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 al-
gorithm is available where the generic operations are instantiated with
list operations.

This formalization was performed as part of the IsaFoR/CeTA project \cite{ IsaFoR/CeTA }, 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
  1.2 Instantiation using list operations .................... 9

1 A work-list algorithm for reflexive-transitive clo-
sures

t
theory RTrancl
import
 ../Regular-Sets/Regexp-Method
begin

In previous work \cite{ previous_work } we described a generic work-list algorithm to com-
pute reflexive-transitive closures for finite relations: given a finite relation $r$, it computed $r^*$. 

In the following, we develop a similar, though different work-list algo-
rithm for reflexive-transitive closures, it computes $r^* \text{ init}$ for a given re-
lation $r$ and finite set $\text{init}$. The main differences are that

*Supported by FWF (Austrian Science Fund) project P22767-N13.

\footnote{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 \( N := 3^2^n \) 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 \textit{size} function. The selection function \textit{sel} is used to mean to split a non-empty collection into one element and a remaining collection. The union on \( 'b \) is given by \textit{un}.

locale subsumption =
fixes $r :: 'a ⇒ 'b$
and subsumes :: 'a ⇒ 'a ⇒ bool
and set-of :: 'b ⇒ 'a set
assumes
  subsumes-refl: $\forall a. \text{subsumes} \ a \ a$
  and subsumes-trans: $\forall a \ b \ c. \text{subsumes} \ a \ b \implies \text{subsumes} \ b \ c \implies \text{subsumes} \ a \ c$
  and subsumes-step: $\forall a \ b \ c. \text{subsumes} \ a \ b \implies c \in \text{set-of} \ (r \ b) \implies \exists d \in \text{set-of} \ (r \ a). \text{subsumes} \ d \ c$
begin
abbreviation $R$ where $R \equiv \{(a, b). \ b \in \text{set-of} \ (r \ a)\}$
end
locale subsumption-impl = subsumption $r$ subsumes set-of
for $r :: 'a ⇒ 'b$
and subsumes :: 'a ⇒ 'a ⇒ bool
and set-of :: 'b ⇒ 'a set +
fixes
  sel :: 'b ⇒ 'a × 'b
and un :: 'b ⇒ 'b ⇒ 'b
and size :: 'b ⇒ nat
assumes
  set-of-fin: $\forall b. \text{finite} \ (\text{set-of} \ b)$
  and sel: $\forall b \ a \ c. \text{set-of} \ b \neq {} \implies \text{sel} \ b = (a, c) \implies \text{set-of} \ b = \text{insert} \ a \ (\text{set-of} \ c)$
  ∧ size $b \geq$ size $c$
and un: $\text{set-of} \ (\text{un} \ a \ b) = \text{set-of} \ a \cup \text{set-of} \ b$
locale relation-subsumption-impl = subsumption-impl $r$ subsumes set-of sel un size
for $r \text{ subsumes set-of sel un size} +$
assumes
  rtrancl-fin: $\forall a. \text{finite} \ \{b. (a, b) \in (r \ a) \}^*$
begin
lemma finite-Rs: assumes init: finite init
  shows finite $(R^* \{ " \text{init} \})$
proof −
  let $?R = \lambda a. \{b. (a, b) \in R^*\}$
  let $?S = \{?R \ a \ | \ a \in \text{init}\}$
  have id: $R^* \{ " \text{init} \} = \bigcup ?S$ by auto
  show ?thesis unfolding id
proof (rule)
  fix $M$
  assume $M \in ?S$
  then obtain $a$ where $M = ?R \ a$ by auto
  show finite $M$ unfolding $M$ by (rule rtrancl-fin)
next
  show finite $\{\{b. (a, b) \in R^*\} \ | \ a \in \text{init}\}$
    using init by auto
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 (∃ b ∈ 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 = λ (todo, fin). card (R^* ` (set-of todo) − fin)
let ?r2 = λ (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 todo) − 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
      assume nempty: set-of todo ≠ {} and pair1: pair = sel todo and pair2: (a,tod)
      = pair and nnmem: ¬ (∃ b ∈ fin. subsumes b a)
      from pair1 pair2 have pair: sel todo = (a,tod) by auto
      from nnmem subsumes-refl[of a] have nnmem: 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^* ` (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 nnmem at by auto
            show ?l ⊆ ?r
            proof
show \( \not \subseteq \not \) using ar by auto

next
have \( R^* \subseteq \) set-of \( (r a) \) by auto
proof
fix \( b \)
assume \( b \in R^* \subseteq \) set-of \( (r a) \)
then obtain \( c \) where \( cb: (c, b) \in R^* \) 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 at show \( b \in R^* \subseteq \) set-of todo by auto
qed
thus \( \subseteq \not \subseteq \not \) using sel unfolding \( \not \) by auto
qed
qed
qed

declare \( mk-rtrancl-main.simps[simp del] \)

lemma \( mk-rtrancl-main-sound: set-of todo \cup fin \subseteq R^* \Rightarrow mk-rtrancl-main \)
todo fin \( \subseteq R^* \)
proof (induct todo fin rule: mk-rtrancl-main.induct)
case (1 todo fin)
note simp = mk-rtrancl-main.simps[of todo fin]
show \( \) proof
(cases set-of todo = \{\})
case True
show \( ?thesis \) unfolding simp using True 1(3) by auto
next
case False
hence \( \) nempty: \( (set-of todo = \{\}) = False \) by auto
obtain \( a \) tod where \( sel \) todo = \( (a, tod) \) 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 \( \exists \ b \in fin.\) subsumes \( b \ a)\)
case True
hence \( \) mk-rtrancl-main todo fin = mk-rtrancl-main tod fin
unfolding simp by simp
with IH1[OF True] 1(3) show \( ?thesis \) using sel by auto
next
case 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])
from sel 1(3) have subset: set-of todo \cup insert a fin \subseteq R^* \text{ "init by auto}

\{ 
  fix b 
  assume b: b \in set-of (r a) 
  hence ab: (a,b) \in R \text{ by auto} 
  from sel 1(3) have a \in R^* \text{ "init by auto} 
  then obtain c where c: c \in init and ca: (c,a) \in R^* O R \text{ by auto} 
  have (c,b) \in R^* 
    by (rule set-mp[OF - cb], reexp) 
    with c have b \in R^* \text{ "init by auto} 
  \}

with subset 
show set-of (un (r a) tod) \cup (insert a fin) \subseteq R^* \text{ "init} 
unfolding un using sel by auto 
qed 
qed 
qed 

|lemma| mk-rtrancl-main-complete: |
|\[ |[ \forall a. a \in \text{init} \implies \exists b. b \in \text{set-of todo} \cup \text{fin} \wedge \text{subsumes b a} | |
|\implies | \[ \forall a b. a \in \text{fin} \implies b \in \text{set-of (r a)} \implies \exists c. c \in \text{set-of todo} \cup \text{fin} \wedge \text{subsumes c b} \] | |
|\implies | c \in R^* \text{ "init} |
|\implies | \exists b. b \in mk-rtrancl-main todo fin \wedge \text{subsumes b c} |

|proof| (induct todo fin rule: mk-rtrancl-main.induct) |
|case| (1 todo fin) |
|from| 1(5) have c: c \in R^* \text{ "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 \in init and ac: (a,c) \in R^* \text{ 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 \in fin and db: subsumes d b by auto |
|from| step(2) have cb: c \in set-of (r b) by auto |
|from| subsumes-step[OF db cb] obtain a where a: a \in set-of (r d) \text{ and ac: subsumes a c by auto} |
|from| finr[OF d a] obtain e where e: e \in fin and ea: subsumes e a unfolding True by auto |
from subsumes-trans[of ca] e
show ?case by auto
qed

next
case False

hence nempty: (set-of todo = {}) = False by simp
obtain A tod where sel: sel todo = (A,tod) by force

note simp = nempty simp if-False Let-def sel
note sel = sel[of False sel]

note IH1 = 1(1)[OF False refl sel]
note IH2 = 1(2)[OF False refl sel]

show ?thesis
proof (cases ∃ b ∈ fin. subsumes b A)
case True note oTrue = this

hence id: mk-rtrancl-main todo fin = mk-rtrancl-main tod fin

unfolding simp by simp
from True obtain b where b: b ∈ fin and ba: subsumes b A by auto
show ?thesis unfolding id

proof (rule IH1[of True])

fix a
assume a: a ∈ init
from init[of a] obtain c where c: c ∈ set-of todo ∪ fin and ca: subsumes c a

proof (cases c = A)
case False

thus ?thesis using c ca sel by auto

next
case True


qed

next

fix a c
assume a: a ∈ fin and c: c ∈ set-of (r a)
from finr[of a c] obtain e where e: e ∈ set-of todo ∪ fin and ec: subsumes e c

proof (cases A = e)
case False

with e ec show ?thesis using sel by auto

next
case True

from subsumes-trans[of ba[unfolded True] ec]

show ?thesis using b by auto

qed

next

case 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.

\[
\text{partial-function\,(\text{tailrec}) mk\text{-}rtrancl\text{-}list\text{-}main\ where}
\]

\[
\begin{align*}
\text{[code]: mk\text{-}rtrancl\text{-}list\text{-}main\ subsumes\ r\ todo\ fin } &= \text{ (case\ todo\ of\ } [] \Rightarrow \text{ fin} \notag \\
\text{ &\quad |\ \text{Cons } a\ \text{tod }\Rightarrow 
\begin{cases}
\text{ if } (\exists\ b\ \in\ \text{set\ fin}.\ subsumes\ b\ a)\ \text{then mk\text{-}rtrancl\text{-}list\text{-}main\ subsumes\ r}
\text{ tod\ fin} \\
\text{ else mk\text{-}rtrancl\text{-}list\text{-}main\ subsumes\ r\ (r\ a\ @\ tod)\ (a\ \#\ fin)}
\end{cases}
\end{align*}
\]

\[
\text{definition mk\text{-}rtrancl\text{-}list\ where}
\]

\[
\begin{align*}
\text{mk\text{-}rtrancl\text{-}list\ subsumes\ r\ init } &= \text{ mk\text{-}rtrancl\text{-}list\text{-}main\ subsumes\ r\ init} \
\text{[]}
\end{align*}
\]

\[
\text{locale subsumption\text{-}list } = \text{ subsumption } r\ \text{subsumes set}
\]

\[
\text{for } r::'a \Rightarrow 'a\ \text{list} \text{ and subsumes }::'a \Rightarrow 'a \Rightarrow \text{bool}
\]

\[
\text{locale relation\text{-}subsumption\text{-}list } = \text{ subsumption\text{-}list } r\ \text{subsumes for } r\ \text{subsumes +}
\]

\[
\text{assumes rtrancl\text{-}fin: } \bigwedge\ a.\ \text{finite } \{ b.\ (a,b)\ \in\ \{ (a,b)\ \middle|\ b\ \in\ \text{set } (r\ a)\} \}^*\]

\[
\text{abbreviation (input) sel\text{-}list\ where sel\text{-}list\ x } = \text{ case\ x\ of\ Cons\ h\ t } \Rightarrow (h,t)
\]

\[
\text{sublocale subsumption\text{-}list } \subseteq \text{ subsumption\text{-}impl } r\ \text{subsumes set sel\text{-}list append length}
\]

\[
\text{proof (unfold\text{-}locales,\ rule finite\text{-}set)}
\]

\[
\text{fix } b\ a\ c
\]

\[
\text{assume } \text{set } b \neq \{\} \text{ and sel\text{-}list } b = (a,c)
\]

\[
\text{thus } \text{set } b = \text{ insert } a\ (\text{set } c) \wedge \text{length } c < \text{length } b
\]

\[
\text{by (cases } b,\ \text{auto)}
\]

\[
\text{qed auto}
\]

\[
\text{sublocale relation\text{-}subsumption\text{-}list } \subseteq \text{ relation\text{-}subsumption\text{-}impl } r\ \text{subsumes set sel\text{-}list append length}
\]

\[
\text{by (unfold\text{-}locales,\ rule rtrancl\text{-}fin)}
\]

\[
\text{context relation\text{-}subsumption\text{-}list}
\]

\[
\text{begin}
\]

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

\[
\text{lemma mk\text{-}rtrancl\text{-}list\text{-}main: fin } = \text{ set } \text{finl } \Rightarrow \text{ set } (mk\text{-}rtrancl\text{-}list\text{-}main\ subsumes r\ todo\ finl) = \text{ mk\text{-}rtrancl\text{-}main\ todo}\ finl
\]

\[
\text{proof (induct todo\ finl\ arbitrary: f\ inl\ rule: mk\text{-}rtrancl\text{-}main\ induct)}
\]

\[
\text{case } (1\ \text{todo}\ \text{finl})
\]

\[
\text{note simp } = \text{ mk\text{-}rtrancl\text{-}list\text{-}main\ simps[of - todo\ finl]} \text{ mk\text{-}rtrancl\text{-}main\ simps[of todo\ finl]}
\]

\[
\text{show } \text{?case (is } \text{?l } = \text{?r)}
\]
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)
end

References
