

Termination Restricted to Right-Forward Closures

René Thiemann¹, Dieter Hofbauer², Ulysse Le Huitouze³, and
Johannes Waldmann⁴

¹University of Innsbruck, Austria

²ASW Saarland, Germany

³University of Rennes, France

⁴HTWK Leipzig, Germany

July 17, 2026

Abstract

In this AFP entry we formalize Dershowitz’ theorem that termination of a term rewrite system (TRS) starting from arbitrary terms is equivalent to termination starting from terms in the right-forward closures of right-hand sides, provided that the TRS is right-linear or orthogonal [1]. Our proof deviates from the original one in that no reorderings of steps in infinite derivations are required, making it more precise in its argumentation. We also integrate a later result that one can weaken orthogonality to locally confluent overlay TRSs [2].

In order to arrive at these statements, we had to formalize two further results: Following Gramlich, we prove that for locally confluent overlay TRSs the notions of termination and innermost termination coincide [3]; and we verify that narrowing with a right-linear TRS preserves linearity of terms.

For further details, we refer to our FSCD 2026 paper [4].

Contents

1	Rewriting with the Innermost Strategy	2
2	Gramlich’s Criterion to Prove Termination by Innermost Termination	7
3	Linearity Preservation by Unification	14
4	Narrowing and Linearity Preservation	25
4.1	Preparations for Narrowing	25
4.2	Narrowing Relations	28

1 Rewriting with the Innermost Strategy

```

theory Innermost-Rewriting
  imports First-Order-Rewriting.Trs
begin

inductive-set inn-rstep :: ('f, 'v) trs  $\Rightarrow$  ('f, 'v) term rel for R
  where inn-rstep[intro]:  $(l, r) \in R \Longrightarrow \text{set } (\text{args } (l \cdot \sigma)) \subseteq \text{NF-trs } R \Longrightarrow (C\langle l \cdot \sigma \rangle, C\langle r \cdot \sigma \rangle) \in \text{inn-rstep } R$ 

lemma inn-rstep-rstep:  $\text{inn-rstep } R \subseteq \text{rstep } R$ 
  by (standard, auto dest: inn-rstep.cases)

lemma inn-rstep-ctxt: assumes  $(s, t) \in \text{inn-rstep } R$ 
  shows  $(C\langle s \rangle, C\langle t \rangle) \in \text{inn-rstep } R$ 
  using assms
proof (induct)
  case *:  $(\text{inn-rstep } l \ r \ \sigma \ D)$ 
  from inn-rstep[OF *, of  $C \circ_c D$ ]
  show ?case by auto
qed

lemma ctxt-closed-inn-rstep[simp, intro]:  $\text{ctxt.closed } (\text{inn-rstep } R)$ 
  using inn-rstep-ctxt[of - - R] by fast

inductive-set inn-nrrstep :: ('f, 'v) trs  $\Rightarrow$  ('f, 'v) term rel for R
  where inn-nrrstep[intro]:  $(l, r) \in R \Longrightarrow \text{set } (\text{args } (l \cdot \sigma)) \subseteq \text{NF-trs } R \Longrightarrow C \neq \text{Hole} \Longrightarrow (C\langle l \cdot \sigma \rangle, C\langle r \cdot \sigma \rangle) \in \text{inn-nrrstep } R$ 

lemma inn-nrrstep-nrrstep:  $\text{inn-nrrstep } R \subseteq \text{nrrstep } R$ 
  by (standard, elim inn-nrrstep.cases, intro nrrstepI, auto)

lemma inn-nrrstep-inn-rstep:  $\text{inn-nrrstep } R \subseteq \text{inn-rstep } R$ 
  by (standard, elim inn-nrrstep.cases, blast)

lemma inn-nrrstep-ctxt: assumes  $(s, t) \in \text{inn-nrrstep } R$ 
  shows  $(C\langle s \rangle, C\langle t \rangle) \in \text{inn-nrrstep } R$ 
  using assms
proof (induct)
  case *:  $(\text{inn-nrrstep } l \ r \ \sigma \ D)$ 
  from inn-nrrstep[OF  $*(1-2)$ , of  $C \circ_c D$ ]  $*(3)$ 
  show ?case by (cases C, auto)
qed

lemma inn-rstep-ctxt-inn-nrrstep: assumes  $(s, t) \in \text{inn-rstep } R$ 

```

```

and  $C \neq \text{Hole}$ 
shows  $(C\langle s \rangle, C\langle t \rangle) \in \text{inn-nrrstep } R$ 
using assms
proof (induct)
  case *: (inn-rstep  $l\ r\ \sigma\ D$ )
  from inn-nrrstep[OF  $*(1-2)$ , of  $C \circ_c D$ ]  $*(3)$ 
  show ?case by (cases  $C$ , auto)
qed

lemma NF-inn-rstep-rstep:  $\text{NF } (\text{inn-rstep } R) = \text{NF } (\text{rstep } R)$ 
proof
  show  $\text{NF-trs } R \subseteq \text{NF } (\text{inn-rstep } R)$  using inn-rstep-rstep[of  $R$ ] by auto
  show  $\text{NF } (\text{inn-rstep } R) \subseteq \text{NF-trs } R$ 
  proof
    fix  $t$ 
    assume  $t: t \in \text{NF } (\text{inn-rstep } R)$ 
    obtain  $n$  where  $\text{size } t = n$  by auto
    with  $t$  show  $t \in \text{NF-trs } R$ 
    proof (induct  $n$  arbitrary:  $t$  rule: less-induct)
      case (less  $n\ t$ )
      show ?case
      proof (rule ccontr)
        assume  $\neg ?thesis$ 
        then obtain  $s$  where  $(t,s) \in \text{rstep } R$  by auto
        then obtain  $C\ l\ r\ \sigma$  where  $(l,r) \in R$  and  $t: t = C \langle l \cdot \sigma \rangle$  and  $s = C \langle$ 
 $r \cdot \sigma \rangle$  by auto
        from inn-rstep[OF this(1), of  $\sigma\ C$ , folded this(2,3)] less(2)
        obtain  $u$  where  $u: u \in \text{set } (\text{args } (l \cdot \sigma))$  and  $\text{uNF}: u \notin \text{NF-trs } R$  by auto
        from supt-imp-args[of  $l \cdot \sigma$ ]  $u$ 
        have  $l \cdot \sigma \triangleright u$  by fast
        with  $t$  have  $tu: t \triangleright u$ 
        using subterm.order.strict-trans2 by auto
        with less(3) have  $\text{size } u < n$  using supt-size by auto
        from less(1)[OF this - refl] uNF have  $\text{uNF}: u \notin \text{NF } (\text{inn-rstep } R)$  by auto
        from  $tu$  obtain  $C$  where  $t = C \langle u \rangle$  by blast
        from less(2)[unfolded this] uNF show False
        by (meson NF-iff-no-step inn-rstep-ctxt)
      qed
    qed
  qed
qed

```

```

lemma NF-inn-nrrstep-nrrstep:  $\text{NF } (\text{inn-nrrstep } R) = \text{NF } (\text{nrrstep } R)$ 
proof
  show  $\text{NF } (\text{nrrstep } R) \subseteq \text{NF } (\text{inn-nrrstep } R)$  using inn-nrrstep-nrrstep[of  $R$ ] by auto
  show  $\text{NF } (\text{inn-nrrstep } R) \subseteq \text{NF } (\text{nrrstep } R)$ 
  proof
    fix  $s$ 

```

```

assume  $NF: s \in NF$  (inn-nrrstep  $R$ )
show  $s \in NF$  (nrrstep  $R$ )
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain  $t$  where  $(s,t) \in nrrstep\ R$  by auto
  then obtain  $C\ l\ r\ \sigma$  where  $s = C\langle l \cdot \sigma \rangle$  and  $t = C\langle r \cdot \sigma \rangle$  and  $(l,r) \in$ 
 $R$  and  $C: C \neq Hole$ 
    by (meson nrrstepE)
    hence  $l \cdot \sigma \notin NF$  (rstep  $R$ ) by auto
    from this[folded NF-inn-rstep-rstep]
    have  $l \cdot \sigma \notin NF$  (inn-rstep  $R$ ) by auto
    then obtain  $t$  where  $(l \cdot \sigma, t) \in inn-rstep\ R$  by auto
    from inn-rstep-ctxt-inn-nrrstep[OF this C]  $s$ 
    have  $s \notin NF$  (inn-nrrstep  $R$ ) by auto
    with  $NF$  show False by auto
  qed
qed
qed

```

```

inductive-set inn-rrstep ::  $(f, 'v)\ trs \Rightarrow (f, 'v)\ term\ rel$  for  $R$ 
  where inn-rrstep[intro]:  $(l,r) \in R \Longrightarrow set\ (args\ (l \cdot \sigma)) \subseteq NF-trs\ R \Longrightarrow (l \cdot \sigma,$ 
 $r \cdot \sigma) \in inn-rrstep\ R$ 

```

```

lemma inn-rrstep-rrstep:  $inn-rrstep\ R \subseteq rstep\ R$ 
  by (standard, elim inn-rrstep.cases, intro rstepI, auto)

```

```

lemma inn-rrstep-inn-rstep:  $inn-rrstep\ R \subseteq inn-rstep\ R$ 
  by (standard, elim inn-rrstep.cases)
  (metis ctxt.cop-nil inn-rstep.intros)

```

```

lemma inn-rstep-iff-inn-rrstep-or-inn-nrrstep:  $inn-rstep\ R = (inn-rrstep\ R \cup inn-nrrstep\ R)$ 

```

```

proof
  show  $inn-rstep\ R \subseteq inn-rrstep\ R \cup inn-nrrstep\ R$ 
  proof (rule subrelI)
    fix  $s\ t$  assume  $(s,t) \in inn-rstep\ R$ 
    from inn-rstep.cases[OF this] obtain  $l\ r\ C\ \sigma$  where  $step: (l,r) \in R$   $set\ (args$ 
 $(l \cdot \sigma)) \subseteq NF-trs\ R$ 
     $s = C\langle l \cdot \sigma \rangle\ t = C\langle r \cdot \sigma \rangle$  by metis
    from inn-rrstep[OF step(1-2)] inn-nrrstep[OF step(1-2), of C] step(3-4)
    show  $(s,t) \in inn-rrstep\ R \cup inn-nrrstep\ R$  by (cases C, auto)
  qed
next
  show  $inn-rrstep\ R \cup inn-nrrstep\ R \subseteq inn-rstep\ R$ 
  using inn-rrstep-inn-rstep inn-nrrstep-inn-rstep by auto
qed

```

```

lemma inn-rstep-cases[consumes 1, case-names root nonroot]:

```

$\llbracket (s,t) \in \text{inn-rstep } R; (s,t) \in \text{inn-rrstep } R \implies P; (s,t) \in \text{inn-nrrstep } R \implies P \rrbracket$
 $\implies P$

by (auto simp: inn-rstep-iff-inn-rrstep-or-inn-nrrstep)

lemma *inn-nrrstep-args*:

assumes $(s, t) \in \text{inn-nrrstep } R$

shows $\exists f \ ss \ ts. s = \text{Fun } f \ ss \wedge t = \text{Fun } f \ ts \wedge \text{length } ss = \text{length } ts$

$\wedge (\exists j < \text{length } ss. (ss!j, ts!j) \in \text{inn-rstep } R \wedge (\forall i < \text{length } ss. i \neq j \longrightarrow ss!i = ts!i))$

using *assms*

proof (*cases, goal-cases*)

case *: ($1 \ l \ r \ \sigma \ C$)

from $*(5)$ **obtain** $f \ \text{bef } D \ \text{aft}$ **where** $C: C = \text{More } f \ \text{bef } D \ \text{aft}$ **by** (*cases C, auto*)

define ss **where** $ss = \text{bef } @ \ D \langle l \cdot \sigma \rangle \# \ \text{aft}$

define ts **where** $ts = \text{bef } @ \ D \langle r \cdot \sigma \rangle \# \ \text{aft}$

define j **where** $j = \text{length } \text{bef}$

show *?thesis*

proof (*intro exI conjI*)

show $s = \text{Fun } f \ ss$ **unfolding** $* \ ss\text{-def } C$ **by** *simp*

show $t = \text{Fun } f \ ts$ **unfolding** $* \ ts\text{-def } C$ **by** *simp*

show $\text{length } ss = \text{length } ts$ **unfolding** $ss\text{-def } ts\text{-def}$ **by** *simp*

show $j < \text{length } ss$ **unfolding** $j\text{-def } ss\text{-def}$ **by** *simp*

show $(ss ! j, ts ! j) \in \text{inn-rstep } R$

unfolding $ss\text{-def } ts\text{-def } j\text{-def}$ **using** $\text{inn-rstep}[OF \ *(3,4), \text{ of } D]$ **by** *auto*

show $\forall i < \text{length } ss. i \neq j \longrightarrow ss ! i = ts ! i$

unfolding $ss\text{-def } ts\text{-def } j\text{-def}$ **by** (*auto simp: nth-append*)

qed

qed

lemma *Tinf-imp-SN-inn-nrrstep*: **assumes** $t \in \text{Tinf } (\text{inn-rstep } R)$

shows $\text{SN-on } (\text{inn-nrrstep } R) \ \{t\}$

proof

fix g

assume $g \ 0 \in \{t\}$ **and** $\forall i. (g \ i, g \ (\text{Suc } i)) \in \text{inn-nrrstep } R$

hence $\text{steps: } \bigwedge i. (g \ i, g \ (\text{Suc } i)) \in \text{inn-nrrstep } R$

and $t: t = g \ 0$ **by** *auto*

from $\text{steps}[of \ 0]$ **obtain** $f \ ts$ **where** $g \ 0: g \ 0 = \text{Fun } f \ ts$

using $\text{inn-nrrstep-nrrstep}[of \ R] \ \text{nrrstep-imp-Fun}$ **by** *blast*

define n **where** $n = \text{length } ts$

have $\exists ts. g \ i = \text{Fun } f \ ts \wedge \text{length } ts = n$ **for** i

proof (*induct i*)

case 0

show *?case* **unfolding** $g \ 0 \ n\text{-def}$ **by** *auto*

next

case $(\text{Suc } i)$

then obtain ts **where** $g \ i = \text{Fun } f \ ts$ **and** $\text{length } ts = n$ **by** *auto*

with $\text{steps}[of \ i] \ \text{inn-nrrstep-nrrstep}[of \ R]$ **show** *?case* **using** *nrrstep-args* **by**

fastforce
qed
hence $\forall i. \exists ts. g\ i = \text{Fun } f\ ts \wedge \text{length } ts = n$ **by** *auto*
from *choice[OF this]* **obtain** ts **where** $g\ i = \text{Fun } f\ (ts\ i)$ **and** $len: \text{length } (ts\ i) = n$ **for** i **by** *blast*
have $\exists j < n. (ts\ i\ !\ j, ts\ (Suc\ i)\ !\ j) \in \text{inn-rstep } R \wedge (\forall k < n. k \neq j \longrightarrow ts\ (Suc\ i)\ !\ k = ts\ i\ !\ k)$ **for** i
using *steps[of i, unfolded g]* **using** *len[of i] len[of Suc i]*
using *inn-nrrstep-args* **by** *force*
hence $\forall i. \exists j < n. (ts\ i\ !\ j, ts\ (Suc\ i)\ !\ j) \in \text{inn-rstep } R \wedge (\forall k < n. k \neq j \longrightarrow ts\ (Suc\ i)\ !\ k = ts\ i\ !\ k)$
by *blast*
from *choice[OF this]* **obtain** j
where $j\ i < n \wedge (ts\ i\ !\ j\ i, ts\ (Suc\ i)\ !\ j\ i) \in \text{inn-rstep } R$
 $k < n \implies k \neq j\ i \implies ts\ (Suc\ i)\ !\ k = ts\ i\ !\ k$ **for** $i\ k$
by *blast*
hence $\text{range } j \subseteq \{..<n\}$ **by** *auto*
hence *finite (range j)*
using *finite-subset* **by** *blast*
from *pigeonhole-infinite[OF - this]* **obtain** J
where *inf: infinite {i. j i = J}* **by** *auto*
define I **where** $I = \{i. j\ i = J\}$
obtain i **where** $i \in I$ **unfolding** *I-def* **using** *inf* **by** *force*
from *this[unfolded I-def]* j **have** $J: J < n$ **by** *auto*
define h **where** $h\ i = ts\ i\ !\ J$ **for** i
{
 fix i
 assume $i \in I$
 from *this[unfolded I-def]* J **have** $j\ i = J$ **by** *auto*
 from *j(2)[of i, unfolded this, folded h-def]*
 have $(h\ i, h\ (Suc\ i)) \in \text{inn-rstep } R$.
} **note** $\text{inI} = \text{this}$
{
 fix i
 assume $i \notin I$
 hence $J \neq j\ i$ **unfolding** *I-def* **by** *auto*
 from *j(3)[OF J this]*
 have $(h\ i, h\ (Suc\ i)) \in \{\}^{\hat{=}}$ **by** (*auto simp: h-def*)
} **note** $\text{ninI} = \text{this}$
have *chain: chain (inn-rstep R $\cup$ $\{\}^{\hat{=}}$) h* **using** inI ninI **by** *auto*
have *inf: INFM j. (h j, h (Suc j)) $\in$ inn-rstep R* **using** *inf[folded I-def] inI*
by (*metis (no-types, lifting) INFM-iff-infinite INFM-mono I-def mem-Collect-eq*)
from *chain inf*
have $\neg \text{SN-rel-on-alt } (\text{inn-rstep } R) (\{\}^{\hat{=}}) \{h\ 0\}$
unfolding *SN-rel-on-alt-def* **by** *blast*
hence $\text{nSN}: \neg \text{SN-on } (\text{inn-rstep } R) \{h\ 0\}$
unfolding *SN-rel-on-conv[symmetric]*
unfolding *SN-rel-on-Id* **by** *auto*
have $t \triangleright h\ 0$ **unfolding** $t\ g\ h\text{-def}$ **using** $J[\text{folded } len[\text{of } 0]]$ **by** *simp*

```

with assms[unfolded Tinf-def] have SN-on (inn-rstep R) {h 0} by auto
with nSN show False ..
qed

lemma Tinf-inn-rstep-imp-first-root-step: assumes s ∈ Tinf (inn-rstep R)
shows ∃ t u. (s,t) ∈ (inn-nrrstep R)* ∧ (t,u) ∈ inn-rrstep R ∧ ¬ SN-on
(inn-rstep R) {u}
proof –
from assms have ¬ SN-on (inn-rstep R) {s} unfolding Tinf-def by auto
then obtain f where f0: f 0 = s and steps: ∧ i. (f i, f (Suc i)) ∈ inn-rstep R
by auto
show ?thesis
proof (cases ∃ i. (f i, f (Suc i)) ∈ inn-rrstep R)
case True
define i where i = (LEAST i. (f i, f (Suc i)) ∈ inn-rrstep R)
from LeastI-ex[OF True, folded i-def]
have inn-rrstep: (f i, f (Suc i)) ∈ inn-rrstep R .
have nSN: ¬ SN-on (inn-rstep R) {f j} for j using steps
by (simp add: chain-imp-not-SN-on steps)
have nrr: (f j, f (Suc j)) ∈ inn-nrrstep R if j < i for j
using not-less-Least[OF that[unfolded i-def]] steps[of j]
using inn-rstep-cases by blast
hence steps: (s, f i) ∈ (inn-nrrstep R)* unfolding f0[symmetric]
by (metis rtrancl-fun-conv)
with nSN[of Suc i] inn-rrstep show ?thesis by metis
next
case False
with steps have nrr: ∧ i. (f i, f (Suc i)) ∈ inn-nrrstep R
by (meson inn-rstep-cases)
hence ¬ SN-on (inn-nrrstep R) {s} using f0 by auto
with Tinf-imp-SN-inn-nrrstep[OF assms]
have False by auto
thus ?thesis ..
qed
qed

end

```

2 Gramlich's Criterion to Prove Termination by Innermost Termination

Gramlich showed that for locally confluent overlay TRSs, innermost termination and termination coincide. We formalize this result in this theory.

```

theory Gramlich-Innermost-Switch
imports

```

begin

lemma *WCR-on-rstep-imp-WCR-on-nrrstep*: **assumes** *WCR-on* (*rstep* *R*) {*t*. *SN-on* (*rstep* *R*) {*t*}}

shows *WCR-on* (*nrrstep* *R*) {*t*. *SN-on* (*nrrstep* *R*) {*t*}}

proof

fix *s t u*

assume *SN*: *s* ∈ {*t*. *SN-on* (*nrrstep* *R*) {*t*}}

and *st*: (*s*, *t*) ∈ *nrrstep* *R*

and *su*: (*s*, *u*) ∈ *nrrstep* *R*

from *st*[*unfolded nrrstep-iff-arg-rstep*]

obtain *f ss i t'* **where**

s: *s* = *Fun f ss*

and *i*: *i* < *length ss*

and *t*: *t* = *Fun f* (*ss*[*i* := *t'*])

and *st*: (*ss* ! *i*, *t'*) ∈ *rstep* *R*

by *auto*

from *nrrstep-imp-pos-term*[*OF su*[*unfolded s*]] **obtain** *j u'* **where**

j: *j* < *length ss*

and *u*: *u* = *Fun f* (*ss*[*j* := *u'*])

and *su*: (*ss* ! *j*, *u'*) ∈ *rstep* *R*

by *auto*

show (*t*, *u*) ∈ (*nrrstep* *R*)[↓]

proof (*cases i = j*)

case *False*

define *v* **where** *v*: *v* = *Fun f* (*ss*[*i* := *t'*, *j* := *u'*])

have *tv*: (*t*, *v*) ∈ *nrrstep* *R* **unfolding** *t v* **using** *False i j su*

using *nrrstep-iff-arg-rstep* **by** *force*

have *v*: *v* = *Fun f* (*ss*[*j* := *u'*, *i* := *t'*]) **unfolding** *v* **using** *i j False*

by (*simp add: list-update-swap*)

have *uv*: (*u*, *v*) ∈ *nrrstep* *R* **unfolding** *u v* **using** *False i j st*

using *nrrstep-iff-arg-rstep* **by** *force*

from *tv uv* **show** *?thesis* **by** *blast*

next

case *True*

from *SN*[*unfolded s*] **have** *set ss* ⊆ {*t*. *SN-on* (*rstep* *R*) {*t*}}

by *simp* (*metis SN-nrrstep-imp-args-SN-rstep SN-on-subset-SN-terms term.sel(4)*)

with *i* **have** *ss* ! *i* ∈ {*t*. *SN-on* (*rstep* *R*) {*t*}} **by** (*auto simp: set-conv-nth*)

from *WCR-onD*[*OF assms this st su*[*folded True*]] **obtain** *v'*

where *tv*: (*t'*, *v'*) ∈ (*rstep* *R*)^{*} **and** *uv*: (*u'*, *v'*) ∈ (*rstep* *R*)^{*} **by** *auto*

define *v* **where** *v*: *v* = *Fun f* (*ss*[*i* := *v'*])

have (*t*, *v*) ∈ (*nrrstep* *R*)^{*} **unfolding** *t v* **using** *tv i*

by (*rule arg-rsteps-into-nrrsteps*)

moreover **have** (*u*, *v*) ∈ (*nrrstep* *R*)^{*} **unfolding** *u v* *True*[*symmetric*] **using**

uv i

by (*rule arg-rsteps-into-nrrsteps*)

ultimately **show** *?thesis* **by** *blast*

qed
qed

lemma *SN-innermost-switch-locally-confluent-overlay-local:*

assumes *WCR: WCR-on (rstep R) {t. SN-on (rstep R) {t}}*

and *overlay: $\bigwedge l r. (False, l, r) \notin \text{critical-pairs ren } R R$*

and *wf: wf-trs R*

shows *SN-on (inn-rstep R) = SN-on (rstep R)*

proof (rule ccontr)

assume $\neg ?thesis$

then obtain *T* **where** *SN-on (inn-rstep R) T $\neq$ SN-on (rstep R) T* **by** *blast*

with *SN-on-subset1 [OF - inn-rstep-rstep, of R]*

have *nSN: $\neg (SN-on (rstep R) T)$*

and *SN: SN-on (inn-rstep R) T* **by** *blast+*

then obtain *init* **where** *init $\in T$ and nSN: $\neg SN-on (rstep R) \{init\}$* **by** *fast*

with *SN* **have** *SN: SN-on (inn-rstep R) {init}* **by** *fast*

from *not-SN-imp-subt-Tinf [OF nSN]* **obtain** *s*

where *init-s: init $\supseteq s$ and Tinf: $s \in Tinf (rstep R)$* **by** *auto*

from *ctxt-closed-SN-on-subt [OF ctxt-closed-inn-rstep SN init-s]*

have *SN: SN-on (inn-rstep R) {s}* .

from *Tinf-rstep-imp-first-root-step [OF Tinf]*

obtain *t u* **where** *tu: (s, t) $\in (nrrstep R)^*$*

(t, u) $\in rstep R$

t $\in Tinf (rstep R)$

$\neg SN-on (rstep R) \{u\}$

by *blast*

from *tu(4)* **obtain** *v* **where** *uv: u $\supseteq v$ v $\in Tinf (rstep R)$*

using *not-SN-imp-subt-Tinf* **by** *blast*

let *?P = $\lambda (n :: nat) s t. s \in Tinf (rstep R) \wedge (s, t) \in rstep R \ O \ \{\supseteq\} \wedge t \in Tinf (rstep R)$*

let *?Q = $\lambda (n :: nat) s1 t1 s2 t2. (t1, s2) \in (nrrstep R)^*$*

have *P0: ?P 0 t v* **using** *tu uv* **by** *auto*

have $\exists S T. S \ 0 = t \wedge$

$T \ 0 = v \wedge$

$(\forall n. ?P \ n \ (S \ n) \ (T \ n) \wedge (T \ n, S \ (Suc \ n)) \in (nrrstep R)^*)$

proof (rule dependent-nat-choice2-start[**where** *?Q = ?Q* **and** *?P = ?P, OF P0*])

fix *s t n*

assume *s $\in Tinf (rstep R) \wedge (s, t) \in rstep R \ O \ \{\supseteq\} \wedge t \in Tinf (rstep R)$*

hence *t $\in Tinf (rstep R)$* **by** *auto*

from *Tinf-rstep-imp-first-root-step [OF this]*

obtain *s' u* **where** *s'u: (t, s') $\in (nrrstep R)^*$ (s', u) $\in rstep R$ s' $\in Tinf (rstep R)$*

$\neg SN-on (rstep R) \{u\}$

by *auto*

from *s'u(4)* **obtain** *v* **where** *uv: u $\supseteq v$ v $\in Tinf (rstep R)$*

using *not-SN-imp-subt-Tinf* **by** *blast*

show $\exists s' t'. ?P (Suc \ n) \ s' \ t' \wedge (t, s') \in (nrrstep R)^*$ **using** *s'u uv* **by** *blast*

qed
then obtain $S \ U$ where
 start: $S \ 0 = t \ U \ 0 = v$ **and**
 STinf: $\bigwedge n. S \ n \in Tinf \ (rstep \ R)$ **and**
 steps: $\bigwedge n. (S \ n, U \ n) \in rstep \ R \ O \ \{\supseteq\}$ **and**
 UTinf: $\bigwedge n. U \ n \in Tinf \ (rstep \ R)$ **and**
 nsteps: $\bigwedge n. (U \ n, S \ (Suc \ n)) \in (nrrstep \ R)^*$
 by blast
from steps have $\forall n. \exists u. (S \ n, u) \in rstep \ R \wedge u \supseteq U \ n$ **by blast**
from choice[OF this] obtain T where
 rsteps: $\bigwedge n. (S \ n, T \ n) \in rstep \ R$ **and**
 subt: $\bigwedge n. T \ n \supseteq U \ n$ **by blast**
from rstepE[OF rsteps] have $\forall n. \exists l \ r \ \sigma. (l, r) \in R \wedge S \ n = l \cdot \sigma \wedge T \ n = r$
 $\cdot \sigma$
 bymetis
from choice[OF this] obtain l where $\forall n. \exists r \ \sigma. (l \ n, r) \in R \wedge S \ n = l \ n \cdot \sigma$
 $\wedge T \ n = r \cdot \sigma$
 bymetis
from choice[OF this] obtain r where $\forall n. \exists \sigma. (l \ n, r \ n) \in R \wedge S \ n = l \ n \cdot \sigma$
 $\wedge T \ n = r \ n \cdot \sigma$
 bymetis
from choice[OF this] obtain σ where *rsteps*: $\bigwedge n. (l \ n, r \ n) \in R$
 and *S*: $\bigwedge n. S \ n = l \ n \cdot \sigma \ n$
 and *T*: $\bigwedge n. T \ n = r \ n \cdot \sigma \ n$
 by blast
{
 fix $x \ n$
 assume $x \in vars-term \ (l \ n)$
 hence $l \ n \triangleright Var \ x$ **using** *rsteps*(1)[of n] *wf*
 unfolding wf-trs-def by force
 hence $l \ n \cdot \sigma \ n \triangleright Var \ x \cdot \sigma \ n$ **by blast**
 from this[folded S] STinf[of n]
 have *SN-on* (*rstep* R) $\{\sigma \ n \ x\}$ **by** (*auto simp: Tinf-def*)
} note SN-vars = this
have *vars*: *vars-term* ($r \ n$) $\subseteq$ *vars-term* ($l \ n$) **for** n
 using *wf rsteps*[of n] **unfolding wf-trs-def by auto**
{
 fix n
 have $\forall x \in vars-term \ (r \ n). (Var \ x \cdot \sigma \ n, U \ n) \notin \{\supseteq\}$
 proof
 fix x
 assume $x \in vars-term \ (r \ n)$
 with vars have $x \in vars-term \ (l \ n)$ **by auto**
 from SN-vars[OF this]
 have *SN-on* (*rstep* R) $\{\sigma \ n \ x\}$.
 with UTinf[of n, unfolded Tinf-def]
 show $(Var \ x \cdot \sigma \ n, U \ n) \notin \{\supseteq\}$ **using** *SN-on-imp-SN-on-subt*[of R]
 by auto
 qed

```

from subt-instance-and-not-subst-imp-subt[OF subt[of n, unfolded T] this]
have  $\exists u \sqsubseteq r \ n. \ U \ n = u \cdot \sigma \ n$  .
}
hence  $\forall \ n. \ \exists u \sqsubseteq r \ n. \ U \ n = u \cdot \sigma \ n$  by metis
from choice[OF this] obtain u where
  subt:  $\bigwedge \ n. \ r \ n \sqsupseteq u \ n$  and U:  $\bigwedge \ n. \ U \ n = u \ n \cdot \sigma \ n$ 
by auto
define  $\delta$  where  $\delta \ n \ x = (SOME \ u. (\sigma \ n \ x, u) \in (inn-rstep \ R) \uparrow)$  for  $n \ x$ 
{
  fix  $x \ n$ 
  assume  $x \in vars-term \ (l \ n)$ 
  from SN-vars[OF this]
  have SN-on (rstep R)  $\{\sigma \ n \ x\}$  .
  hence SN-on (inn-rstep R)  $\{\sigma \ n \ x\}$  using inn-rstep-rstep[of R] by fast
  hence  $\exists \ u. (\sigma \ n \ x, u) \in (inn-rstep \ R) \uparrow$  by (rule SN-on-imp-normalizability)
  from someI-ex[OF this, folded  $\delta$ -def]
  have NF:  $(\sigma \ n \ x, \delta \ n \ x) \in (inn-rstep \ R)^!$  .
  hence steps:  $(\sigma \ n \ x, \delta \ n \ x) \in (inn-rstep \ R)^*$ 
    and NF:  $\delta \ n \ x \in NF-trs \ R$  by (auto simp: NF-inn-rstep-rstep)
  hence  $(\sigma \ n \ x, \delta \ n \ x) \in (rstep \ R)^*$  using inn-rstep-rstep[of R]
    using rtrancl-mono by blast
  note NF steps this
} note  $\sigma \delta = this$ 

have fun-l: is-Fun ( $l \ n$ ) for  $n$  using rsteps[of  $n$ ] wf unfolding wf-trs-def by
force
have  $l \sigma \delta: (l \ n \cdot \sigma \ n, l \ n \cdot \delta \ n) \in (nrrstep \ R)^*$  for  $n$ 
by (rule term-subst-rsteps-nrrsteps[OF - fun-l], rule  $\sigma \delta(3)$ , auto)

have args: set ( $args \ (l \ n \cdot \delta \ n)$ )  $\subseteq NF-trs \ R$  for  $n$ 
proof
  fix  $v$ 
  assume  $v \in set \ (args \ (l \ n \cdot \delta \ n))$ 
  hence sub:  $l \ n \cdot \delta \ n \triangleright v$ 
    by (metis fun-l is-Fun-Fun-conv subst-apply-eq-Var supt.arg term.distinct(1)
      term.exhaust-sel)
  {
    fix  $y$ 
    assume  $(v, y) \in rstep \ R$ 
    then obtain  $l' \ r' \ \mu \ C$  where  $v = C \ \langle l' \cdot \mu \rangle$  and  $lr'$ :  $(l', r') \in R$  by auto
    with sub have  $l \ n \cdot \delta \ n \triangleright l' \cdot \mu$ 
      by (metis ctxt-supteq subterm.less-trans suptI)
    hence  $l \ n \cdot \delta \ n \sqsupseteq l' \cdot \mu$  and neg:  $l \ n \cdot \delta \ n \neq l' \cdot \mu$  by auto
    from supteq-subst-cases'[OF this(1)]
    have False
  }
proof
  from  $lr'$  have NF:  $l' \cdot \mu \notin NF-trs \ R$  by auto
  assume  $\exists x. x \in vars-term \ (l \ n) \wedge \delta \ n \ x \sqsupseteq l' \cdot \mu$ 
  with  $\sigma \delta(1)$ [of -  $n$ ] NF show False by fastforce

```

```

next
  assume  $\exists w \sqsubseteq l\ n. \text{is-Fun } w \wedge l' \cdot \mu = w \cdot \delta\ n$ 
  then obtain  $w$  where  $∗: w \sqsubseteq l\ n \text{ is-Fun } w\ l' \cdot \mu = w \cdot \delta\ n$  by auto
  from mgu-vd-complete[OF  $∗(3)$ ][symmetric], of ren]
  obtain  $\mu 1\ \mu 2$  where mgu: mgu- $vd$  ren  $w\ l' = \text{Some } (\mu 1, \mu 2)$  by auto
  from  $∗\ \text{neq}$  have  $w \triangleleft l\ n$  by auto
  then obtain  $C$  where  $l\ n = C \langle w \rangle$  and  $C \neq \text{Hole}$  by auto
  from critical-pairsI[OF rsteps(1) lr' this(1)  $∗(2)$  mgu refl refl refl] this(2)
    overlay show False by auto
qed
}
thus  $v \in \text{NF-trs } R$  by auto
qed
hence lNF:  $l\ n \cdot \delta\ n \in \text{NF } (\text{nrrstep } R)$  for  $n$ 
  by (rule args-NF-rstep-imp-NF-nrrstep)

define  $W$  where  $W\ n = l\ n \cdot \delta\ n$  for  $n$ 
have W0-NF:  $W\ 0 \in \text{NF } (\text{nrrstep } R)$  using lNF[of 0] by (auto simp: W-def)

from WCR-on-rstep-imp-WCR-on-nrrstep[OF WCR]
have WCR: WCR-on  $(\text{nrrstep } R) \{t. \text{SN-on } (\text{nrrstep } R) \{t\}\}$  .

{
  fix  $n$ 
  from inn-rstep[OF rsteps(1), of  $n\ \delta\ n\ \text{Hole}$ , OF args]
  have inn-rstep:  $(l\ n \cdot \delta\ n, r\ n \cdot \delta\ n) \in \text{inn-rstep } R$  by auto

  from vars[of  $n$ ] subt[of  $n$ ]
  have varsu: vars-term  $(u\ n) \subseteq \text{vars-term } (l\ n)$ 
    using supteq-imp-vars-term-subset by blast
  from UTinf[unfolded  $U$ , of  $n$ ] have  $\neg \text{SN-on } (\text{rstep } R) \{u\ n \cdot \sigma\ n\}$ 
    unfolding Tinf-def by auto
  hence fun-u: is-Fun  $(u\ n)$  using
    SN-vars[of -  $n$ ] varsu
    by (cases  $u\ n$ , auto)

  have  $u\sigma\delta$ :  $(u\ n \cdot \sigma\ n, u\ n \cdot \delta\ n) \in (\text{nrrstep } R)^{\widehat{*}}$ 
  proof (rule term-subst-rsteps-nrrsteps[OF - fun-u], rule  $u\sigma\delta(3)$ )
    fix  $x$ 
    show  $x \in \text{vars-term } (u\ n) \implies x \in \text{vars-term } (l\ n)$  using vars[of  $n$ ]
      subt[of  $n$ ] by (meson in-mono supteq-imp-vars-term-subset)
  qed

  from UTinf[of  $n$ , unfolded  $U$ ]
  have SNu: SN-on  $(\text{nrrstep } R) \{u\ n \cdot \sigma\ n\}$  by (rule Tinf-imp-SN-nrrstep)

  with  $u\sigma\delta$  have SN-on  $(\text{nrrstep } R) \{u\ n \cdot \delta\ n\}$  by (rule steps-preserve-SN-on)

```

hence $SN\text{-on } (inn\text{-}nrrstep\ R) \{u\ n \cdot \delta\ n\}$ **using** $inn\text{-}nrrstep\text{-}nrrstep$ **by** *fast*
 then obtain v **where** $iuv: (u\ n \cdot \delta\ n, v) \in (inn\text{-}nrrstep\ R)^\wedge^*$ **and** $vNF: v \in$
 $NF\ (nrrstep\ R)$
 unfolding $NF\text{-}inn\text{-}nrrstep\text{-}nrrstep[symmetric]$ **by** $(meson\ SN\text{-}reaches\text{-}NF)$

from $u\sigma\delta\ iuv\ vNF$ **have** $uv: (u\ n \cdot \sigma\ n, v) \in (nrrstep\ R)^\nabla$
 using $rtrancl\text{-}mono[OF\ inn\text{-}nrrstep\text{-}nrrstep,\ of\ R]$ **by** *force*

from $nsteps[of\ n,\ unfolded\ U\ S]\ l\sigma\delta[of\ Suc\ n]\ lNF[of\ Suc\ n]$
have $ul: (u\ n \cdot \sigma\ n, l\ (Suc\ n) \cdot \delta\ (Suc\ n)) \in (nrrstep\ R)^\nabla$
 by *blast*

from $Newman\text{-}local[OF\ SNu\ WCR]$
have $CR\text{-}on\ (nrrstep\ R) \{u\ n \cdot \sigma\ n\}$.
from $CR\text{-}on\text{-}same\text{-}NF[OF\ this\ -\ ul\ uv]$
have $v: v = l\ (Suc\ n) \cdot \delta\ (Suc\ n)$ **by** *simp*

with iuv
have $(u\ n \cdot \delta\ n, l\ (Suc\ n) \cdot \delta\ (Suc\ n)) \in (inn\text{-}nrrstep\ R)^*$ **by** *auto*
hence $iul: (u\ n \cdot \delta\ n, l\ (Suc\ n) \cdot \delta\ (Suc\ n)) \in (inn\text{-}rstep\ R)^*$
 using $rtrancl\text{-}mono[OF\ inn\text{-}nrrstep\text{-}inn\text{-}rstep[of\ R]]$ **by** *auto*

have $(r\ n \cdot \delta\ n, u\ n \cdot \delta\ n) \in \{\supseteq\}$ **using** $subt[of\ n]$ **by** *auto*

from $inn\text{-}rstep\ this\ iul$
have $(W\ n, W\ (Suc\ n)) \in inn\text{-}rstep\ R\ O\ \{\supseteq\}\ O\ (inn\text{-}rstep\ R)^*$
 unfolding $W\text{-}def$ **by** *blast*

} **note** $inf\text{-}W\text{-}seq = this$

have $(t, W\ 0) \in (nrrstep\ R)^*$ **using** $\langle S\ 0 = t \rangle$ **unfolding** $S\ W\text{-}def$ **using** $l\sigma\delta$
by *auto*

with $tu(1)$ **have** $(s, W\ 0) \in (nrrstep\ R)^*$ **by** *auto*
with $W0\text{-}NF$ **have** $sW0: (s, W\ 0) \in (nrrstep\ R)^\dagger$ **by** *auto*
from $Tinf\text{-}imp\text{-}SN\text{-}nrrstep[OF\ Tinf]$
have $SNs: SN\text{-}on\ (nrrstep\ R) \{s\}$.
hence $SN\text{-}on\ (inn\text{-}nrrstep\ R) \{s\}$ **using** $inn\text{-}nrrstep\text{-}nrrstep$ **by** *blast*
then obtain v **where** $(s, v) \in (inn\text{-}nrrstep\ R)^\dagger$
 by $(metis\ SN\text{-}on\text{-}imp\text{-}normalizability)$
hence $isv: (s, v) \in (inn\text{-}nrrstep\ R)^*$ **and** $v: v \in NF\ (nrrstep\ R)$
 using $NF\text{-}inn\text{-}nrrstep\text{-}nrrstep[of\ R]$ **by** *auto*
from isv **have** $(s, v) \in (nrrstep\ R)^*$ **using** $rtrancl\text{-}mono[OF\ inn\text{-}nrrstep\text{-}nrrstep]$
by *auto*

with v **have** $sv: (s, v) \in (nrrstep\ R)^\dagger$ **by** *auto*
from $CR\text{-}on\text{-}same\text{-}NF[OF\ Newman\text{-}local[OF\ SNs\ WCR] - sv\ sW0]$
have $v = W\ 0$ **by** *auto*
with isv **have** $(s, W\ 0) \in (inn\text{-}nrrstep\ R)^*$ **by** *auto*
hence $(s, W\ 0) \in (inn\text{-}rstep\ R)^*$ **using** $rtrancl\text{-}mono[OF\ inn\text{-}nrrstep\text{-}inn\text{-}rstep]$

```

by auto
with SN
have SN-W0: SN-on (inn-rstep R) {W 0}
  by (metis steps-preserve-SN-on)

from inf-W-seq have  $\neg$  SN-on (inn-rstep R O  $\{\supseteq\}$  O (inn-rstep R)*) {W 0} by
blast

moreover from SN-on-r-imp-SN-on-supt-union-r[OF ctxt-closed-inn-rstep SN-W0]
have SN-on ( $\{\supseteq\} \cup$  inn-rstep R) {W 0} .
from SN-on-transcl[OF this]
have *: SN-on (( $\{\supseteq\} \cup$  inn-rstep R)+) {W 0} .
have id:  $\{\supseteq\} = \{\supseteq\}^=$  by auto
have SN-on (inn-rstep R O  $\{\supseteq\}$  O (inn-rstep R)*) {W 0}
  by (rule SN-on-subset1[OF *], unfold id, regeXP)

ultimately show False ..
qed

lemma SN-innermost-switch-locally-confluent-overlay:
  assumes WCR: WCR-on (rstep R) {t. SN-on (rstep R) {t}}
  and overlay:  $\bigwedge l r. (False, l, r) \notin \text{critical-pairs ren } R R$ 
  and wf: wf-trs R
shows SN (inn-rstep R) = SN (rstep R)
  unfolding SN-innermost-switch-locally-confluent-overlay-local[OF assms] ..

end

```

3 Linearity Preservation by Unification

```

theory Linear-Unification
  imports
    First-Order-Terms.Unification-More
begin

```

A sufficient criterion to ensure that $t \cdot \sigma$ is linear.

```

lemma linear-term-subst: linear-term t
   $\implies (\bigwedge x. x \in \text{vars-term } t \implies \text{linear-term } (\sigma x))$ 
   $\implies (\bigwedge x y. x \in \text{vars-term } t \implies y \in \text{vars-term } t \implies x \neq y \implies \text{vars-term } (\sigma x)$ 
 $\cap \text{vars-term } (\sigma y) = \{\})$ 
   $\implies \text{linear-term } (t \cdot \sigma)$ 
proof (induct t)
  case (Fun f ts)
  show ?case unfolding eval-term.simps linear-term.simps
  proof (intro conjI ballI)
    fix tsig
    assume tsig  $\in$  set (map ( $\lambda s. s \cdot \sigma$ ) ts)

```

```

then obtain ti where ti: ti ∈ set ts and tsig: tsig = ti · σ by auto
show linear-term tsig unfolding tsig
  by (rule Fun(1)[OF ti], insert Fun(2-) ti, auto)
next
from Fun(2)[unfolded linear-term.simps] have part: is-partition (map vars-term
ts) by auto
show is-partition (map vars-term (map (λs. s · σ) ts))
  unfolding map-map o-def linear-term.simps
  unfolding is-partition-alt is-partition-alt-def length-map
proof (intro allI impI, goal-cases)
case (1 i j)
show ?case
proof (rule ccontr)
assume ¬ ?thesis
then obtain y where y ∈ vars-term (ts ! i · σ) y ∈ vars-term (ts ! j · σ)
  using 1 by auto
from this[unfolded vars-term-subst] obtain xi xj where
  xij: xi ∈ vars-term (ts ! i) xj ∈ vars-term (ts ! j) and
  y: y ∈ vars-term (σ xi) y ∈ vars-term (σ xj) by auto
from part[unfolded is-partition-alt is-partition-alt-def length-map, rule-format,
OF 1]
  xij 1 have xi ≠ xj by auto
  from Fun(4)[OF - - this] have vars-term (σ xi) ∩ vars-term (σ xj) = {}
using xij 1
  by force
  with y show False by auto
qed
qed
qed
qed auto

```

Unification of two var disjoint terms where one of them is linear results in a partially linear substitution and linear terms

definition *vars-mset-left* :: $((f, v)term \times (f, v)term) multiset \Rightarrow 'v multiset$ **where**
vars-mset-left m = *sum-mset* (*image-mset* (*vars-term-ms* o *fst*) m)

definition *vars-mset-right* :: $((f, v)term \times (f, v)term) multiset \Rightarrow 'v multiset$ **where**
vars-mset-right m = *sum-mset* (*image-mset* (*vars-term-ms* o *snd*) m)

definition *linear-mset* :: $'a multiset \Rightarrow bool$ **where**
linear-mset m = $(\forall x. count\ m\ x \leq 1)$

lemma *count-sum-mset-image-mset*:
count (*sum-mset* (*image-mset* f m)) x = *sum-mset* (*image-mset* ($\lambda a. count\ (f\ a)$ x) m)
by (*induct* m, *auto*)

```

lemma linear-vars-term-ms: linear-mset (vars-term-ms t) = linear-term t
  unfolding linear-mset-def
proof (induct t)
  case (Fun f ts)
  show ?case
proof
  assume cnt:  $\forall x. \text{count } (\text{vars-term-ms } (\text{Fun } f \text{ ts})) x \leq 1$ 
  show linear-term (Fun f ts) unfolding linear-term.simps
  proof (intro conjI ballI)
    fix t
    assume t:  $t \in \text{set } ts$ 
    {
      fix x
      have count (vars-term-ms t) x  $\leq$  count (vars-term-ms (Fun f ts)) x
        using split-list[OF t] by auto
      hence count (vars-term-ms t) x  $\leq 1$  using cnt[rule-format, of x] by auto
    }
    with Fun[OF t] show linear-term t by simp
  next
  show is-partition (map vars-term ts) unfolding is-partition-def
  proof (clarsimp)
    fix j i
    assume ji:  $j < \text{length } ts \wedge i < j$ 
    show vars-term (ts ! i)  $\cap$  vars-term (ts ! j) = {}
    proof (rule ccontr)
      assume  $\neg ?thesis$ 
      then obtain x where  $x \in \# \text{vars-term-ms } (ts ! i) \wedge x \in \# \text{vars-term-ms } (ts$ 
! j) by auto
      hence count: count (vars-term-ms (ts ! i)) x  $\geq 1 \wedge$  count (vars-term-ms (ts
! j)) x  $\geq 1$ 
      by auto
      from ji(1) obtain aft bef where  $ts = bef @ ts ! j \# aft$  and bef: bef
= take j ts
      using id-take-nth-drop by blast
      from ji have i:  $i < \text{length } bef$  unfolding bef by auto
      from i obtain b m where  $bef = b @ bef ! i \# m$ 
      using id-take-nth-drop by blast
      also have bef ! i = ts ! i unfolding bef using ji by auto
      finally have  $ts = b @ ts ! i \# m @ ts ! j \# aft$  using ts by auto
      from arg-cong[OF this, of mset]
      have count (vars-term-ms (Fun f ts)) x  $\geq$  count (vars-term-ms (ts ! i)) x
+ count (vars-term-ms (ts ! j)) x
      by simp
      with count have count (vars-term-ms (Fun f ts)) x  $\geq 2$  by linarith
      with cnt[rule-format, of x] show False by auto
    qed
  qed
  qed
next

```

```

assume lin: linear-term (Fun f ts)
hence part: is-partition (map vars-term ts) by auto
show  $\forall x. \text{count } (\text{vars-term-ms } (\text{Fun } f \text{ ts})) \ x \leq 1$ 
proof
  fix x
  {
    fix t
    assume  $t \in \text{set } ts$ 
    from Fun[OF this] lin this have  $\text{count } (\text{vars-term-ms } t) \ x \leq 1$  by auto
  } note IH = this
show  $\text{count } (\text{vars-term-ms } (\text{Fun } f \text{ ts})) \ x \leq 1$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  hence count:  $\text{count } (\text{vars-term-ms } (\text{Fun } f \text{ ts})) \ x \geq 2$  by auto
  hence  $x \in \# \text{vars-term-ms } (\text{Fun } f \text{ ts})$ 
    by (metis Suc-1 Suc-le-eq count-greater-eq-one-iff less-imp-le-nat)
  then obtain t where  $t \in \text{set } ts$  and  $x \in \text{vars-term } t$  by auto
  hence  $\text{count } (\text{vars-term-ms } t) \ x \geq 1$  by simp
  with IH[OF t] have c1:  $\text{count } (\text{vars-term-ms } t) \ x = 1$  by linarith
  from t obtain i where  $i < \text{length } ts$  and  $t = ts ! i$ 
    by (auto simp: set-conv-nth)
  from t x i
  have  $j < \text{length } ts \implies j \neq i \implies x \notin \text{vars-term } (ts ! j)$  for j
  using part[unfolded is-partition-alt is-partition-alt-def, rule-format, unfolded
length-map, of i j]
    by auto
  define A where  $A = \{0..<\text{length } ts\} - \{i\}$ 
  have  $ts = \text{map } (\lambda i. ts ! i) [0..<\text{length } ts]$ 
    by (intro nth-equalityI, auto)
  from arg-cong[OF this, of mset]
  have  $\text{mset } ts = \text{image-mset } (!! ts) (\text{mset-set } \{0..<\text{length } ts\})$  by auto
  also have ... =
     $\text{add-mset } (ts ! i) (\text{image-mset } (!! ts) (\text{mset-set } A))$ 
    using i by (simp add: mset-set.remove A-def)
  finally have id:  $\text{mset } ts = \text{add-mset } (ts ! i) (\text{image-mset } (!! ts) (\text{mset-set } A))$  .
  from count
  have  $2 \leq \text{count } (\sum \# (\text{image-mset } \text{vars-term-ms } (\text{mset } ts))) \ x$  by simp
  also have ... =  $1 + \text{count } (\sum \# (\text{image-mset } \text{vars-term-ms } (\text{image-mset } (!! ts) (\text{mset-set } A)))) \ x$ 
    unfolding id by (simp add: c1[unfolded t])
  also have  $\text{count } (\sum \# (\text{image-mset } \text{vars-term-ms } (\text{image-mset } (!! ts) (\text{mset-set } A)))) \ x$ 
    =  $(\sum a \in \# \text{image-mset } (!! ts) (\text{mset-set } A). \text{count } (\text{vars-term-ms } a) \ x)$ 
    unfolding count-sum-mset-image-mset ..
  also have ... =  $(\sum a \in \# \text{image-mset } (!! ts) (\text{mset-set } A). 0)$ 
proof (rule arg-cong[of - - sum-mset], rule image-mset-cong)
  fix tj
  show  $tj \in \# \text{image-mset } (!! ts) (\text{mset-set } A) \implies \text{count } (\text{vars-term-ms } tj)$ 

```

$x = 0$

using j unfolding $A\text{-def}$ by force
 qed
 also have $\dots = 0$ by simp
 finally show $False$ by simp
 qed
 qed
 qed
 qed auto

lemma *linear-term-count*: assumes *linear-term* t
 shows $\text{count } (\text{vars-term-ms } t) \ x \leq 1$
 using *assms*[folded *linear-vars-term-ms*, *unfolded linear-mset-def*] ..

lemma *linear-term-Var-subst*: *linear-term* $(t \cdot (\text{Var } o \ r)) \implies \text{linear-term } t$
proof (*induct t*)
 case (*Fun f ts*)
 hence *is-partition* $(\text{map } (\lambda x. \bigcup_{x \in \text{vars-term}} x. \{r \ x\}) \ ts)$
 by (*auto simp: o-def vars-term-subst*)
 also have $\text{map } (\lambda x. \bigcup_{x \in \text{vars-term}} x. \{r \ x\}) \ ts = \text{map } ((\cdot) \ r \ o \ \text{vars-term}) \ ts$ by
auto
 finally have *is-partition* $(\text{map } ((\cdot) \ r \ o \ \text{vars-term}) \ ts)$.
 hence *is-partition* $(\text{map } \text{vars-term } ts)$ unfolding *is-partition-def* *length-map* by
auto
 with *Fun* show ?case by *auto*
 qed *auto*

lemma *vars-mset-right-add*[*simp*]: *vars-mset-right* $(\text{add-mset } p \ E) = \text{vars-term-ms}$
 $(\text{snd } p) + \text{vars-mset-right } E$
 unfolding *vars-mset-right-def* by *auto*

lemma *right-linear-var-disjoint-mgu-mset*: fixes $E :: ((f, 'v)\text{term} \times (f, 'v)\text{term})$
multiset
 and $u :: (f, 'v)\text{term}$
 assumes *set-mset* $(\text{vars-mset-left } E) \cap \text{set-mset } (\text{vars-mset-right } E) = \{\}$
 and *linear-mset* $(\text{vars-mset-right } E)$
 and *is-mgu* σ $(\text{set-mset } E)$
 and *vars-term* $u \cap \text{set-mset } (\text{vars-mset-right } E) = \{\}$
 and *linear-term* u
 shows *linear-term* $(u \cdot \sigma)$
proof –
 define *disjLR* where $\text{disjLR } E = (\text{set-mset } (\text{vars-mset-left } E) \cap \text{set-mset } (\text{vars-mset-right } E) = \{\})$
 for $E :: ((f, 'v)\text{term} \times (f, 'v)\text{term})$ *multiset*
 define *lin* where $\text{lin } E = \text{linear-mset } (\text{vars-mset-right } E)$
 for $E :: ((f, 'v)\text{term} \times (f, 'v)\text{term})$ *multiset*
 define *disjU* where $\text{disjU } u \ E = (\text{vars-term } u \cap \text{set-mset } (\text{vars-mset-right } E) = \{\})$

```

    for  $u :: (f, v)term$  and  $E :: ((f, v)term \times (f, v)term) multiset$ 
    have  $lin0: lin\ E$  unfolding  $lin-def$  using  $assms$  by auto
    have  $disjLR0: disjLR\ E$  unfolding  $disjLR-def$  using  $assms$  by auto
    have  $disjU0: disjU\ u\ E$  unfolding  $disjU-def$  using  $assms$  by auto
    have  $is-mgu0: is-mgu\ \sigma\ (set-mset\ E)$  by fact
    from  $lin0\ disjLR0\ disjU0\ is-mgu0\ \langle linear-term\ u \rangle$  show ?thesis
    proof (induction  $E$  arbitrary:  $u\ \sigma$  rule:  $wf-induct[OF\ wf-unifless]$ )
      case less:  $(1\ E\ u\ \sigma)$ 
      note  $lin = less.prem1$ 
      note  $disjLR = less.prem2$ 
      note  $disjU = less.prem3$ 
      note  $mgu = less.prem4$ 
      note  $linu = less.prem5$ 
      note  $IH = less.IH[rule-format, OF\ UNIF1-unifless]$ 
      show ?case
      proof (cases  $\exists\ s\ t. (s, t) \in \# E \wedge is-Fun\ s \wedge is-Fun\ t$ )
        case True
        then obtain  $s\ t\ F$  where  $E: E = add-mset\ (s, t)\ F$  and  $is-Fun\ s$  and  $is-Fun\ t$ 
        by (metis insert-DiffM)
        then obtain  $f\ g\ ss\ ts$  where  $s: s = Fun\ f\ ss$  and  $t: t = Fun\ g\ ts$  by (cases
s; cases t; auto)
        from  $mgu[unfolded\ E\ s\ t]$  have  $Fun\ f\ ss \cdot \sigma = Fun\ g\ ts \cdot \sigma$ 
        by (auto simp: is-mgu-def)
        hence  $fg: f = g$  and  $len: length\ ss = length\ ts$  by (auto intro: map-eq-imp-length-eq)
        define  $G$  where  $G = F + mset\ (zip\ ss\ ts)$ 
        have  $UNIF1\ Var\ E\ G$  unfolding  $E\ s\ t\ fg\ G-def$  by (rule decomp[OF len])
        note  $IH = IH[OF\ this]$ 
        have  $is-mgu\ \sigma\ (set-mset\ E) = is-mgu\ \sigma\ (set-mset\ G)$  unfolding  $E\ s\ t\ fg\ G-def$ 
        using  $is-mgu-insert-decomp[OF\ len, of\ \sigma\ g\ set-mset\ F]$  by auto
        with  $mgu$  have  $mgu: is-mgu\ \sigma\ (set-mset\ G)$  by auto
        have  $vr: vars-mset-right\ E = vars-mset-right\ G$ 
        unfolding  $vars-mset-right-def\ E\ s\ t\ G-def$ 
        by (simp add: o-def, induct rule: list-induct2[OF len], auto)
        have  $vl: vars-mset-left\ E = vars-mset-left\ G$ 
        unfolding  $vars-mset-left-def\ E\ s\ t\ G-def$ 
        by (simp add: o-def, induct rule: list-induct2[OF len], auto)
        from  $lin$  have  $lin: lin\ G$  unfolding  $lin-def\ vr$  .
        from  $disjU$  have  $disjU: disjU\ u\ G$  unfolding  $disjU-def\ vr$  .
        from  $disjLR$  have  $disjLR: disjLR\ G$  unfolding  $disjLR-def\ vl\ vr$  .
        note  $IH = IH[OF\ lin\ disjLR\ disjU\ mgu\ linu]$ 
        show ?thesis
        by (rule IH)
      next
      case no-Fun-Fun: False
      show ?thesis
      proof (cases  $\exists\ s\ t. (s, t) \in \# E \wedge is-Var\ t$ )
        case True

```

```

then obtain  $s \ t \ F$  where  $E: E = \text{add-mset } (s, t) \ F$  and  $\text{is-Var } t$ 
  by (metis insert-DiffM)
then obtain  $x$  where  $t: t = \text{Var } x$  by (cases s; cases t; auto)
let  $? \tau = \text{subst } x \ s$ 
have  $x: x \in \# \text{ vars-mset-right } E$  unfolding  $E \ t$  by (auto simp: vars-mset-right-def)

from disjLR  $x$ 
have  $xs: x \notin \text{vars-term } s$  by (auto simp: E disjLR-def vars-mset-left-def)
from UNIF1-mono[OF UNIF1-singleton-Var-right[OF this], of F]
have unif1: UNIF1  $? \tau \ E \ (\text{subst-mset } ? \tau \ F)$  by (auto simp: E t)
{
  fix  $l \ r$ 
  assume  $(l, r) \in \# \ F$ 
  then obtain  $G$  where  $F: F = \text{add-mset } (l, r) \ G$ 
    by (metis insert-DiffM)
  from  $x \ \text{disjLR}$ 
  have  $xl: x \notin \text{vars-term } l$  by (auto simp: E F disjLR-def vars-mset-left-def)
  from lin[unfolded E t lin-def linear-mset-def, rule-format, of x]
  have  $\text{count } (\text{vars-mset-right } F) \ x = 0$ 
    by (auto simp: vars-mset-right-def)
  hence  $x \notin \# \text{ vars-mset-right } F$ 
    by (simp add: count-eq-zero-iff)
  hence  $xr: x \notin \text{vars-term } r$  by (auto simp: F vars-mset-right-def)
  note  $xl \ xr$ 
} note  $x F = \text{this}$ 
have  $\text{subst-mset } ? \tau \ F = \text{image-mset id } F$  unfolding subst-mset-def
proof (rule image-mset-cong, clarsimp)
  fix  $l \ r$ 
  assume  $(l, r) \in \# \ F$ 
  from  $x F$  [OF this]
  have  $xl: x \notin \text{vars-term } l \ x \notin \text{vars-term } r$  .
  thus  $l \cdot ? \tau = l \wedge r \cdot ? \tau = r$  by auto
qed
hence  $\text{substF}: \text{subst-mset } ? \tau \ F = F$  by auto
with unif1 have unif1: UNIF1  $? \tau \ E \ F$  by auto
note  $IH = IH$  [OF this]
from is-mgu-UNIF1 [OF unif1 mgu] obtain  $\sigma' \ \delta$ 
  where  $\text{mgu}': \text{is-mgu } \sigma' \ (\text{set-mset } E) \ \text{is-mgu } \delta \ (\text{set-mset } F)$ 
  and  $\text{id}: \sigma' = ? \tau \circ_s \delta$  by auto
from is-mgu-is-mgu-var-renaming [OF mgu'(1) mgu, unfolded id] obtain
   $\gamma$  where  $\text{sub}: ? \tau \circ_s \delta = \sigma \circ_s (\text{Var} \circ \gamma)$  by auto
have  $u \cdot \delta = u \cdot ? \tau \circ_s \delta$ 
proof (rule term-subst-eq)
  fix  $y$ 
  assume  $y \in \text{vars-term } u$ 
with disjU [unfolded disjU-def E t] have  $y \neq x$  unfolding vars-mset-right-def
by auto
  thus  $\delta \ y = (? \tau \circ_s \delta) \ y$ 
    by (auto simp: subst-compose-def subst-def)

```

```

qed
from this[unfolded sub] have uelt:  $u \cdot \delta = u \cdot \sigma \circ_s (\text{Var} \circ \gamma)$  by auto

have LFE:  $\text{vars-mset-left } F \subseteq \# \text{ vars-mset-left } E$  unfolding vars-mset-left-def
E by auto
have RFE:  $\text{vars-mset-right } F \subseteq \# \text{ vars-mset-right } E$  unfolding vars-mset-right-def
E by auto
hence count (vars-mset-right F) y  $\leq$  count (vars-mset-right E) y for y
by (rule mset-subset-eq-count)
with lin have lin: lin F unfolding lin-def linear-mset-def
using dual-order.trans by blast
from disjLR have disjLR: disjLR F unfolding disjLR-def using LFE RFE
by (meson disjoint-iff mset-subset-eqD)
from disjU have disjU: disjU u F unfolding disjU-def using RFE
by (meson disjoint-iff mset-subset-eqD)
from IH[OF lin disjLR disjU mgu'(2) linu]
have IH: linear-term (u ·  $\delta$ ) .
from IH[unfolded uelt]
have linear-term (u ·  $\sigma \cdot (\text{Var} \circ \gamma)$ ) by auto
thus linear-term (u ·  $\sigma$ ) by (rule linear-term-Var-subst)
next
case no-right-Var: False
from no-right-Var no-Fun-Fun
have left-Var:  $\bigwedge s t. (s, t) \in \# E \implies \text{is-Var } s$  by auto
show ?thesis
proof (cases  $E = \{\#\}$ )
case True
hence is-mgu Var (set-mset E) by auto
from is-mgu-is-mgu-var-renaming[OF this mgu]
obtain  $\gamma$  where id:  $\text{Var} = \sigma \circ_s (\text{Var} \circ \gamma)$  by auto
have linear-term (u · Var) using linu by auto
hence linear-term (u ·  $\sigma \cdot (\text{Var} \circ \gamma)$ )
by (subst (asm) id, auto)
thus linear-term (u ·  $\sigma$ ) by (rule linear-term-Var-subst)
next
case False
then obtain s t where (s, t)  $\in \# E$  by auto
with left-Var[OF this] obtain x where
(Var x, t)  $\in \# E$  by auto
then obtain F where E:  $E = \text{add-mset } (\text{Var } x, t) F$  by (rule mset-add)
let ? $\tau$  = subst x t
let ?F = subst-mset ? $\tau$  F
let ?F' = image-mset ( $\lambda p. (\text{fst } p \cdot ?\tau, \text{snd } p)$ ) F
from disjLR
have xt:  $x \notin \text{vars-term } t$  by (auto simp: E disjLR-def vars-mset-left-def
vars-mset-right-def)
from UNIF1-mono[OF UNIF1-singleton-Var-left[OF this], of F]
have unif1: UNIF1 (subst x t) E ?F by (auto simp: E)
have ?F = ?F'

```

```

    unfolding subst-mset-def
  proof (rule image-mset-cong, clarsimp)
    fix l r
    assume lr: (l,r) ∈# F
    have r · ?τ = r · Var
    proof (rule term-subst-eq)
      fix y
      assume y ∈ vars-term r
      with disjLR[unfolded E disjLR-def] lr have x ≠ y
        by (auto simp: vars-mset-left-def vars-mset-right-def)
      thus ?τ y = Var y by (simp add: subst-def)
    qed
    thus r · ?τ = r by auto
  qed
  note unif1 = unif1[unfolded this]
  note IH = IH[OF this]
  have vr: vars-mset-right ?F' = vars-mset-right F
    unfolding vars-mset-right-def by (induct F, auto)
  have vl: set-mset (vars-mset-left ?F') ⊆ set-mset(vars-mset-left F) ∪
vars-term t
    unfolding vars-mset-left-def by (auto simp: vars-term-subst subst-def
split: if-splits)
  have vars-term-ms t ⊆# vars-mset-right E unfolding E by auto
  hence count (vars-term-ms t) y ≤ count (vars-mset-right E) y for y
    by (rule mset-subset-eq-count)
  with lin have linear-mset (vars-term-ms t) unfolding lin-def lin-
ear-mset-def
    using dual-order.trans by blast
  hence lint: linear-term t
    using linear-vars-term-ms by auto
  have RFE: vars-mset-right ?F' ⊆# vars-mset-right E
    unfolding vr unfolding vars-mset-right-def E by auto
  hence count (vars-mset-right ?F') y ≤ count (vars-mset-right E) y for y
    by (rule mset-subset-eq-count)
  with lin have lin': lin ?F' unfolding lin-def linear-mset-def
    using dual-order.trans by blast
  {
    fix y
    assume y: y ∈# vars-mset-right F y ∈ vars-term t
    hence y ∈# vars-term-ms t by auto
  }
  with y(1) have {#y,y#} ⊆# vars-mset-right E unfolding E vars-mset-right-add
snd-conv
    by (metis add-mset-add-single single-subset-iff subset-mset.add-mono)
  from mset-subset-eq-count[OF this, of y]
    lin[unfolded lin-def linear-mset-def, rule-format, of y]
    have False by simp
  } note disj-F-t = this

  have disjLR': disjLR ?F'

```

```

proof (rule ccontr)
  assume  $\neg ?thesis$ 
  from this[unfolded disjLR-def vr, simplified]
  obtain y where y:  $y \in \# \text{ vars-mset-left } ?F' \ y \in \# \text{ vars-mset-right } F$ 
    by auto
  with vl have disj:  $y \in \# \text{ vars-mset-left } F \vee y \in \text{ vars-term } t$  by auto
  from y(2) disjLR[unfolded E disjLR-def]
  have  $y \notin \text{ set-mset } (\text{ vars-mset-left } (\text{ add-mset } (\text{ Var } x, t) F))$ 
    by (auto simp: vars-mset-right-def)
  with disj have  $y \in \text{ vars-term } t$  unfolding vars-mset-left-def by auto
  from disj-F-t[OF y(2) this]
  show False .
qed
let ?v = u · ? $\tau$ 
have disjU':  $\text{ disjU } ?v ?F'$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  from this[unfolded disjU-def vr] obtain y
    where y:  $y \in \text{ vars-term } ?v \ y \in \# \text{ vars-mset-right } F$ 
    by auto
  from y(1) have  $y \in \text{ vars-term } u \vee y \in \text{ vars-term } t$ 
    by (auto simp: vars-term-subst subst-def split: if-splits)
  with disj-F-t[OF y(2)] have  $y \in \text{ vars-term } u$  by auto
  with disjU[unfolded disjU-def] y(2)
  show False unfolding E by auto
qed
have linu':  $\text{ linear-term } ?v$ 
proof (rule linear-term-subst[OF linu])
  show  $\text{ linear-term } (? \tau \ y)$  for y using lint by (auto simp: subst-def)
  fix y z
  assume yz:  $y \in \text{ vars-term } u \ z \in \text{ vars-term } u \ y \neq z$ 
  with disjU[unfolded disjU-def]
  have  $y \notin \# \text{ vars-mset-right } E \ z \notin \# \text{ vars-mset-right } E$  by auto
  hence yzt:  $y \notin \text{ vars-term } t \ z \notin \text{ vars-term } t$  unfolding E by auto
  show  $\text{ vars-term } (? \tau \ y) \cap \text{ vars-term } (? \tau \ z) = \{\}$ 
  proof (cases  $y = x \vee z = x$ )
    case False
    thus ?thesis using yz by (auto simp: subst-def)
  next
    case True
    thus ?thesis
  proof
    assume yx:  $y = x$ 
    with yz have  $z \neq x$  by auto
    hence one:  $\text{ vars-term } (? \tau \ z) = \{z\}$  by (auto simp: subst-def)
    from yx have two:  $\text{ vars-term } (? \tau \ y) = \text{ vars-term } t$  by simp
    from one two yzt show ?thesis by auto
  next
    assume zx:  $z = x$ 

```

```

    with yz have  $y \neq x$  by auto
    hence one: vars-term ( $? \tau y$ ) =  $\{y\}$  by (auto simp: subst-def)
    from zx have two: vars-term ( $? \tau z$ ) = vars-term  $t$  by simp
    from one two yz show ?thesis by auto
  qed
qed
qed
note IH = IH[OF lin' disjLR' disjU' - linu']

from is-mgu-UNIF1[OF unif1 mgu] obtain  $\sigma' \delta$ 
  where mgu': is-mgu  $\sigma'$  (set-mset  $E$ ) is-mgu  $\delta$  (set-mset  $?F'$ )
  and id:  $\sigma' = ? \tau \circ_s \delta$  by auto
from is-mgu-is-mgu-var-renaming[OF mgu'(1) mgu, unfolded id] obtain
   $\gamma$  where sub:  $? \tau \circ_s \delta = \sigma \circ_s (\text{Var} \circ \gamma)$  by auto
let  $? \gamma = \text{Var} \circ \gamma :: ('f, 'v)\text{subst}$ 
from IH[OF mgu'(2)] have IH: linear-term ( $u \cdot ? \tau \circ_s \delta$ ) by simp
from this[unfolded sub]
  have linear-term ( $u \cdot \sigma \cdot ? \gamma$ ) by simp
  thus linear-term ( $u \cdot \sigma$ ) by (rule linear-term-Var-subst)
qed
qed
qed
qed
qed

```

lemma *right-linear-var-disjoint-mgu*: fixes $s t :: ('f, 'v)\text{term}$
 assumes disj: vars-term $s \cap \text{vars-term } t = \{\}$
 and lin: linear-term t
 and mgu: is-mgu $\sigma \{(s, t)\}$
 and linu: linear-term u
 and disju: vars-term $u \cap \text{vars-term } t = \{\}$
 shows linear-term ($u \cdot \sigma$)
proof (rule right-linear-var-disjoint-mgu-mset[of $\{\#(s, t)\# \} \sigma u$, OF - - - linu])
 show is-mgu σ (set-mset $\{\#(s, t)\# \}$) using mgu by auto
 show linear-mset (vars-mset-right $\{\#(s, t)\# \}$) using lin
 unfolding vars-mset-right-def by (auto simp: linear-vars-term-ms lin)
 show vars-term $u \cap \text{set-mset} (\text{vars-mset-right } \{\#(s, t)\# \}) = \{\}$
 using disj unfolding vars-mset-right-def by auto
 show set-mset (vars-mset-left $\{\#(s, t)\# \}) \cap \text{set-mset} (\text{vars-mset-right } \{\#(s, t)\# \}) = \{\}$
 using disj unfolding vars-mset-left-def vars-mset-right-def by auto
qed

Corollary: Unification of two linear var disjoint terms results in a linear substitution and linear unified terms.

lemma *linear-var-disjoint-is-mgu*: fixes $s t :: ('f, 'v)\text{term}$
 assumes disj: vars-term $s \cap \text{vars-term } t = \{\}$
 and lin: linear-term s linear-term t
 and mgu: is-mgu $\sigma \{(s, t)\}$

```

shows vars-term  $u \cap$  vars-term  $t = \{\}$   $\implies$  linear-term  $u \implies$  linear-term  $(u \cdot \sigma)$ 
  and vars-term  $u \cap$  vars-term  $s = \{\}$   $\implies$  linear-term  $u \implies$  linear-term  $(u \cdot \sigma)$ 
  and linear-term  $(s \cdot \sigma)$  linear-term  $(t \cdot \sigma)$ 
  and linear-term  $(\sigma x)$ 
proof -
  from disj have disj': vars-term  $t \cap$  vars-term  $s = \{\}$  by auto
  from mgu have mgu': is-mgu  $\sigma \{(t,s)\}$  by (simp add: is-mgu-insert-swap)
  note one = right-linear-var-disjoint-mgu[OF disj lin(2) mgu]
  note two = right-linear-var-disjoint-mgu[OF disj' lin(1) mgu']
  from one[OF lin(1) disj] show linear-term  $(s \cdot \sigma)$  .
  from two[OF lin(2) disj'] show linear-term  $(t \cdot \sigma)$  .
  let  $?x = \text{Var } x :: (f',v)\text{term}$ 
  from disj have vars-term  $?x \cap$  vars-term  $s = \{\} \vee$  vars-term  $?x \cap$  vars-term  $t$ 
  =  $\{\}$  by auto
  with one[of ?x] two[of ?x] show linear-term  $(\sigma x)$  by auto
  show vars-term  $u \cap$  vars-term  $t = \{\} \implies$  linear-term  $u \implies$  linear-term  $(u \cdot \sigma)$ 
    using one by metis
  show vars-term  $u \cap$  vars-term  $s = \{\} \implies$  linear-term  $u \implies$  linear-term  $(u \cdot \sigma)$ 
    using two by metis
qed
end

```

4 Narrowing and Linearity Preservation

In this theory we define narrowing and show that narrowing of a linear term with a right-linear TRS results in a linear term again. Moreover, some further results on narrowing are formalized.

```

theory Linear-Narrowing
imports
  Automatic-Refinement.Misc
  Linear-Unification
  First-Order-Rewriting.Trs
begin

```

4.1 Preparations for Narrowing

```

definition subst-term-mset ::  $(f',v)\text{term} \Rightarrow (f',v)\text{subst} \Rightarrow (f',v)\text{term multiset}$ 
  where subst-term-mset  $t \sigma = \text{image-mset } \sigma (\text{vars-term-ms } t)$ 

```

lemma vars-term-ms-subst-compose-split:

```

   $(\delta \text{ '# } (\sum x \in \#xs. \text{vars-term-ms } (\mu x)), (\mu \circ_s \delta) \text{ '# } xs) \in (\text{mult } \{\triangleleft\})^=$ 
   $\wedge (\text{Ball } (\mu \text{ ' set-mset } xs) \text{ is-Var} \vee (\delta \text{ '# } (\sum x \in \#xs. \text{vars-term-ms } (\mu x)), (\mu \circ_s$ 
 $\delta) \text{ '# } xs) \in \text{mult } \{\triangleleft\})$ 
   $(\text{is } - \in ?R^= \wedge -)$ 
proof (induct xs)
  case empty
  then show ?case by auto
next

```

```

case (add x xs)
define a where a = {# (μ ∘s δ) x #}
define b where b = (μ ∘s δ) ‘# xs
define c where c = δ ‘# vars-term-ms (μ x)
define d where d = δ ‘# (∑ x∈#xs. vars-term-ms (μ x))
have ab: (μ ∘s δ) ‘# add-mset x xs = a + b by (simp add: a-def b-def)
have cd: δ ‘# (∑ x∈#add-mset x xs. vars-term-ms (μ x)) = c + d by (simp
add: c-def d-def)
have tr: trans {<} by (simp add: trans-supt)
have irr: irrefl-on T {<} for T :: ('a,'b)term set by (simp add: irrefl-onI)
have ext: (x,y) ∈ ?R ⇒ (x + z, y + z) ∈ ?R for x y z
  using mult-cancel[OF tr irr, of x z y] by auto
hence exte: (x,y) ∈ ?R= ⇒ (x + z, y + z) ∈ ?R= for x y z
  by auto
from add
have (d,b) ∈ ?R= Ball (μ ‘ set-mset xs) is-Var ∨ (d,b) ∈ ?R by (auto simp:
b-def d-def)
from exte[OF this(1), of c] ext[of d b c] this(2)
have IH: (c + d, c + b) ∈ ?R= Ball (μ ‘ set-mset xs) is-Var ∨ (c + d, c + b)
∈ ?R
  by (auto simp: ac-simps)

have step-main: (c,a) ∈ ?R= ∧ (is-Var (μ x) ∨ (c,a) ∈ ?R)
proof (cases μ x)
  case (Var y)
  thus ?thesis by (auto simp: subst-compose-def c-def a-def)
next
  case (Fun f ts)
  have (c,a) ∈ mult1 {<}
    by (auto simp: Fun a-def c-def subst-compose-def intro!: mult1I)
    (metis eval-term.simps(2) subst-image-subterm term.set-intros(4))
  thus ?thesis by (auto simp add: mult-def)
qed
with ext[of c a b] exte[of c a b]
have step: (c + b, a + b) ∈ ?R= is-Var (μ x) ∨ (c + b, a + b) ∈ ?R by auto

have one: (c + d, a + b) ∈ ?R=
  using IH step
  by (meson tr transE trans-mult trans-on-reflcl)
have two: Ball (μ ‘ set-mset (add-mset x xs)) is-Var ∨ (c + d, a + b) ∈ ?R
proof (cases is-Var (μ x))
  case False
  from step(2) False have (c + b, a + b) ∈ ?R by auto
  with IH(1) have (c + d, a + b) ∈ ?R
    by (metis Un-iff pair-in-Id-conv transD trans-mult)
  thus ?thesis by auto
next
  case x: True
  show ?thesis

```

```

proof (cases Ball ( $\mu$  ' set-mset xs) is-Var)
  case True
    with x show ?thesis by auto
  next
    case False
    with IH(2) have (c + d, c + b)  $\in$  ?R by auto
    with step(1) have (c + d, a + b)  $\in$  ?R
      by (metis Un-iff pair-in-Id-conv transD trans-mult)
    thus ?thesis by auto
  qed
qed

from one two show ?case
  unfolding cd ab by auto
qed

lemma linear-term-vars-term-ms: assumes vars-term r  $\subseteq$  vars-term l
  and linear-term r
  shows vars-term-ms r  $\subseteq$  # vars-term-ms l
proof (intro mset-subset-eqI, rule ccontr)
  fix x
  assume not:  $\neg$  count (vars-term-ms r) x  $\leq$  count (vars-term-ms l) x
  hence c: count (vars-term-ms r) x  $\geq$  1 by linarith
  hence x  $\in$  vars-term r by (auto simp: not-in-iff)
  with assms have x  $\in$  vars-term l by auto
  hence cl: count (vars-term-ms l) x  $\geq$  1 by (auto simp: not-in-iff)
  with not have count (vars-term-ms r) x  $>$  1 by linarith
  with linear-term-count[OF assms(2), of x] show False by auto
qed

lemma vars-term-ms-map-vars-term[simp]: vars-term-ms (map-vars-term f t) = f
  ' # vars-term-ms t
proof (induct t)
  case (Fun f ts)
    thus ?case by (simp, induct ts, auto)
qed auto

lemma linear-term-map-inj-on: assumes linear-term (map-vars-term f t)
  shows inj-on f (vars-term t)
proof (rule ccontr)
  let ?ft = map-vars-term f t
  from assms have linear-mset (vars-term-ms ?ft)
    using linear-vars-term-ms by blast
  also have vars-term-ms ?ft = f ' # vars-term-ms t by simp
  finally have lin: linear-mset (f ' # vars-term-ms t) .
  assume  $\neg$  ?thesis
  then obtain x y where xy: x  $\in$  # vars-term-ms t y  $\in$  # vars-term-ms t x  $\neq$  y f
  x = f y
    unfolding inj-on-def by auto

```

```

define xs where xs = vars-term-ms t - {#x, y#}
from xy(1-3) have vars-term-ms t = {#x, y#} + xs unfolding xs-def
by (metis diff-union-swap insert-DiffM2 insert-subset-eq-iff mset-subset-eq-single
    subset-eq-diff-conv subset-mset.add-diff-inverse)
from arg-cong[OF this, of ('#) f] xy(4)
have f '# vars-term-ms t = {# f x, f x #} + f '# xs by auto
hence count (f '# vars-term-ms t) (f x) ≥ 2 by auto
with lin[unfolded linear-mset-def, rule-format, of f x] show False by auto
qed

```

4.2 Narrowing Relations

```

context
  fixes ren :: 'v :: infinite renaming2
begin

```

definition

```

narrows-r-p-s :: ('f, 'v) trs ⇒ ('f, 'v) rule ⇒ pos ⇒ ('f, 'v) subst ⇒ ('f, 'v) trs
where
narrows-r-p-s R r p μ ≡ {(s, t). ∃ μ2. p ∈ poss s ∧ is-Fun (s |- p) ∧ r ∈ R ∧
```

$$mgu\text{-}vd\ ren\ (s\ |\text{-}\ p)\ (fst\ r) = Some\ (\mu,\ \mu2)$$

```

  ∧ t = replace-at (s · μ) p (snd r · μ2)}

```

definition

```

narrow-step-s :: ('f, 'v) trs ⇒ ('f, 'v) subst ⇒ ('f, 'v) trs where
narrow-step-s R μ = {st | st lr p. st ∈ narrows-r-p-s R lr p μ}

```

definition

```

narrow-step R = {st | st lr p μ. st ∈ narrows-r-p-s R lr p μ}

```

```

lemma narrow-step-to-s: narrow-step R = ⋃ (range (narrow-step-s R))
unfolding narrow-step-def narrow-step-s-def by auto

```

theorem

```

right-linear-rule-narrowing: fixes R :: ('f, 'v) trs
assumes linr: linear-term r
and lins: linear-term s
and narr: (s, t) ∈ narrows-r-p-s R (l, r) p μ
shows linear-term t

```

proof –

```

define r1 where r1 = rename-1 ren
define r2 where r2 = rename-2 ren
let ?r1 = map-vars-term r1
let ?r2 = map-vars-term r2
define s' where s' = ?r1 s
define l' where l' = ?r2 l
define r' where r' = ?r2 r
define sp' where sp' = ?r1 (s |- p)
from narr[unfolded narrows-r-p-s-def] obtain μ2
where p: p ∈ poss s

```

```

    and mgu: mgu-vd ren (s |- p) l = Some ( $\mu$ ,  $\mu 2$ )
    and t: t = replace-at (s ·  $\mu$ ) p (r ·  $\mu 2$ )
  by auto
from p have sp'-id: ?r1 (s |- p) = sp' unfolding sp'-def by simp
with mgu[unfolded mgu-vd-def mgu-var-disjoint-generic-def,
        folded r1-def r2-def]
obtain  $\gamma$  where mgu: mgu sp' l' = Some  $\gamma$ 
  and  $\mu$ :  $\mu$  =  $\gamma \circ r1$  and  $\mu 2$ :  $\mu 2$  =  $\gamma \circ r2$ 
  by (auto split: option.splits simp: l'-def)
have  $\mu$ : u ·  $\mu$  = ?r1 u ·  $\gamma$  for u unfolding  $\mu$  o-def
  by (metis eval-eq-map-vars)
have  $\mu 2$ : u ·  $\mu 2$  = ?r2 u ·  $\gamma$  for u unfolding  $\mu 2$  o-def
  by (metis eval-eq-map-vars)
from lins p have linsp: linear-term (s |- p) by (rule subt-at-linear)
from mgu-sound[OF mgu] have imgu: is-imgu  $\gamma$  {(sp', l')} by auto
hence is-mgu  $\gamma$  {(sp', l')} by (rule is-imgu-imp-is-mgu)
hence mgu: is-mgu  $\gamma$  {(l', sp')}
  using is-mgu-insert-swap by blast
note ren = rename-12[of ren, folded r1-def r2-def]
have vars-term l' ∩ vars-term sp' = {}
  using ren(3) unfolding map-vars-term-eq vars-term-subst l'-def sp'-def
  by auto
note main = right-linear-var-disjoint-mgu[OF this - mgu]
from linsp have linsp': linear-term sp' using ren(1) unfolding sp'-def
  by (metis inj-on-subset linear-term-map-inj-on-linear-term top-greatest)

have main: linear-term u  $\implies$ 
  vars-term u ∩ vars-term sp' = {}  $\implies$ 
  linear-term (u ·  $\gamma$ ) for u
  using main[OF linsp', of u] by (simp add: term.set-map)
have p': p ∈ poss s' using p unfolding s'-def by auto

define u where u = replace-at s' p r'
from t[unfolded  $\mu$   $\mu 2$ , folded r'-def s'-def]
have t = replace-at (s' ·  $\gamma$ ) p (r' ·  $\gamma$ ) by simp
also have ... = u ·  $\gamma$  using p'
  by (simp add: ctxt-of-pos-term-subst u-def)
finally have t: t = u ·  $\gamma$  .

have sp': sp' = s' |- p unfolding sp'-def s'-def using p by auto
show ?thesis unfolding t
proof (rule main)
  have vsu: vars-term-ms u = vars-ctxt-ms (ctxt-of-pos-term p s') + vars-term-ms
r'
    unfolding u-def vars-term-ms-ctxt-apply by auto
  have vss: vars-term-ms s' = vars-ctxt-ms (ctxt-of-pos-term p s') + vars-term-ms
sp'
    using p' unfolding sp' by (metis ctxt-supt-id vars-term-ms-ctxt-apply)
  have lins': linear-term s' using lins unfolding s'-def using ren(1)

```

```

    by (simp add: inj-on-def linear-term-map-inj-on-linear-term)
show linear-term u
  unfolding linear-vars-term-ms[symmetric]
  unfolding linear-mset-def
proof
  fix x
  {
    assume  $x \in \# \text{ vars-ctxt-ms } (\text{ctxt-of-pos-term } p \ s')$ 
    hence  $x \in \text{vars-term } s'$  using  $p'$ 
    by (metis set-mset-vars-term-ms union-iff vss)
    hence  $x \in \text{range } r1$  unfolding  $s'$ -def
    by (auto simp: term.set-map)
  } note one = this
  {
    assume  $x \in \# \text{ vars-term-ms } r'$ 
    hence  $x \in \text{range } r2$  unfolding  $r'$ -def
    by (auto simp: term.set-map)
  } note two = this
  show  $\text{count } (\text{vars-term-ms } u) \ x \leq 1$ 
  proof (cases  $x \in \# \text{ vars-ctxt-ms } (\text{ctxt-of-pos-term } p \ s')$ )
    case True
    from one[OF this] two ren have  $x \notin \# \text{ vars-term-ms } r'$  by auto
    hence  $\text{count } (\text{vars-term-ms } u) \ x = \text{count } (\text{vars-ctxt-ms } (\text{ctxt-of-pos-term } p \ s')) \ x$ 
    unfolding  $vsu$  using not-in-iff by fastforce
    also have  $\dots \leq \text{count } (\text{vars-term-ms } s') \ x$  unfolding  $vss$  by simp
    also have  $\dots \leq 1$  using  $lins'$  unfolding linear-vars-term-ms[symmetric]
  linear-mset-def
    by simp
    finally show ?thesis .
  next
  case False
  from  $linr$  have  $linr'$ :  $\text{linear-term } r'$  unfolding  $r'$ -def using  $ren(2)$ 
  by (simp add: inj-on-def linear-term-map-inj-on-linear-term)
  from False have  $\text{count } (\text{vars-term-ms } u) \ x = \text{count } (\text{vars-term-ms } r') \ x$ 
  unfolding  $vsu$  using not-in-iff by fastforce
  also have  $\dots \leq 1$  using  $linr'$  unfolding linear-vars-term-ms[symmetric]
  linear-mset-def
    by simp
    finally show ?thesis .
  qed
qed

show  $\text{vars-term } u \cap \text{vars-term } sp' = \{\}$ 
proof (rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain  $x$  where  $x: x \in \# \text{ vars-term-ms } u \ x \in \# \text{ vars-term-ms } sp'$  by
  auto
  with  $vsu$  have  $disj: x \in \# \text{ vars-ctxt-ms } (\text{ctxt-of-pos-term } p \ s') \vee x \in \#$ 

```

$\text{vars-term-ms } r'$ **by** *auto*
from $x(2)$ **have** $x \in \text{range } r1$ **unfolding** sp'-def **by** (*auto simp: term.set-map*)
with ren **have** $x \notin \text{range } r2$ **by** *auto*
hence $x \notin \text{vars-term } r'$ **unfolding** $r'\text{-def}$ **by** (*auto simp: term.set-map*)
with disj **have** $x \in \# \text{vars-ctxt-ms } (\text{ctxt-of-pos-term } p \ s')$ **by** *auto*
with $x(2)$ **have** $\{\#x, x\# \} \subseteq \# \text{vars-term-ms } s'$ **unfolding** vss
by (*metis add-mset-add-single single-subset-iff subset-mset.add-mono*)
from $\text{mset-subset-eq-count}$ [*OF this, of x*]
have $\text{count } (\text{vars-term-ms } s') \ x \geq 2$
by *auto*
hence $\neg \text{linear-mset } (\text{vars-term-ms } s')$ **unfolding** linear-mset-def
by (*metis Suc-1 Suc-le-eq linorder-not-less*)
with $\text{lin}s'$ **show** *False* **using** $\text{linear-vars-term-ms}$ **by** *blast*
qed
qed
qed

theorem *right-linear-rule-narrow-step*:
assumes $\bigwedge \text{lr}. \text{lr} \in R \implies \text{linear-term } (\text{snd } \text{lr})$
and $\text{lin}s$: *linear-term* s
and narr : $(s, t) \in \text{narrow-step } R$
shows *linear-term* t
proof –
from narr [*unfolded narrow-step-def*] **obtain** $l \ r \ p \ \mu$
where $\text{step}: (s, t) \in \text{narrows-r-p-s } R \ (l, r) \ p \ \mu$ **by** *auto*
hence $(l, r) \in R$ **unfolding** narrows-r-p-s-def **by** *auto*
from $\text{assms}(1)$ [*OF this*] **have** *linear-term* r **by** *auto*
from *right-linear-rule-narrowing* [*OF this lin}s \text{step}*]
show *?thesis* .
qed

lemma *narrowing-right-linear-one-step-simulation-r-p-s*: **fixes** $R :: ('f, 'v)\text{trs}$
assumes $(s \cdot \sigma, t) \in \text{rstep-r-p-s } R \ (l, r) \ p \ \mu$
and $\text{lin}s$: *linear-term* s
and $\text{lin}r$: *linear-term* r
and var-cond : $\text{vars-term } r \subseteq \text{vars-term } l$
shows $p \notin \text{fun-poss } s \wedge$
 $(\exists \delta. t = s \cdot \delta \wedge (\text{subst-term-mset } s \ \delta, \text{subst-term-mset } s \ \sigma) \in \text{mult1}$
 $((\text{rstep } R)^{\wedge -1}))$
 $\vee$
 $p \in \text{fun-poss } s \wedge$
 $(\exists \mu1 \ \mu2 \ \delta \ u. t = u \cdot \delta \wedge \text{linear-term } u \wedge (s, u) \in \text{narrows-r-p-s } R \ (l, r) \ p$
 $\mu1 \wedge \sigma = \mu1 \circ_s \delta \wedge \mu = \mu2 \circ_s \delta$
 $\wedge (\text{subst-term-mset } u \ \delta, \text{subst-term-mset } s \ \sigma) \in (\text{mult } \{\triangleleft\})^=)$

proof –
from $\text{assms}(1)$ [*unfolded rstep-r-p-s-def' Let-def, simplified*]
have $p: p \in \text{poss } (s \cdot \sigma)$ **and** $\text{lr}: (l, r) \in R$ **and** $\text{ssig}: s \cdot \sigma \vdash p = l \cdot \mu$
and $t: t = \text{replace-at } (s \cdot \sigma) \ p \ (r \cdot \mu)$

```

    by auto
  show ?thesis
  proof (cases  $p \in \text{poss } s \wedge \text{is-Fun } (s \mid - p)$ )
    case False
    from pos-into-subst[OF refl p this]
    obtain q1 q2 where pq:  $p = q1 @ q2$  and q1:  $q1 \in \text{poss } s$  and is-Var ( $s \mid -$ 
q1) by auto
    then obtain x where sq1:  $s \mid - q1 = \text{Var } x$  by auto
    define C where  $C = \text{ctxt-of-pos-term } q1 \ s$ 
    from q1 have ssig':  $s \cdot \sigma = (C \cdot_c \sigma) \langle \sigma \ x \rangle$ 
    using sq1 by (metis ctxt-supt-id eval-ctxt eval-term.simps(1) C-def)
    have q2:  $q2 \in \text{poss } (\sigma \ x)$  using p pq sq1 by (simp add: q1)
    have match:  $\sigma \ x \mid - q2 = l \cdot \mu$  unfolding C-def ssig'[unfolded ssig' pq, symmetric]

    using p pq sq1 q1 q2 by (metis C-def eval-term.simps(1) ssig' subt-at-subst
subterm-poss-conv)
    define u where  $u = \text{replace-at } (\sigma \ x) \ q2 \ (r \cdot \mu)$ 
    have  $(\sigma \ x, u) \in \text{rstep-r-p-s } R \ (l, r) \ q2 \ \mu$ 
    unfolding u-def rstep-r-p-s-def Let-def using lr match q2
    by (auto intro: replace-at-ident)
    hence step:  $(\sigma \ x, u) \in \text{rstep } R$  by (rule rstep-r-p-s-imp-rstep)
    have ctxt:  $\text{ctxt-of-pos-term } p \ (s \cdot \sigma) = (\text{ctxt-of-pos-term } q1 \ (s \cdot \sigma)) \circ_c$ 
 $\text{ctxt-of-pos-term } q2 \ (\sigma \ x)$ 
    unfolding pq by (subst ctxt-of-pos-term-append, insert q1 sq1, auto)
    define  $\delta$  where  $\delta = \sigma(x := u)$ 
    have  $t: t = \text{replace-at } (s \cdot \sigma) \ q1 \ u$ 
    unfolding t u-def unfolding ctxt by simp
    note repl = linear-term-replace-in-subst[OF lins q1 sq1, of  $\sigma \ \delta \ u$ , folded t]
    have  $t: t = s \cdot \delta$ 
    by (rule repl, auto simp:  $\delta$ -def)
    show ?thesis unfolding fun-poss-poss
    proof (rule disjI1, intro conjI[OF False] exI[of -  $\delta$ ] conjI t)
      from q1 sq1 have  $x \in \text{vars-term } s$ 
      by (metis subt-at-imp-supteq subteq-Var-imp-in-vars-term)
      hence  $x \in \# \text{vars-term-ms } s$  by auto
      then obtain xs where vars:  $\text{vars-term-ms } s = \text{add-mset } x \ xs$  by (rule
mset-add)
      from lins have linear-mset ( $\text{vars-term-ms } s$ )
      by (simp add: linear-vars-term-ms)
      from this[unfolded linear-mset-def vars, rule-format, of x]
      have xs:  $x \notin \# xs$  by (simp add: not-in-iff)
      have so:  $\text{subst-term-mset } s \ \sigma = \text{add-mset } (\sigma \ x) \ (\text{image-mset } \sigma \ xs)$ 
      unfolding vars subst-term-mset-def by auto
      have s $\delta$ :  $\text{subst-term-mset } s \ \delta = \text{add-mset } u \ (\text{image-mset } \sigma \ xs)$ 
      unfolding vars subst-term-mset-def using xs by (auto simp:  $\delta$ -def, induct
xs, auto)
      show  $(\text{subst-term-mset } s \ \delta, \text{subst-term-mset } s \ \sigma) \in \text{mult1 } ((\text{rstep } R)^{\wedge - 1})$ 
      unfolding so s $\delta$ 
      by (rule mult1I[of -  $\sigma \ x \ \text{image-mset } \sigma \ xs - \{\#u\}$  ( $\text{rstep } R)^{\wedge - 1}$ ], insert

```

```

step, auto)
qed
next
case True
hence  $p: p \in \text{poss } s \text{ is-Fun } (s \mid p)$  by auto
from  $\text{ssig } p$  have  $s \mid p \cdot \sigma = l \cdot \mu$  by simp
from  $\text{mgu-vd-complete}[OF \text{ this, of ren}]$  obtain
 $\mu1 \ \mu2 \ \delta$  where  $\text{mgu: mgu-vd ren } (s \mid p) \ l = \text{Some } (\mu1, \mu2)$ 
and  $\text{sig: } \sigma = \mu1 \circ_s \delta$ 
and  $\text{mu: } \mu = \mu2 \circ_s \delta$ 
and  $\text{unif: } s \mid p \cdot \mu1 = l \cdot \mu2$ 
by auto
define  $u$  where  $u = \text{replace-at } (s \cdot \mu1) \ p \ (r \cdot \mu2)$ 
have  $\text{su: } (s, u) \in \text{narrows-r-p-s } R \ (l, r) \ p \ \mu1$ 
unfolding  $\text{narrows-r-p-s-def } u\text{-def}$  using  $\text{mgu } p \ lr$  by auto
from  $\text{right-linear-rule-narrowing}[OF \text{ linr lins su}]$ 
have  $\text{linu: linear-term } u$  .
define  $C$  where  $C = \text{ctxt-of-pos-term } p \ s$ 
from  $p$  have  $\text{ctxt: ctxt-of-pos-term } p \ (s \cdot \sigma) = C \cdot_c \sigma$  for  $\sigma :: (f, v)\text{subst}$ 
by  $(\text{metis ctxt-of-pos-term-subst } C\text{-def})$ 
have  $\text{tu: } t = u \cdot \delta$  unfolding  $t \ u\text{-def } \text{ctxt}$ 
unfolding  $\text{sig } \mu$  by simp
show ?thesis unfolding  $\text{fun-poss-poss}$ 
proof (rule  $\text{disjI2}$ , intro  $\text{exI}[of \ - \ u] \ \text{exI } \text{conjI } \text{linu } p$ )
show  $t = u \cdot \delta$  by fact
show  $(s, u) \in \text{narrows-r-p-s } R \ (l, r) \ p \ \mu1$  by fact
show  $\sigma = \mu1 \circ_s \delta$  by fact
show  $\mu = \mu2 \circ_s \delta$  by fact
define  $VS$  where  $VS = \text{vars-term } s$ 
define  $\text{mid-set}$  where  $\text{mid-set} = \delta \ \# \ \text{vars-term-ms } (s \cdot \mu1)$ 
have  $\text{subst-u: subst-term-mset } u \ \delta = \delta \ \# \ \text{vars-ctxt-ms } (C \cdot_c \mu1) + \delta \ \#$ 
 $\text{vars-term-ms } (r \cdot \mu2)$ 
unfolding  $u\text{-def } \text{subst-term-mset-def } \text{ctxt } \text{vars-term-ms-ctxt-apply } o\text{-def}$  by
simp
have  $\text{subst-term-mset } s \ \sigma = (\mu1 \circ_s \delta) \ \# \ \text{vars-term-ms } s$  unfolding  $\text{subst-term-mset-def}$ 
sig by auto
also have  $(\text{mid-set}, \dots) \in (\text{mult } \{\triangleleft\})^= \wedge$ 
 $(\text{Ball } (\mu1 \ \# \ \text{set-mset } (\text{vars-term-ms } s)) \ \text{is-Var} \vee (\text{mid-set}, \dots) \in \text{mult } \{\triangleleft\})$ 
unfolding  $\text{vars-term-ms-subst } \text{mid-set-def}$ 
by  $(\text{rule } \text{vars-term-ms-subst-compose-split})$ 
finally have one:  $(\text{mid-set}, \text{subst-term-mset } s \ \sigma) \in (\text{mult } \{\triangleleft\})^=$ 
 $\text{Ball } (\mu1 \ \# \ VS) \ \text{is-Var} \vee (\text{mid-set}, \text{subst-term-mset } s \ \sigma) \in \text{mult } \{\triangleleft\}$ 
by  $(\text{auto simp: } VS\text{-def})$ 

have  $s\text{-split: } s = C \langle s \mid p \rangle$  using  $p$  unfolding  $C\text{-def}$ 
by  $(\text{simp add: ctxt-supt-id})$ 
from  $\text{arg-cong}[OF \text{ this, of } \lambda \ t. \ t \cdot \mu1]$ 
have  $s\mu1: s \cdot \mu1 = (C \cdot_c \mu1) \langle s \mid p \cdot \mu1 \rangle$  by simp
from  $\text{linear-term-vars-term-ms}[OF \text{ var-cond linr}]$ 

```

```

have vars-rl: vars-term-ms r  $\subseteq$  # vars-term-ms l .
then obtain xs where vars-l: vars-term-ms l = vars-term-ms r + xs
by (metis subset-mset.add-diff-inverse)

define VS' where VS' = ( $\sum x \in \#xs.$  vars-term-ms ( $\mu 2$  x))
have mid-set =  $\delta$  ' # vars-ctxt-ms (C ·c  $\mu 1$ ) +  $\delta$  ' # vars-term-ms (l ·  $\mu 2$ )
unfolding mid-set-def s $\mu 1$  vars-term-ms-ctxt-apply unif by auto
also have ... =  $\delta$  ' # vars-ctxt-ms (C ·c  $\mu 1$ ) +  $\delta$  ' # vars-term-ms (r ·  $\mu 2$ )
+  $\delta$  ' # VS'
by (simp add: vars-l VS'-def)
also have ... = subst-term-mset u  $\delta$  +  $\delta$  ' # VS'
unfolding subst-u unfolding vars-term-ms-subst vars-l by auto
finally have mid-u: mid-set = subst-term-mset u  $\delta$  +  $\delta$  ' # VS' .
hence subst-term-mset u  $\delta \subseteq$  # mid-set by auto
with subset-implies-mult[of subst-term-mset u  $\delta$  mid-set { $\triangleleft$ }]
have two: (subst-term-mset u  $\delta$ , mid-set)  $\in$  (mult { $\triangleleft$ })=
by auto
{
  assume VS'  $\neq$  { # }
  with mid-u have subst-term-mset u  $\delta \subset$  # mid-set unfolding mid-u
  by (simp add: subset-mset.less-le)
  hence (subst-term-mset u  $\delta$ , mid-set)  $\in$  mult { $\triangleleft$ }
  by (rule subset-implies-mult)
} note two' = this

from two one show refl-step: (subst-term-mset u  $\delta$ , subst-term-mset s  $\sigma$ )  $\in$ 
(mult { $\triangleleft$ })=
by (meson transD trans-mult trans-on-converse trans-on-reflcl)
qed
qed
qed

lemma narrowing-right-linear-one-step-simulation: fixes R :: ('f,'v)trs
assumes (s ·  $\sigma$ , t)  $\in$  rstep R
and lins: linear-term s
and right-lin:  $\bigwedge$  lr. lr  $\in$  R  $\implies$  linear-term (snd lr)
and var-cond:  $\bigwedge$  lr. lr  $\in$  R  $\implies$  vars-term (snd lr)  $\subseteq$  vars-term (fst lr)
shows ( $\exists \delta. t = s \cdot \delta \wedge$  (subst-term-mset s  $\delta$ , subst-term-mset s  $\sigma$ )  $\in$  mult1
((rstep R)^-1))
 $\vee$  ( $\exists \mu \delta u. t = u \cdot \delta \wedge$  linear-term u  $\wedge$  (s,u)  $\in$  narrow-step-s R  $\mu \wedge \sigma = \mu \circ_s$ 
 $\delta$ 
 $\wedge$  (subst-term-mset u  $\delta$ , subst-term-mset s  $\sigma$ )  $\in$  (mult { $\triangleleft$ })=)

proof -
from assms(1) obtain l r p  $\mu$  where step: (s ·  $\sigma$ , t)  $\in$  rstep-r-p-s R (l,r) p  $\mu$ 
by (meson rstep-iff-rstep-r-p-s)
from this[unfolded rstep-r-p-s-def' Let-def, simplified]
have lr: (l,r)  $\in$  R by auto
from right-lin[OF this] have lin: linear-term r by auto

```

from *var-cond*[*OF lr*] **have** *vars-term* $r \subseteq \text{vars-term } l$ **by** *auto*
from *narrowing-right-linear-one-step-simulation-r-p-s*[*OF step lins lin this*]
show *?thesis* **unfolding** *narrow-step-s-def* **by** *blast*
qed

lemma *rstep-instance-imp-narrows-r-p-s*: **assumes** *step*: $(s \cdot \sigma, t) \in \text{rstep-r-p-s } R$
 $r \ p \ \tau$
and *not σ* : $p \in \text{poss } s \text{ is-Fun } (s \mid - p)$
shows $\exists \mu \ \mu 2 \ \delta \ u. (s, u) \in \text{narrows-r-p-s } R \ r \ p \ \mu \wedge s \cdot \sigma = s \cdot \mu \cdot \delta \wedge t = u \cdot \delta \wedge \sigma = \mu \circ_s \delta \wedge \tau = \mu 2 \circ_s \delta$
proof –
from *step*[*unfolded rstep-r-p-s-def*] **have** $r: r \in R$ **and** *id*: $s \cdot \sigma \mid - p = \text{fst } r \cdot \tau$
and $t = \text{replace-at } (s \cdot \sigma) \ p \ (\text{snd } r \cdot \tau)$ **by** *auto*
from *not σ* (1) **id** **have** $s \mid - p \cdot \sigma = \text{fst } r \cdot \tau$ **by** *auto*
from *mgu-vd-complete*[*OF this, of ren*] **obtain** $\mu 1 \ \mu 2 \ \delta$ **where**
mgu: *mgu-vd ren* $(s \mid - p) (\text{fst } r)$
 $= \text{Some } (\mu 1, \mu 2)$ **and** $\sigma: \sigma = \mu 1 \circ_s \delta$ **and** $\tau: \tau = \mu 2 \circ_s \delta$ **and** *id*: $s \mid - p \cdot \mu 1$
 $= \text{fst } r \cdot \mu 2$ **by** *auto*
have *narr*: $(s, \text{replace-at } (s \cdot \mu 1) \ p \ (\text{snd } r \cdot \mu 2)) \in \text{narrows-r-p-s } R \ r \ p \ \mu 1$
unfolding *narrows-r-p-s-def* **using** *not σ* *r mgu* **by** *auto*
show *?thesis*
proof (*intro exI conjI, rule narr*)
show $s \cdot \sigma = s \cdot \mu 1 \cdot \delta$ **unfolding** σ **by** *simp*
show $t = \text{replace-at } (s \cdot \mu 1) \ p \ (\text{snd } r \cdot \mu 2) \cdot \delta$
unfolding $t \ \sigma \ \tau$ **using** *not σ* (1)
by (*simp add: ctxt-of-pos-term-subst*)
qed (*rule σ , rule τ*)
qed

lemma *narrows-r-p-s-imp-rstep-r-p-s*: **assumes** *narr*: $(s, t) \in \text{narrows-r-p-s } R \ r \ p$
 μ
shows $\exists \delta. (s \cdot \mu, t) \in \text{rstep-r-p-s } R \ r \ p \ \delta$
proof –
from *narr*[*unfolded narrows-r-p-s-def*] **obtain** $\mu 2$ **where** $p: p \in \text{poss } s$
and $r: r \in R$
and *mgu*: *mgu-vd ren* $(s \mid - p) (\text{fst } r) = \text{Some } (\mu, \mu 2)$
and $t = \text{replace-at } (s \cdot \mu) \ p \ (\text{snd } r \cdot \mu 2)$ **by** *auto*
from p **have** $p\mu: p \in \text{poss } (s \cdot \mu)$ **by** *auto*
from *mgu-vd-sound*[*OF mgu*] **have** *id*: $s \mid - p \cdot \mu = \text{fst } r \cdot \mu 2$ **by** *simp*
show *?thesis*
by (*rule exI[of - $\mu 2$], unfold rstep-r-p-s-def' t, insert r p p μ id, auto*)
qed

lemma *narrows-r-p-s-imp-rstep*: **assumes** *narr*: $(s, t) \in \text{narrows-r-p-s } R \ r \ p \ \mu$
shows $(s \cdot \mu, t) \in \text{rstep } R$
using *narrows-r-p-s-imp-rstep-r-p-s*[*OF narr*]
using *rstep-r-p-s-imp-rstep* **by** *blast*

lemma *narrow-step-s-to-rstep*: $(s, t) \in \text{narrow-step-s } R \ \mu \implies (s \cdot \mu, t) \in \text{rstep } R$

using *narrows-r-p-s-imp-rstep*[*of s t R - - μ*]
unfolding *narrow-step-s-def* **by** *auto*

lemma *rstep-imp-narrows-r-p-s*: **assumes** *step*: $(s, t) \in \text{rstep-r-p-s } R \ r \ p \ \tau$
and *wf*: *wf-trs* *R*
shows $\exists \mu \ \delta \ u. (s, u) \in \text{narrows-r-p-s } R \ r \ p \ \mu \wedge s = s \cdot \mu \cdot \delta \wedge t = u \cdot \delta \wedge u = t \cdot \mu \wedge \mu \circ_s \delta = \text{Var}$
proof –
from *wf*[*unfolded wf-trs-def*] **have** *var-cond*: $\bigwedge l \ r. (l, r) \in R \implies \text{is-Fun } l$ **by** *force*
from *wf*[*unfolded wf-trs-def*] **have** *var-cond'*: $\bigwedge l \ r. (l, r) \in R \implies \text{vars-term } r \subseteq \text{vars-term } l$ **by** *force*

from *step*[*unfolded rstep-r-p-s-def*] **have** *p*: $p \in \text{poss } s$ **and** *r*: $r \in R$ **and** *id*: $s \mid\!-\ p \cdot \text{Var} = \text{fst } r \cdot \tau$
and *t*: $t = \text{replace-at } s \ p \ (\text{snd } r \cdot \tau)$
by *auto*
obtain *l1 r1* **where** *lr1*: $r = (l1, r1)$ **by** *force*
from *var-cond*[*OF r*[*unfolded lr1*]] **have** *is-Fun*: *is-Fun* $(s \mid\!-\ p)$ **using** *id lr1* **by** (*cases l1, auto*)
from *mgu-vd-complete*[*OF id, of ren*] **obtain** $\mu1 \ \mu2 \ \delta$ **where**
mgu: *mgu-vd* *ren* $(s \mid\!-\ p) (\text{fst } r)$
 $= \text{Some } (\mu1, \mu2)$ **and** *Var*: $\text{Var} = \mu1 \circ_s \delta$ **and** $\tau: \tau = \mu2 \circ_s \delta$ **and** *id*: $s \mid\!-\ p \cdot \mu1 = \text{fst } r \cdot \mu2$ **by** *auto*
define *u* **where** $u = \text{replace-at } (s \cdot \mu1) \ p \ (\text{snd } r \cdot \mu2)$

have *narr*: $(s, u) \in \text{narrows-r-p-s } R \ r \ p \ \mu1$
unfolding *narrows-r-p-s-def u-def* **using** *is-Fun p r mgu* **by** *auto*
from *narrows-r-p-s-imp-rstep*[*OF this*] **have** $(s \cdot \mu1, u) \in \text{rstep } R$.

with *var-cond'* **have** *vsu*: *vars-term* $(s \cdot \mu1) \supseteq \text{vars-term } u$
unfolding *rstep.simps* **by** (*force simp: vars-term-ctxt-apply vars-term-subst*)

have $\exists y. \mu1 \ x = \text{Var } y$ **for** *x* **using** *arg-cong*[*OF Var, of $\lambda \sigma. \sigma \ x$, unfolded subst-compose-def*]
by (*cases $\mu1 \ x$, auto*)
then obtain $\mu1'$ **where** $\mu1: \mu1 = \text{Var } o \ \mu1'$ **unfolding** *o-def* **by** *metis*
have $\delta \mu1: \delta (\mu1' \ x) = \text{Var } x$ **for** *x* **using** *arg-cong*[*OF Var*[*unfolded $\mu1$ subst-compose-def o-def*], *of $\lambda \sigma. \sigma \ x$*]
by *simp*
have $u \cdot (\delta \circ_s \mu1) = u \cdot \text{Var}$
proof (*rule term-subst-eq*)
fix *x*
assume $x \in \text{vars-term } u$
with *vsu* **obtain** *y* **where** $x \in \text{vars-term } (\mu1 \ y)$
by (*auto simp: vars-term-subst*)
from *this*[*unfolded $\mu1$*]
have $\mu1': \mu1' \ y = x$ **by** *simp*

```

    from  $\delta mu$ [of  $y$ , unfolded this]
    have  $\delta x = Var\ y$  by auto
    from  $arg-cong$ [OF this, of  $\lambda t. t \cdot \mu 1$ ]
    show  $(\delta \circ_s \mu 1)\ x = Var\ x$  unfolding  $subst-compose-def$ 
      unfolding  $\mu 1\ o-def$  using  $\mu 1'$  by simp
qed
hence  $uu: u \cdot \delta \cdot \mu 1 = u$  by auto

have  $s': s = s \cdot \mu 1 \cdot \delta$  using  $Var$  by (simp add: eval-subst)
have  $tu: t = u \cdot \delta$ 
  unfolding  $t\ \tau\ u-def$  using  $p\ Var$ 
  by (simp add: ctxt-of-pos-term-subst)
have  $ut: u = t \cdot \mu 1$ 
  by (subst  $uu[symmetric]$ , unfold  $tu$ , auto)
show ?thesis
  by (intro  $exI\ conjI$ , rule  $narr$ , rule  $s'$ , auto intro:  $tu\ ut\ simp: Var$ )
qed

lemma  $rstep-imp-narrows-s$ : assumes  $step: (s,t) \in rstep\ R$ 
  and  $wf: wf-trs\ R$ 
shows  $\exists\ \mu\ \delta\ u. (s,u) \in narrow-step-s\ R\ \mu \wedge s = s \cdot \mu \cdot \delta \wedge t = u \cdot \delta \wedge u = t \cdot \mu$ 
 $\wedge\ \mu \circ_s \delta = Var$ 
proof -
  from  $step$  obtain  $r\ p\ \tau$  where  $(s,t) \in rstep-r-p-s\ R\ r\ p\ \tau$ 
  using  $rstep-iff-rstep-r-p-s$  by blast
  note  $main = rstep-imp-narrows-r-p-s$ [OF this]
  from  $main$ [OF  $wf$ ] obtain  $\mu\ \delta\ u$ 
  where  $*$ :  $(s, u) \in narrows-r-p-s\ R\ r\ p\ \mu\ s = s \cdot \mu \cdot \delta\ t = u \cdot \delta\ u = t \cdot \mu\ \mu \circ_s$ 
 $\delta = Var$ 
  by fast
  thus ?thesis unfolding  $narrow-step-s-def$  by fast
qed

lemma  $rstep-imp-narrows$ : assumes  $step: (s,t) \in rstep\ R$ 
  and  $wf: wf-trs\ R$ 
shows  $\exists\ \mu\ \delta\ u. (s,u) \in narrow-step\ R \wedge t = u \cdot \delta \wedge u = t \cdot \mu$ 
  using  $rstep-imp-narrows-s$ [OF  $assms$ ] unfolding  $narrow-step-to-s$  by fast

lemma  $narrow-instance-pos-in-subst$ : assumes
   $narr: (t \cdot \gamma, s) \in narrows-r-p-s\ (R :: ('f,'v)trs)\ (l,r)\ p\ \mu$ 
  and  $pos: p \notin fun-poss\ t$ 
  and  $vars: vars-term\ r \subseteq vars-term\ l$ 
  and  $lin: linear-term\ (t :: ('f,'v)term)\ linear-term\ r$ 
shows  $\exists\ \gamma. t \cdot \gamma = s$ 
proof -
  from  $narrows-r-p-s-imp-rstep-r-p-s$ [OF  $narr$ ]
  obtain  $\delta$  where  $(t \cdot (\gamma \circ_s \mu), s) \in rstep-r-p-s\ R\ (l,r)\ p\ \delta$  by auto
  from  $narrowing-right-linear-one-step-simulation-r-p-s$ [OF this  $lin\ vars$ ]  $pos$ 
  show ?thesis by auto

```

qed

lemma *narrow-instance-pos-in-term*: **fixes** $t :: (f, v)term$
assumes $narr: (t, s) \in narrows-r-p-s\ R\ (l, r)\ p\ \mu1$
and $narr-inst: (t \cdot \gamma, u) \in narrows-r-p-s\ R\ (l, r)\ p\ \mu2$
shows $\exists \delta. s \cdot \delta = u \wedge \gamma \circ_s \mu2 = \mu1 \circ_s \delta$
proof –
define C **where** $C = ctxt-of-pos-term\ p\ t$
define tp **where** $tp = t \mid -\ p$
from $narr[unfolded\ narrows-r-p-s-def\ fst-conv\ snd-conv]$ **obtain** $\mu1'$ **where**
 $p: p \in poss\ t$ **and** $mgu: mgu-vd\ ren\ tp\ l = Some\ (\mu1, \mu1')$
and $s: s = (ctxt-of-pos-term\ p\ (t \cdot \mu1)) \langle r \cdot \mu1' \rangle$ **using** $tp-def$ **by** *blast*
have $ctxt: ctxt-of-pos-term\ p\ (t \cdot \mu1) = C \cdot_c \mu1$ **for** $\mu1 :: (f, v)subst$
using p **unfolding** $C-def$ **by** $(rule\ ctxt-of-pos-term-subst)$
from s **this**
have $s: s = (C \cdot_c \mu1) \langle r \cdot \mu1' \rangle$ **by** *auto*
have $t: t = C \langle tp \rangle$ **using** p **unfolding** $C-def\ tp-def$
by $(simp\ add: ctxt-supt-id)$
from $mgu-vd-sound[OF\ mgu]$ **have** $\mu1: tp \cdot \mu1 = l \cdot \mu1'$ **by** *auto*
from p **have** $t \cdot \gamma \mid -\ p = tp \cdot \gamma$ **unfolding** $tp-def$ **by** *auto*

with $narr-inst[unfolded\ narrows-r-p-s-def\ fst-conv\ snd-conv, simplified]$
obtain $\mu2'$ **where** $mgu2: mgu-vd\ ren\ (tp \cdot \gamma)\ l = Some\ (\mu2, \mu2')$
and $u: u = (ctxt-of-pos-term\ p\ (t \cdot (\gamma \circ_s \mu2))) \langle r \cdot \mu2' \rangle$ **by** *auto*
from $u[unfolded\ ctxt]$
have $u: u = (C \cdot_c (\gamma \circ_s \mu2)) \langle r \cdot \mu2' \rangle$ **by** *simp*

from $mgu-vd-sound[OF\ mgu2]$ **have** $tp \cdot (\gamma \circ_s \mu2) = l \cdot \mu2'$ **by** *auto*
from $mgu-vd-complete[OF\ this, of\ ren, unfolded\ mgu]$
obtain δ **where** $delt: \gamma \circ_s \mu2 = \mu1 \circ_s \delta \wedge \mu2' = \mu1' \circ_s \delta$ **by** *auto*
auto
show *?thesis* **unfolding** $s\ u$ **using** $delt$
by $(intro\ exI[of\ -\ \delta], auto)$
qed

lemma *SN-on-narrows-imp-SN-on-rstep*: **fixes** $R :: (f, v)trs$
assumes $SN-on\ (narrow-step\ R)\ \{s\}$
and $wf: wf-trs\ R$
shows $SN-on\ (rstep\ R)\ \{s\}$
proof
fix t
assume $start: t \in \{s\}$ **and** $steps: \forall i. (t\ i, t\ (Suc\ i)) \in rstep\ R$
hence $t0s: t\ 0 \cdot Var = s$ **by** *auto*

define $step-cond$ **where** $step-cond = (\lambda\ (i :: nat)\ ti\ (\sigma i :: (f, v)subst)\ tsi\ (\sigma si :: (f, v)subst). (ti, tsi) \in narrow-step\ R)$
note $simu = dependent-nat-choice2-start[of\ \lambda\ i\ u\ \mu. t\ i \cdot \mu = u\ s\ Var\ step-cond, OF\ t0s]$

```

{
  fix v and  $\mu :: ('f, 'v)subst$  and n
  assume v:  $t\ n \cdot \mu = v$ 
  from steps have  $(t\ n, t\ (Suc\ n)) \in rstep\ R$  by auto
  hence  $(v, t\ (Suc\ n) \cdot \mu) \in rstep\ R$  unfolding v[symmetric] ..
  from rstep-imp-narrows[OF this wf]
  obtain  $\mu' u$  where  $(v, u) \in narrow-step\ R$   $u = t\ (Suc\ n) \cdot \mu \cdot \mu'$ 
    by blast
  hence  $\exists x' y'. t\ (Suc\ n) \cdot y' = x' \wedge step-cond\ n\ v\ \mu\ x'\ y'$ 
    unfolding step-cond-def
    by (intro exI[of - u] exI[of -  $\mu \circ_s \mu'$ ], auto)
}
from simu[OF this, unfolded step-cond-def] obtain u where
  u 0 = s
  (u i, u (Suc i))  $\in narrow-step\ R$  for i
  by blast

thus False using  $\langle SN-on\ (narrow-step\ R)\ \{s\} \rangle$  by blast
qed

lemma exists-narrow-steps-to-infinite-rsteps: fixes  $R :: ('f, 'v)trs$ 
  defines  $Q \equiv \lambda (n :: nat)\ s\ (\sigma :: ('f, 'v)subst)\ u\ (\delta :: ('f, 'v)subst).$ 
     $(\exists\ \mu. (s, u) \in narrow-step-s\ R\ \mu \wedge \sigma = \mu \circ_s \delta)$ 
  assumes wf: wf-trs R
    and single-steps:  $\bigwedge n\ s\ \sigma. P\ n\ s\ \sigma \implies \exists u\ \delta. P\ (Suc\ n)\ u\ \delta \wedge Q\ n\ s\ \sigma\ u\ \delta$ 
    and P0:  $P\ 0\ r\ \sigma$ 
  shows  $\neg SN-on\ (narrow-step\ R)\ \{r\}$ 
     $\neg SN-on\ (rstep\ R)\ ((narrow-step\ R)^* \text{ `` }\{r\}\text{ ''})$ 
  proof -
    from dependent-nat-choice2-start[of P r  $\sigma$  Q, OF P0 single-steps]
    obtain s  $\delta$  where 0:  $s\ 0 = r\ \delta\ 0 = \sigma$ 
      and *:  $P\ n\ (s\ n)\ (\delta\ n) \forall n. Q\ n\ (s\ n)\ (\delta\ n)\ (s\ (Suc\ n))\ (\delta\ (Suc\ n))$  for n by
    blast
    from choice[OF *(2)[unfolded Q-def]] obtain  $\mu$ 
      where steps:  $\bigwedge i. (s\ i, s\ (Suc\ i)) \in narrow-step-s\ R\ (\mu\ i)$ 
      and  $\delta$ :  $\bigwedge i. \delta\ i = \mu\ i \circ_s \delta\ (Suc\ i)$ 
      by auto
    define  $\gamma$  where  $\gamma = rec-nat\ Var\ (\lambda i\ \gamma. \gamma \circ_s \mu\ i)$ 
    have  $\gamma$ :  $\gamma\ 0 = Var\ \bigwedge i. \gamma\ (Suc\ i) = \gamma\ i \circ_s \mu\ i$  unfolding  $\gamma$ -def by auto
    define t where  $t = r \cdot \sigma$ 

    have  $\sigma$ :  $\sigma = \gamma\ i \circ_s \delta\ i$  for i
    proof (induct i)
      case (Suc i)
      thus ?case by (simp add:  $\delta$ [of i]  $\gamma$  subst-compose-assoc)
    qed (auto simp: 0  $\gamma$ )
    have rsteps:  $(s\ i \cdot \mu\ i, s\ (Suc\ i)) \in rstep\ R$  for i using narrow-step-s-to-rstep[OF
  steps[of i]] .
  {

```

```

fix  $i$ 
from  $rstep\text{-}vars\text{-}term[OF - rsteps]$   $wf$ 
have  $vars\text{-}term (s\ i \cdot \mu\ i) \supseteq vars\text{-}term (s\ (Suc\ i))$ 
  by ( $auto\ simp: wf\text{-}trs\text{-}def$ )
} note  $vars\text{-}si = this$ 

have  $vars: vars\text{-}term (s\ i) \subseteq vars\text{-}term (r \cdot \gamma\ i)$  for  $i$ 
proof ( $induct\ i$ )
  case ( $Suc\ i$ )
  with  $vars\text{-}si[of\ i]$  show  $?case$  by ( $force\ simp: vars\text{-}term\text{-}subst\ \gamma$ )
qed ( $auto\ simp: 0\ \gamma$ )

have  $ts\gamma: t = r \cdot \gamma\ i \cdot \delta\ i$  for  $i$  unfolding  $t\text{-}def\ \sigma[of\ i]\ 0$  by  $simp$ 

from  $steps$  have  $nsteps: (s\ i, s\ (Suc\ i)) \in narrow\text{-}step\ R$  for  $i$ 
  unfolding  $narrow\text{-}step\text{-}to\text{-}s$  by  $auto$ 
with  $0(1)$  show  $\neg SN\text{-}on\ (narrow\text{-}step\ R)\ \{r\}$  by  $force$ 

define  $meas\text{-}fun$  where  $meas\text{-}fun\ i = size\ (funs\text{-}term\text{-}ms\ (r \cdot \gamma\ i))$  for  $i$ 
define  $dmeas\text{-}fun$  where  $dmeas\text{-}fun\ i = size\ (funs\text{-}term\text{-}ms\ t) - meas\text{-}fun\ i$  for
 $i$ 

{
  fix  $i$ 
  define  $rg$  where  $rg = r \cdot \gamma\ i$ 
  have  $sub: vars\text{-}term (s\ i) \subseteq vars\text{-}term\ rg$  using  $vars[of\ i]$  unfolding  $rg\text{-}def$  by
 $auto$ 

  let  $?diff = \sum \# ((funs\text{-}term\text{-}ms \circ \mu)\ i\ '\# vars\text{-}term\text{-}ms\ rg)$ 
  have  $bnd: meas\text{-}fun\ (Suc\ i) \leq size\ (funs\text{-}term\text{-}ms\ t)$ 
    unfolding  $ts\gamma[of\ Suc\ i]$   $meas\text{-}fun\text{-}def\ funs\text{-}term\text{-}ms\text{-}subst\text{-}apply$  by  $simp$ 

  have  $funs\text{-}term\text{-}ms\ (r \cdot \gamma\ (Suc\ i)) = funs\text{-}term\text{-}ms\ (rg \cdot \mu\ i)$  unfolding  $\gamma\ rg\text{-}def$ 
by  $simp$ 
  also have  $\dots = funs\text{-}term\text{-}ms\ rg + sum\text{-}mset\ ((funs\text{-}term\text{-}ms\ o\ \mu)\ i\ '\#$ 
 $vars\text{-}term\text{-}ms\ rg)$ 
    unfolding  $funs\text{-}term\text{-}ms\text{-}subst\text{-}apply\ o\text{-}def$  by  $simp$ 
  finally have  $id: funs\text{-}term\text{-}ms\ (r \cdot \gamma\ (Suc\ i)) = funs\text{-}term\text{-}ms\ rg + ?diff$ 
    by  $auto$ 
  from  $arg\text{-}cong[OF\ this, of\ size]$ 
  have  $meas: meas\text{-}fun\ (Suc\ i) = meas\text{-}fun\ i + size\ ?diff$ 
    by ( $auto\ simp: meas\text{-}fun\text{-}def\ rg\text{-}def$ )
  hence  $meas\text{-}fun\ (Suc\ i) \geq meas\text{-}fun\ i$  by  $auto$ 
  with  $bnd$  have  $le: dmeas\text{-}fun\ (Suc\ i) \leq dmeas\text{-}fun\ i$  unfolding  $dmeas\text{-}fun\text{-}def$ 

```

```

by auto
{
  assume  $\exists x \in \text{vars-term } (s \ i). \text{ is-Fun } (\mu \ i \ x)$ 
  with sub obtain  $x$  where  $x: x \in \# \text{ vars-term-ms } rg$  and  $f: \text{ is-Fun } (\mu \ i \ x)$  by
auto
  then obtain  $xs$  where  $\text{vars}: \text{vars-term-ms } rg = \text{add-mset } x \ xs$  by (metis
mset-add)
  with  $f$  have  $\text{size } ?diff \neq 0$  by (cases  $\mu \ i \ x$ , auto)
  hence  $\text{meas-fun } (Suc \ i) > \text{meas-fun } i$  unfolding  $\text{meas}$  by linarith
  with  $bnd$  have  $\text{dmeas-fun } (Suc \ i) < \text{dmeas-fun } i$  unfolding  $\text{dmeas-fun-def}$ 
by auto
}
note  $le \ this$ 
} note  $\text{dmeas} = this$ 

have  $\exists j. \forall i \geq j. (\text{dmeas-fun } i, \text{dmeas-fun } (Suc \ i)) \in \{(x, y). y \leq x\} - \{(x, y). y < x\}$ 
by (rule non-strict-ending[of  $\text{dmeas-fun}$ ], insert  $\text{dmeas SN-nat-gt}$ , auto simp:
SN-defs)
then obtain  $j$  where  $\text{dmeas-eq}: \bigwedge i. i \geq j \implies \text{dmeas-fun } (Suc \ i) = \text{dmeas-fun } i$  by fastforce

have  $\mu Var: \mu \ i \ ' \text{vars-term } (s \ i) \subseteq \text{range } Var$  if  $i \geq j$  for  $i$ 
using  $\text{dmeas-eq}[OF \ that] \ \text{dmeas}(2)[of \ i]$  by force

define  $vs$  where  $vs \ i = \text{vars-term } (s \ (i + j) \cdot \mu \ (i + j))$  for  $i$ 
have  $\text{fin}[simp,intro]: \text{finite } (vs \ i)$  for  $i$  unfolding  $vs\text{-def}$  by auto
define  $\text{meas-var}$  where  $\text{meas-var } i = \text{card } (vs \ i)$  for  $i$ 
{
  fix  $i$ 
  assume  $i \geq j$ 
  from  $\mu Var[OF \ this]$  have  $\forall x. \exists y. x \in \text{vars-term } (s \ i) \longrightarrow \mu \ i \ x = Var \ y$ 
by auto
  from  $\text{choice}[OF \ this]$  obtain  $\mu i$  where  $\bigwedge x. x \in \text{vars-term } (s \ i) \implies \mu \ i \ x = Var \ (\mu i \ x)$ 
by auto
  hence  $s \ i \cdot \mu \ i = s \ i \cdot (Var \ o \ \mu i)$ 
by (intro term-subst-eq, auto)
  hence  $\exists \mu i. s \ i \cdot \mu \ i = s \ i \cdot (Var \ o \ \mu i)$  by auto
}
hence  $\forall i. \exists \mu'. i \geq j \longrightarrow s \ i \cdot \mu \ i = s \ i \cdot (Var \ o \ \mu')$  by blast
from  $\text{choice}[OF \ this]$  obtain  $\mu'$  where
 $\mu': \bigwedge i. i \geq j \implies s \ i \cdot \mu \ i = s \ i \cdot (Var \ o \ (\mu' \ i))$  by blast
{
  fix  $i$ 
  have  $vs \ (Suc \ i) = \mu' \ (Suc \ (i + j)) \ ' \text{vars-term } (s \ (Suc \ (i + j)))$ 

```

```

    using  $\mu'$ [of Suc (i + j)] vs-def[of Suc i]
    by (auto simp: vars-term-subst)
  also have  $\dots \subseteq \mu' (Suc (i + j))$  ‘vs i’
    using vars-si unfolding vs-def by auto
  ultimately have sub:  $vs (Suc i) \subseteq \mu' (Suc i + j)$  ‘vs i’ by auto
  from card-mono[OF finite-imageI[OF fin] this]
  have meas-var (Suc i)  $\leq$  meas-var i unfolding meas-var-def using card-image-le[OF
fin]
    by (metis le-trans)
  note sub this
} note meas-var = this

have  $\exists j. \forall i \geq j. (meas-var\ i, meas-var\ (Suc\ i)) \in \{(x, y). y \leq x\} - \{(x, y). y < x\}$ 
  by (rule non-strict-ending[of meas-var], insert meas-var SN-nat-gt, auto simp:
SN-defs)
then obtain k where meas-var-eq:  $\bigwedge i. i \geq k \implies meas-var\ (Suc\ i) = meas-var\ i$  by fastforce
define l where l = Suc k + j

{
  fix ii
  assume ii: ii  $\geq$  l
  define i where i = ii - Suc 0
  define i' where i' = i - j
  have i  $\geq$  k + j and ii: ii = Suc i and iij: ii  $\geq$  j using ii unfolding l-def
i-def by auto
  hence i: i = i' + j Suc i' + j = Suc i i'  $\geq$  k unfolding i'-def by auto
  from meas-var-eq[OF i(3), unfolded meas-var-def]
  have card: card (vs (Suc i')) = card (vs i') by auto
  from meas-var(1)[of i', unfolded i(2)]
  have vs (Suc i')  $\subseteq \mu' (Suc\ i)$  ‘vs i'’ by auto
  with card have inj-on ( $\mu' (Suc\ i)$ ) (vs i')
    by (metis card-image-le card-seteq eq-card-imp-inj-on fin finite-imageI)
  from this[unfolded vs-def, folded i(1)]
  have inj-on ( $\mu' (Suc\ i)$ ) (vars-term (s i ·  $\mu$  i)) by auto
  with vars-si[of i]
  have inj-on ( $\mu' ii$ ) (vars-term (s ii)) unfolding ii by (metis inj-on-subset)
} note inj = this

define sl where sl i = s (i + l) for i
{
  fix i
  assume i: i  $\geq$  l
  hence ij: i  $\geq$  j unfolding l-def by auto
  let ?V = vars-term (s i)
  from inj[OF i]
  have inj-on ( $\mu' i$ ) ?V .

```

```

from the-inv-into-f-f[OF this]
obtain inv where inv:  $\bigwedge x. x \in ?V \implies inv (\mu' i x) = x$  by blast
have  $s i \cdot \mu i \cdot (Var o inv) = s i \cdot ((Var o \mu' i) \circ_s (Var o inv))$ 
  using  $\mu'[OF ij]$  by simp
also have  $\dots = s i \cdot Var$ 
  by (rule term-subst-eq, insert inv, auto simp: o-def subst-compose-def)
finally have  $s i \cdot \mu i \cdot (Var o inv) = s i$  by simp
from rstep-subst[OF rsteps[of i], of (Var o inv), unfolded this]
have  $\exists \gamma. (s i, s (Suc i) \cdot \gamma) \in rstep R$  by auto
}
hence  $\forall i. \exists \gamma. (sl i, sl (Suc i) \cdot \gamma) \in rstep R$  unfolding sl-def by simp
from choice[OF this] obtain  $\gamma$  where rsteps:  $(sl i, sl (Suc i) \cdot \gamma) \in rstep R$ 
for i
  by auto

define  $\gamma'$  where  $\gamma' = rec-nat Var (\lambda i gam. \gamma i \circ_s gam)$ 
define t where  $t i = sl i \cdot \gamma' i$  for i
have rsteps:  $(t i, t (Suc i)) \in rstep R$  for i
  unfolding t-def  $\gamma'$ -def using rstep-subst[OF rsteps[of i]] by simp
have t0:  $t 0 = s l$  unfolding t-def sl-def  $\gamma'$ -def by simp
from rsteps t0 have nSN:  $\neg SN-on (rstep R) \{s l\}$  by auto

have  $s l \in (narrow-step R)^* \text{ `` } \{r\}$ 
proof (induct l)
  case *: 0
    with 0 show ?case by auto
  next
    case (Suc i)
    with nsteps[of i] show ?case by (metis rtrancl-image-advance)
  qed
with nSN show  $\neg SN-on (rstep R) ((narrow-step R)^* \text{ `` } \{r\})$ 
  by (metis SN-onE rsteps t0)
qed

end
end

```

5 Dershowitz' Right-Forward Closures for Proving Termination

Right-Forward Closures (RFC) are defined, and it is shown that termination on all terms is equivalent to termination starting on RFC for TRSs that are right-linear, or for overlay TRSs that are locally confluent.

```

theory Right-Forward-Closure
imports
  Linear-Narrowing
  First-Order-Rewriting.Critical-Pairs

```

```

    Gramlich-Innermost-Switch
begin

context
  fixes ren :: 'v :: infinite renaming2
begin

definition right-forw-closure :: ('f,'v)trs  $\Rightarrow$  ('f,'v)term set where
  right-forw-closure R = (narrow-step ren R)* “ rhss R

lemma right-forw-closure-SN-main: fixes R :: ('f,'v)trs
  assumes right-lin:  $\bigwedge$  lr. lr  $\in$  R  $\Longrightarrow$  linear-term (snd lr)
  and wf: wf-trs R
  and nSN:  $\neg$  SN (rstep R)
  shows  $\neg$  SN-on (rstep R) (right-forw-closure R)
proof -
  from wf have var-cond:  $\bigwedge$  lr. lr  $\in$  R  $\Longrightarrow$  vars-term (snd lr)  $\subseteq$  vars-term (fst
  lr)
  by (auto simp: wf-trs-def)
  from wf have var-cond2:  $\bigwedge$  lr. lr  $\in$  R  $\Longrightarrow$  is-Fun (fst lr)
  by (force simp: wf-trs-def)
  note simu = narrowing-right-linear-one-step-simulation[of - - - R ren, OF - -
  right-lin var-cond]
  let ?R = rstep R
  let ?N = narrow-step ren R
  let ?NS = narrow-step-s ren R
  let ?RFC = right-forw-closure R
  have R-to-RFC: lr  $\in$  R  $\Longrightarrow$  snd lr  $\in$  ?RFC for lr unfolding right-forw-closure-def
  by auto
  from nSN obtain t where  $\neg$  SN-on (rstep R) {t} unfolding SN-defs by auto
  from not-SN-imp-subst-Tinf[OF this] obtain t where t  $\in$  Tinf (rstep R) by
  auto
  from Tinf-rstep-imp-first-root-step[OF this]
  obtain tm1 t0 where Tinf: tm1  $\in$  Tinf ?R and (tm1,t0)  $\in$  rstep R and nt0:
   $\neg$  SN-on ?R {t0} by auto
  from this(2) obtain l r0  $\sigma$ 0 where lr: (l,r0)  $\in$  R and tm1: tm1 = l  $\cdot$   $\sigma$ 0 and
  t0: t0 = r0  $\cdot$   $\sigma$ 0
  by (rule rstepE)
  have r0-rfc: r0  $\in$  ?RFC using R-to-RFC[OF lr] by simp
  have linr: linear-term r0 using right-lin[OF lr] by simp
  define SNT where SNT = {t. SN-on ?R {t}}
  define stm :: ('f,'v)term  $\Rightarrow$  - where stm = subst-term-mset
  {
    fix x
    assume x  $\in$  vars-term r0
    with var-cond[OF lr] have x: x  $\in$  vars-term l by auto
    from var-cond2[OF lr] obtain f ls where l: l = Fun f ls by auto
    with x obtain li where li: li  $\in$  set ls and x  $\in$  vars-term li by auto

```

```

hence subt:  $li \cdot \sigma 0 \supseteq \sigma 0 x$ 
  by (metis eval-term.simps(1) supseq-subst vars-term-supseq)
hence tm1  $\triangleright \sigma 0 x$  unfolding tm1 using li
  by (metis l subst-image-subterm x)
with Tinf[unfolded Tinf-def] have SN-on  $?R \{\sigma 0 x\}$  by auto
hence  $\sigma 0 x \in SNT$  unfolding SNT-def by auto
}
hence r0-SNT:  $set-mset (stm r0 \sigma 0) \subseteq SNT$ 
  unfolding subst-term-mset-def stm-def by auto
define Rel where Rel = restrict-SN-supt  $?R$ 
define mRel where mRel = mult ( $Rel^{\wedge-1}$ )
have SN-Rel: SN Rel unfolding Rel-def
  by (rule SN-restrict-SN-supt-rstep)
hence wf ( $Rel^{\wedge-1}$ ) by (rule SN-imp-wf)
hence wf-mRel: wf mRel unfolding mRel-def by (rule wf-mult)
define RR where RR =  $?R$ 
have R-SNT:  $(u, v) \in ?R \implies u \in SNT \implies v \in SNT$  for u v unfolding SNT-def
  by (simp add: step-preserves-SN-on)

have supt-SNT:  $u \triangleright v \implies u \in SNT \implies v \in SNT$  for u v unfolding SNT-def
  by fast

{
  fix s  $\sigma$   $\delta$ 
  assume SNT:  $set-mset (stm s \sigma) \subseteq SNT$  and decr:  $(stm s \delta, stm s \sigma) \in mult1$ 
  ( $?R^{-1}$ )
  {
    fix u
    assume  $u \in \# stm s \delta$ 
    with decr obtain v where  $v \in \# stm s \sigma$  and disj:  $u = v \vee (v, u) \in ?R$ 
    unfolding mult1-def RR-def[symmetric] by force
    from SNT v have  $v \in SNT$  by auto
    with R-SNT[OF - this] disj have  $u \in SNT$  by auto
  }
  hence SNT':  $set-mset (stm s \delta) \subseteq SNT$  by auto
  from decr[unfolded mult1-def, simplified]
  obtain t M K where split:  $stm s \sigma = add-mset t M stm s \delta = M + K$ 
  and steps:  $\bigwedge u. u \in \# K \implies (t, u) \in ?R$  by auto
  {
    fix u
    assume  $u \in \# K$ 
    with split SNT SNT' have SNT:  $t \in SNT$   $u \in SNT$  by auto
    from steps[OF u] have  $(t, u) \in ?R$  by auto
    with SNT have  $(u, t) \in Rel^{\wedge-1}$  unfolding Rel-def restrict-SN-supt-def
    restrict-SN-def SNT-def by auto
  }
  hence  $(stm s \delta, stm s \sigma) \in mult1 (Rel^{\wedge-1})$  unfolding mult1-def split by auto
  hence  $(stm s \delta, stm s \sigma) \in mRel$  unfolding mRel-def by (simp add: mult-def)
  note SNT' this

```

```

} note rstep-into-mRel = this

have tr: trans {<}
  by (simp add: trans-supt)

{
  fix s t σ δ
  assume SNT: set-mset (stm t σ) ⊆ SNT and decr: (stm s δ, stm t σ) ∈ mult
  {<}
  from mult-implies-one-step[OF tr decr] obtain I J K
    where split: stm t σ = I + J stm s δ = I + K and J: J ≠ {#} and step:
    ∧ k. k ∈ #K ⇒ ∃ j ∈ #J. k < j by auto
    {
      fix u
      assume u ∈ # stm s δ
      then obtain v where v ∈ # stm t σ u = v ∨ v ▷ u unfolding split using
      step by auto
      with supt-SNT[of v u] SNT have u ∈ SNT by auto
    }
    hence SNT': set-mset (stm s δ) ⊆ SNT by auto
    {
      fix k
      assume k: k ∈ # K
      from step[OF this] obtain j where j: j ∈ # J j ▷ k by auto
      from SNT SNT' k j have k ∈ SNT j ∈ SNT by (auto simp: split)
      with j(2) have (j,k) ∈ Rel unfolding Rel-def restrict-SN-supt-def re-
strict-SN-def SNT-def by auto
      hence ∃ j ∈ # J. (k,j) ∈ Rel-1 using j by auto
    }
    hence (stm s δ, stm t σ) ∈ mRel
    unfolding split mRel-def using one-step-implies-mult[OF J, of K Rel-1 I]
    by auto
    note SNT' this
  } note supt-into-mRel = this

define P where P s σ = (linear-term s ∧ set-mset (stm s σ) ⊆ SNT ∧ s ∈
?RFC) for s σ

define narr-step where narr-step s σ u μ δ = ((s, u) ∈ ?NS μ ∧ σ = μ ∘s δ)
  for s u :: (f, v)term and μ σ δ :: (f, v)subst

define step-cond where step-cond s σ t u δ = (t = u · δ ∧
  P u δ ∧ ((stm u δ, stm s σ) ∈ mRel ∨ stm u δ = stm s σ ∧ (∃ μ. narr-step s
σ u μ δ)))
  for s σ t δ u

{
  fix s :: (f, v)term and σ t
  assume P s σ and (s · σ, t) ∈ ?R

```

hence *lin*: *linear-term* $s (s \cdot \sigma, t) \in ?R$ and *SNT*: *set-mset* $(stm\ s\ \sigma) \subseteq SNT$
 and *RFC*: $s \in ?RFC$
 by (*auto simp*: *P-def*)
 from *simu*[*OF this*(2,1)] obtain $\mu\ \delta\ u$ where
 $(t = s \cdot \delta \wedge (stm\ s\ \delta, stm\ s\ \sigma) \in mult1\ ((rstep\ R)^{-1})) \vee$
 $(t = u \cdot \delta \wedge linear-term\ u\ (s, u) \in ?NS\ \mu \wedge$
 $(stm\ u\ \delta, stm\ s\ \sigma) \in (mult\ \{\triangleleft\})^= \wedge$
 $((stm\ u\ \delta, stm\ s\ \sigma) \in mult\ \{\triangleleft\} \vee narr-step\ s\ \sigma\ u\ \mu\ \delta))$ (**is** $?A \vee ?B$)
 by (*auto simp*: *stm-def narr-step-def*)
 hence $\exists\ u. t = u \cdot \delta \wedge P\ u\ \delta \wedge$
 $((stm\ u\ \delta, stm\ s\ \sigma) \in mRel \vee stm\ u\ \delta = stm\ s\ \sigma \wedge narr-step\ s\ \sigma\ u\ \mu$
 $\delta)$
 proof
 assume $?A$
 with *rstep-into-mRel*[*OF SNT*, *of* δ]
 have $t = s \cdot \delta \wedge (stm\ s\ \delta, stm\ s\ \sigma) \in mRel \wedge set-mset\ (stm\ s\ \delta) \subseteq SNT$ by
blast
 with $\langle ?A \rangle$ *RFC lin* show *?thesis* unfolding *P-def*
 by (*intro exI*[*of* - *s*], *auto*)
 next
 assume $?B$
 hence $B: t = u \cdot \delta\ linear-term\ u\ (s, u) \in ?NS\ \mu\ (stm\ u\ \delta, stm\ s\ \sigma) \in (mult\ \{\triangleleft\})^=$
 and *disj*: $(stm\ u\ \delta, stm\ s\ \sigma) \in mult\ \{\triangleleft\} \vee narr-step\ s\ \sigma\ u\ \mu\ \delta$ by *auto*
 note *supt* = *supt-into-mRel*[*OF SNT*, *of* $u\ \delta$]
 from $B(3)$ have $(s, u) \in ?N$ unfolding *narrow-step-def narrow-step-s-def* by
fastforce
 with *RFC* have *RFC*: $u \in ?RFC$ unfolding *right-forw-closure-def* by (*metis*
rtrancl-image-advance)
 from $B(4)$ *supt SNT* have *SNT*: *set-mset* $(stm\ u\ \delta) \subseteq SNT$ by *auto*
 show *?thesis*
 proof (*cases* $(stm\ u\ \delta, stm\ s\ \sigma) \in mult\ \{\triangleleft\}$)
 case *True*
 from $B\ True\ supt\ lin\ RFC$ show *?thesis* by (*auto simp*: *P-def*)
 next
 case *False*
 with $B(4)$ have *eq*: $stm\ u\ \delta = stm\ s\ \sigma$ by *auto*
 from *disj False eq B lin RFC SNT* show *?thesis* unfolding *P-def* by *blast*
 qed
 qed
 hence $\exists\ u\ \delta. step-cond\ s\ \sigma\ t\ u\ \delta$ by (*auto simp*: *step-cond-def*)
 } note *simu* = *this*

have $P0: P\ r0\ \sigma0$ unfolding *P-def* using *linr r0-SNT R-to-RFC*[*OF lr*] by
force
 from *nt0* obtain t where $t\ 0 = t0$ and *steps*: $\bigwedge i. (t\ i, t\ (Suc\ i)) \in ?R$ by
auto
 hence $r0 \cdot \sigma0 = t\ 0$ using *t0* by *auto*

```

with  $P0$  have  $r0 \cdot \sigma0 = t\ 0 \wedge P\ r0\ \sigma0$  by auto
note  $dep\text{-}choice = dependent\text{-}nat\text{-}choice2\text{-}start[of\ \lambda\ i\ s\ sig.\ s \cdot sig = t\ i \wedge P\ s\ sig\ r0\ \sigma0$ 
 $\lambda\ i\ p1\ p2\ q1\ q2.\ step\text{-}cond\ p1\ p2\ (t\ (Suc\ i))\ q1\ q2,\ OF\ this]$ 
{
  fix  $s\ sig\ i$ 
  assume  $s \cdot sig = t\ i \wedge P\ s\ sig$ 
  with  $simu[of\ s\ sig\ t\ (Suc\ i)]\ steps[of\ i]$  obtain  $u\ \delta$  where
     $*: step\text{-}cond\ s\ sig\ (t\ (Suc\ i))\ u\ \delta$  by auto
  let  $?q = (u, \delta)$ 
  from  $*$  have  $u \cdot \delta = t\ (Suc\ i) \wedge P\ u\ \delta$ 
  unfolding  $step\text{-}cond\text{-}def$  by auto
  with  $*$  have
     $\exists\ u\ \delta.\ (u \cdot \delta = t\ (Suc\ i) \wedge P\ u\ \delta) \wedge$ 
     $step\text{-}cond\ s\ sig\ (t\ (Suc\ i))\ u\ \delta$  by auto
}
from  $dep\text{-}choice[OF\ this]$  obtain  $r\ \sigma$  where
 $*: P\ (r\ i)\ (\sigma\ i)\ r\ i \cdot \sigma\ i = t\ i\ step\text{-}cond\ (r\ i)\ (\sigma\ i)\ (t\ (Suc\ i))\ (r\ (Suc\ i))\ (\sigma\ (Suc\ i))$ 
for  $i$ 
by blast
define  $stmp$  where  $stmp\ i = stm\ (r\ i)\ (\sigma\ i)$  for  $i$ 
{
  fix  $i$ 
  have  $(stmp\ i,\ stmp\ (Suc\ i)) \in Id \cup mRel^{*-1}$ 
  unfolding  $stmp\text{-}def$  using  $*(3)[of\ i]$  unfolding  $step\text{-}cond\text{-}def$  by auto
}
hence  $rel\text{-}chain: \forall\ i.\ (stmp\ i,\ stmp\ (Suc\ i)) \in Id \cup mRel^{*-1}$  by auto
from  $wf\text{-}mRel$  have  $SN: SN\ (mRel^{*-1})$ 
by  $(simp\ add: SN\text{-}iff\text{-}wf)$ 
have  $\exists j.\ \forall i \geq j.\ (stmp\ i,\ stmp\ (Suc\ i)) \in Id - mRel^{-1}$ 
by  $(rule\ non\text{-}strict\text{-}ending[OF\ rel\text{-}chain],\ insert\ SN,\ auto\ simp: SN\text{-}def)$ 
then obtain  $j$  where  $eq\text{-}steps: \bigwedge i.\ i \geq j \implies (stmp\ i,\ stmp\ (Suc\ i)) \in Id - mRel^{*-1}$  by auto
from  $*(1)[of\ j,\ unfolded\ P\text{-}def]$ 
have  $rj: r\ j \in ?RFC$  by auto
{
  fix  $n\ s\ \sigma'$ 
  from  $eq\text{-}steps[of\ n + j]\ *(3)[of\ n + j,\ unfolded\ step\text{-}cond\text{-}def]$ 
  obtain  $\mu$  where  $narr\text{-}step\ (r\ (n + j))\ (\sigma\ (n + j))\ (r\ (Suc\ (n + j)))\ \mu\ (\sigma\ (Suc\ (n + j)))$ 
  using  $stmp\text{-}def$  by fastforce
  hence  $s = r\ (n + j) \wedge \sigma' = \sigma\ (n + j) \implies \exists u\ \delta.\ (u = r\ (Suc\ n + j) \wedge \delta = \sigma\ (Suc\ n + j)) \wedge (\exists \mu.\ (s,\ u) \in ?NS\ \mu \wedge \sigma' = \mu \circ_s \delta)$ 
  by  $(intro\ exI[of\text{-}r\ (Suc\ (n + j))]\ exI[of\text{-}\sigma\ (Suc\ (n + j))],\ unfold\ narr\text{-}step\text{-}def,\ auto)$ 
}
from  $exists\text{-}narrow\text{-}steps\text{-}to\text{-}infinite\text{-}rsteps[OF\ wf,\ of\ \lambda\ i\ t\ sig.\ t = r\ (i + j) \wedge sig = \sigma\ (i + j)\ ren\ r\ j\ \sigma\ j,\ OF\ this]$ 

```

```

have main:  $\neg$  SN-on ?N {r j}  $\neg$  SN-on ?R (?N* “ {r j})
  by auto
from rj main(1) have  $\neg$  SN-on ?N (rhss R) unfolding right-forw-closure-def
  by (metis SN-on-Image-rtrancl-iff SN-on-def singletonD)
from rj main(2) show  $\neg$  SN-on ?R (right-forw-closure R)
  unfolding right-forw-closure-def
  by (metis Image-singleton-iff SN-on-def rtrancl-image-advance-rtrancl)
qed

```

```

theorem right-linear-SN-rstep-RFC:
  assumes wf: wf-trs R
  and rlin:  $\bigwedge l r. lr \in R \implies \text{linear-term } (\text{snd } lr)$ 
  shows SN (rstep R)  $\longleftrightarrow$  SN-on (rstep R) (right-forw-closure R) (is ?A = ?B)
proof
  show ?A  $\implies$  ?B by force
next
  assume ?B
  with right-forw-closure-SN-main[OF rlin wf]
  show ?A by blast
qed

```

```

context
  fixes R :: ('f,'v)trs
  assumes wf: wf-trs R
  and WCR: WCR (rstep R)
  and overlay:  $\bigwedge l r. (False, l, r) \notin \text{critical-pairs ren } R R$ 
begin

```

```

lemma WCR0-not-SN-imp-non-terminating-innermost-rhs-instance:
  assumes  $\neg$  SN (rstep R)
  shows  $\exists r \sigma. r \in \text{rhss } R \wedge \neg$  SN-on (inn-rstep R) {r ·  $\sigma$ }  $\wedge \sigma$  ‘ vars-term r  $\subseteq$ 
  NF (rstep R)
proof –
  let ?R = rstep R
  let ?IR = inn-rstep R
  from WCR have WCR-on (rstep R) {t. SN-on (rstep R) {t}}
    unfolding WCR-on-def by auto
  from SN-innermost-switch-locally-confluent-overlay[OF this overlay wf] assms
  have  $\neg$  SN ?IR by auto
  then obtain t0 where  $\neg$  SN-on ?IR {t0} unfolding SN-defs by blast
  from not-SN-imp-subt-Tinf[OF this] obtain t1 where t1  $\in$  Tinf ?IR by blast
  from Tinf-inn-rstep-imp-first-root-step[OF this]
  obtain s t where (s, t)  $\in$  inn-rrstep R and nSN:  $\neg$  SN-on ?IR {t} by auto
  from inn-rrstep.cases[OF this(1)] obtain l r  $\sigma$  where lr: (l, r)  $\in$  R
    and NF: set (args (l ·  $\sigma$ ))  $\subseteq$  NF-trs R and t: t = r ·  $\sigma$  and s = l ·  $\sigma$  by metis

```

hence $r: r \in rhss\ R$ **by** *auto*
 from NF **have** $NF: \bigwedge u. u \triangleleft l \cdot \sigma \implies u \in NF\text{-}trs\ R$
by (*metis* (*no-types*, *lifting*) $NF\text{-}subterm\ subset\text{-}iff$
 $subterm.dual\text{-}order.strict\text{-}iff\text{-}order\ supseq.cases\ term.sel(4)$)
show *?thesis*
proof (*intro* $exI[of - r]$ $exI[of - \sigma]$ *conjI* $r\ nSN[unfolded\ t]$)
 {
 fix x
 assume $x \in vars\text{-}term\ r$
 hence $x \in vars\text{-}term\ l$ **using** $lr\ wf[unfolded\ wf\text{-}trs\text{-}def]$ **by** *auto*
 hence $l \triangleright Var\ x$ **using** $lr\ wf[unfolded\ wf\text{-}trs\text{-}def]$ **by** (*cases* l , *auto*)
 hence $l \cdot \sigma \triangleright Var\ x \cdot \sigma$ **by** *blast*
 from $NF[OF\ this]$ **have** $\sigma\ x \in NF\ (rstep\ R)$ **by** *auto*
 }
thus $\sigma \text{ ' vars-term } r \subseteq NF\text{-}trs\ R$ **by** *auto*
qed
qed

lemma *WCRO-one-step-simulation-by-narrowing*: **fixes** $s :: (f, v)term$
assumes $nSN: \neg SN\text{-}on\ (inn\text{-}rstep\ R)\ \{s \cdot \sigma\}$
and $NF\text{-}\sigma$: $\sigma \text{ ' vars-term } s \subseteq NF\ (rstep\ R)$
shows $\exists u\ \mu\ \delta. (s, u) \in narrow\text{-}step\text{-}s\ ren\ R\ \mu$
 $\wedge \neg SN\text{-}on\ (inn\text{-}rstep\ R)\ \{u \cdot \delta\}$
 $\wedge \delta \text{ ' vars-term } u \subseteq NF\ (rstep\ R)$
 $\wedge \sigma = \mu \circ_s \delta$
proof –
let $?Q = lhss\ R$
let $?IR = inn\text{-}rstep\ R$
let $?R = rstep\ R$
define VS **where** $VS = vars\text{-}term\ s$
from nSN **obtain** t **where** $step: (s \cdot \sigma, t) \in ?IR$ **and** $nSN: \neg SN\text{-}on\ ?IR\ \{t\}$
by (*meson* *step-reflects-SN-on*)
from $inn\text{-}rstep.cases[OF\ step]$ **obtain** $l\ r\ C\ \tau$
where $Cid: s \cdot \sigma = C\langle l \cdot \tau \rangle\ t = C\langle r \cdot \tau \rangle$
and $lr: (l, r) \in R$
and $NF\text{-}\tau$: $set\ (args\ (l \cdot \tau)) \subseteq NF\text{-}trs\ R$
by *metis*
define p **where** $p = hole\text{-}pos\ C$
from Cid **have** $p: p \in poss\ (s \cdot \sigma)$ **unfolding** $p\text{-}def$ **by** *auto*
from Cid **have** $unif: s \cdot \sigma \mid\text{-} p = l \cdot \tau$ **unfolding** $p\text{-}def$ **by** *auto*
from Cid **have** $t: t = replace\text{-}at\ (s \cdot \sigma)\ p\ (r \cdot \tau)$ **unfolding** $p\text{-}def$ **by** *auto*
from $NF\text{-}\tau$ **have** $NF\text{-}\tau$: $\bigwedge u. u \triangleleft l \cdot \tau \implies u \in NF\ ?R$
by (*metis* (*no-types*, *lifting*) $NF\text{-}subterm\ subset\text{-}iff$
 $subterm.dual\text{-}order.strict\text{-}iff\text{-}order\ supseq.cases\ term.sel(4)$)
from $poss\text{-}subst\text{-}choice[OF\ p]$ **consider** $(p)\ p \in poss\ s\ is\text{-}Fun\ (s \mid\text{-} p)$
 $\mid\ (sigma)\ x\ q$ **where** $x \in vars\text{-}term\ s\ q \in poss\ (\sigma\ x)\ s \cdot \sigma \mid\text{-} p = \sigma\ x \mid\text{-} q$
by *auto*
thus *?thesis*
proof *cases*

```

case sigma
from sigma(2,3)[unfolded unif] obtain C where  $\sigma x = C \langle l \cdot \tau \rangle$ 
  by (metis ctxt-supt-id)
from rstepI[OF lr this refl] NF-sigma sigma(1) have False by auto
thus ?thesis ..
next
case p
with unif have unif:  $s \mid\!-\! p \cdot \sigma = l \cdot \tau$  by auto
from mgu-vd-complete[OF this, of ren]
obtain  $\mu 1 \ \mu 2 \ \delta$  where mgu: mgu- $\text{vd}$  ren ( $s \mid\!-\! p$ )  $l = \text{Some } (\mu 1, \mu 2)$ 
  and sigma:  $\sigma = \mu 1 \circ_s \delta$  and tau:  $\tau = \mu 2 \circ_s \delta$  and unif:  $s \mid\!-\! p \cdot \mu 1 = l \cdot \mu 2$ 
  by auto
define C where  $C = \text{ctxt-of-pos-term } p \ s$ 
from p have Cmu:  $C \cdot_c \mu 1 = \text{ctxt-of-pos-term } p \ (s \cdot \mu 1)$  for  $\mu 1 :: (f, v)\text{subst}$ 
unfolding C-def
  by (auto simp: ctxt-of-pos-term-subst)
define u where  $u = (C \cdot_c \mu 1) \langle r \cdot \mu 2 \rangle$ 
have  $(s, u) \in \text{narrows-r-p-s ren } R \ (l, r) \ p \ \mu 1$ 
  unfolding narrows-r-p-s-def u-def Cmu using p lr mgu by auto
hence narr:  $(s, u) \in \text{narrow-step-s ren } R \ \mu 1$  unfolding narrow-step-s-def by
blast

have  $t = (C \cdot_c \sigma) \langle r \cdot \tau \rangle$  unfolding t Cmu by simp
also have  $\dots = u \cdot \delta$  unfolding sigma tau u-def
  by simp
finally have tu:  $t = u \cdot \delta$  .

with nSN have nSN:  $\neg \text{SN-on } ?IR \ \{u \cdot \delta\}$  by simp

from wf[unfolded wf-trs-def] lr have vc: vars-term  $r \subseteq \text{vars-term } l$  by auto
from NF-sigma have NF:  $x \in VS \implies \sigma x \in NF \ ?R$  for  $x$  by (auto simp:
VS-def)

{
  fix v
  assume  $v \in \delta \text{ 'vars-term } u$ 

  then obtain x where  $x \in \text{vars-term } u$  and  $v: v = \delta x$  by auto
  from this[unfolded u-def] have  $x \in \text{vars-ctxt } (C \cdot_c \mu 1) \vee x \in \text{vars-term } (r \cdot$ 
 $\mu 2)$ 
    by (simp add: vars-term-ctxt-apply)
  hence  $x \in \text{vars-term } (s \cdot \mu 1)$ 
  proof
    have  $s = C \langle s \mid\!-\! p \rangle$  unfolding C-def using p
    by (simp add: ctxt-supt-id)
    from arg-cong[OF this, of  $\lambda t. t \cdot \mu 1$ ]
    have smu1:  $s \cdot \mu 1 = (C \cdot_c \mu 1) \langle s \mid\!-\! p \cdot \mu 1 \rangle$  by auto

```

```

    assume  $x \in \text{vars-ctxt } (C \cdot_c \mu 1)$ 
    thus  $x \in \text{vars-term } (s \cdot \mu 1)$  unfolding smu1
    by (simp add: vars-term-ctxt-apply)
  next
  from  $p$  have vars-sp:  $\text{vars-term } (s \mid -p) \subseteq \text{vars-term } s$  by (metis vars-term-subt-at)
    assume  $x \in \text{vars-term } (r \cdot \mu 2)$ 
    with  $vc$  have  $x \in \text{vars-term } (l \cdot \mu 2)$ 
    by (auto simp: vars-term-subst)
    from this[folded unif] vars-sp show  $x \in \text{vars-term } (s \cdot \mu 1)$ 
    by (auto simp: vars-term-subst)
  qed
  hence  $v \in \delta \text{ ' vars-term } (s \cdot \mu 1)$  unfolding  $v$  by auto
  from this[unfolded vars-term-subst VS-def[symmetric]]
  have  $v \in \delta \text{ ' } \bigcup (\text{vars-term ' } \mu 1 \text{ ' } VS)$  .

  then obtain  $x \ y$  where  $x \in VS$  and  $y: y \in \text{vars-term } (\mu 1 \ x)$  and  $v: v = \delta$ 
  y by auto
    from  $NF[OF \text{ this}(1)]$  have  $NF: \sigma \ x \in NF \ ?R$  by auto
    from  $y$  have  $\text{Var } y \trianglelefteq \mu 1 \ x$  by auto
    hence  $\text{Var } y \cdot \delta \trianglelefteq \mu 1 \ x \cdot \delta$  by blast
    hence  $v \trianglelefteq \sigma \ x$  unfolding  $\sigma$   $v$  by (auto simp: subst-compose-def)
    with  $NF$  have  $v \in NF \ ?R$  by (rule NF-subterm)
  }
  hence  $NFu: \delta \text{ ' vars-term } u \subseteq NF \ ?R$  by auto

  show ?thesis
    by (intro exI[of -  $u$ ], rule exI[of -  $\mu 1$ ], rule exI[of -  $\delta$ ], intro conjI narr nSN
    sigma NFu)
  qed
  qed

```

lemma *WCRO-not-SN-rstep-imp-not-SN-rstep*: **assumes** $\neg SN \ (rstep \ R)$
shows $\neg SN\text{-on } (rstep \ R)$ (*right-forw-closure* R)
proof –
define $nfs :: \text{bool}$ **where** $nfs = \text{undefined}$
define P **where** $P \ (n :: \text{nat}) \ (r :: ('f, 'v)\text{term}) \ \sigma = (\sigma \text{ ' vars-term } r \subseteq NF\text{-trs } R$
 $\wedge \neg SN\text{-on } (\text{inn-rstep } R) \ \{r \cdot \sigma\})$
for $n \ r \ \sigma$
from *WCRO-not-SN-imp-non-terminating-innermost-rhs-instance*[*OF* *assms*]
obtain $r \ \sigma$ **where** $P0: P \ 0 \ r \ \sigma$ **and** $r: r \in rhss \ R$
unfolding $P\text{-def}$ **by** *auto*
define Q **where** $Q \ (n :: \text{nat}) \ s \ (\sigma :: ('f, 'v)\text{subst}) \ u \ (\delta :: ('f, 'v)\text{subst}) =$
 $(\exists \mu. (s, u) \in \text{narrow-step-s ren } R \ \mu \wedge \sigma = \mu \circ_s \delta)$ **for** $n \ s \ \sigma \ u \ \delta$
have $P \ n \ s \ \sigma \implies \exists u \ \delta. P \ (Suc \ n) \ u \ \delta \wedge Q \ n \ s \ \sigma \ u \ \delta$ **for** $n \ s \ \sigma$
using *WCRO-one-step-simulation-by-narrowing*[*of* $s \ \sigma$]
unfolding $P\text{-def}$ $Q\text{-def}$ **by** *auto*
from *exists-narrow-steps-to-infinite-rsteps*[*OF* *wf*, *of* $P \text{ ren } r \ \sigma$, *OF* *this*[*unfolded* $Q\text{-def}$] $P0$]

```

have main:  $\neg$  SN-on (narrow-step ren R)  $\{r\} \neg$  SN-on (rstep R) ((narrow-step ren R)* “ $\{r\}$ ) by auto

from r have (narrow-step ren R)* “ $\{r\} \subseteq$  right-forw-closure R
  unfolding right-forw-closure-def by auto
with main(2) show  $\neg$  SN-on (rstep R) (right-forw-closure R) unfolding SN-defs
by blast
from main(1) have  $\neg$  SN-on (narrow-step ren R) (rhss R) using r by fast
qed
end

theorem WCRO-SN-rstep-RFC: assumes wf: wf-trs R
  and WCR: WCR (rstep R)
  and overlay:  $\bigwedge l r. (False, l, r) \notin$  critical-pairs ren R R
shows SN (rstep R)  $\longleftrightarrow$  SN-on (rstep R) (right-forw-closure R) (is ?A = ?B)
proof
  show ?A  $\implies$  ?B by force
next
  assume ?B
  with WCRO-not-SN-rstep-imp-not-SN-rstep[OF assms] show ?A by blast
qed
end
end

```

References

- [1] N. Dershowitz. Termination of linear rewriting systems (preliminary version). In S. Even and O. Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981. Proceedings*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458. Springer, 1981.
- [2] N. Dershowitz and C. Hoot. Natural termination. *Theor. Comput. Sci.*, 142(2):179–207, 1995.
- [3] B. Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. In A. Voronkov, editor, *Logic Programming and Automated Reasoning, International Conference, LPAR ’92, St. Petersburg, Russia, July 15-20, 1992. Proceedings*, volume 624 of *Lecture Notes in Computer Science*, pages 285–296. Springer, 1992.
- [4] R. Thiemann, D. Hofbauer, U. Le Huitouze, and J. Waldmann. New and Formalized Proofs for Right-Forward Closures and Core Matrix Interpretations. In F. Pfenning, editor, *11th International Conference on Formal Structures for Computation and Deduction (FSCD 2026)*, volume 378 of *Leibniz International Proceedings in Informatics (LIPIcs)*,

pages 32:1–32:19, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.