llbracket x \succ y ; y \succ z \rrbracket \Longrightarrow x \succ z$
lemma gt-trans[trans]: $\llbracket x \succ y ; y \succ z \rrbracket \Longrightarrow x \succ z$
by (rule compat [OF gt-imp-ge])
by (rule compat [OF gt-imp-ge])
end
end
locale one-mono-ordered-semiring-1 $=$ order-pair gt
locale one-mono-ordered-semiring-1 $=$ order-pair gt
for $g t::$ ' $a$ :: ordered-semiring- $1 \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infix $\left.\succ 50\right)+$
for $g t::$ ' $a$ :: ordered-semiring- $1 \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infix $\left.\succ 50\right)+$
assumes plus-gt-left-mono: $x \succ y \Longrightarrow x+z \succ y+z$
assumes plus-gt-left-mono: $x \succ y \Longrightarrow x+z \succ y+z$
and default-gt-zero: default $\succ 0$

```
    and default-gt-zero: default \(\succ 0\)
```

```
begin
lemma plus-gt-right-mono: }x\succy\Longrightarrowa+x\succa+
    unfolding add.commute[of a] by (rule plus-gt-left-mono)
lemma plus-gt-both-mono: }x\succy\Longrightarrowa\succb\Longrightarrowx+a\succy+
    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\geq0^x\succy}
locale SN-strict-mono-ordered-semiring-1 = SN-one-mono-ordered-semiring-1 +
    fixes mono :: 'a :: ordered-semiring-1 # bool
    assumes mono:\llbracketmono }x;y\succz;x\geq0\rrbracket\Longrightarrowx*y\succx*
locale both-mono-ordered-semiring-1 = order-pair gt
    for gt :: ' }a::\mathrm{ ordered-semiring-1 }\mp@subsup{|}{}{\prime}'a=>\mathrm{ bool (infix }\succ50)
    fixes arc-pos :: ' }a=>\mathrm{ bool
    assumes plus-gt-both-mono: \llbracketx\succy;z\succu\rrbracket\Longrightarrowx+z\succy+u
    and times-gt-left-mono: }x\succy\Longrightarrowx*z\succy*
    and times-gt-right-mono: }y\succz\Longrightarrowx*y\succx*
    and zero-leastI: x\succ0
    and zero-leastII: 0}\succ>>x=
    and zero-leastIII: (x :: 'a)\geq0
    and arc-pos-one: arc-pos (1 :: 'a)
    and arc-pos-default: arc-pos default
    and arc-pos-zero: ᄀ arc-pos 0
    and arc-pos-plus: arc-pos x \Longrightarrowarc-pos (x+y)
    and arc-pos-mult:\llbracketarc-pos x; arc-pos y\rrbracket \Longrightarrow arc-pos (x*y)
    and not-all-ge: \bigwedge cd. arc-pos d\Longrightarrow\existse. e\geq0^ arc-pos e ^\neg(c\geqd*e)
begin
lemma max0-id: max 0 (x :: 'a) = x
    unfolding max-comm[of 0]
    by (rule max-id[OF zero-leastIII])
end
locale SN-both-mono-ordered-semiring-1 = both-mono-ordered-semiring-1 +
    assumes SN:SN {(x,y) . arc-pos y ^x\succy}
locale weak-SN-strict-mono-ordered-semiring-1 =
    fixes weak-gt :: ' }a\mathrm{ :: ordered-semiring-1 }=>\mp@subsup{}{}{\prime}a=>\mathrm{ bool
        and default :: 'a
        and mono :: 'a }=>\mathrm{ bool
        assumes weak-gt-mono: }\forallxy.(x,y)\in\mathrm{ set xys }\longrightarrow\mathrm{ weak-gt x y }\Longrightarrow\existsg\mathrm{ gt.
SN-strict-mono-ordered-semiring-1 default gt mono ^ ( }\forallxy.(x,y)\in\mathrm{ set xys }
gt x y)
```

```
locale weak-SN-both-mono-ordered-semiring-1 =
    fixes weak-gt :: 'a :: ordered-semiring-1 \(\Rightarrow^{\prime} a \Rightarrow\) bool
        and default :: ' \(a\)
        and arc-pos :: ' \(a \Rightarrow\) bool
    assumes weak-gt-both-mono: \(\forall x y .(x, y) \in\) set xys \(\longrightarrow\) weak-gt \(x y \Longrightarrow \exists g t\).
SN-both-mono-ordered-semiring-1 default gt arc-pos \(\wedge(\forall x y .(x, y) \in\) set \(x y s \longrightarrow\)
gt \(x\) y)
class poly-carrier \(=\) ordered-semiring- \(1+\) comm-semiring- 1
locale poly-order-carrier \(=S N\)-one-mono-ordered-semiring-1 default gt
    for default :: ' \(a\) :: poly-carrier and \(g t(\) infix \(\succ 50)+\)
    fixes power-mono :: bool
    and discrete :: bool
    assumes times-gt-mono: \(\llbracket y \succ z ; x \geq 1 \rrbracket \Longrightarrow y * x \succ z * x\)
    and power-mono: power-mono \(\Longrightarrow x \succ y \Longrightarrow y \geq 0 \Longrightarrow n \geq 1 \Longrightarrow x^{\wedge} n \succ y\)
\({ }^{\wedge} n\)
    and discrete: discrete \(\Longrightarrow x \geq y \Longrightarrow \exists k . x=\left(((+) 1)^{\wedge} k\right) y\)
class large-ordered-semiring-1 \(=\) poly-carrier +
    assumes ex-large-of-nat: \(\exists x\). of-nat \(x \geq y\)
context ordered-semiring-1
begin
lemma pow-mono: assumes \(a b: a \geq b\) and \(b: b \geq 0\)
    shows \(a^{\wedge} n \geq b{ }^{\wedge} n \wedge b{ }^{\wedge} n \geq 0\)
proof (induct \(n\) )
    case 0
    show ?case by (auto simp: ge-refl one-ge-zero)
next
    case (Suc n)
    hence \(a b n: a^{\wedge} n \geq b^{\wedge} n\) and \(b n: b{ }^{\wedge} n \geq 0\) by auto
    have bsn: \(b\) ^ Suc \(n \geq 0\) unfolding power-Suc
        using times-left-mono[OF bn b] by auto
    have \(a\) ^Suc \(n=a * a{ }^{\wedge} n\) unfolding power-Suc by simp
    also have \(\ldots \geq b * a{ }^{\wedge} n\)
        by (rule times-left-mono[OF ge-trans[OF abn bn] ab])
    also have \(b * a \wedge n \geq b * b\) ^ \(n\)
        by (rule times-right-mono[OF babn])
    finally show ?case using bsn unfolding power-Suc by simp
qed
lemma pow-ge-zero[intro]: assumes \(a: a \geq(0::\) ' \(a)\)
    shows \(a^{\wedge} n \geq 0\)
proof (induct \(n\) )
    case 0
    from one-ge-zero show ?case by simp
next
    case (Suc n)
```

show ?case using times-left-mono[OF Suc a] by simp
qed
end
lemma of-nat-ge-zero[intro,simp]: of-nat $n \geq(0::$ ' $a$ :: 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$) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a *$ $b \geq 0$
using times-left-mono[of blall by auto
lemma pow-mono-one: assumes $a: a \geq(1:: ' a$ :: ordered-semiring-1)
shows $a^{\wedge} n \geq 1$
proof (induct $n$ )
case (Suc n)
show ?case unfolding power-Suc
using ge-trans[OF times-right-mono[OF ge-trans[OF a one-ge-zero] Suc], of 1] a
by (auto simp: field-simps)
qed (auto simp: ge-refl)
lemma pow-mono-exp: assumes $a: a \geq(1:: ' a::$ ordered-semiring-1)
shows $n \geq m \Longrightarrow a{ }^{\wedge} n \geq a^{\wedge} m$
proof (induct $m$ arbitrary: $n$ )
case 0
show ? case using pow-mono-one $[O F a]$ by auto
next
case (Suc mnn)
then obtain $n$ where $n n$ : $n n=$ Suc $n$ by (cases $n n$, auto)
note Suc $=$ Suc[unfolded $n n$ ]
hence rec: $a^{\wedge} n \geq a へ m$ 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 $a:\left(a::{ }^{\prime} a::\right.$ ordered-semiring- 1$) \geq 1$
and $b: b \geq 1$
shows $a * b \geq 1$
proof -
from ge-trans $[$ OF $b$ one-ge-zero $]$ have $b 0: b \geq 0$.
from times-left-mono[OF b0 a] have $a * b \geq b$ by simp

```
    from ge-trans[OF this b] show ?thesis .
qed
lemma sum-list-ge-mono: fixes as :: ('a :: ordered-semiring-0) list
    assumes length as = length bs
    and }\bigwedgei.i<length bs \Longrightarrowas!i\geqbs!
    shows sum-list as \geq sum-list bs
    using assms
proof (induct as arbitrary: bs)
    case (Nil bs)
    from Nil(1) show ?case by (simp add: ge-refl)
next
    case (Cons a as bbs)
    from Cons(2) obtain b bs where bbs: bbs = b # bs and len: length as = length
bs by (cases bbs,auto)
    note ge = Cons(3)[unfolded bbs]
    {
        fix }
        assume i< length bs
        hence Suc i< length ( b# bs) by simp
        from ge[OF this] have as!i\geqbs!i by simp
    }
    from Cons(1)[OF len this] have IH: sum-list as \geq sum-list bs .
    from ge[of 0] have ab:a\geqb by simp
    from ge-trans[OF plus-left-mono[OF ab] plus-right-mono[OF IH]]
    show ?case unfolding bbs by simp
qed
lemma sum-list-ge-0-nth: fixes xs :: ('a :: ordered-semiring-0)list
    assumes ge: \ i. i< length xs \Longrightarrowxs!i\geq0
    shows sum-list xs \geq0
proof -
    let ?l = replicate (length xs) (0 :: 'a)
    have length xs = length ?l by simp
    from sum-list-ge-mono[OF this] ge have sum-list xs \geq sum-list ?l by simp
    also have sum-list ?l = 0 using sum-list-0[of ?l] by auto
    finally show ?thesis.
qed
lemma sum-list-ge-0: fixes xs :: (' }a\mathrm{ :: ordered-semiring-0)list
    assumes ge: \bigwedgex. x\in set xs \Longrightarrowx\geq0
    shows sum-list xs \geq0
    by (rule sum-list-ge-0-nth, insert ge[unfolded set-conv-nth], auto)
lemma foldr-max: a set as \Longrightarrow 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 \geqa by auto
    from ge-trans[OF - this] show ?thesis by auto
    qed
qed
```

lemma of-nat-mono[intro]: assumes $n \geq m$ shows (of-nat $n::{ }^{\prime} a::$ ordered-semiring- 1 )
$\geq$ of-nat $m$
proof -
let $? n=$ of-nat $::$ nat $\Rightarrow{ }^{\prime} a$
from assms
show ?thesis
proof (induct $m$ arbitrary: $n$ )
case 0
show ?case by auto
next
case (Suc mnn)
then obtain $n$ where $n n$ : $n n=S u c n$ by (cases nn, auto)
note $S u c=$ Suc[unfolded $n n$ ]
hence rec: ? $n n \geq$ ? $n m$ by simp
show ?case unfolding nn of-nat-Suc
by (rule plus-right-mono $[$ OF rec $]$ )
qed
qed
non infinitesmal is the same as in the CADE07 bounded increase paper
definition non-inf :: 'a rel $\Rightarrow$ bool
where non-inf $r \equiv \forall a f . \exists i .(f i, f(S u c i)) \notin r \vee(f i, a) \notin r$
lemma non-infI[intro]: assumes $\wedge a f . \llbracket \bigwedge i .(f i, f(S u c i)) \in r \rrbracket \Longrightarrow \exists i .(f i$,
a) $\notin r$
shows non-inf $r$
using assms unfolding non-inf-def by blast
lemma non-infE[elim]: assumes non-inf $r$ and $\bigwedge i .(f i, f(S u c i)) \notin r \vee(f i$,
a) $\notin r \Longrightarrow P$
shows $P$
using assms unfolding non-inf-def by blast
lemma non-inf-image:
assumes ni: non-inf $r$ and image: $\bigwedge a b .(a, b) \in s \Longrightarrow(f a, f b) \in r$
shows non-inf s
proof
fix $a g$

```
    assume \(s: \bigwedge i .(g i, g(\) Suc \(i)) \in s\)
    define \(h\) where \(h=f \circ g\)
    from image \([O F s]\) have \(h: \wedge i\). \((h i, h(S u c i)) \in r\) 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
qed
lemma \(S N\)-imp-non-inf: \(S N r \Longrightarrow\) non-inf \(r\)
    by (intro non-infI, auto)
lemma non-inf-imp-SN-bound: non-inf \(r \Longrightarrow S N\{(a, b) .(b, c) \in r \wedge(a, b) \in r\}\)
    by (rule, auto)
end
```


## 5 Carriers of Strongly Normalizing Orders

theory SN-Order-Carrier imports
SN-Orders
HOL.Rat
begin
This theory shows that standard semirings can be used in combination with polynomials, e.g. the naturals, integers, and arbitrary Archemedean fields by using delta-orders.

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

### 5.1 The standard semiring over the naturals

instantiation nat :: large-ordered-semiring-1
begin
instance by (intro-classes, auto)
end
definition nat-mono $::$ nat $\Rightarrow$ bool where nat-mono $x \equiv x \neq 0$
interpretation nat-SN: SN-strict-mono-ordered-semiring-1 1 ( $>$ ) :: nat $\Rightarrow$ nat $\Rightarrow$ bool nat-mono
by (unfold-locales, insert SN-nat-gt, auto simp: nat-mono-def)
interpretation nat-poly: poly-order-carrier 1 ( $>$ ) :: nat $\Rightarrow$ nat $\Rightarrow$ bool True discrete
proof (unfold-locales)
fix $x y$ :: nat
assume $g e$ : $x \geq y$
obtain $k$ where $k: x-y=k$ by auto

```
    show \exists k. x = ((+) 1 ^~k) y
    proof (rule exI[of-k])
    from ge k have }x=k+y\mathrm{ by simp
    also have ... =((+) 1 ~~ k) y
        by (induct k, auto)
    finally show }x=((+)\mp@subsup{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 $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow$ bool where delta-gt $\delta \equiv(\lambda x y . x-y \geq \delta)$
lemma non-inf-delta-gt: assumes delta: $\delta>0$ shows non-inf $\{(a, b)$. delta-gt $\delta a b\}$ (is non-inf ? $r$ )
proof
let $? g t=$ delta-gt $\delta$
fix $a::^{\prime} a$ and $f$
assume $\wedge i .(f i, f(S u c i)) \in ? r$
hence $g t$ : $\bigwedge i$. ? gt (fi) (f (Suc i)) by simp
\{
fix $i$
have $f i \leq f 0-\delta *$ of-nat $i$
proof (induct $i$ )
case (Suc i)
thus ?case using $g t[$ of $i$, unfolded delta-gt-def] by (auto simp: field-simps)
qed $\operatorname{simp}$
\} note $f i=$ this
\{
fix $r::$ ' $a$
have of-nat (nat (ceiling r)) $\geq r$
by (metis ceiling-le-zero le-of-int-ceiling less-le-not-le nat-0-iff not-less of-nat-0 of-nat-nat)
\} note ceil-elim $=$ this
define $i$ where $i=$ nat $($ ceiling $((f 0-a) / \delta))$
from $f i[$ of $i]$ have $f i-f 0 \leq-\delta *$ of-nat (nat (ceiling $((f 0-a) / \delta))$ )
unfolding $i$-def by simp
also have $\ldots \leq-\delta *((f 0-a) / \delta)$ using ceil-elim $[$ of $(f 0-a) / \delta]$ delta
by (metis le-imp-neg-le minus-mult-commute mult-le-cancel-left-pos)
also have $\ldots=-f 0+a$ 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 $S N\{(x, y) .0 \leq y \wedge$ delta-gt $\delta$ $x y\}$

```
proof -
    from non-inf-imp-SN-bound[OF non-inf-delta-gt[OF dpos], of - \delta]
    show ?thesis unfolding delta-gt-def by auto
qed
definition delta-mono :: 'a :: floor-ceiling => bool where delta-mono }x\equivx\geq
subclass (in floor-ceiling) large-ordered-semiring-1
proof
    fix }x:: '
    from ex-le-of-int[of x] obtain z}\mathrm{ where }x:x\leqof-int z by aut
    have z\leqint (nat z) by auto
    with }x\mathrm{ have }x\leqof\mathrm{ -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 \exists y. x\leqof-nat y by blast
qed (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg max-def)
lemma delta-interpretation: assumes dpos: \delta>0 and default: }\delta\leq\mathrm{ def
    shows SN-strict-mono-ordered-semiring-1 def (delta-gt \delta) delta-mono
proof -
    from dpos default have defz: 0 \leq def by auto
    show ?thesis
    proof (unfold-locales)
        show SN {(x,y). y \geq0^delta-gt \delta x y} by (rule delta-gt-SN[OF dpos])
    next
        fix x y z :: 'a
        assume delta-mono x and yz: delta-gt \delta y z
        hence x: 1\leqx unfolding delta-mono-def by simp
    have }\existsd>0. delta-gt \delta = ( \lambdaxy.d\leqx-y
            by (rule exI[of-\delta], auto simp: dpos delta-gt-def)
    from this obtain d where d:0<d and rat: delta-gt \delta=(\lambdaxy.d\leqx-y)
by auto
    from yz have yzd: d\leqy-z by (simp add: rat)
    show delta-gt \delta (x*y)(x*z)
    proof (simp only: rat)
        let ?p }=(x-1)*(y-z
        from x have x1:0\leqx-1 by auto
        from yzd d have yz0:0\leqy-z by auto
        have 0\leq?p
            by (rule mult-nonneg-nonneg[OF x1 yz0])
        have }x*y-x*z=x*(y-z) using right-diff-distrib[of x y z] by aut
        also have \ldots=((x-1)+1)*(y-z) by auto
        also have \ldots. =?p+1*(y-z) by (rule ring-distribs(2))
        also have \ldots=? p + (y-z) by simp
        also have }\ldots\geq(0+d)\mathrm{ using yzd <0 {? p` by auto
```

```
        finally
        show }d\leqx*y-x*z\mathrm{ by auto
        qed
    qed (insert dpos, auto simp: delta-gt-def default defz)
qed
lemma delta-poly: assumes dpos: }\delta>0\mathrm{ and default: }\delta\leq\mathrm{ def
    shows poly-order-carrier def (delta-gt \delta) (1\leq\delta) False
proof -
    from delta-interpretation[OF dpos default]
    interpret SN-strict-mono-ordered-semiring-1 def delta-gt \delta delta-mono .
    interpret poly-order-carrier def delta-gt \delta False False
    proof(unfold-locales)
            fix yzx :: 'a
            assume gt:delta-gt \delta y z and ge: x\geq1
            from ge have ge: x \geq0 and m:delta-mono x unfolding delta-mono-def by
auto
            show delta-gt \delta (y*x) (z*x)
                    using mono[OF m gt ge] by (auto simp: field-simps)
    next
            fix }xy ::' 'a and n :: na
            assume False thus delta-gt \delta (x^n) (y^n) ..
    next
            fix x y :: 'a
            assume False
            thus \exists k.x = ((+) 1 ~ k) y by simp
    qed
    show ?thesis
    proof(unfold-locales)
            fix x y :: ' }a\mathrm{ and n :: nat
            assume one: }1\leq\delta\mathrm{ and gt:delta-gt }\deltaxy\mathrm{ and }y:y\geq0\mathrm{ and n: 1 
            then obtain p where n: n = Suc p and x:x\geq1 and y2:0\leqy and xy:x
\geqy 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^ Suc (Suc p)) (x* y^ Suc p) by (auto simp:
field-simps)
            also have ...\geqy*y^ 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 \existsk. x = ((+) 1 ^~ k) y by simp
    qed (rule times-gt-mono, auto)
```

lemma delta-minimal-delta: assumes $\bigwedge x y .(x, y) \in$ set xys $\Longrightarrow x>y$

$$
\text { shows } \exists \delta>0 . \forall x y .(x, y) \in \text { set xys } \longrightarrow \text { delta-gt } \delta x y
$$

using assms
proof (induct xys)
case Nil
show ? case by (rule exI[of-1], auto)
next
case (Cons xy xys)
show ?case
proof (cases xy)
case (Pair x y)
with Cons have $x>y$ by auto
then obtain $d 1$ where $d 1=x-y$ and d1pos: $d 1>0$ and $d 1 \leq x-y$ by auto
hence xy: delta-gt d1 $x y$ unfolding delta-gt-def by auto
from Cons obtain $d 2$ where d2pos: $d 2>0$ and xys: $\forall x y .(x, y) \in$ set xys
$\longrightarrow$ delta-gt d2 $x y$ by auto
obtain $d$ where $d: d=\min d 1 d 2$ by auto
with d1pos d2pos $x y$ have dpos: $d>0$ and delta-gt $d x y$ unfolding delta-gt-def

## by auto

with xys d Pair have $\forall x y .(x, y) \in \operatorname{set}(x y \# x y s) \longrightarrow$ delta-gt dxy unfolding delta-gt-def by force
with dpos show ?thesis by auto
qed
qed
interpretation weak-delta-SN: weak-SN-strict-mono-ordered-semiring-1 (>) 1 delta-mono proof
fix xysp :: ( ${ }^{\prime} a \times$ 'a) list
assume orient: $\forall x y .(x, y) \in$ set $x y s p \longrightarrow x>y$
obtain xys where xsy: xys $=(1,0) \#$ xysp by auto
with orient have $\wedge x y .(x, y) \in$ set xys $\Longrightarrow x>y$ by auto
with delta-minimal-delta have $\exists \delta>0 . \forall x y .(x, y) \in$ set xys $\longrightarrow$ delta-gt $\delta x$ $y$ by auto
then obtain $\delta$ where dpos: $\delta>0$ and orient: $\bigwedge x y .(x, y) \in$ set xys $\Longrightarrow$ delta-gt $\delta x y$ by auto
from orient have orient1: $\forall x y .(x, y) \in$ set xysp $\longrightarrow$ delta-gt $\delta x y$ and orient2: delta-gt $\delta 10$ unfolding xsy by auto
from orient2 have oned: $\delta \leq 1$ unfolding delta-gt-def by auto
show $\exists \mathrm{gt}$. SN-strict-mono-ordered-semiring-1 1 gt delta-mono $\wedge(\forall x y .(x, y)$ $\in$ set $x y s p \longrightarrow g t x y)$
by (intro exI conjI, rule delta-interpretation[OF dpos oned], rule orient1) qed

### 5.3 The standard semiring over the integers

definition int-mono :: int $\Rightarrow$ bool where int-mono $x \equiv x \geq 1$
instantiation int :: large-ordered-semiring-1
begin
instance
proof
fix $y$ :: int
show $\exists x$. of-nat $x \geq y$
by (rule exI [of - nat $y]$, simp)
qed (auto simp: mult-right-mono mult-left-mono mult-right-mono-neg)
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 $\Rightarrow$ int $\Rightarrow$
bool int-mono
proof (unfold-locales)
have $[$ simp $]: \bigwedge x::$ int $.(-1<x)=(0 \leq x)$ by auto
show $S N\{(x, y) . y \geq 0 \wedge(y::$ 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 $\Rightarrow$ int $\Rightarrow$ bool True discrete proof (unfold-locales)
fix $x y$ :: int
assume ge: $x \geq y$
then obtain $k$ where $k: x-y=k$ and $k p: 0 \leq k$ by auto
then obtain $n k$ where $n k: n k=n a t k$ and $k: x-y=$ int $n k$ by auto
show $\exists k . x=\left((+) 1^{\sim} k\right) y$
proof (rule exI[of-nk])
from $k$ have $x=$ int $n k+y$ by $\operatorname{simp}$
also have $\ldots=\left((+) 1^{\sim} n k\right) y$
by (induct nk, auto)
finally show $x=\left((+) 1^{\sim} n k\right) y$.
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 $\mid$ Num-arc int
instantiation arctic :: ord
begin
fun less-eq-arctic $::$ arctic $\Rightarrow$ arctic $\Rightarrow$ 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\leqx)
fun less-arctic :: arctic }=>\mathrm{ arctic }=>\mathrm{ 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 }=>\mathrm{ arctic }=>\mathrm{ 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 }=>\mathrm{ arctic }=>\mathrm{ 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
```

instance
proof
proof
fix $x y z::$ arctic
fix $x y z::$ arctic
show $x+y=y+x$
show $x+y=y+x$
by (cases $x$, cases $y$, auto, cases $y$, auto)
by (cases $x$, cases $y$, auto, cases $y$, auto)
show $(x+y)+z=x+(y+z)$
show $(x+y)+z=x+(y+z)$
by (cases $x$, auto, cases $y$, auto, cases $z$, auto)
by (cases $x$, auto, cases $y$, auto, cases $z$, auto)
show $(x * y) * z=x *(y * z)$
show $(x * y) * z=x *(y * z)$
by (cases $x$, auto, cases $y$, auto, cases $z$, auto)
by (cases $x$, auto, cases $y$, auto, cases $z$, auto)
show $x * 0=0$
show $x * 0=0$
by (cases $x$, auto simp: zero-arctic-def)
by (cases $x$, auto simp: zero-arctic-def)
show $x *(y+z)=x * y+x * z$
show $x *(y+z)=x * y+x * z$
by (cases $x$, auto, cases $y$, auto, cases $z$, auto)
by (cases $x$, auto, cases $y$, auto, cases $z$, auto)
show $(x+y) * z=x * z+y * z$
show $(x+y) * z=x * z+y * z$
by (cases $x$, auto, cases $y$, cases $z$, auto, cases $z$, auto)
by (cases $x$, auto, cases $y$, cases $z$, auto, cases $z$, auto)
show $1 * x=x$
show $1 * x=x$
by (cases $x$, simp-all add: one-arctic-def)
by (cases $x$, simp-all add: one-arctic-def)
show $x * 1=x$
show $x * 1=x$
by (cases $x$, simp-all add: one-arctic-def)

```
    by (cases \(x\), simp-all add: one-arctic-def)
```

```
    show 0 + x = x
    by (simp add: zero-arctic-def)
show 0*x=0
    by (simp add: zero-arctic-def)
show (0 :: arctic) # 1
    by (simp add: zero-arctic-def one-arctic-def)
show }x+0=x\mathrm{ by (cases }x\mathrm{ , auto simp:zero-arctic-def)
show }x\geq
    by (cases x, auto)
show (1 :: arctic) \geq0
    by (simp add:zero-arctic-def one-arctic-def)
show max x y = max y x unfolding max-def
    by (cases }x,(\mathrm{ cases y, auto)+)
show max x y \geqx unfolding max-def
    by (cases x, (cases y, auto)+)
assume ge: x \geqy
from ge show }x+z\geqy+
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
from ge show }x*z\geqy*
    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 zx\geq max zy unfolding max-def
    by (cases z, cases x, auto, cases x, (cases y, auto)+)
next
    fix x y z :: arctic
    assume }x\geqy\mathrm{ and }y\geq
    thus }x\geq
    by (cases x, cases y, auto, cases y, cases z, auto, cases z, auto)
next
    fix x y z :: arctic
    assume }y\geq
    thus }x*y\geqx*
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
next
    fix x y z :: arctic
    show }x\geqy\Longrightarrow0\geqz\Longrightarrowy*z\geqx*
    by (cases z, cases x, auto simp: zero-arctic-def)
qed
end
```

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

```
interpretation arctic-SN: SN-both-mono-ordered-semiring-1 1 (>) pos-arctic
proof
    fix \(x y z\) :: arctic
    assume \(x \geq y\) and \(y>z\)
    thus \(x>z\)
        by (cases \(z\), simp, cases \(y\), simp, cases \(x\), auto)
next
    fix \(x\) y \(z::\) arctic
    assume \(x>y\) and \(y \geq z\)
    thus \(x>z\)
        by (cases \(z, \operatorname{simp}\), cases \(y\), simp, cases \(x\), auto)
next
    fix \(x y z\) :: arctic
    assume \(x>y\)
    thus \(x \geq y\)
        by (cases \(x\), (cases \(y\), auto) + )
next
    fix \(x y z u::\) arctic
    assume \(x>y\) and \(z>u\)
    thus \(x+z>y+u\)
        by (cases \(y\), cases \(u\), simp, cases \(z\), auto, cases \(x\), auto, cases \(u\), auto, cases \(z\),
auto, cases \(x\), auto, cases \(x\), auto, cases \(z\), auto, cases \(x\), auto)
next
    fix \(x\) y \(z::\) arctic
    assume \(x>y\)
    thus \(x * z>y * z\)
        by (cases \(y\), simp, cases \(z\), simp, cases \(x\), auto)
next
    fix \(x\) :: arctic
    assume \(0>x\)
    thus \(x=0\)
        by (cases \(x\), auto simp: zero-arctic-def)
next
    fix \(x\) :: arctic
    show pos-arctic 1 unfolding one-arctic-def by simp
    show \(x>0\) unfolding zero-arctic-def by simp
    show ( \(1::\) arctic) \(\geq 0\) unfolding zero-arctic-def by simp
    show \(x \geq 0\) unfolding zero-arctic-def by simp
    show \(\neg\) pos-arctic 0 unfolding zero-arctic-def by simp
next
    fix \(x y\)
    assume pos-arctic \(x\)
    thus pos-arctic \((x+y)\) by (cases \(x\), simp, cases \(y\), auto)
next
    fix \(x y\)
    assume pos-arctic \(x\) and pos-arctic \(y\)
    thus pos-arctic \((x * y)\) by (cases \(x\), simp, cases \(y\), auto)
next
```

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

    proof - \{
    fix \(x\)
    assume \(\exists f . f 0=x \wedge(\forall i .(f i, f(\) 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: $\forall i . f i>f(S u c i) \wedge \operatorname{pos-arctic}(f(S u c i))$ by auto
let $? g=\lambda i$. get-arctic-num $(f i)$
have $\forall i$. ?g $($ Suc $i) \geq 0 \wedge$ ?g $i>$ ?g (Suc $i)$
proof
fix $i$
from steps have $i$ : $f i>f(S u c i) \wedge$ pos-arctic $(f(S u c i))$ by auto
from $i$ obtain $n$ where $f$ : $f i=N u m$-arc $n$ by (cases $f$ (Suc i), simp, cases
$f i$, auto)
from $i$ obtain $m$ where $f s i: f(S u c i)=$ Num-arc $m$ by (cases $f(S u c i)$,
auto)
with $i$ have $g z: 0 \leq m$ by simp
from $i f$ fsi have $n>m$ by auto
with $f$ fsi $g z$
show ?g $($ Suc $i) \geq 0 \wedge$ ?g $i>? g($ Suc $i)$ by auto
qed
from this obtain $g$ where $\forall$ i. $g($ Suc $i) \geq 0 \wedge((>)::$ int $\Rightarrow$ int $\Rightarrow$ bool $)(g$
i) $(g$ (Suc $i))$ by auto
hence $\exists f . f 0=g 0 \wedge(\forall i .(f i, f($ Suc $i)) \in\{(x, y) . y \geq 0 \wedge x>y\})$ by
auto
with int-SN.SN have False unfolding $S N$-defs by auto
\}
thus ?thesis unfolding $S N$-defs by auto
qed
next
fix $y z x$ :: arctic
assume $y>z$
thus $x * y>x * z$
by (cases $x$, simp, cases $z$, simp, cases $y$, auto)
next
fix $c d$
assume pos-arctic $d$
then obtain $n$ where $d: d=$ Num-arc $n$ and $n: 0 \leq n$
by (cases d, auto)
show $\exists e . e \geq 0 \wedge$ pos-arctic $e \wedge \neg c \geq d * e$
proof (cases c)
case MinInfty
show ?thesis
by (rule exI[of - Num-arc 0],
unfold d MinInfty zero-arctic-def, simp)
next
case (Num-arc m)
show ?thesis
by (rule exI[of-Num-arc (abs $m+1$ )], insert $n$,
unfold d Num-arc zero-arctic-def, simp)
qed
qed

### 5.5 The arctic semiring over an arbitrary archimedean field

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

```
datatype 'a arctic-delta = MinInfty-delta | Num-arc-delta 'a
instantiation arctic-delta :: (ord) ord
begin
fun less-eq-arctic-delta :: 'a arctic-delta }=>\mathrm{ 'a arctic-delta }=>\mathrm{ 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\leqx)
fun less-arctic-delta :: 'a arctic-delta # 'a arctic-delta }=>\mathrm{ bool where
    less-arctic-delta MinInfty-delta x = True
| less-arctic-delta (Num-arc-delta -) MinInfty-delta = False
| less-arctic-delta (Num-arc-delta y)(Num-arc-delta x)=(y<x)
```


## instance ..

end
instantiation arctic-delta :: (linordered-field) ordered-semiring-1
begin
fun plus-arctic-delta $::$ ' $a$ arctic-delta $\Rightarrow$ ' $a$ arctic-delta $\Rightarrow$ ' $a$ arctic-delta where
plus-arctic-delta MinInfty-delta $y=y$
| plus-arctic-delta x MinInfty-delta $=x$
| plus-arctic-delta (Num-arc-delta $x)$ (Num-arc-delta $y)=($ Num-arc-delta $(\max x$
y))
fun times-arctic-delta $::$ ' $a$ arctic-delta $\Rightarrow$ ' $a$ arctic-delta $\Rightarrow{ }^{\prime}$ ' $a$ 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+
    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=
    by (cases x, auto simp: zero-arctic-delta-def)
show }x*(y+z)=x*y+x*
    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=
    by (cases x, simp-all add: one-arctic-delta-def)
show 0 + x = x
    by (simp add: zero-arctic-delta-def)
show 0 * x = 0
    by (simp add: zero-arctic-delta-def)
show (0 :: 'a arctic-delta) }=
    by (simp add: zero-arctic-delta-def one-arctic-delta-def)
    show }x+0=x\mathrm{ by (cases }x\mathrm{ , auto simp: zero-arctic-delta-def)
    show }x\geq
    by (cases x, auto)
show (1 :: 'a arctic-delta) \geq0
    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 \geqx unfolding max-def
    by (cases x, (cases y, auto)+)
assume ge: }x\geq
from ge show }x+z\geqy+
    by (cases x, cases y, cases z, auto, cases y, cases z, auto, cases z, auto)
    from ge show }x*z\geqy*
    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 zx \geqmax 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\geqy\mathrm{ and }y\geq
    thus }x\geq
    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
    thus }x*y\geqx*
    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 $x \geq y \Longrightarrow 0 \geq z \Longrightarrow y * z \geq x * z$
by (cases $z$, cases $x$, auto simp: zero-arctic-delta-def)
qed
end

$$
\mathrm{x}>\mathrm{d} \mathrm{y} \text { is interpreted as } \mathrm{y}=-\inf \text { or }(\mathrm{x}, \mathrm{y}!=-\inf \text { and } \mathrm{x}>\mathrm{d} \mathrm{y})
$$

fun gt-arctic-delta $::$ ' $a$ :: floor-ceiling $\Rightarrow$ ' $a$ arctic-delta $\Rightarrow$ 'a arctic-delta $\Rightarrow$ bool where gt-arctic-delta $\delta$ - MinInfty-delta $=$ True
| gt-arctic-delta $\delta$ MinInfty-delta (Num-arc-delta -) = False
| gt-arctic-delta $\delta$ (Num-arc-delta $x)($ Num-arc-delta $y)=$ delta-gt $\delta x y$
fun get-arctic-delta-num $::$ 'a arctic-delta $\Rightarrow{ }^{\prime} a$
where get-arctic-delta-num (Num-arc-delta $n$ ) $=n$
fun pos-arctic-delta :: ('a :: floor-ceiling) arctic-delta $\Rightarrow$ bool
where pos-arctic-delta MinInfty-delta $=$ False
| pos-arctic-delta (Num-arc-delta $n)=(0 \leq n)$
lemma arctic-delta-interpretation: assumes dpos: $\delta>0$ shows $S N$-both-mono-ordered-semiring-1
1 (gt-arctic-delta $\delta$ ) pos-arctic-delta
proof -
from delta-interpretation [OF dpos] interpret $S N$-strict-mono-ordered-semiring-1
$\delta$ delta-gt $\delta$ delta-mono by simp
show ?thesis
proof
fix $x$ y $z::$ 'a arctic-delta
assume $x \geq y$ and gt-arctic-delta $\delta y z$
thus gt-arctic-delta $\delta x z$
by (cases $z, \operatorname{simp}$, cases $y, \operatorname{simp}$, cases $x, \operatorname{simp}, \operatorname{simp}$ add: compat)
next
fix $x$ y $z$ :: 'a arctic-delta
assume gt-arctic-delta $\delta x y$ and $y \geq z$
thus gt-arctic-delta $\delta x z$
by (cases $z$, simp, cases $y$, simp, cases $x, \operatorname{simp}, \operatorname{simp}$ add: compat2)
next
fix $x y$ :: 'a arctic-delta
assume gt-arctic-delta $\delta x y$
thus $x \geq y$
by (cases $x$, insert dpos, (cases $y$, auto simp: delta-gt-def)+)
next
fix $x y z u$
assume gt-arctic-delta $\delta x y$ and gt-arctic-delta $\delta z u$
thus gt-arctic-delta $\delta(x+z)(y+u)$
by (cases $y$, cases $u$, simp, cases $z$, simp, cases $x$, simp, simp add: delta-gt-def,
cases $z$, cases $x$, simp, cases $u$, simp, simp, cases $x$, simp, cases $z$, simp, cases $u$, simp add: delta-gt-def, simp add: delta-gt-def)

```
next
    fix x y z
    assume gt-arctic-delta \delta x y
    thus gt-arctic-delta }\delta(x*z)(y*z
        by (cases y, simp, cases z, simp, cases x, simp, simp add: plus-gt-left-mono)
    next
    fix }
    assume gt-arctic-delta \delta 0 x
    thus }x=
        by (cases x, auto simp: zero-arctic-delta-def)
next
    fix }
    show pos-arctic-delta 1 unfolding one-arctic-delta-def by simp
    show gt-arctic-delta \delta x 0 unfolding zero-arctic-delta-def by simp
    show (1 :: 'a arctic-delta) \geq 0 unfolding zero-arctic-delta-def by simp
    show }x\geq0\mathrm{ unfolding zero-arctic-delta-def by simp
    show \neg pos-arctic-delta 0 unfolding zero-arctic-delta-def by simp
next
    fix x y :: 'a arctic-delta
    assume pos-arctic-delta x
    thus pos-arctic-delta (x+y) by (cases x, simp, cases y, auto)
next
    fix x y :: 'a arctic-delta
    assume pos-arctic-delta x and pos-arctic-delta y
    thus pos-arctic-delta (x*y) by (cases x, simp, cases y, auto)
next
    show SN {(x,y). pos-arctic-delta y ^ gt-arctic-delta \delta x y} (is SN ?rel)
    proof - {
        fix }
        assume \exists f.f 0=x^(\foralli.(fi,f(Suc i)) \in?rel)
        from this obtain f}\mathrm{ where f0=x and seq: }\foralli.(fi,f(Suc i))\in? rel by
auto
        from seq have steps: }\forall\mathrm{ i. gt-arctic-delta }\delta(fi)(f(Suc i))\wedge pos-arctic-delta
(f (Suc i)) by auto
        let ?g=\lambda i.get-arctic-delta-num (fi)
        have }\foralli.?g(Suc i)\geq0^\mathrm{ delta-gt }\delta(?g i)(?g (Suc i)
        proof
            fix }
            from steps have i: gt-arctic-delta \delta (f i) (f (Suc i)) ^ pos-arctic-delta (f
(Suc i)) by auto
    from i obtain n where fi:fi=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\leqm by simp
            from ifi fsi have delta-gt \delta n m}\mathrm{ by auto
            with fi fsi gz
            show ?g (Suc i)\geq0^ delta-gt \delta(?g i)(?g (Suc i)) by auto
            qed
```

from this obtain $g$ where $\forall$ i. $g($ Suc $i) \geq 0 \wedge$ delta-gt $\delta(g i)(g($ Suc $i))$ by auto
hence $\exists f . f 0=g 0 \wedge(\forall i .(f i, f($ Suc $i)) \in\{(x, y) . y \geq 0 \wedge$ delta-gt $\delta x$ $y\})$ by auto
with $S N$ have False unfolding $S N$-defs by auto
\}
thus ?thesis unfolding $S N$-defs by auto
qed
next
fix $c d::$ 'a arctic-delta
assume pos-arctic-delta d
then obtain $n$ where $d: d=N u m$-arc-delta $n$ and $n: 0 \leq n$
by (cases d, auto)
show $\exists e . e \geq 0 \wedge$ pos-arctic-delta $e \wedge \neg c \geq d * e$
proof (cases c)
case MinInfty-delta
show ?thesis
by (rule exI[of - Num-arc-delta 0], unfold d MinInfty-delta zero-arctic-delta-def, simp)
next
case (Num-arc-delta m)
show ?thesis
by (rule exI[of - Num-arc-delta (abs $m+1$ )], insert $n$, unfold d Num-arc-delta zero-arctic-delta-def, simp)
qed
next
fix $x y z$
assume gt: gt-arctic-delta $\delta y z$
\{
fix $x y z$
assume $g t$ : delta-gt $\delta y z$
have delta-gt $\delta(x+y)(x+z)$
using plus-gt-left-mono[OF gt] by (auto simp: field-simps)
\}
with gt show gt-arctic-delta $\delta(x * y)(x * z)$
by (cases $x$, simp, cases $z$, simp, cases $y$, simp-all)
qed
qed
fun weak-gt-arctic-delta :: ('a :: floor-ceiling) arctic-delta $\Rightarrow{ }^{\prime}$ 'a arctic-delta $\Rightarrow$ bool where weak-gt-arctic-delta - MinInfty-delta $=$ True
| weak-gt-arctic-delta MinInfty-delta (Num-arc-delta -) = False
| weak-gt-arctic-delta (Num-arc-delta $x)($ Num-arc-delta $y)=(x>y)$
interpretation weak-arctic-delta-SN: weak-SN-both-mono-ordered-semiring-1 weak-gt-arctic-delta 1 pos-arctic-delta

## proof

fix $x y s$
assume orient: $\forall x y .(x, y) \in$ set $x y s \longrightarrow$ weak-gt-arctic-delta $x y$
obtain xysp where xysp: xysp = map ( $\lambda$ (ax, ay). (case ax of Num-arc-delta $x$ $\Rightarrow x$, case ay of Num-arc-delta $y \Rightarrow y)$ ) (filter $(\lambda$ (ax,ay). ax $\neq$ MinInfty-delta $\wedge$ $a y \neq$ MinInfty-delta) xys)
(is - = map ?f -)
by auto
have $\forall x y .(x, y) \in$ set $x y s p \longrightarrow x>y$
proof (intro allI impI)
fix $x y$
assume $(x, y) \in$ set $x y s p$
with xysp obtain $a x a y$ where $(a x, a y) \in$ set $x y s$ and $a x \neq$ MinInfty-delta and $a y \neq$ MinInfty-delta and $(x, y)=? f(a x, a y)$ by auto
hence (Num-arc-delta $x$, Num-arc-delta $y$ ) $\in$ set xys by (cases ax, simp, cases ay, auto)
with orient show $x>y$ by force
qed
with delta-minimal-delta[of xysp] obtain $\delta$ where dpos: $\delta>0$ and orient2: $\Lambda$ $x y .(x, y) \in$ set $x y s p \Longrightarrow$ delta-gt $\delta x y$ by auto
have orient: $\forall x y .(x, y) \in$ set xys $\longrightarrow$ gt-arctic-delta $\delta x y$
proof(intro allI impI)
fix $a x$ ay
assume axay: $(a x, a y) \in$ set xys
with orient have orient: weak-gt-arctic-delta ax ay by auto
show gt-arctic-delta $\delta$ ax ay
proof (cases ay, simp)
case (Num-arc-delta $y$ ) note $a y=$ this
show ?thesis
proof (cases ax)
case MinInfty-delta
with ay orient show ?thesis by auto
next
case (Num-arc-delta $x$ ) note $a x=$ this
from ax ay axay have $(x, y) \in$ set xysp unfolding xysp by force
from ax ay orient2 [OF this] show ?thesis by simp
qed
qed
qed
show $\exists g t$. SN-both-mono-ordered-semiring-1 1 gt pos-arctic-delta $\wedge(\forall x y .(x, y)$ $\in$ set $x y s \longrightarrow g t x y)$
by (intro exI conjI, rule arctic-delta-interpretation[OF dpos], rule orient)
qed
end

## References

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


[^0]:    ${ }^{1}$ http://cl-informatik.uibk.ac.at/software/ceta

