Rely-Guarantee Specification of a Concurrent Garbage Collector

Roger C. Su and Nisansala P. Yatapanage

October 27, 2026

Abstract

We specify a model of the Ben-Ari concurrent garbage collector in the rely-guarantee paradigm, and verify its correctness. The aim is to show that the algorithm can be verified with rely-guarantee without the use of shared auxiliary variables. The approach is to reason that a certain property is maintained by one component for all instances of a variable, and then instantiate it to the particular instance required by the rely of the other component.

Contents

1 Introduction **1**

2 Preliminaries **2**

3 Definitions **3**

4 Lemmas **6**
4.1 Stability Lemmas 6
4.2 Lemma 1 9
4.3 Lemma 2 10

5 Theorems **11**
5.1 Holds for All 11
5.2 Propagate 12
5.3 Every Reachable Node Is Marked 13

1 Introduction

Jones and Yatapanage [JY19] showed that Ben-Ari's concurrent garbage collector [BA84] is difficult to specify in the rely-guarantee paradigm without shared auxiliary variables, which break compositionality. This proof-

development, which accompanies [Yat26a, Yat26b], expands on an alternative approach briefly mentioned in [JY19]. The idea is to maintain compositionality by using *Holds For All* reasoning. The Mutator is shown to satisfy a certain property over all instances of a variable, without requiring any knowledge about which particular instance the Collector requires for its rely condition. The property is then instantiated to the required context.

The specification focusses on the Collector's Marking phase, which is where the interference from the Mutator manifests. It specifies and establishes (1) that the Mutator maintains the property required for the Collector's rely to hold, and (2) that the postcondition of the Collector's Marking phase is reached.

Further details can be found in [Yat26a, Yat26b].

theory *RG-Garbage-Collector* **imports** *Main* **begin**

2 Preliminaries

lemma *holds-for-all*:
  assumes $guar\text{-}b\ \sigma\ \sigma'$
    and $p\ s\ \sigma$
    and $\forall\ w.\ (guar\text{-}b\ \sigma\ \sigma' \wedge p\ w\ \sigma \longrightarrow q\ w\ \sigma\ \sigma')$
    and $p\ s\ \sigma \wedge q\ s\ \sigma\ \sigma' \longrightarrow rely\text{-}a\ \sigma\ \sigma'$
  shows $rely\text{-}a\ \sigma\ \sigma'$
  using *assms* **by** *blast*

definition *reach*
  $::\ 'a\ set \Rightarrow 'a\ rel \Rightarrow 'a\ set$ **where**
  $reach\ S\ R \equiv rtrancl\ R\ ``\ S$

lemma *reach-base*:
  $w \in S \Longrightarrow w \in reach\ S\ R$
  unfolding *reach-def* **by** *blast*

lemma *reach-step*:
  $[\![\ w \in reach\ S\ R\ ;\ (w,w') \in R\ ]\!] \Longrightarrow w' \in reach\ S\ R$
  unfolding *reach-def* **apply** *clarsimp*
  by (*meson ImageI rtrancl.simps*)

lemma *reach-induct*:
  assumes $w \in reach\ S\ R$
    and $\bigwedge\ v.\ v \in S \Longrightarrow P\ v$
    and $\bigwedge\ u\ v.\ u \in reach\ S\ R \Longrightarrow P\ u \Longrightarrow (u,v) \in R \Longrightarrow P\ v$
  shows $P\ w$
  using *assms* **unfolding** *reach-def* **apply** *clarsimp*
  by (*metis* (*full-types*) *ImageI rtrancl-induct*)

3 Definitions

The *nodes* in memory are represented as an arbitrary type $'a$.

Part of the system state is the local variables of the Collector, which cannot be accessed by the Mutator.

record $'a$ *localC* =
  safe :: $'a$ *set*
  consid :: $'a$ *set*

The state of the system is defined as the following record-type.

record $'a$ *state* =
  roots :: $'a$ *set*
  hp :: $'a$ *rel* — the heap is modelled as a directed graph (i.e. a relation)
  free :: $'a$ *set*
  marked :: $'a$ *set*
  tbm :: $'a$ *option*
  localCvar :: $'a$ *localC*

Then we formulate the notion of "a node being reachable in a state".

definition *reachable*
  :: $'a$ *state* $\Rightarrow$ $'a$ $\Rightarrow$ *bool* **where**
  reachable σ x $\equiv$ $x \in$ *reach* (*roots* σ) (*hp* σ)

lemma *reachable-exists*:
  reachable σ $y \Longrightarrow y \in$ *roots* $\sigma \vee (\exists\ x.$ *reachable* σ $x \wedge (x,y) \in$ *hp* $\sigma)$
  unfolding *reachable-def reach-def* **apply** *clarsimp*
  by (*metis ImageI rtrancl.cases*)

We now define the various clauses that make up of our specification.

In the definitions below, if *xy* is a pair in the heap, then *fst xy* is the node in the domain ($(a,j) \in$ *dom hp* in the paper), and *snd xy* is any child of the node ($hp(a,j)$ in the paper).

All the marked nodes in the given set have marked children.

definition *all-ok*
  :: $'a$ *set* $\Rightarrow$ $'a$ *state* $\Rightarrow$ *bool* **where**
  all-ok s σ $\equiv$ $\forall$ $xy \in$ *hp* σ.
    fst $xy \in$ *marked* $\sigma \wedge$
    fst $xy \in s$
    $\longrightarrow$ (*snd* $xy \in$ *marked* σ)

For all marked nodes in the given set that have unmarked children, there is another path to that child.

definition *can-be-marked*
  :: $'a$ *set* $\Rightarrow$ $'a$ *state* $\Rightarrow$ *bool* **where**
  can-be-marked s σ $\equiv \forall$ $xy \in$ *hp* σ.

$fst\ xy \in marked\ \sigma \cap s \wedge$
$snd\ xy \notin marked\ \sigma \wedge$
$reachable\ \sigma\ (snd\ xy)$
$\longrightarrow (\exists\ zw \in hp\ \sigma.$
$(fst\ zw \in\ reach\ (roots\ \sigma)\ (hp\ \sigma) \vee fst\ zw \in reach\ (marked\ \sigma \cap -s)\ (hp\ \sigma)) \wedge$
$(fst\ zw \notin s \vee fst\ zw \notin marked\ \sigma) \wedge$
$fst\ xy \neq fst\ zw \wedge$
$snd\ xy = snd\ zw)$

The *tbm* field is the only child node that (1) is unmarked and (2) has a marked parent.

definition *tbm-put*
$::\ 'a\ set \Rightarrow\ 'a\ state \Rightarrow bool$ **where**
$tbm\text{-}put\ s\ \sigma \equiv \forall\ xy \in hp\ \sigma.$
$tbm\ \sigma \neq Some\ (snd\ xy) \wedge$
$fst\ xy \in s \wedge$
$fst\ xy \in marked\ \sigma$
$\longrightarrow (snd\ xy \in marked\ \sigma)$

The rely condition of the collector.

definition *rely-collector*
$::\ 'a\ state \Rightarrow\ 'a\ state \Rightarrow bool$ **where**
$rely\text{-}collector\ \sigma\ \sigma' \equiv marked\ \sigma \subseteq marked\ \sigma' \wedge localCvar\ \sigma = localCvar\ \sigma' \wedge$
$(all\text{-}ok\ (safe\ (localCvar\ \sigma))\ \sigma \longrightarrow$
$can\text{-}be\text{-}marked\ (safe\ (localCvar\ \sigma))\ \sigma' \vee marked\ \sigma \subset marked\ \sigma')$

The guarantee condition of the Propagate phase of the Collector.

definition *guar-collector*
$::\ 'a\ state \Rightarrow\ 'a\ state\ \Rightarrow bool$ **where**
$guar\text{-}collector\ \sigma\ \sigma' \equiv marked\ \sigma \subseteq marked\ \sigma' \wedge$
$consid\ (localCvar\ \sigma) \subseteq consid\ (localCvar\ \sigma') \wedge$
$safe\ (localCvar\ \sigma') \subseteq safe\ (localCvar\ \sigma) \wedge$
$((safe\ (localCvar\ \sigma') = consid\ (localCvar\ \sigma')) \vee$
$((safe\ (localCvar\ \sigma') \subset consid\ (localCvar\ \sigma))) \wedge (marked\ \sigma \subset marked\ \sigma))$

The post condition of the Propagate phase of the Collector.

definition *post-propagate*
$::\ 'a\ state \Rightarrow\ 'a\ state\ \Rightarrow bool$ **where**
$post\text{-}propagate\ \sigma\ \sigma' \equiv \forall\ (x,y) \in hp\ \sigma.$
$x \in marked\ \sigma' \cap consid\ (localCvar\ \sigma') \longrightarrow$
$(y \in marked\ \sigma' \vee (marked\ \sigma \subset marked\ \sigma'))$

The post condition of the remove action of the Mutator.

definition *post-change-remove*
$::\ 'a\ state \Rightarrow\ 'a\ state \Rightarrow bool$ **where**
$post\text{-}change\text{-}remove\ \sigma\ \sigma' \equiv \exists\ xy.$
$xy \notin hp\ \sigma' \wedge hp\ \sigma = hp\ \sigma' \cup \{xy\} \wedge reachable\ \sigma\ (fst\ xy) \wedge$
$tbm\ \sigma' = None$

The post condition of the redirect action of the Mutator.

definition *post-change-redirect*
:: $'a$ *state* $\Rightarrow$ $'a$ *state* $\Rightarrow$ *bool* **where**
post-change-redirect σ σ' $\equiv$ $\exists$ x y z.
hp σ $-$ *hp* σ' $=$ $\{(x,y)\}$ $\wedge$ *hp* σ' $-$ *hp* σ $=$ $\{(x,z)\}$ $\wedge$ *reachable* σ x $\wedge$
tbm σ' $=$ *Some* z $\wedge$ *reachable* σ z

The common post condition of the remove and redirect actions of the Mutator. It describes the behaviour when the edge $(x,\ y)$ is removed or redirected to $(x,\ a)$.

definition *post-change*
:: $'a$ *state* $\Rightarrow$ $'a$ *state* $\Rightarrow$ *bool* **where**
post-change σ σ' $\equiv$ *roots* σ $\neq$ $\{\}$ $\wedge$
(*post-change-remove* σ σ' $\vee$ *post-change-redirect* σ σ') $\wedge$
(*marked* σ $=$ *marked* σ') $\wedge$
(*roots* σ $=$ *roots* σ') $\wedge$
(*tbm* σ $=$ *tbm* σ') $\wedge$
(*localCvar* σ $=$ *localCvar* σ')

The post condition of the action of the Mutator that sets *tbm*.

definition *post-mark-tbm*
:: $'a$ *state* $\Rightarrow$ $'a$ *state* $\Rightarrow$ *bool* **where**
post-mark-tbm σ σ' $\equiv$
((*tbm* σ $=$ *None* $\wedge$ *marked* σ' $=$ *marked* σ) $\vee$
($\exists x$. *tbm* σ $=$ *Some* x $\wedge$ *marked* σ' $=$ *marked* σ $\cup$ $\{x\}$)) $\wedge$
tbm σ' $=$ *None* $\wedge$ *hp* σ $=$ *hp* σ' $\wedge$ *roots* σ $=$ *roots* σ' $\wedge$
(*localCvar* σ $=$ *localCvar* σ')

An invariant that the roots always remain marked.

definition *inv-roots*
:: $'a$ *state* $\Rightarrow$ *bool* **where**
inv-roots σ $\equiv$ *roots* σ $\subseteq$ *marked* σ

lemma *inv-roots-change*:
$[\![$ *post-change* σ σ' ; *inv-roots* σ $]\!]$ $\Longrightarrow$ *inv-roots* σ'
unfolding *post-change-def inv-roots-def* **by** *blast*

lemma *inv-roots-mark*:
$[\![$ *post-mark-tbm* σ σ' ; *inv-roots* σ $]\!]$ $\Longrightarrow$ *inv-roots* σ'
unfolding *post-mark-tbm-def inv-roots-def* **by** *blast*

A Mutator step preserves the invariant.

lemma *inv-roots-step*:
$[\![$ *post-change* σ σ' $\vee$ *post-mark-tbm* σ σ' ; *inv-roots* σ $]\!]$ $\Longrightarrow$ *inv-roots* σ'
using *inv-roots-change inv-roots-mark* **by** *blast*

4 Lemmas

This lemma establishes one of the premises when instantiating *holds-for-all* to the concurrent garbage collector.

lemma *p-q-gives-rely*:
  assumes *all-ok* (*safe* (*localCvar* σ)) σ
    and *marked* $\sigma \subseteq$ *marked* $\sigma' \wedge$ *localCvar* σ = *localCvar* $\sigma' \wedge$
      (*can-be-marked* (*safe* (*localCvar* σ)) $\sigma' \vee$ *marked* $\sigma \subset$ *marked* σ')
  shows *rely-collector* σ σ'
  using *assms* **unfolding** *rely-collector-def* **by** *blast*

lemma *all-ok-conseqs*:
  assumes *all-ok s* σ
  shows *can-be-marked s* σ
    and *tbm-put s* σ
  using *assms* **unfolding** *all-ok-def can-be-marked-def tbm-put-def* **by** *auto*

4.1 Stability Lemmas

A Mutator remove step preserves *all-ok*.

lemma *stable-all-ok-remove*:
  assumes *post-change* σ $\sigma' \wedge$ *post-change-remove* σ σ'
    and *all-ok s* σ
  shows *all-ok s* σ'
  using *assms* **unfolding** *post-change-remove-def post-change-def all-ok-def*
  apply *clarsimp*
  by (*metis* (*no-types*, *opaque-lifting*) *fst-conv snd-conv*)

lemma *reachable-post-pre*:
  assumes *assm-step*: *post-change* σ σ'
    and *assm-ok*: *all-ok s* σ
  shows $\forall$ *x*. *reachable* σ' *x* $\longrightarrow$ *reachable* σ *x*
proof −
  from *assm-step* **have** *post-change-remove* σ $\sigma' \vee$ *post-change-redirect* σ σ'
    unfolding *post-change-def* **by** *simp*
  then show *?thesis*
  proof (*rule disjE*, *goal-cases case-remove case-redirect*)
    case *case-remove*
    then show *?case*
      unfolding *post-change-remove-def reachable-def* **apply** *clarsimp*
      by (*smt* (*verit*, *ccfv-SIG*) *DiffE Diff-insert-absorb assm-step post-change-def*
        reach-base reach-induct reach-step reachable-def)
  next
    case *case-redirect*
    then show *?case*
      unfolding *post-change-redirect-def reachable-def* **apply** *clarsimp*
      by (*smt* (*verit*, *ccfv-SIG*) *Diff-iff Diff-insert assm-step case-redirect*
       insert-Diff-single insert-absorb insert-iff post-change-def reach-base reach-induct

reach-step reachable-def singletonD singletonI snd-conv)
qed
qed

The *all-ok* property and a redirect step by the Mutator together establishes *can-be-marked*.

lemma *stable-can-be-marked-redirect*:
assumes *assm-step*: *post-change* σ σ' $\wedge$ *post-change-redirect* σ σ'
and *assm-ok*: *all-ok* s σ
and *assm-roots-marked*: *roots* $\sigma \subseteq$ *marked* σ
shows *can-be-marked* s σ'
proof −
from *assm-step* **have** *ln-marked-eq*: *marked* $\sigma' =$ *marked* σ
unfolding *post-change-def* **by** *simp*
then have *roots* $\sigma' \subseteq$ *marked* σ'
using *assm-step assm-roots-marked* **unfolding** *post-change-def* **by** *simp*

from *assm-step* **obtain** x y z **where** *assm-xyz*:
hp σ − *hp* $\sigma' = \{(x,y)\} \wedge$
hp σ' − *hp* $\sigma = \{(x,z)\} \wedge$
reachable σ x $\wedge$
tbm $\sigma' =$ *Some* z $\wedge$
reachable σ z
unfolding *post-change-redirect-def* **by** *force*

show *?thesis*
proof −
{ fix a b **assume** *assm-ab*: — The assumption-part of *can-be-marked*
$(a,b) \in$ *hp* $\sigma' \wedge (a \in$ *marked* $\sigma' \cap s) \wedge b \notin$ *marked* $\sigma' \wedge$ *reachable* σ' b

then have *ln-ab*: $(a \in$ *marked* $\sigma \cap s) \wedge b \notin$ *marked* $\sigma \wedge$ *reachable* σ b
using *assm-ab ln-marked-eq reachable-post-pre assm-step assm-ok* **by** *blast*

— The assumptions on (a, b) imply that this edge is exactly (x, z) from earlier.
then have *ln-ab-xy*: $(a,b) = (x,z)$
using *assm-ab assm-ok assm-xyz* **unfolding** *all-ok-def* **by** *fastforce*

— The node z must have another parent c
have $z \in$ *roots* $\sigma \vee (\exists$ c. *reachable* σ $c \wedge (c,z) \in$ *hp* $\sigma)$
using *assm-xyz reachable-exists* **by** *fast*
moreover have $z \notin$ *roots* σ
using *assm-roots-marked assm-ab ln-ab-xy ln-marked-eq* **by** *blast*
ultimately obtain c **where** *ln-c*:
reachable σ $c \wedge (c,z) \in$ *hp* σ
by *blast*
then have *ln-c1*: $c \neq x \wedge (c,z) \in$ *hp* σ'
using *assm-xyz* **by** *blast*

— We now establish the conclusion-part of *can-be-marked* s σ', using (c, z) as

the witness to the existential quantifier.

```
— Clause 2 of 3, done first because it is needed later
have ln-c2: c ∉ s ∨ c ∉ marked σ′
  by (metis all-ok-def assm-ab assm-ok ln-ab-xy ln-c ln-marked-eq
      fst-conv snd-conv)

— Clause 1 of 3
moreover
have c ∈ reach (roots σ′) (hp σ′) ∨ c ∈ reach (marked σ′ ∩ − s) (hp σ′)
 (is ?Clause1 c)
proof −
  define Q where Q = (λw. ?Clause1 w ∨ (w ∈ marked σ′ ∧ w ∈ s))
  have ln-marked-Q: ⋀ w. w ∈ marked σ ⟹ Q w
    by (metis reach-base assm-step post-change-def IntI Q-def Compl-iff)
  have ⋀ w. reachable σ w ⟹ Q w
  proof −
    fix w assume assm-reachable: reachable σ w
    show Q w
    proof (rule reach-induct[where P=Q and S=roots σ and R=hp σ],
          goal-cases)
      case 1
      then show ?case by (meson assm-reachable reachable-def)
    next
      case 2
      then show ?case using assm-roots-marked ln-marked-Q by blast
    next
      case (3 u v)
      assume assm-ln-1: u ∈ reach (roots σ) (hp σ)
         and assm-ln-2: Q u
         and assm-ln-3: (u,v) ∈ hp σ
      have v-r: reachable σ v
        unfolding reachable-def by (rule reach-step[OF assm-ln-1 assm-ln-3])
      show ?case
      proof (cases (u,v) = (x,y))
        case True
        then have v ∈ marked σ
          by (metis assm-ab assm-ln-3 assm-ok all-ok-def ln-ab-xy ln-marked-eq
                    Int-iff fst-conv snd-conv)
        then show ?thesis using ln-marked-Q by simp
      next
        case False
        then have (u,v) ∈ hp σ′
          using assm-xyz assm-ln-3 by auto
        moreover have Q u
          using assm-ln-2 unfolding Q-def by auto
        ultimately show ?thesis unfolding Q-def
          apply (elim disjE)
            apply (metis reach-step)
```

```
            apply (metis reach-step)
            by (metis assm-ok all-ok-def assm-ln-3 ln-marked-Q ln-marked-eq
                    Q-def fst-conv snd-conv)
        qed
      qed
    qed — for ⋀ w. reachable σ w ⟹ Q w
    then show ?thesis
      using ln-c ln-c2 unfolding Q-def by blast
  qed — for Clause 2 of 3

  — Clause 3 of 3
  moreover have a ≠ c ∧ b = z
    using ln-ab-xy ln-c1 by blast

  ultimately have (∃ zw ∈ hp σ'.
          (fst zw ∈ reach (roots σ') (hp σ') ∨
           fst zw ∈ reach (marked σ' ∩ − s) (hp σ')) ∧
          (fst zw ∉ s ∨ fst zw ∉ marked σ') ∧ a ≠ fst zw ∧ b = snd zw)
    using ln-c1 by force
 }
 then show ?thesis unfolding can-be-marked-def by simp
qed
qed

lemma stable-can-be-marked-remove:
  assumes post-change σ σ' ∧ post-change-remove σ σ'
    and all-ok s σ
  shows can-be-marked s σ'
  unfolding can-be-marked-def apply clarsimp
  by (metis all-ok-def assms fst-eqD snd-eqD stable-all-ok-remove)
```

4.2 Lemma 1

This subsection establishes Lemma 1 from the paper, which concerns the properties that follow from a Mutator change.

```
lemma lemma-1-can-be-marked:
  assumes assm-step: post-change σ σ'
    and assm-all-ok: all-ok s σ
    and assm-roots-marked: roots σ ⊆ marked σ
  shows can-be-marked s σ'
proof −
  from assm-step have post-change-remove σ σ' ∨ post-change-redirect σ σ'
    using post-change-def by fast
  then show ?thesis
  proof (standard, goal-cases remove redirect)
    case remove
    then show ?case
      using stable-can-be-marked-remove
      by (metis stable-can-be-marked-remove remove assm-step assm-all-ok)
```

```
  next
    case redirect
    then show ?case
      using assms stable-can-be-marked-redirect post-change-def by blast
  qed
qed

lemma lemma-1-tbm-put:
  assumes assm-step:   post-change σ σ'
      and assm-all-ok: all-ok s σ
    shows tbm-put s σ'
proof −
  have ⋀ xy. xy ∈ hp σ' ⟹ tbm σ' ≠ Some (snd xy) ⟹ xy ∈ hp σ
  proof −
    fix xy
    assume assm-local-hp: xy ∈ hp σ'
       and assm-local-tbm: tbm σ' ≠ Some (snd xy)
    have post-change-remove σ σ' ∨ post-change-redirect σ σ'
      using assm-step unfolding post-change-def by blast
    then show xy ∈ hp σ
      apply standard
      using assm-local-hp post-change-remove-def apply blast
      by (metis assm-local-hp assm-local-tbm post-change-redirect-def
              DiffI singletonD snd-conv)
  qed
  then show ?thesis
    using assm-all-ok assm-step
    unfolding all-ok-def post-change-def tbm-put-def
    by fastforce
qed

lemma lemma-1:
  assumes post-change σ σ'
      and all-ok s σ
      and roots σ ⊆ marked σ
    shows can-be-marked s σ' ∧ tbm-put s σ'
  using assms lemma-1-can-be-marked lemma-1-tbm-put by blast
```

4.3 Lemma 2

This subsection establishes Lemma 2 from the paper: after a Mutator step that marks *tbm* and *tbm-put*, either *all-ok* holds or the number of marked nodes increases.

```
lemma lemma-2:
  assumes tbm-put s σ
      and post-mark-tbm σ σ'
    shows all-ok s σ' ∨ (marked σ ⊂ marked σ')
  using assms unfolding tbm-put-def post-mark-tbm-def apply clarsimp
  by (metis all-ok-def antisym-conv1 insert-iff option.distinct(1) option.inject
```

subset-insertI)

5 Theorems

5.1 Holds for All

This establishes the *forall-guar-p-w* premise when instantiating the *holds-for-all* lemma to the concurrent garbage collector. Here, p is *all-ok w* σ and q is *can-be-marked w* $\sigma' \lor$ *marked* $\sigma \subset$ *marked* σ'.

lemma *works-for-all-w*:
  assumes *post-change* σ $\sigma' \lor$ *post-mark-tbm* σ σ'
    and *inv-roots* σ
    and *all-ok w* σ
  shows *can-be-marked w* $\sigma' \lor$ *marked* $\sigma \subset$ *marked* σ'
  by (*metis assms all-ok-conseqs inv-roots-def lemma-1 lemma-2*)

This lemma continues the *works-for-all-w* reasoning, but with q extended to include the case where the Collector's local variables are unchanged and the marking only increases.

theorem *works-for-all-w2*:
  assumes *post-change* σ $\sigma' \lor$ *post-mark-tbm* σ σ'
    and *inv-roots* σ
    and *all-ok w* σ
  shows *marked* $\sigma \subseteq$ *marked* $\sigma' \land$ *localCvar* $\sigma =$ *localCvar* $\sigma' \land$
      (*can-be-marked w* $\sigma' \lor$ *marked* $\sigma \subset$ *marked* σ')
proof (*intro conjI*, *goal-cases*)
  case *1* **show** *?case*
    using *assms*(*1*) **unfolding** *post-change-def post-mark-tbm-def* **by** *blast*
next
  case *2* **show** *?case*
    using *assms*(*1*) **unfolding** *post-change-def post-mark-tbm-def* **by** *blast*
next
  case *3* **show** *?case*
    using *assms works-for-all-w* **by** *blast*
qed

This theorem shows that the rely of the Collector holds, based on the *holds-for-all* reasoning.

theorem *using-holds-for-all2*:
  assumes *post-change* σ $\sigma' \lor$ *post-mark-tbm* σ σ'
    and *all-ok s* σ
    and $\forall$ *w*. ((*post-change* σ $\sigma' \lor$ *post-mark-tbm* σ σ') $\land$ *all-ok w* σ)
      $\longrightarrow$ (*marked* $\sigma \subseteq$ *marked* $\sigma' \land$ *localCvar* $\sigma =$ *localCvar* $\sigma' \land$
        (*can-be-marked w* $\sigma' \lor$ *marked* $\sigma \subset$ *marked* σ'))
    and *all-ok s* $\sigma \land$ *marked* $\sigma \subseteq$ *marked* $\sigma' \land$ *localCvar* $\sigma =$ *localCvar* $\sigma' \land$
      (*can-be-marked s* $\sigma' \lor$ *marked* $\sigma \subset$ *marked* σ')
      $\longrightarrow$ *rely-collector* σ σ'

shows *rely-collector* σ σ'
using *assms holds-for-all* **by** *force*

5.2 Propagate

lemma *chain-mono*:
$[\![\bigwedge m.\ f\ m \subseteq f\ (Suc\ m)\ ;\ a \leq b\ ]\!] \Longrightarrow f\ a \subseteq f\ b$
using *lift-Suc-mono-le* **by** *blast*

lemma *chain-anti*:
$[\![\bigwedge m.\ f\ (Suc\ m) \subseteq f\ m\ ;\ a \leq b\ ]\!] \Longrightarrow f\ b \subseteq f\ a$
using *lift-Suc-antimono-le* **by** *blast*

This theorem shows that *post-propagate* holds over a sequence of steps, where each step either satisfies *rely-collector* or *guar-collector*, and the *safe* and *consid* sets are initially empty.

theorem *propagate-seq*:
fixes $st :: nat \Rightarrow 'a\ state$
assumes *step*: $\bigwedge m.$ *rely-collector* $(st\ m)\ (st\ (Suc\ m))$
$\vee$ *guar-collector* $(st\ m)\ (st\ (Suc\ m))$
and *init-empty*: *safe* $(localCvar\ (st\ 0)) = \{\} \wedge$
consid $(localCvar\ (st\ 0)) = \{\}$
shows *post-propagate* $(st\ 0)\ (st\ n)$
proof −
have *init-eq*: *safe* $(localCvar\ (st\ 0)) = consid\ (localCvar\ (st\ 0))$
using *init-empty* **by** *simp*
have *ok0*: *all-ok* $(safe\ (localCvar\ (st\ 0)))\ (st\ 0)$
using *init-empty all-ok-def* **by** *blast*
have *mark-step*: $\bigwedge m.$ *marked* $(st\ m) \subseteq$ *marked* $(st\ (Suc\ m))$
by (*metis rely-collector-def guar-collector-def local.step*)
have *safe-step*: $\bigwedge m.$ *safe* $(localCvar\ (st\ (Suc\ m))) \subseteq$ *safe* $(localCvar\ (st\ m))$
by (*metis rely-collector-def guar-collector-def local.step dual-order.refl*)

have $\bigwedge x\ y.\ (x,y) \in hp\ (st\ 0) \Longrightarrow x \in marked\ (st\ n) \Longrightarrow$
$x \in consid\ (localCvar\ (st\ n)) \Longrightarrow$
$y \in marked\ (st\ n) \vee marked\ (st\ 0) \subset marked\ (st\ n)$
proof −
fix $x\ y$ **assume** $(x,y) \in hp\ (st\ 0)$
and $x \in marked\ (st\ n)$
and *local-assm3*: $x \in consid\ (localCvar\ (st\ n))$
show $y \in marked\ (st\ n) \vee marked\ (st\ 0) \subset marked\ (st\ n)$
proof −
{ assume $\neg\ (marked\ (st\ 0) \subset marked\ (st\ n))$
moreover have $marked\ (st\ 0) \subseteq marked\ (st\ n)$
by (*rule chain-mono*[*of* $\lambda m.$ *marked* $(st\ m)$, *OF mark-step*, *of* $0\ n$]) *simp*
ultimately have *eqn*: $marked\ (st\ 0) = marked\ (st\ n)$
by *simp*

— marked is constant on 0..n

```
      have eqm: ⋀m. m ≤ n ⟹ marked (st m) = marked (st 0)
      proof −
        fix m assume assm: m ≤ n
        then have marked (st m) ⊆ marked (st n)
          by (rule chain-mono[of λm. marked (st m), OF mark-step, of m n])
        moreover have marked (st 0) ⊆ marked (st m)
          apply (rule chain-mono[of λm. marked (st m), OF mark-step, of 0 m])
          by simp
        ultimately show marked (st m) = marked (st 0)
          using eqn by blast
      qed

      — safe = consid holds at every state up to n, by induction from 0
      have all-eq: ⋀m. m ≤ n ⟹
                    safe (localCvar (st m)) = consid (localCvar (st m))
      proof −
        fix m show m ≤ n ⟹
                    safe (localCvar (st m)) = consid (localCvar (st m))
          apply (induction m)
           apply (rule init-eq)
          apply clarsimp
          by (metis rely-collector-def local.step guar-collector-def psubsetE)
      qed

      have safe (localCvar (st n)) = consid (localCvar (st n))
        by (rule all-eq[OF order-refl])
      moreover
      have s-down: safe (localCvar (st n)) ⊆ safe (localCvar (st 0))
        by (rule chain-anti[of λm. safe (localCvar (st m)),
            OF safe-step, of 0 n]) simp
      ultimately have x ∈ safe (localCvar (st 0))
        using local-assm3 by auto
      then have y ∈ marked (st n)
        using init-empty by blast
    }
    then show ?thesis by blast
  qed
 qed

 then show ?thesis unfolding post-propagate-def by force
qed
```

5.3 Every Reachable Node Is Marked

This final theorem establishes the overall post condition of the Collector, that "if a node is reachable initially, then it has been marked by the Collector.

```
theorem reachable-subset-marked:
 assumes assm-post: post-propagate σ σ′
```

```
    and assm-marked: marked σ' = marked σ
    and assm-roots: roots σ ⊆ marked σ
    and assm-consid: consid (localCvar σ') = Field (hp σ)
      — consid contains all the nodes in the heap.
  shows ∀ w. reachable σ w ⟶ w ∈ marked σ
proof (intro allI impI)
  fix w assume reachable σ w
  hence hw: w ∈ reach (roots σ) (hp σ) unfolding reachable-def .
  show w ∈ marked σ
  proof (rule reach-induct[where P=λu. u ∈ marked σ, OF hw])
    fix u assume u ∈ roots σ
    thus u ∈ marked σ using assm-roots by blast
  next
    fix u v
    assume u ∈ marked σ and (u,v) ∈ hp σ
    moreover from this have u ∈ marked σ' ∧ u ∈ consid (localCvar σ')
      using assm-marked Field-iff assm-consid by force
    ultimately show v ∈ marked σ
    by (metis assm-marked assm-post post-propagate-def Int-iff psubsetE split-conv)
  qed
qed

end
```

References

[BA84] Mordechai Ben-Ari. Algorithms for on-the-fly garbage collection. *ACM Transactions on Programming Languages and Systems*, 6(3):333–344, 1984. doi:10.1145/579.587.

[JY19] Cliff B. Jones and Nisansala Yatapanage. Investigating the limits of rely/guarantee relations based on a concurrent garbage collector example. *Formal Aspects of Computing*, 31:353–374, 2019. doi:10.1007/s00165-019-00482-3.

[Yat26a] Nisansala P. Yatapanage. The complexity of interference: When rely/guarantee does not work. *CoRR*, abs/2609.27392, 2026. arXiv:2609.27392.

[Yat26b] Nisansala P. Yatapanage. The complexity of interference: When rely/guarantee does not work. In *The 12th International Symposium on Dependable Software Engineering: Theories, Tools and Applications (SETTA)*, 2026. to appear.