Executable Transitive Closures

René Thiemann

August 28, 2014

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\textsuperscript{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
  1.2 Instantiation 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 $r^*$. In the following, we develop a similar, though different work-list algorithm for reflexive-transitive closures, it computes $r^* |init$ for a given relation $r$ and finite set $init$. The main differences are that

\textsuperscript{1}Supported by FWF (Austrian Science Fund) project P22767-N13.

\textsuperscript{1}http://cl-informatik.uibk.ac.at/software/ceta
• The relation \( r \) does not have to be finite, only \( \{ b \mid (a, b) \in r^* \} \) 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 \textit{size} function. The selection function \textit{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 \textit{un}.

\textbf{locale} subsumption =
fixes \( r :: 'a \Rightarrow 'b \)
and \( \text{subsumes} :: 'a \Rightarrow 'a \Rightarrow \text{bool} \)
and \( \text{set-of} :: 'b \Rightarrow 'a \text{ set} \)
assumes
\( \text{subsumes-refl}: \bigwedge a. \text{subsumes} a a \)
and \( \text{subsumes-trans}: \bigwedge a b c. \text{subsumes} a b \Longrightarrow \text{subsumes} b c \Longrightarrow \text{subsumes} a c \)
and \( \text{subsumes-step}: \bigwedge a b c. \text{subsumes} a b \Longrightarrow c \in \text{set-of} (r b) \Longrightarrow \exists d \in \text{set-of} (r a). \text{subsumes} d c \)

locale \text{subsumption-impl} = subsumption r \text{ subsumes set-of for}
\( r :: 'a \Rightarrow 'b \)
and \( \text{subsumes} :: 'a \Rightarrow 'a \Rightarrow \text{bool} \)
and \( \text{set-of} :: 'b \Rightarrow 'a \text{ set} \)
fixes
\( \text{sel} :: 'b \Rightarrow 'a \times 'b \)
and \( \text{un} :: 'b \Rightarrow 'b \Rightarrow 'b \)
and \( \text{size} :: 'b \Rightarrow \text{nat} \)
assumes
\( \text{set-of-fin}: \bigwedge b. \text{finite} (\text{set-of} b) \)
and \( \text{sel}: \bigwedge b a c. \text{set-of} b \neq \{\} \Longrightarrow \text{sel} b = (a,c) \Longrightarrow \text{set-of} b = \text{insert} a (\text{set-of} c) \land \text{size} b \geq \text{size} c \)
and \( \text{un}: \text{set-of} (\text{un} a b) = \text{set-of} a \cup \text{set-of} b \)

locale \text{relation-subsumption-impl} = \text{subsumption-impl} r \text{ subsumes set-of sel un size}
for \( r \) \text{ subsumes set-of sel un size}
assumes
\( \text{rtrancl-fin}: \bigwedge a. \text{finite} \{ (a,b). b \in \text{set-of} (r a) \} \)

begin

lemma \text{finite-Rs}: assumes \( \text{init}: \text{finite init} \)
shows \( \text{finite} (R^* \cdot \text{"init}) \)
proof
let \( \forall R = \lambda a. \{ b . (a,b) \in R^* \} \)
let \( \forall S = \{ \forall R a | a . a \in \text{init} \} \)
have \( \forall \text{init} = \bigcup \forall S \) by auto
show \( \text{thesis} \) unfolding \( \text{id} \)
proof (rule)
fix \( M \)
assume \( M \in \forall S \)
then obtain \( a \) where \( M = ?R a \) by auto
show \( \text{finite} M \) unfolding \( M \) by (rule \text{rtrancl-fin})
next
show \( \text{finite} \{ (b. (a,b) \in R^* | a . a \in \text{init}) \} \)
using \( \text{init} \) 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-less[of 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 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 nmem: ¬ (∃ b ∈ 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 OF finite-Diff[of 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 \( \emptyset \neq \emptyset \) using \texttt{ar} by \texttt{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 \( ca: c \in\) set-of \((r a)\) by blast
have \( (a, b) \in R^* \)
by (rule set-mp[OF - \texttt{ab}], regexp)
with \texttt{at} show \( b \in R^* \) set-of todo by \texttt{auto}
qed
thus \( \emptyset \subseteq \emptyset \) using \texttt{sel} unfolding \texttt{un} by \texttt{auto}
qed
qed
qed

declare \texttt{mk-rtrancl-main.simps[\texttt{simp del}]}

lemma \texttt{mk-rtrancl-main-sound}: set-of todo \(\cup\) fin \(\subseteq\) \( R^* \) init \(\Rightarrow\) \texttt{mk-rtrancl-main}
proof (induct todo fin rule: \texttt{mk-rtrancl-main.induct})
case (1 todo fin)
  note \texttt{simp} = \texttt{mk-rtrancl-main.simps[of todo fin]}
  show \(?case\)
  proof (cases set-of todo = \{\})
    case True
    show \(?thesis\) unfolding \texttt{simp} using True \(3\) by \texttt{auto}
  next
    case False
    hence \texttt{nempty}: \((\texttt{set-of todo} = \{\}) = \texttt{False}\) by \texttt{auto}
    obtain \( a \) tod where \texttt{sel}: \(\texttt{sel todo} = (a, \texttt{tod})\) by \texttt{force}
    note \texttt{sel} = \texttt{sel[OF \texttt{False sel}]}
    note \texttt{IH1} = \texttt{1(1)[OF False refl sel[\texttt{symmetric}]]}
    note \texttt{IH2} = \texttt{1(2)[OF False refl sel[\texttt{symmetric}]]}
    note \texttt{simp} = \texttt{simp nempty if-False Let-def sel}
    show \(?thesis\)
    proof (cases \(\exists\) \( b \in\) fin. \texttt{subsumes b a)\)
      case True
      hence \texttt{mk-rtrancl-main todo fin} = \texttt{mk-rtrancl-main tod fin}
      unfolding \texttt{simp by simp}
      with \texttt{IH1[OF \texttt{OF True}]} \(1(3)\) show \(?thesis\) using \texttt{sel by auto}
    next
      case False
      hence \texttt{id}: \texttt{mk-rtrancl-main todo fin} = \texttt{mk-rtrancl-main (un (r a) tod) (insert a fin)}
      unfolding \texttt{simp by simp}
      show \(?thesis\) unfolding \texttt{id}
      proof (rule \texttt{IH2[OF \texttt{OF False}]}))
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 ∈ 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 subsumes-trans[OF ca ac] 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[ symmetric] - - - c]
note IH2 = 1(2)[OF False refl sel[ symmetric] - - - c]
show ?thesis
proof (cases ∃ b ∈ fin. subsumes b A)
case True note aTrue = 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 by blast
show ∃ b. b ∈ set-of tod ∪ fin ∧ subsumes b a
proof (cases c = A)
case False
thus ?thesis using c ca sel by auto
next
case True
show ?thesis using b subsumes-trans[OF ba, of a] ca unfolding True[symmetric] by auto
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 by auto
show ∃ d. d ∈ set-of tod ∪ fin ∧ subsumes d 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
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

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.

```plaintext
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 for 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 finl
  note simp = mk-rtrancl-list-main.simps[of - todo finl] mk-rtrancl-main.simps[of todo fin]
  show ?case (is ?l = ?r)
```

9
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
end

References

    for program termination. In Proc. POPL ’01, pages 81–92. ACM Press,