

# Rely-Guarantee Specification of a Concurrent Garbage Collector

Roger C. Su and Nisansala P. Yatapanage

October 16, 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

|          |                                          |           |
|----------|------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                      | <b>1</b>  |
| <b>2</b> | <b>Preliminaries</b>                     | <b>2</b>  |
| <b>3</b> | <b>Definitions</b>                       | <b>3</b>  |
| <b>4</b> | <b>Lemmas</b>                            | <b>6</b>  |
| 4.1      | Stability Lemmas . . . . .               | 6         |
| 4.2      | Lemma 1 . . . . .                        | 9         |
| 4.3      | Lemma 2 . . . . .                        | 10        |
| <b>5</b> | <b>Theorems</b>                          | <b>11</b> |
| 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-b*  $\sigma \ \sigma'$   
**and**  $p \ s \ \sigma$   
**and**  $\forall \ w. (guar-b \ \sigma \ \sigma' \wedge p \ w \ \sigma \longrightarrow q \ w \ \sigma \ \sigma')$   
**and**  $p \ s \ \sigma \wedge q \ s \ \sigma \ \sigma' \longrightarrow rely-a \ \sigma \ \sigma'$   
**shows** *rely-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*:

$\llbracket w \in reach \ S \ R ; (w, w') \in R \rrbracket \Longrightarrow w' \in reach \ S \ R$   
**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  $\sigma$   $x \equiv x \in \text{reach } (\text{roots } \sigma) (hp \ \sigma)$ 
```

```
lemma reachable-exists:
  reachable  $\sigma$   $y \implies y \in \text{roots } \sigma \vee (\exists x. \text{reachable } \sigma x \wedge (x,y) \in hp \ \sigma)$ 
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  $\text{fst } xy$  is the node in the domain ( $(a,j) \in \text{dom } hp$  in the paper), and  $\text{snd } xy$  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  $s \ \sigma \equiv \forall xy \in hp \ \sigma.
    \text{fst } xy \in \text{marked } \sigma \wedge
    \text{fst } xy \in s
    \longrightarrow (\text{snd } xy \in \text{marked } \sigma)$ 
```

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  $s \ \sigma \equiv \forall xy \in hp \ \sigma.$ 
```

$$\begin{aligned}
&fst\ xy \in marked\ \sigma \cap s \wedge \\
&snd\ xy \notin marked\ \sigma \wedge \\
&reachable\ \sigma\ (snd\ xy) \\
&\longrightarrow (\exists\ zw \in hp\ \sigma. \\
&\quad (fst\ zw \in reach\ (roots\ \sigma)\ (hp\ \sigma) \vee fst\ zw \in reach\ (marked\ \sigma \cap -s)\ (hp\ \sigma)) \wedge \\
&\quad (fst\ zw \notin s \vee fst\ zw \notin marked\ \sigma) \wedge \\
&\quad fst\ xy \neq fst\ zw \wedge \\
&\quad snd\ xy = snd\ zw)
\end{aligned}$$

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

**definition** *tbm-put*

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

The rely condition of the collector.

**definition** *rely-collector*

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

The guarantee condition of the Propagate phase of the Collector.

**definition** *guar-collector*

$$\begin{aligned}
&:: 'a\ state \Rightarrow 'a\ state \Rightarrow bool\ \mathbf{where} \\
&guar\text{-}collector\ \sigma\ \sigma' \equiv marked\ \sigma \subseteq marked\ \sigma' \wedge \\
&\quad consid\ (localCvar\ \sigma) \subseteq consid\ (localCvar\ \sigma') \wedge \\
&\quad safe\ (localCvar\ \sigma') \subseteq safe\ (localCvar\ \sigma) \wedge \\
&\quad ((safe\ (localCvar\ \sigma') = consid\ (localCvar\ \sigma')) \vee \\
&\quad\quad ((safe\ (localCvar\ \sigma') \subset consid\ (localCvar\ \sigma))) \wedge (marked\ \sigma \subset marked\ \sigma'))
\end{aligned}$$

The post condition of the Propagate phase of the Collector.

**definition** *post-propagate*

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

The post condition of the remove action of the Mutator.

**definition** *post-change-remove*

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

The post condition of the redirect action of the Mutator.

**definition** *post-change-redirect*

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

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*

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

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

**definition** *post-mark-tbm*

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

An invariant that the roots always remain marked.

**definition** *inv-roots*

$:: 'a \text{ state} \Rightarrow \text{bool} \text{ where}$   
 $\text{inv-roots } \sigma \equiv \text{roots } \sigma \subseteq \text{marked } \sigma$

**lemma** *inv-roots-change:*

$\llbracket \text{post-change } \sigma \sigma' ; \text{inv-roots } \sigma \rrbracket \Longrightarrow \text{inv-roots } \sigma'$   
**unfolding** *post-change-def inv-roots-def* **by** *blast*

**lemma** *inv-roots-mark:*

$\llbracket \text{post-mark-tbm } \sigma \sigma' ; \text{inv-roots } \sigma \rrbracket \Longrightarrow \text{inv-roots } \sigma'$   
**unfolding** *post-mark-tbm-def inv-roots-def* **by** *blast*

A Mutator step preserves the invariant.

**lemma** *inv-roots-step:*

$\llbracket \text{post-change } \sigma \sigma' \vee \text{post-mark-tbm } \sigma \sigma' ; \text{inv-roots } \sigma \rrbracket \Longrightarrow \text{inv-roots } \sigma'$   
**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*  $\sigma$ ))  $\sigma$   
**and** *marked*  $\sigma \subseteq \text{marked } \sigma' \wedge \text{localCvar } \sigma = \text{localCvar } \sigma' \wedge$   
 $(\text{can-be-marked } (\text{safe } (\text{localCvar } \sigma)) \sigma' \vee \text{marked } \sigma \subset \text{marked } \sigma')$   
**shows** *rely-collector*  $\sigma \sigma'$   
**using** *assms* **unfolding** *rely-collector-def* **by** *blast*

**lemma** *all-ok-conseqs*:  
**assumes** *all-ok*  $s \sigma$   
**shows** *can-be-marked*  $s \sigma$   
**and** *tbm-put*  $s \sigma$   
**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*  $\sigma \sigma' \wedge \text{post-change-remove } \sigma \sigma'$   
**and** *all-ok*  $s \sigma$   
**shows** *all-ok*  $s \sigma'$   
**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*  $\sigma \sigma'$   
**and** *assm-ok*: *all-ok*  $s \sigma$   
**shows**  $\forall x. \text{reachable } \sigma' x \longrightarrow \text{reachable } \sigma x$   
**proof** –  
**from** *assm-step* **have** *post-change-remove*  $\sigma \sigma' \vee \text{post-change-redirect } \sigma \sigma'$   
**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*  $\sigma$   $\sigma' \wedge$  *post-change-redirect*  $\sigma$   $\sigma'$   
**and** *assm-ok*: *all-ok*  $s$   $\sigma$   
**and** *assm-roots-marked*: *roots*  $\sigma \subseteq$  *marked*  $\sigma$   
**shows** *can-be-marked*  $s$   $\sigma'$

**proof** —

**from** *assm-step* **have** *ln-marked-eq*: *marked*  $\sigma' =$  *marked*  $\sigma$   
**unfolding** *post-change-def* **by** *simp*  
**then have** *roots*  $\sigma' \subseteq$  *marked*  $\sigma'$   
**using** *assm-step* *assm-roots-marked* **unfolding** *post-change-def* **by** *simp*

**from** *assm-step* **obtain**  $x$   $y$   $z$  **where** *assm-xyz*:

*hp*  $\sigma$  — *hp*  $\sigma' = \{(x,y)\} \wedge$   
*hp*  $\sigma' -$  *hp*  $\sigma = \{(x,z)\} \wedge$   
*reachable*  $\sigma$   $x \wedge$   
*tbn*  $\sigma' =$  *Some*  $z \wedge$   
*reachable*  $\sigma$   $z$

**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*  $\sigma' b$

**then have** *ln-ab*:  $(a \in$  *marked*  $\sigma \cap s) \wedge b \notin$  *marked*  $\sigma \wedge$  *reachable*  $\sigma b$   
**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*  $\sigma \vee (\exists c. \text{reachable } \sigma c \wedge (c,z) \in \text{hp } \sigma)$   
**using** *assm-xyz* *reachable-exists* **by** *fast*

**moreover have**  $z \notin$  *roots*  $\sigma$

**using** *assm-roots-marked* *assm-ab* *ln-ab-xy* *ln-marked-eq* **by** *blast*

**ultimately obtain**  $c$  **where** *ln-c*:

*reachable*  $\sigma$   $c \wedge (c,z) \in$  *hp*  $\sigma$

**by** *blast*

**then have** *ln-c1*:  $c \neq x \wedge (c,z) \in$  *hp*  $\sigma'$

**using** *assm-xyz* **by** *blast*

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

the witness to the existential quantifier.

— Clause 2 of 3, done first because it is needed later

**have** *ln-c2*:  $c \notin s \vee c \notin \text{marked } \sigma'$   
**by** (*metis all-ok-def assem-ab assem-ok ln-ab-xy ln-c ln-marked-eq*  
*fst-conv snd-conv*)

— Clause 1 of 3

**moreover**  
**have**  $c \in \text{reach } (\text{roots } \sigma') (hp \ \sigma') \vee c \in \text{reach } (\text{marked } \sigma' \cap - \ s) (hp \ \sigma')$   
**(is** *?Clause1 c*)

**proof** –  
**define** *Q* **where**  $Q = (\lambda w. \text{?Clause1 } w \vee (w \in \text{marked } \sigma' \wedge w \in s))$   
**have** *ln-marked-Q*:  $\bigwedge w. w \in \text{marked } \sigma \implies Q \ w$   
**by** (*metis reach-base assem-step post-change-def IntI Q-def Compl-iff*)  
**have**  $\bigwedge w. \text{reachable } \sigma \ w \implies Q \ w$   
**proof** –  
**fix** *w* **assume** *assem-reachable*:  $\text{reachable } \sigma \ w$   
**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 assem-reachable reachable-def*)  
**next**  
**case** 2  
**then show** *?case* **using** *assem-roots-marked ln-marked-Q* **by** *blast*  
**next**  
**case** ( $\exists u \ v$ )  
**assume** *assem-ln-1*:  $u \in \text{reach } (\text{roots } \sigma) (hp \ \sigma)$   
**and** *assem-ln-2*:  $Q \ u$   
**and** *assem-ln-3*:  $(u, v) \in hp \ \sigma$   
**have** *v-r*:  $\text{reachable } \sigma \ v$   
**unfolding** *reachable-def* **by** (*rule reach-step[OF assem-ln-1 assem-ln-3]*)  
**show** *?case*  
**proof** (*cases (u,v) = (x,y)*)  
**case** *True*  
**then have**  $v \in \text{marked } \sigma$   
**by** (*metis assem-ab assem-ln-3 assem-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**  $(u, v) \in hp \ \sigma'$   
**using** *assem-xyz assem-ln-3* **by** *auto*  
**moreover have**  $Q \ u$   
**using** *assem-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  $\bigwedge w. \text{reachable } \sigma \ w \implies Q \ w$ 
  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  $a \neq c \wedge b = z$ 
    using ln-ab-xy ln-c1 by blast

  ultimately have  $(\exists zw \in \text{hp } \sigma'. \text{fst } zw \in \text{reach } (\text{roots } \sigma') (\text{hp } \sigma') \vee$ 
     $\text{fst } zw \in \text{reach } (\text{marked } \sigma' \cap - s) (\text{hp } \sigma')) \wedge$ 
     $(\text{fst } zw \notin s \vee \text{fst } zw \notin \text{marked } \sigma') \wedge a \neq \text{fst } zw \wedge b = \text{snd } zw)$ 
    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  $\sigma \ \sigma' \wedge \text{post-change-remove } \sigma \ \sigma'$ 
    and all-ok  $s \ \sigma$ 
  shows can-be-marked  $s \ \sigma'$ 
  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  $\sigma \ \sigma'$ 
    and assm-all-ok: all-ok  $s \ \sigma$ 
    and assm-roots-marked:  $\text{roots } \sigma \subseteq \text{marked } \sigma$ 
  shows can-be-marked  $s \ \sigma'$ 
proof —
  from assm-step have post-change-remove  $\sigma \ \sigma' \vee \text{post-change-redirect } \sigma \ \sigma'$ 
    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)
  end
end

```

```

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  $\sigma$   $\sigma'$ 
    and assm-all-ok: all-ok  $s$   $\sigma$ 
    shows tbm-put  $s$   $\sigma'$ 
proof -
  have  $\bigwedge xy. xy \in hp\ \sigma' \implies tbm\ \sigma' \neq Some\ (snd\ xy) \implies xy \in hp\ \sigma$ 
  proof -
    fix  $xy$ 
    assume assm-local-hp:  $xy \in hp\ \sigma'$ 
      and assm-local-tbm:  $tbm\ \sigma' \neq Some\ (snd\ xy)$ 
    have post-change-remove  $\sigma$   $\sigma' \vee post-change-redirect\ \sigma\ \sigma'$ 
      using assm-step unfolding post-change-def by blast
    then show  $xy \in hp\ \sigma$ 
      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  $\sigma$   $\sigma'$ 
    and all-ok  $s$   $\sigma$ 
    and roots  $\sigma \subseteq marked\ \sigma$ 
    shows can-be-marked  $s$   $\sigma' \wedge tbm-put\ s\ \sigma'$ 
  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  $s$   $\sigma$ 
    and post-mark-tbm  $\sigma\ \sigma'$ 
    shows all-ok  $s$   $\sigma' \vee (marked\ \sigma \subset marked\ \sigma')$ 
  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  $\sigma$*  and  $q$  is *can-be-marked w  $\sigma' \vee \text{marked } \sigma \subset \text{marked } \sigma'$* .

**lemma** *works-for-all-w*:

**assumes** *post-change  $\sigma \sigma' \vee \text{post-mark-tbm } \sigma \sigma'$*   
**and** *inv-roots  $\sigma$*   
**and** *all-ok w  $\sigma$*   
**shows** *can-be-marked w  $\sigma' \vee \text{marked } \sigma \subset \text{marked } \sigma'$*   
**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  $\sigma \sigma' \vee \text{post-mark-tbm } \sigma \sigma'$*   
**and** *inv-roots  $\sigma$*   
**and** *all-ok w  $\sigma$*   
**shows** *marked  $\sigma \subseteq \text{marked } \sigma' \wedge \text{localCvar } \sigma = \text{localCvar } \sigma' \wedge$*   
*(can-be-marked w  $\sigma' \vee \text{marked } \sigma \subset \text{marked } \sigma'$ )*

**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  $\sigma \sigma' \vee \text{post-mark-tbm } \sigma \sigma'$*   
**and** *all-ok s  $\sigma$*   
**and**  $\forall w. ((\text{post-change } \sigma \sigma' \vee \text{post-mark-tbm } \sigma \sigma') \wedge \text{all-ok w } \sigma)$   
 $\longrightarrow (\text{marked } \sigma \subseteq \text{marked } \sigma' \wedge \text{localCvar } \sigma = \text{localCvar } \sigma' \wedge$   
 $(\text{can-be-marked w } \sigma' \vee \text{marked } \sigma \subset \text{marked } \sigma'))$   
**and** *all-ok s  $\sigma \wedge \text{marked } \sigma \subseteq \text{marked } \sigma' \wedge \text{localCvar } \sigma = \text{localCvar } \sigma' \wedge$*   
 $(\text{can-be-marked s } \sigma' \vee \text{marked } \sigma \subset \text{marked } \sigma')$   
 $\longrightarrow \text{rely-collector } \sigma \sigma'$

shows *rely-collector*  $\sigma \sigma'$   
 using *assms holds-for-all* **by** *force*

## 5.2 Propagate

**lemma** *chain-mono*:

$\llbracket \bigwedge m. f m \subseteq f (Suc m) ; a \leq b \rrbracket \implies f a \subseteq f b$   
 using *lift-Suc-mono-le* **by** *blast*

**lemma** *chain-anti*:

$\llbracket \bigwedge m. f (Suc m) \subseteq f m ; a \leq b \rrbracket \implies f b \subseteq f a$   
 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  $st :: nat \Rightarrow 'a \text{ state}$

assumes *step*:  $\bigwedge m. \text{rely-collector } (st m) (st (Suc m))$   
 $\vee \text{guar-collector } (st m) (st (Suc m))$

and *init-empty*:  $\text{safe } (localCvar (st 0)) = \{\}$   $\wedge$   
 $\text{consid } (localCvar (st 0)) = \{\}$

shows *post-propagate*  $(st 0) (st n)$

**proof** —

have *init-eq*:  $\text{safe } (localCvar (st 0)) = \text{consid } (localCvar (st 0))$

using *init-empty* **by** *simp*

have *ok0*:  $\text{all-ok } (\text{safe } (localCvar (st 0))) (st 0)$

using *init-empty all-ok-def* **by** *blast*

have *mark-step*:  $\bigwedge m. \text{marked } (st m) \subseteq \text{marked } (st (Suc m))$

**by** (*metis rely-collector-def guar-collector-def local.step*)

have *safe-step*:  $\bigwedge m. \text{safe } (localCvar (st (Suc m))) \subseteq \text{safe } (localCvar (st m))$

**by** (*metis rely-collector-def guar-collector-def local.step dual-order.refl*)

have  $\bigwedge x y. (x, y) \in hp (st 0) \implies x \in \text{marked } (st n) \implies$

$x \in \text{consid } (localCvar (st n)) \implies$

$y \in \text{marked } (st n) \vee \text{marked } (st 0) \subset \text{marked } (st n)$

**proof** —

fix  $x y$  **assume**  $(x, y) \in hp (st 0)$

**and**  $x \in \text{marked } (st n)$

**and** *local-assm3*:  $x \in \text{consid } (localCvar (st n))$

**show**  $y \in \text{marked } (st n) \vee \text{marked } (st 0) \subset \text{marked } (st n)$

**proof** —

{ **assume**  $\neg (\text{marked } (st 0) \subset \text{marked } (st n))$

**moreover** **have**  $\text{marked } (st 0) \subseteq \text{marked } (st n)$

**by** (*rule chain-mono*[*of*  $\lambda m. \text{marked } (st m)$ , *OF* *mark-step*, *of*  $0 n$ ]) *simp*

**ultimately** **have** *eqn*:  $\text{marked } (st 0) = \text{marked } (st n)$

**by** *simp*

— *marked* is constant on  $0..n$

```

have eqm:  $\bigwedge m. m \leq n \implies \text{marked } (st\ m) = \text{marked } (st\ 0)$ 
proof –
  fix m assume assm:  $m \leq n$ 
  then have  $\text{marked } (st\ m) \subseteq \text{marked } (st\ n)$ 
    by (rule chain-mono[of  $\lambda m. \text{marked } (st\ m)$ , OF mark-step, of m n])
  moreover have  $\text{marked } (st\ 0) \subseteq \text{marked } (st\ m)$ 
    apply (rule chain-mono[of  $\lambda m. \text{marked } (st\ m)$ , OF mark-step, of 0 m])
    by simp
  ultimately show  $\text{marked } (st\ m) = \text{marked } (st\ 0)$ 
    using eqn by blast
qed

— safe = consid holds at every state up to n, by induction from 0
have all-eq:  $\bigwedge m. m \leq n \implies$ 
   $\text{safe } (\text{localCvar } (st\ m)) = \text{consider } (\text{localCvar } (st\ m))$ 
proof –
  fix m show  $m \leq n \implies$ 
     $\text{safe } (\text{localCvar } (st\ m)) = \text{consider } (\text{localCvar } (st\ m))$ 
    apply (induction m)
    apply (rule init-eq)
    apply clarsimp
    by (metis rely-collector-def local.step guar-collector-def psubsetE)
qed

have  $\text{safe } (\text{localCvar } (st\ n)) = \text{consider } (\text{localCvar } (st\ n))$ 
  by (rule all-eq[OF order-refl])
moreover
have s-down:  $\text{safe } (\text{localCvar } (st\ n)) \subseteq \text{safe } (\text{localCvar } (st\ 0))$ 
  by (rule chain-anti[of  $\lambda m. \text{safe } (\text{localCvar } (st\ m))$ ,
    OF safe-step, of 0 n]) simp
ultimately have  $x \in \text{safe } (\text{localCvar } (st\ 0))$ 
  using local-assm3 by auto
then have  $y \in \text{marked } (st\ n)$ 
  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*  $\sigma\ \sigma'$

```

    and assm-marked:  $\text{marked } \sigma' = \text{marked } \sigma$ 
    and assm-roots:  $\text{roots } \sigma \subseteq \text{marked } \sigma$ 
    and assm-consid:  $\text{consid } (\text{localCvar } \sigma') = \text{Field } (\text{hp } \sigma)$ 
      — consid contains all the nodes in the heap.
    shows  $\forall w. \text{reachable } \sigma w \longrightarrow w \in \text{marked } \sigma$ 
  proof (intro allI impI)
    fix w assume reachable  $\sigma w$ 
    hence hw:  $w \in \text{reach } (\text{roots } \sigma) (\text{hp } \sigma)$  unfolding reachable-def .
    show  $w \in \text{marked } \sigma$ 
  proof (rule reach-induct[where  $P = \lambda u. u \in \text{marked } \sigma, OF hw$ ])
    fix u assume  $u \in \text{roots } \sigma$ 
    thus  $u \in \text{marked } \sigma$  using assm-roots by blast
  next
    fix u v
    assume  $u \in \text{marked } \sigma$  and  $(u, v) \in \text{hp } \sigma$ 
    moreover from this have  $u \in \text{marked } \sigma' \wedge u \in \text{consid } (\text{localCvar } \sigma')$ 
      using assm-marked Field-iff assm-consid by force
    ultimately show  $v \in \text{marked } \sigma$ 
      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](https://doi.org/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](https://doi.org/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](https://arxiv.org/abs/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.