Proof of the Schorr-Waite algorithm

Farhad Mehta and Tobias Nipkow

theory *SchorrWaite* = *Pointers*:

1 Machinery for the Schorr-Waite proof

constdefs
— Relations induced by a mapping
$rel :: ('a \Rightarrow 'a\ ref) \Rightarrow ('a \times 'a)\ set$
$rel\ m == \{(x,y).\ m\ x = Ref\ y\}$
$relS :: ('a \Rightarrow 'a\ ref)\ set \Rightarrow ('a \times 'a)\ set$
$relS\ M == (\bigcup\ m \in M.\ rel\ m)$
$addrs :: 'a\ ref\ set \Rightarrow 'a\ set$
$addrs\ P == \{a.\ Ref\ a \in P\}$
$reachable :: ('a \times 'a)\ set \Rightarrow 'a\ ref\ set \Rightarrow 'a\ set$
$reachable\ r\ P == (r^*\ ``\ addrs\ P)$

lemmas *rel-defs* = *relS-def rel-def*

— Rewrite rules for relations induced by a mapping

lemma *self-reachable*: $b \in B \Longrightarrow b \in R^*\ ``\ B$
apply *blast*
done

lemma *oneStep-reachable*: $b \in R``B \Longrightarrow b \in R^*\ ``\ B$
apply *blast*
done

lemma *still-reachable*: $[\![B \subseteq Ra^*``A;\ \forall\ (x,y) \in Rb-Ra.\ y \in (Ra^*``A)]\!] \Longrightarrow Rb^*\ ``\ B \subseteq Ra^*\ ``\ A$
apply (*clarsimp simp only*:*Image-iff intro*:*subsetI*)
apply (*erule rtrancl-induct*)
 apply *blast*
apply (*subgoal-tac* $(y,\ z) \in Ra \cup (Rb-Ra)$)
 apply (*erule UnE*)
 apply (*auto intro*:*rtrancl-into-rtrancl*)
apply *blast*
done

lemma *still-reachable-eq*: $[\![\ A \subseteq Rb^*``B;\ B \subseteq Ra^*``A;\ \forall\ (x,y) \in Ra-Rb.\ y \in (Rb^*``B);$

$\forall\ (x,y) \in Rb - Ra.\ y \in (Ra^* `` A)\rrbracket \Longrightarrow Ra^* `` A = Rb^* `` B$
apply (*rule equalityI*)
apply (*erule still-reachable* ,*assumption*)+
done

lemma *reachable-null*: *reachable mS* $\{Null\} = \{\}$
apply (*simp add*: *reachable-def addrs-def*)
done

lemma *reachable-empty*: *reachable mS* $\{\} = \{\}$
apply (*simp add*: *reachable-def addrs-def*)
done

lemma *reachable-union*: (*reachable mS aS* $\cup$ *reachable mS bS*) = *reachable mS* (*aS* $\cup$ *bS*)
apply (*simp add*: *reachable-def rel-defs addrs-def*)
apply *blast*
done

lemma *reachable-union-sym*: *reachable r* (*insert a aS*) = (r^* `` *addrs* $\{a\}$) $\cup$ *reachable r aS*
apply (*simp add*: *reachable-def rel-defs addrs-def*)
apply *blast*
done

lemma *rel-upd1*: $(a,b) \notin rel\ (r(q:=t)) \Longrightarrow (a,b) \in rel\ r \Longrightarrow a=q$
apply (*rule classical*)
apply (*simp add*:*rel-defs fun-upd-apply*)
done

lemma *rel-upd2*: $(a,b) \notin rel\ r \Longrightarrow (a,b) \in rel\ (r(q:=t)) \Longrightarrow a=q$
apply (*rule classical*)
apply (*simp add*:*rel-defs fun-upd-apply*)
done

constdefs
— Restriction of a relation
$restr :: ('a \times 'a)\ set \Rightarrow ('a \Rightarrow bool) \Rightarrow ('a \times 'a)\ set$ ((-/ | -) [50, 51] 50)
$restr\ r\ m == \{(x,y).\ (x,y) \in r \wedge \neg\ m\ x\}$

— Rewrite rules for the restriction of a relation

lemma *restr-identity*[*simp*]:
$(\forall x.\ \neg\ m\ x) \Longrightarrow (R\ |m) = R$
by (*auto simp add*:*restr-def*)

lemma *restr-rtrancl*[*simp*]: $\llbracket m\ l \rrbracket \Longrightarrow (R\ |\ m)^* `` \{l\} = \{l\}$
by (*auto simp add*:*restr-def elim*:*converse-rtranclE*)

lemma [*simp*]: $[\![m\ l]\!] \Longrightarrow (l,x) \in (R \mid m)^* = (l=x)$
by (*auto simp add*:*restr-def elim*:*converse-rtranclE*)

lemma *restr-upd*: $((rel\ (r\ (q := t)))|(m(q := True))) = ((rel\ (r))|(m(q := True)))$

apply (*auto simp*:*restr-def rel-def fun-upd-apply*)
apply (*rename-tac a b*)
apply (*case-tac a*=*q*)
apply *auto*
done

lemma *restr-un*: $((r \cup s)|m) = (r|m) \cup (s|m)$
by (*auto simp add*:*restr-def*)

lemma *rel-upd3*: $(a,\ b) \notin (r|(m(q := t))) \Longrightarrow (a,b) \in (r|m) \Longrightarrow a = q$
apply (*rule classical*)
apply (*simp add*:*restr-def fun-upd-apply*)
done

constdefs
— A short form for the stack mapping function for List
$S :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a\ ref) \Rightarrow ('a \Rightarrow 'a\ ref) \Rightarrow ('a \Rightarrow 'a\ ref)$
$S\ c\ l\ r == (\lambda x.\ if\ c\ x\ then\ r\ x\ else\ l\ x)$

— Rewrite rules for Lists using S as their mapping

lemma [*rule-format*,*simp*]:
$\forall p.\ a \notin set\ stack \longrightarrow List\ (S\ c\ l\ r)\ p\ stack = List\ (S\ (c(a:=x))\ (l(a:=y))\ (r(a:=z)))\ p\ stack$
apply(*induct-tac stack*)
apply(*simp add*:*fun-upd-apply S-def*)+
done

lemma [*rule-format*,*simp*]:
$\forall p.\ a \notin set\ stack \longrightarrow List\ (S\ c\ l\ (r(a:=z)))\ p\ stack = List\ (S\ c\ l\ r)\ p\ stack$
apply(*induct-tac stack*)
apply(*simp add*:*fun-upd-apply S-def*)+
done

lemma [*rule-format*,*simp*]:
$\forall p.\ a \notin set\ stack \longrightarrow List\ (S\ c\ (l(a:=z))\ r)\ p\ stack = List\ (S\ c\ l\ r)\ p\ stack$
apply(*induct-tac stack*)
apply(*simp add*:*fun-upd-apply S-def*)+
done

lemma [*rule-format*,*simp*]:
$\forall p.\ a \notin set\ stack \longrightarrow List\ (S\ (c(a:=z))\ l\ r)\ p\ stack = List\ (S\ c\ l\ r)\ p\ stack$
apply(*induct-tac stack*)
apply(*simp add*:*fun-upd-apply S-def*)+

done

consts
— Recursive definition of what is means for a the graph/stack structure to be reconstructible
$stkOk :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a\ ref) \Rightarrow ('a \Rightarrow 'a\ ref) \Rightarrow ('a \Rightarrow 'a\ ref) \Rightarrow ('a \Rightarrow 'a\ ref) \Rightarrow 'a\ ref \Rightarrow 'a\ list \Rightarrow\ bool$
primrec
stkOk-nil: $stkOk\ c\ l\ r\ iL\ iR\ t\ [] = True$
stkOk-cons: $stkOk\ c\ l\ r\ iL\ iR\ t\ (p\#stk) = (stkOk\ c\ l\ r\ iL\ iR\ (Ref\ p)\ (stk) \wedge$
$iL\ p = (if\ c\ p\ then\ l\ p\ else\ t) \wedge$
$iR\ p = (if\ c\ p\ then\ t\ else\ r\ p))$

— Rewrite rules for stkOk

lemma [*simp*]: $\bigwedge t.\ \llbracket\ x \notin set\ xs;\ Ref\ x{\neq}t\ \rrbracket \Longrightarrow$
$stkOk\ (c(x := f))\ l\ r\ iL\ iR\ t\ xs = stkOk\ c\ l\ r\ iL\ iR\ t\ xs$
apply (*induct xs*)
apply (*auto simp*:*eq-sym-conv*)
done

lemma [*simp*]: $\bigwedge t.\ \llbracket\ x \notin set\ xs;\ Ref\ x{\neq}t\ \rrbracket \Longrightarrow$
$stkOk\ c\ (l(x := g))\ r\ iL\ iR\ t\ xs = stkOk\ c\ l\ r\ iL\ iR\ t\ xs$
apply (*induct xs*)
apply (*auto simp*:*eq-sym-conv*)
done

lemma [*simp*]: $\bigwedge t.\ \llbracket\ x \notin set\ xs;\ Ref\ x{\neq}t\ \rrbracket \Longrightarrow$
$stkOk\ c\ l\ (r(x := g))\ iL\ iR\ t\ xs = stkOk\ c\ l\ r\ iL\ iR\ t\ xs$
apply (*induct xs*)
apply (*auto simp*:*eq-sym-conv*)
done

lemma *stkOk-r-rewrite* [*simp*]: $\bigwedge x.\ x \notin set\ xs \Longrightarrow$
$stkOk\ c\ l\ (r(x := g))\ iL\ iR\ (Ref\ x)\ xs = stkOk\ c\ l\ r\ iL\ iR\ (Ref\ x)\ xs$
apply (*induct xs*)
apply (*auto simp*:*eq-sym-conv*)
done

lemma [*simp*]: $\bigwedge x.\ x \notin set\ xs \Longrightarrow$
$stkOk\ c\ (l(x := g))\ r\ iL\ iR\ (Ref\ x)\ xs = stkOk\ c\ l\ r\ iL\ iR\ (Ref\ x)\ xs$
apply (*induct xs*)
apply (*auto simp*:*eq-sym-conv*)
done

lemma [*simp*]: $\bigwedge x.\ x \notin set\ xs \Longrightarrow$
$stkOk\ (c(x := g))\ l\ r\ iL\ iR\ (Ref\ x)\ xs = stkOk\ c\ l\ r\ iL\ iR\ (Ref\ x)\ xs$
apply (*induct xs*)
apply (*auto simp*:*eq-sym-conv*)

done

2 The Schorr-Waite algorithm

```
theorem SchorrWaiteAlgorithm:
VARS c m l r t p q root
 {R = reachable (relS {l, r}) {root} ∧ (∀x. ¬ m x) ∧ iR = r ∧ iL = l}
 t := root; p := Null;
 WHILE p ≠ Null ∨ t ≠ Null ∧ ¬ t^.m
 INV {∃stack.
         List (S c l r) p stack ∧                                   −− (i1)
         (∀x ∈ set stack. m x) ∧                                    −− (i2)
         R = reachable (relS{l, r}) {t,p} ∧                         −− (i3)
         (∀x. x ∈ R ∧ ¬m x −→                                       −− (i4)
               x ∈ reachable (relS{l,r}|m) ({t}∪set(map r stack))) ∧
         (∀x. m x −→ x ∈ R) ∧                                       −− (i5)
         (∀x. x ∉ set stack −→ r x = iR x ∧ l x = iL x) ∧           −− (i6)
         (stkOk c l r iL iR t stack)                               −−− (i7) }
 DO IF t = Null ∨ t^.m
      THEN IF p^.c
             THEN q := t; t := p; p := p^.r; t^.r := q              −−− pop
             ELSE q := t; t := p^.r; p^.r := p^.l;                  −− swing
                    p^.l := q; p^.c := True          FI
      ELSE q := p; p := t; t := t^.l; p^.l := q;                    −− push
             p^.m := True; p^.c := False             FI       OD
{(∀x. (x ∈ R) = m x) ∧ (r = iR ∧ l = iL) }
 (is VARS c m l r t p q root {?Pre c m l r root} (?c1; ?c2; ?c3) {?Post c m l r})
```

```
proof (vcg)
  let While {(c, m, l, r, t, p, q, root). ?whileB m t p}
    {(c, m, l, r, t, p, q, root). ?inv c m l r t p} ?body = ?c3
  {
    — Precondition leads to Invariant:
    fix c m l r t p q root
    assume ?Pre c m l r root
    thus ?inv c m l r root Null  by (auto simp add: reachable-def addrs-def)
  next

    — Postcondition follows from Invariant termination:
    fix c m l r t p q
    let ∃stack. ?Inv stack  =  ?inv c m l r t p
    assume a: ?inv c m l r t p ∧ ¬(p ≠ Null ∨ t ≠ Null ∧ ¬ t^.m)
    then obtain stack where inv: ?Inv stack by blast
    from a have pNull: p = Null and tDisj: t=Null ∨ (t≠Null ∧ t^.m ) by auto
    let ?I1 ∧ - ∧ - ∧ ?I4 ∧ ?I5 ∧ ?I6 ∧ -  =  ?Inv stack
    from inv have i1: ?I1 and i4: ?I4 and i5: ?I5 and i6: ?I6 by simp+
    from pNull i1 have stackEmpty: stack = [] by simp
    from tDisj i4 have RisMarked[rule-format]: ∀x.  x ∈ R −→ m x
```

by(*auto simp*: *reachable-def addrs-def stackEmpty*)
from *i5 i6* **show** $(\forall x.(x \in R) = m\ x) \wedge r = iR \wedge l = iL$
by(*auto simp*: *stackEmpty expand-fun-eq intro*:*RisMarked*)
next

— Invariant is preserved
fix *c m l r t p q root*
let $\exists$ *stack*. *?Inv stack* = *?inv c m l r t p*
let $\exists$ *stack*. *?popInv stack* = *?inv c m l* $(r(p \rightarrow t))$ *p* $(p\hat{\ }.r)$
let $\exists$ *stack*. *?swInv stack* =
?inv $(c(p \rightarrow True))$ *m* $(l(p \rightarrow t))$ $(r(p \rightarrow p\hat{\ }.l))$ $(p\hat{\ }.r)$ *p*
let $\exists$ *stack*. *?puInv stack* =
?inv $(c(t \rightarrow False))$ $(m(t \rightarrow True))$ $(l(t \rightarrow p))$ *r* $(t\hat{\ }.l)$ *t*
let *?ifB1* = $(t = Null \vee t\hat{\ }.m)$
let *?ifB2* = $p\hat{\ }.c$

assume $(\exists$ *stack*.*?Inv stack*$) \wedge (p \neq Null \vee t \neq Null \wedge \neg\, t\hat{\ }.m)$ (**is** - $\wedge$ *?whileB*)
then obtain *stack* **where** *inv*: *?Inv stack* **and** *whileB*: *?whileB* **by** *blast*
let *?I1* $\wedge$ *?I2* $\wedge$ *?I3* $\wedge$ *?I4* $\wedge$ *?I5* $\wedge$ *?I6* $\wedge$ *?I7* = *?Inv stack*
from *inv* **have** *i1*: *?I1* **and** *i2*: *?I2* **and** *i3*: *?I3* **and** *i4*: *?I4*
and *i5*: *?I5* **and** *i6*: *?I6* **and** *i7*: *?I7* **by** *simp*+
have *stackDist*: *distinct* (*stack*) **using** *i1* **by** (*rule List-distinct*)

show (*?ifB1* $\longrightarrow$ (*?ifB2* $\longrightarrow$ ($\exists$ *stack*.*?popInv stack*)) $\wedge$
($\neg$*?ifB2* $\longrightarrow$ ($\exists$ *stack*.*?swInv stack*))) $\wedge$
($\neg$*?ifB1* $\longrightarrow$ ($\exists$ *stack*.*?puInv stack*))
proof −
{
assume *ifB1*: $t = Null \vee t\hat{\ }.m$ **and** *ifB2*: $p\hat{\ }.c$
from *ifB1 whileB* **have** *pNotNull*: $p \neq Null$ **by** *auto*
then obtain *addr-p* **where** *addr-p-eq*: *p* = *Ref addr-p* **by** *auto*
with *i1* **obtain** *stack-tl* **where** *stack-eq*: *stack* = (*addr p*) # *stack-tl*
by *auto*
with *i2* **have** *m-addr-p*: $p\hat{\ }.m$ **by** *auto*
have *stackDist*: *distinct* (*stack*) **using** *i1* **by** (*rule List-distinct*)
from *stack-eq stackDist* **have** *p-notin-stack-tl*: *addr p* $\notin$ *set stack-tl* **by** *simp*
let *?poI1*$\wedge$ *?poI2*$\wedge$ *?poI3*$\wedge$ *?poI4*$\wedge$ *?poI5*$\wedge$ *?poI6*$\wedge$ *?poI7* = *?popInv stack-tl*
have *?popInv stack-tl*
proof −

— List property is maintained:
from *i1 p-notin-stack-tl ifB2*
have *poI1*: *List* (*S c l* $(r(p \rightarrow t))$) $(p\hat{\ }.r)$ *stack-tl*
by(*simp add*: *addr-p-eq stack-eq*, *simp add*: *S-def*)

moreover
— Everything on the stack is marked:
from *i2* **have** *poI2*: $\forall$ *x* $\in$ *set stack-tl*. *m x* **by** (*simp add*:*stack-eq*)
moreover

— Everything is still reachable:
let (R = *reachable ?Ra ?A*) = *?I3*
let *?Rb* = (*relS* {*l*, *r*($p \to t$)})
let *?B* = {*p*, *p*^.*r*}
— Our goal is R = *reachable ?Rb ?B*.
have *?Ra** ‘‘ *addrs ?A* = *?Rb** ‘‘ *addrs ?B* (**is** *?L* = *?R*)
proof
  show *?L* ⊆ *?R*
  proof (*rule still-reachable*)
    show *addrs ?A* ⊆ *?Rb** ‘‘ *addrs ?B*
      by(*fastsimp simp*:*addrs-def relS-def rel-def addr-p-eq*
      intro:*oneStep-reachable Image-iff*[*THEN iffD2*])
    show ∀ (*x*,*y*) ∈ *?Ra*−*?Rb*. *y* ∈ (*?Rb** ‘‘ *addrs ?B*)
      by (*clarsimp simp*:*relS-def*)
        (*fastsimp simp add*:*rel-def Image-iff addrs-def dest*:*rel-upd1*)
  qed
  show *?R* ⊆ *?L*
  proof (*rule still-reachable*)
    show *addrs ?B* ⊆ *?Ra** ‘‘ *addrs ?A*
      by(*fastsimp simp*:*addrs-def rel-defs addr-p-eq*
        intro:*oneStep-reachable Image-iff*[*THEN iffD2*])
  next
    show ∀ (*x*, *y*)∈*?Rb*−*?Ra*. *y*∈(*?Ra**‘‘*addrs ?A*)
      by (*clarsimp simp*:*relS-def*)
        (*fastsimp simp add*:*rel-def Image-iff addrs-def dest*:*rel-upd2*)
  qed
qed
with *i3* **have** *poI3*: R = *reachable ?Rb ?B* **by** (*simp add*:*reachable-def*)
moreover

— If it is reachable and not marked, it is still reachable using...
let ∀ *x*. *x* ∈ *R* ∧ ¬ *m x* ⟶ *x* ∈ *reachable ?Ra ?A* = *?I4*
let *?Rb* = *relS* {*l*, *r*($p \to t$)} | *m*
let *?B* = {*p*} ∪ *set* (*map* (*r*($p \to t$)) *stack-tl*)
  — Our goal is ∀ *x*. *x* ∈ *R* ∧ ¬ *m x* ⟶ *x* ∈ *reachable ?Rb ?B*.
let *?T* = {*t*, *p*^.*r*}

have *?Ra** ‘‘ *addrs ?A* ⊆ *?Rb** ‘‘ (*addrs ?B* ∪ *addrs ?T*)
proof (*rule still-reachable*)
  have *rewrite*: ∀ *s*∈*set stack-tl*. (*r*($p \to t$)) *s* = *r s*
    by (*auto simp add*:*p-notin-stack-tl intro*:*fun-upd-other*)
  show *addrs ?A* ⊆ *?Rb** ‘‘ (*addrs ?B* ∪ *addrs ?T*)
    by (*fastsimp cong*:*map-cong*
      simp:*stack-eq addrs-def rewrite intro*:*self-reachable*)
  show ∀ (*x*, *y*)∈*?Ra*−*?Rb*. *y*∈(*?Rb**‘‘(*addrs ?B* ∪ *addrs ?T*))
    by (*clarsimp simp*:*restr-def relS-def*)
  (*fastsimp simp add*:*rel-def Image-iff addrs-def dest*:*rel-upd1*)
qed

— Bring a term from the right to the left of the subset relation.
hence *subset*: $?Ra^*$ ‘‘ *addrs* $?A$ − $?Rb^*$ ‘‘ *addrs* $?T \subseteq ?Rb^*$ ‘‘ *addrs* $?B$
by *blast*
have *poI4*: $\forall x.\ x \in R \wedge \neg\ m\ x \longrightarrow x \in$ *reachable* $?Rb\ ?B$
proof (*rule allI*, *rule impI*)
fix x
assume *a*: $x \in R \wedge \neg\ m\ x$
— First, a disjunction on $p\hat{}.r$ used later in the proof
have *pDisj*:$p\hat{}.r = Null \vee (p\hat{}.r \neq Null \wedge p\hat{}.r\hat{}.m)$ **using** *poI1 poI2*
by *auto*
— x belongs to the left hand side of *subset*:
have *incl*: $x \in\ ?Ra^*$‘‘*addrs* $?A$ **using** *a i4*
by (*simp only*:*reachable-def*, *clarsimp*)
have *excl*: $x \notin\ ?Rb^*$‘‘ *addrs* $?T$ **using** *pDisj ifB1 a*
by (*auto simp add*:*addrs-def*)
— And therefore also belongs to the right hand side of *subset*,
— which corresponds to our goal.
from *incl excl subset* **show** $x \in$ *reachable* $?Rb\ ?B$
by (*auto simp add*:*reachable-def*)
qed
moreover

— If it is marked, then it is reachable
from *i5* **have** *poI5*: $\forall x.\ m\ x \longrightarrow x \in R$ **.**
moreover

— If it is not on the stack, then its l and r fields are unchanged
from *i7 i6 ifB2*
have *poI6*: $\forall x.\ x \notin$ *set stack-tl* $\longrightarrow (r(p \rightarrow t))\ x = iR\ x \wedge l\ x = iL\ x$
by(*auto simp*: *addr-p-eq stack-eq fun-upd-apply*)

moreover

— If it is on the stack, then its l and r fields can be reconstructed
from *p-notin-stack-tl i7* **have** *poI7*: *stkOk* $c\ l\ (r(p \rightarrow t))\ iL\ iR\ p$ *stack-tl*
by (*clarsimp simp*:*stack-eq addr-p-eq*)

ultimately show *?popInv stack-tl* **by** *simp*
qed
hence $\exists$ *stack*. *?popInv stack* **..**
}
moreover

— Proofs of the Swing and Push arm follow.
— Since they are in principle simmilar to the Pop arm proof,
— we show fewer comments and use frequent pattern matching.
{
— Swing arm
assume *ifB1*: *?ifB1* **and** *nifB2*: $\neg$*?ifB2*

from *ifB1 whileB* **have** *pNotNull*: $p \neq Null$ **by** *clarsimp*
then obtain *addr-p* **where** *addr-p-eq*: $p = Ref\ addr\text{-}p$ **by** *clarsimp*
with *i1* **obtain** *stack-tl* **where** *stack-eq*: $stack = (addr\ p)\ \#\ stack\text{-}tl$
  by *clarsimp*
with *i2* **have** *m-addr-p*: $p\hat{\ }.m$ **by** *clarsimp*
from *stack-eq stackDist* **have** *p-notin-stack-tl*: $(addr\ p) \notin set\ stack\text{-}tl$
  by *simp*
let $?swI1 \wedge ?swI2 \wedge ?swI3 \wedge ?swI4 \wedge ?swI5 \wedge ?swI6 \wedge ?swI7 = ?swInv\ stack$
have *?swInv stack*
proof −

  — List property is maintained:
 from *i1 p-notin-stack-tl nifB2*
 have *swI1*: *?swI1*
  by (*simp add*:*addr-p-eq stack-eq*, *simp add*:*S-def*)
 moreover

  — Everything on the stack is marked:
 from *i2*
 have *swI2*: *?swI2* **.**
 moreover

  — Everything is still reachable:
 let $R = reachable\ ?Ra\ ?A = ?I3$
 let $R = reachable\ ?Rb\ ?B = ?swI3$
 have $?Ra^{*}$ `` $addrs\ ?A = ?Rb^{*}$ `` $addrs\ ?B$
 proof (*rule still-reachable-eq*)
  show $addrs\ ?A \subseteq ?Rb^{*}$ `` $addrs\ ?B$
   by(*fastsimp simp*:*addrs-def rel-defs addr-p-eq*
    intro:*oneStep-reachable Image-iff* [*THEN iffD2*])
 next
  show $addrs\ ?B \subseteq ?Ra^{*}$ `` $addrs\ ?A$
   by(*fastsimp simp*:*addrs-def rel-defs addr-p-eq*
    intro:*oneStep-reachable Image-iff* [*THEN iffD2*])
 next
  show $\forall (x, y) \in ?Ra - ?Rb.\ y \in (?Rb^{*}$ ``$addrs\ ?B)$
   by (*clarsimp simp*:*relS-def*)
 (*fastsimp simp add*:*rel-def Image-iff addrs-def fun-upd-apply dest*:*rel-upd1*)
 next
  show $\forall (x, y) \in ?Rb - ?Ra.\ y \in (?Ra^{*}$ ``$addrs\ ?A)$
   by (*clarsimp simp*:*relS-def*)
 (*fastsimp simp add*:*rel-def Image-iff addrs-def fun-upd-apply dest*:*rel-upd2*)
 qed
 with *i3*
 have *swI3*: *?swI3* **by** (*simp add*:*reachable-def*)
 moreover

  — If it is reachable and not marked, it is still reachable using...
 let $\forall x.\ x \in R \wedge \neg\ m\ x \longrightarrow x \in reachable\ ?Ra\ ?A = ?I4$

let $\forall x.\ x \in R \wedge \neg\ m\ x \longrightarrow x \in$ *reachable ?Rb ?B* $=$ *?swI4*
let *?T* $= \{t\}$
have *?Ra** ‘‘*addrs ?A* $\subseteq$ *?Rb** ‘‘(*addrs ?B* $\cup$ *addrs ?T*)
proof (*rule still-reachable*)
  have *rewrite*: ($\forall$ *s*$\in$*set stack-tl*. (*r*(*addr p* := *l*(*addr p*))) *s* = *r s*)
    by (*auto simp add*:*p-notin-stack-tl intro*:*fun-upd-other*)
  show *addrs ?A* $\subseteq$ *?Rb** ‘‘ (*addrs ?B* $\cup$ *addrs ?T*)
    by (*fastsimp cong*:*map-cong simp*:*stack-eq addrs-def rewrite*
      intro:*self-reachable*)
next
  show $\forall$ (*x*, *y*)$\in$*?Ra*$-$*?Rb*. *y*$\in$(*?Rb** ‘‘(*addrs ?B* $\cup$ *addrs ?T*))
    by (*clarsimp simp*:*relS-def restr-def*)
     (*fastsimp simp add*:*rel-def Image-iff addrs-def fun-upd-apply*
       dest:*rel-upd1*)
qed
then have *subset*: *?Ra** ‘‘*addrs ?A* $-$ *?Rb** ‘‘*addrs ?T* $\subseteq$ *?Rb** ‘‘*addrs ?B*
  by *blast*
have *?swI4*
proof (*rule allI*, *rule impI*)
  fix *x*
  assume *a*: $x \in R \wedge \neg\ m\ x$
  with *i4 addr-p-eq stack-eq* **have** *inc*: *x* $\in$ *?Ra** ‘‘*addrs ?A*
    by (*simp only*:*reachable-def*, *clarsimp*)
  with *ifB1 a*
  have *exc*: *x* $\notin$ *?Rb** ‘‘ *addrs ?T*
    by (*auto simp add*:*addrs-def*)
  from *inc exc subset* **show** *x* $\in$ *reachable ?Rb ?B*
    by (*auto simp add*:*reachable-def*)
qed
moreover

— If it is marked, then it is reachable
from *i5*
have *?swI5* **.**
moreover

— If it is not on the stack, then its l and r fields are unchanged
from *i6 stack-eq*
have *?swI6*
  by *clarsimp*
moreover

— If it is on the stack, then its l and r fields can be reconstructed
from *stackDist i7 nifB2*
have *?swI7*
  by (*clarsimp simp*:*addr-p-eq stack-eq*)

ultimately show *?thesis* **by** *auto*
qed

```
  then have ∃ stack. ?swInv stack by blast
}
moreover

{
  — Push arm
  assume nifB1: ¬?ifB1
  from nifB1 whileB have tNotNull: t ≠ Null by clarsimp
  then obtain addr-t where addr-t-eq: t = Ref addr-t by clarsimp
  with i1 obtain new-stack
    where new-stack-eq: new-stack = (addr t) # stack
    by clarsimp
  from tNotNull nifB1 have n-m-addr-t: ¬ (t^.m) by clarsimp
  with i2 have t-notin-stack: (addr t) ∉ set stack by blast
  let ?puI1∧?puI2∧?puI3∧?puI4∧?puI5∧?puI6∧?puI7 = ?puInv new-stack
  have ?puInv new-stack
  proof −

      — List property is maintained:
    from i1 t-notin-stack
    have puI1: ?puI1
      by (simp add:addr-t-eq new-stack-eq, simp add:S-def)
    moreover

      — Everything on the stack is marked:
    from i2
    have puI2: ?puI2
      by (simp add:new-stack-eq fun-upd-apply)
    moreover

      — Everything is still reachable:
    let R = reachable ?Ra ?A = ?I3
    let R = reachable ?Rb ?B = ?puI3
    have ?Ra* `` addrs ?A = ?Rb* `` addrs ?B
    proof (rule still-reachable-eq)
      show addrs ?A ⊆ ?Rb* `` addrs ?B
        by(fastsimp simp:addrs-def rel-defs addr-t-eq
          intro:oneStep-reachable Image-iff[THEN iffD2])
    next
      show addrs ?B ⊆ ?Ra* `` addrs ?A
        by(fastsimp simp:addrs-def rel-defs addr-t-eq
          intro:oneStep-reachable Image-iff[THEN iffD2])
    next
      show ∀(x, y)∈?Ra−?Rb. y∈(?Rb*``addrs ?B)
        by (clarsimp simp:relS-def)
        (fastsimp simp add:rel-def Image-iff addrs-def dest:rel-upd1)
    next
      show ∀(x, y)∈?Rb−?Ra. y∈(?Ra*``addrs ?A)
        by (clarsimp simp:relS-def)
```

(*fastsimp simp add*:*rel-def Image-iff addrs-def fun-upd-apply*
dest:*rel-upd2*)
qed
with *i3*
have *puI3*: *?puI3* **by** (*simp add*:*reachable-def*)
moreover

— If it is reachable and not marked, it is still reachable using...
let $\forall x.\ x \in R \wedge \neg\ m\ x \longrightarrow x \in$ *reachable ?Ra ?A* = *?I4*
let $\forall x.\ x \in R \wedge \neg$ *?new-m* $x \longrightarrow x \in$ *reachable ?Rb ?B* = *?puI4*
let *?T* = $\{t\}$
have *?Ra** ``*addrs ?A* $\subseteq$ *?Rb*$^*$ ``(*addrs ?B* $\cup$ *addrs ?T*)
proof (*rule still-reachable*)
show *addrs ?A* $\subseteq$ *?Rb** `` (*addrs ?B* $\cup$ *addrs ?T*)
by (*fastsimp simp*:*new-stack-eq addrs-def intro*:*self-reachable*)
next
show $\forall (x, y) \in$ *?Ra*$-$*?Rb*. $y \in$(*?Rb** ``(*addrs ?B* $\cup$ *addrs ?T*))
by (*clarsimp simp*:*relS-def new-stack-eq restr-un restr-upd*)
(*fastsimp simp add*:*rel-def Image-iff restr-def addrs-def*
fun-upd-apply addr-t-eq dest:*rel-upd3*)
qed
then have *subset*: *?Ra** ``*addrs ?A* $-$ *?Rb*$^*$ ``*addrs ?T* $\subseteq$ *?Rb** ``*addrs ?B*
by *blast*
have *?puI4*
proof (*rule allI*, *rule impI*)
fix x
assume *a*: $x \in R \wedge \neg$ *?new-m* x
have *xDisj*: x=(*addr* t) $\vee$ $x \neq$(*addr* t) **by** *simp*
with *i4 a* **have** *inc*: $x \in$ *?Ra** ``*addrs ?A*
by (*fastsimp simp*:*addr-t-eq addrs-def reachable-def*
intro:*self-reachable*)
have *exc*: $x \notin$ *?Rb** `` *addrs ?T*
using *xDisj a n-m-addr-t*
by (*clarsimp simp add*:*addrs-def addr-t-eq*)
from *inc exc subset* **show** $x \in$ *reachable ?Rb ?B*
by (*auto simp add*:*reachable-def*)
qed
moreover

— If it is marked, then it is reachable
from *i5*
have *?puI5*
by (*auto simp*:*addrs-def i3 reachable-def addr-t-eq fun-upd-apply*
intro:*self-reachable*)
moreover

— If it is not on the stack, then its l and r fields are unchanged
from *i6*
have *?puI6*

by (*simp add*:*new-stack-eq*)
moreover

— If it is on the stack, then its l and r fields can be reconstructed
from *stackDist i6 t-notin-stack i7*
have *?puI7* **by** (*clarsimp simp*:*addr-t-eq new-stack-eq*)

ultimately show *?thesis* **by** *auto*
qed
then have $\exists$ *stack*. *?puInv stack* **by** *blast*

}
ultimately show *?thesis* **by** *blast*
qed
}
qed

end