Reasoning about High-Level Tree Update and its Low-Level Implementation

Philippa Gardner and Uri Zarfaty, Imperial College London

Abstract

We relate Context Logic reasoning about a high-level tree update language with Separation Logic reasoning about a low-level implementation.

1 Introduction

Separation Logic (SL) was introduced by O'Hearn, Reynolds and Yang [5, 4, 7], to specify properties about low-level code for manipulating mutable data structures in memory. Parkinson and Bierman introduced abstract predicates for SL [6], to hide details of the implementation from the client's view. Meanwhile, with Calcagno [2], we generalised SL to develop local Hoare reasoning using Context Logic (CL) for directly specifying properties about arbitrary structured data update such as XML update. Since then, we have had many interesting discussions on the connections between these abstraction approaches. In this paper, we relate the two approaches using tree update.

The key idea of local Hoare reasoning is that the reasoning follows the footprint of the program: 'To understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses [the footprint]. The value of any other cell will automatically remain unchanged. [5]' With tree update, the high-level footprints are the subtrees affected by the high-level update commands. The low-level footprints meanwhile consist of those parts of the memory corresponding the high-level footprints *plus* the nearby nodes requiring additional pointer surgery. Tying up the high-level and low-level reasoning is therefore rather subtle.

We work with a high-level tree structure (forests) consisting of nodes labelled with unique identifiers and lists of children which may vary in length. This simple structure gives the essence of XML viewed as an in-place memory model. We have studied the full XML structure in our formal specification of the W3C Document Object Model (DOM) [8, 3]. Consider the high-level command 'delete n', which removes the tree identified by n. E.g., when $n = \mathsf{n}_3$, we have the update

The high-level local Hoare axiom for 'delete n' is:

$$\{n[\text{true}]\} \text{ delete } n \ \{0\}$$

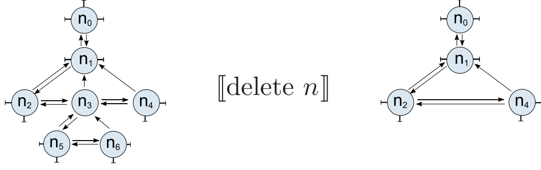
The pre-condition is a CL formula stating that the tree has a top node n and an unspecified subtree. The post-condition denotes the empty tree. The axiom is small, in that the pre-condition only touches the portion of the tree accessed by the command (the high-level footprint) and provides the minimal safety condition necessary for the command to execute. Using the high-level *frame* rule, we can uniformly extend the reasoning to the rest of the tree using application:

$$\{K \cdot n[\text{true}]\} \text{ delete } n \ \{K \cdot 0\} \qquad (1)$$

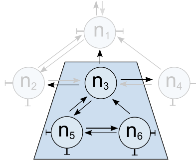
The pre-condition states that the tree can be split into a subtree with top node n, and a context satisfying CL formula K which is unaffected by the command. The post-condition has the same structure, with the empty tree substituted for the tree at n.

There are many ways of implementing the high-level delete command. We focus on one natural approach; our ideas apply to any implementation requiring pointer surgery. We represent our tree structure by a collection of heap cells corresponding to the tree nodes, each containing links to the node's parent, previous and next sibling, and first child. The low-level program for deleting a tree with top node n, denoted $[\![\text{delete } n]\!]$, first does a depth-first removal of each node under n, then removes n and joins the appropriate par-

ent and sibling nodes. The overall update looks like:



The low-level Hoare triple for ⟦delete n⟧ requires two types of SL formula. It uses a formula $[\![n[\text{true}]]\!]^I$, resulting from a translation of the CL formula $n[\text{true}]$ to a SL formula which describes a heap corresponding to a tree with top node n. It is defined using the abstract predicate 'subtree $n[\text{true}]$ I'. The translation depends on an interface $I = (l, i, u, j, r)$, where l, u, r correspond to the left-sibling, parent and right-sibling, and i, j to the first and last node of the forest: in our example, 'subtree $n[\text{true}]$ I' is satisfied by



with $i = n_3 = j$. We also require a *crust formula* crust^I which specifies properties about the three nearby nodes l, u, r (in our example, the shaded n_2, n_1, n_4), since all three nodes potentially require pointer surgery.

The low-level Hoare specification for ⟦delete n⟧ is

$$\{\exists i, j.\, \text{crust}^I * [\![n[\text{true}]]\!]^I\}\ [\![\text{delete } n]\!]\ \{\exists i, j.\, \text{crust}^I * [\![0]\!]^I\}$$

The pre-condition describes the low-level footprint of ⟦delete n⟧: the heap corresponding to a tree with top node n plus the crust. The post-condition describes the result of the deletion: the heap corresponding to the empty tree given by $[\![0]\!]^I$ and the crust. In our example, i, j are instantiated to n_3 in the precondition, and n_2 and n_4 respectively in the post condition.

We can uniformly extend the low-level reasoning to the rest of the heap using the low-level frame rule. E.g., the low-level Hoare triple corresponding to (1) is:

$$\{\exists i', j'.\text{crust}^{I'} * [\![K \cdot n[\text{true}]]\!]^{I'}\}[\![\text{delete } n]\!]\{\exists i', j'.\text{crust}^{I'} * [\![K \cdot 0]\!]^{I'}\}$$

Application at the high-level corresponds to separation across interfaces at the low level, in that $[\![K \cdot P]\!]^{I'} = \exists I.\, [\![K]\!]_I^{I'} * [\![P]\!]^I$. The above triple can be derived from the specification of ⟦delete n⟧ by essentially adding on the formulæ $\text{crust}^{I'}$ and $[\![K]\!]_I^{I'}$ using $*$, and removing the inner crust^I (using the adjoint of $*$). The formula crust^I has been subsumed by context $[\![K]\!]_I^{I'}$ and the new formula $\text{crust}^{I'}$. Our results seem to suggest that the crust plays an integral part of the low-level reasoning: i.e., it is not possible to use abstract predicates to completely abstract from the implemention details. Instead, CL reasoning seems to provide the right level of abstraction for specifying libraries for manipulating structured data.

The uniformity of the relationship between our high-level and low-level reasoning suggests that we might push our comparison further, by extending the relational reasoning introduced by Yang and Benton [9, 1]. They work with the same heap model at both levels of reasoning, whereas our reasoning is on different data models. We currently do not see the need for their level of integration here.

2 Tree Update Language

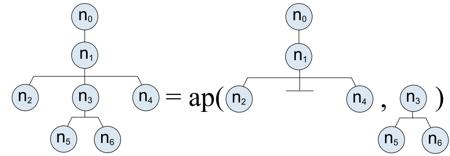
We introduce a simple, but expressive, high-level tree update language. Our tree structures are intentionally simple: they consist of ordered trees (forests) with unique node identifiers.

Definition 2.1 (Trees/contexts)**.** Given set $\mathcal{N} = \{\mathsf{m}, \mathsf{n}, \dots\}$ of identifiers, the *trees* $\mathsf{t} \in \mathcal{T}$, *contexts* $\mathsf{c} \in \mathcal{C}$ and application $\text{ap}\colon \mathcal{C} \times \mathcal{T} \to \mathcal{T}$ are:

$$\begin{aligned} &\mathsf{t} ::= 0 \mid \mathsf{n}[\mathsf{t}] \mid \mathsf{t}|\mathsf{t} \qquad & \text{ap}(-, \mathsf{t}) &\triangleq \mathsf{t} \\ &\mathsf{c} ::= - \mid \mathsf{n}[\mathsf{c}] \mid \mathsf{t}|\mathsf{c} \mid \mathsf{c}|\mathsf{t} & \text{ap}(\mathsf{n}[\mathsf{c}], \mathsf{t}) &\triangleq \mathsf{n}[\text{ap}(\mathsf{c}, \mathsf{t})] \\ & & \text{ap}(\mathsf{t}|\mathsf{c}, \mathsf{t}') &\triangleq \mathsf{t}|(\text{ap}(\mathsf{c}, \mathsf{t}')) \\ & & \text{ap}(\mathsf{c}|\mathsf{t}, \mathsf{t}') &\triangleq \text{ap}(\mathsf{c}, \mathsf{t}')|\mathsf{t} \end{aligned}$$

Trees and contexts satisfy a *structural congruence* $\equiv$ stating that $|$ is associative, and that 0 is its left and right identity. *Well-formed* trees and contexts are those with unique identifiers, making context application a partial operation. We write $\mathsf{t}_1 \# \mathsf{t}_2$ to mean that two well-formed trees have disjoint sets of identifiers, and often omit the 0 leaves from a tree to make it more readable: e.g., write $\mathsf{n}[\mathsf{m}_1|\mathsf{m}_2]$ instead of $\mathsf{n}[\mathsf{m}_1[0]|\mathsf{m}_2[0]]$.

Example 2.2 (Trees and tree contexts)**.** The tree $\mathsf{n}_0[\mathsf{n}_1[\mathsf{n}_2|\mathsf{n}_3[\mathsf{n}_5|\mathsf{n}_6]|\mathsf{n}_4]]$ can be separated into the context $\mathsf{n}_0[\mathsf{n}_1[\mathsf{n}_2|-|\mathsf{n}_4]]$ and subtree $\mathsf{n}_3[\mathsf{n}_5|\mathsf{n}_6]$:



The update language is a high-level, stateful, imperative language, based on variable assignment and in-place tree update. The program state includes a main working tree, together with a high-level variable store containing variables for both node identifiers (references into the tree, with nil representing the

$$\frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}'])}{\text{delete } n_{\uparrow},\mathsf{s},\mathsf{t} \leadsto \mathsf{s},\mathrm{ap}(\mathsf{c},0)} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}'])}{\text{delete } n_{\downarrow},\mathsf{s},\mathsf{t} \leadsto \mathsf{s},\mathrm{ap}(\mathsf{c},\mathsf{n}[0])} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}']) \quad \langle \mathsf{t}''\rangle \equiv [\![X]\!]\mathsf{s} \quad \mathsf{t}''\#\mathsf{t}}{\text{insert } X \text{ at } n_{\leftarrow},\mathsf{s},\mathsf{t} \leadsto \mathsf{s},\mathrm{ap}(\mathsf{c},\mathsf{t}''|\mathsf{n}[\mathsf{t}'])}$$

$$\frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}'])}{x := \text{copy } n_{\uparrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|x\leftarrow\langle \mathsf{n}[\mathsf{t}']\rangle],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}'])}{x := \text{copy } n_{\downarrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|x\leftarrow\langle \mathsf{t}'\rangle],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}']) \quad \langle \mathsf{t}''\rangle \equiv [\![X]\!]\mathsf{s} \quad \mathsf{t}''\#\mathsf{t}}{\text{insert } X \text{ at } n_{\swarrow},\mathsf{s},\mathsf{t} \leadsto \mathsf{s},\mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}''|\mathsf{t}'])}$$

$$\frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}'[\mathsf{t}_1|\mathsf{n}[\mathsf{t}_2]|\mathsf{t}_3])}{n' := \text{lookup } n_{\uparrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|n'\leftarrow \mathsf{n}'],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathsf{t}_1|\mathsf{n}[\mathsf{t}_2]|\mathsf{t}_3}{n' := \text{lookup } n_{\uparrow},\mathsf{s},\mathsf{t} \leadsto \text{fault}}$$

$$\frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}_1]|\mathsf{n}'[\mathsf{t}_2])}{n' := \text{lookup } n_{\rightarrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|n'\leftarrow \mathsf{n}'],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{m}[\mathsf{t}_1|\mathsf{n}[\mathsf{t}_2]])}{n' := \text{lookup } n_{\rightarrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|n'\leftarrow \text{nil}],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathsf{t}_1|\mathsf{n}[\mathsf{t}_2]}{n' := \text{lookup } n_{\rightarrow},\mathsf{s},\mathsf{t} \leadsto \text{fault}}$$

$$\frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}_1|\mathsf{n}'[\mathsf{t}_2]])}{n' := \text{lookup } n_{\searrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|n'\leftarrow \mathsf{n}'],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[0])}{n' := \text{lookup } n_{\searrow},\mathsf{s},\mathsf{t} \leadsto [\mathsf{s}|n'\leftarrow \text{nil}],\mathsf{t}} \qquad \frac{\mathsf{s}(n)=\mathsf{n} \quad \mathsf{t}\not\equiv \mathrm{ap}(\mathsf{c},\mathsf{n}[\mathsf{t}'])}{\mathbb{C}_{\mathrm{up}}(n),\mathsf{s},\mathsf{t} \leadsto \text{fault}}$$

The cases for skip, variable assignment, sequencing, if-then-else, and while-do are omitted as they are standard. For insert and node lookup, only some of the cases are given; the other cases are analogous.

Figure 1. Operational Semantics of the Tree Update Language

empty reference) and tree shapes (trees modulo renaming of identifiers, allowing high-level manipulation of tree structures). The choice of having tree shapes, as opposed to trees, in the store illustrates a seemingly paradoxical property of high-level, imperative tree update. Whilst some way of identifying nodes is required to specify the location of in-place updates, these identifiers are not considered an important part of the high-level structure itself. This follows the spirit of DOM, which assumes object identifiers for nodes and nodeLists that are not present in the underlying XML structure. It is even possible to draw an analogy with heap update, where heap addresses are typically unimportant to the data structures represented in the heap, but necessary for manipulating those structures.

Definition 2.3 (Tree shapes)**.** *Tree shapes* $\mathsf{t}_\circ \in \mathcal{T}_\circ$ are:

$$\mathsf{t}_\circ ::= 0 \mid \circ[\mathsf{t}_\circ] \mid \mathsf{t}_\circ|\mathsf{t}_\circ$$

where $\circ$ is a constant. Write $\langle\mathsf{t}\rangle$ for the shape of a concrete tree t, with $\langle 0\rangle = 0$, $\langle \mathsf{n}[\mathsf{t}]\rangle = \circ[\langle\mathsf{t}\rangle]$ and $\langle \mathsf{t}_1|\mathsf{t}_2\rangle = \langle\mathsf{t}_1\rangle|\langle\mathsf{t}_2\rangle$. Write $\mathsf{t}_1 \simeq \mathsf{t}_2$ when $\langle\mathsf{t}_1\rangle = \langle\mathsf{t}_2\rangle$.

Definition 2.4 (Program state)**.** The program state consists of a *working tree* $\mathsf{t} \in \mathcal{T}$ and a *variable store* $\mathsf{s} \in \mathcal{S}$, which consists of a pair of finite partial functions $\mathsf{s}\colon \mathrm{Var}_{\mathcal{N}} \rightharpoonup_{\mathrm{fin}} (\mathcal{N} \uplus \{\mathrm{nil}\}) \times \mathrm{Var}_{\mathcal{T}_\circ} \rightharpoonup_{\mathrm{fin}} \mathcal{T}_\circ$ mapping node variables $\mathrm{Var}_{\mathcal{N}} = \{m, n, \ldots\}$ to node identifiers or nil, and tree shape variables $\mathrm{Var}_{\mathcal{T}_\circ} = \{x, y, \ldots\}$ to tree shapes. We write $[\mathsf{s}|n \leftarrow \mathsf{n}]$ for the variable store s overwritten with $\mathsf{s}(n) = \mathsf{n}$, and similarly for $[\mathsf{s}|x \leftarrow \mathsf{t}_\circ]$. $\mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s})$ denotes the tree shape variables in $dom(\mathsf{s})$.

To specify node and tree-shape values, our language uses simple expressions. Nodes are specified either with node variables or the constant nil; we forbid direct reference to constants other than nil, as this contradicts the role of identifiers. Tree shapes, meanwhile, are specified using a combination of tree shape variables and constant tree shape structures. E.g., the expression $\circ[x|x]$ describes the tree shape consisting of two copies of the shape x under a single parent node. Since our language includes conditional constructs, we also have simple Boolean expressions, which consist of logical combinations of equality tests on expressions.

Definition 2.5 (Expressions)**.** *Node expressions* $N \in \mathrm{Exp}_{\mathcal{N}}$, *tree shape expressions* $X \in \mathrm{Exp}_{\mathcal{T}_\circ}$ and *Boolean expressions* $B \in \mathrm{Exp}_{\mathbb{B}}$ are defined by:

$$\begin{array}{ll} N ::= n \mid \mathrm{nil} & n \in \mathrm{Var}_{\mathcal{N}} \\ X ::= 0 \mid x \mid \circ[X] \mid X|X & x \in \mathrm{Var}_{\mathcal{T}_\circ} \\ B ::= N = N \mid X = X \mid \mathrm{false} \mid B \Rightarrow B & \end{array}$$

$[\![E]\!]\mathsf{s}$ has the standard semantics.

Our language consists of the standard basic imperative commands together with four tree update commands: deletion, insertion, copying and node lookup. Deletion removes a subtree; insertion adds a tree, specified by a tree shape expression; copying stores the shape of a subtree in a tree shape variable; and node lookup looks up a neighbouring node in the tree structure. These four commands are sufficient to express a wide range of tree manipulation. E.g., replacing a subtree at a node can be expressed using deletion followed by insertion. Also, allocation ('new') can be expressed by the insertion of a literal; e.g., inserting the expression $\circ[0]$ creates a single new node at a given location. It turns out that, given our simple tree structure, the copy command is also derivable, using a combination of lookup, insertion and recursion; nevertheless, we include it in our language for æsthetic reasons.

The four update commands are specified by a single location, given by node variable n. However, there is

still a choice as to the precise action of the update command and, since the options are not inter-derivable, we allow for a number of possibilities: for delete and copy, the action can affect the tree at n or its subforest; for insert, the action can insert as a left sibling, right sibling, first child or last child of n; for node lookup, the action can affect the same positions as insert, as well as the parent node. In Defn. 2.6, the tree update commands therefore all have variants, marked $\star$, for describing the precise location of their action.

Definition 2.6 (Tree Update Language)**.** The *commands* of the tree update language are defined by:

$$
\begin{array}{lll}
\mathbb{C} ::= & \text{skip} & \text{skip} \\
& n := N \mid x := X & \text{variable assignment} \\
& \mathbb{C}_{\text{up}}(n) & \text{tree update at } n \\
& \mathbb{C} \,;\, \mathbb{C} & \text{sequencing} \\
& \text{if } B \text{ then } \mathbb{C} \text{ else } \mathbb{C} & \text{if-then-else} \\
& \text{while } B \text{ do } \mathbb{C} & \text{while-do}
\end{array}
$$

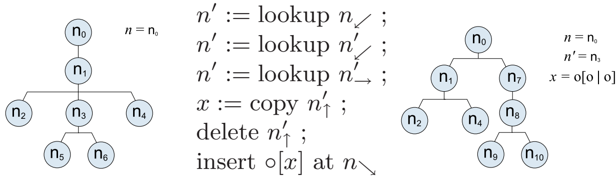
where the tree update commands are:

$$
\begin{array}{lll}
\mathbb{C}_{\text{up}}(n) ::= & \text{delete } n_\star & \star \in \{\uparrow, \downarrow\} \\
& \text{insert } X \text{ at } n_\star & \star \in \{\leftarrow, \rightarrow, \swarrow, \searrow\} \\
& x := \text{copy } n_\star & \star \in \{\uparrow, \downarrow\} \\
& n' := \text{lookup } n_\star & \star \in \{\uparrow, \leftarrow, \rightarrow, \swarrow, \searrow\}
\end{array}
$$

The operational semantics is given in Figure 1, using an evaluation relation $\rightsquigarrow$ relating configuration triples $\mathbb{C}, \mathsf{s}, \mathsf{t}$, terminal states s, t, and faults, where $\text{dom}(\mathsf{s}) \supseteq \text{free}(\mathbb{C})$, the free program variables of $\mathbb{C}$.

A key property of the operational semantics is that all the commands are *local*; the result of a command on a tree is the same as the result of the command on the tree in a larger context. This locality property is essential for our style of reasoning. E.g., consider the behaviour of $n' := \text{lookup } n_\rightarrow$. If the right sibling of n exists then its identifier is stored at n'. If the right sibling does not exist but n has a parent then n' stores the value nil. However, if the node n is not present in the tree, or if the right sibling of n does not exist and n has no parent, then the command must fault.

Example 2.7 (Tree update)**.** Consider a simple program, which traverses a tree (starting at n_0), copies and deletes a subtree (at n_3), and inserts it as the last child of the starting node, under a new node:



3 High-Level Tree Reasoning

We describe CL for analysing our tree structure, and local Hoare reasoning using CL for specifying properties about our tree update language. First, we present the *logical environment* which is a function from logical tree variables to trees. Contrast these tree variables with the tree shape program variables (Definition 2.4), which refer to specific values in the store and are not quantified. We permit our node variables to have the dual role as program variables and logical variables.

Definition 3.1 (Logical environment)**.** An *environment* $\mathsf{e} \in \mathcal{E}$ is a function $\mathsf{e}\colon \text{LVar}_\mathcal{T} \to \mathcal{T}$ mapping logical tree variables $\text{LVar}_\mathcal{T} = \{t, \dots\}$ to trees. Write $[\mathsf{e}|t \leftarrow \mathsf{t}]$ for the environment e overwritten with $\mathsf{e}(t) = \mathsf{t}$.

Local data update typically identifies the portion of data to be replaced, removes it, and inserts new data in the same place. The key idea behind CL is that, in order to reason about such data update, it is necessary to reason about both the data (trees) and the interim contexts. CL applied to trees is built from the standard classical connectives, the structural connectives, and specific operators for trees. The structural connectives consist of the application connective '$\cdot$', its right adjoints $\lhd$ and $\rhd$, and a context formula '$-$' describing the empty context. In fact, we focus on the de Morgan duals '$\blacktriangleleft$' and '$\blacktriangleright$' of the right adjoints. Given a context formula K and tree formulæ P and P': the tree formula $K \cdot P$ describes a tree that can be split into a context satisfying K and a subtree satisfying P; the tree formula $K \blacktriangleleft P$ describes a tree which can be inserted into a context satisfying K to obtain a tree satisfying P; and the context formula $P' \blacktriangleright P$ describes a context which can be placed around a tree satisfying P' to obtain a tree satisfying P. The specific formulæ consist of: the logical variable t describing a specific tree given by the environment; the tree formulæ X and $\langle t \rangle$ describing trees of a certain shape; and the context formulæ $N[K]$, $P|K$ and $K|P$ analysing the branching and composition structure of contexts and, via application, trees.

Definition 3.2 (CL tree formulæ)**.** The tree formulæ $P \in \mathcal{P}$ and context formulæ $K \in \mathcal{K}$ are:

$$
\begin{array}{lll}
P ::= & K ::= & \\
\quad X \mid t \mid \langle t \rangle & \quad N[K] \mid P|K \mid K|P & \text{specific formulæ} \\
\mid K \cdot P \mid K \blacktriangleleft P & \mid - \mid P \blacktriangleright P & \text{structural formulæ} \\
\mid P \Rightarrow P \mid \text{false} & \mid K \Rightarrow K \mid \text{False} & \text{Boolean formulæ} \\
\mid \exists n.\, P \mid \exists t.\, P & \mid \exists n.\, K \mid \exists t.\, K & \text{quantification}
\end{array}
$$

Definition 3.3 (Satisfaction relations)**.** Given environment e and variable store s, the satisfaction relations $\mathsf{e}, \mathsf{s}, \mathsf{t} \vDash_\mathcal{T} P$ and $\mathsf{e}, \mathsf{s}, \mathsf{c} \vDash_\mathcal{C} K$ are: (with $\text{dom}(\mathsf{s}) \supseteq$

free $(P) \cup$ free (K), the free program variables in P, K),

$$\begin{array}{lll}
\mathsf{e,s,t} \vDash_{\mathcal{T}} X & \Leftrightarrow \langle \mathsf{t} \rangle \equiv [\![X]\!]\mathsf{s} \\
\mathsf{e,s,t} \vDash_{\mathcal{T}} t & \Leftrightarrow \mathsf{t} \equiv \mathsf{e}(t) \\
\mathsf{e,s,t} \vDash_{\mathcal{T}} \langle t \rangle & \Leftrightarrow \mathsf{t} \simeq \mathsf{e}(t) \\
\mathsf{e,s,t} \vDash_{\mathcal{T}} K \cdot P & \Leftrightarrow \exists \mathsf{c,t'}.\, \mathsf{t} \equiv \mathsf{ap(c,t')} \wedge \mathsf{e,s,c} \vDash_{\mathcal{C}} K \wedge \mathsf{e,s,t'} \vDash_{\mathcal{T}} P \\
\mathsf{e,s,t} \vDash_{\mathcal{T}} K \blacktriangleleft P & \Leftrightarrow \exists \mathsf{c}.\, \mathsf{ap(c,t)}\downarrow \wedge \mathsf{e,s,c} \vDash_{\mathcal{C}} K \wedge \mathsf{e,s,ap(c,t)} \vDash_{\mathcal{T}} P
\end{array}$$

$$\begin{array}{lll}
\mathsf{e,s,c} \vDash_{\mathcal{C}} N[K] & \Leftrightarrow \exists \mathsf{n,c'}.\, \mathsf{n} = [\![N]\!]\mathsf{s} \wedge \mathsf{c} \equiv \mathsf{n[c']} \wedge \mathsf{e,s,c'} \vDash_{\mathcal{C}} K \\
\mathsf{e,s,c} \vDash_{\mathcal{C}} P|K & \Leftrightarrow \exists \mathsf{t,c'}.\, \mathsf{c} \equiv \mathsf{t|c'} \wedge \mathsf{e,s,t} \vDash_{\mathcal{T}} P \wedge \mathsf{e,s,c'} \vDash_{\mathcal{C}} K \\
\mathsf{e,s,c} \vDash_{\mathcal{C}} K|P & \Leftrightarrow \exists \mathsf{c',t}.\, \mathsf{c} \equiv \mathsf{c'|t} \wedge \mathsf{e,s,c'} \vDash_{\mathcal{C}} K \wedge \mathsf{e,s,t} \vDash_{\mathcal{T}} P \\
\mathsf{e,s,c} \vDash_{\mathcal{C}} - & \Leftrightarrow \mathsf{c} \equiv - \\
\mathsf{e,s,c} \vDash_{\mathcal{C}} P \blacktriangleright P' & \Leftrightarrow \exists \mathsf{t}.\, \mathsf{ap(c,t)}\downarrow \wedge \mathsf{e,s,t} \vDash_{\mathcal{T}} P \wedge \mathsf{e,s,ap(c,t)} \vDash_{\mathcal{T}} P'
\end{array}$$

with standard cases for the classical connectives and quantification. Operator binding, from tightest to loosest, is $\neg$, $\cdot$, $\wedge$, $\vee$, $\{\blacktriangleright, \blacktriangleleft, \triangleright, \triangleleft\}$ and $\Rightarrow$.

Definition 3.4 (Derived formulæ)**.** We use the standard classical logic connectives, and write $N[P]$ for $N[-] \cdot P$ and $P|P'$ for $(P|-) \cdot P'$. We also define:

$$\begin{aligned}
\circ[P] &\triangleq \exists n.\, n[P] \\
\Diamond P &\triangleq \mathrm{True} \cdot P \\
K \triangleleft P &\triangleq \neg(K \blacktriangleleft \neg P) \\
P \triangleright P' &\triangleq \neg(P \blacktriangleright \neg P') \\
P_1 \equiv P_2 &\triangleq (- \wedge (\mathrm{true} \triangleright (P_1 \Leftrightarrow P_2))) \cdot \mathrm{true} \\
P_1 \bowtie P_2 &\triangleq (- \wedge \neg(P_1 \triangleright \neg P_2)) \cdot \mathrm{true}
\end{aligned}$$

$$\begin{aligned}
N_1 = N_2 &\triangleq N_1[0] \equiv N_2[0] \\
X_1 = X_2 &\triangleq X_1 \equiv X_2 \\
\langle P \rangle &\triangleq \exists t.\, t \wedge (\langle t \rangle \bowtie P)
\end{aligned}$$

$\Diamond P$ specifies a tree containing some subtree satisfying P. $K \triangleleft P$ describes a tree that satisfies P whenever it is successfully placed in a context satisfying K, whilst $P \triangleright P'$ describes a context that satisfies P' whenever a tree satisfying P is inserted in it. $P_1 \equiv P_2$ holds iff P_1 and P_2 are satisfied by the same trees. $P_1 \bowtie P_2$ holds iff there is *some* tree satisfying both P_1 and P_2. The expression equalities are self-explanatory. We write B to denote the formula corresponding to the Boolean expression B, made up of these equalities and the standard classical connectives; these formulæ are used to declare Hoare triples for the if-then-else and while-do commands. Finally, a general renaming formula $\langle P \rangle$, specifies all trees satisfying P up to renaming.

Example 3.5 (Formula examples)**.** The tree in Example 2.2 satisfies the following CL formulæ:

(1) $n[\mathrm{true}]$, where $\mathsf{s}(n) = \mathsf{n}_0$ (3) $(0 \blacktriangleright \circ[\circ[0]]) \cdot \mathrm{true}$

(2) $\Diamond(\circ[\circ[0]|\circ[0]])$ (4) $\exists n, t.\, n[(0 \blacktriangleright t) \cdot \langle t \rangle]$

We describe local Hoare reasoning about our update language using CL. Just as for SL, it depends on a fault-avoiding, partial correctness interpretation of the triples: a Hoare triple $\{P\}\, \mathbb{C}\, \{Q\}$ holds iff, for all e,s,t:

$$\begin{array}{c}
\mathrm{free}(\mathbb{C}) \cup \mathrm{free}(P) \cup \mathrm{free}(Q) \subseteq \mathrm{dom}(\mathsf{s}) \wedge \mathsf{e,s,t} \vDash_{\mathcal{T}} P \Rightarrow \\
\mathbb{C}, \mathsf{s}, \mathsf{t} \not\rightsquigarrow \mathrm{fault} \wedge \forall \mathsf{s',t'}.\, \mathbb{C}, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s'}, \mathsf{t'} \Rightarrow \mathsf{e}, \mathsf{s'}, \mathsf{t'} \vDash_{\mathcal{T}} Q
\end{array}$$

Definition 3.6 (Small Axioms)**.** The Small Axioms for the commands given in Figure 1 are in Figure 2.

The Small Axiom for delete $n_{\uparrow}$ has a precondition $n[\mathrm{true}]$ describing a tree with root node n, and a postcondition 0 stating that the result is the empty tree. Notice that $n' :=$ lookup $n_{\rightarrow}$ and $n' :=$ lookup $n_{\searrow}$ have two axioms: one for the behaviour that returns a value n and one for nil. The inference rules include the standard Hoare Logic Rules plus the Frame Rule:

$$\frac{\{P\}\ \mathbb{C}\ \{Q\}}{\{K \cdot P\}\ \mathbb{C}\ \{K \cdot Q\}}\ \mathrm{mod}(\mathbb{C}) \cap \mathrm{free}(K) = \emptyset$$

$\mathrm{mod}(\mathbb{C})$ denotes the modified variables of $\mathbb{C}$. All the commands have weakest preconditions. We can also derive safety preconditions of programs: e.g., $\Diamond n[\circ[\circ[\mathrm{true}]\,|\,\circ[\mathrm{true}]\,|\,\mathrm{true}]\,|\,\mathrm{true}]$ for Example 2.7.

4 Low-Level Heap Reasoning

We give a brief description of the heap update language and SL. See [10] for further details. The only difference in the reasoning is that we assume a logical environment containing tree and tree shape variables, allow comparisons between environment values, and provide quantification over such logical variables.

Definition 4.1 (Heaps and stores)**.** A *heap* $\mathsf{h} \in \mathcal{H}$ is a finite partial function $\mathsf{h}\colon \mathbb{N}^+ \rightharpoonup_{\mathrm{fin}} \mathbb{N}$ from locations $\mathsf{n} \in \mathbb{N}^+$ to values $\mathsf{v} \in \mathbb{N}$, where values are either locations or nil (0). A *variable store* $\mathsf{s} \in \mathcal{S}$ is a function $\mathsf{s} : \mathrm{Var} \to \mathbb{N}$ mapping variables $\mathrm{Var} = \{m, n, \dots\}$ to values. We write $\mathsf{n} \mapsto \mathsf{n}'$ for the singleton heap mapping n to n′, $\mathsf{h} * \mathsf{h}'$ for the union of two *disjoint* heaps h and h′, and $[\mathsf{s}|n \leftarrow \mathsf{n}]$ for s overwritten with $\mathsf{s}(n) = \mathsf{n}$.

Definition 4.2 (Expressions)**.** *Value expressions* $E \in \mathrm{Exp}_{\mathbb{N}}$ and *Boolean expressions* $B \in \mathrm{Exp}_{\mathbb{B}}$ are given by:

$$\begin{aligned}
E &::= \mathsf{v} \mid n \mid E + E \mid E - E \\
B &::= E = E \mid \mathrm{false} \mid B \Rightarrow B
\end{aligned}$$

where $\mathsf{v} \in \mathbb{N}$ and $n \in \mathrm{Var}$. $[\![E]\!]\mathsf{s}$ and $[\![B]\!]\mathsf{s}$ are standard.

Definition 4.3 (Heap update language)**.** The commands of the heap update language are:

$$\begin{array}{lll}
\mathbb{C} ::= & n := E & \text{assignment} \\
& n := \mathrm{cons}(k) & \text{allocation} \\
& \mathrm{dispose}\ E & \text{disposal} \\
& [E] := F & \text{mutation} \\
& n := [E] & \text{lookup}
\end{array}$$

plus skip, sequencing, if-then-else and while-do. fault is dispose nil. See [10] for the operational semantics.

$$
\begin{array}{rll}
\{0\} & \text{skip} & \{0\} \\
\{0 \wedge (n = n_0)\} & n := N & \{0 \wedge (n = N\{n_0/n\})\} \\
\{0 \wedge (X \equiv \langle t \rangle)\} & x := X & \{0 \wedge (x \equiv \langle t \rangle)\} \\
\\
\{n[\text{true}]\} & \text{delete } n_{\uparrow} & \{0\} \\
\{n[\text{true}]\} & \text{delete } n_{\downarrow} & \{n[0]\} \\
\{n[t]\} & \text{insert } X \text{ at } n_{\leftarrow} & \{X|n[t]\} \\
\{n[t]\} & \text{insert } X \text{ at } n_{\swarrow} & \{n[X|t]\} \\
\{n[t]\} & x := \text{copy } n_{\uparrow} & \{n[t] \wedge (x \equiv \langle n[t] \rangle)\} \\
\{n[t]\} & x := \text{copy } n_{\downarrow} & \{n[t] \wedge (x \equiv \langle t \rangle)\} \\
\\
\{m[t_1|n[t]|t_2] \wedge (n = N) \wedge (n' = n_0)\} & n' := \text{lookup } n_{\uparrow} & \{m[t_1|n[t]|t_2] \wedge (n = N\{n_0/n'\}) \wedge (n'{=}m)\} \\
\{n[t]|n_2[t_2] \wedge (n = N) \wedge (n' = n_0)\} & n' := \text{lookup } n_{\rightarrow} & \{n[t]|n_2[t_2] \wedge (n = N\{n_0/n'\}) \wedge (n'{=}n_2)\} \\
\{m[t_1|n[t]] \wedge (n = N) \wedge (n' = n_0)\} & n' := \text{lookup } n_{\rightarrow} & \{m[t_1|n[t]] \wedge (n = N\{n_0/n'\}) \wedge (n'{=}\text{nil})\} \\
\{n[t|n_2[t_2]] \wedge (n = N) \wedge (n' = n_0)\} & n' := \text{lookup } n_{\searrow} & \{n[t|n_2[t_2]] \wedge (n = N\{n_0/n'\}) \wedge (n'{=}n_2)\} \\
\{n[0] \wedge (n = N) \wedge (n' = n_0)\} & n' := \text{lookup } n_{\searrow} & \{n[0] \wedge (n = N\{n_0/n'\}) \wedge (n'{=}\text{nil})\}
\end{array}
$$

Figure 2. Small Axioms for the High-level Commands

We use the Heap Logic of SL, consisting of the separation operator $*$, its right adjoint $-\!\!*$, a points-to operator $\mapsto$ and emp. We assume a logical environment e for tree and tree shape variables, and define tree expressions and formulæ for comparisons and quantification.

Definition 4.4 (Environment)**.** An *environment* $\mathsf{e} \in \mathcal{E}$ is a pair of functions $\mathsf{e}\colon \mathrm{LVar}_{\mathcal{T}} \to \mathcal{T} \times \mathrm{LVar}_{\mathcal{T}_\circ} \to \mathcal{T}_\circ$ mapping logical tree variables $\mathrm{LVar}_{\mathcal{T}} = \{t, \dots\}$ to trees and logical tree shape variables $\mathrm{LVar}_{\mathcal{T}_\circ} = \{x, \dots\}$ to tree shapes. A *tree expression* $T \in \mathrm{Exp}_{\mathcal{T}}$ is defined by:

$$T ::= 0 \mid t \mid E[T] \mid T|T \qquad E \in \mathrm{Exp}_{\mathbb{N}}$$

$[\![T]\!]\mathsf{es}$ is standard.

Definition 4.5 (Heap Logic formulæ)**.** The heap formulæ P are given by:

$$
\begin{array}{rll}
P ::= & \text{emp} \mid E \mapsto E \mid B & \text{specific formulæ} \\
& P * P \mid P -\!\!* P & \text{structural formulæ} \\
& P \Rightarrow P \mid \text{false} & \text{Boolean formulæ} \\
& T \equiv T \mid x \equiv \langle T \rangle & \text{comparisons} \\
& \exists n.P \mid \exists t.P \mid \exists x.P & \text{quantification}
\end{array}
$$

Given an environment e, variable store s and heap h, the semantics of the heap logic is given by a satisfaction relation $\mathsf{e}, \mathsf{s}, \mathsf{h} \vDash_{\mathcal{H}} P$. It is defined inductively on the structure of P. We just give the non-standard cases:

$$
\begin{array}{l}
\mathsf{e,s,h} \vDash_{\mathcal{H}} T \equiv T' \Leftrightarrow [\![T]\!]\mathsf{es} \equiv [\![T']\!]\mathsf{es} \\
\mathsf{e,s,h} \vDash_{\mathcal{H}} x \equiv \langle T \rangle \Leftrightarrow \mathsf{e}(x) \equiv \langle [\![T]\!]\mathsf{es} \rangle
\end{array}
$$

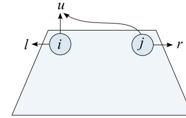
We use the following formulæ to describe the existential magic wand, node expression equality requiring in addition that the heap is empty, adjacent cells in the heap, and shape equivalence of tree expressions:

$$
\begin{array}{c}
P -\!\!*_{\exists}\, Q \triangleq \neg(P -\!\!* \neg Q) \\
E \doteq F \triangleq (E = F) \wedge \text{emp} \\
E \mapsto F_\text{o}, \dots, F_n \triangleq (E \mapsto F_\text{o}) * \cdots * (E + n \mapsto F_n) \\
T \simeq T' \triangleq \exists x.\, x \equiv \langle T \rangle \wedge x \equiv \langle T' \rangle
\end{array}
$$

The Hoare triples of Separation Logic [5, 10] are analogous to those for CL. See [10] for details.

5 Implementing Trees and Contexts

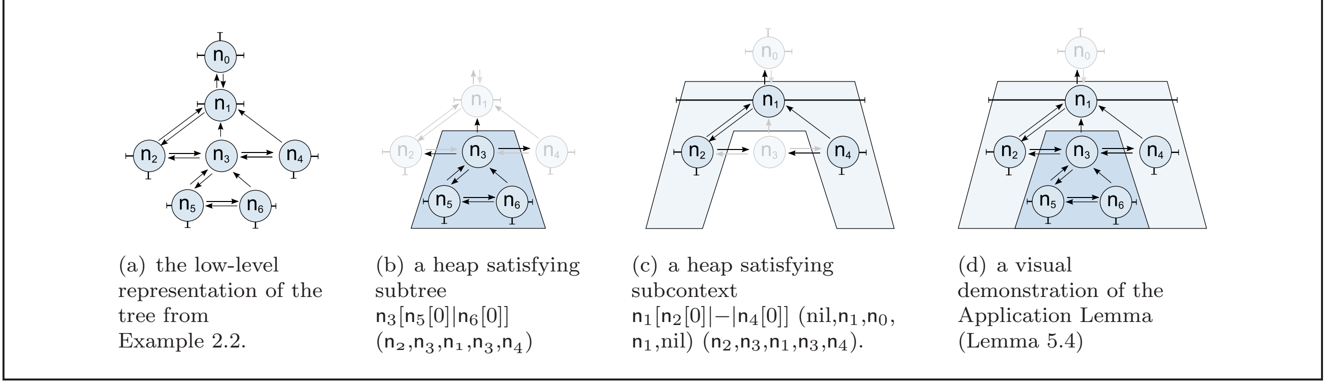
Trees are implemented using a simple linked structure, consisting of a collection of heap cells corresponding to the tree nodes, each containing links to the node's parent, previous and next siblings, and first child. To each tree node identifier n, we associate the heap location, also denoted n. The distance between such heap nodes must be at least 4 for the pointers. Similarly, the high-level tree value nil is identified with the heap value nil. E.g., the tree in Example 2.2 has the heap representation given in Figure 3a. To formally define this structure, we introduce an abstract predicate describing the set of states representing a subtree (subforest) t with an external 'interface' given by a tuple (l,i,u,j,r) which describes the first (i) and last (j) of the top-level nodes in the representation, and the targets of the left (l), right (r) and up (u) pointers:



In Figure 3a, the interface is $(\text{nil}, \mathsf{n}_0, \text{nil}, \mathsf{n}_0, \text{nil})$.

Definition 5.1 (Tree representation)**.** A subtree t with interface $I = (l, i, u, j, r)$ is represented inductively by:

$$
\begin{array}{l}
\text{subtree } 0\ (l, i, u, j, r) \triangleq (l \doteq j) * (i \doteq r) \\
\text{subtree } \mathsf{n}[\mathsf{t}]\ (l, i, u, j, r) \triangleq \exists d_i, d_j.\, (i \doteq \mathsf{n}) * (j \doteq \mathsf{n}) \\
\quad * \, \mathsf{n} \mapsto l, u, r, d_i * \text{subtree } \mathsf{t}\ (\text{nil}, d_i, \mathsf{n}, d_j, \text{nil}) \\
\text{subtree } \mathsf{t}_1|\mathsf{t}_2\ (l, i, u, j, r) \triangleq \exists k_l, k_r. \\
\quad \text{subtree } \mathsf{t}_1\ (l, i, u, k_l, k_r) * \text{subtree } \mathsf{t}_2\ (k_l, k_r, u, j, r)
\end{array}
$$



(a) the low-level representation of the tree from Example 2.2.

(b) a heap satisfying subtree $\mathsf{n}_3[\mathsf{n}_5[0]|\mathsf{n}_6[0]]$ $(\mathsf{n}_2,\mathsf{n}_3,\mathsf{n}_1,\mathsf{n}_3,\mathsf{n}_4)$

(c) a heap satisfying subcontext $\mathsf{n}_1[\mathsf{n}_2[0]|-|\mathsf{n}_4[0]]$ $(\text{nil},\mathsf{n}_1,\mathsf{n}_0,\mathsf{n}_1,\text{nil})$ $(\mathsf{n}_2,\mathsf{n}_3,\mathsf{n}_1,\mathsf{n}_3,\mathsf{n}_4)$.

(d) a visual demonstration of the Application Lemma (Lemma 5.4)

Figure 3. Representing Trees and Contexts in the Heap

Write tree $\mathsf{t}\ i \triangleq \exists j.\,\text{subtree}\ \mathsf{t}\ (\text{nil}, i, \text{nil}, j, \text{nil})$ to represent top-level trees. Write subtree I to describe an arbitrary subtree with interface I, and subtree $T\ I$ for the extension of subtree to tree expressions.

The low-level representation of the subtree $\mathsf{n}_3[\mathsf{n}_5|\mathsf{n}_6]$ from Example 2.2 is given in Figure 3b. The nodes n_2, n_1 and n_4 form an essential part of the subtree interface. Reasoning about these low-level subtrees requires reasoning about this additional interface (the crust, Defn. 8.1). The 0 case is a little confusing. Consider Figure 3c. If 0 is put into the hole, the n_4 points to $j = l = n_2$, and n_2 points to $i = r = n_4$.

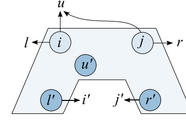
We now model the high-level variable store. Node variables n are translated directly to low-level name variables n. Tree shape variables take up 'space' and are represented in the heap. We associate every tree shape variable x with a unique name variable $\&x$, which points to a heap location containing a concrete representation of its shape value. Since low-level computations may also use other name variables, which do not correspond to anything at the high-level, we relate the high-level state to a *set* of low-level states.

Definition 5.2 (Tree and store representation)**.** Given tree t and store s_t, the set $[\![\mathsf{s}_t, \mathsf{t}]\!]$ is defined by:

$$(\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!] \Leftrightarrow \mathsf{s}_h \supseteq \mathsf{s}_t|_{\text{Var}_\mathcal{N}} \wedge \mathsf{h} = \mathsf{h}_\mathsf{t} * \prod_{x \in \text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x \wedge$$
$$\exists \mathsf{n}.\, \mathsf{s}_h, \mathsf{h}_\mathsf{t} \vDash_\mathcal{H} \text{tree}\ \mathsf{t}\ \mathsf{n} \wedge$$
$$\forall x \in \text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t).\, \exists \mathsf{t}_x.\langle \mathsf{t}_x \rangle \equiv \mathsf{s}_t(x) \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_\mathcal{H} \text{tree}\ \mathsf{t}_x\ \&x$$

where the $\prod$ denotes iterated separation.

Subcontexts are similar to subtrees, except that as well as their 'external' interface, they also have an 'internal' interface for the hole given by (l', i', u', j', r'), where l', u',r' describe the sibling and parent nodes of the context hole, and i', j' describe the first and last locations in the hole to which they point:



Definition 5.3 (Context representation)**.** A subcontext c with a top interface (l, i, u, j, r) and a hole interface (l', i', u', j', r') is defined inductively as follows:

$$\text{subcontext}\ -\ (l,i,u,j,r)\ (l',i',u',j',r') \triangleq$$
$$(l \doteq l') * (i \doteq i') * (u \doteq u') * (j \doteq j') * (r \doteq r')$$
$$\text{subcontext}\ \mathsf{n}[\mathsf{c}]\ (l,i,u,j,r)\ (l',i',u',j',r') \triangleq \exists d_i, d_j.\,(i \doteq \mathsf{n}) * (j \doteq \mathsf{n})$$
$$* \mathsf{n} \mapsto l,u,r,d_i * \text{subcontext}\ \mathsf{c}\ (\text{nil},d_i,n,d_j,\text{nil})\ (l',i',u',j',r')$$
$$\text{subcontext}\ \mathsf{t}|\mathsf{c}\ (l,i,u,j,r)\ (l',i',u',j',r') \triangleq \exists k_l, k_r.$$
$$\text{subtree}\ \mathsf{t}\ (l,i,u,k_l,k_r) * \text{subcontext}\ \mathsf{c}\ (k_l,k_r,u,j,r)\ (l',i',u',j',r')$$
$$\text{subcontext}\ \mathsf{c}|\mathsf{t}\ (l,i,u,j,r)\ (l',i',u',j',r') \triangleq \exists k_l, k_r.$$
$$\text{subcontext}\ \mathsf{c}\ (l,i,u,k_l,k_r)\ (l',i',u',j',r') * \text{subtree}\ \mathsf{t}\ (k_l,k_r,u,j,r)$$

We write context $\mathsf{c}\ i\ I' \triangleq \exists j.\,\text{subcontext}\ \mathsf{c}\ (\text{nil},i,\text{nil},j,\text{nil})\ I'$ to represent top-level contexts, and subcontext $I\ I'$ to describe an arbitrary subcontext with interfaces I and I'. The low-level representation of the subcontext $\mathsf{n}_1[\mathsf{n}_2| - |\mathsf{n}_4]$ is in Figure 3c with n_0 and n_3 part of the external and internal interface.

Lemma 5.4 (Application Lemma)**.** *Context application corresponds to star-separation across interfaces: i.e.,* subtree $\text{ap}(\mathsf{c}, \mathsf{t})\ I \Leftrightarrow \exists I'.\,\text{subcontext}\ \mathsf{c}\ I\ I' *$ subtree $\mathsf{t}\ I'$*. Figure 3d illustrates the application of the subcontext in Figure 3c to the subtree in Figure 3b.*

6 Language Translation

We give a translation of our high-level language into the low-level language. Let $\mathsf{n} \mapsto l,u,r,d$. Then:

$$n.\text{left} \triangleq n \quad n.\text{up} \triangleq n{+}1 \quad n.\text{right} \triangleq n{+}2 \quad n.\text{down} \triangleq n{+}3$$
$$n := \text{new-node}() \triangleq n := \text{cons}(4)$$
$$\text{dispose-node}\ n \triangleq \begin{array}{l}\text{dispose}\ n.\text{left}\ ;\ \text{dispose}\ n.\text{up}\ ;\\ \text{dispose}\ n.\text{right}\ ;\ \text{dispose}\ n.\text{down}\end{array}$$

We use three utility functions for updating forests:

dispose-forest $n \triangleq$
  if $n = \text{nil}$ then skip else
    r:=[n.right] ; dispose-forest r ;
    d:=[n.down] ; dispose-forest d ; dispose-node n
(i', j') := copy-forest n $(l', u', r') \triangleq$
  if $n = \text{nil}$ then $i' := r'$; $j' := l'$ else
    i' := new-node() ; [i'.left] := l' ; [i'.up] := u' ;
    r:=[n.right] ; (r'_i, j'):=copy-forest r (i', u', r') ;
    d:=[n.down] ; (d'_i, d'_j):=copy-forest d (nil,i',nil) ;
    [i'.right] := r'_i ; [i'.down] := d'_i
b := compare-forests n $n' \triangleq$
  if (n=nil $\wedge$ n'=nil) then $b := 1$ else
  if (n=nil $\vee$ n'=nil) then $b := 0$ else
    r := [n.right] ; r' := [n'.right] ; b := compare-forests r r' ;
    if $b = 0$ then skip else
      d:=[n.down] ; d':=[n'.down] ; b := compare-forests d d'

dispose-forest n just disposes the subforest starting at n. The pointer surgery is given in the translation. Recall that, in our high-level language, tree shape variables can be assigned new values using tree shape expressions, which are a mix of tree shape variables and constants. In our implementation, the different tree shape variable values are represented at different locations in the heap. We define a utility (i,j) := copy-expression X (l,u,r), for copying the tree shape expressions to locations in the heap, by:

(i,j) := copy-expression 0 $(l,u,r) \triangleq i := r$; $j := l$
(i,j) := copy-expression x $(l,u,r) \triangleq (i,j)$:=copy-forest &x (l,u,r)
(i,j) := copy-expression $\circ[X]$ $(l,u,r) \triangleq$
    i := new-node() ; j:=i ; [i.left]:=l ; [i.up]:=u ; [i.right]:=r ;
    (d_i,d_j) := copy-expression X (nil,i,nil) ; [i.down] := d_i
(i,j) := copy-expression $X_1|X_2$ $(l,u,r) \triangleq$
    (i,k_l) := copy-expression X_1 (l,u,nil) ;
    if i = nil then (i,j) := copy-expression X_2 (l,u,r)
    else (k_r,j) := copy-expression X_2 (k_l,u,r) ; [k_l.right] := k_r

Since comparing tree shape expressions is possible in high-level Boolean expressions B, but not in low-level ones, high-level Booleans are translated in two stages: the first, $[\![B]\!]_1$, translates any comparisons of tree shape expressions in B into update commands that copy and compare the values of the expressions; the second, $[\![B]\!]_2$, returns the low-level Boolean expression corresponding to B, using the results of the comparisons performed above to manage comparison between tree shape expressions. In order to perform a conditional test on B at the low-level, we first execute $[\![B]\!]_1$ and then perform the test $[\![B]\!]_2$.

For each pair X and X', we associate a fresh name variable $b_{(X=X')}$. The translation $[\![-]\!]_1$ from high-level Boolean expressions to low-level programs and the translation $[\![-]\!]_2$ from high-level Boolean expressions to low-level Boolean expressions is defined by:

$[\![N = N']\!]_1 \triangleq$ skip
$[\![X = X']\!]_1 \triangleq (i,j)$:= copy-expression X (nil, nil, nil) ;
  (i',j') := copy-expression X' (nil, nil, nil) ;
  $b_{(X=X')}$:= compare-forests i i' ;
  dispose-forest i ; dispose-forest i'
$[\![\text{false}]\!]_1 \triangleq$ skip $[\![B \Rightarrow B']\!]_1 \triangleq [\![B]\!]_1$; $[\![B']\!]_1$

$[\![N{=}N']\!]_2 \triangleq N{=}N'$ $[\![\text{false}]\!]_2 \triangleq$ false
$[\![X{=}X']\!]_2 \triangleq b_{(X=X')}{=}1$ $[\![B \Rightarrow B']\!]_2 \triangleq [\![B]\!]_2 \Rightarrow [\![B']\!]_2$

We give the translation of the high-level update language into the low-level language, using the utility functions to do the hard work and then pointer surgery to maintain the tree invariant. E.g., the translation of the command delete $n_\uparrow$ first removes n's subforest using dispose-forest, then removes n itself using dispose-node, and finally cleans up the pointers where n used to be using simple pointer mutation.

Definition 6.1 (Update language translation)**.** The translation is given in Figure 4.

Theorem 6.2 (Relating the operational semantics)**.** *The translation in Figure 3 satisfies the properties:*

(a) $\mathbb{C},\mathsf{s}_t,\mathsf{t} \rightsquigarrow \mathsf{s}'_t,\mathsf{t}'$
  $\Rightarrow \forall(\mathsf{s}_h,\mathsf{h}) \in [\![\mathsf{s}_t,\mathsf{t}]\!].\ \exists(\mathsf{s}'_h,\mathsf{h}') \in [\![\mathsf{s}'_t,\mathsf{t}']\!].\ [\![\mathbb{C}]\!],\mathsf{s}_h,\mathsf{h} \rightsquigarrow \mathsf{s}'_h,\mathsf{h}'$
(b) $\mathbb{C},\mathsf{s}_t,\mathsf{t} \rightsquigarrow \text{fault} \Rightarrow \exists(\mathsf{s}_h,\mathsf{h}) \in [\![\mathsf{s}_t,\mathsf{t}]\!].\ [\![\mathbb{C}]\!],\mathsf{s}_h,\mathsf{h} \rightsquigarrow \text{fault}$
(c) $[\![\mathbb{C}]\!],\mathsf{s}_h,\mathsf{h} \rightsquigarrow \mathsf{s}'_h,\mathsf{h}' \wedge (\mathsf{s}_h,\mathsf{h}) \in [\![\mathsf{s}_t,\mathsf{t}]\!] \wedge \mathbb{C},\mathsf{s}_t,\mathsf{t} \not\rightsquigarrow \text{fault}$
  $\Rightarrow \exists \mathsf{s}'_t,\mathsf{t}'.\ (\mathsf{s}'_h,\mathsf{h}') \in [\![\mathsf{s}'_t,\mathsf{t}']\!] \wedge \mathbb{C},\mathsf{s}_t,\mathsf{t} \rightsquigarrow \mathsf{s}'_t,\mathsf{t}'$
(d) $[\![\mathbb{C}]\!],\mathsf{s}_h,\mathsf{h} \rightsquigarrow \text{fault} \wedge (\mathsf{s}_h,\mathsf{h}) \in [\![\mathsf{s}_t,\mathsf{t}]\!] \Rightarrow \mathbb{C},\mathsf{s}_t,\mathsf{t} \rightsquigarrow \text{fault}$

The translation is *fault-avoiding* (d), in that a low-level translation will never fault if the high-level command does not fault. However, it is not always *fault-preserving* in that a specific low-level execution might not fault even if the high-level command does fault. This is due to the necessity of keeping the tree store in the heap. E.g., if a node n is not in the tree but happens to be in the heap representation of the store, then the command delete $n_\uparrow$ would delete part of the store, rather than fault. For this reason, (c) requires the additional premise that the high-level command does not fault.

7 Context Logic Translation

Our next step is to translate the CL formulæ to the Separation Logic formulæ. There are two parts to the translation. One part provides the translated SL formulæ $[\![P]\!]^I$ and $[\![K]\!]^I_{I'}$, which specify the corresponding subtrees and subcontexts with their appropriate interfaces. Now recall that the low-level heap not only represents the main data structure (the trees

$$
\begin{array}{l}
[\![\mathrm{skip}]\!] \triangleq \mathrm{skip}\\[4pt]
[\![n := N]\!] \triangleq n := N\\[4pt]
[\![x := X]\!] \triangleq\\
\quad (i,j){:=}\text{copy-expression } X \ (\mathrm{nil},\mathrm{nil},\mathrm{nil})\ ;\\
\quad \text{dispose-forest } \&x\ ;\\
\quad \&x := i\\[4pt]
[\![\mathrm{delete}\ n_{\uparrow}]\!] \triangleq\\
\quad d := [n.\mathrm{down}]\ ;\ r := [n.\mathrm{right}]\ ;\\
\quad u := [n.\mathrm{up}]\ ;\ l := [n.\mathrm{left}]\ ;\\
\quad \text{dispose-forest } d\ ;\ \text{dispose-node } n\ ;\\
\quad \mathrm{if}\ r \neq \mathrm{nil}\ \mathrm{then}\ [r.\mathrm{left}] := l\ ;\\
\quad \mathrm{if}\ l \neq \mathrm{nil}\ \mathrm{then}\ [l.\mathrm{right}] := r\ \mathrm{else}\\
\qquad \mathrm{if}\ u \neq \mathrm{nil}\ \mathrm{then}\ [u.\mathrm{down}] := r\\[4pt]
[\![\mathrm{delete}\ n_{\downarrow}]\!] \triangleq\\
\quad d := [n.\mathrm{down}]\ ;\ \text{dispose-forest } d\ ;\\
\quad [n.\mathrm{down}] := \mathrm{nil}\\[4pt]
[\![\mathbb{C}_1\ ;\ \mathbb{C}_2]\!] \triangleq [\![\mathbb{C}_1]\!]\ ;\ [\![\mathbb{C}_2]\!]
\end{array}
$$

$$
\begin{array}{l}
[\![\mathrm{insert}\ X\ \mathrm{at}\ n_{\leftarrow}]\!] \triangleq\\
\quad l := [n.\mathrm{left}]\ ;\ u := [n.\mathrm{up}]\ ;\\
\quad (i,j){:=}\text{copy-expression } X\ (l,u,n)\ ;\\
\quad [n.\mathrm{left}] := j\ ;\\
\quad \mathrm{if}\ l \neq \mathrm{nil}\ \mathrm{then}\\
\qquad [l.\mathrm{right}] := i\ \mathrm{else}\ [u.\mathrm{down}] := i\\[4pt]
[\![\mathrm{insert}\ X\ \mathrm{at}\ n_{\swarrow}]\!] \triangleq\\
\quad d := [n.\mathrm{down}]\ ;\\
\quad (i,j){:=}\text{copy-expression } X\ (\mathrm{nil},n,d)\ ;\\
\quad \mathrm{if}\ d = \mathrm{nil}\ \mathrm{then\ skip\ else}\ [d.\mathrm{left}] := j\ ;\\
\quad [n.\mathrm{down}] := i\\[4pt]
[\![x := \mathrm{copy}\ n_{\uparrow}]\!] \triangleq\\
\quad \text{dispose-forest } \&x\ ;\\
\quad \&x := \text{new-node}()\ ;\\
\quad d := [n.\mathrm{down}]\ ;\\
\quad (i,j){:=}\text{copy-forest } d\ (\mathrm{nil},\&x,\mathrm{nil})\ ;\\
\quad [\&x.\mathrm{down}] := i\\[4pt]
[\![\mathrm{if}\ B\ \mathrm{then}\ \mathbb{C}_1\ \mathrm{else}\ \mathbb{C}_2]\!] \triangleq\\
\quad [\![B]\!]_1\ ;\ \mathrm{if}\ [\![B]\!]_2\ \mathrm{then}\ [\![\mathbb{C}_1]\!]\ \mathrm{else}\ [\![\mathbb{C}_2]\!]
\end{array}
$$

$$
\begin{array}{l}
[\![x := \mathrm{copy}\ n_{\downarrow}]\!] \triangleq\\
\quad \text{dispose-forest } \&x\ ;\\
\quad d := [n.\mathrm{down}]\ ;\\
\quad (\&x,j){:=}\text{copy-forest } d\ (\mathrm{nil},\mathrm{nil},\mathrm{nil})\\[4pt]
[\![n' := \mathrm{lookup}\ n_{\uparrow}]\!] \triangleq\\
\quad n' := [n.\mathrm{up}]\ ;\ \mathrm{if}\ n' = \mathrm{nil}\ \mathrm{then}\\
\qquad \mathrm{fault\ else\ skip}\\[4pt]
[\![n' := \mathrm{lookup}\ n_{\rightarrow}]\!] \triangleq\\
\quad n' := [n.\mathrm{right}]\ ;\ u := [n.\mathrm{up}];\\
\quad \mathrm{if}\ (n' = \mathrm{nil} \wedge u = \mathrm{nil})\ \mathrm{then}\\
\qquad \mathrm{fault\ else\ skip}\\[4pt]
[\![n' := \mathrm{lookup}\ n_{\searrow}]\!] \triangleq\\
\quad n' := [n.\mathrm{down}]\ ;\\
\quad \mathrm{if}\ n' = \mathrm{nil}\ \mathrm{then\ skip\ else}\\
\qquad n'' := [n'.\mathrm{right}]\ ;\\
\qquad \mathrm{while}\ n'' \neq \mathrm{nil}\ \mathrm{do}\\
\qquad (n' := n''\ ;\ n'' := [n'.\mathrm{right}])\\[4pt]
[\![\mathrm{while}\ B\ \mathrm{do}\ \mathbb{C}]\!] \triangleq [\![B]\!]_1\ ;\\
\quad \mathrm{while}\ [\![B]\!]_2\ \mathrm{do}\ ([\![\mathbb{C}]\!]\ ;\ [\![B]\!]_1)
\end{array}
$$

Figure 4. Language Translation

or contexts), but also represents the values of the tree-shaped variables, with $\&x$ denoting the heap address representing the value of tree shape variable x. The other part declares these values in the heap by $[\![P]\!]^I_{\vec{x}} \triangleq \exists\vec{x}.\,\big([\![P]\!]^I * \prod_{x\in\vec{x}}(\exists t.\,\langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\big)$ where $\{\vec{x}\}$ contains the free tree shape variables in P. This formula declares that, in some part of the heap disjoint from $[\![P]\!]^I$, there exists a disjoint tree t at address $\&x$ for each x, where the tree shape variable x has value $\langle t\rangle$ in the logical environment. Since the value of x is determined precisely by the heap at $\&x$, it is possible to existentially quantify this logical variable x.

Definition 7.1 (Logic translation)**.** The translation of a CL formula P into a heap formula $[\![P]\!]_{\vec{x}}$, where $\{\vec{x}\}$ contains the tree shape variables in P, is given by $[\![P]\!]_{\vec{x}} \triangleq \exists i,j.\,[\![P]\!]^{(\mathrm{nil},i,\mathrm{nil},j,\mathrm{nil})}_{\vec{x}}$ where $[\![P]\!]^I_{\vec{x}} \triangleq \exists\vec{x}.\,\big([\![P]\!]^I * \prod_{x\in\vec{x}}(\exists t.\,\langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\big)$, and $[\![P]\!]^I$ and $[\![K]\!]^I_{I'}$ are defined inductively by:

$$
\begin{aligned}
[\![0]\!]^{(l,i,u,j,r)} &\triangleq (l \doteq j) * (i \doteq r)\\
[\![x]\!]^I &\triangleq \exists t.\,\langle t\rangle \equiv x \wedge \mathrm{subtree}\ t\ I\\
[\![\circ[X]]\!]^{(l,i,u,j,r)} &\triangleq \exists d_i,d_j.\,(i\doteq j) * i\mapsto l,u,r,d_i * [\![X]\!]^{(\mathrm{nil},d_i,i,d_j,\mathrm{nil})}\\
[\![X|X']\!]^{(l,i,u,j,r)} &\triangleq \exists k_l,k_r.\,[\![X]\!]^{(l,i,u,k_l,k_r)} * [\![X']\!]^{(k_l,k_r,u,j,r)}\\
[\![t]\!]^I &\triangleq \mathrm{subtree}\ t\ I\\
[\![\langle t\rangle]\!]^I &\triangleq \exists t'.\,t' \simeq t \wedge \mathrm{subtree}\ t'\ I\\
[\![K \cdot P]\!]^I &\triangleq \exists I'.\,[\![K]\!]^I_{I'} * [\![P]\!]^{I'}\\
[\![K \blacktriangleleft P]\!]^I &\triangleq \mathrm{subtree}\ I \wedge \exists I'.\,[\![K]\!]^{I'}_I \mathrel{-\!\!*_\exists} [\![P]\!]^{I'}\\
[\![P \Rightarrow P']\!]^I &\triangleq \mathrm{subtree}\ I \wedge \big([\![P]\!]^I \Rightarrow [\![P']\!]^I\big)
\end{aligned}
$$

$$
\begin{aligned}
[\![N[K]]\!]^{(l,i,u,j,r)}_{I'} &\triangleq \exists d_i,d_j.\,\begin{array}{l}(i\doteq N) * (j\doteq N) * N\mapsto l,u,r,d_i\ *\\ [\![K]\!]^{(\mathrm{nil},d_i,n,d_j,\mathrm{nil})}_{I'}\end{array}\\
[\![P|K]\!]^{(l,i,u,j,r)}_{I'} &\triangleq \exists k_l,k_r.\,[\![P]\!]^{(l,i,u,k_l,k_r)} * [\![K]\!]^{(k_l,k_r,u,j,r)}_{I'}\\
[\![K|P]\!]^{(l,i,u,j,r)}_{I'} &\triangleq \exists k_l,k_r.\,[\![K]\!]^{(l,i,u,k_l,k_r)}_{I'} * [\![P]\!]^{(k_l,k_r,u,j,r)}\\
[\![-]\!]^{(l,i,u,j,r)}_{(l',i',u',j',r')} &\triangleq (l\doteq l')*(i\doteq i')*(u\doteq u')*(j\doteq j')*(r\doteq r')\\
[\![P \blacktriangleright P']\!]^I_{I'} &\triangleq \mathrm{subcontext}\ I\ I' \wedge [\![P]\!]^{I'} \mathrel{-\!\!*_\exists} [\![P']\!]^I\\
[\![K \Rightarrow K']\!]^I_{I'} &\triangleq \mathrm{subcontext}\ I\ I' \wedge \big([\![K]\!]^I_{I'} \Rightarrow [\![K']\!]^I_{I'}\big)
\end{aligned}
$$

plus obvious cases for false, False and quantification. We write $[\![B]\!]^I$ for the translation of the formula corresponding to the Boolean expression B, as in Defn. 3.4.

The translation of the tree formulæ is mostly straightforward. The tree constructor formulæ are handled in an analogous fashion to Defn. 5.1. Note that the tree shape formula X is translated by its various cases: $[\![0]\!]^I$, $[\![x]\!]^I$, $[\![\circ[X]]\!]^I$ and $[\![X|X']\!]^I$. Context application corresponds to star separation across some interface I', as in Lemma 5.4. The translation of the two modalities, $\blacktriangleleft$ and $\blacktriangleright$, is a bit more subtle. For example, a heap representing a subtree $K \blacktriangleleft P$ with interface I satisfies the formula $\exists I'.\,[\![K]\!]^{I'}_I \mathrel{-\!\!*_\exists} [\![P]\!]^{I'}$, which states that it is possible to add a heap describing a subcontext K with interfaces I' and I (for some I') and obtain a heap describing a subtree P with interface I'. However, this alone does not ensure that the original heap describes a subtree; for that, we need an additional formula subtree I. The same requirement is necessary when translating implication. The transla-

tion of the tree variables and quantification is routine. The translation of the context formulæ is analogous.

Lemma 7.2 (Type Soundness)**.** *The translation is type sound: for all P and $\{\vec{x}\} \supseteq \text{free}_{\mathcal{T}_\circ}(P)$,*

$$\forall \mathsf{e}, \mathsf{s}_h, \mathsf{h}.\ \mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{x}} \Rightarrow \exists \mathsf{s}_t, \mathsf{t}.\ \begin{array}{l}(\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!] \wedge \\ \text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t) = \{\vec{x}\}\end{array}$$

Lemma 7.3 (Soundness and completeness)**.** *The translation is sound and complete: for all P,*

$$\begin{array}{c}\forall \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}, \mathsf{s}_h, \mathsf{h}.\ (\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!] \Rightarrow \\ (\mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \Leftrightarrow \mathsf{e}_t, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)})\end{array}$$

There is an analogous result for context formulæ.

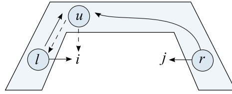
Theorem 7.4 (Hoare Triple equivalence)**.** *For all CL formulæ P, Q, tree update commands $\mathbb{C}$ and tree shape variables $\{\vec{x}\} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P, Q)$:*

$$\{P\}\ \mathbb{C}\ \{Q\} \Leftrightarrow \{[\![P]\!]_{\vec{x}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{x}}\}$$

This theorem relates Hoare triples on high-level trees with Hoare triples on low-level, *complete* trees where the interface is $I = (\text{nil}, i, \text{nil}, j, \text{nil})$ for some i and j.

8 High-level/Low-level Reasoning

To get a true relationship between the high-level and low-level reasoning, we must extend Theorem 7.4 to link Hoare triples on high-level trees to Hoare triples on low-level subtrees with arbitrary interfaces. This is not straightforward as the high-level and low-level footprints are different. For example, the high-level command delete $n_\uparrow$ affects just a subtree at n, whereas its translation $[\![\text{delete } n_\uparrow]\!]$ has the potential to affect the parent, left-sibling and right-sibling of n (if they exist). At the low level, we must therefore introduce an additional predicate describing this *crust* of nodes surrounding the subtree. For the implementation studied in this paper, the crust predicate specifies properties of the nodes immediately near the subtree:

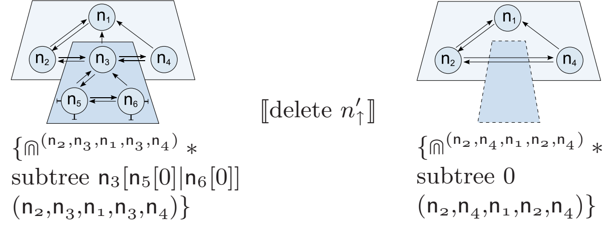


The crust formula is subtle. Given interface $I = (l, i, u, j, r)$, it specifies the potential footprint associated with the pointer surgery: if l and r denote nodes then they must be in the crust; and if $l = \text{nil}$ and u denotes a node then u must be in the crust.

Definition 8.1 (Crust)**.** The tree crust at an interface (l,i,u,j,r), denoted $\Cap^{(l,i,u,j,r)}$, is defined by:

$$\begin{array}{c}\Cap^{(l,i,u,j,r)} \triangleq \exists r_r, d_r, l_l, d_l, u_u, r_u.\ (r{\mapsto}j,u,r_r,d_r \vee r \doteq \text{nil}) * \\ \Big(l{\mapsto}l_l,u,i,d_l \vee \big(l \doteq \text{nil} * (u{\mapsto}l_l,u_u,r_u,i \vee u \doteq \text{nil})\big)\Big)\end{array}$$

Example 8.2 (Crust example)**.** Consider delete $n'_\uparrow$ from Example 2.7, where $\mathsf{s}(n') = \mathsf{n}_3$. The translated command $[\![\text{delete } n'_\uparrow]\!]$ has the following specification:



$\{\Cap^{(\mathsf{n}_2,\mathsf{n}_3,\mathsf{n}_1,\mathsf{n}_3,\mathsf{n}_4)} *$ subtree $\mathsf{n}_3[\mathsf{n}_5[0]|\mathsf{n}_6[0]]$ $(\mathsf{n}_2,\mathsf{n}_3,\mathsf{n}_1,\mathsf{n}_3,\mathsf{n}_4)\}$ $\quad [\![\text{delete } n'_\uparrow]\!] \quad$ $\{\Cap^{(\mathsf{n}_2,\mathsf{n}_4,\mathsf{n}_1,\mathsf{n}_2,\mathsf{n}_4)} *$ subtree 0 $(\mathsf{n}_2,\mathsf{n}_4,\mathsf{n}_1,\mathsf{n}_2,\mathsf{n}_4)\}$

Theorem 8.3 (Axiom translation)**.** *For every Small Axiom $\{P\}\ \mathbb{C}\ \{Q\}$ in Defn. 3.6, the low-level Hoare triple is derivable for $\{\vec{x}\} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P)$:*

$$\begin{array}{c}\{\exists i, j.\, \Cap^{(l,i,u,j,r)} * [\![P]\!]_{\vec{x}}^{(l,i,u,j,r)}\} \\ [\![\mathbb{C}]\!] \\ \{\exists i, j.\, \Cap^{(l,i,u,j,r)} * [\![Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\}\end{array}$$

Theorem 8.4 (Inference rule translation)**.** *For every inference rule in Defn. ??, there is a derivable low-level rule obtained by replacing all high-level Hoare triples $\{P\}\ \mathbb{C}\ \{Q\}$ with the low-level Hoare triples*

$$\{\exists i, j.\, \Cap^{(l,i,u,j,r)} * [\![P]\!]_{\vec{x}}^{(l,i,u,j,r)}\}\ [\![\mathbb{C}]\!]\ \{\exists i, j.\, \Cap^{(l,i,u,j,r)} * [\![Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\}.$$

where $\{\vec{x}\} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P)$.

Corollary 8.5. *Any triple $\{P\}\ \mathbb{C}\ \{Q\}$ derivable at the high-level can also be derived at the low level: z $\vdash \{P\}\ \mathbb{C}\ \{Q\} \Rightarrow\ \vdash \{[\![P]\!]_{\vec{x}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{x}}\}$, for $\{\vec{x}\} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P, Q)$.*

In our crust definition, we have focussed on full nodes $n{\mapsto}l,u,r,d$. This abstraction is shorthand for $n \mapsto l * n+1 \mapsto u * n+2 \mapsto r * n+3 \mapsto d$. We can define a smaller crust, by only referring to the specific pointers manipulated by the low-level program. In Example 8.2 for example, the specific pointers are the left pointer of n_2 and the right ponter of n_4.

Definition 8.6 (Small Crust)**.** The small crust at an interface (l,i,u,j,r), denoted $\Cap_S^{(l,i,u,j,r)}$, is defined by:

$$\begin{array}{c}\Cap_S^{(l,i,u,j,r)} \triangleq \big(r \mapsto j \vee r \doteq \text{nil}\big) * \\ \Big(l+2 \mapsto i \vee \big(l \doteq \text{nil} * \big(u+3 \mapsto i \vee u \doteq \text{nil}\big)\big)\Big)\end{array}$$

Analogous results to Theorem 8.3 and Theorem 8.4 hold. With this definition of small crust, it is natural to explore alternative definitions of subtrees and subcontexts:

$$\begin{array}{rcl}\text{alt-subtree } \mathsf{t}\ J & \triangleq & \exists i, j.\ \Cap_S^I * \text{subtree } \mathsf{t}\ I \\ \text{alt-subcontext } \mathsf{c}\ J\ J' & \triangleq & \exists i, j, i', j'. \\ & & (\Cap_S^{I'} \mathrel{-\!\!*} \text{subcontext } \mathsf{c}\ I\ I') * \Cap_S^I\end{array}$$

where $J = (l, u, r)$ and $I = (l, i, u, j, r)$, and similarly for J' and I'. With these definitions, the statements of Theorem 8.3 and Theorem 8.4 become simpler since the crust information can be hidden from view. However, the crust information just appears in a different place. Recall that 'subtree $\mathtt{t}_1|\mathtt{t}_2$ I' is simply defined using 'subtree $\mathtt{t}_1$ I_1' and 'subtree $\mathtt{t}_2$ I_2'. In comparison, 'alt-subtree $\mathtt{t}_1|\mathtt{t}_2$ J' is complicated, since it involves removing the right part of the crust (the r part) from 'alt-subtree $\mathtt{t}_1$ J_1' and the left and upper part of the crust (the l and u part) from 'alt-subtree $\mathtt{t}_2$ J_2'. The crust gets in the way of concatenating trees. It is therefore not possible to escape fully from the crust when reasoning about low-level tree update.

In summary, we believe that CL is ideal for specifying libraries for structured data update such as tree update, since it reasons about structured data at the right level of abstraction. Abstract predicates play an essential role in linking the high-level reasoning using CL with the low-level reasoning using SL. However, the abstract predicates do not abstract completely from the implementation details, since the crust information forms a necssary part of the reasoning. CL enables us to work with the 'fiction' that we are updating trees, regardless of the underlying implementation.

References

[1] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *POPL*, 2004.

[2] C. Calcagno, P. Gardner, and U. Zarfaty. Context logic & tree update. In *POPL*, 2005.

[3] P. Gardner, G. Smith, M. Wheelhouse, and U. Zarfaty. Local Hoare reasoning about DOM. In *PODS*, 2008. Preliminary version in PLanX 2008.

[4] S. Isthiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, pages 14–26. ACM Press, 2001.

[5] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL*, 2001.

[6] M. Parkinson and G. Bierman. Separation logic and abstraction. In *POPL*. ACM, 2005.

[7] J. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS*, 2002.

[8] G. Smith. *Local Hoare Reasoning about DOM.* PhD thesis, Imperial College London, 2009.

[9] H. Yang. Relational separation logic. *Theoretical Computer Science*, 375(1), 2007.

[10] H. Yang and P. O'Hearn. A semantic basis for local reasoning. In *FOSSACS*, 2002.

Appendix

Lemma A.1 (Weakest Preconditions). *The weakest preconditions of the Tree Update Language commands (Defn. 2.6) are expressible and derivable in the logic.*

Proof. For the atomic commands in Defn. 3.6, the weakest preconditions are:

$$
\begin{array}{rll}
\{P\} & \text{skip} & \{P\} \\
\{P[N/n]\} & n := N & \{P\} \\
\{P[X/x]\} & x := X & \{P\} \\
\\
\{(0 \triangleright P) \cdot n[\text{true}]\} & \text{delete } n_{\uparrow} & \{P\} \\
\{(n[0] \triangleright P) \cdot n[\text{true}]\} & \text{delete } n_{\downarrow} & \{P\} \\
\{\exists t.\, ((X|n[t]) \triangleright P) \cdot n[t]\} & \text{insert } X \text{ at } n_{\leftarrow} & \{P\} \\
\{\exists t.\, (n[X|t] \triangleright P) \cdot n[t]\} & \text{insert } X \text{ at } n_{\swarrow} & \{P\} \\
\{\exists t.\, \Diamond n[t] \wedge P[\langle n[t]\rangle/x]\} & x := \text{copy } n_{\uparrow} & \{P\} \\
\{\exists t.\, \Diamond n[t] \wedge P[\langle t\rangle/x]\} & x := \text{copy } n_{\downarrow} & \{P\} \\
\\
\{\exists m.\, \Diamond m[\text{true}|n[\text{true}]|\text{true}] \wedge (n = N) \wedge P[m/n']\} & n' := \text{lookup } n_{\uparrow} & \{P\} \\
\left\{\begin{array}{l}\exists m,n_2.\Diamond((n[\text{true}]|n_2[\text{true}]) \vee \\ (m[\text{true}|n[\text{true}]] \wedge n_2 = \text{nil}))\end{array} \wedge (n = N) \wedge P[n_2/n']\right\} & n' := \text{lookup } n_{\rightarrow} & \{P\} \\
\left\{\begin{array}{l}\exists n_2.\, \Diamond(n[\text{true}|n_2[\text{true}]] \vee \\ (n[0] \wedge n_2 = \text{nil}))\end{array} \wedge (n = N) \wedge P[n_2/n']\right\} & n' := \text{lookup } n_{\searrow} & \{P\}
\end{array}
$$

The derivations are all similar to Example **??**.

Lemma A.2 (Safety Precondition). *The safety precondition of the program in Example 2.7 is expressible and derivable in the logic.*

Proof. The result follows by inverse reasoning, starting from the postcondition true, using the weakest preconditions, and simplifying at each step.

$\{\text{true}\}$

  insert $\circ[x]$ at $n_{\searrow}$

$\{\exists t.\, (n[t|\circ[x]] \triangleright \text{true}) \cdot n[t]\} \Leftrightarrow \{\text{true}\}$

  delete $n'_{\uparrow}$;

$\{(0 \triangleright \text{true}) \cdot n'[\text{true}]\} \Leftrightarrow \{\Diamond n'[\text{true}]\}$

  $x :=$ copy $n'_{\uparrow}$;

$\{\exists t.\, \Diamond n'[t] \wedge \Diamond n'[\text{true}]\} \Leftrightarrow \{\Diamond n'[\text{true}]\}$

  $n' :=$ lookup $n'_{\rightarrow}$;

$\{\exists m, n_2.\, \Diamond((n'[\text{true}]|n_2[\text{true}]) \vee (m[\text{true}|n[\text{true}]] \wedge n_2 = \text{nil})) \wedge \Diamond n_2[\text{true}]\} \Leftrightarrow \{\Diamond(n'[\text{true}]|\circ[\text{true}])\}$

  $n' :=$ lookup $n'_{\swarrow}$;

$\{\exists n_2.\, \Diamond(n'[n_2[\text{true}]|\text{true}] \vee (n'[0] \wedge n_2 = \text{nil})) \wedge \Diamond(n_2[\text{true}]|\circ[\text{true}])\} \Leftrightarrow \{\Diamond n'[\circ[\text{true}]|\circ[\text{true}]|\text{true}]\}$

  $n' :=$ lookup $n_{\swarrow}$;

$\{\exists n_2.\, \Diamond(n[n_2[\text{true}]|\text{true}] \vee (n[0] \wedge n_2 = \text{nil})) \wedge \Diamond n_2[\circ[\text{true}]|\circ[\text{true}]|\text{true}]\} \Leftrightarrow \{\Diamond n[\circ[\circ[\text{true}] \mid \circ[\text{true}] \mid \text{true}] \mid \text{tru}$

Lemma 5.4 (Application Lemma). *Context application corresponds to star-separation across interfaces:*

$$\text{subtree } \text{ap}(\mathsf{c},\mathsf{t})\ I \Leftrightarrow \exists I'.\,\text{subcontext } \mathsf{c}\ I\ I' * \text{subtree } \mathsf{t}\ I'$$

Proof. Result follows by straightforward induction on c.

$$\begin{aligned}
&\text{subtree } \text{ap}(-,\mathsf{t})\ (l,i,u,j,r) \Leftrightarrow \text{subtree } \mathsf{t}\ (l,i,u,j,r)\\
&\Leftrightarrow \exists l',i',u',j',r'.\,(l \doteq l') * (i \doteq i') * (u \doteq u') * (j \doteq j') * (r \doteq r') * \text{subtree } \mathsf{t}\ (l',i',u',j',r')\\
&\Leftrightarrow \exists l',i',u',j',r'.\,\text{subcontext } -\ (l,i,u,j,r)\ (l',i',u',j',r') * \text{subtree } \mathsf{t}\ (l',i',u',j',r')
\end{aligned}$$

$$\begin{aligned}
&\text{subtree } \text{ap}(\mathsf{n}[\mathsf{c}],\mathsf{t})\ (l,i,u,j,r) \Leftrightarrow \text{subtree } \mathsf{n}[\text{ap}(\mathsf{c},\mathsf{t})]\ (l,i,u,j,r)\\
&\Leftrightarrow \exists d_i, d_j.\,(i \doteq \mathsf{n}) * (j \doteq \mathsf{n}) * \mathsf{n} \mapsto l, u, r, d_i * \text{subtree } \text{ap}(\mathsf{c},\mathsf{t})\ (\text{nil},d_i,\mathsf{n},d_j,\text{nil})\\
&\Leftrightarrow \exists d_i, d_j.\,(i \doteq \mathsf{n}) * (j \doteq \mathsf{n}) * \mathsf{n} \mapsto l, u, r, d_i * \exists l',i',u',j',r'.\\
&\quad \text{subcontext } \mathsf{c}\ (\text{nil},d_i,\mathsf{n},d_j,\text{nil})\ (l',i',u',j',r') * \text{subtree } \mathsf{t}\ (l',i',u',j',r')\\
&\Leftrightarrow \exists l',i',u',j',r'.\,\text{subcontext } \mathsf{n}[\mathsf{c}]\ (l,i,u,j,r)\ (l',i',u',j',r') * \text{subtree } \mathsf{t}\ (l',i',u',j',r')
\end{aligned}$$

$$\begin{aligned}
&\text{subtree } \text{ap}(\mathsf{t}|\mathsf{c},\mathsf{t}')\ (l,i,u,j,r) \Leftrightarrow \text{subtree } \mathsf{t}|\,\text{ap}(\mathsf{c},\mathsf{t}')\ (l,i,u,j,r)\\
&\Leftrightarrow \exists k_l, k_r.\,\text{subtree } \mathsf{t}\ (l,i,u,k_l,k_r) * \text{subtree } \text{ap}(\mathsf{c},\mathsf{t}')\ (k_l,k_r,u,j,r)\\
&\Leftrightarrow \exists k_l, k_r.\,\text{subtree } \mathsf{t}\ (l,i,u,k_l,k_r) * \exists l',i',u',j',r'.\\
&\quad \text{subcontext } \mathsf{c}\ (k_l,k_r,u,j,r)\ (l',i',u',j',r') * \text{subtree } \mathsf{t}'\ (l',i',u',j',r')\\
&\Leftrightarrow \exists l',i',u',j',r'.\,\text{subcontext } \mathsf{t}|\mathsf{c}\ (l,i,u,j,r)\ (l',i',u',j',r') * \text{subtree } \mathsf{t}'\ (l',i',u',j',r')
\end{aligned}$$

$$\begin{aligned}
&\text{subtree } \text{ap}(\mathsf{c}|\mathsf{t},\mathsf{t}')\ (l,i,u,j,r) \Leftrightarrow \text{subtree } \text{ap}(\mathsf{c},\mathsf{t}')|\mathsf{t}\ (l,i,u,j,r)\\
&\Leftrightarrow \exists k_l, k_r.\,\text{subtree } \text{ap}(\mathsf{c},\mathsf{t}')\ (l,i,u,k_l,k_r) * \text{subtree } \mathsf{t}\ (k_l,k_r,u,j,r)\\
&\Leftrightarrow \exists k_l, k_r.\,\exists l',i',u',j',r'.\,\text{subcontext } \mathsf{c}\ (l,i,u,k_l,k_r)\ (l',i',u',j',r') * \text{subtree } \mathsf{t}'\ (l',i',u',j',r')\\
&\quad * \text{subtree } \mathsf{t}\ (k_l,k_r,u,j,r)\\
&\Leftrightarrow \exists l',i',u',j',r'.\,\text{subcontext } \mathsf{c}|\mathsf{t}\ (l,i,u,j,r)\ (l',i',u',j',r') * \text{subtree } \mathsf{t}'\ (l',i',u',j',r')
\end{aligned}$$

Lemma A.3 (Exactness). *Subtree and subonctext describe at most one tree or context for any given interfaces.*

$$\forall t_1, t_2, I.\, \text{subtree } t_1\ I \Rightarrow\ (\text{subtree } t_2\ I \Leftrightarrow t_1 \equiv t_2)$$
$$\forall tc_1, c_2, I, I'.\, \text{subcontext } c_1\ I\ I' \Rightarrow\ (\text{subcontext } c_2\ I\ I' \Leftrightarrow c_1 \equiv c_2)$$

Proof. We show the right-to-left implication first: namely, that subtree and subcontext respect the tree and context equivalence relations. This follows by simple induction. The case for trees is:

$$\text{subtree } 0|\mathsf{t}\ (l,i,u,j,r) \Leftrightarrow \exists k_l, k_r.\, (k_l \doteq j) * (k_r \doteq r) * \text{subtree } \mathsf{t}\ (k_l,k_r,u,j,r) \Leftrightarrow \text{subtree } \mathsf{t}\ (l,i,u,j,r)$$
$$\text{subtree } \mathsf{t}|0\ (l,i,u,j,r) \Leftrightarrow \exists k_l, k_r.\, \text{subtree } \mathsf{t}\ (l,i,u,k_l,k_r) * (k_l \doteq j) * (k_r \doteq r) \Leftrightarrow \text{subtree } \mathsf{t}\ (l,i,u,j,r)$$

$$\text{subtree } \mathsf{t}_1|(\mathsf{t}_2|\mathsf{t}_3)\ (l,i,u,j,r) \Leftrightarrow \exists k_l^1, k_r^1, k_l^2, k_r^2.\, \begin{array}{l}\text{subtree } \mathsf{t}_1\ (l,i,u,k_l^1,k_r^1) *\\ \text{subtree } \mathsf{t}_2\ (k_l^1,k_r^1,u,k_l^2,k_r^2) *\\ \text{subtree } \mathsf{t}_3\ (k_l^1,k_r^1,u,j,r)\end{array} \Leftrightarrow \text{subtree } (\mathsf{t}_1|\mathsf{t}_2)|\mathsf{t}_3\ (l,i,u,j,r)$$

The case for contexts is similar.

Next we show the left-to-right implication: that for given interfaces, subtree and subcontext describe at most one tree. For trees, we have three cases: $t_1 \equiv t_2 \equiv 0$; $t_1 \equiv \mathsf{n}_1[t_1^1]|t_1^2 \wedge t_2 \equiv 0$; and $t_1 \equiv \mathsf{n}_1[t_1^1]|t_1^2 \wedge t_2 \equiv \mathsf{n}_2[t_2^1]|t_2^2$. The result then follows by induction:

$$\text{subtree } 0\ (l,i,u,j,r) \Rightarrow (\text{subtree } 0\ (l,i,u,j,r) \Leftarrow \text{true})$$
$$\text{subtree } \mathsf{n}_1[t_1^1]|t_1^2\ (l,i,u,j,r) \Rightarrow \neg\text{emp} \Rightarrow (\text{subtree } 0\ (l,i,u,j,r) \Leftarrow \text{false})$$
$$\begin{array}{rl}\text{subtree } \mathsf{n}_1[t_1^1]|t_1^2\ (l,i,u,j,r) & \Rightarrow \exists d_i, d_j, r_n.\, \mathsf{n}_1 \mapsto l, u, r, d_i * \text{subtree } t_1^1\ (\text{nil},d_i,\mathsf{n},d_j,\text{nil}) * \text{subtree } t_1^2\ (\mathsf{n},r_n,u,j,r)\\ & \Rightarrow (\text{subtree } \mathsf{n}_2[t_2^1]|t_2^2\ (l,i,u,j,r) \Leftarrow \mathsf{n}_1 = \mathsf{n}_2 \wedge t_1^1 \equiv t_2^1 \wedge t_1^2 \equiv t_2^2)\end{array}$$

Hence subtree $t_1\ I \Rightarrow (\text{subtree } t_2 \Leftarrow t_1 \equiv t_2)$. For contexts the proof is identical, except with more cases. For all c we have $c \equiv -$, $c \equiv \mathsf{n}_1[c']|t$ or $c \equiv \mathsf{n}_1[t]|c'$. The six combinations follow by the same argument as above.

Lemma A.2 (Name matching). *The locations of a tree or context correspond to those of their heap representations.*

$$\forall \mathsf{t}, \mathsf{s}, \mathsf{h}, I, \mathsf{n}, \mathsf{j} \in [0,3].\, \mathsf{s}, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \Rightarrow\ (\mathsf{n} \in \text{locs}(\mathsf{t}) \Leftrightarrow \mathsf{n} + \mathsf{j} \in \text{dom}(\mathsf{h}))$$
$$\forall \mathsf{c}, \mathsf{s}, \mathsf{h}, I, I', \mathsf{n}, \mathsf{j} \in [0,3].\, \mathsf{s}, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \Rightarrow\ (\mathsf{n} \in \text{locs}(\mathsf{c}) \Leftrightarrow \mathsf{n} + \mathsf{j} \in \text{dom}(\mathsf{h}))$$

Proof. Straightforward induction. It is easy to see from the inductive predicate definitions that every n in c or t contributes one node of the form $\mathsf{n} \mapsto l, u, r, d$, whose heap domain is therefore $\{\mathsf{n}, \mathsf{n}{+}1, \mathsf{n}{+}2, \mathsf{n}{+}3\}$. Furthermore, no other parts of the definitions contribute any additional heap locations.

Theorem 6.2 (Language translation). *The translation is a well-defined, fault-avoiding translation with respect to the high-level and low-level operational semantics and the low-level tree representation. In other words:*

(a) $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \mathsf{s}'_t, \mathsf{t}' \Rightarrow \forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!].\, \exists \mathsf{s}'_h, \mathsf{h}' \in [\![\mathsf{s}'_t, \mathsf{t}']\!].\, [\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$
(b) $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \text{fault} \Rightarrow \exists \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!].\, [\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \text{fault}$

(c) $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}' \wedge \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!] \wedge \mathbb{C}, \mathsf{s}_t, \mathsf{t} \not\rightsquigarrow \text{fault} \Rightarrow \exists \mathsf{s}'_t, \mathsf{t}'.\, \mathsf{s}'_h, \mathsf{h}' \in [\![\mathsf{s}'_t, \mathsf{t}']\!] \wedge \mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \mathsf{s}'_t, \mathsf{t}'$
(d) $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \text{fault} \wedge \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!] \Rightarrow \mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \text{fault}$

Sketch proof. Follows directly from the operational semantics of two update languages and Lemma 5.4. The derivations for delete and copy are given here. The other derivations are similar. All the derivations follow the same principles as the specification proofs in Lemmas **??**, **??** and **??** and Theorem 8.3, proved further on.

To prove the results for delete and copy, we must first describe the behaviour of the utility functions dispose-forest and copy-forest.

dispose-forest For dispose-forest n, we see that:

$$\frac{\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \quad \mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,n,u,j,\text{nil})}{\text{dispose-forest } n, \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}_h^+, \mathsf{h}'} \qquad \frac{\mathsf{s}_h(n) \neq \text{nil} \quad \mathsf{s}_h(n) \notin \text{dom}(\mathsf{h})}{\text{dispose-forest } n, \mathsf{s}_h, \mathsf{h} \rightsquigarrow \text{fault}}$$

where s_h^+ is an extension of s_h containing the (fresh) auxiliary variables. The results follow by by induction:

$\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \wedge \mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,n,u,j,\text{nil})$

if $n = \text{nil}$ then skip

$\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \wedge \mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } 0\ (l,\text{nil},u,j,\text{nil})$

$\Rightarrow \mathsf{h} = \mathsf{h}'$

else

$\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \wedge \mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } n[\mathsf{t}_1] | \mathsf{t}_2\ (l,n,u,j,\text{nil})$

$\Rightarrow \mathsf{h} = \mathsf{h}' * \mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d * \mathsf{h}_1 * \mathsf{h}_2 \wedge \begin{array}{l}\mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } t_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h, \mathsf{h}_2 \vDash_{\mathcal{H}} \text{subtree } t_2\ (n,\mathsf{i}_\mathsf{r},u,j,\text{nil})\end{array}$

r:=[n.right] ; dispose-forest r ;

$\begin{array}{l}\mathsf{s}_h \rightsquigarrow \mathsf{s}_h^+[r \leftarrow \mathsf{i}_\mathsf{r}] \wedge \\ \mathsf{h} \rightsquigarrow \mathsf{h}' * \mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d * \mathsf{h}_1 \wedge\end{array} \mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } t_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil})$

d:=[n.down] ; dispose-forest d ;

$\mathsf{s}_h \rightsquigarrow \mathsf{s}_h^+\left[\begin{smallmatrix} r \leftarrow \mathsf{i}_\mathsf{r} \\ d \leftarrow \mathsf{i}_\mathsf{d}\end{smallmatrix}\right] \wedge \mathsf{h} \rightsquigarrow \mathsf{h}' * \mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d$

dispose-node n

$\mathsf{s}_h \rightsquigarrow \mathsf{s}_h^+ \wedge \mathsf{h} \rightsquigarrow \mathsf{h}'$

$\mathsf{s}_h(n) \neq \text{nil} \wedge \mathsf{s}_h(n) \notin \text{dom}(\mathsf{h})$

if $n = \text{nil}$ then skip else

r:=[n.right] ; . . .

$\rightsquigarrow$ fault

copy-forest For $(i', j') :=$ copy-forest n (l', u', r'), we have:

$$\frac{\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \quad \begin{array}{c}\mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,n,u,j,\text{nil}) \\ \mathsf{s}_h[i' \leftarrow \mathsf{i}', j' \leftarrow \mathsf{j}'], \mathsf{h}'_n \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ (l',i',u',j',r')\end{array} \quad \mathsf{h}'_n \# \mathsf{h} \quad \langle t \rangle \equiv \langle t' \rangle}{(i', j') := \text{copy-forest } n\ (l', u', r'), \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}_h^+[i' \leftarrow \mathsf{i}', j' \leftarrow \mathsf{j}'], \mathsf{h} * \mathsf{h}'_n}$$

$$\frac{\mathsf{s}_h(n) \neq \text{nil} \quad \mathsf{s}_h(n) \notin \text{dom}(\mathsf{h})}{(i', j') := \text{copy-forest } n\ (l', u', r'), \mathsf{s}_h, \mathsf{h} \rightsquigarrow \text{fault}}$$

which is shown by:

$$\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \wedge \begin{array}{c} \mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,n,u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_n \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ (l',i',u',j',r') \wedge \end{array} \mathsf{h}'_n \# \mathsf{h} \wedge \langle \mathsf{t} \rangle \equiv \langle \mathsf{t}' \rangle \qquad \mathsf{s}_h(n) \neq \text{nil} \wedge \mathsf{s}_h(n) \notin \text{dom}(\mathsf{h})$$

if $n = \text{nil}$ then $\quad$ if $n = \text{nil}$ then $\ldots$ else

$$\mathsf{h} = \mathsf{h}' \wedge \begin{array}{c} \mathsf{s}_h, \text{emp} \vDash_{\mathcal{H}} \text{subtree } 0\ (l,\text{nil},u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_n \vDash_{\mathcal{H}} \text{subtree } 0\ (l',i',u',j',r') \wedge \end{array} \mathsf{h}'_n \equiv \text{emp} \wedge \begin{array}{l} i' = \mathsf{s}_h(r') \wedge \\ j' = \mathsf{s}_h(l') \end{array} \qquad \begin{array}{l} \ldots ; r{:=}[n.\text{right}] ; \ldots \\ \rightsquigarrow \text{fault} \end{array}$$

$i' := r'$; $j' := l'$

$$\mathsf{h} = \mathsf{h} * \mathsf{h}'_n \wedge \mathsf{s}_h \rightsquigarrow \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}],$$

else

$$\mathsf{h} = \mathsf{h}' * \mathsf{h}_n \wedge \begin{array}{c} \mathsf{s}_h, \mathsf{h}_n \vDash_{\mathcal{H}} \text{subtree } n[\mathsf{t}_1] | \mathsf{t}_2\ (l,n,u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_n \vDash_{\mathcal{H}} \text{subtree } i'[\mathsf{t}'_1] | \mathsf{t}'_2\ (l',i',u',j',r') \wedge \end{array} \mathsf{h}'_n \# \mathsf{h} \wedge \langle \mathsf{t}_1 \rangle \equiv \langle \mathsf{t}'_1 \rangle \wedge \langle \mathsf{t}_2 \rangle \equiv \langle \mathsf{t}'_2 \rangle$$

$$\Rightarrow \begin{array}{l} \mathsf{h} = \mathsf{h}' * (\mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d) * \mathsf{h}_1 * \mathsf{h}_2 \wedge \\ \mathsf{h}'_n = (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \mathsf{i}'_r, \mathsf{i}'_d) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \end{array} \begin{array}{r} \mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h, \mathsf{h}_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_2\ (n,\mathsf{i}_\mathsf{r},u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_1\ (\text{nil},\mathsf{i}'_\mathsf{d},n',\mathsf{j}'_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_2\ (l',\mathsf{i}'_r,u',j',\text{nil}) \wedge \end{array} \begin{array}{c} \mathsf{h}'_n \# \mathsf{h} \wedge \\ \langle \mathsf{t}_1 \rangle \equiv \langle \mathsf{t}'_1 \rangle \wedge \\ \langle \mathsf{t}_2 \rangle \equiv \langle \mathsf{t}'_2 \rangle \end{array}$$

$i' := \text{new-node}()$; $[i'.\text{left}] := l'$; $[i'.\text{up}] := u'$;

$$\begin{array}{l} \mathsf{s}_h \rightsquigarrow \mathsf{s}_h[i' \leftarrow \mathsf{i}'] \quad \text{(by non-determinism of new)} \\ \mathsf{h} \rightsquigarrow \mathsf{h}' * (\mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d, \text{nil}) * \mathsf{h}_1 * \mathsf{h}_2 \\ \quad * (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \text{nil}, \text{nil}) \wedge \\ \mathsf{h}'_n = (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \mathsf{i}'_r, \mathsf{i}'_d) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \end{array} \begin{array}{r} \mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h, \mathsf{h}_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_2\ (n,\mathsf{i}_\mathsf{r},u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_1\ (\text{nil},\mathsf{i}'_\mathsf{d},n',\mathsf{j}'_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_2\ (l',\mathsf{i}'_r,u',j',\text{nil}) \wedge \end{array} \begin{array}{c} \mathsf{h}'_n \# \mathsf{h} \wedge \\ \langle \mathsf{t}_1 \rangle \equiv \langle \mathsf{t}'_1 \rangle \wedge \\ \langle \mathsf{t}_2 \rangle \equiv \langle \mathsf{t}'_2 \rangle \end{array}$$

$r{:=}[n.\text{right}]$; $(r'_i,j'){:=}\text{copy-forest } r\ (i',u',r')$;

$$\begin{array}{l} \mathsf{s}_h \rightsquigarrow \mathsf{s}^+_h[^{i' \leftarrow \mathsf{i}'}_{r \leftarrow \mathsf{i}_r}\ ^{r'_i \leftarrow \mathsf{i}'_r}_{j' \leftarrow \mathsf{j}'}] \quad \text{(inductive step)} \\ \mathsf{h} \rightsquigarrow \mathsf{h}' * (\mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d, \text{nil}) * \mathsf{h}_1 * \mathsf{h}_2 \\ \quad * (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \text{nil}, \text{nil}) * \mathsf{h}'_2 \wedge \\ \mathsf{h}'_n = (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \mathsf{i}'_r, \mathsf{i}'_d) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \end{array} \begin{array}{r} \mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h, \mathsf{h}_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_2\ (n,\mathsf{i}_\mathsf{r},u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_1\ (\text{nil},\mathsf{i}'_\mathsf{d},n',\mathsf{j}'_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_2\ (l',\mathsf{i}'_r,u',j',\text{nil}) \wedge \end{array} \begin{array}{c} \mathsf{h}'_n \# \mathsf{h} \wedge \\ \langle \mathsf{t}_1 \rangle \equiv \langle \mathsf{t}'_1 \rangle \wedge \\ \langle \mathsf{t}_2 \rangle \equiv \langle \mathsf{t}'_2 \rangle \end{array}$$

$d{:=}[n.\text{down}]$; $(d'_i,d'_j){:=}\text{copy-forest } d\ (\text{nil},i',\text{nil})$;

$$\begin{array}{l} \mathsf{s}_h \rightsquigarrow \mathsf{s}^+_h[^{i' \leftarrow \mathsf{i}'}_{r \leftarrow \mathsf{i}_r}\ ^{r'_i \leftarrow \mathsf{i}'_r}_{j' \leftarrow \mathsf{j}'}\ ^{d \leftarrow \mathsf{i}_d}_{d'_i \leftarrow \mathsf{i}'_d}\ ^{d'_j \leftarrow \mathsf{j}'_d}] \quad \text{(inductive step)} \\ \mathsf{h} \rightsquigarrow \mathsf{h}' * (\mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d, \text{nil}) * \mathsf{h}_1 * \mathsf{h}_2 \\ \quad * (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \text{nil}, \text{nil}) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \\ \mathsf{h}'_n = (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \mathsf{i}'_r, \mathsf{i}'_d) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \end{array} \begin{array}{r} \mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h, \mathsf{h}_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_2\ (n,\mathsf{i}_\mathsf{r},u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_1\ (\text{nil},\mathsf{i}'_\mathsf{d},n',\mathsf{j}'_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_2\ (l',\mathsf{i}'_r,u',j',\text{nil}) \wedge \end{array} \begin{array}{c} \mathsf{h}'_n \# \mathsf{h} \wedge \\ \langle \mathsf{t}_1 \rangle \equiv \langle \mathsf{t}'_1 \rangle \wedge \\ \langle \mathsf{t}_2 \rangle \equiv \langle \mathsf{t}'_2 \rangle \end{array}$$

$[i'.\text{right}] := r'_i$; $[i'.\text{down}] := d'_i$

$$\begin{array}{l} \mathsf{s}_h \rightsquigarrow \mathsf{s}^+_h[^{i' \leftarrow \mathsf{i}'}_{r \leftarrow \mathsf{i}_r}\ ^{r'_i \leftarrow \mathsf{i}'_r}_{j' \leftarrow \mathsf{j}'}\ ^{d \leftarrow \mathsf{i}_d}_{d'_i \leftarrow \mathsf{i}'_d}\ ^{d'_j \leftarrow \mathsf{j}'_d}] \\ \mathsf{h} \rightsquigarrow \mathsf{h}' * (\mathsf{s}_h(n) \mapsto \mathsf{s}_h(l), \mathsf{s}_h(u), \mathsf{i}_r, \mathsf{i}_d, \text{nil}) * \mathsf{h}_1 * \mathsf{h}_2 \\ \quad * (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \mathsf{i}_r, \mathsf{i}_d) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \\ \mathsf{h}'_n = (\mathsf{i}' \mapsto \mathsf{s}_h(l'), \mathsf{s}_h(u'), \mathsf{i}'_r, \mathsf{i}'_d) * \mathsf{h}'_1 * \mathsf{h}'_2 \wedge \end{array} \begin{array}{r} \mathsf{s}_h, \mathsf{h}_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_1\ (\text{nil},\mathsf{i}_\mathsf{d},n,\mathsf{j}_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h, \mathsf{h}_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_2\ (n,\mathsf{i}_\mathsf{r},u,j,\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_1 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_1\ (\text{nil},\mathsf{i}'_\mathsf{d},n',\mathsf{j}'_\mathsf{d},\text{nil}) \wedge \\ \mathsf{s}_h[^{i' \leftarrow \mathsf{i}'}_{j' \leftarrow \mathsf{j}'}], \mathsf{h}'_2 \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'_2\ (l',\mathsf{i}'_r,u',j',\text{nil}) \wedge \end{array} \begin{array}{c} \mathsf{h}'_n \# \mathsf{h} \wedge \\ \langle \mathsf{t}_1 \rangle \equiv \langle \mathsf{t}'_1 \rangle \wedge \\ \langle \mathsf{t}_2 \rangle \equiv \langle \mathsf{t}'_2 \rangle \end{array}$$

$$\Rightarrow \mathsf{h} \rightsquigarrow \mathsf{h} * \mathsf{h}_n \wedge \mathsf{s}_h \rightsquigarrow \mathsf{s}^+_h[^{i \leftarrow \mathsf{i}}_{j \leftarrow \mathsf{j}}]$$

Delete We show that the theorem holds for delete $n_\uparrow$; the case for delete $n_\downarrow$ is similar, but simpler. From Figure 1, we see that:

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \text{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{\text{delete } n_\uparrow, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}, \text{ap}(\mathsf{c}, 0)} \qquad \frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \not\equiv \text{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{\text{delete } n_\uparrow, \mathsf{s}, \mathsf{t} \rightsquigarrow \text{fault}}$$

For (a), we therefore wish to show that

$$\forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])]\!].\ \exists \mathsf{s}'_h, \mathsf{h}' \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, 0)]\!].\ [\![\text{delete } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$$

where $\mathsf{s}_t(n) = \mathsf{s}_h(n) = \mathsf{n}$. Similarly, for (c), we wish to show that

$$\forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])]\!].\ \forall \mathsf{s}'_h, \mathsf{h}'.\ [\![\text{delete } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}' \Rightarrow \mathsf{s}'_h, \mathsf{h}' \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, 0)]\!]$$

Consider then $\mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])]\!]$. By Defn. **??**, we have $\mathsf{h} = \mathsf{h}_t * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x$, for some h_t and h_xs satisfying $\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t$ (for some n_t) and $\mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{t}_x\ \&x$ for $\langle \mathsf{t}_x \rangle \equiv \mathsf{s}_t(x)$. Using Lemma 5.4, we have:

$$
\begin{array}{l}
\quad \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t \\
\Leftrightarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists l, i, u, j, r.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,i,u,j,r) * \text{subtree } n[\mathsf{t}']\ (l,i,u,j,r) \\
\Leftrightarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists l, i, u, j, r.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,i,u,j,r) * \exists d_i, d_j.\ (i \doteq n) * (j \doteq n) * n \mapsto l, u, r, d_i * \text{subtree } \mathsf{t}'\ (\text{nil},d_i,n,d_j,\text{nil}) \\
\quad d := [n.\text{down}]\ ;\ r := [n.\text{right}]\ ;\ u := [n.\text{up}]\ ;\ l := [n.\text{left}]\ ; \\
\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists i, j.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,i,u,j,r) * \exists d_j.\ (i \doteq n) * (j \doteq n) * n \mapsto l, u, r, d * \text{subtree } \mathsf{t}'\ (\text{nil},d,n,d_j,\text{nil}) \\
\quad \text{dispose-forest } d\ ;\ \text{dispose-node } n\ ; \\
\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists i, j.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,i,u,j,r) \\
\quad \text{if } r \neq \text{nil then } [r.\text{left}] := l\ ; \\
\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists i, j.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,i,u,l,r) \\
\quad \text{if } l \neq \text{nil then } [l.\text{right}] := r \\
\qquad \rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists i, j.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,r,u,l,r) \\
\qquad \Rightarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, 0)\ \mathsf{n}_t \\
\quad \text{if } u \neq \text{nil then } [u.\text{down}] := r \\
\qquad \rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists i, j.\ \text{context } \mathsf{c}\ \mathsf{n}_t\ (l,r,u,l,r) \\
\qquad \Rightarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, 0)\ \mathsf{n}_t
\end{array}
$$

from which (a) and (c) follow immediately.

Meanwhile, for (b) and (d), we wish to show that

$$\mathsf{t} \neq \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}']) \text{ for } \mathsf{n} = \mathsf{s}_t(n) \Leftrightarrow \exists \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!].\ [\![\text{delete } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \text{fault}$$

For the left-to-right implication, we assume $\mathsf{t} \neq \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])$ and let $\mathsf{h} = \mathsf{h}_t * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x$, for some h_t and h_xs satisfying $\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathsf{t}\ \mathsf{n}_t$ (for some n_t) and $\mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{t}_x\ \&x$, where $\langle \mathsf{t}_x \rangle \equiv \mathsf{s}_t(x)$ and $\mathsf{n} \notin \mathrm{dom}(\mathsf{h}_x)$ (this is always possible, since t_x is only defined up to renaming). Then $\mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!]$ and $\mathsf{n} \notin \mathrm{dom}(\mathsf{h})$. Hence, $[\![\text{delete } n_\uparrow]\!]$ faults as soon as it executes $d := [n.\text{down}]$. For the right-to-left implication, we assume $\mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])$ and show that $\forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!].\ [\![\text{delete } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow \text{fault}$. This follows directly from the proof above for (a) and (c).

Copy We show that the theorem holds for $x := \text{copy } n_\uparrow$; the case for $x := \text{copy } n_\downarrow$ is similar, but simpler. From Figure 1, we see that:

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{x := \text{copy } n_\uparrow, \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|x \leftarrow \langle \mathsf{n}[\mathsf{t}'] \rangle], \mathsf{t}} \quad \frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \not\equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{x := \text{copy } n_\uparrow, \mathsf{s}, \mathsf{t} \rightsquigarrow \text{fault}}$$

For (a), we therefore wish to show that

$$\forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])]\!].\ \exists \mathsf{s}'_h, \mathsf{h}' \in [\![\mathsf{s}_t[x \leftarrow \langle \mathsf{n}[\mathsf{t}'] \rangle], \mathsf{t}]\!].\ [\![x := \text{copy } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$$

where $\mathsf{s}_t(n) = \mathsf{s}_h(n) = \mathsf{n}$. Similarly, for (c), we wish to show that

$$\forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])]\!]. \forall \mathsf{s}'_h, \mathsf{h}'. [\![x := \text{copy } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}' \Rightarrow \mathsf{s}'_h, \mathsf{h}' \in [\![\mathsf{s}_t[x \leftarrow \langle \mathsf{n}[\mathsf{t}'] \rangle], \mathsf{t}]\!]$$

Consider then $\mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])]\!]$. By Defn. ??, we have $\mathsf{h} = \mathsf{h}_t * \prod_{y \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_y$, for some h_t and h_ys satisfying $\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t$ (for some n_t) and $\mathsf{s}_h, \mathsf{h}_y \vDash_{\mathcal{H}} \text{tree } \mathsf{t}_y\ \&y$ for $\langle \mathsf{t}_y \rangle \equiv \mathsf{s}_t(y)$. Using Lemma 5.4, we have (using h_t and h_x as nameholders for the relevant part of the updated heap):

$\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{t}_x\ \&x$

dispose-forest $\&x$;

$\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{emp}$

$\&x :=$ new-node() ;

$\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \&x \mapsto \mathrm{nil}, \mathrm{nil}, \mathrm{nil}, \mathrm{nil}$

$d := [n.\text{down}]$;

$\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \begin{matrix}\text{context } \mathsf{c}\ \mathsf{n}_t\ (\mathsf{l},n,\mathsf{u},\mathsf{j},\mathsf{r}) * n \mapsto \mathsf{l}, \mathsf{u}, \mathsf{r}, d\ * \\ \text{subtree } \mathsf{t}'\ (\mathrm{nil},d,n,\mathsf{d_j},\mathrm{nil})\end{matrix} \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \&x \mapsto \mathrm{nil}, \mathrm{nil}, \mathrm{nil}, \mathrm{nil}$

(i, j):=copy-forest d (nil, $\&x$, nil) ;

$\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \begin{matrix}\text{context } \mathsf{c}\ \mathsf{n}_t\ (\mathsf{l},n,\mathsf{u},\mathsf{j},\mathsf{r}) * n \mapsto \mathsf{l}, \mathsf{u}, \mathsf{r}, d\ * \\ \text{subtree } \mathsf{t}'\ (\mathrm{nil},d,n,\mathsf{d_j},\mathrm{nil})\end{matrix} \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \&x \mapsto \mathrm{nil}, \mathrm{nil}, \mathrm{nil}, \mathrm{nil} * \text{subtree } \mathsf{t}''\ (\mathrm{nil},i,\&x,j,\mathrm{nil}) \wedge \langle \mathsf{t}' \rangle \equiv \langle \mathsf{t}'' \rangle$

$[\&x.\text{down}] := i$;

$\rightsquigarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \begin{matrix}\text{context } \mathsf{c}\ \mathsf{n}_t\ (\mathsf{l},n,\mathsf{u},\mathsf{j},\mathsf{r}) * n \mapsto \mathsf{l}, \mathsf{u}, \mathsf{r}, d\ * \\ \text{subtree } \mathsf{t}'\ (\mathrm{nil},d,n,\mathsf{d_j},\mathrm{nil})\end{matrix} \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \&x \mapsto \mathrm{nil}, \mathrm{nil}, \mathrm{nil}, i * \text{subtree } \mathsf{t}''\ (\mathrm{nil},i,\&x,j,\mathrm{nil}) \wedge \langle \mathsf{t}' \rangle \equiv \langle \mathsf{t}'' \rangle$

$\Rightarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{n}_t[\mathsf{t}'']\ \&x \wedge \langle \mathsf{t}' \rangle \equiv \langle \mathsf{t}'' \rangle$

$\Rightarrow \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathrm{ap}(\mathsf{c}, n[\mathsf{t}'])\ \mathsf{n}_t \wedge \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{t}'''\ \&x \wedge \langle \mathsf{t}''' \rangle \equiv \langle \mathsf{n}[\mathsf{t}'] \rangle$

from which (a) and (c) follow immediately.

Meanwhile, for (b) and (d), we wish to show that

$$\mathsf{t} \neq \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}']) \text{ for } \mathsf{n} = \mathsf{s}_t(n) \Leftrightarrow \exists \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!]. [\![x := \text{copy } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \text{fault}$$

For the left-to-right implication, we assume $\mathsf{t} \neq \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])$ and let $\mathsf{h} = \mathsf{h}_t * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x$, for some h_t and h_xs satisfying $\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathsf{t}\ \mathsf{n}_t$ (for some n_t) and $\mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{t}_x\ \&x$, where $\langle \mathsf{t}_x \rangle \equiv \mathsf{s}_t(x)$ and $\mathsf{n} \notin \mathrm{dom}(\mathsf{h}_x)$ (this is always possible, since t_x is only defined up to renaming). Then $\mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!]$ and $\mathsf{n} \notin \mathrm{dom}(\mathsf{h})$. Hence, $[\![x := \text{copy } n_\uparrow]\!]$ faults as soon as it executes $d := [n.\text{down}]$. For the right-to-left implication, we assume $\mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])$ and show that $\forall \mathsf{s}_h, \mathsf{h} \in [\![\mathsf{s}_t, \mathsf{t}]\!]. [\![x := \text{copy } n_\uparrow]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow$ fault. This follows directly from the proof above for (a) and (c).

Lemma 7.2 (Type soundness). *For all P, $\vec{x} \supseteq \mathrm{vars}_{\mathcal{T}_\circ}(P)$:*

$$\forall\, \mathsf{e}, \mathsf{s}_h, \mathsf{h}.\ \mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{x}} \Rightarrow \exists \mathsf{s}_t, \mathsf{t}.\ (\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!] \wedge \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t) = \{\vec{x}\}$$

Proof sketch. By Defn. 7.1, we see that $\mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{x}}$ implies $\mathsf{h} = \mathsf{h}_t * \prod_{x \in \vec{x}} \mathsf{h}_x$, for some h_t and h_xs satisfying $\mathsf{e}', \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \exists i, j.\, [\![P]\!]^{(\mathrm{nil}, i, \mathrm{nil}, j, \mathrm{nil})}$ and $\mathsf{e}', \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x$, and where $\mathsf{e}' \equiv \mathsf{e}[\vec{x} \leftarrow \vec{\mathsf{t}_\circ}]$ is some extension of e. By Defn. **??**, it is therefore sufficient to show that there exist s_t and t such that:

(a) $\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \mathrm{tree}\ \mathsf{t}\ \mathsf{n}$ for some n,

(b) $\mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \mathrm{tree}\ \mathsf{t}_x\ \&x$ for some $\langle \mathsf{t}_x \rangle \equiv \mathsf{s}_t(x)$ and $\mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t) = \{\vec{x}\}$, and

(c) $\mathsf{s}_h \supseteq \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{N}}}$.

Letting s_t be defined by $\mathsf{s}_t(x) = \mathsf{e}'(x)$ and $\mathsf{s}_t(n) = \mathsf{s}_h(n)$ (and undefined everywhere else), we get the second and third conditions directly. To show the first condition, we show that

$$\begin{aligned} &\forall \mathsf{e}, \mathsf{s}_h, \mathsf{h}.\ \mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^I \Rightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \\ &\forall \mathsf{e}, \mathsf{s}_h, \mathsf{h}.\ \mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K]\!]^I_{I'} \Rightarrow \exists \mathsf{c}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subcontext}\ \mathsf{c}\ I\ I' \end{aligned}$$

This follows by straightforward induction on the formula translation, with the cases for the structural formulæ following from Lemma 5.4. For example, for $K = N[K']$, we have:

$$\begin{aligned} \mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![N[K']]\!]^I_{I'} &\Rightarrow \mathsf{e}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists d_i, d_j.\, (i \doteq N) * (j \doteq N) * N \mapsto l,u,r,d_i * [\![K]\!]^{(\mathrm{nil}, d_i, n, d_j, \mathrm{nil})}_{I'} \\ &\Rightarrow \exists \mathsf{c}'.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists d_i, d_j.\, (i \doteq N) * (j \doteq N) * N \mapsto l,u,r,d_i * \mathrm{subcontext}\ \mathsf{c}'\ I\ I' \\ &\Rightarrow \exists \mathsf{n}, \mathsf{c}'.\, \mathsf{n} = [\![N]\!]\mathsf{s}_h \wedge \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subcontext}\ \mathsf{n}[\mathsf{c}']\ I\ I' \\ &\Rightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subcontext}\ c\ I\ I' \end{aligned}$$

The other cases are all similar.

Lemma 7.3 (Logic Translation). *For all P,*

$$\forall \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}, \mathsf{s}_h, \mathsf{h}.\ (\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!] \Rightarrow (\mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \Leftrightarrow \mathsf{e}_t, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)})$$

Proof. Assume for the moment that, for all $\mathsf{s}_h, \mathsf{h}, \mathsf{e}_t, \mathsf{s}_t$ such that $\mathsf{s}_h \supseteq \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{N}}}$ the following properties hold:

$$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^I \Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P$$
$$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} [\![K]\!]^I_{I'} \Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K$$

We first show that this assumption is enough to prove the lemma, and then prove the assumption.

Choose arbitrary e_t, s_t, t and $(\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!]$. Then, by Defn. 5.1, $\mathsf{s}_h \supseteq \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{N}}}$, $\mathsf{h} = \mathsf{h}_\mathsf{t} * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x$, where $\mathsf{s}_h, \mathsf{h}_\mathsf{t} \vDash_{\mathcal{H}} \text{tree } \mathsf{t}\ \mathsf{n}$ for some identifier n and, for all $x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)$, $\mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } \mathsf{t}_x\ \&x$ for some t_x with $\langle \mathsf{t}_x \rangle \equiv \mathsf{s}_t(x)$. This implies that, for all $x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)$, $\mathsf{e}_t \cup \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}_x \vDash \langle \mathsf{t}_x \rangle \equiv x \wedge \text{tree } \mathsf{t}_x\ \&x$, and hence $\mathsf{e}_t \cup \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x \vDash \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \langle \mathsf{t}_x \rangle \equiv x \wedge \text{tree } \mathsf{t}_x\ \&x$.

To show the left-to-right implication, first assume $\mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P$. Since $\mathsf{s}_h, \mathsf{h}_\mathsf{t} \vDash_{\mathcal{H}} \text{tree } \mathsf{t}\ \mathsf{n}$, there exists j such that $\mathsf{s}_h, \mathsf{h}_\mathsf{t} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (\mathrm{nil}, \mathsf{n}, \mathrm{nil}, \mathsf{j}, \mathrm{nil})$. Hence, by assumption, $\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}_\mathsf{t} \vDash_{\mathcal{H}} [\![P]\!]^{(\mathrm{nil}, \mathsf{n}, \mathrm{nil}, \mathsf{j}, \mathrm{nil})}$. It follows that $\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}_\mathsf{t} \cdot \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \mathsf{h}_x \vDash_{\mathcal{H}} [\![P]\!]^{(\mathrm{nil}, \mathsf{n}, \mathrm{nil}, \mathsf{j}, \mathrm{nil})} * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \langle \mathsf{t}_x \rangle \equiv x \wedge \text{tree } \mathsf{t}_x\ \&x$, and hence $\mathsf{e}_t, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists i, j.\, \exists \vec{x}.\, ([\![P]\!]^{(\mathrm{nil}, i, \mathrm{nil}, j, \mathrm{nil})} * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \exists t_x.\, \langle t_x \rangle \equiv x \wedge \text{tree } t_x\ \&x)$ for $\{\vec{x}\} = \mathrm{Var}_{\mathcal{T}_\circ}(\mathsf{s}_t)$, and hence $\mathsf{e}_t, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}$ as required.

For the reverse direction, assume $\mathsf{e}_t, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}$. It follows by Defn. 7.1 that, for some i and j, $\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^{(\mathrm{nil}, \mathsf{i}, \mathrm{nil}, \mathsf{j}, \mathrm{nil})} * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)} \exists t_x.\, \langle t_x \rangle \equiv x \wedge \text{tree } t_x\ \&x$. Hence, $\mathsf{h} = \mathsf{h}'_\mathsf{t} * \prod_{x \in \mathrm{vars}_{\mathcal{T}_\circ}\, \mathsf{s}_t} \mathsf{h}'_x$ with $\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}'_\mathsf{t} \vDash_{\mathcal{H}} [\![P]\!]^{(\mathrm{nil}, \mathsf{i}, \mathrm{nil}, \mathsf{j}, \mathrm{nil})}$ and $\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}'_x \vDash_{\mathcal{H}} \exists t_x.\, \langle t_x \rangle \equiv x \wedge \text{tree } t_x\ \&x$ for each $x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)$. By the assumption, there exists t' such that $\mathsf{s}_h, \mathsf{h}'_t \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ (\mathrm{nil}, \mathsf{i}, \mathrm{nil}, \mathsf{j}, \mathrm{nil}) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P$. We need to show that $\mathsf{t}' = \mathsf{t}$. For each $x \in \mathrm{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)$, there exists t_x and t'_x such that $\mathsf{e}_t \cup \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}_x \vDash \text{tree } \mathsf{t}_x\ \&x$ and $\mathsf{e}_t \cup \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}'_x \vDash \text{tree } \mathsf{t}'_x\ \&x$. Both h'_x and h'_x are subheaps of h, and equal to the trees starting from the same address $\&x$. Since the tree predicate is precise, $\mathsf{h}_x = \mathsf{h}'_x$ and hence $\mathsf{h}_t = \mathsf{h}'_t$. Since $\mathsf{s}_h, \mathsf{h}_t \vDash_{\mathcal{H}} \text{tree } \mathsf{t}'\ \mathsf{n}$ and $\mathsf{s}_h, \mathsf{h}'_t \vDash_{\mathcal{H}} \text{tree } \mathsf{t}'\ \mathsf{i}$, it follows that $\mathsf{i} = \mathsf{n}$ and $\mathsf{t} = \mathsf{t}'$, as required.

We now prove the property assumed at the start: that for all e_t, s_t, s_h, h, where $\mathsf{s}_h \supseteq \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{N}}}$:

$$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^I \Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P$$
$$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K]\!]^I_{I'} \Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K$$

This follows by mutual structural induction on P and K. The case for P is...

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![0]\!]^{(l,i,u,j,r)} \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} (l \doteq j) * (i \doteq r) \\
&\Leftrightarrow \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } 0\ (l,i,u,j,r) \wedge \mathsf{e}_t, \mathsf{s}_t, 0 \vDash_{\mathcal{T}} 0 \\
&\Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,i,u,j,r) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} 0
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![x]\!]^{I} \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists t.\, \langle t \rangle \equiv x \wedge \text{subtree } t\ I \\
&\Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \wedge \langle \mathsf{t} \rangle \equiv \mathsf{s}_t(x) \\
&\Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} x
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\circ[X]]\!]^{(l,i,u,j,r)} \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists d_i, d_j.(i \doteq j) * i \mapsto l,u,r,d_i * [\![X]\!]^{(\mathrm{nil},d_i,i,d_j,\mathrm{nil})} \\
&\Leftrightarrow \exists \mathsf{t}'.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists d_i, d_j.(i \doteq j) * i \mapsto l,u,r,d_i * \text{subtree } \mathsf{t}'\ (\mathrm{nil},d_i,i,d_j,\mathrm{nil}) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} X \\
&\Leftrightarrow \exists \mathsf{n}, \mathsf{t}'.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{n}[\mathsf{t}']\ (l,i,u,j,r) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} X \\
&\Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,i,u,j,r) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} \circ[X]
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![X|X']\!]^{(l,i,u,j,r)} \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists k_l, k_r.\, [\![X]\!]^{(l,i,u,k_l,k_r)} * [\![X']\!]^{(k_l,k_r,u,j,r)} \\
&\Leftrightarrow \exists \mathsf{t}_1, \mathsf{t}_2.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists k_l, k_r.\, \text{subtree } \mathsf{t}_1\ (l,i,u,k_l,k_r) * \text{subtree } \mathsf{t}_2\ (k_l,k_r,u,j,r) \wedge \mathsf{s}_t, \mathsf{t}_1 \vDash_{\mathcal{T}} X \wedge \mathsf{s}_t, \mathsf{t}_2 \vDash_{\mathcal{T}} X' \\
&\Leftrightarrow \exists \mathsf{t}_1, \mathsf{t}_2.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}_1|\mathsf{t}_2\ (l,i,u,j,r) \wedge \mathsf{s}_t, \mathsf{t}_1 \vDash_{\mathcal{T}} X \wedge \mathsf{s}_t, \mathsf{t}_2 \vDash_{\mathcal{T}} X' \\
&\Leftrightarrow \exists \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ (l,i,u,j,r) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} X|X'
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![t]\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ t\ I \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{t} = \mathsf{e}_t(t) \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} t
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\langle t \rangle]\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists t'.\ t' \simeq t \wedge \mathrm{subtree}\ t'\ I \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{t} = \mathsf{e}_t(t') \wedge \langle \mathsf{t} \rangle = \langle \mathsf{e}_t(\mathsf{t}) \rangle \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} \langle t \rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K \cdot P]\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists I'.\ [\![K]\!]^I_{I'} * [\![P]\!]^{I'} \\
&\Leftrightarrow \exists \mathsf{c}, \mathsf{t}'.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists I'.\ \mathrm{subcontext}\ \mathsf{c}\ I\ I' * \mathrm{subcontext}\ \mathsf{t}'\ I' \ \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{T}} K \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P \\
&\Leftrightarrow \exists \mathsf{c}, \mathsf{t}'.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \ \mathrm{ap}(\mathsf{c}, \mathsf{t}')\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}') \vDash_{\mathcal{T}} K \cdot P \qquad \text{(by Lemma 5.4)} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} K \cdot P
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K \blacktriangleleft P]\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ I \wedge \exists I'.\ [\![K]\!]^{I'}_I \mathrel{-\!\!*_\exists} [\![P]\!]^{I'} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists I', \mathsf{h}'.\ (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![K]\!]^{I'}_I \wedge \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^{I'} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists I', \mathsf{h}', \mathsf{c}, \mathsf{t}'.\ \left(\begin{array}{l} (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \mathrm{subcontext}\ \mathsf{c}\ I'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}'\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P \end{array}\right) \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists I', \mathsf{h}', \mathsf{c}, \mathsf{t}'.\ \left(\begin{array}{l} (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \mathrm{subcontext}\ \mathsf{c}\ I'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \ \mathrm{ap}(\mathsf{c}, \mathsf{t})\ I' \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}'\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P \end{array}\right) \qquad \text{(by Lemma 5.4)} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists I', \mathsf{h}', \mathsf{c}.\ \left(\begin{array}{l} (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \mathrm{subcontext}\ \mathsf{c}\ I'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \ \mathrm{ap}(\mathsf{c}, \mathsf{t})\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}) \vDash_{\mathcal{T}} P \end{array}\right) \qquad \text{(by Lemma A.2)} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists I', \mathsf{h}', \mathsf{c}.\ (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \mathrm{subcontext}\ \mathsf{c}\ I'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}) \vDash_{\mathcal{T}} P \quad \text{(by Lemma 5} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists \mathsf{c}.\ \mathrm{ap}(\mathsf{c}, \mathsf{t})\!\downarrow \wedge\, \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}) \vDash_{\mathcal{T}} P \qquad \text{(by Lemma A.3)} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} K \blacktriangleleft P
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P \Rightarrow P']\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ I \wedge [\![P]\!]^I \Rightarrow [\![P']\!]^I \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \left(\begin{array}{l} \exists \mathsf{t}'.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P \\ \Rightarrow \exists \mathsf{t}''.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}''\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}'' \vDash_{\mathcal{T}} P' \end{array}\right) \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \left(\begin{array}{l} \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists \mathsf{t}'.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P \\ \Rightarrow \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \exists \mathsf{t}''.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}''\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}'' \vDash_{\mathcal{T}} P' \end{array}\right) \quad \left(\text{by } \begin{array}{l} A \wedge (B \Rightarrow C) \Leftrightarrow \\ A \wedge (A \wedge B \Rightarrow A \wedge C \end{array}\right. \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \left(\begin{array}{l} \exists \mathsf{t}'.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}'\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P \\ \Rightarrow \exists \mathsf{t}''.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}''\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}'' \vDash_{\mathcal{T}} P' \end{array}\right) \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \left(\begin{array}{l} \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \\ \Rightarrow \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P' \end{array}\right) \qquad \text{(by Lemma A.2)} \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \Rightarrow \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P' \qquad \big(\text{by } A \wedge (B \Rightarrow C) \Leftrightarrow A \wedge (A \wedge B \Rightarrow A \wedge C) \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \Rightarrow P'
\end{aligned}
$$

$$
\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\mathrm{false}]\!]^I \Leftrightarrow \mathrm{false} \Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} \mathrm{false}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\exists n.\ P]\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists n.\ [\![P]\!]^I \\
&\Leftrightarrow \exists \mathsf{n}.\ \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h[n \leftarrow \mathsf{n}], \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^I \\
&\Leftrightarrow \exists \mathsf{n}, \mathsf{t}.\ \mathsf{s}_h[n \leftarrow \mathsf{n}], \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t[n \leftarrow \mathsf{n}], \mathsf{t} \vDash_{\mathcal{T}} P \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} \exists n.\ P
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\exists t.\ P]\!]^I \\
&\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists t.\ [\![P]\!]^I \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{e}_t[t \leftarrow \mathsf{t}] \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^I \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t[t \leftarrow \mathsf{t}], \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \\
&\Leftrightarrow \exists \mathsf{t}.\ \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \mathrm{subtree}\ \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} \exists t.\ P
\end{aligned}
$$

...and for K...

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![N[K]]\!]_{I'}^{(l,i,u,j,r)}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists d_i, d_j.\, (i \doteq N) * (j \doteq N) * N \mapsto l,u,r,d_i * [\![K]\!]_{I'}^{(\mathrm{nil},d_i,n,d_j,\mathrm{nil})}$
$\Leftrightarrow \exists \mathsf{c}'.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists d_i, d_j.\, (i \doteq N) * (j \doteq N) * N \mapsto l,u,r,d_i * \text{subcontext } \mathsf{c}' \ (\mathrm{nil},d_i,n,d_j,\mathrm{nil})\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}' \vDash_{\mathcal{C}} K$
$\Leftrightarrow \exists \mathsf{c}'.\, \exists \mathsf{n}.\, \mathsf{n} = [\![N]\!]\mathsf{s}_t \wedge \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{n}[\mathsf{c}']\ (l,i,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}' \vDash_{\mathcal{C}} K$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ (l,i,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} N[K]$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P|K]\!]_{I'}^{(l,i,u,j,r)}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists k_l, k_r.\, [\![P]\!]^{(l,i,u,k_l,k_r)} * [\![K]\!]_{I'}^{(k_l,k_r,u,j,r)}$
$\Leftrightarrow \exists \mathsf{t}, \mathsf{c}'.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists k_l, k_r.\, \text{subtree } \mathsf{t}\ (l,i,u,k_l,k_r) * \text{subcontext } \mathsf{c}'\ (k_l,k_r,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}' \vDash_{\mathcal{C}} K$
$\Leftrightarrow \exists \mathsf{t}, \mathsf{c}'.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{t}|\mathsf{c}'\ (l,i,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}|\mathsf{c}' \vDash_{\mathcal{C}} P|K$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ (l,i,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} P|K$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K|P]\!]_{I'}^{(l,i,u,j,r)}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists k_l, k_r.\, [\![K]\!]_{I'}^{(l,i,u,k_l,k_r)} * [\![P]\!]^{(k_l,k_r,u,j,r)}$
$\Leftrightarrow \exists \mathsf{c}', \mathsf{t}.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists k_l, k_r.\, \text{subcontext } \mathsf{c}'\ (l,i,u,k_l,k_r)\ I' * \text{subtree } \mathsf{t}\ (k_l,k_r,u,j,r) \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}' \vDash_{\mathcal{C}} K \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P$
$\Leftrightarrow \exists \mathsf{c}', \mathsf{t}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}'|\mathsf{t}\ (l,i,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}'|\mathsf{t} \vDash_{\mathcal{C}} P|K$
$\Leftrightarrow \exists \mathsf{c}'.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ (l,i,u,j,r)\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} P|K$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![-]\!]_{(l',i',u',j',r')}^{(l,i,u,j,r)}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} (l{=}l') * (i{=}i') * (u{=}u') * (j{=}j') * (r{=}r')$
$\Leftrightarrow \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } -\ (l,i,u,j,r)\ (l',i',u',j',r')$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ (l,i,u,j,r)\ (l',i',u',j',r') \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} -$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P' \blacktriangleright P]\!]_{I'}^{I}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } I\ I' \wedge [\![P']\!]^{I'} \mathrel{-\!\!*_{\exists}} [\![P]\!]^{I}$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \exists \mathsf{h}'.\, (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![P']\!]^{I'} \wedge \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^{I}$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \exists \mathsf{h}', \mathsf{t}, \mathsf{t}'.\, \begin{pmatrix} (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P' \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \end{pmatrix}$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \exists \mathsf{h}', \mathsf{t}, \mathsf{t}'.\, \begin{pmatrix} (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P' \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathrm{ap}(\mathsf{c}, \mathsf{t}')\ I' \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P \end{pmatrix}$ (by Lemma 5.4)
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \exists \mathsf{h}', \mathsf{t}'.\, \begin{pmatrix} (\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P' \\ \wedge\, \mathsf{s}_h, \mathsf{h}' * \mathsf{h} \vDash_{\mathcal{H}} \text{subtree } \mathrm{ap}(\mathsf{c}, \mathsf{t}')\ I \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}') \vDash_{\mathcal{T}} P \end{pmatrix}$ (by Lemma A.2)
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \exists \mathsf{h}', \mathsf{t}'.\, \big((\mathsf{h}' * \mathsf{h})\!\downarrow \wedge\, \mathsf{s}_h, \mathsf{h}' \vDash_{\mathcal{H}} \text{subtree } \mathsf{t}'\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}') \vDash_{\mathcal{T}} P\big)$ (by Lemm
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \exists \mathsf{t}'.\, \mathrm{ap}(\mathsf{c}, \mathsf{t}')\!\downarrow \wedge\, \mathsf{e}_t, \mathsf{s}_t, \mathsf{t}' \vDash_{\mathcal{T}} P' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathrm{ap}(\mathsf{c}, \mathsf{t}') \vDash_{\mathcal{T}} P$ (by Lemma A.3)
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} P \blacktriangleright P'$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K \Rightarrow K']\!]_{I'}^{I}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } I\ I' \wedge [\![K]\!]_{I'}^{I} \Rightarrow [\![K']\!]_{I'}^{I}$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \begin{pmatrix} \exists \mathsf{c}'.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}'\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}' \vDash_{\mathcal{C}} K \\ \Rightarrow \exists \mathsf{c}''.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}''\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c}'' \vDash_{\mathcal{C}} K' \end{pmatrix}$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \Rightarrow \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K'$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K \Rightarrow K'$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\mathrm{False}]\!]_{I'}^{I} \Leftrightarrow \mathrm{false} \Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} \mathrm{False}$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\exists n.\, K]\!]_{I'}^{I}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists n.\, [\![K]\!]_{I'}^{I}$
$\Leftrightarrow \exists \mathsf{n}.\, \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h[n \leftarrow \mathsf{n}], \mathsf{h} \vDash_{\mathcal{H}} [\![K]\!]_{I'}^{I}$
$\Leftrightarrow \exists \mathsf{n}, \mathsf{c}.\, \mathsf{s}_h[n \leftarrow \mathsf{n}], \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t[n \leftarrow \mathsf{n}], \mathsf{c} \vDash_{\mathcal{C}} K$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} \exists n.\, K$

$\mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![\exists t.\, K]\!]_{I'}^{I}$
$\Leftrightarrow \mathsf{e}_t \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \exists t.\, [\![K]\!]_{I'}^{I}$

$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{e}_t[t \leftarrow \mathsf{c}] \uplus \mathsf{s}_t|_{\mathrm{Var}_{\mathcal{T}_\circ}}, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![K]\!]_{I'}^{I}$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t[t \leftarrow \mathsf{c}], \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} K$
$\Leftrightarrow \exists \mathsf{c}.\, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} \text{subcontext } \mathsf{c}\ I\ I' \wedge \mathsf{e}_t, \mathsf{s}_t, \mathsf{c} \vDash_{\mathcal{C}} \exists t.\, K$

Theorem 7.4 (Hoare Triple equivalence). *For all P, $\mathbb{C}$, Q and $\vec{x} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P, Q)$:*

$$\{P\}\ \mathbb{C}\ \{Q\} \Leftrightarrow \{[\![P]\!]_{\vec{x}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{x}}\}$$

Proof. Assume for the moment that for all P, $\mathbb{C}$, Q and $\vec{x}, \vec{y} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P, Q)$:

$$\{[\![P]\!]_{\vec{x}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{x}}\} \Leftrightarrow \{[\![P]\!]_{\vec{y}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{y}}\}$$

We first show that assertion is enough to prove the theorem, and then prove the assumption.

For the left-to-right implication, we take any e_h, s_h, h satisfying $\mathsf{e}_h, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{x}}$ and aim to show that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow$ fault and that if $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$ then $\mathsf{e}_h, \mathsf{s}'_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![Q]\!]_{\vec{x}}$. By Lemma 7.2, $\exists \mathsf{s}_t, \mathsf{t}$ such that $(\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!]$ and $\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t) = \{\vec{x}\}$. It follows from Lemma 7.3 that $\mathsf{e}_h|_{\text{LVar}_{\mathcal{T}}}, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P$. By the left Hoare Triple, we know that if $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \mathsf{s}'_t, \mathsf{t}'$ then $\mathsf{e}_h|_{\text{LVar}_{\mathcal{T}}}, \mathsf{s}'_t, \mathsf{t}' \vDash_{\mathcal{T}} Q$, and that $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \not\rightsquigarrow$ fault. By Theorem 6.2, it follows immediately that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow$ fault. Furthermore, whenever $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$ then $(\mathsf{s}'_h, \mathsf{h}') \in [\![\mathsf{s}'_t, \mathsf{t}']\!]$ for some $\mathsf{s}'_t, \mathsf{t}'$ where $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \mathsf{s}'_t, \mathsf{t}'$. Since $\mathsf{e}_h|_{\text{LVar}_{\mathcal{T}}}, \mathsf{s}'_t, \mathsf{t}' \vDash_{\mathcal{T}} Q$, and since $\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}'_t) = \text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t) = \{\vec{x}\}$, we have $\mathsf{e}_h, \mathsf{s}'_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![Q]\!]_{\vec{x}}$ by Lemma 7.3.

The right-to-left implication is analogous. We take any e_t, s_t, t satisfying $\mathsf{e}_t, \mathsf{s}_t, \mathsf{t} \vDash_{\mathcal{T}} P$ and aim to show that $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \not\rightsquigarrow$ fault, and that if $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \mathsf{s}'_t, \mathsf{t}'$ then $\mathsf{e}_t, \mathsf{s}'_t, \mathsf{t}' \vDash_{\mathcal{T}} Q$. By Lemma 7.3, we have that $\mathsf{e}_t, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}$ for all $(\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!]$. By the assumed result above, we know that $\{[\![P]\!]_{\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}\}$. Hence, if $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$ then $\mathsf{e}_t, \mathsf{s}'_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![Q]\!]_{\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}$ and that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow$ fault. By Theorem 6.2, it follows immediately that $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \not\rightsquigarrow$ fault. Furthermore, whenever $\mathbb{C}, \mathsf{s}_t, \mathsf{t} \rightsquigarrow \mathsf{s}'_t, \mathsf{t}'$ then there exist $(\mathsf{s}_h, \mathsf{h}) \in [\![\mathsf{s}_t, \mathsf{t}]\!]$ and $(\mathsf{s}'_h, \mathsf{h}') \in [\![\mathsf{s}'_t, \mathsf{t}]\!]$ such that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$. Since $\mathsf{e}_t, \mathsf{s}'_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![Q]\!]_{\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t)}$, and since $\text{vars}_{\mathcal{T}_\circ}(\mathsf{s}_t) = \text{vars}_{\mathcal{T}_\circ}(\mathsf{s}'_t)$, we have $\mathsf{e}_t, \mathsf{s}'_t, \mathsf{t}' \vDash_{\mathcal{T}} Q$ by Lemma 7.3.

We now prove the property assumed at the start: namely, that for all P, $\mathbb{C}$, Q and $\vec{x}, \vec{y} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \text{free}_{\mathcal{T}_\circ}(P, Q)$:

$$\{[\![P]\!]_{\vec{x}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{x}}\} \Leftrightarrow \{[\![P]\!]_{\vec{y}}\}\ [\![\mathbb{C}]\!]\ \{[\![Q]\!]_{\vec{y}}\}$$

By symmetry, we need only prove the left-to-right implication. We therefore assume that $\mathsf{e}_h, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{y}}$ and aim to show that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow$ fault and that if $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$ then $\mathsf{e}_h, \mathsf{s}'_h, \mathsf{h}' \vDash_{\mathcal{H}} [\![P]\!]_{\vec{y}}$. By Defn. 7.1, we have $\mathsf{e}_h, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]^{(\text{nil},\text{i},\text{nil},\text{j},\text{nil})} * \prod_{x \in \vec{y}} \exists t_x.\ \langle t_x \rangle \equiv x \wedge \text{tree } t_x\ \&x$ for some i and j. Hence, $\mathsf{h} = \mathsf{h_t} * \prod_{x \in \vec{y}} \mathsf{h}_x$ with $\mathsf{e}_t, \mathsf{s}_h, \mathsf{h_t} \vDash_{\mathcal{H}} [\![P]\!]^{(\text{nil},\text{i},\text{nil},\text{j},\text{nil})}$ and $\mathsf{e}_t, \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \exists t_x.\ \langle t_x \rangle \equiv x \wedge \text{tree } t_x\ \&x$ for each $x \in \vec{y}$. Now consider the heap $\mathsf{h}_{\vec{x}} \triangleq \mathsf{h}_t * \prod_{x \in \vec{x} \cap \vec{y}} \mathsf{h}_x$ containing just the store variables in $\vec{x}$; it follows immediately that $\mathsf{e}_h, \mathsf{s}_h, \mathsf{h} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{x} \cap \vec{y}}$. For $x \in \vec{x} \setminus \vec{y}$, define $\mathsf{h}_x \triangleq 0$, and note that $\mathsf{e}_h, \mathsf{s}_h, \mathsf{h}_x \vDash_{\mathcal{H}} \text{tree } 0\ \&x$, and hence $\mathsf{e}_h, \mathsf{s}_h, \text{emp} \vDash_{\mathcal{H}} \exists t_x.\ \langle t_x \rangle \equiv x \wedge \text{tree } t_x\ \&x$. Since $\mathsf{h}_{\vec{x}} = \mathsf{h}_{\vec{x}} * \prod_{x \in \vec{x} \setminus \vec{y}} \mathsf{h}_x$, it follows that $\mathsf{e}_h, \mathsf{s}_h, \mathsf{h}_{\vec{x}} \vDash_{\mathcal{H}} [\![P]\!]_{\vec{x}}$.

By the left Hoare Triple, we know that if $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h}_{\vec{x}} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'_{\vec{x}}$ then $\mathsf{e}_h, \mathsf{s}'_h, \mathsf{h}'_{\vec{x}} \vDash_{\mathcal{T}} [\![Q]\!]_{\vec{x}}$, and that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h}_{\vec{x}} \not\rightsquigarrow$ fault. However, we know that the *footprint* of $[\![\mathbb{C}]\!]$ consists of at most h_t and the parts of the heap reachable from $\&x$ for $x \in \text{free}_{\mathcal{T}_\circ}(\mathbb{C})$. We also know that $\vec{x}, \vec{y} \supseteq \text{free}_{\mathcal{T}_\circ}(\mathbb{C})$. Hence, letting $\mathsf{h}_{fp} = \mathsf{h}_t * \prod_{x \in \text{free}_{\mathcal{T}_\circ}(\mathbb{C})} \mathsf{h}_x$, we have $\mathsf{h}_{\vec{x}} = \mathsf{h}_{fp} * \mathsf{h}_{\vec{x}-fp}$ and $\mathsf{h} = \mathsf{h}_{fp} * \mathsf{h}_{\vec{y}-fp}$ where $\mathsf{h}_{\vec{x}-fp} = \prod_{x \in \vec{x} \setminus \text{free}_{\mathcal{T}_\circ}(\mathbb{C})} \mathsf{h}_x$ and $\mathsf{h}_{\vec{y}-fp} = \prod_{x \in \vec{y} \setminus \text{free}_{\mathcal{T}_\circ}(\mathbb{C})} \mathsf{h}_x$. Since h_{fp} contains the footprint, we know that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h}_{fp} \not\rightsquigarrow$ fault; by safety monotonicity it follows that $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \not\rightsquigarrow$ fault. For the same reasons, we know that if $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'$ then $\mathsf{h}' = \mathsf{h}'_{fp} * \mathsf{h}_{\vec{y}-fp}$ and $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h}_{fp} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'_{fp}$, and hence by the frame property, $[\![\mathbb{C}]\!], \mathsf{s}_h, \mathsf{h}_{\vec{x}} \rightsquigarrow \mathsf{s}'_h, \mathsf{h}'_{fp} * \mathsf{h}_{\vec{x}-fp}$. Hence we see that $\mathsf{e}_h, \mathsf{s}'_h, \mathsf{h}'_{fp} * \mathsf{h}_{\vec{x}-fp} \vDash_{\mathcal{H}} [\![Q]\!]_{\vec{x}}$ and therefore $\mathsf{e}_h, \mathsf{s}'_h, \mathsf{h} \vDash_{\mathcal{H}} [\![Q]\!]_{\vec{y}}$.

In order to prove Theorem 8.3 and Theorem 8.4, we must first give simple low-level specifications for the various utility functions given in Section 6. These all follow by standard inductive proofs. The specifications do not depend on crusts, since the utility functions do not do any pointer surgery.

Lemma ?? (Utility fuction specifications). *The utility fns (Defn. ??) satisfy the following specs:*

$$\{\text{subtree } t\ (l,n,u,j,\text{nil})\}\ \text{dispose-forest } n\ \{\text{emp}\}$$

$$\begin{array}{l}\{\text{subtree } t\ (l,n,u,j,\text{nil})\}\\ \quad (i',j') := \text{copy-forest } n\ (l',u',r')\\ \{\text{subtree } t\ (l,n,u,j,\text{nil}) * \exists t'.\, \text{subtree } t'(l',i',u',j',r') \wedge t \simeq t'\}\end{array}$$

$$\begin{array}{l}\{\text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil})\}\\ \quad b := \text{compare-forests } n\ n'\\ \left\{\begin{array}{l}\text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil}) \wedge\\ \quad ((b = 0 \wedge t \not\simeq t') \vee (b = 1 \wedge t \simeq t'))\end{array}\right\}\end{array}$$

Note the use of nil as the right interface pointer in all the specifications; this follows since all the utility functions recurse along the right-sibling axis to the end of the forest.

Proof. Follows by induction:

dispose-forest *n*

$\{\text{subtree } t\ (l,n,u,j,\text{nil})\}$
if $n = \text{nil}$ then skip
  $\{t \equiv 0 \wedge \text{subtree } t\ (l,\text{nil},u,j,\text{nil})\} \Rightarrow \{\text{emp}\}$
else
  $\{\exists t_1, t_2.\, t \equiv n[t_1] | t_2 \wedge \text{subtree } t\ (l,n,u,j,\text{nil})$
  $\{\exists t_1, t_2, i_r, i_d, j_d.\, n \mapsto l, u, i_r, i_d * \text{subtree } t_1\ (\text{nil},i_d,n,j_d,\text{nil}) * \text{subtree } t_2\ (n,i_r,u,j,\text{nil})\}$
  $r := [n.\text{right}]$; dispose-forest r ;
  $\{\exists t_1, i_d, j_d.\, n \mapsto l, u, r, i_d * \text{subtree } t_1\ (\text{nil},i_d,n,j_d,\text{nil})\}$
  $d := [n.\text{down}]$; dispose-forest d ;
  $\{n \mapsto l, u, r, d\}$
  dispose-node n
  $\{\text{emp}\}$

$(i', j') :=$ copy-forest $n\ (l', u', r')$

$\{\text{subtree } t\ (l,n,u,j,\text{nil})\}$

if $n = \text{nil}$ then $i' := r'$; $j' := l'$

$$\{t \equiv 0 \land \text{subtree } t\ (l,n,u,j,\text{nil}) * i' \doteq r' * j' \doteq l'\}$$

$$\{t \equiv 0 \land \text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } 0\ (l',i',u',j',r')\}$$

else

$$\{\exists t_1, t_2.\, t \equiv n[t_1]|t_2 \land \text{subtree } t\ (l,n,u,j,\text{nil})\}$$

$i' :=$ new-node() ; $[i'.\text{left}] := l'$; $[i'.\text{up}] := u'$;

$$\left\{ \exists t_1, t_2.\, t \equiv n[t_1]|t_2 \land \exists i_r, i_d, j_d.\, n \mapsto l, u, i_r, i_d * i' \mapsto l', u', \text{nil}, \text{nil} \begin{array}{l} * \text{ subtree } t_1\ (\text{nil},i_d,n,j_d,\text{nil}) \\ * \text{ subtree } t_2\ (n,i_r,u,j,\text{nil}) \end{array} \right\}$$

$r := [n.\text{right}]$; $(r'_i, j') :=$ copy-forest $r\ (i', u', r')$;

$$\left\{ \exists t_1, t_2.\, t \equiv n[t_1]|t_2 \land \exists i_d, j_d.\, n \mapsto l, u, r, i_d * i' \mapsto l', u', \text{nil}, \text{nil} \begin{array}{l} * \text{ subtree } t_1\ (\text{nil},i_d,n,j_d,\text{nil}) \\ * \text{ subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \exists t'_2.\, \text{subtree } t'_2\ (i',r_i{}',u',j',r') \land t_2 \simeq t'_2 \end{array} \right\}$$

$d := [n.\text{down}]$; $(d'_i, d'_j) :=$ copy-forest $d\ (\text{nil}, i', \text{nil})$;

$$\left\{ \exists t_1, t_2.\, t \equiv n[t_1]|t_2 \land \exists j_d.\, n \mapsto l, u, r, d * i' \mapsto l', u', \text{nil}, \text{nil} \begin{array}{l} * \text{ subtree } t_1\ (\text{nil},d,n,j_d,\text{nil}) \\ * \exists t'_1.\, \text{subtree } t'_1\ (\text{nil},d'_i,i',d'_j,\text{nil}) \land t_1 \simeq t'_1 \\ * \text{ subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \exists t'_2.\, \text{subtree } t'_2\ (i',r_i{}',u',j',r') \land t_2 \simeq t'_2 \end{array} \right\}$$

$[i'.\text{right}] := r'_i$; $[i'.\text{down}] := d'_i$

$$\left\{ \exists t_1, t_2.\, t \equiv n[t_1]|t_2 \land \exists j_d.\, n \mapsto l, u, r, d * i' \mapsto l', u', r'_i, d'_i \begin{array}{l} * \text{ subtree } t_1\ (\text{nil},d,n,j_d,\text{nil}) \\ * \exists t'_1.\, \text{subtree } t'_1\ (\text{nil},d'_i,i',d'_j,\text{nil}) \land t_1 \simeq t'_1 \\ * \text{ subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \exists t'_2.\, \text{subtree } t'_2\ (i',r_i{}',u',j',r') \land t_2 \simeq t'_2 \end{array} \right\}$$

$$\{\text{subtree } t\ (l,n,u,j,\text{nil}) * \exists t.\, \text{subtree } t'\ (l',i',u',j',r') \land t \simeq t'\}$$

b := compare-forests n n'

$\{\text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil})\}$

if $(n = \text{nil} \wedge n' = \text{nil})$ then $b := 1$

$\quad\{t \equiv 0 \wedge t' \equiv 0 \wedge \text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil}) \wedge (b = 1)\}$

else if $(n = \text{nil} \vee n' = \text{nil})$ then $b := 0$

$$\left\{\exists t_1, t_2. \begin{pmatrix} t \equiv 0 \wedge t' \equiv n'[t_1]|t_2 \\ \vee\, t \equiv n[t_1]|t_2 \wedge t' \equiv 0 \end{pmatrix} \wedge \text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil}) \wedge (b{=}0)\right\}$$

else

$$\left\{\exists t_1, t_2, t_1', t_2'. \begin{matrix} t \equiv n[t_1]|t_2 \wedge \\ t' \equiv n'[t_1']|t_2' \end{matrix} \wedge \text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil})\right\}$$

$\quad r := [n.\text{right}]$; $r' := [n'.\text{right}]$;

$$\left\{\exists t_1, t_2, t_1', t_2'. \begin{matrix} t \equiv n[t_1]|t_2 \wedge \\ t' \equiv n'[t_1']|t_2' \end{matrix} \wedge \exists i_d, j_d, i_d', j_d'.\, n \mapsto l, u, i_d, r * n' \mapsto l', u', i_d', r' \begin{matrix} * \text{subtree } t_1\ (\text{nil},i_d,n,j_d,\text{nil}) \\ * \text{subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \text{subtree } t_1'\ (\text{nil},i_d',n',j_d',\text{nil}) \\ * \text{subtree } t_2'\ (n',r',u',j',\text{nil}) \end{matrix}\right\}$$

$\quad b :=$ compare-forests r r' ;

$$\left\{\exists t_1, t_2, t_1', t_2'. \begin{matrix} t \equiv n[t_1]|t_2 \wedge \\ t' \equiv n'[t_1']|t_2' \end{matrix} \wedge \exists i_d, j_d, i_d', j_d'. \begin{matrix} n \mapsto l, u, i_d, r \\ * \, n' \mapsto l', u', i_d', r' \end{matrix} \begin{matrix} * \text{subtree } t_1\ (\text{nil},i_d,n,j_d,\text{nil}) \\ * \text{subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \text{subtree } t_1'\ (\text{nil},i_d',n',j_d',\text{nil}) \\ * \text{subtree } t_2'\ (n',r',u',j',\text{nil}) \end{matrix} \wedge \begin{pmatrix} (b = 0 \wedge t_2 \not\simeq t_2') \vee \\ (b = 1 \wedge t_2 \simeq t_2') \end{pmatrix}\right\}$$

$\quad$ if $b = 0$ then skip

$\quad\quad\{\exists t_1, t_2, t_1', t_2'.\, t \equiv n[t_1]|t_2 \wedge t' \equiv n'[t_1']|t_2' \wedge \text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil}) \wedge (b = 0 \wedge t_2 \not\simeq t_2')\}$

$\quad$ else $d := [n.\text{down}]$; $d' := [n'.\text{down}]$;

$$\left\{\exists t_1, t_2, t_1', t_2'. \begin{matrix} t \equiv n[t_1]|t_2 \wedge \\ t' \equiv n'[t_1']|t_2' \end{matrix} \wedge \exists j_d, j_d'. \begin{matrix} n \mapsto l, u, d, r \\ * \, n' \mapsto l', u', d', r' \end{matrix} \begin{matrix} * \text{subtree } t_1\ (\text{nil},d,n,j_d,\text{nil}) \\ * \text{subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \text{subtree } t_1'\ (\text{nil},d',n',j_d',\text{nil}) \\ * \text{subtree } t_2'\ (n',r',u',j',\text{nil}) \end{matrix} \wedge\ (t_2 \simeq t_2')\right\}$$

$\quad\quad b :=$ compare-forests d d'

$$\left\{\exists t_1, t_2, t_1', t_2'. \begin{matrix} t \equiv n[t_1]|t_2 \wedge \\ t' \equiv n'[t_1']|t_2' \end{matrix} \wedge \exists j_d, j_d'. \begin{matrix} n \mapsto l, u, d, r \\ * \, n' \mapsto l', u', d', r' \end{matrix} \begin{matrix} * \text{subtree } t_1\ (\text{nil},d,n,j_d,\text{nil}) \\ * \text{subtree } t_2\ (n,r,u,j,\text{nil}) \\ * \text{subtree } t_1'\ (\text{nil},d',n',j_d',\text{nil}) \\ * \text{subtree } t_2'\ (n',r',u',j',\text{nil}) \end{matrix} \begin{matrix} \wedge \begin{pmatrix} (b = 0 \wedge t_1 \not\simeq t_1') \vee \\ (b = 1 \wedge t_1 \simeq t_1') \end{pmatrix} \\ \wedge\, (t_2 \simeq t_2') \end{matrix}\right\}$$

$\{\text{subtree } t\ (l,n,u,j,\text{nil}) * \text{subtree } t'\ (l',n',u',j',\text{nil}) \wedge ((b = 0 \wedge t \not\simeq t') \vee (b = 1 \wedge t \simeq t'))\}$

Lemma ?? (Copy expression specifications). *The copy-expression function (Defn. **??**) satisfies the following specification. Given* $\vec{x} \supseteq \mathrm{free}_{\mathcal{T}_\circ}(X)$*:*

$$\begin{array}{c}\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\}\\ (i,j) := \text{copy-expression}\ X\ (l,u,r)\\ \{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![X]\!]^{(l,i,u,j,r)}\}\end{array}$$

Proof. Given inductively, using a case-by-case analysis:

for $X \equiv 0$:
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\}$
$i := r\ ;\ j := l$
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![0]\!]^{(l,i,u,j,r)}\}$

for $X \equiv x'$:
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\}$
$\{\prod_{x\in\vec{x}-x'}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * (\exists t.\, \langle t\rangle \equiv x' \wedge \mathrm{tree}\ t\ \&x')\}$
$(i,j) := \text{copy-forest}\ \&x'\ (l,u,r)$
$\{\prod_{x\in\vec{x}-x'}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * (\exists t.\, \langle t\rangle \equiv x' \wedge \mathrm{tree}\ t\ \&x' * \exists t'.\, \mathrm{subtree}\ t'\ (l,i,u,j,r) \wedge t' \simeq t)\}$
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![x']\!]^{(l,i,u,j,r)}\}$

for $X \equiv \circ[X']$:
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\}$
$i := \text{new-node}()\ ;\ j := i\ ;\ [i.\mathrm{left}] := l\ ;\ [i.\mathrm{up}] := u\ ;\ [i.\mathrm{right}] := r\ ;$
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * i \doteq j * i \mapsto l,u,r,\mathrm{nil}\}$
$(d_i,d_j) := \text{copy-expression}\ X'\ (\mathrm{nil},i,\mathrm{nil})\ ;\ [i.\mathrm{down}] := d_i$
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * i \doteq j * i \mapsto l,u,r,\mathrm{nil} * [\![X']\!]^{(\mathrm{nil},d_i,i,d_j,\mathrm{nil})}\}$
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![\circ[X']]\!]^{(l,i,u,j,r)}\}$

for $X \equiv X_1|X_2$:
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\}$
$(i,k_l) := \text{copy-expression}\ X_1\ (l,u,\mathrm{nil})\ ;$
$\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![X_1]\!]^{(l,i,u,k_l,\mathrm{nil})}\}$
if $i = \mathrm{nil}$ then $(i,j) := \text{copy-expression}\ X_2\ (l,u,r)$
  $\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![X_1|X_2]\!]^{(l,i,u,j,\mathrm{nil})}\}$
else $(k_r,j) := \text{copy-expression}\ X_2\ (k_l,u,r)\ ;\ [k_l.\mathrm{right}] := k_r$
  $\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![X_1]\!]^{(l,i,u,k_l,k_r)} * [\![X_2]\!]^{(k_l,k_r,u,j,r)}\}$
  $\{\prod_{x\in\vec{x}}(\exists t.\, \langle t\rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) * [\![X_1|X_2]\!]^{(l,i,u,j,r)}\}$

Lemma ?? (Boolean expression specifications). *The Boolean expression translations (Defn. ??) satisfy the following specification. Given* $\vec{x} \supseteq \text{free}_{\mathcal{T}_\circ}(B)$*:*

$$\{[\![B]\!]^I_{\vec{x}}\}\ [\![B]\!]_1\ \{[\![B]\!]^I_{\vec{x}} \wedge [\![B]\!]_2\}$$

For $(i, j) :=$ copy-expression X (l, u, r), the specification states that execution creates a tree of shape X with the appropriate interface. For the Boolean test $[\![B]\!]_1$, the specification states that executing the test on a tree satisfying B correctly sets up the low-level Boolean $[\![B]\!]_2$.

Proof. We show this in two steps. First we prove that for all formulæ P:

$$\{[\![P]\!]^I_{\vec{x}}\}\ [\![B]\!]_1\ \{[\![P]\!]^I_{\vec{x}} * \textstyle\prod_{(X=X')\in B}(b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)\}$$

for $B =$ false: $\{[\![P]\!]^I_{\vec{x}}\}$ skip $\{[\![P]\!]^I_{\vec{x}}\}$

for $B = N{=}N'$: $\{[\![P]\!]^I_{\vec{x}}\}$ skip $\{[\![P]\!]^I_{\vec{x}}\}$

for $B = X{=}X'$:
$\{\exists\vec{x}. \prod_{x\in\vec{x}}(\exists t. \langle t\rangle \equiv x \wedge \text{tree } t\ \&x) * [\![P]\!]^I\}$
$(i, j) :=$ copy-expression X (nil, nil, nil) ; $(i', j') :=$ copy-expression X' (nil, nil, nil) ;
$\{\exists\vec{x}. \prod_{x\in\vec{x}}(\exists t. \langle t\rangle \equiv x \wedge \text{tree } t\ \&x) * [\![P]\!]^I * [\![X]\!]^{(\text{nil},i,\text{nil},j,\text{nil})} * [\![X']\!]^{(\text{nil},i',\text{nil},j',\text{nil})}\}$
$\left\{\exists\vec{x}. \prod_{x\in\vec{x}}(\exists t. \langle t\rangle \equiv x \wedge \text{tree } t\ \&x) * [\![P]\!]^I * \exists t, t'. \begin{array}{l}\text{subtree } t\ (\text{nil},i,\text{nil},j,\text{nil}) * \text{subtree } t'\ (\text{nil},i',\text{nil},j',\text{nil}) \wedge \\ (t \simeq t' \wedge [\![X{=}X']\!]^I \vee t \not\simeq t' \wedge [\![X{\neq}X']\!]^I)\end{array}\right\}$
$b_{(X=X')} :=$ compare-forests $i\ i'$;
$\left\{\exists\vec{x}. \prod_{x\in\vec{x}}(\exists t. \langle t\rangle \equiv x \wedge \text{tree } t\ \&x) * [\![P]\!]^I * \exists t, t'. \begin{array}{l}\text{subtree } t\ (\text{nil},i,\text{nil},j,\text{nil}) * \text{subtree } t'\ (\text{nil},i',\text{nil},j',\text{nil}) \wedge \\ (b_{(X=X')} = 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} = 0 \wedge [\![X{\neq}X']\!]^I)\end{array}\right\}$
dispose-forest i ; dispose-forest i'
$\{\exists\vec{x}. \prod_{x\in\vec{x}}(\exists t. \langle t\rangle \equiv x \wedge \text{tree } t\ \&x) * [\![P]\!]^I * (b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)\}$

for $B = B_1 \Rightarrow B_2$: writing BX for $(b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)$

$$\dfrac{\{[\![P]\!]^I_{\vec{x}}\}\ [\![B_1]\!]_1\ \{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_1}(\text{BX})\} \qquad \dfrac{\dfrac{\{[\![P]\!]^I_{\vec{x}}\}\ [\![B_2]\!]_1\ \{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_2}(\text{BX})\}}{\{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_1,\notin B_2}(\text{BX})\}\ [\![B_2]\!]_1\ \{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_1,B_2}(\text{BX})\}}\ \textsc{Frame}}{\{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_1}(\text{BX})\}\ [\![B_2]\!]_1\ \{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_1,B_2}(\text{BX})\}}\ \textsc{Cons}}{\{[\![P]\!]^I_{\vec{x}}\}\ [\![B_1 \Rightarrow B_2]\!]_1\ \{[\![P]\!]^I_{\vec{x}} * \prod_{(X=X')\in B_1 \Rightarrow B_2}(\text{BX})\}}\ \textsc{Seq}$$

Then we show that for $P = B$:

$$([\![B]\!]^I_{\vec{x}} * \textstyle\prod_{(X=X')\in B}(b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)) \quad\Rightarrow\quad [\![B]\!]_2$$

from which the result follows directly. We do this by first putting B into disjunctive normal form (which is easily shown not to affect the semantics of the translation). Then:

for $B =$ false: $[\![\text{false}]\!]^I_{\vec{x}} \Rightarrow \text{false}$

for $B = N{=}N'$: $[\![N = N']\!]^I_{\vec{x}} \Rightarrow (N = N')$

for $B = X{=}X'$: $([\![X = X']\!]^I_{\vec{x}} * (b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)) \Rightarrow (b_{(X=X')} = 1)$

for $B = X{\neq}X'$: $([\![X \neq X']\!]^I_{\vec{x}} * (b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)) \Rightarrow (b_{(X=X')} = 0)$

Frame Rule

$$\left\{\exists i,j.\,\Cap^{(l,i,u,j,r)} * [\![P]\!]_{\vec{x}}^{(l,i,u,j,r)}\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists i,j.\,\Cap^{(l,i,u,j,r)} * [\![Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\right\}$$

CONS

$$\left\{\exists \vec{x},i,j.\,\Cap^{(l,i,u,j,r)} * \textstyle\prod(\cdots) * [\![P]\!]^{(l,i,u,j,r)}\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists \vec{x},i,j.\,\Cap^{(l,i,u,j,r)} * \textstyle\prod(\cdots) * [\![Q]\!]^{(l,i,u,j,r)}\right\}$$

*

$$\left\{\exists \vec{x}_{\mathrm{mod}},i,j.\,\Cap^{(l,i,u,j,r)} * \textstyle\prod(\cdots) * [\![P]\!]^{(l,i,u,j,r)}\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists \vec{x}_{\mathrm{mod}},i,j.\,\Cap^{(l,i,u,j,r)} * \textstyle\prod(\cdots) * [\![Q]\!]^{(l,i,u,j,r)}\right\}$$

FRAME

$$\left\{\begin{array}{l}(\forall i,j.\,\Cap^{(l,i,u,j,r)} \mathrel{-\!\!*} \exists i',j'.\,\Cap^{(l',i',u',j',r')} * [\![K]\!]_{(l,i,u,j,r)}^{(l',i',u',j',r')})\\ \cdot\,(\exists \vec{x}_{\mathrm{mod}},i,j.\,\Cap^{(l,i,u,j,r)} * \prod(\cdots) * [\![P]\!]^{(l,i,u,j,r)})\end{array}\right\}\ [\![\mathbb{C}]\!]\ \left\{\begin{array}{l}(\forall i,j.\,\Cap^{(l,i,u,j,r)} \mathrel{-\!\!*} \exists i',j'.\,\Cap^{(l',i',u',j',r')} * [\![K]\!]_{(l,i,u,j,r)}^{(l',i',u',j',r')})\\ \cdot\,(\exists \vec{x}_{\mathrm{mod}},i,j.\,\Cap^{(l,i,u,j,r)} * \prod(\cdots) * [\![Q]\!]^{(l,i,u,j,r)})\end{array}\right\}$$

CONS

$$\left\{\begin{array}{l}\exists \vec{x}_{\mathrm{mod}},i,j,i',j'.\,\Cap^{(l',i',u',j',r')} * \prod(\cdots)\\ * [\![K]\!]_{(l,i,u,j,r)}^{(l',i',u',j',r')} * [\![P]\!]^{(l,i,u,j,r)}\end{array}\right\}\ [\![\mathbb{C}]\!]\ \left\{\begin{array}{l}\exists \vec{x}_{\mathrm{mod}},i,j,i',j'.\,\Cap^{(l',i',u',j',r')} * \prod(\cdots)\\ * [\![K]\!]_{(l,i,u,j,r)}^{(l',i',u',j',r')} * [\![Q]\!]^{(l,i,u,j,r)}\end{array}\right\}$$

CONS

$$\left\{\begin{array}{l}\exists \vec{x}_{\mathrm{mod}},i',j'.\,\Cap^{(l',i',u',j',r')} * \prod(\cdots)\\ * [\![K\cdot P]\!]^{(l',i',u',j',r')}\end{array}\right\}\ [\![\mathbb{C}]\!]\ \left\{\begin{array}{l}\exists \vec{x}_{\mathrm{mod}},i',j'.\,\Cap^{(l',i',u',j',r')} * \prod(\cdots)\\ * [\![K\cdot P]\!]^{(l',i',u',j',r')}\end{array}\right\}$$

ELIM

$$\left\{\exists i',j'.\,\Cap^{(l',i',u',j',r')} * [\![K\cdot P]\!]_{\vec{x}}^{(l',i',u',j',r')}\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists i',j'.\,\Cap^{(l',i',u',j',r')} * [\![K\cdot Q]\!]_{\vec{x}}^{(l',i',u',j',r')}\right\}$$

REN

$$\left\{\exists i,j.\,\Cap^{(l,i,u,j,r)} * [\![K\cdot P]\!]_{\vec{x}}^{(l,i,u,j,r)}\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists i,j.\,\Cap^{(l,i,u,j,r)} * [\![K\cdot Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\right\}$$

* where $\vec{x}_{\mathrm{mod}} = \mathrm{mod}_{\mathcal{T}_\circ}(\mathbb{C})$; see text.

Figure 5. Frame Rule derivation

for $B = B_1 \wedge B_2$: writing BX for $(b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)$,

$$\dfrac{\dfrac{([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow ([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1}(\mathrm{BX})) \Rightarrow [\![B_1]\!]_2 \qquad ([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow ([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1}(\mathrm{BX})) \Rightarrow [\![B_1]\!]_2}{(([\![B_1]\!]_{\vec{x}}^I \wedge [\![B_2]\!]_{\vec{x}}^I) * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow ([\![B_1]\!]_2 \wedge [\![B_2]\!]_2)}}{([\![B_1 \wedge B_2]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow [\![B_1 \wedge B_2]\!]_2}$$

for $B = B_1 \vee B_2$: writing BX for $(b_{(X=X')} \doteq 1 \wedge [\![X{=}X']\!]^I \vee b_{(X=X')} \doteq 0 \wedge [\![X{\neq}X']\!]^I)$,

$$\dfrac{\dfrac{([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow ([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1}(\mathrm{BX})) \Rightarrow [\![B_1]\!]_2 \qquad ([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow ([\![B_1]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1}(\mathrm{BX})) \Rightarrow [\![B_1]\!]_2}{(([\![B_1]\!]_{\vec{x}}^I \vee [\![B_2]\!]_{\vec{x}}^I) * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow ([\![B_1]\!]_2 \vee [\![B_2]\!]_2)}}{([\![B_1 \vee B_2]\!]_{\vec{x}}^I * \prod_{(X=X')\in B_1,B_2}(\mathrm{BX})) \Rightarrow [\![B_1 \vee B_2]\!]_2}$$

Theorem 8.3 (Small Axiom translation). *For every high-level Small Axiom $\{P\}\ \mathbb{C}\ \{Q\}$ in Defn. 3.6, the following low-level Hoare Triple holds and is derivable:*

$$\{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![P]\!]_{\vec{x}}^{(l,i,u,j,r)}\}\ [\![\mathbb{C}]\!]\ \{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\}$$

where $\vec{x} \supseteq \mathrm{free}_{\mathcal{T}_\circ}(\mathbb{C}) \cup \mathrm{free}_{\mathcal{T}_\circ}(P)$.

Proof. These are all standard SL proofs.

Theorem 8.4 (Inference rule translation). *For every inference rule in Defn. **??**, there is a corresponding, derivable low-level rule obtained by replacing all high-level Hoare Triples $\{P\}\ \mathbb{C}\ \{Q\}$ with the low-level Triples $\{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![P]\!]_{\vec{x}}^{(l,i,u,j,r)}\}\ [\![\mathbb{C}]\!]\ \{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\}$. For example, the following translated Frame Rule holds for all tree update commands $\mathbb{C}$, and CL formulæ P and K, where $\mathrm{mod}(\mathbb{C}) \cap \mathrm{free}(K) = \emptyset$:*

$$\frac{\{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![P]\!]_{\vec{x}}^{(l,i,u,j,r)}\}\ [\![\mathbb{C}]\!]\ \{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\}}{\{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![K \cdot P]\!]_{\vec{x}}^{(l,i,u,j,r)}\}\ [\![\mathbb{C}]\!]\ \{\exists i, j.\, ⋒^{(l,i,u,j,r)} * [\![K \cdot Q]\!]_{\vec{x}}^{(l,i,u,j,r)}\}}$$

Proof sketch. Again, these are standard SL proofs. The cases for Consequence, Disjunction, Variable Elimination, Sequencing, If-Then-Else and While-Do are given in Figure **??**, with the translated high-level premises and conclusions marked in bold. The key derivation is that of the Frame Rule, given in Figure 5. This is performed in two parts. The first, marked *, is a general result regarding the auxiliary tree shape variables:

$$\frac{\left\{\exists \vec{x}.\, [\![P]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists \vec{x}.\, [\![Q]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}}{\left\{\exists \vec{x}_{\mathrm{mod}}.\, [\![P]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists \vec{x}_{\mathrm{mod}}.\, [\![Q]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}}$$

where $\vec{x}_{\mathrm{mod}} = \mathrm{mod}_{\mathcal{T}_\circ}(\mathbb{C})$. In other words, the non-modified variables need not be existentially quantified, since their value cannot change. Noting that $x \in \mathrm{mod}_{\mathcal{T}_\circ}(\mathbb{C}) \Leftrightarrow \&x \in \mathrm{mod}([\![\mathbb{C}]\!])$ (and that $x \notin \mathrm{mod}([\![\mathbb{C}]\!])$ since it is a logical variable), the result follows by frame and consequence:

$$\dfrac{\dfrac{\left\{\exists \vec{x}.\, [\![P]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists \vec{x}.\, [\![Q]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}}{\left\{\begin{array}{c}(\exists \vec{x}_{\mathrm{cst}}.\, \prod_{x \in \vec{x}_{\mathrm{cst}}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) \\ -\!\!* \prod_{x \in \vec{x}_{\mathrm{cst}}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)) \cdot \\ (\exists \vec{x}.\, [\![P]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x))\end{array}\right\}\ [\![\mathbb{C}]\!]\ \left\{\begin{array}{c}(\exists \vec{x}_{\mathrm{cst}}.\, \prod_{x \in \vec{x}_{\mathrm{cst}}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x) \\ -\!\!* \prod_{x \in \vec{x}_{\mathrm{cst}}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)) \cdot \\ (\exists \vec{x}.\, [\![Q]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x))\end{array}\right\}}}{\left\{\exists \vec{x}_{\mathrm{mod}}.\, [\![P]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}\ [\![\mathbb{C}]\!]\ \left\{\exists \vec{x}_{\mathrm{mod}}.\, [\![Q]\!]^I * \prod_{x \in \vec{x}} (\exists t.\, \langle t \rangle \equiv x \wedge \mathrm{tree}\ t\ \&x)\right\}}$$

where $\vec{x}_{\mathrm{cst}} = \mathrm{free}_{\mathcal{T}_\circ}(\mathbb{C}) \setminus \mathrm{mod}_{\mathcal{T}_\circ}(\mathbb{C})$. The rest of the Frame Rule derivation, given in Figure 5, uses the SL adjoint $-\!\!*$ to remove the crust around the heap representing P, and $*$ to add to it the context representing K, together with a fresh crust. The use of the Rule of Consequence relies on Lemma A.4, given below.

Lemma A.4 (Crust property). *The crust has the following property:*

$$\forall I, I'.\, \Cap^{I} * \text{subcontext } I\ I' \Rightarrow \text{true} * \Cap^{I'}$$

Proof. Follows by induction on contexts:

$\Cap^{I} * \text{subcontext } -\ I\ I' \Rightarrow \Cap^{I} * (I \doteq I') \Rightarrow \text{true} * \Cap^{I'}$

$\Cap^{(l,i,u,j,r)} * \text{subcontext } \mathsf{n}[\mathsf{c}]\ (l,i,u,j,r)\ I'$
$\Rightarrow \exists d_i, d_j.\, \Cap^{(l,i,u,j,r)} * (i \doteq j \doteq \mathsf{n}) * n \mapsto l, u, r, d_i * \text{subcontext } \mathsf{c}\ (\text{nil},d_i,n,d_j,\text{nil})\ I'$
$\Rightarrow \exists d_i, d_j.\, \text{true} * \Cap^{(\text{nil},d_i,n,d_j,\text{nil})} * \text{subcontext } \mathsf{c}\ (\text{nil},d_i,n,d_j,\text{nil})\ I'$
$\Rightarrow \text{true} * \Cap^{I'}$ (by induction)

$\Cap^{(l,i,u,j,r)} * \text{subcontext } \mathsf{c}|\mathsf{t}\ (l,i,u,j,r)\ I'$
$\Rightarrow \exists k_l, k_r.\, \Cap^{(l,i,u,j,r)} * \text{subcontext } \mathsf{c}\ (l,i,u,k_l,k_r)\ I' * \text{subtree } \mathsf{t}\ (k_l,k_r,u,j,r)$
$\Rightarrow \exists k_l, k_r.\, \Cap^{(l,i,u,j,r)} * \text{subcontext } \mathsf{c}\ (l,i,u,k_l,k_r)\ I' * \text{true} * (k_r = \text{nil} \vee \exists r_k, d_k.\, k_r \mapsto k_l, u, r_k, d_k)$
$\Rightarrow \exists k_l, k_r.\, \Cap^{(l,i,u,k_l,k_r)} * \text{subcontext } \mathsf{c}\ (l,i,u,k_l,k_r)\ I' * \text{true}$
$\Rightarrow \text{true} * \Cap^{I'}$ (by induction)

$\Cap^{(l,i,u,j,r)} * \text{subcontext } \mathsf{t}|\mathsf{c}\ (l,i,u,j,r)\ I'$
$\Rightarrow \exists k_l, k_r.\, \Cap^{(l,i,u,j,r)} * \text{subtree } \mathsf{t}\ (l,i,u,k_l,k_r) * \text{subcontext } \mathsf{c}\ (k_l,k_r,u,j,r)\ I'$
$\Rightarrow \exists k_l, k_r.\, \Cap^{(l,i,u,j,r)} * \text{true} * (k_l = \text{nil} \vee \exists l_k, d_k.\, k_l \mapsto l_k, u, k_r, d_k) * \text{subcontext } \mathsf{c}\ (k_l,k_r,u,j,r)\ I'$
$\Rightarrow \exists k_l, k_r.\, \Cap^{(k_l,k_r,u,j,r)} * \text{true} * \text{subcontext } \mathsf{c}\ (k_l,k_r,u,j,r)\ I'$
$\Rightarrow \text{true} * \Cap^{I'}$ (by induction)