Executable Transitive Closures*

René Thiemann

May 28, 2015

Abstract

We provide a generic work-list algorithm to compute the (reflexive-)transitive closure of relations where only successors of newly detected states are generated. In contrast to our previous work [2], the relations do not have to be finite, but each element must only have finitely many (indirect) successors. Moreover, a subsumption relation can be used instead of pure equality. An executable variant of the algorithm is available where the generic operations are instantiated with list operations.

This formalization was performed as part of the IsaFoR/CeTA project\(^1\) [3], and it has been used to certify size-change termination proofs where large transitive closures have to be computed.

Contents

1 A work-list algorithm for reflexive-transitive closures 1
  1.1 The generic case .................................................. 2
  1.2Instantiation using list operations ............................. 9

1 A work-list algorithm for reflexive-transitive closures

theory RTrancl
imports ../Regular-Sets/Regexp-Method
begin

In previous work [2] we described a generic work-list algorithm to compute reflexive-transitive closures for finite relations: given a finite relation \(r\), it computed \(r^\ast\).

In the following, we develop a similar, though different work-list algorithm for reflexive-transitive closures, it computes \(r^\ast \ '' \ init\) for a given relation \(r\) and finite set \(init\). The main differences are that

\*Supported by FWF (Austrian Science Fund) project P22767-N13.
\(^1\) http://cl-informatik.uibk.ac.at/software/ceta
The relation $r$ does not have to be finite, only \{ $b. (a, b) \in r^*$ \} has to be finite for each $a$. Moreover, it is no longer required that $r$ is given explicitly as a list of pairs. Instead $r$ must be provided in the form of a function which computes for each element the set of one-step successors.

- One can use a subsumption relation to indicate which elements to no longer have to be explored.

These new features have been essential to certify size-change termination proofs [1] where the transitive closure of all size-change graphs has to be computed. Here, the relation is size-change graph composition.

- Given an initial set of size-change graphs with $n$ arguments, there are roughly $N := 3^n^2$ many potential size-change graphs that have to be considered as left-hand sides of the composition relation. Since the composition relation is even larger than $N$, an explicit representation of the composition relation would have been too expensive. However, using the new algorithm the number of generated graphs is usually far below the theoretical upper bound.

- Subsumption was useful to generate even fewer elements.

1.1 The generic case

Let $r$ be some finite relation.

We present a standard work-list algorithm to compute all elements that are reachable from some initial set. The algorithm is generic in the sense that the underlying data structure can freely be chosen, you just have to provide certain operations like union, selection of an element.

In contrast to [2], the algorithm does not demand that $r$ is finite and that $r$ is explicitly provided (e.g., as a list of pairs). Instead, it suffices that for every element, only finitely many elements can be reached via $r$, and $r$ can be provided as a function which computes for every element $a$ all one-step successors w.r.t. $r$. Hence, $r$ can in particular be any well-founded and finitely branching relation.

The algorithm can further be parametrized by a subsumption relation which allows for early pruning.

In the following locales, $r$ is a relation of type `$a \Rightarrow a'$, the successors of an element are represented by some collection type `$b'$ which size can be measured using the `size` function. The selection function `sel` is used to meant to split a non-empty collection into one element and a remaining collection. The union on `$b'$ is given by `un`.

locale subsumption =
fixes $r :: \forall a \Rightarrow \forall b$
and $\text{subsames} :: \forall a \Rightarrow \forall a \Rightarrow \text{bool}$
and $\text{set-of} :: \forall b \Rightarrow \forall a \Rightarrow \text{set}$
assumes
  $\text{subsames-refl} :: \Lambda a. \text{subsames} a a$
and $\text{subsames-trans} :: \Lambda a b c. \text{subsames} a b \Rightarrow \text{subsames} b c \Rightarrow \text{subsames} c a$
and $\text{subsames-step} :: \Lambda a b c. \text{subsames} a b \Rightarrow c \in \text{set-of} (r b) \Rightarrow \exists d \in \text{set-of} (r a). \text{subsames} d c$
begin
abbreviation $R$ where $R \equiv \{ (a, b) | b \in \text{set-of} (r a) \}$
end

locale subsumption-impl $=$ subsumption $r$ subsames \text{set-of}
for $r :: \forall a \Rightarrow \forall b$
and $\text{subsames} :: \forall a \Rightarrow \forall a \Rightarrow \text{bool}$
and $\text{set-of} :: \forall b \Rightarrow \forall a \Rightarrow \text{set}$
fixes
  $\text{sel} :: \forall b \Rightarrow \forall a \Rightarrow \forall b \Rightarrow \forall a \Rightarrow \text{set}$
and $\text{un} :: \forall b \Rightarrow \forall b \Rightarrow \forall b$
and $\text{size} :: \forall b \Rightarrow \forall \text{nat}$
assumes
  $\text{set-of-fin} :: \Lambda b. \text{finite} (\text{set-of} b)$
and $\text{sel} :: \Lambda b a c. \text{set-of} b \neq \{ \} \Rightarrow \text{sel} b = (a, c) \Rightarrow \text{set-of} b = \text{insert} a \text{ (set-of c)}$
and $\text{size} :: \forall b \Rightarrow \forall \text{nat} > \text{size} c$
and $\text{un} :: \text{set-of} (\text{un} a b) = \text{set-of} a \cup \text{set-of} b$

locale relation-subsumption-impl $=$ subsumption-impl $r$ subsames \text{set-of} \text{sel} \text{un} \text{size}
for $r$ subsames \text{set-of} \text{sel} \text{un} \text{size}$+$
assumes $\text{rtrancl-fin} :: \Lambda a. \text{finite} \{ (b. (a, b) \in \text{set-of} (r a) \}^*$
begin

lemma finite-Rs: assumes init: finite init
  shows finite $\{ \text{set-of} \}^*$
proof
  let $\lambda ?R = \lambda a. \{ (b. (a, b) \in \text{set-of} \}^*$
  let $\lambda ?S = \{ ?R a | a \in \text{init} \}$
  have id: $\forall \text{set-of} \text{init} \Rightarrow$ $\text{set-of} \text{init}$
  show $\text{thesis}$ unfolding id
  proof (rule)
    fix $M$
    assume $M \in ?S$
    then obtain $a$ where $M = ?R a$ by auto
    show $\text{finite} M$ unfolding $M$ by (rule rtrancl-fin)
next
  show finite $\{ (a, b) \in \text{set-of} \}^*$
    using init by auto
  qed
  qed

a standard work-list algorithm with subsumption
function mk-rtrancl-main where
mk-rtrancl-main todo fin = (if set-of todo = [] then fin
  else let (a,tod) = sel todo
    in (if (\exists b \in fin. subsumes b a) then mk-rtrancl-main tod fin
      else mk-rtrancl-main (un (r a) tod) (insert a fin)))
by pat-completeness auto

termination mk-rtrancl-main
proof –
  let \(?r1 = \lambda (todo, fin). card (R^* " (set-of todo) − fin)
  let \(?r2 = \lambda (todo, fin). size todo
  show \(?thesis
  proof
    show wf (measures [\(?r1,\(?r2]) by simp
  next
    fix todo fin pair tod a
    assume nempty: set-of todo ≠ [] and pair1: pair = sel todo and pair2: (a,tod)
    = pair
      from pair1 pair2 have pair: sel todo = (a,tod) by simp
      from set-of-fin have fin: finite (set-of todo) by auto
      note sel = sel[OF nempty pair]
      show ((tod,fin),(todo,fin)) ∈ measures [\(?r1,\(?r2]
      proof (rule measures-lesseq[OF - measures-less], unfold split)
        from sel
        show size tod < size todo by simp
      next
        from sel have subset: R^* " set-of tod − fin ⊆ R^* " set-of todo − fin (is
          ?l ⊆ ?r) by auto
        show card ?l ≤ card ?r
          by (rule card-mono[OF - subset], rule finite-Diff, rule finite-Rs[OF fin])
        qed
  next
    fix todo fin a tod pair a
    assume nempty: set-of todo ≠ [] and pair1: pair = sel todo and pair2: (a,tod)
    = pair and nmem: ¬ (\exists b \in fin. subsumes b a)
      from pair1 pair2 have pair: sel todo = (a,tod) by auto
      from nmem subsumes-refl[of a] have nmem: a ∉ fin by auto
      from set-of-fin have fin: finite (set-of todo) by auto
      note sel = sel[OF nempty pair]
      show ((un (r a) tod,insert a fin),(todo,fin)) ∈ measures [\(?r1,\(?r2]
      proof (rule measures-less, unfold split,
        rule psubset-card-mono[OF finite-Diff[OF finite-Rs[OF fin]]])
        let ?l = R^* " (set-of (un (r a) tod) − insert a fin
        let ?r = R^* " set-of todo − fin
        from sel have at: a ∈ set-of todo by auto
        have ar: a ∈ ?r using nmem at by auto
        show ?l ⊆ ?r
        proof

show \( l \neq r \) using \( \text{ar} \) by auto
next
have \( R^* \) “ set-of \( (r a) \) \( \subseteq R^* \) “ set-of todo
proof
fix \( b \)
assume \( b \in R^* \) “ set-of \( (r a) \)
then obtain \( c \) where \( cb : (c,b) \in R^* \) and \( ca : c \in \text{set-of} \ (r a) \) by blast
hence \( ab : (a,b) \in R O R^* \) by auto
have \( (a,b) \in R^* \)
by (rule set-mp[OF - \( ab \)], regexp)
with \( \text{at show} \ b \in R^* \) “ set-of todo by auto
qed
thus \( l \subseteq R^* \) using \( \text{sel} \) unfolding \( \text{un} \) by auto
qed
qed
qed
qed

declare mk-rtrancl-main.simps[simp del]

lemma mk-rtrancl-main-sound: set-of todo \( \cup \) \( \text{fin} \) \( \subseteq R^* \) “ init \( = \) mk-rtrancl-main
\( \text{todo} \ \text{fin} \) \( \subseteq R^* \) “ init
proof (induct todo \( \text{fin} \) rule: mk-rtrancl-main.induct)
case (1 todo \( \text{fin} \) )
note simp = mk-rtrancl-main.simps[of todo \( \text{fin} \) ]
show \?case
proof (cases set-of todo = { })
case True
show \?thesis unfolding simp using True (3) by auto
next
case False
hence nempty: (set-of todo = { }) \( = \) False by auto
obtain \( a \) \( \text{tod} \) where \( \text{self} : \text{sel} \ \text{todo} = (a,\text{tod}) \) by force
note sel = sel[OF False sel]

note IH1 = 1(1)[OF False refl sel[symmetric]]
note IH2 = 1(2)[OF False refl sel[symmetric]]
note simp = simp nempty if-False Let-def sel
show \?thesis
proof (cases \( \exists \ b \in \text{fin.} \ \text{subsumes} \ b \ a \) )
case True
hence mk-rtrancl-main todo \( \text{fin} \) = mk-rtrancl-main tod \( \text{fin} \)
unfolding simp by simp
with IH1[OF True] (3) show \?thesis using sel by auto
next
case False
hence id: mk-rtrancl-main todo \( \text{fin} \) = mk-rtrancl-main (un (r a) tod) (insert a \( \text{fin} \) ) unfolding simp by simp
show \?thesis unfolding id
proof (rule IH2[OF False])

5
from sel 1(3) have subset: set-of todo ∪ insert a fin ⊆ R⁺⁺ "init by auto"
{
  fix b
  assume b: b ∈ set-of (r a)
  hence ab: (a, b) ∈ R by auto
  from sel 1(3) have a ∈ R⁺⁺ "init by auto"
  then obtain c where c: c ∈ init and ca: (c, a) ∈ R⁺⁺ O R by auto
  have (c, b) ∈ R⁺⁺
    by (rule set-mp[OF - cb], regexp)
    with c have b ∈ R⁺⁺ "init by auto"
}
with subset
show set-of (un (r a) tod) ∪ (insert a fin) ⊆ R⁺⁺ "init
unfolding un using sel by auto
qed
qed
qed
lemma mk-rtrancl-main-complete:
[∀ a. a ∈ init ⇒ ∃ b. b ∈ set-of todo ∪ fin ∧ subsumes b a]
⇒ ∀ a b . a ∈ fin ⇒ b ∈ set-of (r a) ⇒ ∃ c. c ∈ set-of todo ∪ fin ∧
subsumes c b
⇒ c ∈ R⁺⁺ "init"
⇒ ∃ b. b ∈ mk-rtrancl-main todo fin ∧ subsumes b c.
proof (induct todo fin rule: mk-rtrancl-main.induct)
case (1 todo fin)
from 1(5) have c: c ∈ R⁺⁺ "init .
note finr = 1(4)
note init = 1(3)
note simp = mk-rtrancl-main.simps[of todo fin]
show ?case
proof (cases set-of todo = {})
case True
  hence id: mk-rtrancl-main todo fin = fin unfolding simp by simp
from c obtain a where a: a ∈ init and ac: (a, c) ∈ R⁺⁺ by blast
show ?thesis unfolding id using ac
proof (induct rule: rtrancl-induct)
case base
  from init[OF a] show ?case unfolding True by auto
next
case (step b c)
  from step(3) obtain d where d: d ∈ fin and db: subsumes d b by auto
  from step(2) have cb: c ∈ set-of (r b) by auto
  from subsumes-step[OF db cb] obtain a where a: a ∈ set-of (r d) and ac:
subsumes a c by auto
  from finr[OF d a] obtain e where e: e ∈ fin and ea: subsumes e a unfolding
True by auto
}

from substms-\textit{trans}[\textit{OF} \ \textit{ca} \ \textit{ac}] \ e

show \ ?\textit{case} by auto

qed

next

case \textit{False}

hence \textit{nempty}: \ (\textit{set-of} \ \textit{todo} = \{\}) = \textit{False} by simp

obtain \textit{A} \ \textit{tod} where \ selt: sel \ \textit{todo} = (\textit{A},\textit{tod}) by force

note \textit{simp} = \textit{nempty} \ simp \ \textit{if-False} \ \textit{Let-def} \ selt

note \textit{sel} = \textit{set}(\textit{OF} \ \textit{False} \ \textit{selt})

note \textit{IH1} = \textit{1(1)}[\textit{OF} \ \textit{False} \ \textit{refl} \ \textit{selt}][\textit{symmetric}]

note \textit{IH2} = \textit{1(2)}[\textit{OF} \ \textit{False} \ \textit{refl} \ \textit{selt}][\textit{symmetric}]

show \ ?\textit{thesis}

proof \ (cases \ \exists \ \textit{b} \ \in \ \textit{fin}. \ substms \ \textit{b} \ \textit{A})

\hspace{1cm}case \ \textit{True} note \ \textit{aTrue} = \textit{this}

\hspace{2cm}hence \ \textit{id}: \ \textit{mk-rtrancl-main} \ \textit{todo} \ \textit{fin} = \ \textit{mk-rtrancl-main} \ \textit{tod} \ \textit{fin}

\hspace{2cm}unfolding \ simp \ by \ simp

\hspace{1cm}from \ \textit{True} obtain \ \textit{b} \ where \ \textit{b}: \ \textit{b} \ \in \ \textit{fin} \ \textit{and} \ \textit{ba}: \ substms \ \textit{b} \ \textit{A} \ by \ auto

\hspace{1cm}show \ ?\textit{thesis} unfolding \ \textit{id}

\hspace{2cm}fix \ \textit{a}

\hspace{2cm}assume \ \textit{a}: \ \textit{a} \ \in \ \textit{init}

\hspace{2cm}from \ init[\textit{OF} \ \textit{a}] \ obtain \ \textit{c} \ where \ \textit{c}: \ \textit{c} \ \in \ \textit{set-of} \ \textit{todo} \ \cup \ \textit{fin} \ \textit{and} \ \textit{ca}: \ substms

\hspace{3cm}c \ a \ by \ blast

\hspace{4cm}show \ \exists \ \textit{b}. \ \textit{b} \ \in \ \textit{set-of} \ \textit{tod} \ \cup \ \textit{fin} \ \wedge \ substms \ \textit{b} \ \textit{a}

\hspace{4cm}proof \ (cases \ \textit{c} = \textit{A})

\hspace{5cm}case \ \textit{False}

\hspace{6cm}thus \ ?\textit{thesis} using \ \textit{c} \ \textit{ca} \ \textit{sel} \ by \ auto

\hspace{5cm}next

\hspace{6cm}case \ \textit{True}

\hspace{7cm}show \ ?\textit{thesis} using \ \textit{b} \ substms-\textit{trans}[\textit{OF} \ \textit{ba}, \ \textit{of} \ \textit{a}] \ \textit{ca} \ \textit{unfolding} \ \textit{True}[\textit{symmetric}] \ by \ auto

\hspace{4cm}qed

\hspace{2cm}next

\hspace{3cm}fix \ \textit{a} \ \textit{c}

\hspace{4cm}assume \ \textit{a}: \ \textit{a} \ \in \ \textit{fin} \ \textit{and} \ \textit{c}: \ \textit{c} \ \in \ \textit{set-of} \ \textit{(r} \ \textit{a})

\hspace{3cm}from \ finr[\textit{OF} \ \textit{a} \ \textit{c}] \ obtain \ \textit{e} \ where \ \textit{e}: \ \textit{e} \ \in \ \textit{set-of} \ \textit{todo} \ \cup \ \textit{fin} \ \textit{and} \ \textit{ec}: \ substms

\hspace{4cm}e \ \textit{c} \ by \ auto

\hspace{5cm}show \ \exists \ \textit{d}. \ \textit{d} \ \in \ \textit{set-of} \ \textit{tod} \ \cup \ \textit{fin} \ \wedge \ substms \ \textit{d} \ \textit{c}

\hspace{5cm}proof \ (cases \ \textit{A} = \textit{e})

\hspace{6cm}case \ \textit{False}

\hspace{7cm}with \ \textit{ec} \ \textit{show} \ ?\textit{thesis} \ using \ \textit{sel} \ by \ auto

\hspace{6cm}next

\hspace{7cm}case \ \textit{True}

\hspace{8cm}from \ substms-\textit{trans}[\textit{OF} \ \textit{ba}[\textit{unfolded} \ \textit{True}] \ \textit{ec}]

\hspace{8cm}show \ ?\textit{thesis} using \ \textit{b} \ by \ auto

\hspace{7cm}qed

\hspace{5cm}qed

\hspace{4cm}next

\hspace{3cm}case \ \textit{False}
hence id: mk-rtrancl-main todo fin = mk-rtrancl-main (un (r A) tod) (insert A fin) unfolding simp by simp

show ?thesis unfolding id

proof (rule IH2[OF False])
  fix a
  assume a: a ∈ init
  from init[OF a]
  show ∃ b. b ∈ set-of (un (r A) (tod)) ∪ insert A fin ∧ subsumes b a
    using sel unfolding un by auto

next
  fix a b
  assume a: a ∈ insert A fin and b: b ∈ set-of (r a)
  show ∃ c. c ∈ set-of (un (r A) tod) ∪ insert A fin ∧ subsumes c b
    proof (cases a ∈ fin)
    case True
      from finr[OF True b] show ?thesis using sel unfolding un by auto
    next
    case False
      with a have a: A = a by simp
      show ?thesis unfolding a un using b subsumes-refl[of b] by blast
    qed
  qed
  qed
  qed

definition mk-rtrancl where mk-rtrancl init ≡ mk-rtrancl-main init {}

lemma mk-rtrancl-sound: mk-rtrancl init ⊆ R∗ ∘ set-of init
  unfolding mk-rtrancl-def
  by (rule mk-rtrancl-main-sound, auto)

lemma mk-rtrancl-complete: assumes a: a ∈ R∗ ∘ set-of init
  shows ∃ b. b ∈ mk-rtrancl init ∧ subsumes b a
  unfolding mk-rtrancl-def
  proof (rule mk-rtrancl-main-complete[of - a])
    fix a
    assume a: a ∈ set-of init
    thus ∃ b. b ∈ set-of init ∪ {} ∧ subsumes b a using subsumes-refl[of a] by blast
  qed auto

lemma mk-rtrancl-no-subsumption: assumes subsumes = (op =)
  shows mk-rtrancl init = R∗ ∘ set-of init
  by auto

end
1.2 Instantiation using list operations

It follows an implementation based on lists. Here, the working list algorithm is implemented outside the locale so that it can be used for code generation. In general, it is not terminating, therefore we use partial_function instead of function.

\begin{verbatim}
partial-function (tailrec) mk-rtrancl-list-main where
| [code]: mk-rtrancl-list-main subsumes r todo fin = (case todo of [] ⇒ fin
| Cons a tod ⇒
| (if (∃ b ∈ set fin. subsumes b a) then mk-rtrancl-list-main subsumes r tod fin
| else mk-rtrancl-list-main subsumes r (r a @ tod) (a # fin)))

definition mk-rtrancl-list where
mk-rtrancl-list subsumes r init ≡ mk-rtrancl-list-main subsumes r init []

locale subsumption-list = subsumption r subsumes set
for r :: 'a ⇒ 'a list and subsumes :: 'a ⇒ 'a ⇒ bool

locale relation-subsumption-list = subsumption-list r subsumes
assumes rtrancl-fin: (∀ a. finite {b. (a,b) ∈ { (a,b) . b ∈ set (r a)} ^*}

abbreviation (input) sel-list where sel-list x = case x of Cons h t ⇒ (h,t)

sublocale subsumption-list ⊆ subsumption-impl r subsumes set sel-list append length
proof (unfold-locales, rule finite-set)
fix b a c
assume set b ≠ {} and sel-list b = (a,c)
thus set b = insert a (set c) ∧ length c < length b
  by (cases b, auto)
qed auto

sublocale relation-subsumption-list ⊆ relation-subsumption-impl r subsumes set sel-list append length
by (unfold-locales, rule rtrancl-fin)

context relation-subsumption-list begin

The main equivalence proof between the generic work list algorithm and the one operating on lists

lemma mk-rtrancl-list-main: fin = set finl =⇒ set (mk-rtrancl-list-main subsumes r todo finl) = mk-rtrancl-main todo fin

proof (induct todo fin arbitrary: finl rule: mk-rtrancl-main.induct)
  case (1 todo fin fin)
  note simp = mk-rtrancl-list-main.simps[of - todo finl] mk-rtrancl-main.simps[of todo fin]
  show ?case (is ?l = ?r)
\end{verbatim}
proof (cases todo)
case Nil
  show ?thesis unfolding simp unfolding Nil 1(3) by simp
next
case (Cons a tod)
  show ?thesis proof (cases ∃ b ∈ fin. subsumes b a)
    case True
      from True have l: ?l = set (mk-rtrancl-list-main subsumes r tod finl)
      unfolding simp unfolding Cons 1(3) by simp
    from True have r: ?r = mk-rtrancl-main tod fin
    unfolding simp unfolding Cons by auto
    show ?thesis unfolding l r
      by (rule 1(1)[OF - refl - True], insert 1(3) Cons, auto)
  next
    case False
      from False have l: ?l = set (mk-rtrancl-list-main subsumes r (r a ® tod) (a # finl))
      unfolding simp unfolding Cons 1(3) by simp
      from False have r: ?r = mk-rtrancl-main (r a ® tod) (insert a fin)
      unfolding simp unfolding Cons by auto
      show ?thesis unfolding l r
        by (rule 1(2)[OF - refl - False], insert 1(3) Cons, auto)
    qed
  qed
qed

lemma mk-rtrancl-list: set (mk-rtrancl-list subsumes r init) = mk-rtrancl init
  unfolding mk-rtrancl-list-def mk-rtrancl-def
  by (rule mk-rtrancl-list-main, simp)

dep

References
