# Locality Refinement

Thomas Dinsdale-Young, Philippa Gardner, and Mark Wheelhouse

Imperial College London
{td202, pg, mjw03}@ic.ac.uk

**Abstract.** We study refinement in the setting of local reasoning. In particular, we explore general translations that preserve and that break locality.

## 1 Introduction

Program refinement is the verifiable transformation of an abstract (high-level) formal specification into a concrete (low-level) executable program. We study program refinement in the setting of local reasoning.

The principle of local reasoning is that if we know how local computation behaves on some state then we can infer its behavior if the state is extended: it simply leaves the additional state unchanged. On this principle, O'Hearn and Reynolds founded separation logic [10], which achieved remarkable success at local reasoning about C-style heap update in a Hoare logic framework. Generalising separation logic techniques to more abstract state models, Calcagno, Gardner and Zarfaty developed context logic [1], which has been successfully applied to reasoning about the W3C DOM tree update library [7].

Previously, where context logic has been applied to reasoning about programs that manipulate abstract state such as trees, sequences and terms, the reasoning has been justified using that same abstract state, by proving soundness with respect to an operational semantics. In this paper, we instead look at justifying such reasoning in terms of *implementations* of the abstract state. This is an instance of the classic problem of data refinement [9, 3], but with the added twist that our emphasis is on local reasoning.

In this paper, our motivating example is the stepwise refinement of a module that provides local commands for manipulating a tree structure, as illustrated in Fig. 1. We show how this tree module $\mathbb{T}$ may be implemented using the familiar separation logic heap module $\mathbb{H}$ and an abstract list module $\mathbb{L}$. We then show how this list module $\mathbb{L}$ can be implemented in terms of the heap $\mathbb{H}$. Our development provides two general techniques for verifying local modules with respect to their implementations, which we term *locality-preserving* and *locality-breaking* translations.

With the first technique, locality at the abstract level is, broadly speaking, implemented by locality at the lower level. However, typically implementations operate on a larger state than the abstract footprint, for instance, by performing pointer surgery on the surrounding state. We introduce the notion of *crust* to

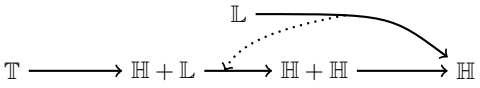


**Fig. 1.** Translations presented in this paper

capture this additional state. This crust intrudes on the context, and so breaks the disjointness that exists at the high level. We then relate the high-level locality with low-level locality through a *fiction of disjointness*.

With the second technique, locality at the abstract level is not preserved by the translation. Although it is possible to think about such a translation using a (large) crust, we instead prove soundness using a locality-breaking translation. We establish a *fiction of locality* at the high-level, by demonstrating that the translation preserves the axioms in any high-level context.

While we attempt to give the intuition behind our techniques, we omit the full proofs of our results, which may be found in the full version of this paper [5].

**Related Work** There as been much work on abstraction and information hiding in separation logic. In particular, the work of Parkinson and Bierman on abstract predicates [11] addresses the problem of abstraction in a separation logic setting. An abstract predicate is, to the client, an opaque object that encapsulates the unknown representation of an abstract datatype. In their approach, abstract predicates inherit some of the benefits of locality from separation logic: an operation on one abstract predicate leaves others alone. However, it does not permit local reasoning within the structure represented by the abstract predicate, which this paper addresses. Filipović *et al.* have also considered data refinement with separation logic [6] showing how to handle aliasing issues in the refinement setting. Their work has a similar theme to ours, choosing only to verify client programs that use module commands correctly with regards to the specification provided by the module. We differ in that we choose to focus on translations between different levels of abstraction.

## 2 Preliminaries

### 2.1 State Models

We work with multiple data structures at multiple levels of abstraction. To handle these structures in a uniform way, we model our program states using context algebras. We will see that many standard state models fit this pattern.

**Definition 1 (Context Algebra).** *A* context algebra $\mathcal{A} = (\mathcal{C}, \mathcal{D}, \bullet, \circ, \mathbf{I}, \mathbf{0})$ *comprises:*

- *a non-empty set of abstract states,* $\mathcal{D}$*;*
- *a non-empty set of state contexts,* $\mathcal{C}$*;*
- *a partially-defined associative context composition function,* $\bullet : \mathcal{C} \times \mathcal{C} \rightharpoonup \mathcal{C}$*;*

- *a partially-defined context application function,* $\circ : \mathcal{C} \times \mathcal{D} \rightharpoonup \mathcal{D}$*, with* $c_1 \circ (c_2 \circ d) = (c_1 \bullet c_2) \circ d$ *(undefined terms are considered equal);*
- *a distinguished set of identity contexts,* $\mathbf{I} \subseteq \mathcal{C}$*; and*
- *a distinguished set of empty states,* $\mathbf{0} \subseteq \mathcal{D}$*;*

*having the following properties: for all* $c \in \mathcal{C}$*,* $d \in \mathcal{D}$*, and* $i' \in \mathbf{I}$

- $i \circ d$ *is defined for some* $i \in \mathbf{I}$*, and whenever* $i' \circ d$ *is defined,* $i' \circ d = d$*;*
- *the relation* $\{(c, d) | \exists o \in \mathbf{0}.\, c \circ o = d\}$ *is a total surjective function;*
- $i \bullet c$ *is defined for some* $i \in \mathbf{I}$*, and whenever* $i' \bullet c$ *is defined,* $i' \bullet c = c$*;*
- $c \bullet i$ *is defined for some* $i \in \mathbf{I}$*, and whenever* $c \bullet i'$ *is defined,* $c \bullet i' = c$*.*

*Example 1.* The following are examples of context algebras:

(a) *Heaps* $h \in \mathrm{H}$ are defined as:

$$h ::= \mathrm{emp} \mid a \mapsto v \mid h * h$$

where $a \in \mathbb{N}^+$ ranges over unique *heap addresses*, $v \in \mathrm{Val}$ ranges over *values*, and $*$ is associative and commutative with identity emp. (Heaps are thus finite partial functions from addresses to values.) Heaps form the *heap context algebra*, $\mathcal{H} = (\mathrm{H}, \mathrm{H}, *, *, \{\mathrm{emp}\}, \{\mathrm{emp}\})$. All separation algebras [2] can be viewed as context algebras in this way.

(b) *Variable stores* $\sigma \in \Sigma$ are defined as:

$$\sigma ::= \mathrm{emp} \mid \boldsymbol{x} \Rrightarrow v \mid \sigma * \sigma$$

where $\boldsymbol{x} \in \mathrm{Var}$ ranges over unique *program variables*, $v \in \mathrm{Val}$ ranges over values, and $*$ is associative and commutative with identity emp. Variable stores form the *variable store context algebra*, $\mathcal{V} = (\Sigma, \Sigma, *, *, \{\mathrm{emp}\}, \{\mathrm{emp}\})$.

(c) *Trees* $t \in \mathrm{T}$ and *tree contexts* $c \in \mathrm{C}$ are defined as follows:

$$t ::= \varnothing \mid n[t] \mid t \otimes t$$
$$c ::= \_ \mid n[c] \mid t \otimes c \mid c \otimes t$$

where $n \in \mathrm{N}$ ranges over unique *node identifiers*, and $\otimes$ is associative with identity $\varnothing$. The context composition and application are standard (substituting a tree or context in the hole). Trees and tree contexts form the *Tree context algebra*, $\mathcal{T} = (\mathrm{C}, \mathrm{T}, \bullet, \circ, \{\_\}, \{\varnothing\})$.

(d) Given context algebras, $\mathcal{A}_1$ and $\mathcal{A}_2$, their product $\mathcal{A}_1 \times \mathcal{A}_2$ (defined in the natural fashion) is also a context algebra. For example, $\mathcal{H} \times \mathcal{V}$ and $\mathcal{T} \times \mathcal{V}$ describe states consisting of trees or heaps, and variables stores.

### 2.2 Predicates

Predicates are either sets of abstract states (denoted $p, q$) or sets of state contexts (denoted $f, g$). We do not fix a particular assertion language, although we do use standard logical notation for conjunction, disjunction, negation and quantification. We lift operations on states and contexts to predicates: for instance, $x \mapsto v$ denotes the predicate $\{x \mapsto v\}$; $\exists v.\, x \mapsto v$ denotes $\{x \mapsto v \mid v \in \mathrm{Val}\}$; the separating application $f \circ p$ denotes $\{c \circ d \mid c \in f \wedge d \in p\}$; and so on. We also use the notation $\prod^*$ to denote iterated $*$. We use set-theoretic notation for predicate membership and containment.

### 2.3 Language Syntax

**Definition 2 (Programming Language).** *Given a set of basic commands $\varphi \in \Phi$, the language $\mathcal{L}_\Phi$ is defined by the following grammar:*

$$\begin{aligned}\mathbb{C} ::= &\texttt{skip} \mid \varphi \mid \boldsymbol{x} := \boldsymbol{E} \mid \mathbb{C}; \mathbb{C} \mid \texttt{if } \boldsymbol{B} \texttt{ then } \mathbb{C} \texttt{ else } \mathbb{C} \mid \texttt{while } \boldsymbol{B} \texttt{ do } \mathbb{C} \mid \\ &\texttt{procs } \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_1} := \texttt{f}_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_1})\{\mathbb{C}\} \cdots \texttt{ in } \mathbb{C} \mid \\ &\texttt{call } \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_k} := \texttt{f}(\boldsymbol{E}_1, ..., \boldsymbol{E}_{n_k}) \mid \texttt{local } \boldsymbol{x} \texttt{ in } \mathbb{C}\end{aligned}$$

*where $\boldsymbol{x}, \boldsymbol{r}, \ldots \in \mathrm{Var}$ range over program variables, $\boldsymbol{E}, \boldsymbol{E}_1, \ldots \in \mathrm{Exp}_{\mathrm{Val}}$ range over value expressions, $\boldsymbol{B} \in \mathrm{Exp}_{\mathrm{Bool}}$ ranges over boolean expressions, and $\texttt{f}, \texttt{f}_1, \ldots \in$* PName *range over procedure names.*

### 2.4 Axiomatic Semantics

We give the semantics of the language $\mathcal{L}_\Phi$ as a program logic based on local Hoare reasoning. The state model, $\mathcal{A} \times \mathcal{V}$, combines two context algebras: the variable store context algebra, $\mathcal{V}$, used to interpret program variables; and the context algebra, $\mathcal{A}$, manipulated only by the commands of $\Phi$. A set of axioms $\mathrm{Ax} \subseteq \mathcal{P}(\mathcal{D}_\mathcal{A} \times \Sigma) \times \Phi \times \mathcal{P}(\mathcal{D}_\mathcal{A} \times \Sigma)$ provides the semantics for the commands of $\Phi$, where $\mathcal{D}_\mathcal{A}$ is the set of abstract states from $\mathcal{A}$ and $\Sigma$ is the set of variable stores from $\mathcal{V}$.

The judgements of our proof system have the form $\Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}$, where $p, q \in \mathcal{P}(\mathcal{D}_\mathcal{A} \times \Sigma)$ are predicates, $\mathbb{C} \in \mathcal{L}_\Phi$ is a program and $\Gamma$ is a procedure specification environment. A procedure specification environment associates procedure names with pairs of pre- and postconditions (parameterised by the argument and return values of the procedure respectively). The interpretation of judgements is that, in the presence of procedures satisfying $\Gamma$, when executed from a state satisfying $p$, the program $\mathbb{C}$ will either diverge or terminate in a state satisfying $q$.

The proof rules of the program logic are given in Fig. 2. The semantics of value expressions $[\![\boldsymbol{E}]\!]_\sigma$ is the value of $\boldsymbol{E}$ in variable store $\sigma$. The variable store $\rho$ denotes an arbitrary variable store that evaluates all of the program variables that are read but not written in each command under consideration. We write $\mathrm{vars}(\rho)$ and $\mathrm{vars}(\boldsymbol{E})$ to denote the variables in $\rho$ and $\boldsymbol{E}$ respectively.

The Frame rule is the natural frame rule for context algebras. The rules Assgn, Local, PDef and PCall are standard, written in a slightly non-standard way since we are working with context algebras together with the variable store context algebra. Since we are treating variables as resource, the Assgn rule not only requires the resource $\boldsymbol{x} \Rightarrow v$, but also the resource $\rho$ containing the other variables used in $\boldsymbol{E}$. For the Local rule, recall that the predicate $p$ specifies a set of pairs consisting of resource from $\mathcal{D}_\mathcal{A}$ and variable resource. The predicate $(\mathbf{I}_\mathcal{A} \times \boldsymbol{x} \Rightarrow -) \circ p$ therefore specifies that the resource from $\mathcal{D}_\mathcal{A}$ stays the same and, since $(\mathbf{I}_\mathcal{A} \times \boldsymbol{x} \Rightarrow -) \circ p \neq \emptyset$, that the variable store has been increased by $\boldsymbol{x} \Rightarrow -$. For the PDef and PCall rules, the procedures $\texttt{f}$ have parametrized predicates $P = \lambda \overrightarrow{x}.p$ as the precondition and $Q = \lambda \overrightarrow{r}.q$ as

$$\frac{\Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}{\Gamma \vdash \{f \circ p\}\ \mathbb{C}\ \{f \circ q\}}\ \textsc{Frame} \qquad \frac{(p, \varphi, q) \in \mathrm{Ax}}{\Gamma \vdash \{p\}\ \varphi\ \{q\}}\ \textsc{Axiom}$$

$$\frac{\mathrm{vars}(\rho) = \mathrm{vars}(E) - \{x\}}{\Gamma \vdash \{\mathbf{0}_{\mathcal{A}} \times (\boldsymbol{x} \Rrightarrow v * \rho)\}\ \boldsymbol{x} := E\ \left\{\mathbf{0}_{\mathcal{A}} \times (\boldsymbol{x} \Rrightarrow [\![E]\!]_{(\boldsymbol{x} \Rightarrow v * \rho)} * \rho)\right\}}\ \textsc{Assgn}$$

$$\frac{\Gamma \vdash \{(\mathbf{I}_{\mathcal{A}} \times \boldsymbol{x} \Rrightarrow -) \circ p\}\ \mathbb{C}\ \{(\mathbf{I}_{\mathcal{A}} \times \boldsymbol{x} \Rrightarrow -) \circ q\} \quad (\mathbf{I}_{\mathcal{A}} \times \boldsymbol{x} \Rrightarrow -) \circ p \neq \emptyset}{\Gamma \vdash \{p\}\ \texttt{local}\ \boldsymbol{x}\ \texttt{in}\ \mathbb{C}\ \{q\}}\ \textsc{Local}$$

$$\frac{\forall (\mathtt{f}_i : P \to Q) \in \Gamma.\ \Gamma', \Gamma \vdash \begin{array}{c}\{\exists \overrightarrow{v}.\, P(\overrightarrow{v}) \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow \overrightarrow{v} * \overrightarrow{\boldsymbol{r}_i} \Rrightarrow -)\} \\ \mathbb{C}_i \\ \{\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow - * \overrightarrow{\boldsymbol{r}_i} \Rrightarrow \overrightarrow{w})\}\end{array} \qquad \Gamma', \Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}{\Gamma' \vdash \{p\}\ \texttt{procs}\ \overrightarrow{r_1} := \mathtt{f}_1(\overrightarrow{\boldsymbol{x}_1})\{\mathbb{C}_1\}, \ldots, \overrightarrow{r_k} := \mathtt{f}_k(\overrightarrow{\boldsymbol{x}_k})\{\mathbb{C}_k\}\ \texttt{in}\ \mathbb{C}\ \{q\}}\ \textsc{PDef}$$

$$\frac{\mathrm{vars}(\rho) = \mathrm{vars}(E) - \{\overrightarrow{r}\}}{\Gamma, (\mathtt{f} : P \to Q) \vdash \begin{array}{c}\left\{P([\![\overrightarrow{E}]\!]_{(\overrightarrow{r} \Rrightarrow \overrightarrow{v} * \rho)}) \times (\overrightarrow{r} \Rrightarrow \overrightarrow{v} * \rho)\right\} \\ \texttt{call}\ \overrightarrow{r} := \mathtt{f}(\overrightarrow{E}) \\ \{\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{r} \Rrightarrow \overrightarrow{w} * \rho)\}\end{array}}\ \textsc{PCall}$$

**Fig. 2.** Some Local Hoare Logic rules for $\mathcal{L}_{\Phi}$.

the postcondition, with $P(\overrightarrow{v}) = p[\overrightarrow{v}/\overrightarrow{x}]$ and $Q(\overrightarrow{w}) = q[\overrightarrow{w}/\overrightarrow{r}]$. We omit the Cons, Disj, Skip, Seq, If and While rules which are standard. For all of our examples, the conjunction rule is admissible; in general, this is not the case.

## 3 Abstract Modules

The language given in §2 and its semantics are parameterised by a context algebra, a set of commands and a set of axioms. Together, these parameters constitute an abstract description of a module.

**Definition 3 (Abstract Module).** *An* abstract module $\mathbb{A} = (\mathcal{A}_{\mathbb{A}}, \Phi_{\mathbb{A}}, \mathrm{Ax}_{\mathbb{A}})$ *consists of a context algebra* $\mathcal{A}_{\mathbb{A}}$ *with abstract state set* $\mathcal{D}_{\mathbb{A}}$*, a set of commands* $\Phi_{\mathbb{A}}$ *and a set of axioms* $\mathrm{Ax}_{\mathbb{A}} \subseteq \mathcal{P}(\mathcal{D}_{\mathbb{A}} \times \Sigma) \times \Phi_{\mathbb{A}} \times \mathcal{P}(\mathcal{D}_{\mathbb{A}} \times \Sigma)$.

*Notation.* We write $\mathcal{L}_{\mathbb{A}}$ for the language $\mathcal{L}_{\Phi_{\mathbb{A}}}$. We write $\vdash_{\mathbb{A}}$ for the proof judgement determined by the abstract module. When $\mathbb{A}$ can be inferred from context, we may simply write $\vdash$ instead of $\vdash_{\mathbb{A}}$.

### 3.1 Heap Module

The first and most familiar abstract module we consider is the abstract heap module, $\mathbb{H} = (\mathcal{H}, \Phi_{\mathbb{H}}, \mathrm{Ax}_{\mathbb{H}})$, which extends the core language with standard heap-update commands. The context algebra $\mathcal{H}$ was defined in Example 1. We give the heap update commands in Definition 4, and the axioms for describing the behavior of these commands in Definition 5.

**Definition 4 (Heap Update Commands).** *The set of heap update commands* $\Phi_{\mathbb{H}}$ *comprises: allocation,* $\boldsymbol{n} := \texttt{alloc}(E)$*; disposal,* $\texttt{dispose}(E, E')$*; mutation,* $[E] := E'$*; and lookup* $\boldsymbol{n} := [E]$.

**Definition 5 (Heap Axioms).** *The set of heap axioms* $\text{Ax}_{\mathbb{H}}$ *comprises:*

$$\left\{\begin{array}{l}\text{emp} \times \boldsymbol{n} \Rightarrow v * \rho \\ \wedge\, [\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow v} \geq 1\end{array}\right\} \quad \boldsymbol{n} := \texttt{alloc}(E) \quad \left\{\begin{array}{l}\exists x.\, x \mapsto - * \ldots \\ * \, x + [\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow v} \mapsto - \\ \times \boldsymbol{n} \Rightarrow x * \rho\end{array}\right\}$$

$$\left\{\begin{array}{l}[\![E]\!]_\rho \mapsto - * \ldots \\ * \, [\![E]\!]_\rho + [\![E']\!]_\rho \mapsto - \times \rho\end{array}\right\} \quad \texttt{dispose}(E, E') \quad \{\text{emp} \times \rho\}$$

$$\{[\![E]\!]_\rho \mapsto - \times \rho\} \quad [E] := E' \quad \{[\![E]\!]_\rho \mapsto [\![E']\!]_\rho \times \rho\}$$

$$\{[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow v} \mapsto x \times \boldsymbol{n} \Rightarrow v * \rho\} \quad \boldsymbol{n} := [E] \quad \{[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow v} \mapsto x \times \boldsymbol{n} \Rightarrow x * \rho\}$$

### 3.2 Tree Module

Another familiar abstract module that we consider is the abstract tree module, $\mathbb{T} = (\mathcal{T}, \Phi_{\mathbb{T}}, \text{Ax}_{\mathbb{T}})$, which extends the core language with tree update commands acting on a single tree, similar to a document in DOM. The tree context algebra $\mathcal{T}$ was defined in Example 1. We give the the tree update commands in Definition 6 and their corresponding axioms in Definition 7.

**Definition 6 (Tree Update Commands).** *The set of tree update commands* $\Phi_{\mathbb{T}}$ *comprises: relative traversal,* `getUp`*,* `getLeft`*,* `getRight`*,* `getFirst`*,* `getLast`*; node creation,* `newNodeAfter`*; and subtree deletion* `deleteTree`*.*

**Definition 7 (Tree Axioms).** *The set of tree update axioms* $\text{Ax}_{\mathbb{T}}$ *includes:*

$$\left\{\begin{array}{l}[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[t] \otimes m[t'] \\ \times \boldsymbol{n} \Rightarrow n * \rho\end{array}\right\} \boldsymbol{n} := \texttt{getRight}(E) \left\{\begin{array}{l}[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[t] \otimes m[t'] \\ \times \boldsymbol{n} \Rightarrow m * \rho\end{array}\right\}$$

$$\left\{\begin{array}{l}m[t' \otimes [\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[t]] \\ \times \boldsymbol{n} \Rightarrow n * \rho\end{array}\right\} \boldsymbol{n} := \texttt{getRight}(E) \left\{\begin{array}{l}m[t' \otimes [\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[t]] \\ \times \boldsymbol{n} \Rightarrow \textbf{\textit{null}} * \rho\end{array}\right\}$$

$$\left\{\begin{array}{l}[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[t' \otimes m[t]] \\ \times \boldsymbol{n} \Rightarrow n * \rho\end{array}\right\} \boldsymbol{n} := \texttt{getLast}(E) \left\{\begin{array}{l}[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[t' \otimes m[t]] \\ \times \boldsymbol{n} \Rightarrow m * \rho\end{array}\right\}$$

$$\left\{\begin{array}{l}[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[\varnothing] \\ \times \boldsymbol{n} \Rightarrow n * \rho\end{array}\right\} \boldsymbol{n} := \texttt{getLast}(E) \left\{\begin{array}{l}[\![E]\!]_{\rho * \boldsymbol{n} \Rightarrow n}[\varnothing] \\ \times \boldsymbol{n} \Rightarrow \textbf{\textit{null}} * \rho\end{array}\right\}$$

$$\{[\![E]\!]_\rho[t] \times \rho\} \ \texttt{newNodeAfter}(E) \ \{\exists m.\, [\![E]\!]_\rho[t] \otimes m[\varnothing] \times \rho\}$$

$$\{[\![E]\!]_\rho[t] \times \rho\} \quad \texttt{deleteTree}(E) \quad \{\varnothing \times \rho\}$$

*The omitted axioms are analogous to those given above.*

### 3.3 List Module

We will study an implementation of the tree module using lists of unique addresses. We therefore define an abstract module for manipulating lists whose elements are unique, $\mathbb{L} = (\mathcal{L}, \Phi_{\mathbb{L}}, \text{Ax}_{\mathbb{L}})$. The list context algebra $\mathcal{L}$ is given in

Definition 10. The list update commands are given in Definition 11 and their corresponding axioms are given in Definition 12.

Superficially, our abstract list stores resemble heaps, in the sense that we have multiple lists each with unique addresses. For example, the list store $(i \Mapsto v_1 + v_2 + v_3) * (j \Mapsto w_1 + v_1)$ consists of two separate lists, at different addresses $i$ and $j$. We however treat the individual lists abstractly. For example, the same list store can be written $(i \Mapsto v_1 + - + v_3) \circ (i \Mapsto v_2 * j \Mapsto w_1 + v_1)$ where, this time, list context $i \Mapsto v_1 + - + v_3$ is separate from the two lists $i \Mapsto v_2 * j \Mapsto w_1 + v_1$.

We sometimes need to represent completed lists: that is, lists that cannot be extended. For example, the command `getHead` requires a complete list to be able to determine accurately the first element in the list. This is indicated by surrounding the list in square brackets, as in $j \Mapsto [w_1 + v_1]$. Completed lists may be separated into a context and sublist, as in $j \Mapsto [w_1 + -] \circ j \Mapsto v_1$, but not extended: $j \Mapsto w_1 + - \circ j \Mapsto [v_1]$ is undefined.

**Definition 8 (List Stores and Contexts).** Lists $l \in \mathrm{L}$, list contexts $lc \in \textsc{Lc}$, list stores $ls \in \textsc{Ls}$, *and* list store contexts $lsc \in \textsc{Lsc}$ *are defined by:*

$$\begin{array}{rlrl} l & ::= \varepsilon \mid v \mid l + l & \qquad ls & ::= \mathrm{emp} \mid i \Mapsto l \mid i \Mapsto [l] \mid ls * ls \\ lc & ::= - \mid lc + l \mid l + lc & \qquad lsc & ::= ls \mid i \Mapsto lc \mid i \Mapsto [lc] \mid lsc * lsc \end{array}$$

*where* $v \in \mathrm{Val}$ *ranges over values, which are taken to occur uniquely in each list or list context,* $i \in \textsc{Laddr}$ *ranges over list addresses, which are taken to occur uniquely in each list store or list store context,* $+$ *is taken to be associative with identity* $\varepsilon$*, and* $*$ *is taken to be associative and commutative with identity* emp.

**Definition 9 (Application and Composition).** *The application of list store contexts to list stores* $\circ : \textsc{Lsc} \times \textsc{Ls} \rightharpoonup \textsc{Ls}$ *is defined inductivelyby:*

$$\begin{array}{rl} \mathrm{emp} \circ ls & = ls \\ (lsc * i \Mapsto l) \circ ls & = (lsc \circ ls) * i \Mapsto l \\ (lsc * i \Mapsto [l]) \circ ls & = (lsc \circ ls) * i \Mapsto [l] \\ (lsc * i \Mapsto lc) \circ (ls * i \Mapsto l) & = (lsc \circ ls) * i \Mapsto lc_{[l/\_]} \\ (lsc * i \Mapsto [lc]) \circ (ls * i \Mapsto l) & = (lsc \circ ls) * i \Mapsto [lc_{[l/\_]}] \end{array}$$

*where* $lc_{[l/\_]}$ *denotes the stand replacement of the hole in* $lc$ *by* $l$*. The result of the application is undefined, when either the right-hand side is badly formed or no case applies. The composition* $\bullet : \textsc{Lsc} \times \textsc{Lsc} \rightharpoonup \textsc{Lsc}$ *is defined similarly.*

**Definition 10 (List-Store Context Algebra).** *The* list-store context algebra, $\mathcal{L} = (\textsc{Lsc}, \mathrm{Ls}, \bullet, \circ, \{\mathrm{emp}\}, \{\mathrm{emp}\})$ *is given by the above definitions.*

**Definition 11 (List Update Commands).** *The set of list commands* $\Phi_{\mathbb{L}}$ *comprises: lookup,* `getHead`*,* `getTail`*,* `getNext`*,* `getPrev`*; stack-style access,* `pop`*,* `push`*; value removal and insertion,* `remove`*,* `insert`*; and construction and destruction,* `newList`*,* `deleteList`*.*

**Definition 12 (List Axioms).** *The set of list axioms* $\mathrm{Ax}_{\mathbb{L}}$ *includes the following small axioms: (the omitted axioms are analogous)*

$$\left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\,v'+l\,]\\ \times\ \boldsymbol{v}\Rightarrow v*\rho\end{array}\right\}\ \boldsymbol{v}:=E.\mathtt{getHead}()\ \left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\,v'+l\,]\\ \times\ \boldsymbol{v}\Rightarrow v'*\rho\end{array}\right\}$$

$$\left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\,\varepsilon\,]\\ \times\ \boldsymbol{v}\Rightarrow v*\rho\end{array}\right\}\ \boldsymbol{v}:=E.\mathtt{getHead}()\ \left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\,\varepsilon\,]\\ \times\ \boldsymbol{v}\Rightarrow \boldsymbol{null}*\rho\end{array}\right\}$$

$$\left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\![E']\!]_{\rho*v\Rightarrow v}+u\\ \times\ \boldsymbol{v}\Rightarrow v*\rho\end{array}\right\}\ \boldsymbol{v}:=E.\mathtt{getNext}(E')\ \left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\![E']\!]_{\rho*v\Rightarrow v}+u\\ \times\ \boldsymbol{v}\Rightarrow u*\rho\end{array}\right\}$$

$$\left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\,l+[\![E']\!]_{\rho*v\Rightarrow v}\,]\\ \times\ \boldsymbol{v}\Rightarrow v*\rho\end{array}\right\}\ \boldsymbol{v}:=E.\mathtt{getNext}(E')\ \left\{\begin{array}{l}[\![E]\!]_{\rho*v\Rightarrow v}\mapsto[\,l+[\![E']\!]_{\rho*v\Rightarrow v}\,]\\ \times\ \boldsymbol{v}\Rightarrow \boldsymbol{null}*\rho\end{array}\right\}$$

$$\{[\![E]\!]_\rho\mapsto[\,l\,]\times\rho\wedge([\![E']\!]_\rho\notin l)\}\quad E.\mathtt{push}(E')\quad\{[\![E]\!]_\rho\mapsto[\,[\![E']\!]_\rho+l\,]\times\rho\}$$

$$\{[\![E]\!]_\rho\mapsto[\![E']\!]_\rho\times\rho\}\quad E.\mathtt{remove}(E')\quad\{[\![E]\!]_\rho\mapsto\varepsilon\times\rho\}$$

$$\left\{\begin{array}{l}[\![E]\!]_\rho\mapsto[\,l+[\![E']\!]_\rho+l'\,]\times\rho\\ \wedge\,([\![E'']\!]_\rho\notin l+[\![E']\!]_\rho+l')\end{array}\right\}\quad E.\mathtt{insert}(E',E'')\quad\{[\![E]\!]_\rho\mapsto[\,l+[\![E']\!]_\rho+[\![E'']\!]_\rho+l'\,]\times\rho\}$$

$$\{\emptyset\times\boldsymbol{i}\Rightarrow i\}\quad\boldsymbol{i}:=\mathtt{newList}()\quad\{\exists j.\,j\mapsto[\,\varepsilon\,]\times\boldsymbol{i}\Rightarrow j\}$$

$$\{[\![E]\!]_\rho\mapsto[\,l\,]\times\rho\}\quad E.\mathtt{deleteList}()\quad\{\emptyset\times\rho\}$$

### 3.4 Combining Abstract Modules

We wish to combine abstract modules in a natural way, that enables programs to be written that intermix commands from different modules.

**Definition 13 (Abstract Module Combination).** *Given abstract modules* $\mathbb{A}_1=(\mathcal{A}_{\mathbb{A}_1},\Phi_{\mathbb{A}_1},\mathrm{Ax}_{\mathbb{A}_1})$ *and* $\mathbb{A}_2=(\mathcal{A}_{\mathbb{A}_2},\Phi_{\mathbb{A}_2},\mathrm{Ax}_{\mathbb{A}_2})$*, their combination* $\mathbb{A}_1+\mathbb{A}_2=(\mathcal{A}_{\mathbb{A}_1}\times\mathcal{A}_{\mathbb{A}_2},\Phi_{\mathbb{A}_1}\oplus\Phi_{\mathbb{A}_2},\mathrm{Ax}_{\mathbb{A}_1}+\mathrm{Ax}_{\mathbb{A}_2})$ *is defined by:*

- $\mathcal{A}_{\mathbb{A}_1}\times\mathcal{A}_{\mathbb{A}_2}$ *is the product of context algebras;*
- $\Phi_{\mathbb{A}_1}\oplus\Phi_{\mathbb{A}_2}=(\Phi_{\mathbb{A}_1}\times\{1\})\cup(\Phi_{\mathbb{A}_2}\times\{2\})$ *is the disjoint union of command sets;*
- $\mathrm{Ax}_{\mathbb{A}_1}+\mathrm{Ax}_{\mathbb{A}_2}$ *is the lifting of the axiom set* $\mathrm{Ax}_{\mathbb{A}_1}$ *(and* $\mathrm{Ax}_{\mathbb{A}_2}$*) to* $\mathrm{Ax}_{\mathbb{A}_1}+\mathrm{Ax}_{\mathbb{A}_2}$ *using the empty states from* $\mathrm{Ax}_{\mathbb{A}_2}$ *(and* $\mathrm{Ax}_{\mathbb{A}_1}$*): formally,* $\mathrm{Ax}_{\mathbb{A}_1}+\mathrm{Ax}_{\mathbb{A}_2}=\{(\pi_1 p,(\varphi,1),\pi_1 q)\mid(p,\varphi,q)\in\mathrm{Ax}_{\mathbb{A}_1}\}\cup\{(\pi_2 p,(\varphi,2),\pi_2 q)\mid(p,\varphi,q)\in\mathrm{Ax}_{\mathbb{A}_1}\}$*, st.* $\pi_1 p=\{(d,o,\sigma)\mid(d,\sigma)\in p,\,o\in\mathbf{0}_2\}$, $\pi_2 p=\{(o,d,\sigma)\mid(d,\sigma)\in p,\,o\in\mathbf{0}_1\}$.

When the command sets $\Phi_{\mathbb{A}_1}$ and $\Phi_{\mathbb{A}_2}$ are disjoint we may drop the tags when referring to the commands in the combined abstract module. When we do use the tags, we indicate them with an appropriately placed subscript.

An example of module combination is $\mathbb{H}+\mathbb{L}$, which we use in §5.1 as the basis for implementing $\mathbb{T}$. The combination comprises both the commands for manipulating lists and for manipulating heaps, and their semantics are not allowed to interfere with each other.

## 4 Module Translations

We define what it means to correctly implement one module in terms of another, using translations which are reminiscent of downward simulations in [8].

**Definition 14 (Sound Module Translation).** *A* module translation $\mathbb{A} \to \mathbb{B}$ *from abstract module* $\mathbb{A}$ *to abstract module* $\mathbb{B}$ *consists of*

- *a* state translation *function* $[\![-]\!] : \mathcal{D}_{\mathbb{A}} \to \mathcal{P}(\mathcal{D}_{\mathbb{B}})$*, and*
- *a* substitutive implementation *function* $[\![-]\!] : \mathcal{L}_{\mathbb{A}} \to \mathcal{L}_{\mathbb{B}}$ *obtained by substituting each basic command of* $\Phi_{\mathbb{A}}$ *with a call to a procedure written in* $\mathcal{L}_{\mathbb{B}}$*.*

*A module translation is* sound *if, for all* $p, q \in \mathcal{P}(\mathcal{D}_{\mathbb{A}} \times \Sigma)$ *and* $\mathbb{C} \in \mathcal{L}_{\mathbb{A}}$*,*

$$\vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\} \quad \Longrightarrow \quad \vdash_{\mathbb{B}} \{[\![p]\!]\}\ [\![\mathbb{C}]\!]\ \{[\![q]\!]\}\,.$$

*where the predicate translation* $[\![-]\!] : \mathcal{P}(\mathcal{D}_{\mathbb{A}} \times \Sigma) \to \mathcal{P}(\mathcal{D}_{\mathbb{B}} \times \Sigma)$ *is the natural lifting of the state translation given by* $[\![p]\!] = \bigvee_{(d,\sigma)\in p} [\![d]\!] \times \sigma$*.*

We will see that sometimes the module structure is preserved by the translations and sometimes it is not; also, sometimes the proof structure is preserved, sometimes not. Notice that, since we are only considering partial correctness, it is always acceptable for the implementation to diverge. In order to make termination guarantees, we could work with total correctness; our decision not to is for simplicity and based on prevailing trends in separation logic and context logic literature [10, 2, 1]. It is possible for our predicate translation to lose information. For instance, if all predicates were unsatisfiable under translation, it would be possible to implement every abstract command with `skip`; such an implementation is useless. It may be desirable to consider some injectivity condition which distinguishes states and predicates of interest. Our results do not rely on this.

**Modularity.** A translation $\mathbb{A}_1 \to \mathbb{A}_2$ can be lifted naturally to a translation $\mathbb{A}_1 + \mathbb{B} \to \mathbb{A}_2 + \mathbb{B}$. We would hope that this translation would be sound, but this is not necessarily the case. Here, we consider general techniques for defining translations that inductively transform proofs from module $\mathbb{A}_1$ to proofs in module $\mathbb{A}_2$. These translations will be modular: the lifting gives a sound translation.

## 5 Locality-preserving Translations

Sometimes there is a close correspondence between locality in an abstract module and locality in its implementation. Consider Fig. 3 which depicts a simple tree (a), and representations of it in the heap module $\mathbb{H}$ (b), and in the combined heap and list module $\mathbb{H} + \mathbb{L}$ (c). In (b), a node is represented by a memory block of four fields, recording the addresses of the left sibling, parent, right sibling and first child. In (c), a node is represented by a list of the child nodes and a block of two fields, recording the address of the parent and the child list. Just as the tree

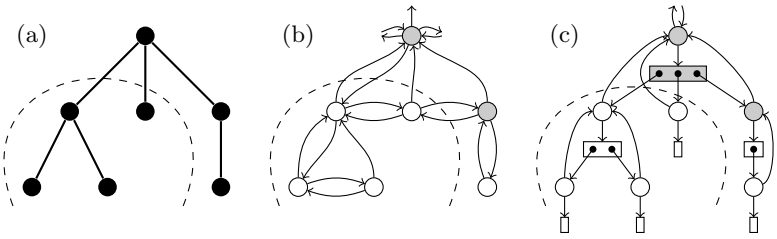


**Fig. 3.** An abstract tree from T (a), and its representations in H (b) and H × Ls (c).

in (a) can be decomposed (as shown by the dashed lined), its representations can also be decomposed: the representations preserve context application. However, we must account for the pointers in the representations which cross the boundary between context and subtree. This means that the representation of a tree must be parameterised by an *interface* to the surrounding context. Similarly, contexts are parameterised by interfaces both to the inner subtree and outer context. We split the interface $I$ into two components: the reference the surrounding context makes *in* to the subtree (the *in* part), and the reference the subtree makes *out* to the surrounding context (the *out* part).

Consider deleting the subtree indicated by the dashed lines in the figure. In the abstract tree, this deletion only operates on the subtree: the axiom for deletion has just the subtree as its precondition. In the implementations, however, the deletion also operates on the representation of the surrounding context: in (b), this is the parent node and right sibling; in (c), the parent node and child list. We therefore introduce the idea of a *crust* predicate, $\Cap{I}^F$, that comprises the minimal additional state required by an implementation. The crust is parameterised by interface $I$ and an additional crust parameter $F$ that fully determine it. In the figure, the crusts for the subtree in (b) and (c) are shown shaded. (In the list-based representation, the sibling nodes form part of the crust because they are required for node insertion.)

We define a general notion of local translation, which incorporates three key properties: *application preservation*, *crust inclusion*, and *axiom correctness*. Application preservation, we have seen, requires that the low-level representations of abstract states can be decomposed in the same manner as the abstract states themselves. Crust inclusion requires that a substate's crust is subsumed by any outer context (together with its own outer crust). This allows us to frame on arbitrary contexts despite the crust already being present (we simply remove the inner crust from the context before applying it). Finally, axiom correctness requires that the implementations of the basic commands meet the specifications given by the abstract module's axioms.

**Theorem 1 (Locality-Preserving Translation).** *For interface set* $\mathcal{I} = \mathcal{I}_{\text{in}} \times \mathcal{I}_{\text{out}}$ *and crust parameter set* $\mathcal{F}$*, a* locality-preserving *translation* $\mathbb{A} \to \mathbb{B}$ *comprises:*

- *representation functions* $\langle\!\langle - \rangle\!\rangle^- : \mathcal{D}_{\mathbb{A}} \times \mathcal{I} \to \mathcal{P}(\mathcal{D}_{\mathbb{B}})$ *and* $\langle\!\langle - \rangle\!\rangle^-_- : \mathcal{C}_{\mathbb{A}} \times \mathcal{I} \times \mathcal{I} \to \mathcal{P}(\mathcal{C}_{\mathbb{B}})$*;*
- *a crust predicate* $\Subset^F_I$*, parameterised by* $I \in \mathcal{I}$ *and* $F \in \mathcal{F}$*; and*
- *a substitutive implementation function* $[\![-]\!] : \mathcal{L}_{\mathbb{A}} \to \mathcal{L}_{\mathbb{B}}$*,*

*for which the following properties hold:*

1. **application preservation:** *for all* $f \in \mathcal{P}(\mathcal{C}_{\mathbb{A}})$*,* $p \in \mathcal{P}(\mathcal{D}_{\mathbb{A}})$ *and* $I \in \mathcal{I}$*,*

$$\langle\!\langle f \circ_{\mathbb{A}} p \rangle\!\rangle^I \quad = \quad \exists I'.\, \langle\!\langle f \rangle\!\rangle^I_{I'} \circ_{\mathbb{B}} \langle\!\langle p \rangle\!\rangle^{I'};$$

2. **crust inclusion:** *for all* $\overrightarrow{out}', \overrightarrow{out} \in \mathcal{I}_{\text{out}}$*,* $F \in \mathcal{F}$*,* $c \in \mathcal{C}_{\mathbb{A}}$*, there exist* $f \in \mathcal{P}(\mathcal{C}_{\mathbb{B}})$*,* $F' \in \mathcal{F}$ *such that, for all* $\overrightarrow{in} \in \mathcal{I}_{\text{in}}$*,*

$$\left(\exists \overrightarrow{in}'.\, \Subset^F_{\overrightarrow{in}', \overrightarrow{out}'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}, \overrightarrow{out}}\right) = f \bullet \Subset^{F'}_{\overrightarrow{in}, \overrightarrow{out}}; \text{ and}$$

3. **axiom correctness:** *for all* $(p, \varphi, q) \in \text{Ax}_{\mathbb{A}}$*,* $\overrightarrow{out} \in \mathcal{I}_{\text{out}}$ *and* $F \in \mathcal{F}$*,*

$$\vdash_{\mathbb{B}} \left\{ (\!| p |\!)^{\overrightarrow{out}, F} \right\} \; [\![\varphi]\!] \; \left\{ (\!| q |\!)^{\overrightarrow{out}, F} \right\},$$

*where* $(\!| p |\!)^{\overrightarrow{out}, F} = \bigvee_{(d,\sigma) \in p} (\exists \overrightarrow{in}.\, \Subset^F_{\overrightarrow{in}, \overrightarrow{out}} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in}, \overrightarrow{out}}) \times \sigma.$

*This is a module translation, with the state translation function* $[\![-]\!] : \mathcal{D}_{\mathbb{A}} \to \mathcal{P}(\mathcal{D}_{\mathbb{B}})$ *defined by* $[\![d]\!] = \exists \overrightarrow{in}.\, \Subset^F_{\overrightarrow{in}, \overrightarrow{out}} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in}, \overrightarrow{out}}$*. A locality-preserving translation is a sound translation.*

This theorem is proved by inductively transforming a high-level proof in $\mathbb{A}$ to the corresponding proof in $\mathbb{B}$, preserving the structure. Application preservation and crust inclusion allow us to transform a high-level frame into a low-level frame, and axiom correctness allows us to soundly replace the high-level commands with their implementations. The remaining proof rules transform naturally. If we choose to include the conjunction rule in our proof system, then we would need to additionally verify that our representation functions preserve conjunction and also that the crust predicate $\exists \overrightarrow{in}.\, \Subset^F_{\overrightarrow{in}, \overrightarrow{out}}$ is precise.

### 5.1 Module Translation: $\mathbb{T} \longrightarrow \mathbb{H} + \mathbb{L}$

We now study the list-based implementation which uses a combination of the heap and list modules given in §3. As we have seen, each node of the tree is represented by a list of addresses of the node's children and a memory block of two fields that record the addresses of the parent node and child list. The representation functions for trees and tree contexts are given below. The *out* part of the interface, $l \in (\mathbb{N}^+)^*$, is a list of the addresses of the top-level nodes of

```
n.parent ≜ n
n.children ≜ n + 1
n := newNode() ≜ n := alloc(2)
disposeNode(n) ≜ dispose(n, 2)

proc n′ := getLast(n){
 local x in
  x := [n.children] ;
  n′ := x.getTail()
}
```

```
proc n′ := getRight(n){
  local x, y in
    x := [n.parent] ;
    y := [x.children] ;
    n′ := y.getNext(n)
}
```

```
proc deleteTree(n){
 local x, y, z in
  x := [n.parent] ;
  y := [x.children] ;
  y.remove(n) ;
  y := [n.children] ;
  z := y.getHead() ;
  while z ≠ null do
   call deleteTree(z) ;
   z := y.getHead()
  disposeList(y) ;
  disposeNode(n)
}
```

**Fig. 4.** Selected procedures for the list-based implementation

the subtree. The *in* part of the interface, $u \in \mathbb{N}^+$, is the address of the subtree's parent node. (We abuse notation, freely combining heaps and list stores with $*$.)

$$\begin{aligned}
\langle\!\langle \varnothing \rangle\!\rangle^{\varepsilon,u} &::= \mathrm{emp} \\
\langle\!\langle n[t] \rangle\!\rangle^{n,u} &::= \exists i, l.\, n \mapsto u, i * i \Rrightarrow [\, l \,] * \langle\!\langle t \rangle\!\rangle^{l,n} \\
\langle\!\langle t_1 \otimes t_2 \rangle\!\rangle^{l,u} &::= \exists l_1, l_2.\, (l \doteq l_1 + l_2) * \langle\!\langle t_1 \rangle\!\rangle^{l_1,u} * \langle\!\langle t_2 \rangle\!\rangle^{l_2,u} \\
\\
\langle\!\langle - \rangle\!\rangle^{l,u}_{l',u'} &::= (l \doteq l') * (u \doteq u') \\
\langle\!\langle n[c] \rangle\!\rangle^{n,u}_{I'} &::= \exists i, l.\, n \mapsto u, i * i \Rrightarrow [\, l \,] * \langle\!\langle c \rangle\!\rangle^{l,n}_{I'} \\
\langle\!\langle t \otimes c \rangle\!\rangle^{l,u}_{I'} &::= \exists l_1, l_2.\, (l \doteq l_1 + l_2) * \langle\!\langle t \rangle\!\rangle^{l_1,u} * \langle\!\langle c \rangle\!\rangle^{l_2,u}_{I'} \\
\langle\!\langle c \otimes t \rangle\!\rangle^{l,u}_{I'} &::= \exists l_1, l_2.\, (l \doteq l_1 + l_2) * \langle\!\langle c \rangle\!\rangle^{l_1,u}_{I'} * \langle\!\langle t \rangle\!\rangle^{l_2,u}
\end{aligned}$$

The crust, $\Cap^F_I$, parameterised by interface $I = l, u$ and free logical variables $F = (l_1, l_2, u')$, is defined as follows:

$$\Cap^{l_1,l_2,u'}_{l,u} ::= \exists i.\, u \mapsto u', i * i \Rrightarrow [\, l_1 + l + l_2 \,] * \left( \prod_{n \in l_1 + l_2}^{*} n \mapsto u' \right)$$

**Definition 15 (Translation: $\mathbb{T} \to \mathbb{H} + \mathbb{L}$).** *For a root address $r$, the state translation is defined as: $[\![d]\!] = \exists l.\, \Cap^{\varepsilon,\varepsilon,\mathbf{null}}_{l,r} * \langle\!\langle d \rangle\!\rangle^{l,r}$. A selection of the procedures constituting the substitutive implementation is given in Fig. 4.*

**Theorem 2.** *The translation defined above is sound.*

### 5.2 Module Translation: $\mathbb{H} + \mathbb{H} \to \mathbb{H}$

Another example of a local translation is given by implementing a pair of heap modules $\mathbb{H} + \mathbb{H}$ in a single heap $\mathbb{H}$. The intuitive approach to this is to simply treat the two heaps as disjoint portions of the same heap and use the same commands for working with both.

**Definition 16 (Translation: $\mathbb{H}+\mathbb{H} \to \mathbb{H}$).** *The state translation is defined as:* $[\![(h_1, h_2)]\!] = \{h_1\} * \{h_2\}$. *The implementation* $[\![\mathbb{C}]\!]$ *is defined to be the detagging of* $\mathbb{C}$*: that is, heap commands from both abstract modules are substituted with the corresponding command from the single abstract module. For example:*

$$[\![n := \mathtt{cons}_1(E_1, \ldots, E_k)]\!] = n := \mathtt{cons}(E_1, \ldots, E_k) = [\![n := \mathtt{cons}_2(E_1, \ldots, E_k)]\!]$$

**Theorem 3.** *The translation defined above is sound.*

(Note, the represetation function in this case does not preserve conjunction.)

## 6 Locality-breaking Translations

There is not always a close correspondence between locality in an abstract module and locality in its implementation. For example, consider an implementation of our list module that represents each list as a singly-linked list in the heap. In the abstract module, the footprint of removing a specific element from a list is just that element in that list. In the implementation however, the list is traversed from its head to reach the element, which is then deleted by modifying the pointer of its predecessor. The footprint is therefore the list fragment from the head of the list to this predecessor, significantly more than the single list node holding the value to be removed. While we could treat this additional footprint as crust, in this case it seems more appropriate to abandon the preservation of locality and instead use a translation that gives a fiction of locality.

Consider a translation from abstract module $\mathbb{A}$ to $\mathbb{B}$. With the exception of the frame rule and axioms, the proof rules for $\mathbb{A}$ can be mapped to the corresponding proof rules of $\mathbb{B}$: that is, from the translated premises we can directly deduce the translated conclusion. To deal with the frame rule, we remove it from proofs in $\mathbb{A}$ by 'pushing' applications of the frame rule to the leaves of the proof tree. In this way, we can transform any local proof to a non-local proof.

**Lemma 1 (Frame-free Derivations).** *Let* $\mathbb{A}$ *be an abstract module. If there is a derivation of* $\vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\}$ *then there is also a derivation that only uses the frame rule in the following ways:*

$$\dfrac{\dfrac{}{\Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}\ (\dagger)}{\Gamma \vdash \{f \circ p\}\ \mathbb{C}\ \{f \circ q\}} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}}{\Gamma \vdash \{(\mathbf{I}_{\mathbb{A}} \times \sigma) \circ p\}\ \mathbb{C}\ \{(\mathbf{I}_{\mathbb{A}} \times \sigma) \circ q\}}$$

*where* (†) *is either* Axiom, Skip *or* Assgn.

By transforming a high-level proof of $\vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\}$ in this way, we can establish $\vdash_{\mathbb{B}} \{[\![p]\!]\}\ [\![\mathbb{C}]\!]\ \{[\![q]\!]\}$ provided that we can prove that the implementation of each command of $\Phi_{\mathbb{A}}$ satisfies the translation of each of its axioms under every frame. (We can reduce considerations to any *singleton* frame by considering any given frame as a disjunction of singletons and applying the Disj rule.)

```
x.value ≜ x
x.next ≜ x + 1
x := newNode() ≜ x := alloc(2)
disposeNode(x) ≜ dispose(x, 2)

proc i.remove(v){
  local u, x, y, z in
  x := [i] ;
  u := [x.value] ;
  y := [x.next] ;
  if u = v
then
  [E] := y ;
  disposeNode(x)
else
  u := [y.value] ;
  while u ≠ v do
    x := y ;
    y := [x.next] ;
    u := [y.value]
  z := [y.next] ;
  [x.next] := z ;
  disposeNode(y)
}

proc v := i.getNext(v′){
  local x in
    x := [i] ;
    v := [x.value] ;
    while v ≠ v′ do
      x := [x.next] ;
      v := [x.value]
    x := [x.next] ;
    if x = null then v := x
    else v := [x.value]
}
```

**Fig. 5.** Selected procedures for the linked-list implementation

**Theorem 4 (Locality-breaking Translation).** *A* locality-breaking *translation* $\mathbb{A} \to \mathbb{B}$ *is one such that, for all* $c \in \mathcal{C}_{\mathbb{A}}$ *and* $(p, \varphi, q) \in \mathrm{Ax}_{\mathbb{A}}$*, the judgement* $\vdash_{\mathbb{B}} \{[\![\{c\} \circ p]\!]\}\ [\![\varphi]\!]\ \{[\![\{c\} \circ q]\!]\}$ *holds. A locality-breaking translation is sound.*

If we include the conjunction rule, then we must verify that every singleton context predicate is precise (i.e. the context algebra must be left-cancellative).

### 6.1 Module Translation: $\mathbb{L} \to \mathbb{H}$

We show a locality-breaking translation $\mathbb{L} \to \mathbb{H}$, which implements abstract lists with singly-linked lists in the heap.

**Definition 17.** *The state translation from list-stores to heaps is defined inductively as follows:*

$$[\![\emptyset]\!] ::= \mathrm{emp} \qquad [\![i \Mapsto l * ls]\!] ::= \mathbf{False}$$
$$[\![i \Mapsto [\,l\,] * ls]\!] ::= \exists x.\, i \mapsto x * \langle\!\langle l \rangle\!\rangle^{(x,\mathbf{null})} * [\![ls]\!]$$

*where*

$$\langle\!\langle \varepsilon \rangle\!\rangle^{(x,y)} ::= (x \doteq y) \qquad \langle\!\langle v \rangle\!\rangle^{(x,y)} ::= x \mapsto v,y$$
$$\langle\!\langle l + l' \rangle\!\rangle^{(x,y)} ::= \exists z.\, \langle\!\langle l \rangle\!\rangle^{(x,z)} * \langle\!\langle l' \rangle\!\rangle^{(z,y)}$$

Note that not all list stores are realised by heaps: only ones in which every list is complete. The intuitive reason behind this is that partial lists are purely abstract notions that provide a useful means to our ultimate end, namely reasoning about complete lists. The abstract module itself does not provide operations for creating or destroying partial lists, and so we would not expect to give specifications for complete programs that concern partial lists.

**Definition 18 (Translation: $\mathbb{L} \to \mathbb{H}$).** *The state translation* $[\![p]\!]$ *is given by Definition 17. A selection of the procedures constituting the substitutive implementation is given in Fig. 5.*

**Theorem 5.** *The translation defined above is sound.*

**Conclusion** We have seen how to define abstract modules in such a way that their combinations and implementations can be reasoned about in a modular fashion. We have defined a number of useful abstract modules and shown how to implement these high-level modules in terms of low-level modules. In particular, we have shown how to implement an abstract tree module in terms of a heap module and a list module. We have shown that we can further refine this by implementing the abstract list module in terms of a heap module. We also made the observation that we can combine multiple abstract heap modules into a single abstract heap module. So the translations of this paper form a chain of implementations, as shown in Fig. 1 in the introduction.

We have only scratched the surface of refinement in the setting of local reasoning. In particular, we are interested in exploring the fiction of disjointness in more depth. With others, we have begun to investigate this fiction with work on concurrent abstract modules [4], and we are keen to fathom fully these fascinating waters.

**Acknowledgments:** Gardner acknowledges support of a Microsoft/RAEng Senior Research Fellowship. Dinsdale-Young and Wheelhouse acknowledge support of an EPSRC DTA award. We thank Mohammad Raza and Uri Zarfaty for detailed discussions of this work.

## References

1. C. Calcagno, P. Gardner, and U. Zarfaty. Context logic and tree update. In *POPL*, volume 40 of *ACM SIGPLAN Notices*, pages 271–282, 2005.
2. C. Calcagno, P. W. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS*, pages 366–378, 2007.
3. W. DeRoever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and Their Comparison*. Cambridge University Press, New York, NY, USA, 1999.
4. T. Dinsdale-Young, M. Dodds, P. Gardner, M. Parkinson, and V. Vafeiadis. Concurrent abstract predicates. In *ECOOP'10 (to appear)*, 2010.
5. T. Dinsdale-Young, P. Gardner, and M. Wheelhouse. Abstract local reasoning. Technical report, Imperial College London, 2010. `http://www.doc.ic.ac.uk/~td202/papers/alrfull.pdf`.
6. I. Filipović, P. O'Hearn, N. Torp-Smith, and H. Yang. Blaming the client: on data refinement in the presence of pointers. *Formal Aspects of Computing*, Online, 2009.
7. P. A. Gardner, G. D. Smith, M. J. Wheelhouse, and U. D. Zarfaty. Local hoare reasoning about dom. In *PODS '08*, pages 261–270, New York, NY, USA, 2008. ACM.
8. J. He, C. A. R. Hoare, and J. W. Sanders. Data refinement refined. In *Proc. of the European symposium on programming on ESOP 86*, pages 187–196, New York, NY, USA, 1986. Springer-Verlag New York, Inc.
9. C. A. R. Hoare. Proof of correctness of data representations. *Acta Inf.*, 1:271–281, 1972.
10. P. W. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL*, Lecture Notes in Computer Science. Springer, 2001.
11. M. Parkinson and G. Bierman. Separation logic and abstraction. *SIGPLAN Not.*, 40(1):247–258, 2005.

# A Correctness of the Locality-preserving Theory

We use the notation $f \mathrel{-\!\!\bullet} g$ for the predicate $\{c \mid \forall c', c'' \in \mathcal{C}.\, (c'' = c \bullet c' \wedge c' \in f) \implies c'' \in g\}$, and $f \mathrel{\bullet\!\!-} g$ for the analogous predicate defined using $c' \bullet c$ (the two right adjoints of context composition).

Assume that we are given:

- abstract modules $\mathbb{A}$ and $\mathbb{B}$;
- a substitutive implementation function $[\![-]\!] : \mathcal{L}_{\mathbb{A}} \to \mathcal{L}_{\mathbb{B}}$;
- a set $\mathcal{I} = \mathcal{I}_{\text{in}} \times \mathcal{I}_{\text{out}}$ of interfaces $I = (\overrightarrow{in}, \overrightarrow{out})$;
- a state representation function $\langle\!\langle - \rangle\!\rangle^{-} : \mathcal{D}_{\mathbb{A}} \times \mathcal{I} \to \mathcal{P}(\mathcal{D}_{\mathbb{B}})$;
- a context representation function $\langle\!\langle - \rangle\!\rangle^{-}_{-} : \mathcal{C}_{\mathbb{A}} \times \mathcal{I} \times \mathcal{I} \to \mathcal{P}(\mathcal{C}_{\mathbb{B}})$; and
- a crust predicate $\Subset^F_I \in \mathcal{P}(\mathcal{C}_{\mathbb{B}})$ parameterised by interface $I \in \mathcal{I}$ and by $F \in \mathcal{F}$, for some set $\mathcal{F}$.

**Definition 19 (Intermediate Translation Functions).** *We define the intermediate state-predicate translation* $(\!|-|\!)^{-} : \mathcal{P}(\mathcal{D} \times \Sigma) \times (\mathcal{I}_{\text{out}} \times \mathcal{F}) \to \mathcal{P}(\mathcal{D}_{\mathbb{B}} \times \Sigma)$ *and the intermediate context-predicate translation* $(\!|-|\!)^{-}_{-} : \mathcal{P}(\mathcal{D} \times \Sigma) \times (\mathcal{I}_{\text{out}} \times \mathcal{F}) \times (\mathcal{I}_{\text{out}} \times \mathcal{F}) \to \mathcal{P}(\mathcal{D}_{\mathbb{B}} \times \Sigma)$ *as follows:*

$$(\!|p|\!)^{\overrightarrow{out},F} = \bigvee_{(d,\sigma)\in p} \left(\exists \overrightarrow{in}.\, \Subset^{F'}_{\overrightarrow{in},\overrightarrow{out}} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in},\overrightarrow{out}}\right) \times \sigma$$

$$(\!|f|\!)^{\overrightarrow{out}',F'}_{\overrightarrow{out},F} = \bigvee_{(c,\sigma)\in f} \left(\forall \overrightarrow{in}''.\, \Subset^{F}_{\overrightarrow{in}'',\overrightarrow{out}} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Subset^{F'}_{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}\right)\right) \times \sigma$$

Assume also that the following properties hold:

**Property 1 (Application Preservation).** *Context application is preserved by the translation* $\langle\!\langle - \rangle\!\rangle^{I}$ *with respect to some interface* $I' = (\overrightarrow{in}', \overrightarrow{out}')$.

$$\langle\!\langle f \circ_1 p \rangle\!\rangle^{I} \quad \equiv \quad \exists I'.\, \langle\!\langle f \rangle\!\rangle^{I}_{I'} \circ_2 \langle\!\langle p \rangle\!\rangle^{I'}$$

**Property 2 (Crust Inclusion).** *For all* $\overrightarrow{out}', \overrightarrow{out} \in \mathcal{I}_{\text{out}}$, $F \in \mathcal{F}$, $c \in \mathcal{C}_{\mathbb{A}}$ *there exist* $q \in \mathcal{P}(\mathcal{C}_{\mathbb{B}})$, $F' \in \mathcal{F}$ *such that for all* $\overrightarrow{in} \in \mathcal{I}_{\text{in}}$

$$\left(\exists \overrightarrow{in}'.\, \Subset^{F}_{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in},\overrightarrow{out}}\right) \equiv q \bullet \Subset^{F'}_{\overrightarrow{in},\overrightarrow{out}}.$$

**Property 3 (Axiom Correctness).** *For all* $(p, \varphi, q) \in \text{Ax}_{\mathbb{A}}$, $\overrightarrow{out} \in \mathcal{I}_{\text{out}}$ *and* $F \in \mathcal{F}$

$$\vdash_{\mathbb{B}} \left\{(\!|p|\!)^{\overrightarrow{out},F}\right\}\ \varphi\ \left\{(\!|q|\!)^{\overrightarrow{out},F}\right\}$$

We wish to establish the following:

**Proposition 1.** *For all* $F, \overrightarrow{out}$ *and for all* $p, q \in \mathcal{P}(\mathcal{A}_{\mathbb{A}} \times \Sigma)$ *and* $\mathbb{C} \in \mathcal{L}_{\mathbb{A}}$

$$\Gamma \vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\} \quad \implies \quad [\![\Gamma]\!] \vdash_{\mathbb{B}} \left\{(\!|p|\!)^{\overrightarrow{out},F}\right\}\ [\![\mathbb{C}]\!]\ \left\{(\!|q|\!)^{\overrightarrow{out},F}\right\},$$

*where*

$$[\![\Gamma]\!] = \left\{ \mathtt{f} : (\!|P|\!)^{\overrightarrow{out}',F'} \to (\!|Q|\!)^{\overrightarrow{out}',F'} \;\middle|\; \begin{array}{l} (\mathtt{f} : P \to Q) \in \mathrm{Ax}_{\mathbb{A}} \\ \wedge\, \overrightarrow{out}' \in \mathcal{I}_{\mathrm{out}} \wedge F' \in \mathcal{F} \end{array} \right\}.$$

The following lemma gives an alternative characterisation of the crust inclusion property:

**Lemma 2 (Crust Inclusion II).** *For all $f \in \mathcal{P}(\mathcal{C}_{\mathbb{A}})$ and all $\overrightarrow{out}'$, $F$, $\overrightarrow{in}$ and $\overrightarrow{out}$,*

$$\left(\exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in},\overrightarrow{out}}\right) \subseteq \exists F'.\, \left(\forall \overrightarrow{in}''.\, \Cap{F'}{\overrightarrow{in}'',\overrightarrow{out}} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}\right)\right) \bullet \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}.$$

*Note that the converse of this property is trivially true.*

*Proof.* Consider an arbitrary context assertion, $f$, and fix $\overrightarrow{out}'$, $F$, $\overrightarrow{in}$ and $\overrightarrow{out}$. Fix $c'$ with

$$\begin{aligned} c' \in & \exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in},\overrightarrow{out}} \\ & \equiv \bigvee_{c \in f} \left(\exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in},\overrightarrow{out}}\right). \end{aligned}$$

There exists $c'' \in f$ such that

$$c' \in \exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle c'' \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in},\overrightarrow{out}}$$

By the Crust Inclusion Property, there exist $q$ and $F'$ such that, for all $\overrightarrow{in}''$,

$$\left(\exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle c'' \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}\right) \equiv q \bullet \Cap{F'}{\overrightarrow{in}'',\overrightarrow{out}}. \tag{1}$$

Hence, $c' \in q \bullet \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}$, and so there are $c_1 \in q$ and $c_2 \in \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}$ with $c' = c_1 \bullet c_2$. Fix $\overrightarrow{in}''$ and $c_2' \in \Cap{F'}{\overrightarrow{in}'',\overrightarrow{out}}$. Since $c_1 \bullet c_2' \in q \bullet \Cap{F'}{\overrightarrow{in}'',\overrightarrow{out}}$, it follows by (1) that

$$\begin{aligned} c_1 \bullet c_2' \in & \left(\exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle c'' \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}\right) \\ & \subseteq \exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}. \end{aligned}$$

The choice of $c_2'$ was arbitrary, and so

$$\forall c_2', c''.\, c_2' \in \Cap{F}{\overrightarrow{in}'',\overrightarrow{out}} \wedge c'' = c_1 \bullet c_2' \implies c_2'' \in \exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}.$$

Hence

$$c_1 \in \Cap{F'}{\overrightarrow{in}'',\overrightarrow{out}} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{F}{\overrightarrow{in}',\overrightarrow{out}'} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}',\overrightarrow{out}'}_{\overrightarrow{in}'',\overrightarrow{out}}\right)$$

and since the choice of $\overrightarrow{in}''$ was arbitrary,

$$c_1 \in \forall \overrightarrow{in}''.\, \Cap{\overrightarrow{in}'', \overrightarrow{out}}^{F'} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}'', \overrightarrow{out}}\right).$$

Since $c' = c_1 \bullet c_2$,

$$c' \in \exists F'.\, \left(\forall \overrightarrow{in}''.\, \Cap{\overrightarrow{in}'', \overrightarrow{out}}^{F'} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}'', \overrightarrow{out}}\right)\right) \bullet \Cap{\overrightarrow{in}, \overrightarrow{out}}^{F'}.$$

Since the choice of $c'$ was arbitrary, we conclude

$$\left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}, \overrightarrow{out}}\right) \subseteq \exists F'.\, \left(\forall \overrightarrow{in}''.\, \Cap{\overrightarrow{in}'', \overrightarrow{out}}^{F'} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F} \bullet \langle\!\langle f \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}'', \overrightarrow{out}}\right)\right) \bullet \Cap{\overrightarrow{in}, \overrightarrow{out}}^{F'}.$$

□

In order to prove Proposition 1, we use the Intermediate Translation Functions, for which application preservation holds:

**Lemma 3 (Application Preservation II).** *For all $f \in \mathcal{P}(\mathcal{C}_{\mathbb{A}} \times \Sigma)$, $p \in \mathcal{P}(\mathcal{D}_{\mathbb{A}} \times \Sigma)$, $\overrightarrow{out} \in \mathcal{I}_{\text{out}}$ and $F \in \mathcal{F}$*

$$(\!| f \circ p |\!)^{\overrightarrow{out}, F} \equiv \exists \overrightarrow{out}', F'.\, (\!| f |\!)^{\overrightarrow{out}, F}_{\overrightarrow{out}', F'} \circ (\!| p |\!)^{\overrightarrow{out}', F'}.$$

*Proof.* By applying Lemma 2 and Property 1, we get:

$$\begin{aligned}
&(\!| f \circ p |\!)^{\overrightarrow{out}', F'} \\
&= \bigvee_{\substack{(c,\sigma') \in f \\ (d,\sigma) \in p}} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F'} \circ \langle\!\langle c \circ d \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}\right) \times (\sigma' * \sigma) \\
&= \bigvee_{\substack{(c,\sigma') \in f \\ (d,\sigma) \in p}} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F'} \circ \exists \overrightarrow{in}, \overrightarrow{out}.\, \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}, \overrightarrow{out}} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in}, \overrightarrow{out}}\right) \times (\sigma' * \sigma) \\
&= \bigvee_{\substack{(c,\sigma') \in f \\ (d,\sigma) \in p}} \left(\exists \overrightarrow{in}, \overrightarrow{out}.\, \exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}, \overrightarrow{out}} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in}, \overrightarrow{out}}\right) \times (\sigma' * \sigma) \\
&= \bigvee_{\substack{(c,\sigma') \in f \\ (d,\sigma) \in p}} \left(\begin{array}{r} \left(\exists \overrightarrow{in}, \overrightarrow{out}.\, \exists F.\, \left(\forall \overrightarrow{in}''.\, \Cap{\overrightarrow{in}'', \overrightarrow{out}}^{F} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}'', \overrightarrow{out}}\right)\right)\right) \\ \bullet\, \Cap{\overrightarrow{in}, \overrightarrow{out}}^{F} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in}, \overrightarrow{out}}\right) \times (\sigma' * \sigma) \end{array}\right) \\
&= \begin{array}{r} \exists \overrightarrow{out}, F.\, \left(\bigvee_{(c,\sigma') \in f} \left(\forall \overrightarrow{in}''.\, \Cap{\overrightarrow{in}'', \overrightarrow{out}}^{F} \mathrel{-\!\!\bullet} \left(\exists \overrightarrow{in}'.\, \Cap{\overrightarrow{in}', \overrightarrow{out}'}^{F'} \bullet \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in}', \overrightarrow{out}'}_{\overrightarrow{in}'', \overrightarrow{out}}\right)\right) \times \sigma'\right) \circ \\ \left(\bigvee_{(d,\sigma) \in p} \left(\Cap{\overrightarrow{in}, \overrightarrow{out}}^{F} \circ \langle\!\langle d \rangle\!\rangle^{\overrightarrow{in}, \overrightarrow{out}}\right) \times \sigma\right) \end{array} \\
&= \exists \overrightarrow{out}, F.\, (\!| f |\!)^{\overrightarrow{out}', F'}_{\overrightarrow{out}, F} \circ (\!| p |\!)^{\overrightarrow{out}, F}
\end{aligned}$$

□

The proof of Proposition 1 inductively transforms a proof in $\mathbb{A}$ to a proof in $\mathbb{B}$.

*Proof (Proposition 1).*

The proof is by induction on the structure of the proof of $\vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\}$ and cases on the last rule of the proof. We assume as the inductive hypothesis that the translated premises have proofs in $\mathbb{B}$ and show how to derive from these a proof of the translated conclusion. (We omit the procedure environment when it plays no role in the derivation.)

**Frame:**

$$\dfrac{\dfrac{\dfrac{\forall \overrightarrow{out}', F'.\ \left\{ (\!|p|\!)^{\overrightarrow{out}',F'} \right\}\ \mathbb{C}\ \left\{ (\!|q|\!)^{\overrightarrow{out}',F'} \right\}}{\forall \overrightarrow{out}', F'.\ \left\{ (\!|f|\!)^{\overrightarrow{out},F}_{\overrightarrow{out}',F'} \circ (\!|p|\!)^{\overrightarrow{out}',F'} \right\}\ \mathbb{C}\ \left\{ (\!|f|\!)^{\overrightarrow{out},F}_{\overrightarrow{out}',F'} \circ (\!|q|\!)^{\overrightarrow{out}',F'} \right\}}\ \textsc{Frame}}{\left\{ \exists \overrightarrow{out}', F'.\ (\!|f|\!)^{\overrightarrow{out},F}_{\overrightarrow{out}',F'} \circ (\!|p|\!)^{\overrightarrow{out}',F'} \right\}\ \mathbb{C}\ \left\{ \exists \overrightarrow{out}', F'.\ (\!|f|\!)^{\overrightarrow{out},F}_{\overrightarrow{out}',F'} \circ (\!|q|\!)^{\overrightarrow{out}',F'} \right\}}\ \textsc{Disj}}{\left\{ (\!|f \circ p|\!)^{\overrightarrow{out},F} \right\}\ \mathbb{C}\ \left\{ (\!|f \circ q|\!)^{\overrightarrow{out},F} \right\}}\ \text{Lemma 3}$$

**Consequence:**

$$\dfrac{\dfrac{p' \subseteq p}{(\!|p'|\!)^{\overrightarrow{out},F} \subseteq (\!|p|\!)^{\overrightarrow{out},F}}\quad \left\{ (\!|p|\!)^{\overrightarrow{out},F} \right\}\ \mathbb{C}\ \left\{ (\!|q|\!)^{\overrightarrow{out},F} \right\}\quad \dfrac{q \subseteq q'}{(\!|q|\!)^{\overrightarrow{out},F} \subseteq (\!|q'|\!)^{\overrightarrow{out},F}}}{\left\{ (\!|p'|\!)^{\overrightarrow{out},F} \right\}\ \mathbb{C}\ \left\{ (\!|q'|\!)^{\overrightarrow{out},F} \right\}}\ \textsc{Cons}$$

**Disjunction:**

$$\dfrac{\dfrac{\forall i \in I.\ \left\{ (\!|p_i|\!)^{\overrightarrow{out},F} \right\}\ \mathbb{C}\ \left\{ (\!|q_i|\!)^{\overrightarrow{out},F} \right\}}{\left\{ \bigvee_{i \in I} (\!|p_i|\!)^{\overrightarrow{out},F} \right\}\ \mathbb{C}\ \left\{ \bigvee_{i \in I} (\!|q_i|\!)^{\overrightarrow{out},F} \right\}}\ \textsc{Disj}}{\left\{ (\!|\bigvee_{i \in I} p_i|\!)^{\overrightarrow{out},F} \right\}\ \mathbb{C}\ \left\{ (\!|\bigvee_{i \in I} q_i|\!)^{\overrightarrow{out},F} \right\}}$$

**Procedure Definition:**

$$\frac{\forall \begin{array}{l}(\mathtt{f}_i : P' \to Q') \in \Gamma, \\ \overrightarrow{out'} \in \mathcal{I}_{\text{out}}, F' \in \mathcal{F}\end{array}.\ [\![\Gamma', \Gamma]\!] \vdash \begin{array}{c}\left\{(\!|\exists \overrightarrow{v}.\, P(\overrightarrow{v}) \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow \overrightarrow{v} * \overrightarrow{\boldsymbol{r}} \Rrightarrow -)|\!)^{\overrightarrow{out'}, F'}\right\} \\ \mathbb{C}_i \\ \left\{(\!|\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow - * \overrightarrow{\boldsymbol{r}_i} \Rrightarrow \overrightarrow{w})|\!)^{\overrightarrow{out'}, F'}\right\}\end{array}}{\forall \begin{array}{l}(\mathtt{f}_i : P' \to Q') \in \Gamma, \\ \overrightarrow{out'} \in \mathcal{I}_{\text{out}}, F' \in \mathcal{F}\end{array}.\ [\![\Gamma', \Gamma]\!] \vdash \begin{array}{c}\left\{\exists \overrightarrow{v}.\, (\!|P(\overrightarrow{v})|\!)^{\overrightarrow{out'}, F'} \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow \overrightarrow{v} * \overrightarrow{\boldsymbol{r}} \Rrightarrow -)\right\} \\ \mathbb{C}_i \\ \left\{\exists \overrightarrow{w}.\, (\!|Q(\overrightarrow{w})|\!)^{\overrightarrow{out'}, F'} \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow - * \overrightarrow{\boldsymbol{r}_i} \Rrightarrow \overrightarrow{w})\right\}\end{array}}$$

$$\frac{\forall (\mathtt{f}_i : P \to Q) \in [\![\Gamma]\!].\ [\![\Gamma', \Gamma]\!] \vdash \begin{array}{c}\{\exists \overrightarrow{v}.\, P(\overrightarrow{v}) \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow \overrightarrow{v} * \overrightarrow{\boldsymbol{r}_i} \Rrightarrow -)\} \\ \mathbb{C}_i \\ \{\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{\boldsymbol{x}_i} \Rrightarrow - * \overrightarrow{\boldsymbol{r}_i} \Rrightarrow \overrightarrow{w})\}\end{array}}{(\star)}$$

$$\frac{(\star) \quad [\![\Gamma', \Gamma]\!] \vdash \left\{(\!|p|\!)^{\overrightarrow{out}, F}\right\}\ \mathbb{C}\ \left\{(\!|q|\!)^{\overrightarrow{out}, F}\right\}}{[\![\Gamma']\!] \vdash \mathtt{procs}\ \overrightarrow{r_1} := \mathtt{f}_1(\overrightarrow{\boldsymbol{x}_1})\{\mathbb{C}_1\}, \ldots, \overrightarrow{r_k} := \mathtt{f}_k(\overrightarrow{\boldsymbol{x}_k})\{\mathbb{C}_k\}\ \mathtt{in}\ \mathbb{C}\ \begin{array}{c}\left\{(\!|p|\!)^{\overrightarrow{out}, F}\right\} \\ \left\{(\!|q|\!)^{\overrightarrow{out}, F}\right\}\end{array}}\ \textsc{PDef}$$

The cases for the remaining rules follow by the pointwise and variable-preserving nature of the translation. □

This completes the proof of Theorem 1.

## B Correctness of the List-based Tree Implementation

In the following section we show that the selected implementations for commands of our abstract tree module are correct. We do this following the general theory for locality preserving translations laid out in § 5. We need to show that the translation from the abstract tree module to the list-based implementation satisfies the application preservation, crust inclusion and axiom correctness properties.

### B.1 application preservation

We need to show that context application is preserved by the representation functions for trees and tree contexts given in § 5.1.

**Lemma 4 (Application Preservation).**

$$\langle\!\langle f \circ p \rangle\!\rangle^I \quad \equiv \quad \exists I'.\, \langle\!\langle f \rangle\!\rangle^I_{I'} * \langle\!\langle p \rangle\!\rangle^{I'}$$

*Proof.* Fix tree $t$. We wish to show, by induction on the structure of context $c$, that $\langle\!\langle c \circ t \rangle\!\rangle^I \equiv \exists I'.\, \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'}$.

$c = -$: For $\langle\!\langle - \rangle\!\rangle^I_{I'}$ to be defined, $I = I'$. Therefore,

$$\begin{aligned}\exists I'.\, \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} &\equiv \langle\!\langle - \rangle\!\rangle^I_I * \langle\!\langle t \rangle\!\rangle^I \\ &\equiv \langle\!\langle t \rangle\!\rangle^I \\ &\equiv \langle\!\langle c \circ t \rangle\!\rangle^I.\end{aligned}$$

$c = n[c']$: Assume $I = n, p$ for some $p$ (otherwise, $\langle\!\langle c \rangle\!\rangle^I_{I'}$ is not defined).

$$\begin{aligned}\exists I'.\, \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} &\equiv \exists I'.\, \langle\!\langle n[c'] \rangle\!\rangle^{n,p}_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} \\ &\equiv \exists I'.\, \exists i, l.\, n \mapsto p, i * i \Mapsto [\, l \,] * \langle\!\langle c' \rangle\!\rangle^{l,n}_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} \\ &\equiv \exists i, l.\, n \mapsto p, i * i \Mapsto [\, l \,] * \langle\!\langle c' \circ t \rangle\!\rangle^{l,n} \\ &\equiv \langle\!\langle n[c' \circ t] \rangle\!\rangle^{n,p} \\ &\equiv \langle\!\langle n[c'] \circ t \rangle\!\rangle^I.\end{aligned}$$

$c = c' \otimes t'$: Assume $I = l, p$ for some $l$ and $p$.

$$\begin{aligned}\exists I'.\, \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} &\equiv \exists I'.\, \langle\!\langle c' \otimes t' \rangle\!\rangle^{l,p}_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} \\ &\equiv \exists I'.\, \exists l_1, l_2.\, (l \doteq l_1 + l_2) * \langle\!\langle c' \rangle\!\rangle^{l_1,p}_{I'} * \langle\!\langle t' \rangle\!\rangle^{l_2,p} * \langle\!\langle t \rangle\!\rangle^{I'} \\ &\equiv \exists l_1, l_2.\, (l \doteq l_1 + l_2) * \langle\!\langle c' \circ t \rangle\!\rangle^{l_1,p} * \langle\!\langle t' \rangle\!\rangle^{l_2,p} \\ &\equiv \langle\!\langle (c' \circ t) \otimes t' \rangle\!\rangle^{l,p} \\ &\equiv \langle\!\langle (c' \otimes t') \circ t \rangle\!\rangle^I.\end{aligned}$$

The remaining case ($c = t' \otimes c'$) follows a similar pattern.

By induction, for all trees $t$ and contexts $c$, $\langle\!\langle c \circ t \rangle\!\rangle^I \equiv \exists I'.\, \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'}$.

Suppose that $f$ is a set of contexts and $p$ a set of trees.

$$\begin{aligned}\langle\!\langle f \circ p \rangle\!\rangle^I &\equiv \langle\!\langle \bigvee_{c \in f, t \in p} c \circ t \rangle\!\rangle^I \\ &\equiv \bigvee_{c \in f, t \in p} \langle\!\langle c \circ t \rangle\!\rangle^I \\ &\equiv \bigvee_{c \in f, t \in p} \exists I'.\, \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} \\ &\equiv \exists I'.\, \bigvee_{c \in f, t \in p} \langle\!\langle c \rangle\!\rangle^I_{I'} * \langle\!\langle t \rangle\!\rangle^{I'} \\ &\equiv \exists I'.\, \langle\!\langle f \rangle\!\rangle^I_{I'} * \langle\!\langle p \rangle\!\rangle^{I'}.\end{aligned}$$

□

### B.2 crust inclusion

**Lemma 5 (Crust Inclusion).** *For all* $\overrightarrow{out'}, F, \overrightarrow{out}, c$ *there exist* $q, F'$ *such that for all* $\overrightarrow{in}$

$$\left(\exists \overrightarrow{in'}.\, \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in'},\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}}\right) \equiv q * \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}.$$

*Proof.* The proof is by induction on the structure of the context $c$.

$c = -$: Choose $F' = F$ and choose $q = \mathrm{emp}$ if $\overrightarrow{out'} = \overrightarrow{out}$ and $q = \mathbf{False}$ otherwise. If $\overrightarrow{out'} \neq \overrightarrow{out}$ then both sides are equivalent to **False**, so assume that $\overrightarrow{out'} = \overrightarrow{out}$. Observe

$$\begin{aligned}\exists \overrightarrow{in'}.\, \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \langle\!\langle - \rangle\!\rangle^{\overrightarrow{in'},\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}} &\equiv \Cap{F}{\overrightarrow{in},\overrightarrow{out}} * \mathrm{emp} \\ &\equiv q * \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}.\end{aligned}$$

$c = n[c']$: By the inductive hypothesis, there exist $q', F'$ such that for all $\overrightarrow{in}$

$$\exists l.\, \Cap{\varepsilon,\varepsilon,\overrightarrow{out'}}{l,n} * \langle\!\langle c' \rangle\!\rangle^{l,n}_{\overrightarrow{in},\overrightarrow{out}} \equiv q' * \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}.$$

Choose $q = \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \overrightarrow{in'} \doteq n * q'$ and $F'$ as given. Observe

$$\begin{aligned}\exists \overrightarrow{in'}.\, \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \langle\!\langle n[c'] \rangle\!\rangle^{\overrightarrow{in'},\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}} &\equiv \exists \overrightarrow{in'}.\, \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \overrightarrow{in'} \doteq n * \exists l, i.\, n \mapsto \overrightarrow{out'}, i * i \Rightarrow [\,l\,] * \langle\!\langle c' \rangle\!\rangle^{l,n}_{\overrightarrow{in},\overrightarrow{out}} \\ &\equiv \exists \overrightarrow{in'}.\, \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \overrightarrow{in'} \doteq n * \exists l.\, \Cap{\varepsilon,\varepsilon,\overrightarrow{out'}}{l,n} * \langle\!\langle c' \rangle\!\rangle^{l,n}_{\overrightarrow{in},\overrightarrow{out}} \\ &\equiv \exists \overrightarrow{in'}.\, \Cap{F}{\overrightarrow{in'},\overrightarrow{out'}} * \overrightarrow{in'} \doteq n * q' * \Cap{F'}{\overrightarrow{in},\overrightarrow{out}} \\ &\equiv q * \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}.\end{aligned}$$

$c = t' \otimes c'$: Observe that there is exactly one choice of $l_1$ such that $\langle\!\langle t' \rangle\!\rangle^{l_1,\overrightarrow{out'}}$ is defined. Let $\hat{l}_1$ be that choice. Observe also that there exists a $q'$ such that

$$\langle\!\langle t' \rangle\!\rangle^{\hat{l}_1,\overrightarrow{out'}} \equiv q' * \prod_{n \in \hat{l}_1}^{*} n \mapsto \overrightarrow{out'}.$$

Let $(l'_1, l'_2, p') = F$. By the inductive hypothesis, there exist $q'', F'$ such that for all $\overrightarrow{in}$

$$\exists l_2.\, \Cap{l'_1+\hat{l}_1,l'_2,\overrightarrow{out'}}{l_2,n} * \langle\!\langle c' \rangle\!\rangle^{l_2,n}_{\overrightarrow{in},\overrightarrow{out}} \equiv q'' * \Cap{F'}{\overrightarrow{in},\overrightarrow{out}}.$$

Choose $q = q' * q''$ and $F'$ as given by the inductive hypothesis. Observe

$$\begin{aligned}
&\exists \overrightarrow{in'}.\, \Subset^{F}_{\overrightarrow{in'},\overrightarrow{out'}} * \langle\!\langle t' \otimes c' \rangle\!\rangle^{\overrightarrow{in'},\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}} \\
&\equiv * \left(\prod\nolimits^{*}_{n \in l'_1 + l'_2} n \mapsto \overrightarrow{out'}\right) * \langle\!\langle t' \rangle\!\rangle^{\hat{l}_1,\overrightarrow{out'}} * \exists l_1.\, \overrightarrow{in'} \doteq \hat{l}_1 + l_2 * \langle\!\langle c' \rangle\!\rangle^{l_2,\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}} \\
&\qquad \exists l_2.\, \exists i.\, \overrightarrow{out'} \mapsto p', i * i \Rrightarrow [\, l'_1 + \hat{l}_1 + l_2 + l'_2 \,] \\
&\equiv * \left(\prod\nolimits^{*}_{n \in l'_1 + l'_2} n \mapsto \overrightarrow{out'}\right) * q' * \left(\prod\nolimits^{*}_{n \in \hat{l}_1} n \mapsto \overrightarrow{out'}\right) * \langle\!\langle c' \rangle\!\rangle^{l_2,\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}} \\
&\equiv q' * \exists l_2.\, \Subset^{l'_1 + \hat{l}_1, l'_2, \overrightarrow{out}}_{l_2,\overrightarrow{out'}} * \langle\!\langle c' \rangle\!\rangle^{l_2,\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}} \\
&\equiv q' * q'' * \Subset^{F'}_{\overrightarrow{in},\overrightarrow{out}}.
\end{aligned}$$

The remaining case is proved in a similar fashion, and hence, for all $\overrightarrow{out'}, F, \overrightarrow{out}, c$ there exist $q, F'$ such that for all $\overrightarrow{in}$

$$\left(\exists \overrightarrow{in'}.\, \Subset^{F}_{\overrightarrow{in'},\overrightarrow{out'}} * \langle\!\langle c \rangle\!\rangle^{\overrightarrow{in'},\overrightarrow{out'}}_{\overrightarrow{in},\overrightarrow{out}}\right) \equiv q * \Subset^{F'}_{\overrightarrow{in},\overrightarrow{out}}.$$

□

### B.3 axiom correctness

We need to show that the high-level axioms for the abstract tree module are preserved by the list-based implementation. We do this in the presence of a specification environment which allows for recursive procedure calls.

Let the procedure environment $\Gamma$ be defined as,

$$\begin{aligned}
\Gamma ::= \{ \quad \texttt{getRight} &: \left(\lambda e.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[t] \otimes m[t'] \wedge (e = n) \rangle\!\rangle^{l,u}\right) \\
&\quad \to \left(\lambda v.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[t] \otimes m[t'] \wedge (v = m) \rangle\!\rangle^{l,u}\right) \\
\texttt{getRight} &: \left(\lambda e.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle m[t' \otimes n[t]] \wedge (e = n) \rangle\!\rangle^{l,u}\right) \\
&\quad \to \left(\lambda v.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle m[t' \otimes n[t]] \wedge (v = \textbf{\textit{null}}) \rangle\!\rangle^{l,u}\right) \\
\texttt{getLast} &: \left(\lambda e.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[t' \otimes m[t]] \wedge (e = n) \rangle\!\rangle^{l,u}\right) \\
&\quad \to \left(\lambda v.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[t' \otimes m[t]] \wedge (v = m) \rangle\!\rangle^{l,u}\right) \\
\texttt{getLast} &: \left(\lambda e.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[\varnothing] \wedge (e = n) \rangle\!\rangle^{l,u}\right) \\
&\quad \to \left(\lambda v.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[\varnothing] \wedge (v = \textbf{\textit{null}}) \rangle\!\rangle^{l,u}\right) \\
\texttt{deleteTree} &: \left(\lambda e.\, \exists l.\, \Subset^{F}_{l,u} * \langle\!\langle n[t] \wedge (e = n) \rangle\!\rangle^{l,u}\right) \\
&\quad \to \left(\exists l.\, \Subset^{F}_{l,u} * \langle\!\langle \varnothing \rangle\!\rangle^{l,u}\right) \\
\}
\end{aligned}$$

We need to show that the bodies of the low-level implementations for the high-level tree commands satisfy this procedure specification environment.

**Lemma 6 (getRight body correctness).** *The implementation of* `getRight` *given in § 5.1 satisfies the specification environment.*

$$\Gamma \vdash \begin{array}{c} \left\{\exists e, l.\, \circledR^F_{l,u} * \langle\!\langle n[t] \otimes m[t'] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow e * \boldsymbol{n}' \Rightarrow -\right\} \\ \texttt{getRight}_{body} \\ \left\{\exists v, l.\, \circledR^F_{l,u} * \langle\!\langle n[t] \otimes m[t'] \wedge (v = m) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow - * \boldsymbol{n}' \Rightarrow v\right\} \end{array}$$

$$\Gamma \vdash \begin{array}{c} \left\{\exists e, l.\, \circledR^F_{l,u} * \langle\!\langle m[t' \otimes n[t]] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow e * \boldsymbol{n}' \Rightarrow -\right\} \\ \texttt{getRight}_{body} \\ \left\{\exists v, l.\, \circledR^F_{l,u} * \langle\!\langle m[t' \otimes n[t]] \wedge (v = \boldsymbol{null}) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow - * \boldsymbol{n}' \Rightarrow v\right\} \end{array}$$

*Proof.* There are two cases to prove. In the first case the node $n$ has a right sibling. Let $F = l_1, l_2, u'$.

$$\begin{array}{l} \left\{\exists e, l.\, \circledR^F_{l,u} * \langle\!\langle n[t] \otimes m[t'] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow e * \boldsymbol{n}' \Rightarrow -\right\} \\ \left\{\begin{array}{l} \exists i, l', j.\, u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * \left(\prod^{*}_{x \in l_1 + l_2} x \mapsto u\right) \\ * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l',n} * \langle\!\langle m[t'] \rangle\!\rangle^{m,u} \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow - \end{array}\right\} \\ \{u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * n \mapsto u, i \times \boldsymbol{n} \Rightarrow n * {}' \Rightarrow -\} \\ \texttt{local } \boldsymbol{x}, \boldsymbol{y} \texttt{ in} \\ \quad \left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * n \mapsto u, i \\ \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow - * \boldsymbol{x} \Rightarrow - * \boldsymbol{y} \Rightarrow - \end{array}\right\} \\ \quad \boldsymbol{x} := [\boldsymbol{n}.\texttt{parent}]\,; \\ \quad \left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * n \mapsto u, i \\ \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow - * \boldsymbol{x} \Rightarrow u * \boldsymbol{y} \Rightarrow - \end{array}\right\} \\ \quad \boldsymbol{y} := [\boldsymbol{x}.\texttt{children}]\,; \\ \quad \left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * n \mapsto u, i \\ \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow - * \boldsymbol{x} \Rightarrow u * \boldsymbol{y} \Rightarrow j \end{array}\right\} \\ \quad \boldsymbol{n}' := \boldsymbol{y}.\texttt{getNext}(\boldsymbol{n}) \\ \quad \left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * n \mapsto u, i \\ \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow m * \boldsymbol{x} \Rightarrow u * \boldsymbol{y} \Rightarrow j \end{array}\right\} \\ \{u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * n \mapsto u, i \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow m\} \\ \left\{\begin{array}{l} \exists i, l', j.\, u \mapsto u', j * j \Mapsto [\, l_1 + n + m + l_2 \,] * \left(\prod^{*}_{x \in l_1 + l_2} x \mapsto u\right) \\ * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l',n} * \langle\!\langle m[t'] \rangle\!\rangle^{m,u} \times \boldsymbol{n} \Rightarrow n * \boldsymbol{n}' \Rightarrow m \end{array}\right\} \\ \left\{\exists v, l.\, \circledR^F_{l,u} * \langle\!\langle n[t] \otimes m[t'] \wedge (v = m) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow - * \boldsymbol{n}' \Rightarrow v\right\} \end{array}$$

In the second case the node $n$ does not have a right sibling. Let $F = l_1, l_2, u'$.

$$
\begin{array}{l}
\left\{\exists e, l.\, \circledR^F_{l,u} * \langle\!\langle m[t' \otimes n[t]] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e * \boldsymbol{n}' \Rrightarrow -\right\} \\
\left\{\begin{array}{l}
\exists i, l', j, l'', k.\, u \mapsto u', k * k \Mapsto [\, l_1 + m + l_2 \,] * \left(\prod_{x \in l_1 + l_2}^{*} x \mapsto u\right) \\
* m \mapsto u, i * i \Mapsto [\, l' + n \,] * \langle\!\langle t' \rangle\!\rangle^{l', m} * n \mapsto m, j * j \Mapsto [\, l'' \,] * \langle\!\langle t \rangle\!\rangle^{l'', n} \\
\times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow -
\end{array}\right\} \\
\quad \{m \mapsto u, i * i \Mapsto [\, l' + n \,] * n \mapsto m, j \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow -\} \\
\quad \texttt{local } \boldsymbol{x}, \boldsymbol{y} \texttt{ in} \\
\qquad \left\{\begin{array}{l}
m \mapsto u, i * i \Mapsto [\, l' + n \,] * n \mapsto m, j \\
\times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow - * \boldsymbol{x} \Rrightarrow - * \boldsymbol{y} \Rrightarrow -
\end{array}\right\} \\
\qquad \boldsymbol{x} := [\boldsymbol{n}.\texttt{parent}]\,; \\
\qquad \{m \mapsto u, i * i \Mapsto [\, l' + n \,] * n \mapsto m, j \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow - * \boldsymbol{x} \Rrightarrow m * \boldsymbol{y} \Rrightarrow -\} \\
\qquad \boldsymbol{y} := [\boldsymbol{x}.\texttt{children}]\,; \\
\qquad \{m \mapsto u, i * i \Mapsto [\, l' + n \,] * n \mapsto m, j \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow - * \boldsymbol{x} \Rrightarrow m * \boldsymbol{y} \Rrightarrow i\} \\
\qquad \boldsymbol{n}' := \boldsymbol{y}.\texttt{getNext}(\boldsymbol{n}) \\
\qquad \{m \mapsto u, i * i \Mapsto [\, l' + n \,] * n \mapsto m, j \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow \boldsymbol{null} * \boldsymbol{x} \Rrightarrow m * \boldsymbol{y} \Rrightarrow i\} \\
\quad \{m \mapsto u, i * i \Mapsto [\, l' + n \,] * n \mapsto m, j \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow \boldsymbol{null}\} \\
\left\{\begin{array}{l}
\exists i, l', j, l'', k.\, u \mapsto u', k * k \Mapsto [\, l_1 + m + l_2 \,] * \left(\prod_{x \in l_1 + l_2}^{*} x \mapsto u\right) \\
* m \mapsto u, i * i \Mapsto [\, l' + n \,] * \langle\!\langle t' \rangle\!\rangle^{l', m} * n \mapsto m, j * j \Mapsto [\, l'' \,] * \langle\!\langle t \rangle\!\rangle^{l'', n} \\
\times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n}' \Rrightarrow \boldsymbol{null}
\end{array}\right\} \\
\left\{\exists v, l.\, \circledR^F_{l,u} * \langle\!\langle m[t' \otimes n[t]] \wedge (v = \boldsymbol{null}) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow - * \boldsymbol{n}' \Rrightarrow v\right\}
\end{array}
$$

□

**Lemma 7 (getLast body correctness).** *The implementation of* `getLast` *given in § 5.1 satisfies the specification environment.*

$$
\Gamma \vdash \begin{array}{c}
\left\{\exists e, l.\, \circledR^F_{l,u} * \langle\!\langle n[t' \otimes m[t]] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e * \boldsymbol{n}' \Rrightarrow -\right\} \\
\texttt{getLast}_{body} \\
\left\{\exists v, l.\, \circledR^F_{l,u} * \langle\!\langle n[t' \otimes m[t]] \wedge (v = m) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow - * \boldsymbol{n}' \Rrightarrow v\right\}
\end{array}
$$

$$
\Gamma \vdash \begin{array}{c}
\left\{\exists e, l.\, \circledR^F_{l,u} * \langle\!\langle n[\varnothing] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e * \boldsymbol{n}' \Rrightarrow -\right\} \\
\texttt{getLast}_{body} \\
\left\{\exists v, l.\, \circledR^F_{l,u} * \langle\!\langle n[\varnothing] \wedge (v = \boldsymbol{null}) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow - * \boldsymbol{n}' \Rrightarrow v\right\}
\end{array}
$$

*Proof.* There are two cases to prove. In the first case the node $n$ has at least one child. Let $F = l_1, l_2, u'$.

$$
\begin{array}{l}
\left\{\exists e, l.\, \circledcirc^F_{l,u} * \langle\!\langle n[t' \otimes m[t]] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e * \boldsymbol{n'} \Rrightarrow -\right\} \\
\left\{\exists l, i, l'.\, \circledcirc^F_{l,u} * n \mapsto u{,}i * i \Mapsto [\, l' + m \,] * \langle\!\langle t' \rangle\!\rangle^{l',n} * \langle\!\langle m[t] \rangle\!\rangle^{m,n} \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow -\right\} \\
\quad \{n \mapsto u{,}i * i \Mapsto [\, l' + m \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow -\} \\
\quad \texttt{local } \boldsymbol{x} \texttt{ in} \\
\qquad \{n \mapsto u{,}i * i \Mapsto [\, l' + m \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow - * \boldsymbol{x} \Rrightarrow -\} \\
\qquad \boldsymbol{x} := [\boldsymbol{n}.\texttt{children}]\,; \\
\qquad \{n \mapsto u{,}i * i \Mapsto [\, l' + m \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow - * \boldsymbol{x} \Rrightarrow i\} \\
\qquad \boldsymbol{n'} := \boldsymbol{x}.\texttt{getTail}() \\
\qquad \{n \mapsto u{,}i * i \Mapsto [\, l' + m \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow m * \boldsymbol{x} \Rrightarrow i\} \\
\quad \{n \mapsto u{,}i * i \Mapsto [\, l' + m \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow m\} \\
\left\{\exists l, i, l'.\, \circledcirc^F_{l,u} * n \mapsto u{,}i * i \Mapsto [\, l' + m \,] * \langle\!\langle t' \rangle\!\rangle^{l',n} * \langle\!\langle m[t] \rangle\!\rangle^{m,n} \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow m\right\} \\
\left\{\exists v, l.\, \circledcirc^F_{l,u} * \langle\!\langle n[t' \otimes m[t]] \wedge (v = m) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow - * \boldsymbol{n'} \Rrightarrow v\right\}
\end{array}
$$

In the second case the node $n$ does not have any children. Let $F = l_1, l_2, u'$.

$$
\begin{array}{l}
\left\{\exists e, l.\, \circledcirc^F_{l,u} * \langle\!\langle n[\varnothing] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e * \boldsymbol{n'} \Rrightarrow -\right\} \\
\left\{\exists l, i.\, \circledcirc^F_{l,u} * n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow -\right\} \\
\quad \{n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow -\} \\
\quad \texttt{local } \boldsymbol{x} \texttt{ in} \\
\qquad \{n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow - * \boldsymbol{x} \Rrightarrow -\} \\
\qquad \boldsymbol{x} := [\boldsymbol{n}.\texttt{children}]\,; \\
\qquad \{n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow - * \boldsymbol{x} \Rrightarrow i\} \\
\qquad \boldsymbol{n'} := \boldsymbol{x}.\texttt{getTail}() \\
\qquad \{n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow \boldsymbol{null} * \boldsymbol{x} \Rrightarrow i\} \\
\quad \{n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow \boldsymbol{null}\} \\
\left\{\exists l, i.\, \circledcirc^F_{l,u} * n \mapsto u{,}i * i \Mapsto [\, \varepsilon \,] \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{n'} \Rrightarrow \boldsymbol{null}\right\} \\
\left\{\exists v, l.\, \circledcirc^F_{l,u} * \langle\!\langle n[\varnothing] \wedge (v = \boldsymbol{null}) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow - * \boldsymbol{n'} \Rrightarrow v\right\}
\end{array}
$$

□

**Lemma 8 (deleteTree body correctness).** *The implementation of* `deleteTree` *given in §5.1 satisfies the procedure specification environment.*

$$
\Gamma \vdash \begin{array}{c}
\left\{\exists e, l.\, \circledcirc^F_{l,u} * \langle\!\langle n[t] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e\right\} \\
\texttt{deleteTree}_{body} \\
\left\{\exists l.\, \circledcirc^F_{l,u} * \langle\!\langle \varnothing \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow -\right\}
\end{array}
$$

*Proof.* Let $F = l_1, l_2, u'$.

$$\left\{\exists e, l.\, \Subset^F_{l,u} * \langle\!\langle n[t] \wedge (e = n) \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rrightarrow e\right\}$$

$$\left\{\begin{array}{l} \exists i, l', j.\, u \mapsto u', j * j \Mapsto [\, l_1 + n + l_2 \,] * \left(\prod_{x \in l_1 + l_2}^{*} x \mapsto u\right) \\ * \, n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \times \boldsymbol{n} \Rrightarrow n \end{array}\right\}$$

$$\left\{u \mapsto u', j * j \Mapsto [\, l_1 + n + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \times \boldsymbol{n} \Rrightarrow n\right\}$$

`local` $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ `in`

$$\left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow - * \boldsymbol{y} \Rrightarrow - * \boldsymbol{z} \Rrightarrow - \end{array}\right\}$$

$\boldsymbol{x} := [\boldsymbol{n}.\texttt{parent}]$ ;

$$\left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow - * \boldsymbol{z} \Rrightarrow - \end{array}\right\}$$

$\boldsymbol{y} := [\boldsymbol{x}.\texttt{children}]$ ;

$$\left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + n + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow j * \boldsymbol{z} \Rrightarrow - \end{array}\right\}$$

$\boldsymbol{y}.\texttt{remove}(\boldsymbol{n})$ ;

$$\left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow j * \boldsymbol{z} \Rrightarrow - \end{array}\right\}$$

$\boldsymbol{y} := [\boldsymbol{n}.\texttt{children}]$ ;

$$\left\{\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l' \,] * \langle\!\langle t \rangle\!\rangle^{l', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow - \end{array}\right\}$$

$$\left\{\begin{array}{l} \left(\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] \\ * \, n \mapsto u, i * i \Mapsto [\, \varepsilon \,] \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u \\ * \, \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow - \end{array}\right) \vee \left(\begin{array}{l} \exists m, t', t'', l''. \\ u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i \\ * \, i \Mapsto [\, m + l'' \,] * \langle\!\langle m[t'] \otimes t'' \rangle\!\rangle^{m + l'', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow - \end{array}\right) \end{array}\right\}$$

$\boldsymbol{z} := \boldsymbol{y}.\texttt{getHead}()$ ;

$$\left\{\begin{array}{l} \left(\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] \\ * \, n \mapsto u, i * i \Mapsto [\, \varepsilon \,] \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u \\ * \, \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow \boldsymbol{null} \end{array}\right) \vee \left(\begin{array}{l} \exists m, t', t'', l''. \\ u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i \\ * \, i \Mapsto [\, m + l'' \,] * \langle\!\langle m[t'] \otimes t'' \rangle\!\rangle^{m + l'', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow m \end{array}\right) \end{array}\right\}$$

`while` $z \neq \boldsymbol{null}$ `do`

$$\left\{\begin{array}{l} \exists m, t', t'', l''.\, u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i * i \Mapsto [\, m + l'' \,] \\ * \, \langle\!\langle m[t'] \otimes t'' \rangle\!\rangle^{m + l'', n} \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow m \end{array}\right\}$$

`call deleteTree(`$\boldsymbol{z}$`)` ;

$$\left\{\begin{array}{l} \exists m, t'', l''.\, u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i * i \Mapsto [\, l'' \,] * \langle\!\langle t'' \rangle\!\rangle^{l'', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow m \end{array}\right\}$$

$$\left\{\begin{array}{l} \left(\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] \\ * \, n \mapsto u, i * i \Mapsto [\, \varepsilon \,] \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u \\ * \, \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow - \end{array}\right) \vee \left(\begin{array}{l} \exists m, t', t'', l''. \\ u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i \\ * \, i \Mapsto [\, m + l'' \,] * \langle\!\langle m[t'] \otimes t'' \rangle\!\rangle^{m + l'', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow - \end{array}\right) \end{array}\right\}$$

$\boldsymbol{z} := \boldsymbol{y}.\texttt{getHead}()$

$$\left\{\begin{array}{l} \left(\begin{array}{l} u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] \\ * \, n \mapsto u, i * i \Mapsto [\, \varepsilon \,] \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u \\ * \, \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow \boldsymbol{null} \end{array}\right) \vee \left(\begin{array}{l} \exists m, t', t'', l''. \\ u \mapsto u', j * j \Mapsto [\, l_1 + l_2 \,] * n \mapsto u, i \\ * \, i \Mapsto [\, m + l'' \,] * \langle\!\langle m[t'] \otimes t'' \rangle\!\rangle^{m + l'', n} \\ \times \boldsymbol{n} \Rrightarrow n * \boldsymbol{x} \Rrightarrow u * \boldsymbol{y} \Rrightarrow i * \boldsymbol{z} \Rrightarrow m \end{array}\right) \end{array}\right\}$$

$$\left\{\begin{array}{l} u \mapsto u', j * j \Rightarrow [\, l_1 + l_2 \,] * n \mapsto u, i * i \Rightarrow [\, \varepsilon \,] \\ \times\, \boldsymbol{n} \Rightarrow n * \boldsymbol{x} \Rightarrow u * \boldsymbol{y} \Rightarrow i * \boldsymbol{z} \Rightarrow \boldsymbol{null} \end{array}\right\}$$
$$\texttt{disposeList}(\boldsymbol{y})\,;$$
$$\{u \mapsto u', j * j \Rightarrow [\, l_1 + l_2 \,] * n \mapsto u, i \times \boldsymbol{n} \Rightarrow n * \boldsymbol{x} \Rightarrow u * \boldsymbol{y} \Rightarrow i * \boldsymbol{z} \Rightarrow \boldsymbol{null}\}$$
$$\texttt{disposeNode}(\boldsymbol{n})$$
$$\{u \mapsto u', j * j \Rightarrow [\, l_1 + l_2 \,] \times \boldsymbol{n} \Rightarrow n * \boldsymbol{x} \Rightarrow u * \boldsymbol{y} \Rightarrow i * \boldsymbol{z} \Rightarrow \boldsymbol{null}\}$$
$$\{u \mapsto u', j * j \Rightarrow [\, l_1 + l_2 \,] \times \boldsymbol{n} \Rightarrow n\}$$
$$\left\{\exists i, l', j.\, u \mapsto u', j * j \Rightarrow [\, l_1 + l_2 \,] * \left(\prod_{x \in l_1 + l_2}^{*} x \mapsto u\right) \times \boldsymbol{n} \Rightarrow n\right\}$$
$$\left\{\exists l.\, \Cap_{l,u}^{F} * \langle\!\langle \varnothing \rangle\!\rangle^{l,u} \times \boldsymbol{n} \Rightarrow -\right\}$$

□

Finally, we observe that for all $u, F$ and $(p, \overrightarrow{r} := \mathtt{f}(\overrightarrow{E}), q) \in \mathrm{Ax}_{\mathbb{T}}$

$$\Gamma \vdash \{(\!|p|\!)^{u,F}\}\ \mathtt{call}\ \overrightarrow{r} := \mathtt{f}(\overrightarrow{E})\ \{(\!|q|\!)^{u,F}\}$$

where $(\!|p|\!)^{u,F} = \bigvee_{(d,\sigma)\in p} \exists l.\, \Cap_{l,u}^{F} * \langle\!\langle d \rangle\!\rangle^{l,u} \times \sigma$. This follows directly from the PCall rule and the definition of $\Gamma$.

## C Correctness of the Locality-breaking Theory

Assume that we are given:

- abstract modules $\mathbb{A}$ and $\mathbb{B}$;
- a substitutive implementation function $[\![-]\!] : \mathcal{L}_{\mathbb{A}} \to \mathcal{L}_{\mathbb{B}}$;
- a pointwise predicate translation function $[\![-]\!] : \mathcal{P}(\mathcal{D}_{\mathbb{A}} \times \Sigma) \to \mathcal{P}(\mathcal{D}_{\mathbb{B}} \times \Sigma)$; and
- for every $(p, \varphi, q) \in \mathrm{Ax}_{\mathbb{A}}$ and $c \in \mathcal{C}_{\mathbb{A}}$, a derivation of $\vdash_{\mathbb{B}} \{[\![\{c\} \circ p]\!]\}\ [\![\varphi]\!]\ \{[\![\{c\} \circ q]\!]\}$.

We wish to establish the following:

**Proposition 2.** *For all $p, q \in \mathcal{P}(\mathcal{A}_{\mathbb{A}} \times \Sigma)$ and $\mathbb{C} \in \mathcal{L}_{\mathbb{A}}$*

$$\Gamma \vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\} \quad \Longrightarrow \quad [\![\Gamma]\!] \vdash_{\mathbb{B}} \{[\![p]\!]\}\ [\![\mathbb{C}]\!]\ \{[\![q]\!]\}\,,$$

*where*

$$[\![\Gamma]\!] = \{\mathtt{f} : [\![P]\!] \to [\![Q]\!] \mid (\mathtt{f} : [\![P]\!] \to [\![Q]\!]) \in \mathrm{Ax}_{\mathbb{A}}\}\,.$$

To do so, we shall use the frame-elimination lemma previously described, which we prove in detail below.

**Lemma 1 (Frame Free).** *Suppose that $\mathbb{A}$ is an extension with $\mathcal{A}_{\mathbb{A}}$ a left-cancellative context algebra. If there is a derivation of $\vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\}$ then there*

*is also a derivation that only uses the frame rule in the following ways:*

$$\dfrac{\dfrac{}{\Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}\ (\dagger)}{\Gamma \vdash \{f \circ p\}\ \mathbb{C}\ \{f \circ q\}}\ \textsc{Frame} \tag{2}$$

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}}{\Gamma \vdash \{(\mathbf{I}_{\mathbb{A}} \times f_{\mathbb{V}}) \circ p\}\ \mathbb{C}\ \{(\mathbf{I}_{\mathbb{A}} \times f_{\mathbb{V}}) \circ q\}}\ \textsc{Frame} \tag{3}$$

*where* (†) *is either an axiom of* $\textsc{Ax}_{\mathbb{A}}$*,* Skip *or* Assgn*.*

*Proof.* We show a more general result, that for a derivation of $\Gamma \vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\}$ there is a derivation of $F(\Gamma) \vdash_{\mathbb{A}} \{p\}\ \mathbb{C}\ \{q\}$ with the required property, where

$$F(\Gamma) = \{\mathtt{f} : (f \circ P) \to (f \circ Q) \mid f \in \mathcal{P}(\mathcal{C}_{\mathbb{A}}), (\mathtt{f} : P \to Q) \in \Gamma)\}.$$

Clearly, $\Gamma \subseteq F(\Gamma) = F(F(\Gamma))$. Since the procedure environment (and it's transformation) are only relevant to the PDef and PCall rules, we omit them when considering the other rules.

The proof is by induction on the structure of the proof. If the last rule of the proof is not Frame or PDef then it is simple to transform the proof: transform the proofs of the premises by induction and simply apply the last rule with $F(\Gamma)$ in place of $\Gamma$.

Consider case when the frame rule is the last rule applied:

$$\dfrac{\dfrac{\vdots}{\{p\}\,\mathbb{C}\,\{q\}}\ (\ddagger)}{\{f \circ p\}\,\mathbb{C}\,\{f \circ q\}}\ \textsc{Frame}$$

By applying the disjunction rule, we can reduce the problem to the case of singleton frames $\{c\}$, transforming the proof as follows:

$$\dfrac{\forall c \in f \quad \dfrac{\dfrac{\vdots}{\{p\}\,\mathbb{C}\,\{q\}}\ (\ddagger)}{\{\{c\} \circ p\}\,\mathbb{C}\,\{\{c\} \circ q\}}\ \textsc{Frame}}{\{f \circ p\}\,\mathbb{C}\,\{f \circ q\}}\ \textsc{Disj}$$

We now consider cases on (‡), the last rule applied before the frame rule.

If the rule is Cons then, since $p \subseteq q$ implies that $\{c\} \circ p \subseteq \{c\} \circ q$, we can move the application of the frame rule earlier in the proof as follows:

$$\dfrac{\{c\} \circ p \subseteq \{c\} \circ p' \quad \dfrac{\dfrac{\vdots}{\{p'\}\,\mathbb{C}\,\{q'\}}}{\{\{c\} \circ p'\}\,\mathbb{C}\,\{\{c\} \circ q'\}}\ \textsc{Frame} \quad \{c\} \circ q \subseteq \{c\} \circ q'}{\{\{c\} \circ p\}\,\mathbb{C}\,\{\{c\} \circ q\}}\ \textsc{Cons}$$

The application of the frame rule can then be removed by induction.

If the rule is Disj then, since $\circ$ is right-distributive over $\vee$, the proof can be transformed as follows:

$$\dfrac{\forall i \in I \quad \dfrac{\vdots}{\{p_i\}\,\mathbb{C}\,\{q_i\}}}{\{\{c\} \circ \bigvee_{i\in I} p_i\}\,\mathbb{C}\,\{\{c\} \circ \bigvee_{i\in I} q_i\}}\ \begin{array}{l}\textsc{Frame}\\ \textsc{Disj}\end{array}$$

The applications of the frame rule can then be removed by induction.

If the rule is Conj then we make use of the left-cancellation property, which implies that $a \in \{c\} \circ \bigwedge_{i\in I} p_i$ if and only if $a \in \bigwedge_{i\in I} \{c\} \circ p_i$. We can transform the proof as follows:

$$\dfrac{\forall i \in I \quad \dfrac{\vdots}{\{p_i\}\,\mathbb{C}\,\{q_i\}}}{\{\{c\} \circ \bigwedge_{i\in I} p_i\}\,\mathbb{C}\,\{\{c\} \circ \bigwedge_{i\in I} q_i\}}\ \begin{array}{l}\textsc{Frame}\\ \textsc{Conj}\end{array}$$

The applications of the frame rule can then be removed by induction.

If the rule is Local then it is possible that the frame $c$ includes a program variable with the same name as one that is scoped by the `local` block. This means we cannot in general push the frame into the local block. Note that, for some $c_{\mathbb{A}} \in \mathcal{D}_{\mathbb{A}}$ and $c_{\mathbb{V}} \in \Sigma$, $\{c\} = \{(c_{\mathbb{A}}, c_{\mathbb{V}})\} = (\mathbf{I}_{\mathbb{A}} \times \{c_{\mathbb{V}}\}) \bullet \{(c_{\mathbb{A}}, \emptyset)\}$. Hence we can transform the proof as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\{(\mathbf{I}_{\mathbb{A}} \times \boldsymbol{v} \Rrightarrow -) \circ p\}\ \mathbb{C}'\ \{(\mathbf{I}_{\mathbb{A}} \times \boldsymbol{v} \Rrightarrow -) \circ q\}}\ \textsc{Frame}}{\{(\{c_{\mathbb{A}}\} \times \boldsymbol{v} \Rrightarrow -) \circ p\}\ \mathbb{C}'\ \{(\{c_{\mathbb{A}}\} \times \boldsymbol{v} \Rrightarrow -) \circ q\}}\ \textsc{Local}}{\{\{(c_{\mathbb{A}}, \emptyset)\} \circ p\}\ \texttt{local}\ \boldsymbol{v}\ \texttt{in}\ \mathbb{C}'\ \{\{(c_{\mathbb{A}}, \emptyset)\} \circ q\}}\ \textsc{Frame}}{\{\{c\} \circ p\}\ \texttt{local}\ \boldsymbol{v}\ \texttt{in}\ \mathbb{C}'\ \{\{c\} \circ q\}}\ \textsc{Frame}$$

The side-condition for the Local rule, that $(\mathbf{I}_{\mathbb{A}} \times \boldsymbol{v} \Rrightarrow -) \circ \{(c_{\mathbb{A}}, \emptyset)\} \circ p \neq \emptyset$, follows from the original side-condition that $(\mathbf{I}_{\mathbb{A}} \times \boldsymbol{v} \Rrightarrow -) \circ p \neq \emptyset$. The applications of the frame rule are either of the form of (3) or can be removed by induction.

If the rule is PCall then we again consider the frame context in terms of its two components, i.e. $c = (c_{\mathbb{A}}, c_{\mathbb{V}})$ for some $c_{\mathbb{A}} \in \mathcal{D}_{\mathbb{A}}$ and $c_{\mathbb{V}} \in \Sigma$. The PCall rule used some $(\mathtt{f} : P \to Q) \in \Gamma$. By definition, $(\mathtt{f} : (\{c_{\mathbb{A}}\} \circ P) \to (\{c_{\mathbb{A}}\} \circ Q)) \in F(\Gamma)$.

Hence we can transform the proof as follows:

$$\dfrac{\dfrac{}{F(\Gamma) \vdash \left\{\begin{array}{c}(\{c_{\mathbb{A}}\} \circ P([\![\overrightarrow{E}]\!]_{\rho[\overrightarrow{y} \mapsto \overrightarrow{v}]})) \times (\rho * \overrightarrow{y} \Rrightarrow \overrightarrow{v}) \\ \texttt{call } \overrightarrow{y} := \texttt{f}(\overrightarrow{E}) \\ \{\exists \overrightarrow{w}.\, (\{c_{\mathbb{A}}\} \circ Q(\overrightarrow{w})) \times (\rho * \overrightarrow{y} \Rrightarrow \overrightarrow{w})\}\end{array}\right\}}\ \textsc{PCall}}{F(\Gamma) \vdash \left\{\begin{array}{c}\{(c_{\mathbb{A}}, c_{\mathbb{V}})\} \circ (P([\![\overrightarrow{E}]\!]_{\rho[\overrightarrow{y} \mapsto \overrightarrow{v}]}) \times (\rho * \overrightarrow{y} \Rrightarrow \overrightarrow{v})) \\ \texttt{call } \overrightarrow{y} := \texttt{f}(\overrightarrow{E}) \\ \{\{(c_{\mathbb{A}}, c_{\mathbb{V}})\} \circ (\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\rho * \overrightarrow{y} \Rrightarrow \overrightarrow{w}))\}\end{array}\right\}}\ \textsc{Frame}$$

The application of the frame rule is of the form of (3) with the frame $\mathbf{I}_{\mathbb{A}} \times \{c_{\mathbb{V}}\}$.

The cases for the remaining rules, corresponding to program constructs, are straight-forward.

Consider case when PDef is the last rule applied:

$$\dfrac{\forall (\texttt{f}_i : P \to Q) \in \Gamma.\ \Gamma', \Gamma \vdash \begin{array}{c}\vdots \\ \{\exists \overrightarrow{v}.\, P(\overrightarrow{v}) \times (\overrightarrow{x} \Rrightarrow \overrightarrow{v} * \overrightarrow{r} \Rrightarrow -)\} \\ \mathbb{C}_i \\ \{\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{x} \Rrightarrow - * \overrightarrow{r} \Rrightarrow \overrightarrow{w})\}\end{array} \qquad \dfrac{\vdots}{\Gamma', \Gamma \vdash \{p\}\ \mathbb{C}\ \{q\}}}{\Gamma' \vdash \{p\}\ \texttt{procs } \overrightarrow{r} := \texttt{f}_1(\overrightarrow{x})\{\mathbb{C}_1\}, \ldots, \overrightarrow{r} := \texttt{f}_k(\overrightarrow{x})\{\mathbb{C}_k\}\ \texttt{in}\ \mathbb{C}\ \{q\}}\ \textsc{PDef}$$

The proofs of the function bodies can be extended by applying the frame rule to give:

$$\forall \begin{array}{l} f \in \mathcal{P}(\mathcal{C}_{\mathbb{A}}), \\ (\texttt{f}_i : P \to Q) \in \Gamma \end{array}.\ \dfrac{\Gamma', \Gamma \vdash \begin{array}{c}\vdots \\ \{\exists \overrightarrow{v}.\, P(\overrightarrow{v}) \times (\overrightarrow{x} \Rrightarrow \overrightarrow{v} * \overrightarrow{r} \Rrightarrow -)\} \\ \mathbb{C}_i \\ \{\exists \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{x} \Rrightarrow - * \overrightarrow{r} \Rrightarrow \overrightarrow{w})\}\end{array}}{\Gamma', \Gamma \vdash \begin{array}{c}\{\exists \overrightarrow{v}.\, (f \circ P(\overrightarrow{v})) \times (\overrightarrow{x} \Rrightarrow \overrightarrow{v} * \overrightarrow{r} \Rrightarrow -)\} \\ \mathbb{C}_i \\ \{\exists \overrightarrow{w}.\, (f \circ Q(\overrightarrow{w})) \times (\overrightarrow{x} \Rrightarrow - * \overrightarrow{r} \Rrightarrow \overrightarrow{w})\}\end{array}}\ \textsc{Frame}$$

These proofs, and the proof of the remaining premise, can be transformed by induction so that they only use the frame rule in the required manner and use the procedure environment $F(\Gamma, \Gamma') = F(\Gamma), F(\Gamma')$. These proofs can then be

recombined to give the required proof transformation:

$$\dfrac{\dfrac{\dfrac{\vdots}{\forall(\mathtt{f}_i : P \to Q) \in F(\Gamma).\ F(\Gamma', \Gamma) \vdash \begin{array}{r}\{\exists\, \overrightarrow{v}.\, P(\overrightarrow{v}) \times (\overrightarrow{\boldsymbol{x}} \Rrightarrow \overrightarrow{v} * \overrightarrow{\boldsymbol{r}} \Rrightarrow -)\} \\ \mathbb{C}_i \\ \{\exists\, \overrightarrow{w}.\, Q(\overrightarrow{w}) \times (\overrightarrow{\boldsymbol{x}} \Rrightarrow - * \overrightarrow{\boldsymbol{r}} \Rrightarrow \overrightarrow{w})\}\end{array}}}{(\star)} \quad \dfrac{\vdots}{F(\Gamma', \Gamma) \vdash \{p\}\ \mathbb{C}\ \{q\}}\ (\star)}{F(\Gamma') \vdash \{p\}\ \mathtt{procs}\ \overrightarrow{\boldsymbol{r}} := \mathtt{f}_1(\overrightarrow{\boldsymbol{x}})\{\mathbb{C}_1\}, \ldots, \overrightarrow{\boldsymbol{r}} := \mathtt{f}_k(\overrightarrow{\boldsymbol{x}})\{\mathbb{C}_k\}\ \mathtt{in}\ \mathbb{C}\ \{q\}}\ \textsc{PDef}$$

□

*Proof (Proposition 2).* Suppose that $\Gamma \vdash_{\mathbb{A}} \{p\}\,\mathbb{C}\,\{q\}$. We first apply Lemma 1 to translate the proof into a frame-free proof. This can be converted into a proof of $[\![\Gamma]\!] \vdash_{\mathbb{B}} \{[\![p]\!]\}\,[\![\mathbb{C}]\!]\,\{[\![q]\!]\}$ by a straightforward inductive argument: each framed axiom is replaced by the derivation of its translation, and each inference rule is replaced by its low-level equivalent, since the translation preserves the necessary properties. □

This completes the proof of Theorem 4.

## D Correctness of the List Implementation

In the following section we show that the selected implementations for commands of our abstract list module are correct. We do this following the general theory for locality breaking translations laid out in § 6. We need to show that each procedure implementation satisfies the high-level specification for that procedure, in any context. We do this in the presence of a specification environment which allows for recursive procedure calls.

Let the procedure environment $\Gamma$ be defined as,

$$\begin{aligned}\Gamma ::= \{\ &\mathtt{getNext} : (\lambda e_1, e_2.\, [\![f \circ i \Mapsto v' + u \wedge (e_1 = i) \wedge (e_2 = v')]\!]) \\ &\qquad \to (\lambda v.\, [\![f \circ i \Mapsto v' + u \wedge (v = u)]\!])\,, \\ &\mathtt{getNext} : (\lambda e_1, e_2.\, [\![f \circ i \Mapsto [\,l + v'\,] \wedge (e_1 = i) \wedge (e_2 = v')]\!]) \\ &\qquad \to (\lambda v.\, [\![f \circ i \Mapsto [\,l + v'\,] \wedge (v = \boldsymbol{null})]\!])\,, \\ &\mathtt{remove} : (\lambda e_1, e_2.\, [\![f \circ i \Mapsto v \wedge (e_1 = i) \wedge (e_2 = v)]\!]) \\ &\qquad \to ([\![f \circ i \Mapsto \varepsilon]\!])\,, \\ \}&\end{aligned}$$

We need to show that the bodies of the implementations for the high-level list commands satisfy this procedure specification environment.

**Lemma 9 (getNext body correctness).** *The implementation of* `getNext` *given in §6.1 satisfies the procedure specification environment.*

$$\Gamma \vdash \begin{array}{c} \left\{ \begin{array}{l} \exists e_1, e_2.\, [\![ f \circ i \Mapsto v' + u \wedge (e_1 = i) \wedge (e_2 = v') ]\!] \\ \times \boldsymbol{i} \Rrightarrow e_1 * \boldsymbol{v'} \Rrightarrow e_2 * \boldsymbol{v} \Rrightarrow - \end{array} \right\} \\ \texttt{getNext}_{body} \\ \left\{ \begin{array}{l} \exists v.\, [\![ f \circ i \Mapsto v' + u \wedge (v = u) ]\!] \\ \times \boldsymbol{i} \Rrightarrow - * \boldsymbol{v'} \Rrightarrow - * \boldsymbol{v} \Rrightarrow v \end{array} \right\} \end{array}$$

$$\Gamma \vdash \begin{array}{c} \left\{ \begin{array}{l} \exists e_1, e_2.\, [\![ f \circ i \Mapsto [\, l + v' \,] \wedge (e_1 = i) \wedge (e_2 = v') ]\!] \\ \times \boldsymbol{i} \Rrightarrow e_1 * \boldsymbol{v'} \Rrightarrow e_2 * \boldsymbol{v} \Rrightarrow - \end{array} \right\} \\ \texttt{getNext}_{body} \\ \left\{ \begin{array}{l} \exists v.\, [\![ f \circ i \Mapsto [\, l + v' \,] \wedge (v = \boldsymbol{null}) ]\!] \\ \times \boldsymbol{i} \Rrightarrow - * \boldsymbol{v'} \Rrightarrow - * \boldsymbol{v} \Rrightarrow v \end{array} \right\} \end{array}$$

*Proof.* There are two cases to prove. In the first case the value $v'$ is not the last value in list $i$. We can assume that the context $f$ at least takes the list $i$ to complete list. If it does not, then the precondition will be equivalent to **False** and correctness if trivial. So let the singleton $f = i \Mapsto [\, l_1 + - + l_2 \,] * ls$ for some lists $l_1$, $l_2$ and a list store $ls$ consisting only of complete lists. Note that $v' \notin l_1$, since elements within list are unique, so in particular $\forall v \in l_1.\, v \neq v'$. We make use of this fact when testing for equality with $v'$.

$$\{\exists e_1, e_2.\, [\![f \circ i \Mapsto v' + u \wedge (e_1 = i) \wedge (e_2 = v')]\!] \times \boldsymbol{i} \Rightarrow e_1 * \boldsymbol{v'} \Rightarrow e_2 * \boldsymbol{v} \Rightarrow -\}$$

$$\left\{ \begin{array}{l} \exists p, x, y, z.\, i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y * y \mapsto u, z * \langle\!\langle l_2 \rangle\!\rangle^{(z,\mathbf{null})} * [\![ls]\!] \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow - \end{array} \right\}$$

$$\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y * y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow -\}$$

`local` $\boldsymbol{x}$ `in`

$$\left\{ \begin{array}{l} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y * y \mapsto u, z \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow - \end{array} \right\}$$

$\boldsymbol{x} := [\boldsymbol{i}]$ ;

$$\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y * y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow p\}$$

$$\left\{ \left( \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto x * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow x \end{array} \right) \vee \left( \begin{array}{l} \exists v, a, l'.\, (l_1 \doteq v + l') * i \mapsto p \\ * \, p \mapsto v, a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow p \end{array} \right) \right\}$$

$\boldsymbol{v} := [\boldsymbol{x}.\mathtt{value}]$ ;

$$\left\{ \left( \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto x * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{array} \right) \vee \left( \begin{array}{l} \exists v, a, l'.\, (l_1 \doteq v + l') * i \mapsto p \\ * \, p \mapsto v, a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow p \end{array} \right) \right\}$$

$$\left\{ \left( \begin{array}{l} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{array} \right) \vee \left( \begin{array}{l} \exists l', a, v, b, l''.\, (l_1 \doteq l' + v + l'') \\ * \, i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b \\ * \, \langle\!\langle l'' \rangle\!\rangle^{(b,x)} * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow a \end{array} \right) \right\}$$

`while` $\boldsymbol{v} \neq \boldsymbol{v'}$ `do`

$$\left\{ \begin{array}{l} \exists l', a, v, b, l''.\, (l_1 \doteq l' + v + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * \langle\!\langle l'' \rangle\!\rangle^{(b,x)} \\ * \, x \mapsto v', y * y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow a \end{array} \right\}$$

$\boldsymbol{x} := [\boldsymbol{x}.\mathtt{next}]$ ;

$$\left\{ \begin{array}{l} \exists l', a, v, b, l''.\, (l_1 \doteq l' + v + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * \langle\!\langle l'' \rangle\!\rangle^{(b,x)} \\ * \, x \mapsto v', y * y \mapsto u, z \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow b \end{array} \right\}$$

$$\left\{ \left( \begin{array}{l} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i \\ * \, \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow x \end{array} \right) \vee \left( \begin{array}{l} \exists l', a, v, b, v'', c, l''. \\ (l_1 \doteq l' + v + v'' + l'') * i \mapsto p \\ * \, \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * b \mapsto v'', c \\ * \, \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v', y * y \mapsto u, z \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow b \end{array} \right) \right\}$$

$\boldsymbol{v} := [\boldsymbol{x}.\mathtt{value}]$

$$\left\{ \left( \begin{array}{l} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v', y \\ * \, y \mapsto u, z \times \boldsymbol{i} \Rightarrow i \\ * \, \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{array} \right) \vee \left( \begin{array}{l} \exists l', a, v, b, v'', c, l''. \\ (l_1 \doteq l' + v + v'' + l'') * i \mapsto p \\ * \, \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * b \mapsto v'', c \\ * \, \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v', y * y \mapsto u, z \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \, \boldsymbol{v} \Rightarrow v'' * \boldsymbol{x} \Rightarrow b \end{array} \right) \right\}$$

$$\left\{\begin{pmatrix} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v',y \\ * \, y \mapsto u,z \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' \\ * \, \boldsymbol{v} \Rrightarrow v' * \boldsymbol{x} \Rrightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists l', a, v, b, l''. (l_1 \doteq l' + v + l'') \\ * \, i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v,b \\ * \, \langle\!\langle l'' \rangle\!\rangle^{(b,x)} * x \mapsto v',y \\ * \, y \mapsto u,z \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' \\ * \, \boldsymbol{v} \Rrightarrow v * \boldsymbol{x} \Rrightarrow a \end{pmatrix}\right\}$$

$$\left\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v',y * y \mapsto u,z \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' * \boldsymbol{v} \Rrightarrow v' * \boldsymbol{x} \Rrightarrow x\right\}$$

$\boldsymbol{x} := [\boldsymbol{x}.\texttt{next}]$ ;

$$\left\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v',y * y \mapsto u,z \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' * \boldsymbol{v} \Rrightarrow v' * \boldsymbol{x} \Rrightarrow y\right\}$$

$\texttt{if } \boldsymbol{x} = \boldsymbol{null} \texttt{ then ... else } \boldsymbol{v} := [\boldsymbol{x}.\texttt{value}]$

$$\left\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v',y * y \mapsto u,z \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' * \boldsymbol{v} \Rrightarrow u * \boldsymbol{x} \Rrightarrow y\right\}$$

$$\left\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v',y * y \mapsto u,z \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' * \boldsymbol{v} \Rrightarrow u\right\}$$

$$\left\{\begin{array}{l} \exists p, x, y, z.\, i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v',y * y \mapsto u,z * \langle\!\langle l_2 \rangle\!\rangle^{(z,\boldsymbol{null})} * [\![ls]\!] \\ \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v'} \Rrightarrow v' * \boldsymbol{v} \Rrightarrow u \end{array}\right\}$$

$$\{\exists v.\, [\![f \circ i \Mapsto v' + u \wedge (v = u)]\!] \times \boldsymbol{i} \Rrightarrow - * \boldsymbol{v'} \Rrightarrow - * \boldsymbol{v} \Rrightarrow v\}$$

In the second case the value $v'$ is the last value in list $i$. In this case the list $i$ is already a complete list, so let the singleton $f = ls$ for some list store $ls$ consisting only of complete lists that do not include $i$. As before, if $ls$ contains a list $i$, or any of the list is incomplete, then the precondition will be equivalent to **False** and the proof is trivial. Again note that $v' \notin l$, since elements within list are unique, so in particular $\forall v \in l.\, v \neq v'$. We make use of this fact when

testing for equality with $v'$.

$$\{\exists e_1, e_2.\, [\![ f \circ i \Mapsto [\, l + v' \,] \wedge (e_1 = i) \wedge (e_2 = v') ]\!] \times \boldsymbol{i} \Rightarrow e_1 * \boldsymbol{v'} \Rightarrow e_2 * \boldsymbol{v} \Rightarrow -\}$$

$$\{\exists p, x.\, i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} * [\![ ls ]\!] \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow -\}$$

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow -\}$$

`local` $\boldsymbol{x}$ `in`

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow -\}$$

$\boldsymbol{x} := [\boldsymbol{i}]$ ;

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow p\}$$

$$\left\{ \begin{pmatrix} (l \doteq \varepsilon) * i \mapsto x \\ * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists v, a, l'.\, (l \doteq v + l') * i \mapsto p \\ * p \mapsto v, a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} \\ * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i \\ * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow - * \boldsymbol{x} \Rightarrow p \end{pmatrix} \right\}$$

$\boldsymbol{v} := [\boldsymbol{x}.\mathtt{value}]$ ;

$$\left\{ \begin{pmatrix} (l \doteq \varepsilon) * i \mapsto x \\ * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists v, a, l'.\, (l \doteq v + l') * i \mapsto p \\ * p \mapsto v, a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} \\ * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i \\ * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow p \end{pmatrix} \right\}$$

$$\left\{ \begin{pmatrix} i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} \\ * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists l', a, v, b, l''.\, (l \doteq l' + v + l'') \\ * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b \\ * \langle\!\langle l'' \rangle\!\rangle^{(b,x)} * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow a \end{pmatrix} \right\}$$

`while` $\boldsymbol{v} \neq \boldsymbol{v'}$ `do`

$$\left\{ \begin{array}{l} \exists l', a, v, b, l''.\, (l \doteq l' + v + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * \langle\!\langle l'' \rangle\!\rangle^{(b,x)} \\ * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow a \end{array} \right\}$$

$\boldsymbol{x} := [\boldsymbol{x}.\mathtt{next}]$ ;

$$\left\{ \begin{array}{l} \exists l', a, v, b, l''.\, (l \doteq l' + v + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * \langle\!\langle l'' \rangle\!\rangle^{(b,x)} \\ * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow b \end{array} \right\}$$

$$\left\{ \begin{pmatrix} i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} \\ * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists l', a, v, b, v'', c, l''. \\ (l \doteq l' + v + v'' + l'') * i \mapsto p \\ * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * b \mapsto v'', c \\ * \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i \\ * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow b \end{pmatrix} \right\}$$

$\boldsymbol{v} := [\boldsymbol{x}.\mathtt{value}]$

$$\left\{ \begin{pmatrix} i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} \\ * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists l', a, v, b, v'', c, l''. \\ (l \doteq l' + v + v'' + l'') * i \mapsto p \\ * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b * b \mapsto v'', c \\ * \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i \\ * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v'' * \boldsymbol{x} \Rightarrow b \end{pmatrix} \right\}$$

$$\left\{ \begin{pmatrix} i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} \\ * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x \end{pmatrix} \vee \begin{pmatrix} \exists l', a, v, b, l''.\, (l \doteq l' + v + l'') \\ * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v, b \\ * \langle\!\langle l'' \rangle\!\rangle^{(b,x)} * x \mapsto v', \boldsymbol{null} \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' \\ * \boldsymbol{v} \Rightarrow v * \boldsymbol{x} \Rightarrow a \end{pmatrix} \right\}$$

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow x\}$$

$\boldsymbol{x} := [\boldsymbol{x}.\mathtt{next}]$ ;

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow v' * \boldsymbol{x} \Rightarrow \boldsymbol{null}\}$$

`if` $\boldsymbol{x} = \boldsymbol{null}$ `then` $\boldsymbol{v} := \boldsymbol{x}$ `else` ...

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow \boldsymbol{null} * \boldsymbol{x} \Rightarrow \boldsymbol{null}\}$$

$$\{i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow \boldsymbol{null}\}$$

$$\{\exists p, x.\, i \mapsto p * \langle\!\langle l \rangle\!\rangle^{(p,x)} * x \mapsto v', \boldsymbol{null} * [\![ ls ]\!] \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v'} \Rightarrow v' * \boldsymbol{v} \Rightarrow \boldsymbol{null}\}$$

$$\{\exists v.\, [\![ f \circ i \Mapsto [\, l + v' \,] \wedge (v = \boldsymbol{null}) ]\!] \times \boldsymbol{i} \Rightarrow - * \boldsymbol{v'} \Rightarrow - * \boldsymbol{v} \Rightarrow v\}$$

□

**Lemma 10 (Remove Body Correctness).** *The implementation of* `remove` *given in §6.1 satisfies the procedure specification environment.*

$$\Gamma \vdash \begin{array}{c} \left\{ \exists e_1, e_2.\, [\![ f \circ i \Mapsto v \wedge (e_1 = i) \wedge (e_2 = v) ]\!] \times \boldsymbol{i} \Rrightarrow e_1 * \boldsymbol{v} \Rrightarrow e_2 \right\} \\ \texttt{remove}_{body} \\ \{ [\![ f \circ i \Mapsto \varepsilon ]\!] \times \boldsymbol{i} \Rrightarrow - * \boldsymbol{v} \Rrightarrow - \} \end{array}$$

*Proof.* We can assume that the context $f$ at least takes the list $i$ to complete list. If it does not, then the precondition will be equivalent to **False** and correctness if trivial. So let the singleton $f = i \Mapsto [\, l_1 + - + l_2 \,] * ls$ for some lists $l_1$, $l_2$ and a list store $ls$ consisting only of complete lists. Note that $v' \notin l_1$, since elements within list are unique, so in particular $\forall v \in l_1.\, v \neq v'$. We make use of this fact

when testing for equality with $v'$.

$$
\begin{array}{l}
\{\exists e_1, e_2.\, [\![ f \circ i \Rightarrow v \wedge (e_1 = i) \wedge (e_2 = v) ]\!] \times \boldsymbol{i} \Rightarrow e_1 * \boldsymbol{v} \Rightarrow e_2\} \\
\big\{\exists p, x, y.\, i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v,y * \langle\!\langle l_2 \rangle\!\rangle^{(y,\mathbf{null})} * [\![ ls ]\!] \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v\big\} \\
\quad \big\{i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v,y \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v\big\} \\
\quad \texttt{local } \boldsymbol{u}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \texttt{ in} \\
\qquad \left\{ \begin{array}{l} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow - * \boldsymbol{x} \Rightarrow - * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad \boldsymbol{x} := [\boldsymbol{i}]\,; \\
\qquad \left\{ \begin{array}{l} i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,x)} * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow - * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad \left\{ \left( \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto x * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow - * \boldsymbol{x} \Rightarrow x \\ * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists v', a, l'.\, (l_1 \doteq v' + l') * i \mapsto p \\ * p \mapsto v', a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow - \\ * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\qquad \boldsymbol{u} := [\boldsymbol{x}.\texttt{value}]\,; \\
\qquad \left\{ \left( \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto x * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow x \\ * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists v', a, l'.\, (l_1 \doteq v' + l') * i \mapsto p \\ * p \mapsto v', a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v' \\ * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\qquad \boldsymbol{y} := [\boldsymbol{x}.\texttt{next}]\,; \\
\qquad \left\{ \left( \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto x * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow x \\ * \boldsymbol{y} \Rightarrow y * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists v', a, l'.\, (l_1 \doteq v' + l') * i \mapsto p \\ * p \mapsto v', a * \langle\!\langle l' \rangle\!\rangle^{(a,x)} * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v' \\ * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow a * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\qquad \texttt{if } \boldsymbol{u} = \boldsymbol{v} \texttt{ then} \\
\qquad\quad \left\{ \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto x * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow x * \boldsymbol{y} \Rightarrow y * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad\quad [\boldsymbol{i}] := \boldsymbol{y}\,; \\
\qquad\quad \left\{ \begin{array}{l} (l_1 \doteq \varepsilon) * i \mapsto y * x \mapsto v,y \\ \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow x * \boldsymbol{y} \Rightarrow y * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad\quad \texttt{disposeNode}(\boldsymbol{x}) \\
\qquad\quad \big\{ (l_1 \doteq \varepsilon) * i \mapsto y \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow x * \boldsymbol{y} \Rightarrow y * \boldsymbol{z} \Rightarrow - \big\} \\
\qquad\quad \big\{ i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,y)} \times \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow - * \boldsymbol{y} \Rightarrow - * \boldsymbol{z} \Rightarrow - \big\}
\end{array}
$$

$$
\begin{array}{l}
\texttt{else} \\
\quad \left\{ \begin{array}{l} \exists v', a, l'.\, (l_1 \doteq v' + l') * i \mapsto p * p \mapsto v', a * \langle\!\langle l' \rangle\!\rangle^{a,x} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v' * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow a * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\quad \left\{ \left( \begin{array}{l} \exists v'.\, (l_1 \doteq v') * i \mapsto p \\ * p \mapsto v', x * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v' * \boldsymbol{x} \Rightarrow p \\ * \boldsymbol{y} \Rightarrow x * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists v', a, v'', b, l'.\, (l_1 \doteq v' + v'' + l') \\ * i \mapsto p * p \mapsto v', a * a \mapsto v'', b \\ * \langle\!\langle l' \rangle\!\rangle^{(b,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v' \\ * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow a * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\quad \boldsymbol{u} := [\boldsymbol{y}.\texttt{value}]\ ; \\
\quad \left\{ \left( \begin{array}{l} \exists v'.\, (l_1 \doteq v') * i \mapsto p \\ * p \mapsto v', x * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow p \\ * \boldsymbol{y} \Rightarrow x * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists v', a, v'', b, l'.\, (l_1 \doteq v' + v'' + l') \\ * i \mapsto p * p \mapsto v', a * a \mapsto v'', b \\ * \langle\!\langle l' \rangle\!\rangle^{(b,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v'' \\ * \boldsymbol{x} \Rightarrow p * \boldsymbol{y} \Rightarrow a * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\quad \left\{ \left( \begin{array}{l} \exists l', a, v'.\, (l_1 \doteq l' + v') \\ * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} \\ * a \mapsto v', x * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow a \\ * \boldsymbol{y} \Rightarrow x * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists l', a, v', b, v'', c, l''. \\ (l_1 \doteq l' + v' + v'' + l'') * i \mapsto p \\ * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', b * b \mapsto v'', c \\ * \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v'' \\ * \boldsymbol{x} \Rightarrow a * \boldsymbol{y} \Rightarrow b * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\quad \texttt{while}\ \boldsymbol{u} \neq \boldsymbol{v}\ \texttt{do} \\
\qquad \left\{ \begin{array}{l} \exists l', a, v', b, v'', c, l''.\, (l_1 \doteq l' + v' + v'' + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} \\ * a \mapsto v', b * b \mapsto v'', c * \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v'' * \boldsymbol{x} \Rightarrow a * \boldsymbol{y} \Rightarrow b * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad \boldsymbol{x} := \boldsymbol{y}\ ; \\
\qquad \left\{ \begin{array}{l} \exists l', a, v', b, v'', c, l''.\, (l_1 \doteq l' + v' + v'' + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} \\ * a \mapsto v', b * b \mapsto v'', c * \langle\!\langle l'' \rangle\!\rangle^{c,x} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v'' * \boldsymbol{x} \Rightarrow b * \boldsymbol{y} \Rightarrow b * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad \boldsymbol{y} := [\boldsymbol{x}.\texttt{next}]\ ; \\
\qquad \left\{ \begin{array}{l} \exists l', a, v', b, v'', c, l''.\, (l_1 \doteq l' + v' + v'' + l'') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} \\ * a \mapsto v', b * b \mapsto v'', c * \langle\!\langle l'' \rangle\!\rangle^{(c,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v'' * \boldsymbol{x} \Rightarrow b * \boldsymbol{y} \Rightarrow c * \boldsymbol{z} \Rightarrow - \end{array} \right\} \\
\qquad \left\{ \left( \begin{array}{l} \exists l', a, v', b, v''. \\ (l_1 \doteq l' + v' + v'') * i \mapsto p \\ * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', b \\ * b \mapsto v'', x * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v'' * \boldsymbol{x} \Rightarrow b \\ * \boldsymbol{y} \Rightarrow x * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists l', a, v', b, v'', c, v''', d, l''. \\ (l_1 \doteq l' + v' + v'' + v''' + l'') \\ * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', b \\ * b \mapsto v'', c * c \mapsto v''', d \\ * \langle\!\langle l'' \rangle\!\rangle^{(d,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v'' \\ * \boldsymbol{x} \Rightarrow b * \boldsymbol{y} \Rightarrow c * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\} \\
\qquad \boldsymbol{u} := [\boldsymbol{y}.\texttt{value}] \\
\qquad \left\{ \left( \begin{array}{l} \exists l', a, v', b, v''. \\ (l_1 \doteq l' + v' + v'') * i \mapsto p \\ * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', b \\ * b \mapsto v'', x * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v \\ * \boldsymbol{u} \Rightarrow v * \boldsymbol{x} \Rightarrow b \\ * \boldsymbol{y} \Rightarrow x * \boldsymbol{z} \Rightarrow - \end{array} \right) \vee \left( \begin{array}{l} \exists l', a, v', b, v'', c, v''', d, l''. \\ (l_1 \doteq l' + v' + v'' + v''' + l'') \\ * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', b \\ * b \mapsto v'', c * c \mapsto v''', d \\ * \langle\!\langle l'' \rangle\!\rangle^{(d,x)} * x \mapsto v, y \\ \times\ \boldsymbol{i} \Rightarrow i * \boldsymbol{v} \Rightarrow v * \boldsymbol{u} \Rightarrow v''' \\ * \boldsymbol{x} \Rightarrow b * \boldsymbol{y} \Rightarrow c * \boldsymbol{z} \Rightarrow - \end{array} \right) \right\}
\end{array}
$$

$$\left\{ \begin{pmatrix} \exists l', a, v'.\, (l_1 \doteq l' + v') \\ * \, i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} \\ * \, a \mapsto v', x * x \mapsto v, y \\ \times \; \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v \\ * \, \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow a \\ * \, \boldsymbol{y} \Rrightarrow x * \boldsymbol{z} \Rrightarrow - \end{pmatrix} \vee \begin{pmatrix} \exists l', a, v', b, v'', c, l''. \\ (l_1 \doteq l' + v' + v'' + l'') * i \mapsto p \\ * \, \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', b * b \mapsto v'', c \\ * \, \langle\!\langle l'' \rangle\!\rangle^{c,x} * x \mapsto v, y \\ \times \; \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v'' \\ * \, \boldsymbol{x} \Rrightarrow a * \boldsymbol{y} \Rrightarrow b * \boldsymbol{z} \Rrightarrow - \end{pmatrix} \right\}$$

$$\left\{ \exists l', a, v'.\, (l_1 \doteq l' + v') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', x * x \mapsto v, y \;\middle|\; \times \; \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow a * \boldsymbol{y} \Rrightarrow x * \boldsymbol{z} \Rrightarrow - \right\}$$

$\boldsymbol{z} := [\boldsymbol{y}.\mathtt{next}]$ ;

$$\left\{ \exists l', a, v'.\, (l_1 \doteq l' + v') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', x * x \mapsto v, y \;\middle|\; \times \; \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow a * \boldsymbol{y} \Rrightarrow x * \boldsymbol{z} \Rrightarrow y \right\}$$

$[\boldsymbol{x}.\mathtt{next}] := \boldsymbol{z}$ ;

$$\left\{ \exists l', a, v'.\, (l_1 \doteq l' + v') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', y * x \mapsto v, y \;\middle|\; \times \; \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow a * \boldsymbol{y} \Rrightarrow x * \boldsymbol{z} \Rrightarrow y \right\}$$

$\mathtt{disposeNode}(\boldsymbol{y})$

$$\left\{ \exists l', a, v'.\, (l_1 \doteq l' + v') * i \mapsto p * \langle\!\langle l' \rangle\!\rangle^{(p,a)} * a \mapsto v', y \;\middle|\; \times \; \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow a * \boldsymbol{y} \Rrightarrow x * \boldsymbol{z} \Rrightarrow y \right\}$$

$$\left\{ i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,y)} \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow - * \boldsymbol{y} \Rrightarrow - * \boldsymbol{z} \Rrightarrow - \right\}$$

$$\left\{ i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,y)} \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v * \boldsymbol{u} \Rrightarrow v * \boldsymbol{x} \Rrightarrow - * \boldsymbol{y} \Rrightarrow - * \boldsymbol{z} \Rrightarrow - \right\}$$

$$\left\{ i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,y)} \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v \right\}$$

$$\left\{ \exists p, x, y.\, i \mapsto p * \langle\!\langle l_1 \rangle\!\rangle^{(p,y)} * \langle\!\langle l_2 \rangle\!\rangle^{(y,\boldsymbol{null})} * [\![ls]\!] \times \boldsymbol{i} \Rrightarrow i * \boldsymbol{v} \Rrightarrow v \right\}$$

$$\{ [\![f \circ i \Mapsto \varepsilon]\!] \times \boldsymbol{i} \Rrightarrow - * \boldsymbol{v} \Rrightarrow - \}$$

□

Finally, we observe that for all $(p, \overrightarrow{\boldsymbol{r}} := \mathtt{f}(\overrightarrow{\boldsymbol{E}}), q) \in \mathrm{Ax}_{\mathbb{T}}$

$$\Gamma \vdash \{[\![p]\!]\}\ \mathtt{call}\ \overrightarrow{\boldsymbol{r}} := \mathtt{f}(\overrightarrow{\boldsymbol{E}})\ \{[\![q]\!]\}$$

This follows directly from the PCall rule and the definition of $\Gamma$.