An example of local reasoning in BI pointer logic: the Schorr-Waite graph marking algorithm

Hongseok Yang
University of Illinois at Urbana-Champaign and University of Birmingham
hyang@cs.uiuc.edu, H.Yang@cs.bham.ac.uk

January 6, 2001

1 Introduction

Handling pointers in a programming logic has been one of the most important and difficult problems in programming language research. The difficulty does not lie in coming up with any form of programming logic for pointer programs; in fact, there exist approaches either based on the "pointer as an index of an array" idea [3] or on the use of semantic-based substitutions [4, 1]. Instead, the core of the problem is to obtain a formalism which matches up with an informal reasoning by programmers and algorithm designers. That is, the verification of a program should be only as complicated as an informal correctness argument for the program. Most of the existing formalisms fail in this criteria: when they are used in verifying a pointer program, the verification based on them becomes significantly more complicated than an informal argument.

Only recently, Reynolds, Ishtiaq and O'Hearn [6, 2, 7] pointed out that one way to achieve such a match is to exploit the locality of memory access within a code fragment: usually for a given code fragment, only small number of cells on the heap are accessed. Ishtiaq and O'Hearn formulated a logic for pointer programs based on the logic of Bunched Implications [5] and proposed a rule which captures the locality:

Frame Introduction

$$\frac{\{\varphi\}\,C\,\{\psi\}}{\{\varphi * \chi\}\,C\,\{\psi * \chi\}}\ \text{(All stack variables assigned in } C \text{ don't appear in } \chi)$$

The rationale behind the rule is that if χ is a fact on memory cells which are not accessed by C, the truth of χ doesn't change before and after the execution of C. The use of $*$ in $- * \chi$, the side condition and the "tight" interpretation of a Hoare triple in [2] guarantee that χ, in fact, depends only on memory cells which are not accessed by C at all. With Frame Introduction, we can initially reason about a local property for a piece of code, which amounts to showing a Hoare triple, and then derive a global fact from the proved Hoare triple. O'Hearn calls this style of reasoning *local reasoning.*

The goal of this paper is to show the promise of the idea by one convincing example: the Schorr-Waite graph marking algorithm, which is a well-known test case for any programming logic for pointers because of the complex pointer manipulations in the algorithm. Since the algorithm mixes two different data structures by implementing one data structure inside the other, it is hard to capture its locality of memory access by other known measures in literatures such as reachability and different field names. The main point of the example is that even for such a program, the locality can still be exploited in the verification with BI pointer logic.

We use the implementation and the specification of the Schorr-Waite graph marking algorithm by Morris and Bornat. They are shown in Section 2. In Section 3, we prove the implementation satisfies the specification. Instead of showing all steps involved, the focus goes on showing the preservation of a loop invariant, which best illustrates the main benefit of BI pointer logic. The verification presented is semi-formal in the sense that we don't give all the detailed semantics of the assertion language and the programming language in this paper, which is slightly different from what is presented in [2].[1] However, we make it clear what is assumed. Throughout the paper, we assume that the reader is familiar with BI pointer logic in [2].

2 Marking Algorithm and Specification

The implementation for the Schorr-Waite graph marking algorithm in this section is from [1], which is again Morris' implementation in [4] with minor variation. Since the implementation is verified several times by others, the "semantic" elements of the proof, such as what pre- and post- conditions and loop invariant should mean semantically, are already known. In fact, in this section, these pre- and post- conditions are used to write a specification, and in the next section, the loop invariant is used in the verification. However, the ways to express semantic elements with (classical) BI formulas and to proceed a proof with BI pointer logic are significantly different from existing work. Especially, the proof in the paper shows that locality of memory access can be reflected in the verification. We show the first point in this section, while delaying the more important second one to the next section.

2.1 Code

The program below marks all nodes reachable from *root*, which are initially unmarked. It is essentially the depth first traversal with clever implementation of a stack: a stack is implemented using pointer reversals.

The language we are using is slightly different from what appeared in [2]: it is a typed version of that in [2]. Each cell in the heap supposedly has four fields; the first two fields contain only links to other nodes, thereby having *location* type, and the last two fields have only boolean values, thereby having *bool* type. For any cell, the third field of the cell denotes which one of *left* and *right* fields is reversed, and the fourth field indicates whether the cell has been visited or not. In the program, we use *left*, *right*, *check* and *mark* to denote 1st, 2nd, 3rd and 4th fields of a memory cell, respectively. All other stack variables have either *bool* type or *location* type.

```
GRAPH-MARKING(root : location)
local
  t, p, pL, q : location
  pC, tM      : boolean
in
  t := root;
  p := nil;
  if (t <> nil) then (tM := t.mark) else (tM := true);
  while ((p <> nil) or ((t <> nil) and not(tM)))
     do
```

[1]The formal semantics is a straightforward modification of that in [2] and will appear in the author's thesis.

```
        if not((t <> nil) and not(tM))
        then pC := p.check;
             if (pC) then q := t;
                          t := p;
                          p:= p.right;
                          t.right := q
                     else q := t;
                          t := p.right;
                          pL := p.left;
                          p.right := pL;
                          p.left := q;
                          p.check := true
        else q := p;
             p := t;
             t := t.left;
             p.left := q;
             p.mark := true;
             p.check := false;
        if (t <> nil) then (tM := t.mark) else (tM := true)
end
```

2.2 Specification

Any reasonable specification for the Schorr-Waite graph marking algorithm should say that if all nodes reachable from *root* are initially unmarked, after executing `GRAPH-MAKRING(root)`, all nodes initially reachable from *root* get marked but only their *mark* and *check* fields are changed. The specification in this section follows this common pattern with slight difference in two aspects: in precondition, we require that all pointers stored in a cell or in the stack variable `root` are not dangling and that all existing cells are reachable from `root`. Even if the second aspect may seem strange to some readers, it reflects the idea of so-called "tight" specification: a specification mentions only those memory cells in concern. In fact, the "full" specification which additionally specifies that all unreachable cells are unchanged follows from the "tight" one by Frame Introduction: if the implementation for the algorithm satisfies the "tight" specification, any properties of unreachable cells can't change because of Frame Introduction.

We present a specification using a Hoare triple with BI formulas as assertions. The assertion language in this section is slightly different from that in [2] in that it is equipped with sorts (*bool*, *location*, *tree* and two different lists, *blist* and *lblist*); *tree* denotes a finite binary tree whose nodes and leaves contain a value of *location* type, *blist* a finite sequence of boolean values and *lblist* a finite sequence of 4-tuples with three locations and one boolean value. The formal semantics of the sorted assertion language is not presented in the paper.[2] We usually omit the annotation of sorts in assertions.

Before giving a specification for the program, we "define" inductive predicates which state certain properties about finite lists, finite trees and the heap storage. When all cells on the heap are relevant to the truth of a predicate, we add R at the end of the name of the predicate to indicate this. The letter R stands for "relevant", and more technically a predicate with R at the end of its name indicates that the predicate is not intuitionistic[3]. We use the predicate $\hookrightarrow$ and the abbreviation

[2] The semantics will appear in the author's thesis.

[3] A predicate P is intuitionistic iff for any s, h, h', $s, h \models P$ and $h \sqsubseteq h'$ imply that $s, h' \models P$

?. The predicate $\hookrightarrow$ is an intuitionistic version of the $\mapsto$ predicate: $(x \hookrightarrow l, r, c, m)$ is equivalent to $(x \mapsto l, r, c, m) * \top$. And ? either in $\mapsto$ or in $\hookrightarrow$ stands for an existentially quantified variable. For instance, the formulas $(x \mapsto ?, ?, c, m)$ and $(x \hookrightarrow l, r, ?, ?)$ are abbreviations of $\exists l, r.\, (x \mapsto l, r, c, m)$ and $\exists c, m.\, (x \hookrightarrow l, r, c, m)$, respectively.

The predicate $\mathit{listMarkedNodesR}(l, p)$ means that p denotes the head of a "linked" list of marked nodes which "implements" l: by following the reversed pointers from p upto nil, a finite linked list of marked nodes should be obtained, and each tuple in the finite list l has to consist of the address and *left*, *right* and *check* fields of the corresponding node in the linked list. As the postfix R indicates, the predicate also requires that all allocated cells should participate in the implementation. In the specification below, we use the predicate to state that p denotes the head of a stack, whose content is l.

$$
\begin{aligned}
&\mathit{listMarkedNodesR}(\mathit{nil}, p)\\
&\qquad \equiv (p = \mathit{nil}) \wedge \mathsf{I}\\
&\mathit{listMarkedNodesR}((n, l, r, \mathit{true}) :: \mathit{lbtl}, p)\\
&\qquad \equiv (n = p) \wedge (p \mapsto l, r, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{lbtl}, r)\\
&\mathit{listMarkedNodesR}((n, l, r, \mathit{false}) :: \mathit{lbtl}, p)\\
&\qquad \equiv (n = p) \wedge (p \mapsto l, r, \mathit{false}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{lbtl}, l)
\end{aligned}
$$

The predicate $\mathit{listMarkedNodesR}(l, \mathit{prev})$ is intended to denote that there is a linked list ending with prev on the heap which represents the "restored" version of the list l: when either the second or the third value of each tuple in l is "restored back appropriately" and the list itself is reversed, the first three values of each tuple in the resulting list consists of the address and *left* and *right* fields of the corresponding node in the linked list. And all allocated cells should be a part of the linked list. In the specification below, we use the predicate to denote the heap which is obtained by restoring back the reversed pointers in the stack.

$$
\begin{aligned}
&\mathit{restoredListR}(\mathit{nil}, \mathit{prev}) \equiv \mathsf{I}\\
&\mathit{restoredListR}((n, l, r, \mathit{true}) :: \mathit{lbtl}, \mathit{prev}) \equiv (n \mapsto l, \mathit{prev}, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{lbtl}, n)\\
&\mathit{restoredListR}((n, l, r, \mathit{false}) :: \mathit{lbtl}, \mathit{prev}) \equiv (n \mapsto \mathit{prev}, r, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{lbtl}, n)
\end{aligned}
$$

The predicate $\mathit{spansR}(\mathit{STree}, x)$ denotes that a tree STree spans all reachable nodes from x and that moreover all allocated nodes on the heap are reachable from x.

$$
\begin{aligned}
&\mathit{spansR}(L(n), t)\\
&\qquad \equiv (n = t) \wedge \mathsf{I}\\
&\mathit{spansR}(N(n, \mathit{ltree}, \mathit{rtree}), t)\\
&\qquad \equiv (n = t) \wedge (\exists l, r.\, (t \mapsto l, r, ?, ?) * \mathit{spansR}(\mathit{ltree}, l) * \mathit{spansR}(\mathit{rtree}, r))
\end{aligned}
$$

The predicate $\mathit{reach}(y, x)$ denotes that there is a path of nodes on the heap from the node y to the node x. And $\mathit{reachRightChildInList}(l, x)$ means that the node x is reachable from the right child of a node in the list l.

$$
\mathit{reach}(y, x) \equiv \exists \mathit{path}.\, \mathit{reachByPathR}(y, x, \mathit{path}) * \top
$$

$$
\begin{aligned}
&\mathit{reachByPathR}(y, x, \mathit{nil}) \equiv (y = x) \wedge \mathsf{I}\\
&\mathit{reachByPathR}(y, x, \mathit{false} :: \mathit{btl}) \equiv \exists \mathit{yl}, \mathit{yr}.\, (y \mapsto \mathit{yl}, \mathit{yr}, ?, ?) * \mathit{reachByPathR}(\mathit{yl}, x, \mathit{btl})\\
&\mathit{reachByPathR}(y, x, \mathit{true} :: \mathit{btl}) \equiv \exists \mathit{yl}, \mathit{yr}.\, (y \mapsto \mathit{yl}, \mathit{yr}, ?, ?) * \mathit{reachByPathR}(\mathit{yr}, x, \mathit{btl})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{reachRightChildInList}(\mathit{nil}, x) \equiv \bot\\
&\mathit{reachRightChildInList}((n, l, r, c) :: \mathit{lbtl}, x) \equiv \mathit{reach}(r, x) \vee \mathit{reachRightChildInList}(\mathit{tl}, x)
\end{aligned}
$$

The predicate $allocated(x)$ denotes that x denotes an allocated cell, $markedR$ that all allocated cells are marked and $unmarkedR$ that all allocated cells are unmarked.

$$allocated(x) \equiv (x \hookrightarrow ?, ?, ?, ?)$$

$$markedR \equiv \forall x.\, allocated(x) \rightarrow (x \hookrightarrow ?, ?, ?, true)$$

$$unmarkedR \equiv \forall x.\, allocated(x) \rightarrow (x \hookrightarrow ?, ?, ?, false)$$

The following two predicates are about dangling pointers; $noDangling(x)$ says that x is not a dangling pointer and $noDanglingR$ that no dangling pointers are stored in memory cells on the heap.

$$noDangling(x) \equiv (x = nil) \vee allocated(x)$$

$$noDanglingR \equiv \forall x, l, r.\, (x \hookrightarrow l, r, ?, ?) \rightarrow noDangling(l) \wedge noDangling(r)$$

Stating the specification mentioned in the beginning of this section with BI formulas is now straightforward except the way to say that all nodes reachable from *root* have the same values for *left* and *right* fields before and after the execution. We use a spanning tree *STree* for the graph of all reachable nodes to store values of *left* and *right* fields of all reachable nodes.

$$\{spansR(STree, root) \wedge unmarkedR \wedge noDanglingR \wedge noDangling(root)\}$$
$$\texttt{GRAPH-MARKING(root)}$$
$$\{spansR(STree, root) \wedge markedR\}$$

Informally, the specification says that if before executing `GRAPH-MARKING(root)`,

- all allocated cells are reachable from *root* and $STree$ is a spanning tree of those allocated cells;
- all allocated cells are not marked;
- there are no dangling pointers stored in any cells; and
- *root* doesn't store a dangling pointer,

then after the execution,

- $STree$ is still a spanning tree of all allocated cells; and
- all allocated cells are marked.

Notice that for two heaps h and h', if they have the same $STree$ as a spanning tree, h and h' have the same allocated cells and the contents of those allocated cells can be different only in the third and fourth fields. Therefore, the mere fact that $STree$ is a spanning tree before and after the execution implies that no new cells are allocated or deallocated during the execution and that the *left* and *right* fields of all cells have the same values before and after the execution.

3 Verification

In this section, we use BI pointer logic to verify that the implementation of the Schorr-Waite graph marking in Section 2 satisfies the specification. The main focus is on the benefit of local reasoning in verification. The verification shows that with local reasoning, the verification becomes significantly simpler in the sense that it is only as complicated as an informal argument. Instead of presenting all the details of the proof, only one piece, which best illustrates local reasoning, is shown.

The proof in this section uses three rules which don't come with a particular language construct.[4] The most important one, Frame Introduction, is from [2] and makes it possible to change a local fact for a code fragment into a global fact by incorporating other "unmentioned" cells.

Frame Introduction

$$\frac{\{\varphi\}\, C\, \{\psi\}}{\{\varphi * \chi\}\, C\, \{\psi * \chi\}} \text{ (Any assignment in } C \text{ doesn't write to } FV(\chi))$$

Auxiliary variables are used frequently in order to state the content of stack variables and the status of the heap before executing a command. And at certain point during the verification, these extra variables should be eliminated from assertions. Auxiliary Elimination is the rule to handle this situation:

Auxiliary Elimination

$$\frac{\{\varphi\}\, C\, \{\psi\}}{\{\exists X.\, \varphi\}\, C\, \{\exists X.\, \psi\}} \text{ (} X \text{ doesn't occur in } C\text{)}$$

The final rule is for slitting states satisfying a precondition into several cases.

Case Analysis

$$\frac{\{\varphi\}\, C\, \{\psi\} \qquad \{\chi\}\, C\, \{\psi\}}{\{\varphi \vee \chi\}\, C\, \{\psi\}}$$

[4]The soundness of the rules will appear in the author's thesis with the semantics of the BI assertion language in this paper.

3.1 Outline

The following code fragment with assertions inserted gives the outline of the proof:

$$\{Precondition\}$$

```
t := root;
p := nil;
```

$$\{\exists Stack.\ preLoopInvR(Stack, p, t, STree, root)\}$$

```
if (t <> nil) then (tM := t.mark) else (tM := true);
```

$$\{(\exists Stack.\ preLoopInvR(Stack, p, t, STree, root)) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM))\}$$

```
while ((p <> nil) or ((t <> nil) and not(tM)))
   do
      if not((t <> nil) and not(tM))
      then pC := p.check;
           if (pC) then q := t;
                        t := p;
                        p:= p.right;
                        t.right := q
                   else q := t;
                        t := p.right;
                        pL := p.left;
                        p.right := pL;
                        p.left := q;
                        p.check := true
      else q := p;
           p := t;
           t := t.left;
           p.left := q;
           p.mark := true;
           p.check := false;
```

$$\{\exists Stack.\ preLoopInvR(Stack, p, t, STree, root)\}$$

```
      if (t <> nil) then (tM := t.mark) else (tM := true)
```

$$\{(\exists Stack.\ preLoopInvR(Stack, p, t, STree, root)) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM))\}$$

$$\{Postcondition\}$$

where *Precondition* and *Postcondition* are macros for pre- and post- conditions in Section 2:

$$Precondition \equiv spansR(STree, root) \wedge unmarkedR \wedge noDanglingR \wedge noDangling(root)$$
$$Postcondition \equiv spansR(STree, root) \wedge markedR$$

and the predicate *preLoopInvR*, which is the main part of the loop invariant, is defined as follows:

$$\begin{aligned}
&preLoopInvR(Stack, P, T, STree, root)\\
&\equiv\ noDanglingR \wedge noDangling(T) \wedge noDangling(P)\\
&\quad \wedge listMarkedNodesR(Stack, P) * (restoredListR(Stack, T) \mathrel{-\!\!*} spansR(STree, root))\\
&\quad \wedge markedR * \begin{pmatrix} unmarkedR \\ \wedge (\forall x.\ allocated(x) \rightarrow (reach(T, x) \vee reachRightChildInList(Stack, x))) \end{pmatrix}
\end{aligned}$$

Informally, $preLoopInvR(Stack, P, T, STree, root)$ says that

- P and T are not dangling, and all allocated cells on the heap don't store dangling pointers;

- P is the head of the stack implemented by pointer reversals, *Stack* represents the content of the stack and all nodes in the stack are marked;

- if the reversed links of all nodes in the stack are restored back, where the first reversed link is restored to T, then all link fields, i.e., *left* and *right* fields, of every node on the heap have initial values, which is stored in $STree$; and

- all unmarked allocated nodes are reachable from T or the *right* field of a node stored in the stack (*Stack*) going only though unmarked nodes.

Notice that $\mathit{listMarkedNodesR}(\mathit{Stack}, P) * (\mathit{restoredListR}(\mathit{Stack}, T) \mathrel{-\!\!*} \mathit{spansR}(STree, root))$ expresses the second and the third points at the same time in a natural way using $*$ and $\mathrel{-\!\!*}$. It states that the heap can be divided into two parts; the first part of the heap implements the stack with p as its head node; if the stack is taken away from the heap, thereby leaving only the second part, and then the stack with all reversed links restored back is added, *left* and *right* fields of all allocated nodes have initial values stored in the spanning tree $STree$.

Let `init`, `body` and `settM` denote the following code fragments:

```
init ≡ t := root;
       p := nil;

body ≡ if not((t <> nil) and not(tM))
       then pC := p.check;
            if (pC) then q := t;
                         t := p;
                         p:= p.right;
                         t.right := q
                    else q := t;
                         t := p.right;
                         pL := p.left;
                         p.right := pL;
                         p.left := q;
                         p.check := true
       else q := p;
            p := t;
            t := t.left;
            p.left := q;
            p.mark := true;
            p.check := false;

settM ≡  if (t <> nil) then (tM := t.mark) else (tM := true)
```

Let *Cond* denote the condition of the while loop in the program. Then, it suffices to show the following three:

- the loop invariant is established initially;

$$\{\mathit{Precondition}\}\ \texttt{init}; \texttt{settM} \left\{ \begin{array}{l} (\exists \mathit{Stack}.\ \mathit{preLoopInvR}(\mathit{Stack}, p, t, STree, root)) \\ \wedge\ (t = \mathit{nil} \vee (t \hookrightarrow ?, ?, ?, tM)) \end{array} \right\}$$

- the loop invariant is preserved by the loop body; and

$$\begin{array}{c}\{(\exists Stack.\, preLoopInvR(Stack, p, t, STree, root)) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM)) \wedge Cond\} \\ \texttt{body; settM} \\ \{(\exists Stack.\, preLoopInvR(Stack, p, t, STree, root)) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM))\}\end{array}$$

- the loop invariant implies the postcondition.

$$\begin{pmatrix}(\exists Stack.\, preLoopInvR(Stack, p, t, STree, root)) \\ \wedge\, (t = nil \vee (t \hookrightarrow ?, ?, ?, tM)) \\ \wedge\, \neg(Cond)\end{pmatrix} \implies Postcondition$$

By the rule for sequencing, the above three can be derived from the following three Hoare triples and one implication of assertions.

- $\{Precondition\}$ `init` $\{\exists Stack.\, preLoopInvR(Stack, p, t, STree, root)\}$
- $\begin{Bmatrix}\exists Stack. \\ preLoopInvR(Stack, p, t, STree, root)\end{Bmatrix}$ `settM` $\begin{Bmatrix}\begin{pmatrix}\exists Stack. \\ preLoopInvR(Stack, p, t, STree, root)\end{pmatrix} \\ \wedge(t = nil \vee (t \hookrightarrow ?, ?, ?, tM))\end{Bmatrix}$
- $\begin{Bmatrix}\begin{pmatrix}\exists Stack. \\ preLoopInvR(Stack, p, t, STree, root)\end{pmatrix} \\ \wedge\, (t = nil \vee (t \hookrightarrow ?, ?, ?, tM)) \\ \wedge\, Cond\end{Bmatrix}$ `body` $\begin{Bmatrix}\exists Stack. \\ preLoopInvR(Stack, p, t, STree, root)\end{Bmatrix}$
- $\begin{pmatrix}(\exists Stack.\, preLoopInvR(Stack, p, t, STree, root)) \\ \wedge\, (t = nil \vee (t \hookrightarrow ?, ?, ?, tM)) \\ \wedge\, \neg(Cond)\end{pmatrix}$ implies $Postcondition$.

Since it is the third case that shows clearly the benefit of local reasoning, we handle only the third one.[5]

3.2 Verification of the correctness of `body`

One common way to informally show the correctness of a program follows three steps: to classify legal input states into certain cases, to compute the program in each case so as to obtain the state change and finally to show that for each case, the changes of memory cells and stack variables, caused by the program, produce legal output states. Before starting the formal verification, we present such an informal correctness argument for the Hoare triple for `body`. During the argument, we point out that all important steps in the informal argument is expressible by implications between assertions.

Informally, the states satisfying the loop invariant and *Cond* can be divided into three cases. The first case is that the current node, denoted by t, is either *nil* or marked, the stack is not empty and the right child of the top node, which is pointed to by p, in the stack is currently being traversed. The second case is the same same as the first one except that the left child of the top node is being traversed. The third case is the remaining possibility: the current node is neither *nil* nor marked. Since either the *left* field or the *right* field is reversed depending on which child is currently being traversed and all nodes in the stack are marked by the loop invariant, the three cases become more specialized as follows:

[5]The complete proof will be presented in the author's thesis.

- the current node is either *nil* or marked, the *right* field of the top node is reversed and the top node is marked;

$$(t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge (tM = true)) \wedge (p \hookrightarrow ?, ?, true, true)$$

- the current node is either *nil* or marked, the *left* field of the top node is reversed and the top node is marked;

$$(t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge (tM = true)) \wedge (p \hookrightarrow ?, ?, false, true)$$

- the current node is neither *nil* nor marked.

$$(t \hookrightarrow ?, ?, ?, false) \wedge (tM = false)$$

The following lemma states that the informal reasoning, in fact, holds.

Lemma 1 (Case Splitting) When all implications in Appendix B.1 hold in the semantics, the assertion

$$(\exists Stack.\, preLoopInvR(Stack, p, t, STree, root)) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM)) \wedge Cond$$

implies

$$\begin{aligned}&\Big((t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge (tM = true)) \wedge (p \hookrightarrow ?, ?, true, true)\Big)\\ \vee &\Big((t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge (tM = true)) \wedge (p \hookrightarrow ?, ?, false, true)\Big)\\ \vee &\Big((t \hookrightarrow ?, ?, ?, false) \wedge (tM = false)\Big)\end{aligned}$$

Proof. The proof is shown in Appendix A.1. □

Each of the three cases takes a different branch in `body`, which makes it easy to figure out the state changes in each case. When P and T denote the initial values of p and t respectively, `body` changes the store for each case as follows:

- in the first case, the top node in the stack is popped and p and t are changed appropriately: the cell $(P \mapsto PL, PR, true, true)$ and two stack variables p and t are updated to $(P \mapsto PL, T, true, true)$, PR and P, respectively;

- in the second case, the right child of the top node becomes the current node and link fields and check field of the top node are changed appropriately: the cell $(P \mapsto PL, PR, false, true)$ and two stack variables p and t are updated to $(P \mapsto T, PL, true, true)$, P and PR, respectively; and

- in the third case, the current node is pushed into the stack and stack variables p and t are changed appropriately: the cell $(T \mapsto TL, TR, TC, false)$ and two stack variables p and t are updated to $(T \mapsto P, TR, false, true)$, T and TL, respectively.

The last step in an informal reasoning is to argue that for each case, the state change doesn't result in an output state which invalidates the postcondition. For instance, in the first case, it is necessary to show that when the cell $(P \mapsto PL, PR, true, true)$ and two stack variables p and t are updated to $(P \mapsto PL, T, true, true)$, PR and P, respectively, the resulting state satisfies the postcondition as long as the input state satisfies the precondition. With BI multiplicative connectives, this informal reasoning is expressible by implications between assertions. The key point is that $(x \mapsto l, r, c, m) * ((x \mapsto l', r', c', m') \mathrel{-\!\!*} C)$ denotes that if the cell $(x \mapsto l, r, c, m)$ is updated to $(x \mapsto l', r', c', m')$, then C holds. Recall that in the predicate *preLoopInvR*, the first argument is meant to denote the content of the stack, the second the address of the top node in the stack and the third the address of the current node and that p and t are stack variables which keep track of addresses of the top node in the stack and the current node, respectively. The following three lemmas express that for each case, the state change doesn't produce an "illegal" output:

Lemma 2 When all implications in Appendix B.1 hold in the semantics, the assertion

$$\begin{aligned}&\mathit{preLoopInvR}((P, PL, PR, \mathit{true}) :: \mathit{Stack}, P, T, STree, \mathit{root})\\ &\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))\\ &\wedge (P \hookrightarrow PL, PR, \mathit{true}, \mathit{true})\end{aligned}$$

implies

$$\begin{aligned}&(P \mapsto PL, PR, \mathit{true}, \mathit{true})\\ &* ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{preLoopInvR}(\mathit{Stack}, PR, P, STree, \mathit{root}))\end{aligned}$$

Proof. The proof is shown in Appendix A.2. □

Lemma 3 When all implications in Appendix B.1 hold in the semantics, the assertion

$$\begin{aligned}&\mathit{preLoopInvR}((P, PL, PR, \mathit{false}) :: \mathit{Stack}, P, T, STree, \mathit{root})\\ &\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))\\ &\wedge (P \hookrightarrow PL, PR, \mathit{false}, \mathit{true})\end{aligned}$$

implies

$$\begin{aligned}&(P \mapsto PL, PR, \mathit{false}, \mathit{true})\\ &* ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{preLoopInvR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, P, PR, STree, \mathit{root}))\end{aligned}$$

Proof. The proof is shown in Appendix A.3. □

Lemma 4 When all implications in Appendix B.1 hold in the semantics, the assertion

$$\mathit{preLoopInvR}(\mathit{Stack}, P, T, STree, \mathit{root}) \wedge (T \hookrightarrow TL, TR, TC, \mathit{true})$$

implies

$$\begin{aligned}&(T \mapsto TL, TR, TC, \mathit{true})\\ &* ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: \mathit{Stack}, T, TL, STree, \mathit{root}))\end{aligned}$$

Proof. The proof is shown in Appendix A.4. $\square$

In the following sections, we formally prove that if an initial state satisfies the loop invariant and the execution of `body` terminates, the resulting state satisfies the postcondition. Using Lemma 1, the initial states are split into three cases, and we show the following three Hoare triples:

$$\left\{\begin{array}{l}\left(\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right)\\ \wedge\,(t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge (tM = true))\\ \wedge\,(p \hookrightarrow ?, ?, true, true)\end{array}\right\} \texttt{body} \left\{\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right\}$$

$$\left\{\begin{array}{l}\left(\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right)\\ \wedge\,(t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge tM = true)\\ \wedge\,(p \hookrightarrow ?, ?, false, true)\end{array}\right\} \texttt{body} \left\{\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right\}$$

$$\left\{\begin{array}{l}\left(\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right)\\ \wedge\,(tM = false) \wedge (t \hookrightarrow ?, ?, ?, false))\end{array}\right\} \texttt{body} \left\{\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right\}$$

Handing each case is the main part of the paper For all three cases, the verification has the same format: there are two stages, one for local reasoning and the other for changing a local fact into a global fact. More concretely, in the first stage, we show a certain "local" specification, given by a Hoare triple, holds for `body`, where the specification mentions only those cells accessed by `body`. In the next stage, we use this "local" fact, given by a Hoare triple, to derive another Hoare triple, which is what we originally intended to show and involves all other unaccessed cells in the code fragment. With the three lemmas (Lemma 2, 3 and 4), which are the main part of an informal argument, the change from a local fact to a global fact becomes straightforward, which supports the claim in the paper: with local reasoning, the formal verification becomes only as complicated as an informal one.

3.2.1 The first case

The Hoare triple, which corresponds to this case, is the following:

$$\left\{\begin{array}{l}\left(\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right)\\ \wedge\,(t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge (tM = true))\\ \wedge\,(\exists L, R.\,(p \hookrightarrow L, R, true, true))\end{array}\right\} \texttt{body} \left\{\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right\}$$

When p denotes an allocated, checked and marked cell and $(t = nil \vee tM = true)$ holds, memory access in `body` shows locality: `body` accesses only the cell initially pointed to by p. We exploit this locality by first reasoning about a certain local fact for the code fragment and using it to derive the original Hoare triple. The following Hoare triple describes such a local fact:

$$\left\{\begin{array}{l}(P \mapsto PL, PR, true, true)\\ \wedge\,(t = nil \vee tM = true)\\ \wedge\,(p = P) \wedge (t = T)\end{array}\right\} \texttt{body} \left\{\begin{array}{l}(P \mapsto PL, T, true, true)\\ \wedge\,(p = PR) \wedge (t = P)\end{array}\right\}$$

Notice that in any states satisfying the precondition in the above Hoare triple, there should be only one allocated cell, pointed to by p, on the heap and that the specification describes how the cell and two stack variables are changed. In a certain sense, the Hoare triple "exactly and only" describes the behavior of `body` when p denotes an allocated, checked and marked cell and $(t = nil \vee tM = true)$ holds. We call the original Hoare triple *global* and the new one in the above *local*. Verifying the local Hoare triple is straightforward as its proof outline below shows:

$$\{(P \mapsto PL, PR, true, true) \wedge (t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)\}$$

```
if not((t <> nil) and not(tM))
then pC := p.check;
     if (pC) then
```

$$\{(P \mapsto PL, PR, true, true) \wedge (p = P) \wedge (t = T)\}$$

```
               q := t;
               t := p;
               p:= p.right;
               t.right := q
          else
```

$$\{\bot\}$$

```
               q := t;
               t := p.right;
               pL := p.left;
               p.right := pL;
               p.left := q;
               p.check := true
else
```

$$\{\bot\}$$

```
     q := p;
     p := t;
     t := t.left;
     p.left := q;
     p.mark := true;
     p.check := false;
```

$$\{(P \mapsto PL, T, true, true) \wedge (p = PR) \wedge (t = P)\}$$

The crucial step of the proof is to change the mode of reasoning from local to global, that is, to derive the global Hoare triple from the local one. It is Frame Introduction and multiplicative connectives $*, -\!\!*$ of BI that make the change possible, as shown below:

$$\frac{\left\{\begin{array}{l}(P \mapsto PL, PR, true, true)\\ \wedge (t = nil \vee tM = true)\\ \wedge (p = P) \wedge (t = T)\end{array}\right\} \texttt{body} \left\{\begin{array}{l}(P \mapsto PL, T, true, true)\\ \wedge (p = PR) \wedge (t = P)\end{array}\right\}}{\left\{\begin{pmatrix}(P \mapsto PL, PR, true, true)\\ \wedge (t = nil \vee tM = true)\\ \wedge (p = P) \wedge (t = T)\end{pmatrix} * ((P \mapsto PL, T, true, true) -\!\!* \, preLoopInvR(Stack', PR, P, STree, root))\right\}}$$

$$\texttt{body}$$

$$\left\{\begin{pmatrix}(P \mapsto PL, T, true, true)\\ \wedge (p = PR) \wedge (t = P)\end{pmatrix} * ((P \mapsto PL, T, true, true) -\!\!* \, preLoopInvR(Stack', PR, P, STree, root))\right\}$$

The last remaining step is to obtain the original global Hoare triple from the above one, which requires to eliminate extra auxiliary variables and to show certain implications between assertions. Auxiliary variables P,PL,PR,T and $Stack'$ can be eliminated easily by applying Auxiliary Elimination:

$$\dfrac{\vdots}{\left\{\begin{array}{l}\exists P, PL, PR, T, Stack'.\\ \left(\begin{array}{l}(P \mapsto PL, PR, true, true)\\ \wedge (t = nil \vee tM = true)\\ \wedge (p = P) \wedge (t = T)\end{array}\right) * ((P \mapsto PL, T, true, true) -\!\!* \; preLoopInvR(Stack', PR, P, STree, root))\end{array}\right\}}$$

`body`

$$\left\{\begin{array}{l}\exists P, PL, PR, T, Stack'.\\ \left(\begin{array}{l}(P \mapsto PL, T, true, true)\\ \wedge (p = PR) \wedge (t = P)\end{array}\right) * ((P \mapsto PL, T, true, true) -\!\!* \; preLoopInvR(Stack', PR, P, STree, root))\end{array}\right\}$$

In order to apply Consequence to obtain the original global Hoare triple from the above one, the two implications between assertions need to be shown: one from the postcondition of the above Hoare triple to that of the global one and the other from the precondition of the global to that of the above. The following sequence of implications shows the first implication. We use certain properties of assertions in Appendix B. The notation $\vdash$ is used to indicate provability in the BI logic or in the first-order classical logic and $\models$ to indicate an semantic implication valid in the BI pointer model.

$$\begin{array}{l}\exists P, PL, PR, T, Stack'.\\ \left(\begin{array}{l}(P \mapsto PL, T, true, true)\\ \wedge (p = PR) \wedge (t = P)\end{array}\right) * ((P \mapsto PL, T, true, true) -\!\!* \; preLoopInvR(Stack', PR, P, STree, root))\end{array}$$

$\Longrightarrow$ Since $(p = PR) \wedge (t = P)$ is pure, by Lemma 6 in Appendix B.2,

$$\begin{array}{l}\exists P, PL, PR, T, Stack'.\\ (P \mapsto PL, T, true, true) * ((P \mapsto PL, T, true, true) -\!\!* \; preLoopInvR(Stack', PR, P, STree, root))\\ \wedge (p = PR) \wedge (t = P)\end{array}$$

$\Longrightarrow$ $\varphi * (\varphi -\!\!* \psi) \vdash \psi$

$$\exists P, PL, PR, T, Stack'.\; preLoopInvR(Stack', PR, P, STree, root) \wedge (p = PR) \wedge (t = P)$$

$\Longrightarrow$

$$\exists Stack'.\; preLoopInvR(Stack', p, t, STree, root)$$

The final step is to show the implication between preconditions, which is shown in the following.

$$\begin{array}{l}(\exists Stack.\; preLoopInvR(Stack, p, t, STree, root))\\ \wedge (\exists PL, PR.\, (p \hookrightarrow PL, PR, true, true))\\ \wedge (t = nil \vee (t \mapsto ?, ?, ?, true) \wedge tM = true)\end{array}$$

$\Longrightarrow$

$$\begin{array}{l}\exists Stack, PL, PR.\\ preLoopInvR(Stack, p, t, STree, root))\\ \wedge (p \hookrightarrow PL, PR, true, true)\\ \wedge (t = nil \vee (t \mapsto ?, ?, ?, true) \wedge tM = true)\end{array}$$

$\Longrightarrow$ $preLoopInvR(Stack, p, t, STree, root) \vdash restoredListR(Stack, p) * \top$ and Rule 2 in Appendix B.1, which is
$[\ (P \hookrightarrow PL, PR, true, true) \wedge listMarkedNodesR(Stack, P) * \top \models \exists Stack'.\, Stack = (P, PL, PR, true) :: Stack'\ ]$

$$\begin{array}{l}\exists Stack, PL, PR, P, T, Stack'.\\ preLoopInvR(Stack, p, t, STree, root))\\ \wedge (p \hookrightarrow PL, PR, true, true)\\ \wedge (t = nil \vee (t \mapsto ?, ?, ?, true) \wedge tM = true)\\ \wedge (Stack = (PL, PR, true, true) :: Stack') \wedge (p = P) \wedge (t = T)\end{array}$$

$\Longrightarrow$

$$\begin{array}{l}\exists PL, PR, P, T, Stack'.\\ preLoopInvR((PL, PR, true, true) :: Stack', P, T, STree, root))\\ \wedge (P \hookrightarrow PL, PR, true, true)\\ \wedge (T = nil \vee (T \mapsto ?, ?, ?, true) \wedge tM = true)\\ \wedge (p = P) \wedge (t = T)\end{array}$$

$\Longrightarrow$ Lemma 2

$$\begin{array}{l}\exists Stack, P, T, PL, PR.\\ (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) \mathrel{-\!\!*} preLoopInvR(Stack', PR, P, STree, root))\\ \wedge (T = nil \vee tM = true)\\ \wedge (p = P) \wedge (t = T)\end{array}$$

$\Longrightarrow$ Since $(t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)$ is pure, by Lemma 6 in Appendix B.2,

$$\begin{array}{l}\exists PL, PR, P, T, Stack'.\\ \begin{pmatrix}(P \mapsto PL, PR, true, true)\\ \wedge (t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)\end{pmatrix} * ((P \mapsto PL, T, true, true) \mathrel{-\!\!*} preLoopInvR(Stack', PR, P, STree, root))\end{array}$$

3.2.2 The second case

In this case, the following "global" Hoare triple needs to be shown:

$$\left\{\begin{array}{l}\begin{pmatrix}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{pmatrix}\\ \wedge (t = nil \vee (t \mapsto ?, ?, ?, true) \wedge tM = true)\\ \wedge (\exists L, R.\, (p \hookrightarrow L, R, false, true))\end{array}\right\} \texttt{body} \left\{\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right\}$$

The verification follows the same structure as in Section 3.2.1. First, we prove a "local" Hoare triple which exactly and only describes what `body` does when p denotes an allocated, unchecked but marked cell and $(t = nil \vee tM = true)$ holds.

$$\{(P \mapsto PL, PR, false, true) \wedge (t = nil \vee (tM = true)) \wedge (p = P) \wedge (t = T)\}$$

```
 if not((t <> nil) and not(tM))
 then pC := p.check;
      if (pC) then
```

$$\{\bot\}$$

```
                  q := t;
                  t := p;
                  p:= p.right;
                  t.right := q
             else
```

$$\{(P \mapsto PL, PR, false, true) \wedge (p = P) \wedge (t = T)\}$$

```
                  q := t;
                  t := p.right;
                  pL := p.left;
                  p.right := pL;
                  p.left := q;
                  p.check := true
 else
```

$$\{\bot\}$$

```
      q := p;
      p := t;
      t := t.left;
      p.left := q;
      p.mark := true;
      p.check := false;
```

$$\{(P \mapsto T, PL, true, true) \wedge (p = P) \wedge (t = PR)\}$$

In order to change from the local mode of reasoning to the global mode, we incorporate other portions of the heap by Frame Introduction.

$$\frac{\left\{\begin{matrix}(P \mapsto PL, PR, false, true) \\ \wedge (t = nil \vee tM = true) \\ \wedge (p = P) \wedge (t = T)\end{matrix}\right\} \texttt{body} \{(P \mapsto T, PL, true, true) \wedge (p = P) \wedge (t = PR)\}}{\begin{matrix}\left\{\begin{pmatrix}(P \mapsto PL, PR, false, true) \\ \wedge (t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)\end{pmatrix} * ((P \mapsto T, PL, true, true) -\!\!* \, preLoopInvR(Stack', P, PR, STree, root))\right\} \\ \texttt{body} \\ \{((P \mapsto T, PL, true, true) \wedge (p = P) \wedge (t = PR)) * ((P \mapsto T, PL, true, true) -\!\!* \, preLoopInvR(Stack', P, PR, STree, root))\}\end{matrix}}$$

And the auxiliary variables are eliminated by applying Auxiliary Elimination.

$$\frac{\vdots}{\begin{matrix}\left\{\begin{matrix}\exists PL, PR, P, T, Stack'. \\ \begin{pmatrix}(P \mapsto PL, PR, false, true) \\ \wedge (t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)\end{pmatrix} * ((P \mapsto T, PL, true, true) -\!\!* \, preLoopInvR(Stack', P, PR, STree, root))\end{matrix}\right\} \\ \texttt{body} \\ \left\{\begin{matrix}\exists PL, PR, P, T, Stack'. \\ ((P \mapsto T, PL, true, true) \wedge (p = P) \wedge (t = PR)) * ((P \mapsto T, PL, true, true) -\!\!* \, preLoopInvR(Stack', P, PR, STree, root))\end{matrix}\right\}\end{matrix}}$$

The final step is to derive the desired global Hoare triple from the above using Consequence. In the following, we show that the postcondition of the above Hoare triple implies that of the global one.

$$\begin{array}{l}\exists PL, PR, P, T, Stack'.\\ ((P \mapsto T, PL, true, true) \wedge (p = P) \wedge (t = PR)) * ((P \mapsto T, PL, true, true) \mathrel{-\!\!*} preLoopInvR(Stack', P, PR, STree, root))\end{array}$$

$\Longrightarrow$ Since $(p = P) \wedge (t = PR)$ is pure, by Lemma 6 in Appendix B.2,

$$\begin{array}{l}\exists PL, PR, P, T, Stack'.\\ (P \mapsto T, PL, true, true) * ((P \mapsto T, PL, true, true) \mathrel{-\!\!*} preLoopInvR(Stack', P, PR, STree, root))\\ \wedge (p = P) \wedge (t = PR)\end{array}$$

$\Longrightarrow$ $\varphi * (\varphi \mathrel{-\!\!*} \psi) \vdash \psi$

$$\exists PL, PR, P, T, Stack'.\, preLoopInvR(Stack', P, PR, STree, root) \wedge (p = P) \wedge (t = PR)$$

$\Longrightarrow$

$$\exists Stack'.\, preLoopInvR(Stack', p, t, STree, root)$$

The implication between preconditions is shown in the following:

$$\begin{array}{l}(\exists Stack.preLoopInvR(Stack, p, t, STree, root))\\ \wedge (\exists PL, PR.\, (p \hookrightarrow PL, PR, false, true))\\ \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge tM = true)\end{array}$$

$\Longrightarrow$

$$\begin{array}{l}\exists Stack, PL, PR.\\ preLoopInvR(Stack, p, t, STree, root)\\ \wedge (p \hookrightarrow PL, PR, false, true)\\ \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge tM = true)\end{array}$$

$\Longrightarrow$ $preLoopInvR(Stack, p, t, STree, root) \vdash restoredListR(Stack, p) * \top$ and Rule 2 in Appendix B.1, which is
$[\ (p \hookrightarrow PL, PR, false, true) \wedge listMarkedNodesR(Stack, p) * \top \models \exists Stack'.\, Stack = (p, PL, PR, false) :: Stack'\ ]$

$$\begin{array}{l}\exists Stack, PL, PR, Stack', P, T.\\ preLoopInvR(Stack, p, t, STree, root)\\ \wedge (p \hookrightarrow PL, PR, false, true)\\ \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, true) \wedge tM = true)\\ \wedge (Stack = (p, PL, PR, false) :: Stack') \wedge (p = P) \wedge (t = T)\end{array}$$

$\Longrightarrow$

$$\begin{array}{l}\exists PL, PR, Stack', P, T.\\ preLoopInvR((P, PL, PR, false) :: Stack', P, T, STree, root)\\ \wedge (P \hookrightarrow PL, PR, false, true)\\ \wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true) \wedge tM = true) \wedge (p = P) \wedge (t = T)\end{array}$$

$\Longrightarrow$ Lemma 3

$$\begin{array}{l}\exists PL, PR, Stack', P, T.\\ (P \mapsto PL, PR, false, true) * ((P \mapsto T, PL, true, true) \mathrel{-\!\!*} preLoopInvR((P, PL, PR, false) :: Stack', P, T, STree, root))\\ \wedge (t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)\end{array}$$

$\Longrightarrow$ Since $(t = nil \vee tM = true) \wedge (p = P) \wedge (t = T)$ is pure, by Lemma 6 in Appendix B.2,

$$\begin{array}{l}\exists P, PL, PR, T, Stack'.\\ \begin{pmatrix}(P \mapsto PL, PR, false, true)\\ \wedge (p = P) \wedge (t = T)\\ \wedge (t = nil \vee tM = false)\end{pmatrix} * ((P \mapsto T, PL, true, true) \mathrel{-\!\!*} preLoopInvR((P, PL, PR, false) :: Stack', P, T, STree, root))\end{array}$$

3.2.3 The third case

The final case is when t denotes an unmarked cell. The "global" Hoare triple, which needs to be verified in this case, is the following:

$$\left\{\begin{array}{l}\left(\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right)\\ \wedge\ (tM = \mathit{false}) \wedge (t \hookrightarrow ?, ?, ?, \mathit{false}))\end{array}\right\} \texttt{body} \left\{\begin{array}{l}\exists Stack.\\ preLoopInvR(Stack, p, t, STree, root)\end{array}\right\}$$

As in previous two cases, we start the proof of the global Hoare triple with showing that a certain "local" Hoare triple is valid, which precisely describes what `body` does when t denotes an unmarked allocated cell.

```
{(T ↦ TL,TR,TC,false) ∧ (p = P) ∧ (t = T) ∧ (tM = false)}
 if not((t <> nil) and not(tM))
 then pC := p.check;
      if (pC) then
                  {⊥}
                  q := t;
                  t := p;
                  p:= p.right;
                  t.right := q
              else
                  {⊥}
                  q := t;
                  t := p.right;
                  pL := p.left;
                  p.right := pL;
                  p.left := q;
                  p.check := true
 else
      {(T ↦ TL,TR,TC,false) ∧ (p = P) ∧ (t = T)}
      q := p;
      p := t;
      t := t.left;
      p.left := q;
      p.mark := true;
      p.check := false;
{(T ↦ P,TR,false,true) ∧ (p = T) ∧ (t = TL)}
```

Changing the mode of reasoning from local to global follows the same pattern as in Section 3.2.1 and 3.2.2: we apply Frame Introduction in order to incorporate all other portions of the heap and

and eliminate extra auxiliary variables with Auxiliary Elimination.

$$\frac{\{(T \mapsto TL, TR, TC, \mathit{false}) \wedge (p = P) \wedge (t = T) \wedge (tM = \mathit{false})\}\ \texttt{body}\ \{(T \mapsto P, TR, \mathit{false}, \mathit{true}) \wedge (p = T) \wedge (t = TL)\}}{\begin{array}{c} \left\{\begin{array}{l} \exists P, T, TL, TR, TC, Stack. \\ ((T \mapsto TL, TR, TC, \mathit{false}) \wedge (p = P) \wedge (t = T) \wedge (tM = \mathit{false})) \\ * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) -\!\!* \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root)) \end{array}\right\} \\ \texttt{body} \\ \left\{\begin{array}{l} \exists P, T, TL, TR, Stack. \\ ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \wedge (p = T) \wedge (t = TL)) \\ * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) -\!\!* \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root)) \end{array}\right\} \end{array}}$$

It remains to show certain implications between assertions to obtain the global Hoare triple from the above one by Consequence. The following sequence shows the implication from the postcondition in the above Hoare triple to the postcondition of the global one:

$$\begin{array}{l} \exists P, T, TL, TR, Stack. \\ ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \wedge (p = T) \wedge (t = TL)) \\ * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) -\!\!* \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root)) \end{array}$$

$\Longrightarrow$ Since $(p = T) \wedge (t = TL)$ is pure, by Lemma 6 in Appendix B.2,

$$\begin{array}{l} \exists P, T, TL, TR, Stack. \\ (T \mapsto P, TR, \mathit{false}, \mathit{true}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) -\!\!* \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root)) \\ \wedge (p = T) \wedge (t = TL) \end{array}$$

$\Longrightarrow$ $\varphi * (\varphi -\!\!* \psi) \vdash \psi$

$$\begin{array}{l} \exists P, T, TL, TR, Stack, Stack'. \\ \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root)) \wedge (p = T) \wedge (t = TL) \wedge (Stack' = (T, P, TR, \mathit{false}) :: Stack) \end{array}$$

$\Longrightarrow$

$$\exists Stack'.\, \mathit{preLoopInvR}(Stack', p, t, STree, root)$$

Finally, the following shows that the precondition in the global Hoare triple implies the precondition in the Hoare triple which we just obtained using Frame Introduction and Auxiliary Elimination:

$$\begin{array}{l} (\exists Stack.\, \mathit{preLoopInvR}(Stack, p, t, STree, root)) \\ \wedge (tM = \mathit{false}) \wedge (\exists TL, TR, TC.\, (t \hookrightarrow TL, TR, TC, \mathit{false})) \end{array}$$

$\Longrightarrow$

$$\begin{array}{l} \exists Stack, TL, TR, TC. \\ \mathit{preLoopInvR}(Stack, p, t, STree, root) \\ \wedge (tM = \mathit{false}) \wedge (t \hookrightarrow TL, TR, TC, \mathit{false}) \end{array}$$

$\Longrightarrow$

$$\begin{array}{l} \exists Stack, TL, TR, TC, T, P. \\ \mathit{preLoopInvR}(Stack, P, T, STree, root) \\ \wedge (tM = \mathit{false}) \wedge (T \hookrightarrow TL, TR, TC, \mathit{false}) \\ \wedge (p = P) \wedge (t = T) \end{array}$$

$\Longrightarrow$ Lemma 4

$$\begin{array}{l} \exists TL, TR, TC, T, P, Stack. \\ (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) -\!\!* \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root)) \\ \wedge (tM = \mathit{false}) \wedge (p = P) \wedge (t = T) \end{array}$$

$\Longrightarrow$ Since $(tM = \mathit{false}) \wedge (p = P) \wedge (t = T)$ is pure, by Lemma 6 in Appendix B.2,

$$
\begin{aligned}
&\exists TL, TR, TC, T, P, Stack.\\
&((T \mapsto TL, TR, TC, \mathit{false}) \wedge (p = P) \wedge (t = T))\\
&\ast ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!\ast} \mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: Stack, T, TL, STree, root))
\end{aligned}
$$

Acknowledgements

Peter O'Hearn and Uday Reddy gave me insightful comments and encouragements for the work: thanks. I also have benefited from discussions with Richard Bornat, Cristiano Calcagno and David Pym. I am grateful for their helpful comments.

References

[1] R. Bornat. Proving pointer programs in Hoare Logic. In *Mathematics of Program Construction*, July 2000.

[2] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Principles of Programming Languages*, January 2001.

[3] R. Leino. *Toward reliable modular programs.* PhD thesis, California Institute of Technology, 1995.

[4] J. M. Morris. A general axiom of assignment. Assignment and linked data structure. A proof of the Schorr-Waite algorithm. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology (Proceedings of the 1981 Marktoberdorf Summer School)*, pages 25–51. Reidel, 1982.

[5] P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 99.

[6] J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*. Palgrave, 2000.

[7] J. C. Reynolds. Lectures on reasoning about shared mutable data structure. IFIP Working Group 2.3 School/Seminar on State-of-the-Art Program Design Using Logic, Tandil, Argentina, September 6-13, 2000.

A Proof of Lemmas

A.1 Proof of Lemma 1

Since for any state satisfying *Cond* either $((t = \mathit{nil}) \vee (tM = \mathit{true})) \wedge (p \neq \mathit{nil})$ or $(t \neq \mathit{nil}) \wedge (tM \neq \mathit{true})$ holds, it suffices to show the following two implications:

$$
\begin{pmatrix}(\exists Stack.\, \mathit{preLoopInvR}(Stack, p, t, STree, root))\\ \wedge\, (t = \mathit{nil} \vee tM = \mathit{true}) \wedge (p \neq \mathit{nil})\end{pmatrix} \Longrightarrow \begin{pmatrix}((t = \mathit{nil} \vee tM = \mathit{true}) \wedge (p \hookrightarrow ?, ?, \mathit{true}, \mathit{true}))\\ \vee\, ((t = \mathit{nil} \vee tM = \mathit{true}) \wedge (p \hookrightarrow ?, ?, \mathit{false}, \mathit{true}))\end{pmatrix}
$$

$$
\begin{pmatrix}(t \neq \mathit{nil}) \wedge (tM \neq \mathit{true})\\ \wedge\, (t = \mathit{nil} \vee (t \hookrightarrow ?, ?, ?, tM))\end{pmatrix} \Longrightarrow \begin{pmatrix}(tM = \mathit{false}) \wedge (t \hookrightarrow ?, ?, ?, \mathit{false})\end{pmatrix}
$$

The following sequence of implications proves the first implication.

$$
\begin{aligned}
&((t = \mathit{nil}) \vee (tM = \mathit{true})) \wedge (p \neq \mathit{nil}) \wedge (\exists Stack.\mathit{preLoopInvR}(Stack, p, t, STree, root))\\
\Longrightarrow\ &\mathit{preLoopInvR}(Stack, p, t, STree, root) \vdash \mathit{noDangling}(p) \text{ and } \mathit{noDangling}(p) \equiv (p = \mathit{nil}) \vee (p \hookrightarrow ?, ?, ?, ?)\\
&((t = \mathit{nil}) \vee (tM = \mathit{true})) \wedge (\exists Stack.\mathit{preLoopInvR}(Stack, p, t, STree, root)) \wedge (p \hookrightarrow ?, ?, ?, ?)\\
\Longrightarrow\ &\mathit{preLoopInvR}(Stack, p, t, STree, root) \vdash \mathit{listMarkedNodesR}(Stack, p) \ast \top\\
&((t = \mathit{nil}) \vee (tM = \mathit{true})) \wedge \mathit{listMarkedNodesR}(Stack, p) \ast \top \wedge (p \hookrightarrow ?, ?, ?, ?)
\end{aligned}
$$

$\Longrightarrow$ Rule 11 in Appendix B.1, which is $listMarkedNodesR(Lst, p) * \top \wedge (p \hookrightarrow ?, ?, ?, M) \models (M = true)$

$((t = nil) \vee (tM = true)) \wedge (p \hookrightarrow ?, ?, ?, true)$

$\Longrightarrow$ Rule 12 in Appendix B.1, which is $(p \hookrightarrow ?, ?, C, ?) \models (C = true) \vee (C = false)$

$((t = nil) \vee (tM = true)) \wedge ((p \hookrightarrow ?, ?, true, true) \vee (p \hookrightarrow ?, ?, false, true))$

$\Longrightarrow$

$((t = nil \vee tM = true) \wedge (p \hookrightarrow ?, ?, true, true)) \vee ((t = nil \vee tM = true) \wedge (p \hookrightarrow ?, ?, false, true))$

And the following sequence of implications proves the second implication.

$(t \neq nil) \wedge (tM \neq true) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM))$

$\Longrightarrow$ Rule 13 in Appendix B.1, which is $\top \models (tM = true) \vee (tM = false)$

$(t \neq nil) \wedge (tM = false) \wedge (t = nil \vee (t \hookrightarrow ?, ?, ?, tM))$

$\Longrightarrow$

$(tM = false) \wedge (t \hookrightarrow ?, ?, ?, false)$

A.2 Proof of Lemma 2

The following sequence of implications shows the lemma:

$$\begin{array}{l} preLoopInvR((P, PL, PR, true) :: Stack, P, T, STree, root) \\ \wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true)) \\ \wedge (P \hookrightarrow PL, PR, true, true) \end{array}$$

$\Longrightarrow$ Definition of $preLoopInvR$

$$\begin{array}{l} noDanglingR \wedge noDangling(P) \wedge noDangling(T) \\ \wedge listMarkedNodesR((P, PL, PR, true) :: Stack, P) * (restoredListR((P, PL, PR, true) :: Stack, T) -\!\!* spansR(STree, root)) \\ \wedge markedR * (unmarkedR \wedge (\forall x.\, allocated(x) \rightarrow reach(T, x) \vee reachRightChildInList((P, PL, PR, true) :: Stack, x))) \\ \wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true)) \\ \wedge (P \hookrightarrow PL, PR, true, true) \end{array}$$

$\Longrightarrow$ Rule 3 in Appendix B.1, which is $(P \hookrightarrow PL, PR, true, true) \wedge noDanglingR \models noDangling(PR)$

$$\begin{array}{l} noDanglingR \wedge noDanglingR(T) \wedge noDanglingR(PR) \wedge noDangling(P) \\ \wedge listMarkedNodesR((P, PL, PR, true) :: Stack, P) * (restoredListR((P, PL, PR, true) :: Stack, T) -\!\!* spansR(STree, root)) \\ \wedge markedR * (unmarkedR \wedge (\forall x.\, allocated(x) \rightarrow reach(T, x) \vee reachRightChildInList((P, PL, PR, true) :: Stack, x))) \\ \wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true)) \\ \wedge (P \hookrightarrow PL, PR, true, true) \end{array}$$

$\Longrightarrow$ Rules 4,5,6 in Appendix B.1, which are

$$\left[\begin{array}{l} (P \hookrightarrow PL, PR, true, true) \wedge noDangling(T) \wedge noDanglingR \\ \qquad \models (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* noDanglingR) \\ (P \hookrightarrow PL, PR, true, true) \wedge noDangling(PR) \\ \qquad \models (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* noDangling(PR)) \\ (P \hookrightarrow PL, PR, true, true) \\ \qquad \models (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* noDangling(P)) \end{array}\right]$$

$$\begin{array}{l} (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* noDanglingR) \\ \wedge (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* noDangling(PR)) \\ \wedge (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* noDangling(P)) \\ \wedge listMarkedNodesR((P, PL, PR, true) :: Stack, P) * (restoredListR((P, PL, PR, true) :: Stack, T) -\!\!* spansR(STree, root)) \\ \wedge markedR * (unmarkedR \wedge (\forall x.\, allocated(x) \rightarrow reach(T, x) \vee reachRightChildInList((P, PL, PR, true) :: Stack, x))) \\ \wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true)) \end{array}$$

$\Longrightarrow$ Definitions of $listMarkedNodesR, restoredListR, reachRightChildInList$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(P))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PR)\\ * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \mathit{reach}(T, x) \vee \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\\
&\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))
\end{aligned}
$$

$\Longrightarrow \varphi * (\psi * \chi \mathrel{-\!\!*} \vartheta) \vdash \psi \mathrel{-\!\!*} (\varphi * (\chi \mathrel{-\!\!*} \vartheta))$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(P))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true})\\ * \begin{pmatrix} (P, PL, T, \mathit{true}, \mathit{true})\\ \mathrel{-\!\!*} \mathit{listMarkedNodesR}(\mathit{Stack}, PR) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\end{pmatrix}\end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \mathit{reach}(T, x) \vee \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\\
&\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))
\end{aligned}
$$

$\Longrightarrow$ Rule 7 in Appendix B.1, which is

$$
\left[\begin{aligned}
&((T = \mathit{nil}) \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \mathit{reach}(T, x) \vee \varphi))\\
&\qquad \models \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \varphi))
\end{aligned}\right]
$$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(P))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true})\\ * \begin{pmatrix} (P, PL, T, \mathit{true}, \mathit{true})\\ \mathrel{-\!\!*} \mathit{listMarkedNodesR}(\mathit{Stack}, PR) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\end{pmatrix}\end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\\
&\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))
\end{aligned}
$$

$\Longrightarrow$ Rule 8 in Appendix B.1, which is

$$
\left[\begin{aligned}
&(P \hookrightarrow PL, PR, \mathit{true}, \mathit{true}) \wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge \varphi)\\
&\qquad \models (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * \mathit{markedR} * (\mathit{unmarkedR} \wedge \varphi))
\end{aligned}\right]
$$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(P))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true})\\ * \begin{pmatrix} (P, PL, T, \mathit{true}, \mathit{true})\\ \mathrel{-\!\!*} \mathit{listMarkedNodesR}(\mathit{Stack}, PR) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\end{pmatrix}\end{pmatrix}\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\end{pmatrix}
\end{aligned}
$$

$\Longrightarrow \varphi \vdash \psi \mathrel{-\!\!*} (\psi * \varphi)$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge (P \mapsto PL, PR, \mathit{true}, \mathit{true}) * ((P \mapsto PL, T, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(P))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true})\\ * \begin{pmatrix} (P, PL, T, \mathit{true}, \mathit{true})\\ \mathrel{-\!\!*} \mathit{listMarkedNodesR}(\mathit{Stack}, PR) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\end{pmatrix}\end{pmatrix}\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{true}, \mathit{true})\\ * \begin{pmatrix} (P \mapsto PL, T, \mathit{true}, \mathit{true})\\ \mathrel{-\!\!*} \begin{pmatrix} (P \mapsto PL, T, \mathit{true}, \mathit{true}) * \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\, \mathit{allocated}(x) \to \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\end{pmatrix}\end{pmatrix}\end{pmatrix}
\end{aligned}
$$

$\Longrightarrow$ Rule 1 in Appendix B.1, which is $(P \mapsto PL, T, \mathit{true}, \mathit{true}) * \mathit{markedR} \models \mathit{markedR}$

$$
\begin{aligned}
&(P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* \; noDanglingR)\\
&\wedge (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* \; noDangling(PR))\\
&\wedge (P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* \; noDangling(P))\\
&\wedge \left(\begin{array}{l}(P \mapsto PL, PR, true, true)\\ * \left(\begin{array}{l}(P, PL, T, true, true)\\ -\!\!* \; listMarkedNodesR(Stack, PR) * (restoredListR(Stack, P) -\!\!* \; spansR(STree, root))\end{array}\right)\end{array}\right)\\
&\wedge \left(\begin{array}{l}(P \mapsto PL, PR, true, true)\\ * \left(\begin{array}{l}(P \mapsto PL, T, true, true)\\ -\!\!* \; markedR * (unmarkedR \wedge (\forall x.\, allocated(x) \rightarrow reach(PR, x) \vee reachRightChildInList(Stack, x)))\end{array}\right)\end{array}\right)
\end{aligned}
$$

$\Longrightarrow$ $(P \mapsto PL, PR, true, true)$ is strictly exact (Lemma 8 in Appendix B.2) and $(\varphi -\!\!* \psi) \wedge (\varphi -\!\!* \chi) \dashv\vdash \varphi -\!\!* (\psi \wedge \chi)$

$$
\begin{aligned}
&(P \mapsto PL, PR, true, true)\\
&\left(\begin{array}{l}(P \mapsto PL, T, true, true)\\ -\!\!* \left(\begin{array}{l}noDanglingR \wedge noDangling(PR) \wedge noDangling(P)\\ \wedge\, listMarkedNodesR(Stack, PR) * (restoredListR(Stack, P) -\!\!* \; spansR(STree, root))\\ \wedge\, markedR * (unmarkedR \wedge (\forall x.\, allocated(x) \rightarrow reach(PR, x) \vee reachRightChildInList(Stack, x)))\end{array}\right)\end{array}\right)
\end{aligned}
$$

$\Longrightarrow$ Definition of $preLoopInvR$

$$
(P \mapsto PL, PR, true, true) * ((P \mapsto PL, T, true, true) -\!\!* \; preLoopInvR(Stack, PR, P, STree, root))
$$

A.3 Proof of Lemma 3

The following sequence of implications shows the lemma:

$$
\begin{aligned}
&preLoopInvR((P, PL, PR, false) :: Stack, P, T, STree, root)\\
&\wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true))\\
&\wedge (P \hookrightarrow PL, PR, false, true)
\end{aligned}
$$

$\Longrightarrow$ Definition of $preLoopInvR$

$$
\begin{aligned}
&noDanglingR \wedge noDangling(P) \wedge noDangling(T)\\
&\wedge listMarkedNodesR((P, PL, PR, false) :: Stack, P) * (restoredListR((P, PL, PR, false) :: Stack, T) -\!\!* \; spansR(STree, root))\\
&\wedge markedR * (unmarkedR \wedge (\forall x. allocatedI(x) \rightarrow reach(T, x) \vee reachRightChildInList((P, PL, PR, false) :: Stack, x)))\\
&\wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true))\\
&\wedge (P \hookrightarrow PL, PR, false, true)
\end{aligned}
$$

$\Longrightarrow$ Rule 3 in Appendix B.1, which is $(P \hookrightarrow PL, PR, false, true) \wedge noDanglingR \models noDangling(PR)$

$$
\begin{aligned}
&noDanglingR \wedge noDangling(P) \wedge noDangling(T) \wedge noDanglingR(PR)\\
&\wedge listMarkedNodesR((P, PL, PR, false) :: Stack, P) * (restoredListR((P, PL, PR, false) :: Stack, T) -\!\!* \; spansR(STree, root))\\
&\wedge markedR * (unmarkedR \wedge (\forall x. allocatedI(x) \rightarrow reach(T, x) \vee reachRightChildInList((P, PL, PR, false) :: Stack, x)))\\
&\wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true))
\end{aligned}
$$

$\Longrightarrow$ Rules 4,5,6 in Appendix B.1, which are

$$
\left[\begin{array}{l}
(P \hookrightarrow PL, PR, false, true) \wedge noDanglingR \wedge noDangling(T)\\
\qquad \models (P \mapsto PL, PR, false, true) * ((P \mapsto T, PL, true, true) -\!\!* \; noDanglingR)\\
(P \hookrightarrow PL, PR, false, true) \wedge noDangling(PR)\\
\qquad \models (P \mapsto PL, PR, false, true) * ((P \mapsto T, PL, true, true) -\!\!* \; noDangling(PR))\\
(P \hookrightarrow PL, PR, false, true)\\
\qquad \models ((P \mapsto T, PL, true, true) -\!\!* \; noDangling(P))
\end{array}\right]
$$

$$
\begin{aligned}
&(P \mapsto PL, PR, false, true) * ((P \mapsto T, PL, true, true) -\!\!* \; noDanglingR)\\
&\wedge (P \mapsto PL, PR, false, true) * ((P \mapsto T, PL, true, true) -\!\!* \; noDangling(P))\\
&\wedge (P \mapsto PL, PR, false, true) * ((P \mapsto T, PL, true, true) -\!\!* \; noDangling(PR))\\
&\wedge listMarkedNodesR((P, PL, PR, false) :: Stack, P) * (restoredListR((P, PL, PR, false) :: Stack, T) -\!\!* \; spansR(STree, root))\\
&\wedge markedR * (unmarkedR \wedge (\forall x. allocated(x) \rightarrow reach(T, x) \vee reachRightChildInList((P, PL, PR, false) :: Stack, x)))\\
&\wedge (T = nil \vee (T \hookrightarrow ?, ?, ?, true))
\end{aligned}
$$

$\Longrightarrow$ Definitions of $listMarkedNodesR, restoredListR, reachRightChildInList$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(P))\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL)\\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathbin{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\\
&\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))
\end{aligned}
$$

$\Longrightarrow \varphi \vdash \psi \mathbin{-\!\!*} (\psi * \varphi)$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(P))\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true})\\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true})\\ \mathbin{-\!\!*} \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL)\\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathbin{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\\
&\wedge (T = \mathit{nil} \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))
\end{aligned}
$$

$\Longrightarrow$ Rule 7 in Appendix B.1, which is

$$
\left[\begin{array}{l} ((T = \mathit{nil}) \vee (T \hookrightarrow ?, ?, ?, \mathit{true}))\\ \wedge\, \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(T, x) \vee \varphi))\\ \qquad \models\ \wedge\, \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \varphi)) \end{array}\right]
$$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(P))\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true})\\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true})\\ \mathbin{-\!\!*} \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL)\\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathbin{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))
\end{aligned}
$$

$\Longrightarrow$ Rule 8 in Appendix B.1, which is

$$
\left[\begin{array}{l} (P \hookrightarrow PL, PR, \mathit{false}, \mathit{true}) \wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge \varphi)\\ \qquad \models (P \hookrightarrow PL, PR, \mathit{false}, \mathit{true}) * \mathit{markedR} * (\mathit{unmarkedR} \wedge \varphi) \end{array}\right]
$$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(P))\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true})\\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true})\\ \mathbin{-\!\!*} \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL)\\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathbin{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix}\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow \varphi \vdash \psi \mathbin{-\!\!*} (\psi * \varphi)$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDanglingR})\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(P))\\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathbin{-\!\!*} \mathit{noDangling}(PR))\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true})\\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true})\\ \mathbin{-\!\!*} \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL)\\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathbin{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix}\\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true})\\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true})\\ \mathbin{-\!\!*} \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \end{pmatrix} \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow$ Rule 1 in Appendix B.1, which is $(P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{markedR} \vdash \mathit{markedR}$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR}) \\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(P)) \\
&\wedge (P \mapsto PL, PR, \mathit{false}, \mathit{true}) * ((P \mapsto T, PL, \mathit{true}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(PR)) \\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true}) \\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) \\ \mathrel{-\!\!*} \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL) \\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix} \\
&\wedge \begin{pmatrix} (P \mapsto PL, PR, \mathit{false}, \mathit{true}) \\ * \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) \\ \mathrel{-\!\!*} \begin{pmatrix} \mathit{markedR} \\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \end{pmatrix} \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow$ $(P \mapsto PL, PR, \mathit{false}, \mathit{true})$ is strictly exact (Lemma 8 in Appendix B.2) and $\varphi \mathrel{-\!\!*} (\psi \wedge \chi) \dashv\vdash (\varphi \mathrel{-\!\!*} \psi) \wedge (\varphi \mathrel{-\!\!*} \chi)$

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) \\
&* \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) \\ \mathrel{-\!\!*} \begin{pmatrix} \mathit{noDanglingR} \wedge \mathit{noDangling}(P) \wedge \mathit{noDangling}(PR) \\ \wedge \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, PL) \\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \\ \wedge\, \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow$ $\vee$-Introduction and Definitions of *listMarkedNodesR* and *reachRightChildInList*

$$
\begin{aligned}
&(P \mapsto PL, PR, \mathit{false}, \mathit{true}) \\
&* \begin{pmatrix} (P \mapsto T, PL, \mathit{true}, \mathit{true}) \\ \mathrel{-\!\!*} \begin{pmatrix} \mathit{noDanglingR} \wedge \mathit{noDangling}(P) \wedge \mathit{noDangling}(PR) \\ \wedge \begin{pmatrix} \mathit{listMarkedNodesR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, P) \\ * (\mathit{restoredListR}((P, T, PL, \mathit{true}) :: \mathit{Stack}, PR) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \\ \wedge \begin{pmatrix} \mathit{markedR} \\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocatedI}(x) \rightarrow \mathit{reach}(PR, x) \vee \mathit{reachRightChildInList}((P, T, PL, \mathit{true}) :: \mathit{Stack}, x))) \end{pmatrix} \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow$ Definition of *preLoopInvR*

$$\mathit{preLoopInvR}(\mathit{Stack}, P, PR, \mathit{STree}, \mathit{root})$$

A.4 Proof of Lemma 4

The following sequence of implications shows the lemma:

$$\mathit{preLoopInvR}(\mathit{Stack}, P, T, \mathit{STree}, \mathit{root}) \wedge (T \hookrightarrow TL, TR, TC, \mathit{true})$$

$\Longrightarrow$ Definition of *preLoopInvR*

$$
\begin{aligned}
&\mathit{noDanglingR} \wedge \mathit{noDangling}(P) \wedge \mathit{noDangling}(T) \\
&\wedge \mathit{listMarkedNodesR}(\mathit{Stack}, P) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \\
&\wedge (T \hookrightarrow TL, TR, TC, \mathit{true})
\end{aligned}
$$

$\Longrightarrow$ Rule 1 in Appendix B.1, which is $(T \hookrightarrow TL, TR, TC, \mathit{false}) \wedge \mathit{noDanglingR} \models \mathit{noDanglingR}(TL)$

$$
\begin{aligned}
&\mathit{noDanglingR} \wedge \mathit{noDangling}(P) \wedge \mathit{noDangling}(T) \wedge \mathit{noDanglingR}(TL) \\
&\wedge \mathit{listMarkedNodesR}(\mathit{Stack}, P) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \\
&\wedge (T \hookrightarrow TL, TR, TC, \mathit{true})
\end{aligned}
$$

$\Longrightarrow$ Rules 4 and 5 in Appendix B.1, which are

$$
\begin{bmatrix}
(T \hookrightarrow TL, TR, TC, \mathit{false}) \wedge \mathit{noDanglingR} \wedge \mathit{noDangling}(P) \\
\qquad \models (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR}) \\
(T \hookrightarrow TL, TR, TC, \mathit{false}) \wedge \mathit{noDanglingR}(TL) \\
\qquad \models (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(TL)) \\
(T \hookrightarrow TL, TR, TC, \mathit{false}) \\
\qquad \models (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(T))
\end{bmatrix}
$$

$$
\begin{aligned}
&(T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(T))\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(TL))\\
&\wedge \mathit{listMarkedNodesR}(\mathit{Stack}, P) * (\mathit{restoredListR}(\mathit{Stack}, P) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))
\end{aligned}
$$

$\Longrightarrow$ Rule 10 in Appendix B.1, which is

$$
\left[\begin{array}{l}
(T \hookrightarrow TL, TR, TC, \mathit{false}) \wedge \mathit{listMarkedNodesR}(\mathit{Stack}, P) * (\mathit{restoredListR}(\mathit{Stack}, T) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root}))\\
\quad \models \begin{pmatrix} \mathit{listMarkedNodesR}(\mathit{Stack}, P) * (T \mapsto TL, TR, TC, \mathit{false})\\ * ((T \mapsto TL, TR, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{Stack}, T) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix}
\end{array}\right]
$$

$$
\begin{aligned}
&(T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(T))\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(TL))\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false}) * \mathit{listMarkedNodesR}(\mathit{Stack}, P)\\ * ((T \mapsto TL, TR, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{Stack}, T) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix}\\
&\wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))
\end{aligned}
$$

$\Longrightarrow$ Rule 9 in Appendix B.1, which is

$$
\left[\begin{array}{l}
(T \hookrightarrow TL, TR, TC, \mathit{false}) \wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(T, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x)))\\
\quad \models \begin{pmatrix} \mathit{markedR} * (T \mapsto TL, TR, TC, \mathit{false})\\ \begin{pmatrix} \mathit{unmarkedR}\\ \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(TL, x) \vee \mathit{reachI}(TR, x) \vee \mathit{reachRightChildInList}(Lst, x)) \end{pmatrix} \end{pmatrix}
\end{array}\right]
$$

$$
\begin{aligned}
&(T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(T))\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(TL))\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false}) * \mathit{listMarkedNodesR}(\mathit{Stack}, P)\\ * ((T \mapsto TL, TR, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{Stack}, T) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix}\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false}) * \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(TL, x) \vee \mathit{reach}(TR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow \varphi \vdash \psi \mathrel{-\!\!*} (\psi * \varphi)$

$$
\begin{aligned}
&(T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(T))\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(TL))\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false})\\ * \begin{pmatrix} (T \mapsto P, TR, \mathit{false}, \mathit{true})\\ \mathrel{-\!\!*} \begin{pmatrix} (T \mapsto P, TR, \mathit{false}, \mathit{true}) * \mathit{listMarkedNodesR}(\mathit{Stack}, P)\\ * ((T \mapsto TL, TR, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{Stack}, T) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix}\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false})\\ * \begin{pmatrix} (T \mapsto P, TR, \mathit{false}, \mathit{true})\\ \mathrel{-\!\!*} \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false}) * \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(TL, x) \vee \mathit{reach}(TR, x) \vee \mathit{reachRightChildInList}(\mathit{Stack}, x))) \end{pmatrix} \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow$ Definitions of $\mathit{listMarkedNodesR}$, $\mathit{restoredListR}$, $\mathit{reachRightChildInList}$ and Rule 1, which is

$\left[\ (T \mapsto TL, TR, TC, \mathit{false}) * \mathit{markedR} \models \mathit{markedR}\ \right]$

$$
\begin{aligned}
&(T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDanglingR})\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(T))\\
&\wedge (T \mapsto TL, TR, TC, \mathit{false}) * ((T \mapsto P, TR, \mathit{false}, \mathit{true}) \mathrel{-\!\!*} \mathit{noDangling}(TL))\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false})\\ * \begin{pmatrix} (T \mapsto P, TR, \mathit{false}, \mathit{true})\\ \mathrel{-\!\!*} \begin{pmatrix} \mathit{listMarkedNodesR}((T, P, TR, \mathit{false}) :: \mathit{Stack}, T)\\ * (\mathit{restoredListR}((T, P, TR, \mathit{false}) :: \mathit{Stack}, TL) \mathrel{-\!\!*} \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{pmatrix} \end{pmatrix} \end{pmatrix}\\
&\wedge \begin{pmatrix} (T \mapsto TL, TR, TC, \mathit{false})\\ \begin{pmatrix} (T \mapsto P, TR, \mathit{false}, \mathit{true})\\ \mathrel{-\!\!*} \begin{pmatrix} \mathit{markedR}\\ * (\mathit{unmarkedR} \wedge (\forall x.\mathit{allocated}(x) \rightarrow \mathit{reach}(TL, x) \vee \mathit{reachRightChildInList}((T, P, TR, \mathit{false}) :: \mathit{Stack}, x))) \end{pmatrix} \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

$\Longrightarrow (T \mapsto TL, TR, TC, \mathit{false})$ is strictly exact (Lemma 8 in Appendix B.2) and $\varphi \mathrel{-\!\!*} (\psi \wedge \chi) \dashv\vdash (\varphi \mathrel{-\!\!*} \psi) \wedge (\varphi \mathrel{-\!\!*} \chi)$

$$(T \mapsto TL, TR, TC, \mathit{false}) * \left(\begin{array}{l} (T \mapsto P, TR, \mathit{false}, \mathit{true}) \\ -\!\!* \left(\begin{array}{l} \mathit{noDanglingR} \wedge \mathit{noDangling}(T) \wedge \mathit{noDangling}(TL) \\ \wedge \left(\begin{array}{l} \mathit{listMarkedNodesR}((T, P, TR, \mathit{false}) :: \mathit{Stack}, T) \\ * (\mathit{restoredListR}((T, P, TR, \mathit{false}) :: \mathit{Stack}, TL) -\!\!* \mathit{spansR}(\mathit{STree}, \mathit{root})) \end{array} \right) \\ \wedge \left(\begin{array}{l} \mathit{markedR} \\ * (\mathit{unmarkedR} \wedge (\forall x. \mathit{allocated}(x) \rightarrow \mathit{reach}(TL, x) \vee \mathit{reachRightChildInList}((T, P, TR, \mathit{false}) :: \mathit{Stack}, x))) \end{array} \right) \end{array} \right) \end{array} \right)$$

$\Longrightarrow$ Definition of $\mathit{preLoopInvR}$

$$\mathit{preLoopInvR}((T, P, TR, \mathit{false}) :: \mathit{Stack}, T, TL, \mathit{STree}, \mathit{root})$$

B Properties of Assertions

B.1 Extra Rules

1. $(x \mapsto l, r, c, \mathit{true}) * \mathit{markedR}$
   $\models \mathit{markedR}$

2. $(x \hookrightarrow l, r, c, \mathit{true}) \wedge \mathit{listMarkedNodesR}(\mathit{lst}, x) * \top$
   $\models \exists \mathit{lst}'. \mathit{lst} = (x, l, r, c) :: \mathit{lst}'$

3. $(x \hookrightarrow l, r, c, m) \wedge \mathit{noDanglingR}$
   $\models \mathit{noDangling}(l) \wedge \mathit{noDangling}(r)$

4. $(x \hookrightarrow l, r, c, m) \wedge \mathit{noDangling}(y) \wedge \mathit{noDanglingR}$
   $\models (x \mapsto l, r, c, m) * ((x \mapsto l, y, c, m) \vee (x \mapsto y, r, c, m) -\!\!* \mathit{noDanglingR})$

5. $(x \hookrightarrow l, r, c, m) \wedge \mathit{noDangling}(y)$
   $\models (x \mapsto l, r, c, m) * ((x \mapsto l', r', c', m') -\!\!* \mathit{noDangling}(y))$

6. $(x \hookrightarrow l, r, c, m)$
   $\models (x \mapsto l, r, c, m) * ((x \mapsto l', r', c', m') -\!\!* \mathit{noDangling}(x))$

7. $((x = \mathit{nil}) \vee (x \hookrightarrow ?, ?, ?, \mathit{true}))$
   $\wedge\ \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall y.\ \mathit{allocated}(y) \rightarrow \mathit{reach}(x, y) \vee \varphi))$
   $\models \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall y.\ \mathit{allocated}(y) \rightarrow \varphi))$

8. $(x \hookrightarrow l, r, c, \mathit{true}) \wedge \mathit{markedR} * (\mathit{unmarkedR} \wedge \varphi)$
   $\models (x \mapsto l, r, c, \mathit{true}) * \mathit{markedR} * (\mathit{unmarkedR} \wedge \varphi)$

9. $(x \hookrightarrow l, r, c, \mathit{false})$
   $\wedge\ \mathit{markedR} * (\mathit{unmarkedR} \wedge (\forall y. \mathit{allocated}(y) \rightarrow \mathit{reach}(x, y) \vee \mathit{reachRightChildInList}(\mathit{lst}, y)))$
   $\models \mathit{markedR} * (x \mapsto l, r, c, \mathit{false}) * \left(\begin{array}{l} \mathit{unmarkedR} \\ \wedge (\forall y. \mathit{allocated}(y) \rightarrow \mathit{reach}(l, y) \vee \mathit{reachI}(r, y) \vee \mathit{reachRightChildInList}(\mathit{lst}, y)) \end{array} \right)$

10. $(x \hookrightarrow l, r, c, \mathit{false}) \wedge \mathit{listMarkedNodesR}(\mathit{lst}, y) * (\mathit{restoredListR}(\mathit{lst}, x) -\!\!* \mathit{spansR}(\mathit{stree}, \mathit{root}))$
    $\models \mathit{listMarkedNodesR}(\mathit{lst}, y) * (x \mapsto l, r, c, \mathit{false}) * ((x \mapsto l, r, \mathit{true}, \mathit{true}) * \mathit{restoredListR}(\mathit{lst}, x) -\!\!* \mathit{spansR}(\mathit{stree}, \mathit{root}))$

11. $\mathit{listMarkedNodesR}(\mathit{lst}, x) * \top \wedge (x \hookrightarrow l, r, c, m) \models m = \mathit{true}$

12. $(x \hookrightarrow l, r, c, m) \models (c = \mathit{true} \vee c = \mathit{false}) \wedge (m = \mathit{true} \vee m = \mathit{false})$

13. for any stack variable x of *bool* type, $\top \models (x = \mathit{true}) \vee (x = \mathit{false})$

B.2 Special BI Formulas

Definition 5 (Pure BI Formula) A BI formula φ is *pure* iff for any stack s, the following two are equivalent:

- there exists a heap h such that $s, h \models \varphi$; and
- for all heaps h, $s, h \models \varphi$.

Lemma 6 For any BI pure formula φ and all BI formulas ψ, χ,

$$(\psi * \chi) \wedge \varphi \models (\psi \wedge \varphi) * \chi \text{ and } (\psi \wedge \varphi) * \chi \models (\psi * \chi) \wedge \varphi$$

Definition 7 (Strictly Exact BI Formula) A BI formula φ is *strictly exact* iff for any stack s, there exists at most one heap h such that $s, h \models \varphi$.

Lemma 8 For any BI strictly exact formula φ, and all BI formulas ψ, χ,

$$(\varphi * \psi) \wedge (\varphi * \chi) \models \varphi * (\psi \wedge \chi)$$