

Executable Transitive Closures*

René Thiemann

December 14, 2021

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.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

*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 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 mean to split a non-empty collection into one element and a remaining collection. The union on $'b$ is given by *un*.

locale *subsumption* =

```

fixes  $r :: 'a \Rightarrow 'b$ 
and  $subsumes :: 'a \Rightarrow 'a \Rightarrow bool$ 
and  $set-of :: 'b \Rightarrow 'a\ set$ 
assumes
   $subsumes-refl: \bigwedge a. subsumes\ a\ a$ 
and  $subsumes-trans: \bigwedge a\ b\ c. subsumes\ a\ b \implies subsumes\ b\ c \implies subsumes\ a\ c$ 
and  $subsumes-step: \bigwedge a\ b\ c. subsumes\ a\ b \implies c \in set-of\ (r\ b) \implies \exists d \in set-of$ 
 $(r\ a). subsumes\ d\ c$ 
begin
abbreviation  $R\ where\ R \equiv \{ (a,b). b \in set-of\ (r\ a) \}$ 
end

```

```

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

```

```

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

```

```

lemma  $finite-Rs: assumes\ init: finite\ init$ 
shows  $finite\ (R^* \text{ `` } init)$ 

```

```

proof –
let  $?R = \lambda a. \{b . (a,b) \in R^*\}$ 
let  $?S = \{ ?R\ a \mid a . a \in init \}$ 
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  $M: M = ?R\ a$  by  $auto$ 
  show  $finite\ M$  unfolding  $M$  by  $(rule\ rtrancl-fin)$ 
next
  show  $finite\ \{\{b. (a, b) \in R^*\} \mid a. a \in init\}$ 
  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 (∃ 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 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
  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 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 ≠ ?r using ar by auto
  next
  have  $R^{\hat{*}} \text{ “ set-of } (r a) \subseteq R^{\hat{*}} \text{ “ set-of todo}$ 
  proof
    fix b
    assume  $b \in R^{\hat{*}} \text{ “ set-of } (r a)$ 
    then obtain c where  $cb: (c,b) \in R^{\hat{*}}$  and  $ca: c \in \text{set-of } (r a)$  by blast
    hence  $ab: (a,b) \in R \ O \ R^{\hat{*}}$  by auto
    have  $(a,b) \in R^{\hat{*}}$ 
      by (rule subsetD[OF - ab], regexp)
    with at show  $b \in R^{\hat{*}} \text{ “ set-of todo}$  by auto
  qed
  thus  $?l \subseteq ?r$  using sel unfolding un by auto
  qed
  qed
  qed
  qed

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

lemma mk-rtrancl-main-sound:  $\text{set-of todo} \cup \text{fin} \subseteq R^{\hat{*}} \text{ “ init} \implies \text{mk-rtrancl-main}$ 
 $\text{todo fin} \subseteq R^{\hat{*}} \text{ “ init}$ 
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
    show ?thesis unfolding simp using True 1(3) by auto
  next
  case False
  hence empty:  $(\text{set-of todo} = \{\}) = \text{False}$  by auto
  obtain a tod where selt:  $\text{sel todo} = (a,tod)$  by force
  note sel = sel[OF False selt]
  note IH1 = 1(1)[OF False refl selt[symmetric]]
  note IH2 = 1(2)[OF False refl selt[symmetric]]
  note simp = simp empty if-False Let-def selt
  show ?thesis
  proof (cases  $\exists b \in \text{fin. subsumes } b a$ )
    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:  $\text{mk-rtrancl-main todo fin} = \text{mk-rtrancl-main } (un (r a) tod)$  (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  $\subseteq R^*$  “init by auto
{
  fix b
  assume b:  $b \in \text{set-of } (r\ a)$ 
  hence ab:  $(a,b) \in R$  by auto
  from sel 1(3) have a  $\in R^*$  “init by auto
  then obtain c where c:  $c \in \text{init}$  and ca:  $(c,a) \in R^*$  by blast
  from ca ab have cb:  $(c,b) \in R^* \circ R$  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  $(\text{un } (r\ a)\ \text{tod}) \cup (\text{insert } a\ \text{fin}) \subseteq R^*$  “init
  unfolding un using sel by auto
qed
qed
qed
qed

```

lemma mk-rtrancl-main-complete:

```

[[ $\bigwedge a. a \in \text{init} \implies \exists b. b \in \text{set-of } \text{todo} \cup \text{fin} \wedge \text{subsumes } b\ a$ ]]
 $\implies$  [[ $\bigwedge a\ b. a \in \text{fin} \implies b \in \text{set-of } (r\ a) \implies \exists c. c \in \text{set-of } \text{todo} \cup \text{fin} \wedge$ 
subsumes c b]]
 $\implies c \in R^*$  “init
 $\implies \exists b. b \in \text{mk-rtrancl-main } \text{todo } \text{fin} \wedge \text{subsumes } b\ c$ 

```

proof (induct todo fin rule: mk-rtrancl-main.induct)

case (1 todo fin)

from 1(5) **have** c: $c \in R^*$ “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: $a \in \text{init}$ **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: $d \in \text{fin}$ **and** db: subsumes d b **by** auto

from step(2) **have** cb: $c \in \text{set-of } (r\ b)$ **by** auto

from subsumes-step[OF db cb] **obtain** a **where** a: $a \in \text{set-of } (r\ d)$ **and** ac: subsumes a c **by** auto

from finr[OF d a] **obtain** e **where** e: $e \in \text{fin}$ **and** ea: subsumes e a **unfolding** True **by** auto

```

    from subsumes-trans[OF ea ac] e
    show ?case by auto
  qed
next
case False
hence nempty: (set-of todo = {}) = False by simp
obtain A tod where selt: sel todo = (A,tod) by force
note simp = nempty simp if-False Let-def selt
note sel = sel[OF False selt]
note IH1 = 1(1)[OF False refl selt[symmetric] - - - c]
note IH2 = 1(2)[OF False refl selt[symmetric] - - - c]
show ?thesis
proof (cases  $\exists b \in \text{fin}. \text{subsumes } b A$ )
  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:  $b \in \text{fin}$  and ba: subsumes b A by auto
  show ?thesis unfolding id
  proof (rule IH1[OF True])
    fix a
    assume a:  $a \in \text{init}$ 
    from init[OF a] obtain c where c:  $c \in \text{set-of todo} \cup \text{fin}$  and ca: subsumes
c a by blast
    show  $\exists b. b \in \text{set-of tod} \cup \text{fin} \wedge \text{subsumes } b a$ 
    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:  $a \in \text{fin}$  and c:  $c \in \text{set-of } (r a)$ 
  from finr[OF a c] obtain e where e:  $e \in \text{set-of todo} \cup \text{fin}$  and ec: subsumes
e c by auto
  show  $\exists d. d \in \text{set-of tod} \cup \text{fin} \wedge \text{subsumes } d c$ 
  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  $\exists b. b \in \text{set-of } (un (r A) (tod)) \cup \text{insert } A \text{ fin} \wedge \text{subsumes } b a$ 
    using sel unfolding un by auto
next
  fix a b
  assume a: a ∈ insert A fin and b: b ∈ set-of (r a)
  show  $\exists c. c \in \text{set-of } (un (r A) tod) \cup \text{insert } A \text{ fin} \wedge \text{subsumes } c b$ 
  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* \equiv *mk-rtrancl-main init {}*

lemma *mk-rtrancl-sound: mk-rtrancl init* $\subseteq R^*$ “ *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 $\exists b. b \in \text{mk-rtrancl } \text{init} \wedge \text{subsumes } b a$
unfolding *mk-rtrancl-def*
proof (rule *mk-rtrancl-main-complete[OF - - a]*)
fix *a*
assume *a: a ∈ set-of init*
thus $\exists b. b \in \text{set-of } \text{init} \cup \{\} \wedge \text{subsumes } b a$ **using** *subsumes-refl[of a] by blast*
qed *auto*

lemma *mk-rtrancl-no-subsumption: assumes subsumes = (=)*
shows *mk-rtrancl init = R^* “ set-of init*
using *mk-rtrancl-sound[of init] mk-rtrancl-complete[of - init] assms*
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`.

```
partial-function(tailrec) mk-rtrancl-list-main where
  [code]: mk-rtrancl-list-main subsumes r todo fin = (case todo of []  $\Rightarrow$  fin
    | Cons a tod  $\Rightarrow$ 
      (if ( $\exists$  b  $\in$  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  $\equiv$  mk-rtrancl-list-main subsumes r init []
```

```
locale subsumption-list = subsumption r subsumes set
  for r :: 'a  $\Rightarrow$  'a list and subsumes :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool
```

```
locale relation-subsumption-list = subsumption-list r subsumes for r subsumes +
  assumes rtrancl-fin:  $\bigwedge$  a. finite {b. (a,b)  $\in$  { (a,b) . b  $\in$  set (r a) }*}
```

```
abbreviation(input) sel-list where sel-list x  $\equiv$  case x of Cons h t  $\Rightarrow$  (h,t)
```

```
sublocale subsumption-list  $\subseteq$  subsumption-impl r subsumes set sel-list append length
```

```
proof(unfold-locales, rule finite-set)
  fix b a c
  assume set b  $\neq$  {} and sel-list b = (a,c)
  thus set b = insert a (set c)  $\wedge$  length c < length b
    by (cases b, auto)
qed auto
```

```
sublocale relation-subsumption-list  $\subseteq$  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  $\implies$  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)
```

```

proof (cases todo)
  case Nil
  show ?thesis unfolding simp unfolding Nil 1(3) by simp
next
  case (Cons a tod)
  show ?thesis
  proof (cases  $\exists b \in \text{fin. subsumes } b \ a$ )
    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

- [1] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL '01*, pages 81–92. ACM Press, 2001.
- [2] C. Sternagel and R. Thiemann. Executable Transitive Closures of Finite Relations. In *Archive of Formal Proofs*. <http://isa-afp.org/entries/Transitive-Closure.shtml>, Mar. 2011. Formalization.

- [3] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs'09*, volume 5674 of *LNCS*, pages 452–468, 2009.