Executable Transitive Closures*

René Thiemann

April 20, 2020

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 project¹ [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.Regrep-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

---

*Supported by FWF (Austrian Science Fund) project P22767-N13.
¹http://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 $size$ 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 \( r :: \text{'a} \Rightarrow \text{'b} \)
and \( \text{subsumes} :: \text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool} \)
and \( \text{set-of} :: \text{'b} \Rightarrow \text{'a} \Rightarrow \text{bool} \)
and \( \text{set-of} :: \text{'b} \Rightarrow \text{'a} \Rightarrow \text{bool} \)

assumes

\[ \text{subsames-refl}: \bigwedge a. \text{subsames} \ a \ a \]
\[ \text{subsames-trans}: \bigwedge a \ b \ c. \ \text{subsames} \ a \ b \ \Longrightarrow \ \text{subsames} \ b \ c \ \Longrightarrow \ \text{subsames} \ a \ c \]
\[ \text{subsames-step}: \bigwedge a \ b \ c. \ \text{subsames} \ a \ b \ \Longrightarrow \ c \in \text{set-of} \ (r \ b) \ \Longrightarrow \ \exists \ d \in \text{set-of} \ (r \ a). \ \text{subsames} \ d \ c \]

begin

abbreviation \( R \) where
\[
R \equiv \{ (a, b) . b \in \text{set-of} \ (r \ a) \}
\]

end

locale \( \text{subsumption-impl} = \text{subsumption} \ r \ \text{subsumes} \ \text{set-of} \)
for \( r :: \text{'a} \Rightarrow \text{'b} \)
and \( \text{subsumes} :: \text{'a} \Rightarrow \text{'a} \Rightarrow \text{bool} \)
and \( \text{set-of} :: \text{'b} \Rightarrow \text{'a} \Rightarrow \text{bool} \)
and \( \text{set-of} :: \text{'b} \Rightarrow \text{'a} \Rightarrow \text{bool} \)

fixes

\[ \text{sel} :: \text{'b} \Rightarrow \text{'a} \times \text{'b} \]
\[ \text{un} :: \text{'b} \Rightarrow \text{'b} \Rightarrow \text{'b} \]
\[ \text{size} :: \text{'b} \Rightarrow \text{nat} \]

assumes

\[ \text{set-of-fin}: \bigwedge b. \ \text{finite} \ (\text{set-of} \ b) \]
\[ \text{sel}: \bigwedge b \ a \ c. \ \text{set-of} \ b \neq \{\} \ \Longrightarrow \ \text{set-of} \ b = (a, c) \ \Longrightarrow \ \text{set-of} \ b = \text{insert} \ a \ (\text{set-of} \ c) \]
\[ \land \ \text{size} \ b \geq \ \text{size} \ c \]
\[ \text{and} \ \text{un}: \ \text{set-of} \ (\text{un} \ a \ b) = \text{set-of} \ a \cup \text{set-of} \ b \]

locale \( \text{relation-subsumption-impl} = \text{subsumption-impl} \ r \ \text{subsumes} \ \text{set-of} \ \text{sel} \ \text{un} \ \text{size} \)
for \( r \ \text{subsumes} \ \text{set-of} \ \text{sel} \ \text{un} \ \text{size} \)

assumes \( \text{rtrancl-fin}: \bigwedge a. \ \text{finite} \ \{ (a, b) . b \in \text{set-of} \ (r \ a) \} \}

begin

lemma \( \text{finite-Rs}: \) assumes \( \text{init}: \ \text{finite} \ \text{init} \)
shows \( \text{finite} \ \ (R^* \ " \ \text{init} \) \)
proof –

let \( ?R = \lambda a. \ \{ b . (a, b) \in R^* \} \)

let \( ?S = \{ ?R \ a . a \in \text{init} \} \)

have \( \text{id}: R^* \ " \ \text{init} \ = \bigcup ?S \) by auto

show \( \text{thesis} \) unfolding \( \text{id} \)
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 \( \text{rtrancl-fin} \))

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 \text{mk-rtrancl-main} \ where
\text{mk-rtrancl-main} \todo \ fin = (\text{if} \ \text{set-of} \ todo = \{\} \ \text{then} \ fin
\text{else} \ \text{let} \ (a,\todo) = \text{sel} \ todo
\text{in} \ (\text{if} \ (\exists \ b \in \fin, \ \text{subsumes} \ b \ a) \ \text{then} \ \text{mk-rtrancl-main} \ \todo \ fin
\text{else} \ \text{mk-rtrancl-main} \ (\text{un} \ (r \ a) \ \todo) \ (\text{insert} \ a \ \fin))))
\text{by} \ \text{pat-completeness} \ \text{auto}

termination \text{mk-rtrancl-main}
\text{proof}
\text{let} \ \forall r1 = \lambda (todo, \fin). \ \text{card} \ (R^\ast \ {\text{``}} \ (\text{set-of} \ todo) \ - \ fin)\text{ by simp}
\text{let} \ \forall r2 = \lambda (todo, \fin). \ \text{size} \ todo
\text{show} \ \forall \text{thesis}
\text{proof}
\text{show} \ \text{wf} \ (\text{measures} \ [\forall r1,\forall r2]) \ \text{by} \ \text{simp}
\text{next}
\text{fix} \ todo \ fin \ pair \ tod \ a
\text{assume nempty: set-of todo} \neq \{\} \ \text{and} \ \text{pair1: pair = sel todo and} \ \text{pair2: (a,tod)}
\text{= pair}
\text{from} \ \text{pair1 pair2 have pair: sel todo = (a,tod) by simp}
\text{from} \ \text{set-of-fin have fin: finite (set-of todo) by auto}
\text{note sel = sel[(OF nempty pair]}
\text{show} \ (((\text{tod,fin}),(\text{todo,fin})) \in \text{measures} \ [\forall r1,\forall r2]}
\text{proof} \ \text{(rule measures-less}[OF - measures-less, \text{unfold split})}
\text{from} \ \text{sel}
\text{show size tod < size todo by simp}
\text{next}
\text{from} \ \text{sel have subset: R^\ast \ {\text{``}} \ (set-of tod - fin \subseteq R^\ast \ {\text{``}} \ set-of todo - fin (is}
\forall l \subseteq \forall r) \ \text{by auto}
\text{show card} \ \forall l \leq \text{card} \ \forall r
\text{by} \ \text{(rule card-mono}[OF - subset, \text{rule finite-Diff, rule finite-Rs}[OF fin])}
\text{qed}
\text{next}
\text{fix} \ todo \ fin \ a \ \text{a tod pair}
\text{assume nempty: set-of todo} \neq \{\} \ \text{and} \ \text{pair1: pair = sel todo and} \ \text{pair2: (a,tod)}
\text{= pair and nnmem: \neg (\exists \ b \in \fin, \ \text{subsumes} \ b \ a)}
\text{from} \ \text{pair1 pair2 have pair: sel todo = (a,tod) by auto}
\text{from} \ \text{nnmem subsumes-refl[of a] have nnmem: a \notin \fin by auto}
\text{from} \ \text{set-of-fin have fin: finite (set-of todo) by auto}
\text{note sel = sel[(OF nempty pair]}
\text{show} \ (((\text{un} \ (r \ a) \ \text{tod,insert} \ a \ \fin),(\text{todo,fin})) \in \text{measures} \ [\forall r1,\forall r2]}
\text{proof} \ \text{(rule measures-less, \text{unfold split),}
\text{ rule psubset-card-mono}[OF finite-Diff[of finite-Rs[of fin]]])}
\text{let} \ \forall l = R^\ast \ {\text{``}} \ \text{set-of} \ (\text{un} \ (r \ a) \ \text{tod}) - \ \text{insert} \ a \ \fin
\text{let} \ \forall r = R^\ast \ {\text{``}} \ \text{set-of todo - fin}
\text{from} \ \text{sel have at: a \in set-of todo by auto}
\text{have ar: a \in \forall r \ \text{using nnmem at by auto}
\text{show} \ \forall l \subseteq \forall r
\text{proof}
show \( \not \in \not r \) using ar by auto

next

have \( R^* \subseteq R^* \) using todo proof
  fix \( b \)
  assume \( b \in R^* \) then obtain \( c \) where \( cb : (c,b) \in R O R^* \) by auto
  have \( (a,b) \in R^* \) by (rule subsetD[OF - ab], regecp)
  with at show \( b \in R^* \) by auto

qed

thus \( \not \in \not r \) using sel unfolding un by auto

qed

qed

qed

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

lemma mk-rtrancl-main-sound: set-of todo fin \( \subseteq R^* \) init \( \Rightarrow \) mk-rtrancl-main todo fin \( \subseteq R^* \) init proof (induct todo fin rule: mk-rtrancl-main.induct)
  case (1 todo fin)
  note simp = mk-rtrancl-main.simps[of todo fin]
  show ?case proof (cases set-of todo = {})
    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
      proof (rule IH2[OF False])
from sel 1(3) have subset: set-of todo \cup insert a fin \subseteq R^* " init by auto
{
  fix b
  assume b: b \in set-of (r a)
  hence ab: (a,b) \in R by auto
  from sel 1(3) have a \in R^* " init by auto
  then obtain c where c \in init and ca: (c,a) \in R^* O R by auto
  have (c,b) \in R^*
    by (rule subsetD[OF - cb], regexp)
  with c have b \in R^* " init by auto
}
with subset
show set-of (un (r a) tod) \cup (insert a fin) \subseteq R^* " init
unfolding un using sel by auto
qed
qed
qed
lemma mk-rtrancl-main-complete:
[\land a. a \in init \implies \exists b. b \in set-of todo \cup fin \land subsumes b a]
\implies \exists a b. a \in fin \implies b \in set-of (r a) \implies \exists c. c \in set-of todo \cup fin \land
subsumes c b]
\implies c \in R^* " init
\implies \exists b. b \in mk-rtrancl-main todo fin \land subsumes b c
proof (induct todo fin rule: mk-rtrancl-main.induct)
case (1 todo fin)
  from 1(5) have c: c \in 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 \in init and ac: (a,c) \in 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 \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) 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 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
      proof (cases c = A)
        case False with e ec show ?thesis using 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
      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

7
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 an 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 = (="
  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.

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

\[\text{[code]}: \quad \text{mk-rtrancl-list-main subsumes } r \text{ todo fin} = (\text{case todo of } [] \Rightarrow \text{fin}
\]

\[\quad \text{| Cons a tod} \Rightarrow
\]

\[\quad \quad \text{(if } (\exists \ b \in \text{set fin. subsumes } b \ a) \text{ then mk-rtrancl-list-main subsumes } r
\]

\[\quad \text{tod fin}
\]

\[\quad \quad \text{else mk-rtrancl-list-main subsumes } r (r \ a \ @ \ tod) (a \ # \ fin)))
\]

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

\[\text{mk-rtrancl-list subsumes } r \text{ init} \equiv \text{mk-rtrancl-list-main subsumes } r \text{ init }[]
\]

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

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

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

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

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

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

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

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

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

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

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

\[\text{qed auto}
\]

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

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

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

\[
\text{The main equivalence proof between the generic work list algorithm and the one operating on lists}
\]

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

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

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

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

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