Counting Immutable Beans - Appendix

Reference Counting Optimized for Purely Functional Programming

SEBASTIAN ULLRICH, Karlsruhe Institute of Technology, Germany
LEONARDO DE MOURA, Microsoft Research, USA

Most functional languages rely on some kind of garbage collection for automatic memory management. They usually eschew reference counting in favor of a tracing garbage collector, which has less bookkeeping overhead at runtime. On the other hand, having an exact reference count of each value can enable optimizations such as destructive updates. We explore these optimization opportunities in the context of an eager, purely functional programming language. We propose a new mechanism for efficiently reclaiming memory used by nonshared values, reducing stress on the global memory allocator. We describe an approach for minimizing the number of reference counts updates using borrowed references and a heuristic for automatically inferring borrow annotations. We implemented all these techniques in a new compiler for an eager and purely functional programming language with support for multi-threading. Our preliminary experimental results demonstrate our approach is competitive and often outperforms state-of-the-art compilers.

Additional Key Words and Phrases: purely functional programming, reference counting, Lean

A APPENDIX

A.1 Pure semantics

For the sake of completeness, we give a semantics specification on λ_{pure} in addition to the λ_{RC} semantics given in the paper.

$$\begin{aligned} \rho \in \mathit{Ctxt} &= \mathit{Var} \rightharpoonup \mathit{Value} \\ v \in \mathit{Value} &::= \mathbf{ctor}_i\ \overline{v} \mid \mathbf{pap}\ c\ \overline{v} \end{aligned}$$

$$\textsc{Const-App-Full}\quad \frac{\delta(c) = \lambda\, \overline{y_c}.\ F \qquad \overline{v} = \overline{\rho(y)} \qquad [\overline{y_c} \mapsto \overline{v}] \vdash F \Downarrow v'}{\rho \vdash c\ \overline{y} \Downarrow v'}$$

$$\textsc{Const-App-Part}\quad \frac{\delta(c) = \lambda\, \overline{y_c}.\ F \qquad \overline{v} = \overline{\rho(y)} \qquad |\ \overline{v}\ | < |\ \overline{y_c}\ |}{\rho \vdash \mathbf{pap}\ c\ \overline{y} \Downarrow \mathbf{pap}\ c\ \overline{v}}$$

$$\textsc{Var-App-Full}\quad \frac{\rho(x) = \mathbf{pap}\ c\ \overline{v} \qquad \delta(c) = \lambda\, \overline{y_c}.\ F \qquad v' = \rho(y) \qquad [\overline{y_c} \mapsto \overline{v}\ v'] \vdash F \Downarrow v''}{\rho \vdash x\ y \Downarrow v''}$$

Authors' addresses: Sebastian Ullrich, Karlsruhe Institute of Technology, Germany, sebastian.ullrich@kit.edu; Leonardo de Moura, Microsoft Research, USA, leonardo@microsoft.com.

2020. XXXX-XXXX/2020/9-ART $15.00
https://doi.org/

$$\textsc{Var-App-Part}\ \frac{\rho(x) = \mathbf{pap}\ c\ \overline{v} \qquad \delta(c) = \lambda\ \overline{y_c}.\ F \qquad v' = \rho(y) \qquad |\ \overline{v}\ v'\ | < |\ \overline{y_c}\ |}{\rho \vdash x\ y \Downarrow \mathbf{pap}\ c\ \overline{v}\ v'}$$

$$\textsc{Ctor-App}\ \frac{\overline{v} = \overline{\rho(y)}}{\rho \vdash \mathbf{ctor}_i\ \overline{y} \Downarrow \mathbf{ctor}_i\ \overline{v}} \qquad \textsc{Proj}\ \frac{\rho(x) = \mathbf{ctor}_j\ \overline{v} \qquad v' = v_i}{\rho \vdash \mathbf{proj}_i\ x \Downarrow v'} \qquad \textsc{Return}\ \frac{\rho(x) = v}{\rho \vdash \mathbf{ret}\ x \Downarrow v}$$

$$\textsc{Let}\ \frac{\rho \vdash e \Downarrow v \qquad \rho[x \mapsto v] \vdash F \Downarrow v'}{\rho \vdash \mathbf{let}\ x = e;\ F \Downarrow v'} \qquad \textsc{Case}\ \frac{\rho(x) = \mathbf{ctor}_i\ \overline{v} \qquad \rho \vdash F_i \Downarrow v'}{\rho \vdash \mathbf{case}\ x\ \mathbf{of}\ \overline{F} \Downarrow v'}$$

We furthermore extend the pure semantics to λ_{RC} in order to express *semantic refinement* for our compiler passes that only change the RC semantics of a program.

$$\textsc{Inc}\ \frac{\rho \vdash F \Downarrow v}{\rho \vdash \mathbf{inc}\ x;\ F \Downarrow v} \qquad \textsc{Dec}\ \frac{\rho \vdash F \Downarrow v}{\rho \vdash \mathbf{dec}\ x;\ F \Downarrow v} \qquad \textsc{Reset}\ \frac{}{\rho \vdash \mathbf{reset}\ x \Downarrow v} \qquad \textsc{Reuse}\ \frac{\rho \vdash \mathbf{ctor}_i\ \overline{y} \Downarrow v}{\rho \vdash \mathbf{reuse}\ x\ \mathbf{in}\ \mathbf{ctor}_i\ \overline{y} \Downarrow v}$$

Definition 1. *We say δ_B* refines *δ_A in the pure semantics if for each constant c with $\delta_A(c) = \lambda\ \overline{y}.\ F$, we have $\delta_B(c) = \lambda\ \overline{y'}.\ F'$ and*

$$[\overline{y} \mapsto \overline{v}] \vdash F \Downarrow v \iff [\overline{y'} \mapsto \overline{v}] \vdash F' \Downarrow v$$

Note that there are no assertions about constants in δ_B but not in δ_A.

A.2 Well-formedness

Definition 2 (well-formedness of pure programs). *Used variables should be defined, and defined variables should be used. Applications should be of the correct arity. Bindings should be fresh.*

$$\frac{\forall c \in \mathrm{dom}(\delta).\ \delta \vdash_{pure} c}{\vdash_{pure} \delta} \qquad \frac{\delta(c) = \lambda\ \overline{y}.\ F \qquad \overline{y} \vdash_{pure} F}{\delta \vdash_{pure} c}$$

$$\frac{}{\Gamma, x \vdash_{pure} \mathbf{ret}\ x} \qquad \frac{\overline{\Gamma, x \vdash_{pure} F}}{\Gamma, x \vdash_{pure} \mathbf{case}\ x\ \mathbf{of}\ \overline{F}} \qquad \frac{\Gamma \vdash_{pure} e \qquad z \in FV(F) \qquad z \notin \Gamma \qquad \Gamma, z \vdash_{pure} F}{\Gamma \vdash_{pure} \mathbf{let}\ z = e;\ F}$$

$$\frac{\delta(c) = \lambda\ \overline{y_c}.\ F' \qquad |\ \overline{y}\ | = |\ \overline{y_c}\ |}{\Gamma, \overline{y} \vdash_{pure} c\ \overline{y}} \qquad \frac{}{\Gamma, \overline{y} \vdash_{pure} \mathbf{pap}\ c\ \overline{y}} \qquad \frac{}{\Gamma, x, y \vdash_{pure} x\ y}$$

$$\frac{}{\Gamma, \overline{y} \vdash_{pure} \mathbf{cnstr}_i\ \overline{y}} \qquad \frac{}{\Gamma, x \vdash_{pure} \mathbf{proj}_i\ x}$$

We will assume $\vdash_{pure} \delta$.

Theorem 1. *If $\Gamma \vdash_{pure} F$, then $\mathrm{FV}(F) \subseteq \Gamma$.*

Proof. By induction over F. □

A.3 reset/reuse

Definition 3 (Well-formedness of reset/reuse instructions). Reset *variables form a separate, affine context* Δ*. They are introduced by* **reset** *and may be consumed by* **reuse***.* **dec** *instructions introduced later will turn the context* linear*, as postulated by the full type system below.*

$$\frac{\forall c \in \mathrm{dom}(\delta).\ \delta \vdash_{reuse} c}{\vdash_{reuse} \delta} \qquad \frac{\delta(c) = \lambda\, \overline{y}.\ F \qquad \overline{y};\cdot \vdash_{reuse} F}{\delta \vdash_{reuse} c}$$

$$\frac{}{\Gamma, x; \Delta \vdash_{reuse} \mathbf{ret}\ x} \qquad \frac{\overline{\Gamma, x; \Delta \vdash_{reuse} F}}{\Gamma, x; \Delta \vdash_{reuse} \mathbf{case}\ x\ \mathbf{of}\ \overline{F}}$$

$$\frac{\Gamma \vdash_{pure} e \qquad z \in FV(F) \qquad z \notin \Gamma\ \Delta \qquad \Gamma, z; \Delta \vdash_{reuse} F}{\Gamma; \Delta \vdash_{reuse} \mathbf{let}\ z = e;\ F}$$

$$\frac{z \notin \Gamma\ \Delta \qquad \Gamma; \Delta, z \vdash_{reuse} F}{\Gamma; \Delta \vdash_{reuse} \mathbf{let}\ z = \mathbf{reset}\ e;\ F}$$

$$\frac{\Gamma; \Delta \vdash_{reuse} F}{\Gamma; \Delta, x \vdash_{reuse} \mathbf{let}\ z = \mathbf{reuse}\ x\ \mathbf{in}\ \mathbf{ctor}_i\ \overline{y};\ F}$$

Theorem 2. *For the specific* δ_{reuse} *described in the paper, we have* $\vdash_{reuse} \delta_{reuse}$*. Moreover,* δ_{reuse} *refines* δ *in the pure semantics.*

Theorem 3. *If* $\Gamma; \Delta \vdash_{reuse} F$*, then* $\mathrm{FV}(F) \subseteq \Gamma\ \Delta$*.*

Proof. By induction over F. □

A.4 Borrow inference

Definition 4 (Program after borrow inference). *We assume that for every constant* $c \in \delta_{reuse}$ *there exists an unused constant name* $c_{\mathbb{O}}$*.*

$$\delta_\beta = \delta'_{reuse}[c_{\mathbb{O}} \mapsto \lambda\, \overline{y}.\ c\ \overline{y} \mid \mathbb{B} \in \beta(c), \delta_{reuse}(c) = \lambda\, \overline{y}.\ F]$$

where δ'_{reuse} *is obtained from* δ_{reuse} *by replacing every occurrence of* **pap** $c\ \overline{y}$ *where* $\mathbb{B} \in \beta(c)$ *by* **pap** $c_{\mathbb{O}}\ \overline{y}$*.*

Definition 5 (Well-formedness of borrow inference). β *is arity-correct. Partially applied constants do not have borrowed parameters.*

$$\frac{\vdash_{reuse} \delta \qquad \delta \vdash_\beta c \quad \forall c \in \mathrm{dom}(\delta)}{\vdash_\beta \delta} \qquad \frac{\delta(c) = \lambda\, \overline{y}.\ F \qquad |\ \overline{y}\ | = |\ \beta(c)\ | \qquad \vdash_\beta F}{\delta \vdash_\beta c}$$

$$\frac{\mathbb{B} \notin \beta(c) \qquad \vdash_\beta F}{\vdash_\beta \mathbf{let}\ x = \mathbf{pap}\ c\ \overline{y};\ F} \qquad \frac{}{\vdash_\beta \mathbf{ret}\ x}$$

The rules for all other instructions proceed by direct induction on F *or* $\overline{F}$*.*

Theorem 4. *For* δ_β *from Definition 4 and any arity-correct* β*, we have* $\vdash_\beta \delta_\beta$*. Moreover,* δ_β *refines* δ_{reuse} *in the pure semantics.*

A.5 A type system for RC-correct programs

A program's behavior should not be changed by compiling it to (or optimizing it in) λ_{RC}. Before designing the compiler, it is helpful to capture the global, dynamic invariants necessary for this in a static type system that reasons about just the local context of a function.

Intuitively, a program is RC-correct if

(1) owned variables are *locally count-correct*: every owned variable is ultimately consumed or **dec**ed on each control flow path, with every **inc** allowing and necessitating one more consumption,
(2) values are not freed while borrowed, and
(3) values from **reset** are handled by exactly one **reuse** or **dec** on each control flow path, and are not used in any other context.

The second constraint deserves further elaboration: we will assume that variables are only borrowed when passed to borrowed parameters, in which case we assume that the borrowed variable is valid for the entire function call, and no

The type system formalizing these constraints is quite simple: since types have been erased from λ_{pure}, the only types are $\mathbb{O}$, $\mathbb{B}$, and $\mathbb{R}$ for owned, borrowed, and reset references, respectively.

$$\tau \in Ty ::= \mathbb{O} \mid \mathbb{B} \mid \mathbb{R}$$

The type system is *linear* to represent conditions (1) and (3); for *borrowed* references, we add the usual weakening and contraction rules from intuitionistic linear logic [Benton et al. 1993] to model their non-linear, or *intuitionistic*, semantics.

$$\textsc{Ty-Var}\ \frac{}{x : \tau \vdash_{RC} x : \tau} \qquad \textsc{Ty-Weaken}\ \frac{\Gamma \vdash_{RC} e : \tau}{\Gamma, x : \mathbb{B} \vdash_{RC} e : \tau} \qquad \textsc{Ty-Contract}\ \frac{\Gamma, x : \mathbb{B}, x : \mathbb{B} \vdash_{RC} e : \tau}{\Gamma, x : \mathbb{B} \vdash_{RC} e : \tau} \qquad \textsc{Ty-Contract-F}\ \frac{\Gamma, x : \mathbb{B}, x : \mathbb{B} \vdash_{RC} F}{\Gamma, x : \mathbb{B} \vdash_{RC} F}$$

We define well-typed programs and constants in terms of well-typed function bodies; the return type is elided since it is always $\mathbb{O}$.

$$\frac{\vdash_{\beta} \delta \qquad \forall c \in \mathrm{dom}(\delta).\ \delta \vdash_{RC} c}{\vdash_{RC} \delta} \qquad \frac{\delta(c) = \lambda\, \overline{y}.\ F \qquad \overline{y : \beta(c)} \vdash_{RC} F}{\delta \vdash_{RC} c}$$

inc introduces a new owned reference from a borrowed or owned reference, **dec** consumes an owned or reset reference.

$$\textsc{Ty-Inc}\ \frac{\tau \in \{\mathbb{O}, \mathbb{B}\} \qquad \Gamma, x : \tau, x : \mathbb{O} \vdash_{RC} F}{\Gamma, x : \tau \vdash_{RC} \mathbf{inc}\ x;\ F} \qquad \textsc{Ty-Dec}\ \frac{\tau \in \{\mathbb{O}, \mathbb{R}\} \qquad \Gamma \vdash_{RC} F}{\Gamma, x : \tau \vdash_{RC} \mathbf{dec}\ x;\ F}$$

Note that the first rule can introduce the same variable with two different types. It may help to view our type system as a *capability* system: A hypothesis of type $\mathbb{O}$ or $\mathbb{R}$ grants (exactly) one consuming usage, while one of type $\mathbb{B}$ grants only non-consuming usage.

Return values must be owned, while non-consuming, immediate uses like in **case** can be owned or borrowed.

$$\textsc{Ty-Return}\ \frac{\Gamma \vdash_{RC} x : \mathbb{O}}{\Gamma \vdash_{RC} \mathbf{ret}\ x} \qquad \textsc{Ty-Case}\ \frac{\tau \in \{\mathbb{O}, \mathbb{B}\} \qquad \overline{\Gamma, x : \tau \vdash_{RC} F}}{\Gamma, x : \tau \vdash_{RC} \mathbf{case}\ x\ \mathbf{of}\ \overline{F}}$$

Applications are typed by splitting up the linear context. Arguments to partial, variable and constructor applications must be owned because, in general, we cannot statically assert that the resulting value will not escape the current function and thus the scope of borrowed references.

$$\textsc{Ty-Const-App-Full}\ \frac{\overline{\Gamma \vdash_{RC} y : \beta(c)}}{\overline{\Gamma} \vdash_{RC} c\ \overline{y} : \mathbb{O}} \qquad \textsc{Ty-Const-App-Part}\ \frac{\beta(c) = \overline{\mathbb{O}}}{\overline{y : \mathbb{O}} \vdash_{RC} \mathbf{pap}\ c\ \overline{y} : \mathbb{O}}$$

$$\textsc{Ty-Var-App}\ \frac{}{x : \mathbb{O}, y : \mathbb{O} \vdash_{RC} x\ y : \mathbb{O}} \qquad \textsc{Ty-Cnstr-App}\ \frac{}{\overline{y : \mathbb{O}} \vdash_{RC} \mathbf{cnstr}_i\ \overline{y} : \mathbb{O}}$$

$$\textsc{Ty-Reset}\ \frac{}{x : \mathbb{O} \vdash_{RC} \mathbf{reset}\ x : \mathbb{R}} \qquad \textsc{Ty-Reuse}\ \frac{}{x : \mathbb{R}, \overline{y : \mathbb{O}} \vdash_{RC} \mathbf{reuse}\ x\ \mathbf{in}\ \mathbf{cnstr}_i\ \overline{y} : \mathbb{O}}$$

In order to type (saturated) applications with borrowed parameters, the rule for **let** should support temporarily obtaining a borrowed reference from an owned reference, much like Wadler [1990]'s **let!** construct. The rule makes the owned reference unavailable during the call to ensure that the borrowed reference is valid for that duration. The result type of e ensures that the borrow cannot survive past the call.

$$\textsc{Ty-Let}\ \frac{\Gamma, \overline{x : \mathbb{B}} \vdash_{RC} e : \tau \qquad \tau \in \{\mathbb{O}, \mathbb{R}\} \qquad \Delta, \overline{x : \mathbb{O}}, z : \tau \vdash_{RC} F}{\Gamma, \Delta, \overline{x : \mathbb{O}} \vdash_{RC} \mathbf{let}\ z = e;\ F}$$

Projections are handled specially. It is sound to treat the projection of a borrowed reference as borrowed because borrowed references are assumed to be valid for the entire function call. On the other hand, when projecting an owned reference, we conservatively treat the result as owned as well by requiring that it is incremented immediately; a more flexible model would need a more sophisticated "borrow checker" that makes sure that the projection cannot outlive the projected reference.

$$\textsc{Ty-Proj-Bor}\ \frac{\Gamma, x : \mathbb{B}, y : \mathbb{B} \vdash_{RC} F}{\Gamma, x : \mathbb{B} \vdash_{RC} \mathbf{let}\ y = \mathbf{proj}_i\ x;\ F} \qquad \textsc{Ty-Proj-Own}\ \frac{\Gamma, x : \mathbb{O}, y : \mathbb{O} \vdash_{RC} F}{\Gamma, x : \mathbb{O} \vdash_{RC} \mathbf{let}\ y = \mathbf{proj}_i\ x;\ \mathbf{inc}\ y;\ F}$$

Definition 6. *The function* valof $: Loc \times State \rightharpoonup Value_{pure}$ *is defined as follows:*

$$\begin{aligned} \mathrm{valof}(l, \sigma) &= \mathbf{ctor}_i\ \overline{\mathrm{valof}(l', \sigma)} && \text{if } \sigma(l) = \mathbf{ctor}_i\ \overline{l'} \\ \mathrm{valof}(l, \sigma) &= \mathbf{pap}\ c\ \overline{\mathrm{valof}(l', \sigma)} && \text{if } \sigma(l) = \mathbf{pap}\ c\ \overline{l'} \end{aligned}$$

Theorem 5 (semantics preservation). *Suppose the program is well-typed,* $\vdash_{RC} \delta$*, and* c *is a parameter-less constant,* $\delta(c) = F$*.*

(1) *If* $\vdash F \Downarrow v$*, then* $\vdash \langle F, \emptyset \rangle \Downarrow \langle l, \sigma \rangle$ *and* $\mathrm{valof}(l, \sigma) = v$*.*
(2) *If* $\vdash \langle F, \emptyset \rangle \Downarrow \langle l, \sigma \rangle$*, then* $\vdash F \Downarrow \mathrm{valof}(l, \sigma)$*.*

Proof. Below. The proof directly follows Chirimar et al. [1996]'s proof of this theorem for a similar linear type system (we direct interested readers to this paper for proofs of further properties such as freedom of memory leaks). The fundamental idea of inducing a *memory graph* from the heap and local variables and proving that the in-degrees of its nodes is equal to the values' reference counts is directly applicable to our owned references. We will quickly discuss parts of our system not present in theirs that needed to be fitted into the proofs:

- *Borrowed references* do not change the reference count and thus the definition of the memory graph does not need to be adjusted. However, the proof needed to be extended with an additional hypothesis that every borrowed reference is reachable from an owned *root* variable in a parent stack frame, which implies that the borrowed reference is valid for the duration of the current function call.
- *Reset references* are restricted by the semantics and type system to be used only in **reuse** and **dec**, but otherwise behave linearly like owned references. Because we replace their former contents with $\perp$ instead of leaving them as dangling pointers, treating reset references like owned references in the memory graph results in the correct behavior without further changes. An additional assumption makes sure that every reset references does in fact have this shape.

□

A.6 Proof of semantics preservation

A *memory graph* $\mathcal{G}$ is a tuple $(V, E, s, t, \bar{l})$ where (V, E, s, t) is a directed multigraph with a *root (multi)set* $\bar{l} \subseteq V$[1]. The *reference count* of a vertex $v \in V$ is the sum of inner and outer references

$$\text{in-degree}(v) + \mid \{i \mid v = l_i\} \mid$$

A *state* is a pair $(\bar{l}, \sigma)$ of a root set $\bar{l}$ into a store σ.

Definition 7. *If* $S = (\bar{l}, \sigma)$ *is a state, the* memory graph $\mathcal{G}(S)$ induced by S *is a memory graph with* $\mathrm{dom}(\sigma)$ *as its vertices and an edge from* $l \in \mathrm{dom}(\sigma)$ *to every* $l' \in \sigma(l)$.

Definition 8. *A state* $S = (\bar{l}, \sigma)$ *is* count-correct *if, for each* $\sigma(l) = (v, i)$*, the reference count of* l *in* $\mathcal{G}(S)$ *is* i.

Definition 9. *A state* $S = (\bar{l}, \sigma)$ *is called* regular*, written* $\Re(S)$*, provided the following conditions hold:*

$\Re1$ *S is count-correct.*
$\Re2$ $\mathrm{dom}(\sigma)$ *is finite.*
$\Re3$ *The reference count is non-zero for every* $l \in \mathrm{dom}(\sigma)$.

Definition 10 (Reference reachability). *A reference* l' *is reachable from* l *in the store* σ, $\mathrm{reachable}_\sigma(l, l')$*, if there is a path from* l *to* l' *in* $\mathcal{G}([], \sigma)$[2].

Theorem 6 (Memory Graph Laws).

B $\Re(\bar{l}, \sigma)$ *iff* $\Re(\bar{l}\,\blacksquare, \sigma)$.
D *If* $\Re(\bar{l}\; l', \sigma)$*, then* $\Re(\bar{l}, \mathrm{dec}(l', \sigma))$.
I *If* $\Re(\bar{l}, \sigma)$*, and* $l' \in \mathrm{dom}(\rho)$*, then* $\Re(\bar{l}\; l', \mathrm{inc}(l', \sigma))$.

Proof.

B As $\blacksquare$ is not a reference, it does not influence reference counts.
D By induction over the total sum of reference counts (which is finite by regularity).

[1]We will operate on such lists up to permutations without further mention
[2]The root set is irrelevant for this definition

I Trivial.

□

Theorem 7 (Preservation of regularity). *Suppose*

$$\vdash_{RC} \delta,\ \overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} F, \qquad \textit{(program and body are well-formed)}$$

$$\mathrm{dom}(\rho) = \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ \overline{y_{\mathbb{R}}}, \mathfrak{R}(\bar{l}\ \rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}}), \sigma), \textit{and} \qquad \textit{(owned variables are rooted)}$$

$$\forall y \in \overline{y_{\mathbb{B}}}.\ \exists l \in \bar{l}.\ \mathrm{reachable}_\sigma(l, \rho(y)). \qquad \textit{(borrowed variables are reachable)}$$

If $\rho \vdash \langle F, \sigma\rangle \Downarrow \langle l', \sigma'\rangle$*, then* $\mathfrak{R}(\bar{l}\ l', \sigma')$*. Moreover, if* $\bar{l} = \overline{l_1}\ \overline{l_2}$ *and* $l \in \mathrm{dom}(\sigma)$ *is not reachable from* $\overline{l_1}\ \mathrm{ran}(\rho)$ *in* $\mathcal{G}(\bar{l}\ \mathrm{ran}(\rho), \sigma)$*, then* $\sigma'(l) = \sigma(l)$ *and* l *is not reachable from* $\overline{l_1}\ l'$ *in* $\mathcal{G}(\bar{l}\ l', \sigma')$ *(the* reachability property*).*

Here $\bar{l}$ are roots outside of the current context, i.e. from a parent stack frame. Note that in our model, borrowed references are assumed to be alive for the whole function call, i.e. they must be reachable from a parent stack frame.

Proof. By induction over $\rho \vdash \langle F, \sigma\rangle \Downarrow \langle l', \sigma'\rangle$.

Case Let + Ctor-App

By case inversions of the typing assumption, we have $\Gamma = \overline{y : \mathbb{O}}$. Thus there exists $\overline{y'_{\mathbb{O}}}$ such that $\bar{y}\ \overline{y'_{\mathbb{O}}} = \overline{y_{\mathbb{O}}}$. For the IH we need to show $\mathfrak{R}(\bar{l}\ \rho(\overline{y'_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}})\ l', \sigma[l' \mapsto (\mathbf{ctor}_i\ \rho(\bar{y}), 1)])$ and the reachability property. l' is fresh, so its in-degree is indeed 0. All $\rho(\bar{y})$ have been moved from roots into l', so they are count-correct as well. The reachability property holds because the store is only extended, not modified, and all locations reachable from l' have already been reachable from $\rho(\overline{y_{\mathbb{O}}})$.

Case Let + Const-App-Part/Var-App-Part

Analogously.

Case Let + Const-App-Full

We have $F = \mathbf{let}\ y = c\ \overline{y'};\ F'$, $\delta(c) = \lambda\ \overline{y_c}.\ F_c$. We first apply the IH to F_c: by the typing assumption, it is well-typed and the types of arguments correspond to their respective parameter types, so owned arguments are rooted and borrowed variables are reachable (either from $\bar{l}$ by the reachability assumption, or from some $y_{\mathbb{O}}$ not passed to c but temporarily borrowed in Ty-Let). We obtain that the new state is regular and fulfills the reachability property after removing all owned variables passed to c from and adding l' to the root set. Thus we can apply the IH to F'.

To show the reachability property, assume $l \in \mathrm{dom}(\sigma)$ is not reachable from $\overline{l_1}\ \mathrm{ran}(\rho)$ where $\bar{l} = \overline{l_1}\ \overline{l_2}$. Then by the first IH, it is unreachable from $\overline{l_1}\ l'$ in the new state and $\sigma'(l) = \sigma(l)$. Thus we can conclude by the second IH. Like Chirimar et al. [1996], we will omit further similar proofs of the reachability property.

Case Let + Var-App-Full

Similarly, but we need additional steps to argue for the regularity of the state passed to F_c: $\rho(x)$ is a root by the inductive assumptions, but is not passed to either of F_c or F'. *dec* thus correctly removes it from the root set by law **D**. Conversely, the $\overline{l'}$ in $\sigma(\rho(x))$ are passed as owned arguments not taken from existing roots (but reachable from the root $\rho(x)$, i.e. in $\mathrm{dom}(\sigma)$), so *inc* correctly adds them to the root set by **I**.

Case Let + Proj

We have $F = \mathbf{let}\ y = \mathbf{proj}_i\ x;\ F'$. We continue by case analysis of the typing assumption.

Case Ty-Proj-Bor
We have $x, y : \mathbb{B}$, so $x \in \overline{y_{\mathbb{B}}}$. Thus x is reachable by assumption, and so is y by the **ctor** reachability rule and transitivity. Therefore we can apply the IH.

Case Ty-Proj-Own
We have $F' = \textbf{inc}\ y;\ F''$ and $x, y : \mathbb{O}$, so $x \in \overline{y_{\mathbb{O}}}$. Because y is both registered as a new root and incremented, we can apply the IH to F''.

Case Return
By the typing assumption, x is the only owned variable left. Thus we can directly apply the regularity assumption.

Case Case
No changes to the context or store, so the IH applies immediately.

Case Inc
By the typing assumption, we have $\tau \in \{\mathbb{O}, \mathbb{B}\}$. In either case, $\rho(x)$ is equal to or reachable from a root and thus $\rho(x) \in \mathrm{dom}(\sigma)$, so we are done by law **I** and the IH.

Case Dec
x is a root by the typing assumption, so we are done by law **D** and the IH.

Case Let + Reset-Uniq
It is easy to see that the new store is count-correct. While x changes its type, it remains a root, so the state is regular and we can apply the IH.

Case Let + Reset-Shared
By laws **B** and **D** and the IH.

Case Let + Reuse-Uniq
Similarly to Ctor-App.

Case Let + Reuse-Shared
By the IH.

□

Lemma 1. *Suppose*

$$\vdash_{RC} \delta,\ \overline{y_{\mathbb{O}} : \mathbb{O}}\ \overline{y_{\mathbb{B}} : \mathbb{B}}\ \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} F, \qquad \textit{(program and body are well-formed)}$$

$$\mathrm{dom}(\rho) = \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ \overline{y_{\mathbb{R}}}, \Re(\bar{l}\ \rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}}), \sigma), \qquad \textit{(owned variables are rooted)}$$

$$\forall y \in \overline{y_{\mathbb{B}}}.\ \exists l \in \bar{l}.\ \mathrm{reachable}_{\sigma}(l, \rho(y)), \textit{and} \qquad \textit{(borrowed variables are reachable)}$$

$$\forall y \in \overline{y_{\mathbb{R}}}.\ \rho(y) = \blacksquare \vee \exists i, n, r. \sigma(\rho(y)) = (\textbf{ctor}_i\ \blacksquare^n, r). \qquad \textit{(reset variables are reset)}$$

(1) *If* $\mathrm{valof}(\rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}), \sigma) \vdash F \Downarrow v$, *then* $\rho \vdash \langle F, \sigma\rangle \Downarrow \langle l, \sigma'\rangle$ *and* $v = \mathrm{valof}(l, \sigma')$.
(2) *If If* $\rho \vdash \langle F, \sigma\rangle \Downarrow \langle l, \sigma'\rangle$, *then* $\mathrm{valof}(\rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}), \sigma) \vdash F \Downarrow \mathrm{valof}(l, \sigma')$.

Proof. The first part is proved by induction over $\mathrm{valof}(\rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}), \sigma) \vdash F \Downarrow v$.

Case $F = \textbf{ret}\ x$
By $\mathrm{valof}(\rho, \sigma)(x) = \mathrm{valof}(\rho(x), \sigma)$.

Case $F = \textbf{case}\ x\ \textbf{of}\ \overline{F'}$
Directly by the IH.

Case $F = \textbf{inc}\ x;\ \ F'$
We have $\mathrm{valof}(\rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ x), \mathrm{inc}(\rho(x), \sigma)) = \mathrm{valof}(\rho(\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ x), \sigma)$ by the definition of valof, so we can apply the IH.

Case $F = \textbf{dec}\ x;\ \ F'$
Let $\Gamma, x : \tau$ be the context. By $\Re 1$, we have that all values reachable from Γ still have reference

count ≥ 1 in $\sigma' := \mathrm{dec}(\rho(x), \sigma)$, so $\mathrm{valof}(\Gamma, \sigma') = \mathrm{valof}(\Gamma, \sigma)$ by the definition of dec, and we can apply the IH.

Case $F = \mathbf{let}\ y = \mathbf{proj}_i\ x;\ F'$ **if** $x \in \overline{y_{\mathbb{O}}}$
By the typing assumption, we have $F' = \mathbf{inc}\ y;\ F''$. Note that F' and F'' are equivalent in the pure semantics, so we can apply the IH to F'' as well, after noticing that $\mathrm{inc}(\cdot, \cdot)$ does not affect $\mathrm{valof}(\cdot, \cdot)$.

Case $F = \mathbf{let}\ y = \mathbf{proj}_i\ x;\ F'$ **if** $x \in \overline{y_{\mathbb{B}}}$
By the IH.

Case $F = \mathbf{let}\ y = \mathbf{reset}\ x;\ F'$
By either Reuse-Uniq or Reuse-Shared. In either case, the assumption about reset variables is fulfilled and we can apply the IH. Note that the pure context is unchanged.

Case $F = \mathbf{let}\ y = \mathbf{reuse}\ x\ \mathbf{in}\ \mathbf{ctor}_i\ \overline{y};\ F'$
By the assumption about reset references and the IH.

Case $F = \mathbf{let}\ z = c\ \overline{y};\ F'$
We have $\delta(c) = \lambda\ \overline{y_c}.\ F_c$. In order to apply the IH on F', we need to show that the value of all remaining context variables has not been changed by the call, which follows from Theorem 7's reachability property.

Case $F = \mathbf{let}\ z = \mathbf{pap}\ c\ \overline{y};\ F'$
There exists an $l' \notin \mathrm{dom}(\sigma)$ by $\mathfrak{R}2$. Apply the IH.

Case $F = \mathbf{let}\ z = \mathbf{ctor}_i\ \overline{y};\ F'$
Analogously.

Case $F = \mathbf{let}\ z = x\ y;\ F'$
We have $\mathrm{erase}_{RC}(F) = \mathbf{let}\ z = x\ y;\ \mathrm{erase}_{RC}(F')$. By case distinction, the two possible cases are Var-App-Full and Var-App-Part, which are handled similarly to the above cases.

The reverse direction is proved by induction over $\rho \vdash \langle F, \sigma\rangle \Downarrow \langle l, \sigma'\rangle$ with similar but simpler cases. □

Theorem 8 (semantics preservation). *Suppose the program is well-typed, $\vdash_{RC} \delta$, and c is a parameter-less constant, $\delta(c) = F$.*

(1) *If* $\vdash F \Downarrow v$, *then* $\vdash \langle F, \emptyset\rangle \Downarrow \langle l, \sigma\rangle$ *and* $\mathrm{valof}(l, \sigma) = v$.
(2) *If* $\vdash \langle F, \emptyset\rangle \Downarrow \langle l, \sigma\rangle$, *then* $\vdash F \Downarrow \mathrm{valof}(l, \sigma)$.

Proof. A direct corollary of Lemma 1. □

A.7 Proof of compilation well-typedness

Theorem 9. *For the specific δ_{RC} given in the paper, we have $\vdash_{RC} \delta_{RC}$.*

Proof. We start with a helper lemma about C.

Lemma 2. *C does not introduce new variables,* $\mathrm{FV}(C(F)) = \mathrm{FV}(F)$.

Proof. By induction over F. □

By the definition of δ_{RC}, we need to show for each $\lambda\ \overline{y}.\ F \in \delta_\beta$

$$\overline{y : \beta(c)} \vdash_{RC} \mathbb{O}^{-}(\overline{y}, C(F)) \text{ with } \beta_l := [\overline{y} \mapsto \beta(c), \ldots \mapsto \mathbb{O}]$$

Aside: In this proof, we will style β_l as an implicit variable *to reduce verbosity.*
By the wellformedness of δ_β (Theorem 4), we have $\vdash_\beta F$ and $\overline{y}; \cdot \vdash_{reuse} F$.

Note that all $y_i : \mathbb{O}$ are alive in $F' := \mathbb{O}^-(\overline{y}, C(F))$: If they do not occur in $C(F)$, they will occur in a **dec** instruction from $\mathbb{O}^-$ instead. Thus we can generalize the goal to

$$\frac{\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}} \subseteq FV(F') \qquad \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}; \overline{y_{\mathbb{R}}} \vdash_{reuse} F \qquad \vdash_{\beta} F \qquad \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ \overline{y_{\mathbb{R}}} \text{distinct}}{\overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} F' \text{ with } \beta_l := [\overline{y_{\mathbb{B}}} \mapsto \overline{\mathbb{B}}, \ldots \mapsto \mathbb{O}]}$$

where $\overline{y_{\mathbb{R}}} := []$ and $\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}} := \overline{y}$.

Unfolding $\mathbb{O}^-$ and applying the **dec** typing rule repeatedly, we remove all $y_{\mathbb{O}} \notin \mathrm{FV}(C(F))$ and, reusing the name $\overline{y_{\mathbb{O}}}$ for the reduced variable list and applying Lemma 2, are left with

$$\frac{\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}} \subseteq FV(F) \qquad \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}; \overline{y_{\mathbb{R}}} \vdash_{reuse} F \qquad \vdash_{\beta} F \qquad \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ \overline{y_{\mathbb{R}}} \text{distinct}}{\overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} C(F) \text{ with } \beta_l := [\overline{y_{\mathbb{B}}} \mapsto \overline{\mathbb{B}}, \ldots \mapsto \mathbb{O}]}$$

We proceed by induction over F, but not before noting a peculiarity about this induction hypothesis: Not only does the type context contain only owned variables that are alive in the remaining body (as one may expect), it also contains each of them no more than once. It turns out that duplicating a reference is only necessary just before applications, which will happen in between the induction steps of the proof. In this sense, we see that the compiler keeps all reference counts as low as possible.

Case $F =$ **ret** x

We need to show

$$\overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} \mathbb{O}^+_x(\texttt{ret } x)$$

By the first two inductive assumptions, we have $\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}} \subseteq \{x\}$ and $x \in \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}$, respectively. Together with the fourth assumption, x may appear at most once in either $\overline{y_{\mathbb{O}}}$ or $\overline{y_{\mathbb{B}}}$. If $\beta_l(x) = \mathbb{B}$, it remains to be shown that

$$x : \mathbb{B}, \overline{y : \mathbb{B}} \vdash_{RC} \texttt{inc } x;\ \texttt{ret } x$$

Applying the **inc** rule, we get to the same goal as in the case $\beta_l(x) = \mathbb{O}$:

$$x : \mathbb{O}, \overline{y' : \mathbb{B}} \vdash_{RC} \texttt{ret } x$$

which is closed by the **ret** rule plus weakening.

Case $F =$ **case** x **of** $\overline{F'}$

We need to show

$$\overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} \texttt{case } x \texttt{ of } \overline{\mathbb{O}^-(\overline{y'}, C(F'))}$$

where $\{\overline{y'}\} = \mathrm{FV}(\texttt{case } x \texttt{ of } \overline{F'})$. By the first two inductive assumptions, we have

$$\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{R}}} \subseteq \overline{y'}$$

$$x \in \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}, \quad \overline{\overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}; \overline{y_{\mathbb{R}}} \vdash_{reuse} F'}$$

Using $x \in \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}$, we can apply the **case** typing rule, leaving us with, for each F'_i,

$$\overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} \mathbb{O}^-(\overline{y'}, C(F'_i))$$

By Theorem 3, we have $\overline{y'} \subseteq \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}\ \overline{y_{\mathbb{R}}}$, and can repeatedly apply the **dec** rule for any $y'_j \notin \mathrm{FV}(C(F'_i)), \beta_l(y'_j) \neq \mathbb{B}$. We are left with

$$\overline{y'_{\mathbb{O}} : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y'_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} C(F'_i)$$

where $\overline{y'_{\mathbb{O}}} = [y \in \overline{y_{\mathbb{O}}} \mid y \in \mathrm{FV}(C(F'_i))]$ (and analogously for $\overline{y'_{\mathbb{R}}}$) , thus $\overline{y'_{\mathbb{O}}}\ \overline{y'_{\mathbb{R}}} \subseteq \mathrm{FV}(C(F'_i))$, which allows us to close the goal by the inductive hypothesis.

Case $F = \mathtt{let}\ y = \mathtt{proj}_i\ x;\ F'$ **if** $\beta_l(x) = \mathbb{O}$

We need to show

$$\overline{y_\mathbb{O} : \mathbb{O}}, \overline{y_\mathbb{B} : \mathbb{B}}, \overline{y_\mathbb{R} : \mathbb{R}} \vdash_{RC} \mathtt{let}\ y = \mathtt{proj}_i\ x;\ \mathtt{inc}\ y;\ \mathbb{O}^-_x(C(F'))$$

From $\beta_l(x) = \mathbb{O}$, we have $x \notin \overline{y_\mathbb{B}}$, so together with $\overline{y_\mathbb{O}}\ \overline{y_\mathbb{B}}; \overline{y_\mathbb{R}} \vdash_{reuse} \mathtt{let}\ y = \mathtt{proj}_i\ x;\ F'$, we have $x \in \overline{y_\mathbb{O}}$. Applying Ty-Proj-Own, we are left with

$$\overline{y_\mathbb{O} : \mathbb{O}}, y : \mathbb{O}, \overline{y_\mathbb{B} : \mathbb{B}}, \overline{y_\mathbb{R} : \mathbb{R}} \vdash_{RC} \mathbb{O}^-_x(C(F'))$$

If $x \notin \mathrm{FV}(C(F'))$, we apply Ty-Dec. In either case, we need to show

$$\overline{y'_\mathbb{O} : \mathbb{O}}, y : \mathbb{O}, \overline{y_\mathbb{B} : \mathbb{B}}, \overline{y_\mathbb{R} : \mathbb{R}} \vdash_{RC} \mathbb{O}^-_x(C(F'))$$

for some $\overline{y'_\mathbb{O}} \subseteq \mathrm{FV}(C(F'))$. We also have $y \in \mathrm{FV}(C(F')) = \mathrm{FV}(F')$ from δ_{reuse}, as well as $\overline{y_\mathbb{R}} \cap FV(C(F')) = \overline{y_\mathbb{R}} \cap FV(C(F))$, so we can apply the induction hypothesis.

Case $F = \mathtt{let}\ y = \mathtt{proj}_i\ x;\ F'$ **if** $\beta_l(x) = \mathbb{B}$

By **Ty-Proj-Bor** and the induction hypothesis.

Case $F = \mathtt{let}\ y = \mathtt{reset}\ x;\ F'$

By **Ty-Let**, **Ty-Reset**, and the induction hypothesis.

Case $F = \mathtt{let}\ z = c\ \overline{y};\ F'$

We need to show

$$\overline{y_\mathbb{O} : \mathbb{O}}, \overline{y_\mathbb{B} : \mathbb{B}}, \overline{y_\mathbb{R} : \mathbb{R}} \vdash_{RC} C_{app}(\overline{y}, \beta(c), \mathtt{let}\ z = c\ \overline{y};\ C(F'))$$

We generalize the goal (using $y^l := b^l := []$) to

$$\frac{\overline{y} = \overline{y^l}\ \overline{y^r} \qquad \overline{\beta(c)} = \overline{b^l}\ \overline{b^r} \qquad |\ \overline{y^l}\ | = |\ \overline{b^l}\ | \qquad \overline{y'_\mathbb{O}} = O(\overline{y^l})}{\overline{y_\mathbb{O} : \mathbb{O}}, \overline{y'_\mathbb{O} : \mathbb{O}}, \overline{y_\mathbb{B} : \mathbb{B}}, \overline{y_\mathbb{R} : \mathbb{R}} \vdash_{RC} C_{app}(\overline{y^r}, \overline{b^r}, \mathtt{let}\ z = c\ \overline{y};\ \mathbb{O}^-(B(\overline{y^l}), C(F')))}$$

where

$$\begin{aligned}
O(\overline{y^l}) &= O_\mathbb{B}(\overline{y^l})\ O_\mathbb{O}(\overline{y^l}) \\
O_\mathbb{B}(\overline{y^l}) &= [y^l_i \in \overline{y^l} \mid \beta(c)_i = \mathbb{O} \wedge \beta_l(y^l_i) = \mathbb{B}] \\
O_\mathbb{O}(\overline{y^l}) &= [y^l_i \in \overline{y^l} \mid \beta(c)_i = \mathbb{O} \wedge \beta_l(y^l_i) = \mathbb{O} \wedge (y^l_i \in \mathrm{FV}(F') \vee \exists j.\ y^l_i = y_j \wedge [j > i \vee \beta(c)_j = \mathbb{B}])] \\
B(\overline{y^l}) &= [y^l_i \in \overline{y^l} \mid \beta(c)_i = \mathbb{B} \wedge \beta_l(y^l_i) = \mathbb{O} \wedge \nexists j < i.\ y^l_i = y_j \wedge \beta(c)_j = \mathbb{B}]
\end{aligned}$$

O and B accurately describe what arguments to increment/decrement in an explicit form: We increment borrowed references that are passed to an owned parameter, as well as owned references passed to an owned parameter that

- are still used in F',
- are also passed to a borrowed parameter, or
- are also passed to another owned parameter later.

We decrement owned references passed to a borrowed parameter and dead in F', but at most once per variable.

We proceed by parallel induction over y_r and b_r, which, by $\vdash_{pure}$, have the same length:

Case $y^r = y'\ y'^r, b^r = \mathbb{O}\ b'^r$

We need to show

$$\cdots \vdash_{RC} \mathbb{O}^+_{y'}(\overline{y'^r} \cup \mathrm{FV}(\mathbb{O}^-(B(\overline{y^l}), C(F'))), C_{app}(\overline{y'^r}, \overline{b'^r}, \mathtt{let}\ z = c\ \overline{y};\ \mathbb{O}^-(B(\overline{y^l}), C(F')))$$

We see that $B(\overline{y^l}\ y') = B(\overline{y^l})$, and that $O(\overline{y^l}\ y')$ is $O(\overline{y^l})$ with y' appended iff

$$\beta_l(y') = \mathbb{B} \vee y' \in \mathrm{FV}(F') \vee y' \in y'^r \vee y' \in B(\overline{y^l})$$

This is exactly the condition for which an **inc** is inserted, so after conditionally applying Ty-Inc, we can apply the inner induction hypothesis.

Case $y^r = y'\, y'^r, b^r = \mathbb{B}\, b'^r$

We need to show

$$\cdots \vdash_{RC} C_{app}(\overline{y'^r}, \overline{b'^r}, \mathbf{let}\ z = c\ \overline{y};\ \mathbb{O}^-_{y'}(\mathbb{O}^-(B(\overline{y^l}), C(F'))))$$

We see that $O(\overline{y^l}\ y') = O(\overline{y^l})$, and that $B(\overline{y^l}\ y')$ is $B(\overline{y^l})$ with y' appended iff $y' \notin B(\overline{y^l}) \wedge \beta_l(y') = \mathbb{O}$. In either case, we have

$$\mathbb{O}^-_{y'}(\mathbb{O}^-(B(\overline{y^l}), C(F'))) = \mathbb{O}^-(B(\overline{y^l}\ y'), C(F'))$$

and can apply the inner induction hypothesis.

Case $y^r = [], b^r = []$

We are left to show

$$\overline{y_{\mathbb{O}} : \mathbb{O}}, \overline{O(\overline{y}) : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}} \vdash_{RC} \mathbf{let}\ z = c\ \overline{y};\ \mathbb{O}^-(B(\overline{y}), C(F'))))$$

We have $\overline{y} \subseteq \overline{y_{\mathbb{O}}}\ \overline{y_{\mathbb{B}}}$ by $\vdash_{pure}$. We thus notice that $B(\overline{y})$ is a submultiset of $\overline{y_{\mathbb{O}}}$ and call the difference list D. We further split D into

$$D_1 = [x \in D \mid x \notin \mathrm{FV}(F')]$$
$$D_2 = [x \in D \mid x \in \mathrm{FV}(F')]$$

We apply Ty-Let, moving D_1 and $O(\overline{y})$ into the first goal, temporarily borrowing $B(\overline{y})$, and copying $\overline{y_{\mathbb{B}}}$ into both goals via contraction, leaving us with

$$\overline{D_1 : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{O(\overline{y}) : \mathbb{O}}, \overline{B(\overline{y}) : \mathbb{B}} \vdash_{RC} c\ \overline{y} : \mathbb{O}$$
$$\overline{D_2 : \mathbb{O}}, \overline{y_{\mathbb{B}} : \mathbb{B}}, \overline{y_{\mathbb{R}} : \mathbb{R}}, \overline{B(\overline{y}) : \mathbb{O}}, z : \mathbb{O} \vdash_{RC} \mathbb{O}^-(B(\overline{y}), C(F'))$$

For the first goal, we notice that every argument used as a borrowed parameter is in $\overline{y_{\mathbb{B}}}$ or $B(\overline{y})$ by $\vdash_{pure}$, so by weakening we can fulfill them. For arguments used as owned parameters, we have to pay closer attention to the exact number of hypotheses: We notice that because $\overline{y_{\mathbb{O}}}$ is distinct, so is D_1, so we have covered the first occurrence of every variable dead in F' and not in $B(\overline{y})$. The missing arguments are by definition exactly $O(\overline{y})$, so we are done. For the second goal, we iteratively apply Ty-Dec and then apply the outer induction hypothesis: we have $D_2 \subseteq \mathrm{FV}(F')$ by definition, $z \in \mathrm{FV}(F')$ and $z \notin D_2 \cup B(\overline{y})$ by $\vdash_{pure}$, and $D_2 \cap B(\overline{y}) = \emptyset$ by definition of the split.

All other application cases are mostly analogous to the constant case (in particular, without any borrowed parameters). For **pap**, $\vdash_\beta$ proves the assumption $\beta(c) = \overline{\mathbb{O}}$. For **reuse**, the hypothesis $x : \mathbb{R}$, whose existence is guaranteed by $\vdash_{reuse}$, additionally has to be moved into the first goal.

□

Theorem 10. *δ_{RC} refines δ_β in the pure semantics.*

Proof. Trivial, given that the only change is insertion of **inc**/**dec** instructions. □

Corollary 1. *Suppose c is a parameter-less constant, $\delta(c) = F$.*

(1) *If* $\vdash \delta(c) \Downarrow v$, *then* $\vdash \langle \delta_{RC}(c), \emptyset \rangle \Downarrow \langle l, \sigma \rangle$ *and* $\mathrm{valof}(l, \sigma) = v$.

(2) *If* $\vdash \langle \delta_{RC}(c), \emptyset \rangle \Downarrow \langle l, \sigma \rangle$, *then* $\vdash \delta(c) \Downarrow \mathrm{valof}(l, \sigma)$.

Proof. By the pure refinement steps (..., Theorem 10), the welltypedness of δ_{RC} (Theorem 9), and the semantics preservation proof of welltyped programs (Theorem 5). □

REFERENCES

P. N. Benton, Gavin M. Bierman, Valeria de Paiva, and Martin Hyland. 1993. A Term Calculus for Intuitionistic Linear Logic. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications (TLCA '93)*. Springer-Verlag, London, UK, UK, 75–90. http://dl.acm.org/citation.cfm?id=645891.671430

Jawahar Chirimar, Carl A. Gunter, and Jon G. Riecke. 1996. Reference counting as a computational interpretation of linear logic. *Journal of Functional Programming* 6, 2 (1996), 195–244. https://doi.org/10.1017/S0956796800001660

P. Wadler. 1990. Linear types can change the world!. In *IFIP TC 2 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel (IFIP TC 2)*, M. Broy and C. Jones (Eds.). North Holland, 347–359.