Lambda-lifting and CPS conversion in an imperative language

Gabriel Kerneis, Juliusz Chroboczek

Lambda-lifting and CPS conversion in an imperative language

Gabriel Kerneis Juliusz Chroboczek
Université Paris Diderot, PPS, Paris, France

February 2012

Abstract

This paper is a companion technical report to the article "Continuation-Passing C: from threads to events through continuations". It contains the complete version of the proofs of correctness of lambda-lifting and CPS-conversion presented in the article.

Contents

1 Introduction

This paper is a companion technical report to the article "Continuation-Passing C: from threads to events through continuations" [4]. It contains the complete version of the proofs presented in the article. It does not, however, give any background or motivation for our work: please refer to the original article.

2 Lambda-lifting in an imperative language

To prove the correctness of lambda-lifting in an imperative, call-by-value language when functions are called in tail position, we do not reason directly on CPC programs, because the semantics of C is too broad and complex for our purposes. The CPC translator leaves most parts of converted programs intact, transforming only control structures and function calls. Therefore, we define a simple language with restricted values, expressions and terms, that captures the features we are most interested in (Section 2.1).

The reduction rules for this language (Section 2.1.1) use a simplified memory model without pointers and enforce that local variables are not accessed outside of their scope, as ensured by our boxing pass. This is necessary since lambda-lifting is not correct in general in the presence of extruded variables.

It turns out that the "naive" reduction rules defined in Section 2.1.1 do not provide strong enough invariants to prove this correctness theorem by induction, mostly because we represent memory with a store that is not invariant with respect to lambda-lifting. Therefore, in Section 2.2, we define an equivalent, "optimised" set of reduction rules which enforces more regular stores and closures.

The proof of correctness is then carried out in Section 2.4 using these optimised rules. We first define the invariants needed for the proof and formulate a strengthened version of the correctness theorem (Theorem 2.28, Section 2.4.1). A comprehensive overview of the proof is then given in Section 2.4.2. The proof is fully detailed in Section 2.4.5, with the help of a number of lemmas to keep the main proof shorter (Sections 2.4.3 and 2.4.4).

The main limitation of this proof is that Theorems 2.9 and 2.28 are implications, not equivalences: we do not prove that if a term does not reduce, it will not reduce once lifted. For instance, this proof does not ensure that lambda-lifting does not break infinite loops.

2.1 Definitions

In this section, we define the terms (Definition 2.1), the reduction rules (Section 2.1.1) and the lambda-lifting transformation itself (Section 2.1.2) for our small imperative language. With these preliminary definitions, we are then able to characterise *liftable parameters* (Definition 2.8) and state the main correctness theorem (Theorem 2.9, Section 2.1.3).

Definition 2.1 (Values, expression and terms)**.** *Values are either boolean and integer constants or* $\mathbf{1}$*, a special value for functions returning* `void`*.*

$$v ::= \quad \mathbf{1} \mid \mathbf{true} \mid \mathbf{false} \mid n \in \mathbf{N}$$

Expressions are either values or variables. We deliberately omit arithmetic and boolean operators, with the sole concern of avoiding boring cases in the proofs.

$$e ::= \quad v \mid x \mid \ldots$$

Terms are consist of assignments, conditionals, sequences, recursive functions definitions and calls.

$$\begin{aligned} T ::= \quad & e \mid x := T \mid \textbf{if } T \textbf{ then } T \textbf{ else } T \mid T \ ; \ T \\ & \mid \textbf{letrec } f(x_1 \dots x_n) = T \textbf{ in } T \mid f(T, \dots, T) \end{aligned}$$

Our language focuses on the essential details affected by the transformations: recursive functions, conditionals and memory accesses. Loops, for instance, are ignored because they can be expressed in terms of recursive calls and conditional jumps — and that is, in fact, how the splitting pass translates them. Since lambda-lifting happens after the splitting pass, our language need to include inner functions (although they are not part of the C language), but it can safely exclude `goto` statements.

2.1.1 Naive reduction rules

Environments and stores Handling inner functions requires explicit closures in the reduction rules. We need environments, written ρ, to bind variables to locations, and a store, written s, to bind locations to values.

Environments and *stores* are partial functions, equipped with a single operator which extends and modifies a partial function: $\cdot + \{\cdot \mapsto \cdot\}$.

Definition 2.2. *The modification (or extension) f' of a partial function f, written $f' = f + \{x \mapsto y\}$, is defined as follows:*

$$f'(t) = \begin{cases} y & \text{when } t = x \\ f(t) & \text{otherwise} \end{cases}$$
$$\mathrm{dom}(f') = \mathrm{dom}(f) \cup \{x\}$$

Definition 2.3 (Environments of variables and functions)**.** *Environments of variables are defined inductively by*

$$\rho ::= \varepsilon \mid (x, l) \cdot \rho,$$

i.e. the empty domain function and $\rho + \{x \mapsto l\}$ (respectively).

Environments of functions associate function names to closures:

$$\mathcal{F} : \{f, g, h, \dots\} \to \{[\lambda x_1 \dots x_n.T, \rho, \mathcal{F}]\}.$$

Note that although we have a notion of locations, which correspond roughly to memory addresses in C, there is no way to copy, change or otherwise manipulate a location directly in the syntax of our language. This is on purpose, since adding this possibility would make lambda-lifting incorrect: it translates the fact, ensured by the boxing pass in the CPC translator, that there are no extruded variables in the lifted terms.

Reduction rules We use classical big-step reduction rules for our language (Figure 1, p. 4).

In the (call) rule, we need to introduce *fresh* locations for the parameters of the called function. This means that we must choose locations that are not already in use, in particular in the environments ρ' and $\mathcal{F}$. To express this choice, we define two ancillary functions, Env and Loc, to extract the environments and locations contained in the closures of a given environment of functions $\mathcal{F}$.

$$(\textsc{val})\ \frac{}{v^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s}} \qquad (\textsc{var})\ \frac{\rho\ x = l \in \mathrm{dom}\ \ s}{x^{\,s} \xrightarrow[\mathcal{F}]{\rho} s\ l^{\,s}}$$

$$(\textsc{assign})\ \frac{a^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s'} \qquad \rho\ x = l \in \mathrm{dom}\ \ s'}{x := a^{\,s} \xrightarrow[\mathcal{F}]{\rho} \mathbf{1}^{\,s'+\{l \mapsto v\}}} \qquad (\textsc{seq})\ \frac{a^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s'} \qquad b^{\,s'} \xrightarrow[\mathcal{F}]{\rho} v'^{\,s''}}{a\ ;\ b^{\,s} \xrightarrow[\mathcal{F}]{\rho} v'^{\,s''}}$$

$$(\textsc{if-t.})\ \frac{a^{\,s} \xrightarrow[\mathcal{F}]{\rho} \mathbf{true}^{\,s'} \qquad b^{\,s'} \xrightarrow[\mathcal{F}]{\rho} v^{\,s''}}{\mathbf{if}\ \ a\ \ \mathbf{then}\ \ b\ \ \mathbf{else}\ \ c^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s''}} \qquad (\textsc{if-f.})\ \frac{a^{\,s} \xrightarrow[\mathcal{F}]{\rho} \mathbf{false}^{\,s'} \qquad c^{\,s'} \xrightarrow[\mathcal{F}]{\rho} v^{\,s''}}{\mathbf{if}\ \ a\ \ \mathbf{then}\ \ b\ \ \mathbf{else}\ \ c^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s''}}$$

$$(\textsc{letrec})\ \frac{b^{\,s} \xrightarrow[\mathcal{F}']{\rho} v^{\,s'} \qquad \mathcal{F}' = \mathcal{F} + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho, \mathcal{F}]\}}{\mathbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \mathbf{in}\ \ b^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s'}}$$

$$(\textsc{call})\ \frac{\begin{array}{c}\mathcal{F}\,f = [\lambda x_1 \ldots x_n.b, \rho', \mathcal{F}'] \qquad \rho'' = (x_1, l_1) \cdot \ldots \cdot (x_n, l_n) \qquad l_i \text{ fresh and distinct} \\ \forall i, a_i^{\,s_i} \xrightarrow[\mathcal{F}]{\rho} v_i^{\,s_{i+1}} \qquad b^{\,s_{n+1}+\{l_i \mapsto v_i\}} \xrightarrow[\mathcal{F}'+\{f \mapsto \mathcal{F}\,f\}]{\rho''\cdot\rho'} v^{\,s'}\end{array}}{f(a_1 \ldots a_n)^{\,s_1} \xrightarrow[\mathcal{F}]{\rho} v^{\,s'}}$$

Figure 1: "Naive" reduction rules

Definition 2.4 (Set of environments, set of locations)**.**

$$\mathrm{Env}(\mathcal{F}) = \bigcup \{\rho, \rho' \mid [\lambda x_1 \ldots x_n.M, \rho, \mathcal{F}'] \in \mathrm{Im}(\mathcal{F}), \rho' \in \mathrm{Env}(\mathcal{F}')\}$$

$$\mathrm{Loc}(\mathcal{F}) = \bigcup \{\mathrm{Im}(\rho) \mid \rho \in \mathrm{Env}(\mathcal{F})\}$$

A location l is said to appear *in $\mathcal{F}$ iff* $l \in \mathrm{Loc}(\mathcal{F})$.

These functions allow us to define fresh locations.

Definition 2.5 (Fresh location)**.** *In the (call) rule, a location is* fresh *when:*

- *$l \notin \mathrm{dom}(s_{n+1})$, i.e. l is not already used in the store before the body of f is evaluated, and*
- *l doesn't appear in $\mathcal{F}' + \{f \mapsto \mathcal{F}\,f\}$, i.e. l will not interfere with locations captured in the environment of functions.*

Note that the second condition implies in particular that l does not appear in either $\mathcal{F}$ or ρ'.

2.1.2 Lambda-lifting

Lambda-lifting can be split into two parts: parameter lifting and block floating[2]. We will focus only on the first part here, since the second one is trivial. Parameter lifting consists in adding a free variable as a parameter of every inner function where it appears free. This step is repeated until every variable is bound in every function, and closed functions can safely be floated to top-level. Note that although the transformation is called lambda-lifting, we do not focus on a single function and try to lift all of its free variables; on the contrary, we define the lifting of a single free parameter x in every possible function.

Smart lambda-lifting algorithms strive to minimize the number of lifted variables. Such is not our concern in this proof: parameters are lifted in every function where they might potentially be free.

Definition 2.6 (Parameter lifting in a term)**.** *Assume that x is defined as a parameter of a given function g, and that every inner function in g is called h_i (for some $i \in \mathbf{N}$). Also assume that function parameters are unique before lambda-lifting.*
Then the lifted form $(M)_*$ *of the term M with respect to x is defined inductively as follows:*

$$(\mathbf{1})_* = \mathbf{1} \qquad (n)_* = n$$
$$(true)_* = true \qquad (false)_* = false$$
$$(y)_* = y \quad and \quad (y \coloneqq a)_* = y \coloneqq (a)_* \quad (even\ if\ y = x)$$
$$(a\ ;\ b)_* = (a)_*\ ;\ (b)_*$$
$$(\mathbf{if}\ a\ \mathbf{then}\ b\ \mathbf{else}\ c)_* = \mathbf{if}\ (a)_*\ \mathbf{then}\ (b)_*\ \mathbf{else}\ (c)_*$$
$$(\mathbf{letrec}\ f(x_1 \dots x_n) = a\ \mathbf{in}\ b)_* = \begin{cases} \mathbf{letrec}\ f(x_1 \dots x_n x) = (a)_*\ \mathbf{in}\ (b)_* & \text{if } f = h_i \\ \mathbf{letrec}\ f(x_1 \dots x_n) = (a)_*\ \mathbf{in}\ (b)_* & \text{otherwise} \end{cases}$$
$$(f(a_1 \dots a_n))_* = \begin{cases} f((a_1)_*, \dots, (a_n)_*, x) & \text{if } f = h_i \text{ for some } i \\ f((a_1)_*, \dots, (a_n)_*) & \text{otherwise} \end{cases}$$

2.1.3 Correctness condition

We show that parameter lifting is correct for variables defined in functions whose inner functions are called exclusively in *tail position*. We call these variables *liftable parameters*.

We first define tail positions as usual [1]:

Definition 2.7 (Tail position)**.** Tail positions *are defined inductively as follows:*
1. *M and N are in tail position in* **if** P **then** M **else** N.
2. *N is in tail position in N and M ; N and* **letrec** $f(x_1 \dots x_n) = M$ **in** N.

A parameter x defined in a function g is liftable if every inner function in g is called exclusively in tail position.

Definition 2.8 (Liftable parameter)**.** *A parameter x is* liftable *in M when:*
- *x is defined as the parameter of a function g,*
- *inner functions in g, named h_i, are called exclusively in tail position in g or in one of the h_i.*

Our main theorem states that performing parameter-lifting on a liftable parameter preserves the reduction:

Theorem 2.9 (Correctness of lambda-lifting)**.** *If x is a liftable parameter in M, then*

$$\exists t, M\,^{\varepsilon} \xrightarrow[\varepsilon]{} v\,^{t} \text{ implies } \exists t', (M)_*^{\varepsilon} \xrightarrow[\varepsilon]{} v\,^{t'}.$$

Note that the resulting store t' changes because lambda-lifting introduces new variables, hence new locations in the store, and changes the values associated with lifted variables; Section 2.4 is devoted to the proof of this theorem. To maintain invariants during the proof, we need to use an equivalent, "optimised" set of reduction rules; it is introduced in the next section.

2.2 Optimised reduction rules

The naive reduction rules (Section 2.1.1) are not well-suited to prove the correctness of lambda-lifting. Indeed, the proof is by induction and requires a number of invariants on the structure of stores and environments. Rather than having a dozen of lemmas to ensure these invariants during the proof of correctness, we translate them as constraints in the reduction rules.

To this end, we introduce two optimisations — minimal stores (Section 2.2.1) and compact closures (Section 2.2.2) — which lead to the definition of an optimised set of reduction rules (Figure 2, Section 2.2.3). The equivalence between optimised and naive reduction rules is shown in Section 2.3.

2.2.1 Minimal stores

In the naive reduction rules, the store grows faster when reducing lifted terms, because each function call adds to the store as many locations as it has function parameters. This yields stores of different sizes when reducing the original and the lifted term, and that difference cannot be accounted for locally, at the rule level.

Consider for instance the simplest possible case of lambda-lifting:

$$\textbf{letrec } g(x) = (\textbf{letrec } h() = x \textbf{ in } h()) \textbf{ in } g(\mathbf{1}) \qquad \text{(original)}$$
$$\textbf{letrec } g(x) = (\textbf{letrec } h(y) = y \textbf{ in } h(x)) \textbf{ in } g(\mathbf{1}) \qquad \text{(lifted)}$$

At the end of the reduction, the store for the original term is $\{l_x \mapsto \mathbf{1}\}$ whereas the store for the lifted term is $\{l_x \mapsto \mathbf{1}; l_y \mapsto \mathbf{1}\}$. More complex terms would yield even larger stores, with many out-of-date copies of lifted variables.

To keep the store under control, we need to get rid of useless variables as soon as possible during the reduction. It is safe to remove a variable x from the store once we are certain that it will never be used again, i.e. as soon as the term in tail position in the function which defines x has been evaluated. This mechanism is analogous to the deallocation of a stack frame when a function returns.

To track the variables whose location can be safely reclaimed after the reduction of some term M, we introduce *split environments*. Split environments are written $\rho_T|\rho$, where ρ_T is called the *tail environment* and ρ the non-tail one; only the variables belonging to the tail environment may be safely reclaimed. The reduction rules build environments so that a variable x belongs to ρ_T if and only if the term M is in tail position in the current function f and x is a parameter of f. In that case, it is safe to discard the locations associated to all of the parameters of f, including x, after M has been reduced because we are sure that the evaluation of f is completed (and there are no first-class functions in the language to keep references on variables beyond their scope of definition).

We also define a *cleaning* operator, $\cdot \setminus \cdot$, to remove a set of variables from the store.

Definition 2.10 (Cleaning of a store)**.** *The store s cleaned with respect to the variables in ρ, written $s \setminus \rho$, is defined as $s \setminus \rho = s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}$.*

2.2.2 Compact closures

Another source of complexity with the naive reduction rules is the inclusion of useless variables in closures. It is safe to remove from the environments of variables contained in closures the variables that are also parameters of the function: when the function is called, and the environment restored, these variables will be hidden by the freshly instantiated parameters.

This is typically what happens to lifted parameters: they are free variables, captured in the closure when the function is defined, but these captured values will never be used since calling the function adds fresh parameters with the same names. We introduce *compact closures* in the optimised reduction rules to avoid dealing with this hiding mechanism in the proof of lambda-lifting.

A compact closure is a closure that does not capture any variable which would be hidden when the closure is called because of function parameters having the same name.

Definition 2.11 (Compact closure and environment)**.** *A closure* $[\lambda x_1 \dots x_n.M, \rho, \mathcal{F}]$ *is* compact *if* $\forall i, x_i \notin \mathrm{dom}(\rho)$ *and* $\mathcal{F}$ *is compact. An environment is* compact *if it contains only compact closures.*

We define a canonical mapping from any environment $\mathcal{F}$ to a compact environment $\mathcal{F}_*$, restricting the domains of every closure in $\mathcal{F}$.

Definition 2.12 (Canonical compact environment)**.** *The* canonical compact environment $\mathcal{F}_*$ *is the unique environment with the same domain as* $\mathcal{F}$ *such that*

$$\begin{aligned} &\forall f \in \mathrm{dom}(\mathcal{F}), \mathcal{F}\ f = [\lambda x_1 \dots x_n.M, \rho, \mathcal{F}'] \\ &\qquad \textit{implies } \mathcal{F}_*\ f = \left[\lambda x_1 \dots x_n.M, \rho|_{\mathrm{dom}(\rho)\setminus\{x_1 \dots x_n\}}, {\mathcal{F}'}_*\right]. \end{aligned}$$

2.2.3 Optimised reduction rules

Combining both optimisations yields the *optimised* reduction rules (Figure 2, p. 8), used Section 2.4 for the proof of lambda-lifting. We ensure minimal stores by cleaning them in the (val), (var) and (assign) rules, which correspond to tail positions; split environments are introduced in the (call) rule to distinguish fresh parameters, to be cleaned, from captured variables, which are preserved. Tail positions are tracked in every rule through split environments, to avoid cleaning variables too early, in a non-tail branch.

We also build compact closures in the (letrec) rule by removing the parameters of f from the captured environment ρ'.

Theorem 2.13 (Equivalence between naive and optimised reduction rules)**.** *Optimised and naive reduction rules are equivalent: every reduction in one set of rules yields the same result in the other. It is necessary, however, to take care of locations left in the store by the naive reduction:*

$$M\,^{\varepsilon} \underset{\varepsilon}{\overset{\varepsilon|\varepsilon}{\Longrightarrow}} v\,^{\varepsilon} \quad \textit{iff} \quad \exists s, M\,^{\varepsilon} \underset{\varepsilon}{\longrightarrow} v\,^{s}$$

We prove this theorem in Section 2.3.

2.3 Equivalence of optimised and naive reduction rules

This section is devoted to the proof of equivalence between the optimised naive reduction rules (Theorem 2.13).

To clarify the proof, we introduce intermediate reduction rules (Figure 3, p. 9), with only one of the two optimisations: minimal stores, but not compact closures.

The proof then consists in proving that optimised and intermediate rules are equivalent (Lemma 2.15 and Lemma 2.16, Section 2.3.1), then that naive and intermediate rules are equivalent (Lemma 2.21 and Lemma 2.22, Section 2.3.2).

$$\text{Naive rules} \underset{\text{Lemma 2.21}}{\overset{\text{Lemma 2.22}}{\Longleftrightarrow}} \text{Intermediate rules} \underset{\text{Lemma 2.16}}{\overset{\text{Lemma 2.15}}{\Longleftrightarrow}} \text{Optimised rules}$$

$$\text{(VAL)}\ \frac{}{v^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s\backslash\rho_T}} \qquad \text{(VAR)}\ \frac{\rho_T \cdot \rho\ x = l \in \text{dom}\ \ s}{x^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} s\ l^{\,s\backslash\rho_T}}$$

$$\text{(ASSIGN)}\ \frac{a^{\,s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} v^{\,s'} \qquad \rho_T \cdot \rho\ x = l \in \text{dom}\ \ s'}{x := a^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} \mathbf{1}^{\,s'+\{l\mapsto v\}\backslash\rho_T}} \qquad \text{(SEQ)}\ \frac{a^{\,s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} v^{\,s'} \qquad b^{\,s'} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v'^{\,s''}}{a\ ;\ b^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v'^{\,s''}}$$

$$\text{(IF-T.)}\ \frac{a^{\,s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} \mathbf{true}^{\,s'} \qquad b^{\,s'} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s''}}{\mathbf{if}\ a\ \mathbf{then}\ b\ \mathbf{else}\ c^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s''}} \qquad \text{(IF-F.)}\ \frac{a^{\,s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} \mathbf{false}^{\,s'} \qquad c^{\,s'} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s''}}{\mathbf{if}\ a\ \mathbf{then}\ b\ \mathbf{else}\ c^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s''}}$$

$$\text{(LETREC)}\ \frac{b^{\,s} \xRightarrow[\mathcal{F}']{\rho_T|\rho} v^{\,s'} \qquad \rho' = \rho_T \cdot \rho|_{\text{dom}(\rho_T\cdot\rho)\backslash\{x_1\ldots x_n\}} \qquad \mathcal{F}' = \mathcal{F} + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho', \mathcal{F}]\}}{\mathbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \mathbf{in}\ \ b^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s'}}$$

$$\text{(CALL)}\ \frac{\begin{array}{c}\mathcal{F}\,f = [\lambda x_1 \ldots x_n.b, \rho', \mathcal{F}'] \qquad \rho'' = (x_1, l_1) \cdot \ldots \cdot (x_n, l_n) \qquad l_i \text{ fresh and distinct} \\ \forall i, a_i^{\,s_i} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} v_i^{\,s_{i+1}} \qquad b^{\,s_{n+1}+\{l_i\mapsto v_i\}} \xRightarrow[\mathcal{F}'+\{f\mapsto\mathcal{F}\,f\}]{\rho''|\rho'} v^{\,s'}\end{array}}{f(a_1 \ldots a_n)^{\,s_1} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s'\backslash\rho_T}}$$

Figure 2: Optimised reduction rules

2.3.1 Optimised and intermediate reduction rules equivalence

In this section, we show that optimised and intermediate reduction rules are equivalent:

$$\text{Intermediate rules} \underset{\text{Lemma 2.16}}{\overset{\text{Lemma 2.15}}{\Longleftrightarrow}} \text{Optimised rules}$$

We must therefore show that it is correct to use compact closures in the optimised reduction rules.

Compact closures carry the implicit idea that some variables can be safely discarded from the environments when we know for sure that they will be hidden. The following lemma formalises this intuition.

Lemma 2.14 (Hidden variables elimination)**.**

$$\forall l, l', M^{\,s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v^{\,s'} \quad \textit{iff} \quad M^{\,s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|(x,l')\cdot\rho} v^{\,s'}$$

$$\forall l, l', M^{\,s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v^{\,s'} \quad \textit{iff} \quad M^{\,s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|(x,l')\cdot\rho} v^{\,s'}$$

Moreover, both derivations have the same height.

Proof. The exact same proof holds for both intermediate and optimised reduction rules.

By induction on the structure of the derivation. The proof relies solely on the fact that $\rho_T \cdot (x, l) \cdot \rho = \rho_T \cdot (x, l) \cdot (x, l') \cdot \rho$.

$$(\textsc{val})\ \frac{}{v^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s\backslash\rho_T}} \qquad (\textsc{var})\ \frac{\rho_T \cdot \rho\ x = l \in \mathrm{dom}\ s}{x^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} s\ l^{\,s\backslash\rho_T}}$$

$$(\textsc{assign})\ \frac{a^{\,s} \overset{|\rho_T\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s'} \qquad \rho_T \cdot \rho\ x = l \in \mathrm{dom}\ s'}{x := a^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} \mathbf{1}^{\,s'+\{l\mapsto v\}\backslash\rho_T}} \qquad (\textsc{seq})\ \frac{a^{\,s} \overset{|\rho_T\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s'} \qquad b^{\,s'} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,s''}}{a\ ;\ b^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,s''}}$$

$$(\textsc{if-t.})\ \frac{a^{\,s} \overset{|\rho_T\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} \mathbf{true}^{\,s'} \qquad b^{\,s'} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s''}}{\mathbf{if}\ a\ \mathbf{then}\ b\ \mathbf{else}\ c^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s''}} \qquad (\textsc{if-f.})\ \frac{a^{\,s} \overset{|\rho_T\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} \mathbf{false}^{\,s'} \qquad c^{\,s'} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s''}}{\mathbf{if}\ a\ \mathbf{then}\ b\ \mathbf{else}\ c^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s''}}$$

$$(\textsc{letrec})\ \frac{b^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}'}{\Longrightarrow}} v^{\,s'} \qquad \rho' = \rho_T \cdot \rho \qquad \mathcal{F}' = \mathcal{F} + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho, \mathcal{F}]\}}{\mathbf{letrec}\ f(x_1 \ldots x_n) = a\ \mathbf{in}\ b^{\,s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s'}}$$

$$(\textsc{call})\ \frac{\mathcal{F}\,f = [\lambda x_1 \ldots x_n.b, \rho', \mathcal{F}'] \qquad \rho'' = (x_1, l_1) \cdot \ldots \cdot (x_n, l_n) \qquad l_i \text{ fresh and distinct} \qquad \forall i, a_i^{\,s_i} \overset{|\rho_T\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v_i^{\,s_{i+1}} \qquad b^{\,s_{n+1}+\{l_i\mapsto v_i\}} \overset{\rho''|\rho'}{\underset{\mathcal{F}'+\{f\mapsto\mathcal{F}\,f\}}{\Longrightarrow}} v^{\,s'}}{f(a_1 \ldots a_n)^{\,s_1} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s'\backslash\rho_T}}$$

Figure 3: Intermediate reduction rules

(seq) $\rho_T \cdot (x,l) \cdot \rho = \rho_T \cdot (x,l) \cdot (x,l') \cdot \rho$. So,

$$a^{\,s} \overset{|\rho_T\cdot(x,l)\cdot(x,l')\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s'} \quad \text{iff} \quad a^{\,s} \overset{|\rho_T\cdot(x,l)\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s'}$$

Moreover, by the induction hypotheses,

$$b^{\,s'} \overset{\rho_T\cdot(x,l)|(x,l')\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,s''} \quad \text{iff} \quad b^{\,s'} \overset{\rho_T\cdot(x,l)|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,s''}$$

Hence,

$$a\ ;\ b^{\,s} \overset{\rho_T\cdot(x,l)|(x,l')\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,s''} \quad \text{iff} \quad a\ ;\ b^{\,s} \overset{\rho_T\cdot(x,l)|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,s''}$$

The other cases are similar.

(val) $v^{\,s} \overset{\rho_T\cdot(x,l)|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s\backslash\rho_T\cdot(x,l)}$ iff $v^{\,s} \overset{\rho_T\cdot(x,l)|(x,l')\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,s\backslash\rho_T\cdot(x,l)}$

(var) $\rho_T \cdot (x,l) \cdot \rho = \rho_T \cdot (x,l) \cdot (x,l') \cdot \rho$ so, with $l'' = \rho_T \cdot (x,l) \cdot \rho\ y$,

$$y^{\,s} \overset{\rho_T\cdot(x,l)|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} s\ l''^{\,s\backslash\rho_T\cdot(x,l)} \quad \text{iff} \quad y^{\,s} \overset{\rho_T\cdot(x,l)|(x,l')\cdot\rho}{\underset{\mathcal{F}}{\Longrightarrow}} s\ l''^{\,s\backslash\rho_T\cdot(x,l)}$$

(assign) $\rho_T \cdot (x,l) \cdot \rho = \rho_T \cdot (x,l) \cdot (x,l') \cdot \rho$. So,

$$a\,^{s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot(x,l)\cdot(x,l')\cdot\rho} v\,^{s'} \quad \text{iff} \quad a\,^{s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot(x,l)\cdot\rho} v\,^{s'}$$

Hence, with $l'' = \rho_T \cdot (x,l) \cdot \rho\ y$,

$$y \coloneqq a\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} \mathbf{1}\,^{s'+\{l''\mapsto v\}\setminus\rho_T\cdot(x,l)} \quad \text{iff} \quad y \coloneqq a\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|(x,l')\cdot\rho} \mathbf{1}\,^{s'+\{l''\mapsto v\}\setminus\rho_T\cdot(x,l)}$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) $\rho_T \cdot (x,l) \cdot \rho = \rho_T \cdot (x,l) \cdot (x,l') \cdot \rho = \rho'$. Moreover, by the induction hypotheses,

$$b\,^{s} \xRightarrow[\mathcal{F}']{\rho_T\cdot(x,l)|(x,l')\cdot\rho} v\,^{s'} \quad \text{iff} \quad b\,^{s} \xRightarrow[\mathcal{F}']{\rho_T\cdot(x,l)|\rho} v\,^{s'}$$

Hence,

$$\textbf{letrec}\ f(x_1 \ldots x_n) = a\ \textbf{in}\ b\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|(x,l')\cdot\rho} v\,^{s'} \quad \text{iff}$$

$$\textbf{letrec}\ f(x_1 \ldots x_n) = a\ \textbf{in}\ b\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s'}$$

(call) $\rho_T \cdot (x,l) \cdot \rho = \rho_T \cdot (x,l) \cdot (x,l') \cdot \rho$. So,

$$\forall i, a_i^{\,s_i} \xRightarrow[\mathcal{F}]{|\rho_T\cdot(x,l)\cdot(x,l')\cdot\rho} v_i^{\,s_{i+1}} \quad \text{iff} \quad a_i^{\,s_i} \xRightarrow[\mathcal{F}]{|\rho_T\cdot(x,l)\cdot\rho} v_i^{\,s_{i+1}}$$

Hence,

$$f(a_1 \ldots a_n)\,^{s_1} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|(x,l')\cdot\rho} v\,^{s'\setminus\rho_T\cdot(x,l)} \quad \text{iff} \quad f(a_1 \ldots a_n)\,^{s_1} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s'\setminus\rho_T\cdot(x,l)}. \qquad \square$$

Now we can show the required lemmas and prove the equivalence between the intermediate and optimised reduction rules.

Lemma 2.15 (Intermediate implies optimised)**.**

$$\textit{If}\ M\,^{s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\,^{s'}\ \textit{then}\ M\,^{s} \xRightarrow[\mathcal{F}_*]{\rho_T|\rho} v\,^{s'}.$$

Proof. By induction on the structure of the derivation. The interesting cases are (letrec) and (call), where compact environments are respectively built and used.

(letrec) By the induction hypotheses,

$$b\,^{s} \xRightarrow[{\mathcal{F}'}_*]{\rho_T|\rho} v\,^{s'}$$

Since we defined canonical compact environments so as to match exactly the way compact environments are built in the optimised reduction rules, the constraints of the (letrec) rule are fulfilled:

$$\mathcal{F}'_* = \mathcal{F}_* + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho', \mathcal{F}_*]\},$$

hence:

$$\textbf{letrec}\ f(x_1 \ldots x_n) = a\ \textbf{in}\ b\,^{s} \xRightarrow[\mathcal{F}_*]{\rho_T|\rho} v\,^{s'}$$

(call) By the induction hypotheses,

$$\forall i, a_i^{\,s_i} \underset{\mathcal{F}_*}{\overset{|\rho_T \cdot \rho}{\Longrightarrow}} v_i^{\,s_{i+1}}$$

and

$$b^{\,s_{n+1}+\{l_i \mapsto v_i\}} \underset{(\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\})_*}{\overset{\rho''|\rho'}{\Longrightarrow}} v^{\,s'}$$

Lemma 2.14 allows to remove hidden variables, which leads to

$$b^{\,s_{n+1}+\{l_i \mapsto v_i\}} \underset{(\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\})_*}{\overset{\rho''|\rho'_{|\,\mathrm{dom}(\rho')\setminus\{x_1 \dots x_n\}}}{\Longrightarrow}} v^{\,s'}$$

Besides,

$$\mathcal{F}_*\ f = \left[\lambda x_1 \dots x_n.b, \rho'_{|\,\mathrm{dom}(\rho')\setminus\{x_1 \dots x_n\}}, {\mathcal{F}'}_*\right]$$

and

$$(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_* = {\mathcal{F}'}_* + \{f \mapsto \mathcal{F}_*\ f\}$$

Hence

$$f(a_1 \dots a_n)^{\,s_1} \underset{\mathcal{F}_*}{\overset{\rho_T|\rho}{\Longrightarrow}} v^{\,s' \setminus \rho_T}.$$

(val) $v^{\,s} \underset{\mathcal{F}_*}{\overset{\rho_T|\rho}{\Longrightarrow}} v^{\,s \setminus \rho_T}$

(var) $x^{\,s} \underset{\mathcal{F}_*}{\overset{\rho_T|\rho}{\Longrightarrow}} s\ l^{\,s \setminus \rho_T}$

(assign) By the induction hypotheses, $a^{\,s} \underset{\mathcal{F}_*}{\overset{|\rho_T \cdot \rho}{\Longrightarrow}} v^{\,s'}$. Hence,

$$x := a^{\,s} \underset{\mathcal{F}_*}{\overset{\rho_T|\rho}{\Longrightarrow}} \mathbf{1}^{\,s'+\{l \mapsto v\} \setminus \rho_T}$$

(seq) By the induction hypotheses,

$$a^{\,s} \underset{\mathcal{F}_*}{\overset{|\rho_T \cdot \rho}{\Longrightarrow}} v^{\,s'} \qquad\qquad b^{\,s'} \underset{\mathcal{F}_*}{\overset{\rho_T|\rho}{\Longrightarrow}} v'^{\,s''}$$

Hence,

$$a\ ;\ b^{\,s} \underset{\mathcal{F}_*}{\overset{\rho_T|\rho}{\Longrightarrow}} v'^{\,s''}$$

(if-true) and (if-false) are proved similarly to (seq). □

Lemma 2.16 (Optimised implies intermediate)**.**

$$\textit{If } M^{\,s} \underset{\mathcal{F}}{\overset{\rho_T|\rho}{\Longrightarrow}} v^{\,s'} \textit{ then } \forall \mathcal{G} \textit{ such that } \mathcal{G}_* = \mathcal{F}, M^{\,s} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} v^{\,s'}.$$

Proof. First note that, since $\mathcal{G}_* = \mathcal{F}$, $\mathcal{F}$ is necessarily compact.

By induction on the structure of the derivation. The interesting cases are (letrec) and (call), where non-compact environments are respectively built and used.

(letrec) Let $\mathcal{G}$ such as $\mathcal{G}_* = \mathcal{F}$. Remember that $\rho' = \rho_T \cdot \rho|_{\mathrm{dom}(\rho_T \cdot \rho) \setminus \{x_1 \ldots x_n\}}$. Let

$$\mathcal{G}' = \mathcal{G} + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho_T \cdot \rho, \mathcal{F}]\}$$

which leads, since $\mathcal{F}$ is compact ($\mathcal{F}_* = \mathcal{F}$), to

$$\begin{aligned} \mathcal{G}'_* &= \mathcal{F} + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho', \mathcal{F}]\} \\ &= \mathcal{F}' \end{aligned}$$

By the induction hypotheses,

$$b\,^{s} \underset{\mathcal{G}'}{\overset{\rho_T|\rho}{\Longrightarrow}} v\,^{s'}$$

Hence,

$$\textbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \textbf{in}\ \ b\,^{s} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} v\,^{s'}$$

(call) Let $\mathcal{G}$ such as $\mathcal{G}_* = \mathcal{F}$. By the induction hypotheses,

$$\forall i, a_i^{\,s_i} \underset{\mathcal{G}}{\overset{|\rho_T \cdot \rho}{\Longrightarrow}} v_i^{\,s_{i+1}}$$

Moreover, since $\mathcal{G}_*\ f = \mathcal{F}\, f$,

$$\mathcal{G}\ f = [\lambda x_1 \ldots x_n.b, (x_i, l_i) \cdot \ldots \cdot (x_j, l_j)\rho', \mathcal{G}']$$

where $\mathcal{G}'_* = \mathcal{F}'$, and the l_i are some locations stripped out when compacting $\mathcal{G}$ to get $\mathcal{F}$. By the induction hypotheses,

$$b\,^{s_{n+1}+\{l_i \mapsto v_i\}} \underset{\mathcal{G}'+\{f \mapsto \mathcal{G}\ f\}}{\overset{\rho''|\rho'}{\Longrightarrow}} v\,^{s'}$$

Lemma 2.14 leads to

$$b\,^{s_{n+1}+\{l_i \mapsto v_i\}} \underset{\mathcal{G}'+\{f \mapsto \mathcal{G}\ f\}}{\overset{\rho''|(x_i,l_i)\cdot\ldots\cdot(x_j,l_j)\rho'}{\Longrightarrow}} v\,^{s'}$$

Hence,

$$f(a_1 \ldots a_n)\,^{s_1} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} v\,^{s' \setminus \rho_T}.$$

(val) $\forall \mathcal{G}$ such as $\mathcal{G}_* = \mathcal{F}, v\,^{s} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} v\,^{s'}$

(var) $\forall \mathcal{G}$ such as $\mathcal{G}_* = \mathcal{F}, x\,^{s} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} s\ l\,^{s \setminus \rho_T}$

(assign) Let $\mathcal{G}$ such as $\mathcal{G}_* = \mathcal{F}$. By the induction hypotheses, $a\,^{s} \underset{\mathcal{G}}{\overset{|\rho_T \cdot \rho}{\Longrightarrow}} v\,^{s'}$. Hence,

$$x := a\,^{s} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} \mathbf{1}\,^{s'+\{l \mapsto v\} \setminus \rho_T}$$

(seq) Let $\mathcal{G}$ such as $\mathcal{G}_* = \mathcal{F}$. By the induction hypotheses,

$$a\,^{s} \underset{\mathcal{G}}{\overset{|\rho_T \cdot \rho}{\Longrightarrow}} v\,^{s'} \qquad\qquad b\,^{s'} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} v'\,^{s''}$$

Hence

$$a\ ;\ b\,^{s} \underset{\mathcal{G}}{\overset{\rho_T|\rho}{\Longrightarrow}} v'\,^{s''}$$

(if-true) and (if-false) are proved similarly to (seq). □

2.3.2 Intermediate and naive reduction rules equivalence

In this section, we show that the naive and intermediate reduction rules are equivalent:

$$\text{Naive rules} \underset{\text{Lemma 2.21}}{\overset{\text{Lemma 2.22}}{\Longleftrightarrow}} \text{Intermediate rules}$$

We must therefore show that it is correct to use minimal stores in the intermediate reduction rules. We first define a partial order on stores:

Definition 2.17 (Store extension)**.**

$$s \sqsubseteq s' \quad \textit{iff} \quad s'|_{\mathrm{dom}(s)} = s$$

Property 2.18. *Store extension ($\sqsubseteq$) is a partial order over stores. The following operations preserve this order: $\cdot \setminus \rho$ and $\cdot + \{l \mapsto v\}$, for some given ρ, l and v.*

Proof. Immediate when considering the stores as function graphs: $\sqsubseteq$ is the inclusion, $\cdot \setminus \rho$ a relative complement, and $\cdot + \{l \mapsto v\}$ a disjoint union (preceded by $\cdot \setminus (l, v')$ when l is already bound to some v'). □

Before we prove that using minimal stores is equivalent to using full stores, we need an alpha-conversion lemma, which allows us to rename locations in the store, provided the new location does not already appear in the store or the environments. It is used when choosing a fresh location for the (call) rule in proofs by induction.

Lemma 2.19 (Alpha-conversion)**.** *If $M\,^{s} \underset{\mathcal{F}}{\overset{\rho_T|\rho}{\Longrightarrow}} v\,^{s'}$ then, for all l, for all l' appearing neither in s nor in $\mathcal{F}$ nor in $\rho \cdot \rho_T$,*

$$M\,^{s[l'/l]} \underset{\mathcal{F}[l'/l]}{\overset{\rho_T[l'/l]|\rho[l'/l]}{\Longrightarrow}} v\,^{s'[l'/l]}.$$

Moreover, both derivations have the same height.

Proof. By induction on the height of the derivation. For the (call) case, we must ensure that the fresh locations l_i do not clash with l'. In case they do, we conclude by applying the induction hypotheses twice: first to rename the clashing l_i into a fresh l_i', then to rename l into l'.

Two preliminary elementary remarks. First, provided l' appears neither in ρ or ρ_T, nor in s,

$$(s \setminus \rho)[l'/l] = (s[l'/l]) \setminus (\rho[l'/l])$$

and

$$(\rho_T \cdot \rho)[l'/l] = \rho_T[l'/l] \cdot \rho[l'/l].$$

Moreover, if $M\,^{s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{s'}$, then $\mathrm{dom}(s') = \mathrm{dom}(s) \setminus \rho_T$ (straightforward by induction). This leads to: $\rho_T = \varepsilon \Rightarrow \mathrm{dom}(s') = \mathrm{dom}(s)$.

By induction on the height of the derivation, because the induction hypothesis must be applied twice in the case of the (call) rule.

(call) $\forall i, \mathrm{dom}(s_i) = \mathrm{dom}(s_{i+1})$. Thus, $\forall i, l' \notin \mathrm{dom}(s_i)$. This leads, by the induction hypotheses, to

$$\forall i, a_i\,^{s_i[l'/l]} \overset{|(\rho_T \cdot \rho)[l'/l]}{\underset{\mathcal{F}}{\Longrightarrow}} v_i\,^{s_{i+1}[l'/l]} \mathcal{F}[l'/l]$$

Moreover , $\mathcal{F}'$ is part of $\mathcal{F}$. As a result, since l' does not appear in $\mathcal{F}$, it does not appear in $\mathcal{F}'$, nor in $\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}$. It does not appear in ρ' either (since ρ' is part of $\mathcal{F}'$). On the other hand, there might be some j such that $l_j = l'$, so l' might appear in ρ''. In that case, we apply the induction hypotheses a first time to rename l_j in some $l'_j \neq l'$. One can chose l'_j such that it does not appear in s_{n+1}, $\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}$ nor in $\rho'' \cdot \rho$. As a result, l'_j is fresh. Since l_j is fresh too, and does not appear in $\mathrm{dom}(s')$ (because of our preliminary remarks), this leads to a mere substitution in ρ'':

$$b\,^{s_{n+1}+\{l_i[l'_j/l_j] \mapsto v_i\}} \overset{\rho''[l'_j/l_j]|\rho'}{\underset{\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\}}{\Longrightarrow}} v\,^{s'}$$

Once this (potentially) disturbing l_j has been renamed (we ignore it in the rest of the proof), we apply the induction hypotheses a second time to rename l to l':

$$b\,^{(s_{n+1}+\{l_i \mapsto v_i\})[l'/l]} \overset{\rho''[l'/l]|\rho'[l'/l]}{\underset{\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\}}{\Longrightarrow}} v\,^{s'[l'/l]}$$

Now, $(s_{n+1} + \{l_i \mapsto v_i\})[l'/l] = s_{n+1}[l'/l] + \{l_i \mapsto v_i\}$. Moreover,

$$\mathcal{F}[l'/l]\ f = [\lambda x_1 \ldots x_n.b, \rho'[l'/l], \mathcal{F}'[l'/l]]$$

and

$$(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})[l'/l] = \mathcal{F}'[l'/l] + \{f \mapsto \mathcal{F}[l'/l]\ f\}$$

Finally, $\rho''[l'/l] = \rho''$. Hence:

$$f(a_1 \ldots a_n)\,^{s_1[l'/l]} \overset{\rho_T[l'/l]|\rho[l'/l]}{\underset{\mathcal{F}[l'/l]}{\Longrightarrow}} v\,^{s'[l'/l] \setminus \rho_T[l'/l]}.$$

(val) $v\,^{s[l'/l]} \overset{\rho_T[l'/l]|\rho[l'/l]}{\underset{\mathcal{F}[l'/]}{\Longrightarrow}} v\,^{s[l'/l] \setminus \rho_T[l'/l]}$

(var) $s[l'/l](\rho_T[l'/l] \cdot \rho[l'/l]\ x) = s(\rho_T \cdot \rho\ x) = v$ implies

$$x\,^{s[l'/l]} \overset{\rho_T[l'/l]|\rho[l'/l]}{\underset{\mathcal{F}[l'/]}{\Longrightarrow}} v\,^{s[l'/l] \setminus \rho_T[l'/l]}$$

(assign) By the induction hypotheses,

$$a^{\,s[l'/l]} \xRightarrow[\mathcal{F}[l'/]]{|(\rho_T\cdot\rho)[l'/l]} v^{\,s'[l'/]}$$

Let $s'' = s' + \{\rho_T \cdot \rho\ x \mapsto v\}$. Then,

$$s'[l'/l] + \{(\rho_T \cdot \rho)[l'/l]\ x \mapsto v\} = s''[l'/l]$$

Hence

$$x := a^{\,s[l'/l]} \xRightarrow[\mathcal{F}[l'/]]{\rho_T[l'/l]|\rho[l'/l]} \mathbf{1}^{\,s''[l'/l]\setminus\rho_T[l'/l]}$$

(seq) By the induction hypotheses,

$$a^{\,s[l'/l]} \xRightarrow[\mathcal{F}[l'/]]{|(\rho_T\cdot\rho)[l'/l]} v^{\,s'[l'/l]}$$

Besides, $\mathrm{dom}(s') = \mathrm{dom}(s)$, therefore $l' \notin \mathrm{dom}(s')$. Then, by the induction hypotheses,

$$b^{\,s'[l'/l]} \xRightarrow[\mathcal{F}[l'/]]{\rho_T[l'/l]|\rho[l'/l]} v'^{\,s''[l'/l]}$$

Hence

$$a\ ;\ b^{\,s[l'/l]} \xRightarrow[\mathcal{F}[l'/]]{\rho_T[l'/l]|\rho[l'/l]} v'^{\,s''[l'/l]}$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) Since l' appears neither in ρ' nor in $\mathcal{F}$, it does not appear in $\mathcal{F}'$ either. By the induction hypotheses,

$$b^{\,s[l'/l]} \xRightarrow[\mathcal{F}'[l'/l]]{\rho_T[l'/l]|\rho[l'/l]} v^{\,s'[l'/l]}$$

Moreover,

$$\mathcal{F}'[l'/l] = \mathcal{F}[l'/l] + \{f \mapsto [\lambda x_1 \ldots x_n.a, \rho'[l'/l], \mathcal{F}]\}$$

Hence

$$\textbf{letrec}\ f(x_1 \ldots x_n) = a\ \textbf{in}\ b^{\,s} \xRightarrow[\mathcal{F}[l'/]]{\rho_T[l'/l]|\rho[l'/l]} v^{\,s'} \qquad \square$$

To prove that using minimal stores is correct, we need to extend them so as to recover the full stores of naive reduction. The following lemma shows that extending a store before an (intermediate) reduction extends the resulting store too:

Lemma 2.20 (Extending a store in a derivation)**.**

$$\textit{Given the reduction } M^{\,s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s'},\ \textit{then } \forall t \sqsupseteq s, \exists t' \sqsupseteq s', M^{\,t} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,t'}.$$

Moreover, both derivations have the same height.

Proof. By induction on the height of the derivation. The most interesting case is (call), which requires alpha-converting a location (hence the induction on the height rather than the structure of the derivation).

(var), (val) and (assign) are straightforward by the induction hypotheses and Property 2.18; (seq), (if-true), (if-false) and (letrec) are straightforward by the induction hypotheses.

(call) Let $t_1 \sqsupseteq s_1$. By the induction hypotheses,

$$\exists t_2 \sqsupseteq s_2, a_1^{\,t_1} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v_1^{\,t_2}$$

$$\exists t_{i+1} \sqsupseteq s_{i+1}, a_i^{\,t_i} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v_i^{\,t_{i+1}}$$

$$\exists t_{n+1} \sqsupseteq s_{n+1}, a_n^{\,t_n} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v_n^{\,t_{n+1}}$$

The locations l_i might belong to $\mathrm{dom}(t_{n+1})$ and thus not be fresh. By alpha-conversion (Lemma 2.19), we chose fresh l_i' (not in $\mathrm{Im}(\rho')$ and $\mathrm{dom}(s')$) such that

$$b^{\,s_{n+1}+\{l_i' \mapsto v_i\}} \overset{(l_i', v_i)|\rho'}{\underset{\mathcal{F}'+\{f \mapsto_{\mathcal{F}} f\}}{\Longrightarrow}} v^{\,s'}$$

By Property 2.18, $t_{n+1} + \{l_i' \mapsto v_i\} \sqsupseteq s_{n+1} + \{l_i' \mapsto v_i\}$. By the induction hypotheses,

$$\exists t' \sqsupseteq s', b^{\,t_{n+1}+\{l_i' \mapsto v_i\}} \overset{(l_i', v_i)|\rho'}{\underset{\mathcal{F}'+\{f \mapsto_{\mathcal{F}} f\}}{\Longrightarrow}} v^{\,t'}$$

Moreover, $t' \setminus \rho_T \sqsupseteq s' \setminus \rho_T$. Hence,

$$f(a_1 \ldots a_n)^{\,t_1} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,t' \setminus \rho_T}.$$

(var) Let $t \sqsupseteq s$. $v^{\,t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,t \setminus \rho_T}$ and $\exists t' = t \setminus \rho_T \sqsupseteq s \setminus \rho_T = s'$ (Property 2.18).

(val) Let $t \sqsupseteq s$. $x^{\,t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} t\ l^{\,t \setminus \rho_T}$ and $\exists t' = t \setminus \rho_T \sqsupseteq s \setminus \rho_T = s'$ (Property 2.18). Moreover, $t\ l = s\ l$ because $l \in \mathrm{dom}(s)$ and $t|_{\mathrm{dom}(s)} = s$.

(assign) Let $t \sqsupseteq s$. By the induction hypotheses,

$$\exists t' \sqsupseteq s', a^{\,t} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,t'}$$

Hence,

$$x \coloneqq a^{\,t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} \mathbf{1}^{\,t'+\{l \mapsto v\} \setminus \rho_T}$$

concludes, since $t' + \{l \mapsto v\} \setminus \rho_T \sqsupseteq t' + \{l \mapsto v\} \setminus \rho_T$ (Property 2.18).

(seq) Let $t \sqsupseteq s$. By the induction hypotheses,

$$\exists t' \sqsupseteq s', a^{\,t} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v^{\,t'}$$

$$\exists t'' \sqsupseteq s'', b^{\,t'} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,t''}$$

Hence,

$$\exists t'' \sqsupseteq s'', a\ ;\ b^{\,t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'^{\,t''}$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) Let $t \sqsupseteq s$. By the induction hypotheses,

$$\exists t' \sqsupseteq s', b\,^{s} \overset{\rho_T|\rho}{\underset{\mathcal{F}'}{\Longrightarrow}} v\,^{s'}$$

Hence,

$$\exists t' \sqsupseteq s', \textbf{letrec } f(x_1 \dots x_n) = a \textbf{ in } b\,^{s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{t'} \qquad \square$$

Now we can show the required lemmas and prove the equivalence between the intermediate and naive reduction rules.

Lemma 2.21 (Intermediate implies naive)**.**

$$\textit{If } M\,^{s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{s'} \textit{ then } \exists t' \sqsupseteq s', M\,^{s} \xrightarrow[\mathcal{F}]{\rho_T \cdot \rho} v\,^{t'}.$$

Proof. By induction on the height of the derivation, because some stores are modified during the proof. The interesting cases are (seq) and (call), where Lemma 2.20 is used to extend intermediary stores. Other cases are straightforward by Property 2.18 and the induction hypotheses.

(seq) By the induction hypotheses,

$$\exists t' \sqsupseteq s', a\,^{s} \xrightarrow[\mathcal{F}]{\rho} v\,^{t'}.$$

Moreover,

$$b\,^{s'} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'\,^{s''}.$$

Since $t' \sqsupseteq s'$, Lemma 2.20 leads to:

$$\exists t \sqsupseteq s'', b\,^{t'} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'\,^{t}$$

and the height of the derivation is preserved. By the induction hypotheses,

$$\exists t'' \sqsupseteq t, b\,^{t'} \xrightarrow[\mathcal{F}]{\rho} v'\,^{t''}$$

Hence, since $\sqsubseteq$ is transitive (Property 2.18),

$$\exists t'' \sqsupseteq s'', a \ ; \ b\,^{s} \xrightarrow[\mathcal{F}]{\rho} v'\,^{t''}.$$

(call) Similarly to the (seq) case, we apply the induction hypotheses and Lemma 2.20:

$$\exists t_2 \sqsupseteq s_2, a_1^{s_1} \xrightarrow[\mathcal{F}]{\rho} v_1^{t_2} \qquad \text{(Induction)}$$

$$\exists t'_{i+1} \sqsupseteq s_{i+1}, a_i^{t_i} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v_i^{t'_{i+1}} \qquad \text{(Lemma 2.20)}$$

$$\exists t_{i+1} \sqsupseteq t'_{i+1} \sqsupseteq s_{i+1}, a_i^{t_i} \xrightarrow[\mathcal{F}]{\rho} v_i^{t_{i+1}} \qquad \text{(Induction)}$$

$$\exists t'_{n+1} \sqsupseteq s_{n+1}, a_n^{t_n} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v_n^{t'_{n+1}} \qquad \text{(Lemma 2.20)}$$

$$\exists t_{n+1} \sqsupseteq t'_{n+1} \sqsupseteq s_{n+1}, a_n^{t_n} \xrightarrow[\mathcal{F}]{\rho} v_n^{t_{n+1}} \qquad \text{(Induction)}$$

The locations l_i might belong to $\mathrm{dom}(t_{n+1})$ and thus not be fresh. By alpha-conversion (Lemma 2.19), we choose a set of fresh l'_i (not in $\mathrm{Im}(\rho')$ and $\mathrm{dom}(s')$) such that

$$b\,^{s_{n+1}+\{l'_i\mapsto v_i\}} \xRightarrow[\mathcal{F}'+\{f\mapsto\mathcal{F}\,f\}]{(l'_i,v_i)|\rho'} v\,^{s'}.$$

By Property 2.18, $t_{n+1}+\{l'_i\mapsto v_i\} \sqsupseteq s_{n+1}+\{l'_i\mapsto v_i\}$. Lemma 2.20 leads to,

$$\exists t \sqsupseteq s', b\,^{t_{n+1}+\{l'_i\mapsto v_i\}} \xRightarrow[\mathcal{F}'+\{f\mapsto\mathcal{F}\,f\}]{(l'_i,v_i)|\rho'} v\,^{t}.$$

By the induction hypotheses,

$$\exists t' \sqsupseteq t \sqsupseteq s', b\,^{t_{n+1}+\{l'_i\mapsto v_i\}} \xrightarrow[\mathcal{F}'+\{f\mapsto\mathcal{F}\,f\}]{(l'_i,v_i)\cdot\rho'} v\,^{t'}.$$

Moreover, $t' \setminus \rho_T \sqsupseteq s' \setminus \rho_T$. Hence,

$$f(a_1 \dots a_n)\,^{s_1} \xrightarrow[\mathcal{F}]{\rho} v\,^{t'\setminus\rho_T}.$$

(val) $v\,^{s} \xrightarrow[\mathcal{F}]{\rho} v\,^{t'}$ with $t' = s \sqsupseteq s \setminus \rho_T = s'$.

(var) $x\,^{s} \xrightarrow[\mathcal{F}]{\rho} s\;l\,^{s''}$ with $t' = s \sqsupseteq s \setminus \rho_T = s'$.

(assign) By the induction hypotheses,

$$\exists s'' \sqsupseteq s', a\,^{s} \xrightarrow[\mathcal{F}]{\rho} v\,^{t'}$$

Hence,

$$x \coloneqq a\,^{s} \xrightarrow[\mathcal{F}]{\rho} \mathbf{1}\,^{t'+\{l\mapsto v\}}$$

concludes since $t'+\{l\mapsto v\} \sqsupseteq s'+\{l\mapsto v\}$ (Property 2.18).

(if-true) and (if-false) are proved similarly to (seq).

(letrec) By the induction hypotheses,

$$\exists t' \sqsupseteq s', b\,^{s} \xrightarrow[\mathcal{F}']{\rho} v\,^{s'}.$$

Hence,

$$\exists t' \sqsupseteq s', \mathbf{letrec}\ f(x_1 \dots x_n) = a\ \mathbf{in}\ b\,^{s} \xrightarrow[\mathcal{F}]{\rho} v\,^{t'}. \qquad \square$$

The proof of the converse property — i.e. if a term reduces in the naive reduction rules, it reduces in the intermediate reduction rules too — is more complex because the naive reduction rules provide very weak invariants about stores and environments. For that reason, we add an hypothesis to ensure that every location appearing in the environments ρ, ρ_T and $\mathcal{F}$ also appears in the store s:

$$\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(s).$$

Moreover, since stores are often larger in the naive reduction rules than in the intermediate ones, we need to generalise the induction hypothesis.

Lemma 2.22 (Naive implies intermediate)**.** *Assume* $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(s)$. *Then,* $M\,^{s} \xrightarrow[\mathcal{F}]{\rho_T \cdot \rho} v\,^{s'}$ *implies*

$$\forall t \sqsubseteq s \text{ such that } \mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(t), \quad M\,^{t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{s'|_{\mathrm{dom}(t)\setminus \mathrm{Im}(\rho_T)}}.$$

Proof. By induction on the structure of the derivation.

(val) Let $t \sqsubseteq s$. Then

$$\begin{aligned} t \setminus \rho_T &= s|_{\mathrm{dom}(t)\setminus \mathrm{Im}(\rho_T)} && \text{because } s|_{\mathrm{dom}(t)} = t \\ &= s'|_{\mathrm{dom}(t)\setminus \mathrm{Im}(\rho_T)} && \text{because } s' = s \end{aligned}$$

Hence,

$$v\,^{t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{t\setminus \rho_T}.$$

(var) Let $t \sqsubseteq s$ such that $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(t)$. Note that $l \in \mathrm{Im}(\rho_T \cdot \rho) \subset \mathrm{dom}(t)$ implies $t\ l = s\ l$. Then,

$$\begin{aligned} t \setminus \rho_T &= s|_{\mathrm{dom}(t)\setminus \mathrm{Im}(\rho_T)} && \text{because } s|_{\mathrm{dom}(t)} = t \\ &= s'|_{\mathrm{dom}(t)\setminus \mathrm{Im}(\rho_T)} && \text{because } s' = s \end{aligned}$$

Hence,

$$x\,^{t} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} t\ l\,^{t\setminus \rho_T}.$$

(assign) Let $t \sqsubseteq s$ such that $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(t)$. By the induction hypotheses, since $\mathrm{Im}(\varepsilon) = \emptyset$,

$$a\,^{t} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{s'|_{\mathrm{dom}(t)}}$$

Note that $l \in \mathrm{Im}(\rho_T \cdot \rho) \subset \mathrm{dom}(t)$ implies $l \in \mathrm{dom}(s'|_{\mathrm{dom}(t)})$. Then

$$\begin{aligned} (s'|_{\mathrm{dom}(t)} + \{l \mapsto v\}) \setminus \rho_T &= (s' + \{l \mapsto v\})|_{\mathrm{dom}(t)} \setminus \rho_T && \text{because } l \in \mathrm{dom}(s'|_{\mathrm{dom}(t)}) \\ &= (s' + \{l \mapsto v\})|_{\mathrm{dom}(t)\setminus \mathrm{Im}(\rho_T)} \end{aligned}$$

Hence,

$$x \coloneqq a\,^{s} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} \mathbf{1}\,^{(s'|_{\mathrm{dom}(t)} + \{l \mapsto v\})\setminus \rho_T}.$$

(seq) Let $t \sqsubseteq s$ such that $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(t)$. By the induction hypotheses, since $\mathrm{Im}(\varepsilon) = \emptyset$,

$$a\,^{t} \overset{|\rho_T \cdot \rho}{\underset{\mathcal{F}}{\Longrightarrow}} v\,^{s'|_{\mathrm{dom}(t)}}$$

Moreover, $s'|_{\mathrm{dom}(t)} \sqsubseteq s'$ and $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(s'|_{\mathrm{dom}(t)}) = \mathrm{dom}(t)$. By the induction hypotheses, this leads to:

$$b\,^{s'|_{\mathrm{dom}(t)}} \overset{\rho_T|\rho}{\underset{\mathcal{F}}{\Longrightarrow}} v'\,^{s''|_{\mathrm{dom}(s'|_{\mathrm{dom}(t)})\setminus \mathrm{Im}(\rho_T)}}.$$

Hence, with $\mathrm{dom}(s'|_{\mathrm{dom}(t)}) = \mathrm{dom}(t)$,

$$a \ ; \ b^{\,t} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v'^{\,s''|_{\mathrm{dom}(t)\setminus\mathrm{Im}(\rho_T)}}.$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) Let $t \sqsubseteq s$ such that $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(t)$.

$$\mathrm{Loc}(\mathcal{F}') = \mathrm{Loc}(\mathcal{F}) \cup \mathrm{Im}(\rho_T \cdot \rho) \text{ implies } \mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}') \subset \mathrm{dom}(t).$$

Then, by the induction hypotheses,

$$b^{\,t} \xRightarrow[\mathcal{F}']{\rho_T|\rho} v^{\,s'|_{\mathrm{dom}(t)\setminus\mathrm{Im}(\rho_T)}}.$$

Hence,

$$\textbf{letrec } f(x_1 \dots x_n) = a \textbf{ in } b^{\,t} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s'|_{\mathrm{dom}(t)\setminus\mathrm{Im}(\rho_T)}}.$$

(call) Let $t \sqsubseteq s_1$ such that $\mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(t)$. Note the following equalities:

$$\begin{aligned} s_1|_{\mathrm{dom}(t)} &= t \\ s_2|_{\mathrm{dom}(t)} &\sqsubseteq s_2 \\ \mathrm{Im}(\rho_T \cdot \rho) \cup \mathrm{Loc}(\mathcal{F}) \subset \mathrm{dom}(s_2|_{\mathrm{dom}(t)}) &= \mathrm{dom}(t) \\ s_3|_{\mathrm{dom}(s_2|_{\mathrm{dom}(t)})} &= s_3|_{\mathrm{dom}(t)} \end{aligned}$$

By the induction hypotheses, they yield:

$$\begin{aligned} a_1^{\,t} &\xRightarrow[\mathcal{F}]{|\rho_T \cdot \rho} v_1^{\,s_2|_{\mathrm{dom}(t)}} \\ a_2^{\,s_2|_{\mathrm{dom}(t)}} &\xRightarrow[\mathcal{F}]{|\rho_T \cdot \rho} v_1^{\,s_3|_{\mathrm{dom}(t)}} \\ \forall i, a_i^{\,s_i|_{\mathrm{dom}(t)}} &\xRightarrow[\mathcal{F}]{|\rho_T \cdot \rho} v_i^{\,s_{i+1}|_{\mathrm{dom}(t)}} \end{aligned}$$

Moreover, $s_{n+1}|_{\mathrm{dom}(t)} \sqsubseteq s_{n+1}$ implies $s_{n+1}|_{\mathrm{dom}(t)} + \{l_i \mapsto v_i\} \sqsubseteq s_{n+1} + \{l_i \mapsto v_i\}$ (Property 2.18) and:

$$\begin{aligned} \mathrm{Im}(\rho'' \cdot \rho') \cup \mathrm{Loc}(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}) &= \mathrm{Im}(\rho'') \cup (\mathrm{Im}(\rho') \cup \mathrm{Loc}(\mathcal{F}')) \\ &\subset \{l_i\} \cup \mathrm{Loc}(\mathcal{F}) \\ &\subset \{l_i\} \cup \mathrm{dom}(t) \\ &\subset \mathrm{dom}(s_{n+1}|_{\mathrm{dom}(t)} + \{l_i \mapsto v_i\}) \end{aligned}$$

Then, by the induction hypotheses,

$$b^{\,s_{n+1}|_{\mathrm{dom}(t)} + \{l_i \mapsto v_i\}} \xRightarrow[\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}]{\rho''|\rho'} v^{\,s'|_{\mathrm{dom}(s_{n+1}|_{\mathrm{dom}(t)} + \{l_i \mapsto v_i\})\setminus\mathrm{Im}(\rho'')}}$$

Finally,

$$\begin{aligned} s'|_{\mathrm{dom}(s_{n+1}|_{\mathrm{dom}(t)}+\{l_i\mapsto v_i\})\setminus\mathrm{Im}(\rho'')} \setminus \rho_T &= s'|_{\mathrm{dom}(t)\cup\{l_i\}\setminus\{l_i\}} \setminus \rho_T = s'|_{\mathrm{dom}(t)} \setminus \rho_T \\ &= (s' \setminus \rho_T)|_{\mathrm{dom}(t)\setminus\mathrm{Im}(\rho_T)} \quad \text{(by definition of } \cdot \setminus \cdot) \end{aligned}$$

Hence,

$$f(a_1 \dots a_n)^{\,t} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,(s'\setminus\rho_T)|_{\mathrm{dom}(t)\setminus\mathrm{Im}(\rho_T)}}. \qquad \square$$

2.4 Correctness of lambda-lifting

In this section, we prove the correctness of lambda-lifting (Theorem 2.9, p. 5) by induction on the height of the optimised reduction.

Section 2.4.1 defines stronger invariants and rewords the correctness theorem with them. Section 2.4.2 gives an overview of the proof. Sections 2.4.3 and 2.4.4 prove a few lemmas needed for the proof. Section 2.4.5 contains the actual proof of correctness.

2.4.1 Strengthened hypotheses

We need strong induction hypotheses to ensure that key invariants about stores and environments hold at every step. For that purpose, we define *aliasing-free environments*, in which locations may not be referenced by more than one variable, and *local positions*. They yield a strengthened version of liftable parameters (Definition 2.25). We then define lifted environments (Definition 2.26) to mirror the effect of lambda-lifting in lifted terms captured in closures, and finally reformulate the correctness of lambda-lifting in Theorem 2.28 with hypotheses strong enough to be provable directly by induction.

Definition 2.23 (Aliasing)**.** *A set of environments $\mathcal{E}$ is* aliasing-free *when:*

$$\forall \rho, \rho' \in \mathcal{E}, \forall x \in \mathrm{dom}(\rho), \forall y \in \mathrm{dom}(\rho'),\ \rho\ x = \rho'\ y \Rightarrow x = y.$$

By extension, an environment of functions $\mathcal{F}$ is aliasing-free when $\mathrm{Env}(\mathcal{F})$ *is aliasing-free.*

The notion of aliasing-free environments is not an artifact of our small language, but translates a fundamental property of the C semantics: distinct function parameters or local variables are always bound to distinct memory locations (Section 6.2.2, paragraph 6 in ISO/IEC 9899 [3]).

A local position is any position in a term except inner functions. Local positions are used to distinguish functions defined directly in a term from deeper nested functions, because we need to enforce Invariant 3 (Definition 2.25) on the former only.

Definition 2.24 (Local position)**.** Local positions *are defined inductively as follows:*

1. *M is in local position in M, $x := M$, M ; M,* **if** M **then** M **else** M *and $f(M, \dots, M)$.*
2. *N is in local position in* **letrec** $f(x_1 \dots x_n) = M$ **in** N.

We extend the notion of liftable parameter (Definition 2.8, p. 5) to enforce invariants on stores and environments.

Definition 2.25 (Extended liftability)**.** *The parameter x is* liftable *in $(M, \mathcal{F}, \rho_T, \rho)$ when:*

1. *x is defined as the parameter of a function g, either in M or in $\mathcal{F}$,*
2. *in both M and $\mathcal{F}$, inner functions in g, named h_i, are defined and called exclusively:*
   (a) *in tail position in g, or*
   (b) *in tail position in some h_j (with possibly $i = j$), or*

(c) in tail position in M,

3. *for all f defined in local position in M, $x \in \mathrm{dom}(\rho_T \cdot \rho) \Leftrightarrow \exists i, f = h_i$,*
4. *moreover, if h_i is called in tail position in M, then $x \in \mathrm{dom}(\rho_T)$,*
5. *in $\mathcal{F}$, x appears necessarily and exclusively in the environments of the h_i's closures,*
6. *$\mathcal{F}$ contains only compact closures and $\mathrm{Env}(\mathcal{F}) \cup \{\rho, \rho_T\}$ is aliasing-free.*

We also extend the definition of lambda-lifting (Definition 2.6, p. 5) to environments, in order to reflect changes in lambda-lifted parameters captured in closures.

Definition 2.26 (Lifted form of an environment)**.**

$$\textit{If } \mathcal{F}\, f = [\lambda x_1 \dots x_n.b, \rho', \mathcal{F}'] \qquad \textit{then}$$

$$(\mathcal{F})_*\ f = \begin{cases} \left[\lambda x_1 \dots x_n x.\, (b)_*, \rho'|_{\mathrm{dom}(\rho')\setminus\{x\}}, (\mathcal{F}')_*\right] & \textit{when } f = h_i \textit{ for some } i \\ [\lambda x_1 \dots x_n.\, (b)_*, \rho', (\mathcal{F}')_*] & \textit{otherwise} \end{cases}$$

Lifted environments are defined such that a liftable parameter never appears in them. This property will be useful during the proof of correctness.

Lemma 2.27. *If x is a liftable parameter in $(M, \mathcal{F}, \rho_T, \rho)$, then x does not appear in $(\mathcal{F})_*$.*

Proof. Since x is liftable in $(M, \mathcal{F}, \rho_T, \rho)$, it appears exclusively in the environments of h_i. By definition, it is removed when building $(\mathcal{F})_*$. □

These invariants and definitions lead to a correctness theorem with stronger hypotheses.

Theorem 2.28 (Correctness of lambda-lifting)**.** *If x is a liftable parameter in $(M, \mathcal{F}, \rho_T, \rho)$, then*

$$M\,^s \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\,^{s'} \textit{ implies } (M)_*\,^s \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v\,^{s'}$$

Since naive and optimised reductions rules are equivalent (Theorem 2.13, p. 7), the proof of Theorem 2.9 (p. 5) is a direct corollary of this theorem.

Corollary 2.29. *If x is a liftable parameter in M, then*

$$\exists t, M\,^\varepsilon \xrightarrow[\varepsilon]{} v\,^t \textit{ implies } \exists t', (M)_*\,^\varepsilon \xrightarrow[\varepsilon]{} v\,^{t'}.$$

2.4.2 Overview of the proof

With the enhanced liftability definition, we have invariants strong enough to perform a proof by induction of the correctness theorem. This proof is detailed in Section 2.4.5.

The proof is not by structural induction but by induction on the height of the derivation. This is necessary because, even with the stronger invariants, we cannot apply the induction hypotheses directly to the premises in the case of the (call) rule: we have to change the stores and environments, which means rewriting the whole derivation tree, before using the induction hypotheses.

To deal with this most difficult case, we distinguish between calling one of the lifted functions ($f = h_i$) and calling another function (either g, where x is defined, or any other function outside of g). Only the former requires rewriting; the latter follows directly from the induction hypotheses.

In the (call) rule with $f = h_i$, issues arise when reducing the body b of the lifted function. During this reduction, indeed, the store contains a new location l' bound by the environment to the lifted variable x, but also contains the location l which contains the original value of x. Our

goal is to show that the reduction of b implies the reduction of $(b)_*$, with store and environments fulfilling the constraints of the (call) rule.

To obtain the reduction of the lifted body $(b)_*$, we modify the reduction of b in a series of steps, using several lemmas:

- the location l of the free variable x is moved to the tail environment (Lemma 2.30);
- the resulting reduction meets the induction hypotheses, which we apply to obtain the reduction of the lifted body $(b)_*$;
- however, this reduction does not meet the constraints of the optimised reduction rules because the location l is not fresh: we rename it to a fresh location l' to hold the lifted variable (Lemma 2.31);
- finally, since we renamed l to l', we need to reintroduce a location l to hold the original value of x (Lemmas 2.32 and 2.33).

The rewriting lemmas used in the (call) case are shown in Section 2.4.3.

For every other case, the proof consists in checking thoroughly that the induction hypotheses apply, in particular that x is liftable in the premises. These verifications consist in checking Invariants 3 to 6 of the extended liftability definition (Definition 2.25) — Invariants 1 and 2 are obvious enough not to be detailed. To keep the main proof as compact as possible, the most difficult cases of liftability, related to aliasing, are proven in some preliminary lemmas (Section 2.4.4).

One last issue arises during the induction when one of the premises does not contain the lifted variable x. In that case, the invariants do not hold, since they assume the presence of x. But it turns out that in this very case, the lifting function is the identity (since there is no variable to lift) and lambda-lifting is trivially correct.

2.4.3 Rewriting lemmas

Calling a lifted function has an impact on the resulting store: new locations are introduced for the lifted parameters and the earlier locations, which are not modified anymore, are hidden. Because of these changes, the induction hypotheses do not apply directly in the case of the (call) rule for a lifted function h_i. We use the following four lemmas to obtain, through several rewriting steps, a reduction of lifted terms meeting the induction hypotheses.

- Lemma 2.30 shows that moving a variable from the non-tail environment ρ to the tail environment ρ_T does not change the result, but restricts the domain of the store. It is used transform the original free variable x (in the non-tail environment) to its lifted copy (which is a parameter of h_i, hence in the tail environment).
- Lemma 2.31 handles alpha-conversion in stores and is used when choosing a fresh location.
- Lemmas 2.32 and 2.33 finally add into the store and the environment a fresh location, bound to an arbitrary value. It is used to reintroduce the location containing the original value of x, after it has been alpha-converted to l'.

Lemma 2.30 (Switching to tail environment)**.** *If* $M\,{}^{s} \xRightarrow[\mathcal{F}]{\rho_T|(x,l)\cdot\rho} v\,{}^{s'}$ *and* $x \notin \mathrm{dom}(\rho_T)$ *then* $M\,{}^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,{}^{s'|_{\mathrm{dom}(s')\setminus\{l\}}}$*. Moreover, both derivations have the same height.*

Proof. By induction on the structure of the derivation. For the (val), (var), (assign) and (call) cases, we use the fact that $s \setminus \rho_T \cdot (x,l) = s'|_{\mathrm{dom}(s')\setminus\{l\}}$ when $s' = s \setminus \rho_T$.

(val) $v\,{}^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,{}^{s\setminus\rho_T\cdot(x,l)}$ and $s \setminus \rho_T \cdot (x,l) = s'|_{\mathrm{dom}(s')\setminus\{l\}}$ with $s' = s \setminus \rho_T$.

(var) $y\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} s\,l'\,^{s\setminus\rho_T\cdot(x,l)}$ and $s \setminus \rho_T \cdot (x,l) = s'|_{\mathrm{dom}(s')\setminus\{l\}}$, with $l' = \rho_T \cdot (x,l) \cdot \rho\ y$ and $s' = s \setminus \rho_T$.

(assign) By hypothesis, $a\,^{s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot(x,l)\cdot\rho} v\,^{s'}$ hence $y := a\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} \mathbf{1}\,^{s'+\{l'\mapsto v\}\setminus\rho_T\cdot(x,l)}$ and $s' + \{l' \mapsto v\} \setminus \rho_T \cdot (x,l) = s'|_{\mathrm{dom}(s')\setminus\{l\}}$ with $l' = \rho_T \cdot (x,l) \cdot \rho\ y$ and $s' = s' + \{l' \mapsto v\} \setminus \rho_T$.

(seq) By hypothesis, $a\,^{s} \xRightarrow[\mathcal{F}]{|\rho_T\cdot(x,l)\cdot\rho} v\,^{s'}$ and, by the induction hypotheses, $b\,^{s'} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s''|_{\mathrm{dom}(s'')\setminus\{l\}}}$ hence

$$a\ ;\ b\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s''|_{\mathrm{dom}(s'')\setminus\{l\}}}.$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) By the induction hypotheses,

$$b\,^{s} \xRightarrow[\mathcal{F}']{\rho_T\cdot(x,l)|\rho} v\,^{s'|_{\mathrm{dom}(s')\setminus\{l\}}}$$

hence

$$\textbf{letrec}\ f(x_1 \dots x_n) = a\ \textbf{in}\ b\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s'|_{\mathrm{dom}(s')\setminus\{l\}}}$$

(call) The hypotheses do not change, and the conclusion becomes:

$$f(a_1 \dots a_n)\,^{s_1} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s'\setminus\rho_T\cdot(x,l)}$$

as expected, since $s' \setminus \rho_T \cdot (x,l) = s''|_{\mathrm{dom}(s'')\setminus\{l\}}$ with $s'' = s' \setminus \rho_T$ □

Lemma 2.31 (Alpha-conversion)**.** *If* $M\,^{s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\,^{s'}$ *then, for all* l*, for all* l' *appearing neither in* s *nor in* $\mathcal{F}$ *nor in* $\rho \cdot \rho_T$*,*

$$M\,^{s[l'/l]} \xRightarrow[\mathcal{F}[l'/l]]{\rho_T[l'/l]|\rho[l'/l]} v\,^{s'[l'/l]}$$

Moreover, both derivations have the same height.

Proof. See Lemma 2.19, p. 2.19. □

Lemma 2.32 (Spurious location in store)**.** *If* $M\,^{s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\,^{s'}$ *and* k *does not appear in either* s*,* $\mathcal{F}$ *or* $\rho_T \cdot \rho$*, then, for all value* u*,* $M\,^{s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\,^{s'+\{k\mapsto u\}}$*. Moreover, both derivations have the same height.*

Proof. By induction on the height of the derivation. The key idea is to add (k,u) to every store in the derivation tree. A collision might occur in the (call) rule, if there is some j such that $l_j = k$. In that case, we need to rename l_j to some fresh variable $l'_j \neq k$ (by alpha-conversion) before applying the induction hypotheses.

(call) By the induction hypotheses,

$$\forall i, a_i^{\,s_i+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} v_i^{\,s_{i+1}+\{k\mapsto u\}}$$

Because k does not appear in $\mathcal{F}$,

$$k \notin \mathrm{Loc}(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}) \subset \mathrm{Loc}(\mathcal{F})$$

For the same reason, it does not appear in ρ'. On the other hand, there might be a j such that $l_j = k$, so k might appear in ρ''. In that case, we rename l_j in some fresh $l'_j \neq k$, appearing in neither s_{n+1}, nor $\mathcal{F}'$ or $\rho'' \cdot \rho'$ (Lemma 2.31). After this alpha-conversion, k does not appear in either $\rho'' \cdot \rho'$, $\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}$, or $s_{n+1} + \{l_i \mapsto v_i\}$. By the induction hypotheses,

$$b^{\,s_{n+1}+\{l_i\mapsto v_i\}+\{k\mapsto u\}} \xRightarrow[\mathcal{F}'+\{f\mapsto\mathcal{F}\,f\}]{\rho''|\rho'} v^{\,s'+\{k\mapsto u\}}$$

Moreover, $s' + \{k \mapsto u\} \setminus \rho_T = s' \setminus \rho_T + \{k \mapsto u\}$ (since k does not appear in ρ_T). Hence

$$f(a_1 \dots a_n)^{\,s_1+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s'+\{k\mapsto u\}\setminus\rho_T}.$$

(val) $v^{\,s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v^{\,s+\{k\mapsto u\}\setminus\rho_T}$ and $s + \{k \mapsto u\} \setminus \rho_T = s \setminus \rho_T + \{k \mapsto u\}$ since k does not appear in ρ_T.

(var) $x^{\,s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} (s+\{k \mapsto u\})\, l^{\,s+\{k\mapsto u\}\setminus\rho_T}$, with $s+\{k \mapsto u\} \setminus \rho_T = s \setminus \rho_T + \{k \mapsto u\}$ since k does not appear in ρ_T, and $(s + \{k \mapsto u\})\ l = s\ l$ since $k \neq l$ (k does not appear in s).

(assign) By the induction hypotheses, $a^{\,s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} v^{\,s'+\{k\mapsto u\}}$. And $k \neq l$ (since k does not appear in s) then $s' + \{k \mapsto u\} + \{l \mapsto v\} = s' + \{l \mapsto v\} + \{k \mapsto u\}$. Moreover, k does not appear in ρ_T then $s' + \{l \mapsto v\} + \{k \mapsto u\} \setminus \rho_T = s' + \{l \mapsto v\} \setminus \rho_T + \{k \mapsto u\}$. Hence

$$x := a^{\,s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} \mathbf{1}^{\,s'+\{l\mapsto v\}\setminus\rho_T+\{k\mapsto u\}}$$

(seq) By the induction hypotheses,

$$a^{\,s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{|\rho_T\cdot\rho} \mathbf{true}^{\,s'+\{k\mapsto u\}}$$

$$b^{\,s'+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v'^{\,s''+\{k\mapsto u\}}$$

Hence

$$a\ ;\ b^{\,s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v'^{\,s''+\{k\mapsto u\}}$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) The location k does not appear in $\mathcal{F}'$, because it does not appear in either $\mathcal{F}$ or $\rho' \subset \rho_T \cdot \rho$ ($\mathcal{F}' = \mathcal{F} + \{f \mapsto [\lambda x_1 \dots x_n.a, \rho', \mathcal{F}]\}$). Then, by the induction hypotheses,

$$b\,^{s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}']{\rho_T|\rho} v\,^{s'+\{k\mapsto u\}}$$

Hence

$$\textbf{letrec } f(x_1 \dots x_n) = a \textbf{ in } b\,^{s+\{k\mapsto u\}} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\,^{s'+\{k\mapsto u\}}.$$ □

Lemma 2.33 (Spurious variable in environments)**.**

$$\forall l, l', M\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|\rho} v\,^{s'} \quad \textit{iff} \quad M\,^{s} \xRightarrow[\mathcal{F}]{\rho_T\cdot(x,l)|(x,l')\cdot\rho} v\,^{s'}$$

Moreover, both derivations have the same height.

Proof. See Lemma 2.14, p. 2.14. □

2.4.4 Aliasing lemmas

We need three lemmas to show that environments remain aliasing-free during the proof by induction in Section 2.4.5. The first lemma states that concatenating two environments in an aliasing-free set yields an aliasing-free set. The other two prove that the aliasing invariant (Invariant 6, Definition 2.25) holds in the context of the (call) and (letrec) rules, respectively.

Lemma 2.34 (Concatenation)**.** *If $\mathcal{E} \cup \{\rho, \rho'\}$ is aliasing-free then $\mathcal{E} \cup \{\rho \cdot \rho'\}$ is aliasing-free.*

Proof. By exhaustive check of cases. We want to prove

$$\forall \rho_1, \rho_2 \in \mathcal{E} \cup \{\rho \cdot \rho'\}, \forall x \in \mathrm{dom}(\rho_1), \forall y \in \mathrm{dom}(\rho_2),\ \rho_1\ x = \rho_2\ y \Rightarrow x = y.$$

given that

$$\forall \rho_1, \rho_2 \in \mathcal{E} \cup \{\rho, \rho'\}, \forall x \in \mathrm{dom}(\rho_1), \forall y \in \mathrm{dom}(\rho_2),\ \rho_1\ x = \rho_2\ y \Rightarrow x = y.$$

If $\rho_1 \in \mathcal{E}$ and $\rho_2 \in \mathcal{E}$, immediate. If $\rho_1 \in \{\rho \cdot \rho'\}$, $\rho_1\ x = \rho\ x$ or $\rho'\ x$. This is the same for ρ_2. Then $\rho_1\ x = \rho_2\ y$ is equivalent to $\rho\ x = \rho'\ y$ (or some other combination, depending on x, y, ρ_1 and ρ_2) which leads to the expected result. □

Lemma 2.35 (Aliasing in (call) rule)**.** *Assume that, in a (call) rule,*
- *$\mathcal{F}\ f = [\lambda x_1 \dots x_n.b, \rho', \mathcal{F}']$,*
- *$\mathrm{Env}(\mathcal{F})$ is aliasing-free, and*
- *$\rho'' = (x_1, l_1) \cdot \dots \cdot (x_n, l_n)$, with fresh and distinct locations l_i.*

Then $\mathrm{Env}(\mathcal{F}' + \{f \mapsto \mathcal{F}\ f\}) \cup \{\rho', \rho''\}$ is also aliasing-free.

Proof. Let $\mathcal{E} = \mathrm{Env}(\mathcal{F}' + \{f \mapsto \mathcal{F}\ f\}) \cup \{\rho'\}$. We know that $\mathcal{E} \subset \mathrm{Env}(\mathcal{F})$ so $\mathcal{E}$ is aliasing-free We want to show that adding fresh and distinct locations from ρ'' preserves this lack of freedom. More precisely, we want to show that

$$\forall \rho_1, \rho_2 \in \mathcal{E} \cup \{\rho''\}, \forall x \in \mathrm{dom}(\rho_1), \forall y \in \mathrm{dom}(\rho_2),\ \rho_1\ x = \rho_2\ y \Rightarrow x = y$$

given that

$$\forall \rho_1, \rho_2 \in \mathcal{E}, \forall x \in \mathrm{dom}(\rho_1), \forall y \in \mathrm{dom}(\rho_2),\ \rho_1\ x = \rho_2\ y \Rightarrow x = y.$$

We reason by checking of all cases. If $\rho_1 \in \mathcal{E}$ and $\rho_2 \in \mathcal{E}$, immediate. If $\rho_1 = \rho_2 = \rho''$ then $\rho''\ x = \rho''\ y \Rightarrow x = y$ holds because the locations of ρ'' are distinct. If $\rho_1 = \rho''$ and $\rho_2 \in \mathcal{E}$ then $\rho_1\ x = \rho_2\ y \Rightarrow x = y$ holds because $\rho_1\ x \neq \rho_2\ y$ (by freshness hypothesis). □

Lemma 2.36 (Aliasing in (letrec) rule)**.** *If* $\mathrm{Env}(\mathcal{F}) \cup \{\rho, \rho_T\}$ *is aliasing free, then, for all* x_i*,*

$$\mathrm{Env}(\mathcal{F}) \cup \{\rho, \rho_T\} \cup \{\rho_T \cdot \rho \mid_{\mathrm{dom}(\rho_T \cdot \rho) \setminus \{x_1 \dots x_n\}}\}$$

is aliasing free.

Proof. Let $\mathcal{E} = \mathrm{Env}(\mathcal{F}) \cup \{\rho, \rho_T\}$ and $\rho'' = \rho_T \cdot \rho|_{\mathrm{dom}(\rho_T \cdot \rho) \setminus \{x_1 \dots x_n\}}$. Adding ρ'', a restricted concatenation of ρ_T and ρ, to $\mathcal{E}$ preserves aliasing freedom, as in the proof of Lemma 2.34. If $\rho_1 \in \mathcal{E}$ and $\rho_2 \in \mathcal{E}$, immediate. If $\rho_1 \in \{\rho''\}$, $\rho_1\ x = \rho\ x$ or $\rho'\ x$. This is the same for ρ_2. Then $\rho_1\ x = \rho_2\ y$ is equivalent to $\rho\ x = \rho'\ y$ (or some other combination, depending on x, y, ρ_1 and ρ_2) which leads to the expected result. □

2.4.5 Proof of correctness

We finally show Theorem 2.28.

Theorem 2.28. *If* x *is a liftable parameter in* $(M, \mathcal{F}, \rho_T, \rho)$*, then*

$$M\ ^{s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\ ^{s'} \text{ implies } (M)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v\ ^{s'}$$

Assume that x is a liftable parameter in $(M, \mathcal{F}, \rho_T, \rho)$. The proof is by induction on the height of the reduction of $M\ ^{s} \xRightarrow[\mathcal{F}]{\rho_T|\rho} v\ ^{s'}$. To keep the proof readable, we detail only the non-trivial cases when checking the invariants of Definition 2.25 to ensure that the induction hypotheses hold.

(call) — first case First, we consider the most interesting case where there exists i such that $f = h_i$. The variable x is a liftable parameter in $(h_i(a_1 \dots a_n), \mathcal{F}, \rho_T, \rho)$ hence in $(a_i, \mathcal{F}, \varepsilon, \rho_T \cdot \rho)$ too.

Indeed, the invariants of Definition 2.25 hold:

- Invariant 3: By definition of a local position, every f defined in local position in a_i is in local position in $h_i(a_1 \dots a_n)$, hence the expected property by the induction hypotheses.
- Invariant 4: Immediate since the premise does not hold : since the a_i are not in tail position in $h_i(a_1 \dots a_n)$, they cannot feature calls to h_i (by Invariant 2).
- Invariant 6: Lemma 2.34, p. 26.

The other invariants hold trivially.

By the induction hypotheses, we get

$$(a_i)_*\ ^{s_i} \xRightarrow[(\mathcal{F})_*]{|\rho_T \cdot \rho} v_i^{s_{i+1}}.$$

By definition of lifting, $(h_i(a_1 \dots a_n))_* = h_i((a_1)_*, \dots, (a_n)_*, x)$. But x is not a liftable parameter in $(b, \mathcal{F}', \rho'', \rho')$ since the Invariant 4 might be broken: $x \notin \mathrm{dom}(\rho'')$ (x is not a parameter of h_i) but h_j might appear in tail position in b.

On the other hand, we have $x \in \mathrm{dom}(\rho')$: since, by hypothesis, x is a liftable parameter in $(h_i(a_1 \dots a_n), \mathcal{F}, \rho_T, \rho)$, it appears necessarily in the environments of the closures of the h_i, such as ρ'. This allows us to split ρ' into two parts: $\rho' = (x, l) \cdot \rho'''$. It is then possible to move (x, l) to the tail environment, according to Lemma 2.30:

$$b\ ^{s_{n+1} + \{l_i \mapsto v_i\}} \xRightarrow[\mathcal{F}' + \{f \mapsto \mathcal{F}\ f\}]{\rho''(x,l)|\rho'''} v\ ^{s'|_{\mathrm{dom}(s') \setminus \{l\}}}$$

This rewriting ensures that x is a liftable parameter in $(b, \mathcal{F}' + \{f \mapsto \mathcal{F}\ f\}, \rho'' \cdot (x, l), \rho''')$.

Indeed, the invariants of Definition 2.25 hold:

– Invariant 3: Every function defined in local position in b is an inner function in h_i so, by Invariant 2, it is one of the h_i and $x \in \mathrm{dom}(\rho'' \cdot (x,l) \cdot \rho''')$.
– Invariant 4: Immediate since $x \in \mathrm{dom}(\rho'' \cdot (x,l) \cdot \rho''')$.
– Invariant 5: Immediate since $\mathcal{F}'$ is included in $\mathcal{F}$.
– Invariant 6: Immediate for the compact closures. Aliasing freedom is guaranteed by Lemma 2.35 (p. 26).

The other invariants hold trivially.

By the induction hypotheses,

$$(b)_*^{s_{n+1}+\{l_i \mapsto v_i\}} \xRightarrow[(\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\})_*]{\rho''(x,l)|\rho'''} v\ {s'}|_{\mathrm{dom}(s')\setminus\{l\}}$$

The l location is not fresh: it must be rewritten into a fresh location, since x is now a parameter of h_i. Let l' be a location appearing in neither $(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*$, nor $s_{n+1} + \{l_i \mapsto v_i\}$ or $\rho'' \cdot {\rho_T}'$. Then l' is a fresh location, which is to act as l in the reduction of $(b)_*$.

We will show that, after the reduction, l' is not in the store (just like l before the lambda-lifting). In the meantime, the value associated to l does not change (since l' is modified instead of l).

Lemma 2.27 implies that x does not appear in the environments of $(\mathcal{F})_*$, so it does not appear in the environments of $(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_* \subset (\mathcal{F})_*$ either. As a consequence, lack of aliasing implies by Definition 2.23 that the label l, associated to x, does not appear in $(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*$ either, so

$$(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*[l'/l] = (\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*\,.$$

Moreover, l does not appear in ${s'}|_{\mathrm{dom}(s')\setminus\{l\}}$. By alpha-conversion (Lemma 2.31, since l' does not appear in the store or the environments of the reduction, we rename l to l':

$$(b)_*^{s_{n+1}[l'/l]+\{l_i \mapsto v_i\}} \xRightarrow[(\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\})_*]{\rho''(x,l')|\rho'''} v\ {s'}|_{\mathrm{dom}(s')\setminus\{l\}}\,.$$

We want now to reintroduce l. Let $v_x = s_{n+1}\ l$. The location l does not appear in $s_{n+1}[l'/l] + \{l_i \mapsto v_i\}$, $(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*$, or $\rho''(x,l') \cdot \rho'''$. Thus, by Lemma 2.32,

$$(b)_*^{s_{n+1}[l'/l]+\{l_i \mapsto v_i\}+\{l \mapsto v_x\}} \xRightarrow[(\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\})_*]{\rho''(x,l')|\rho'''} v\ {s'}|_{\mathrm{dom}(s')\setminus\{l\}}+\{l \mapsto v_x\}\,.$$

Since

$$\begin{aligned} s_{n+1}[l'/l] + \{l_i \mapsto v_i\} + \{l \mapsto v_x\} &= s_{n+1}[l'/l] + \{l \mapsto v_x\} + \{l_i \mapsto v_i\} && \text{because } \forall i, l \neq l_i \\ &= s_{n+1} + \{l' \mapsto v_x\} + \{l_i \mapsto v_i\} && \text{because } v_x = s_{n+1} l \\ &= s_{n+1} + \{l_i \mapsto v_i\} + \{l' \mapsto v_x\} && \text{because } \forall i, l' \neq l_i \end{aligned}$$

and ${s'}|_{\mathrm{dom}(s')\setminus\{l\}} + \{l \mapsto v_x\} = s' + \{l \mapsto v_x\}$, we finish the rewriting by Lemma 2.33,

$$(b)_*^{s_{n+1}+\{l_i \mapsto v_i\}+\{l' \mapsto v_x\}} \xRightarrow[(\mathcal{F}'+\{f \mapsto \mathcal{F}\, f\})_*]{\rho''(x,l')|(x,l)\cdot\rho'''} v\ \ {s'}+\{l \mapsto v_x\}\,.$$

Hence the result:

$$\frac{\begin{array}{c}(\mathcal{F})_* \;\; h_i = [\lambda x_1 \dots x_n x.\, (b)_*, \rho', (\mathcal{F}')_*] \\ \rho'' = (x_1, l_1) \cdot \dots \cdot (x_n, l_n)(x, \rho_T\ x) \qquad l' \text{ and } l_i \text{ fresh and distinct} \\ \forall i, (a_i)_*^{\,s_i} \xRightarrow[(\mathcal{F})_*]{|\rho_T \cdot \rho} v_i^{s_{i+1}} \\ (x)_*^{\,s_{n+1}} \xRightarrow[(\mathcal{F})_*]{|\rho_T \cdot \rho} v_x^{s_{n+1}} \qquad (b)_*^{s_{n+1} + \{l_i \mapsto v_i\} + \{l' \mapsto v_x\}} \xRightarrow[(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*]{\rho''(x,l')|\rho'} v^{\;s' + \{l \mapsto v_x\}}\end{array}}{(h_i(a_1 \dots a_n))_*^{\,s_1} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v^{\,s' + \{l \mapsto v_x\} \setminus \rho_T}}\ (\textsc{call})$$

Since $l \in \mathrm{dom}(\rho_T)$ (because x is a liftable parameter in $(h_i(a_1 \dots a_n), \mathcal{F}, \rho_T, \rho)$), the extraneous location is reclaimed as expected: $s' + \{l \mapsto v_x\} \setminus \rho_T = s' \setminus \rho_T$.

(call) — second case We now consider the case where f is not one of the h_i. The variable x is a liftable parameter in $(f(a_1 \dots a_n), \mathcal{F}, \rho_T, \rho)$ hence in $(a_i, \mathcal{F}, \varepsilon, \rho_T \cdot \rho)$ too.

Indeed, the invariants of Definition 2.25 hold:

- Invariant 3: By definition of a local position, every f defined in local position in a_i is in local position in $f(a_1 \dots a_n)$, hence the expected property by the induction hypotheses.
- Invariant 4: Immediate since the premise does not hold : the a_i are not in tail position in $f(a_1 \dots a_n)$ so they cannot feature calls to h_i (by Invariant 2:).
- Invariant 6: Lemma 2.34, p. 26.

The other invariants hold trivially.

By the induction hypotheses, we get

$$(a_i)_*^{\,s_i} \xRightarrow[(\mathcal{F})_*]{|\rho_T \cdot \rho} v_i^{s_{i+1}},$$

and, by Definition 2.6,

$$(f(a_1 \dots a_n))_* = f((a_1)_*, \dots, (a_n)_*).$$

If x is not defined in b or $\mathcal{F}$, then $()_*$ is the identity function and can trivially be applied to the reduction of b. Otherwise, x is a liftable parameter in $(b, \mathcal{F}' + \{f \mapsto \mathcal{F}\, f\}, \rho'', \rho')$.

Indeed, the invariants of Definition 2.25 hold. Assume that x is defined as a parameter of some function g, in either b or $\mathcal{F}$:

- Invariant 3: We have to distinguish the cases where $f = g$ (with $x \in \mathrm{dom}(\rho'')$) and $f \neq g$ (with $x \notin \mathrm{dom}(\rho'')$ and $x \notin \mathrm{dom}(\rho')$). In both cases, the result is immediate by the induction hypotheses.
- Invariant 4: If $f \neq g$, the premise cannot hold (by the induction hypotheses, Invariant 2). If $f = g$, $x \in \mathrm{dom}(\rho'')$ (by the induction hypotheses, Invariant 2).
- Invariant 5: Immediate since $\mathcal{F}'$ is included in $\mathcal{F}$.
- Invariant 6: Immediate for the compact closures. Aliasing freedom is guaranteed by Lemma 2.35 (p. 26).

The other invariants hold trivially.

By the induction hypotheses,

$$(b)_*^{s_{n+1} + \{l_i \mapsto v_i\}} \xRightarrow[(\mathcal{F}' + \{f \mapsto \mathcal{F}\, f\})_*]{\rho''|\rho'} v^{\,s'}$$

hence:

$$\text{(CALL)}\ \frac{(\mathcal{F})_*\ f = [\lambda x_1 \ldots x_n.\,(b)_*, \rho', (\mathcal{F}')_*] \qquad \rho'' = (x_1, l_1) \cdot \ldots \cdot (x_n, l_n) \qquad l_i \text{ fresh and distinct}}{(f(a_1 \ldots a_n))_*\ ^{s_1} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v^{\,s'\setminus\rho_T}}$$

$$\frac{\forall i, (a_i)_*\ ^{s_i} \xRightarrow[(\mathcal{F})_*]{|\rho_T\cdot\rho} v_i^{s_{i+1}} \qquad (b)_*^{s_{n+1}+\{l_i\mapsto v_i\}} \xRightarrow[(\mathcal{F}'+\{f\mapsto \mathcal{F}\,f\})_*]{\rho''|\rho'} v^{\,s'}}{}$$

(letrec) The parameter x is a liftable in $(\textbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \textbf{in}\ \ b, \mathcal{F}, \rho_T, \rho)$ so x is a liftable parameter in $(b, \mathcal{F}', \rho_T, \rho)$ too.

Indeed, the invariants of Definition 2.25 hold:

- Invariants 3 and 4: Immediate by the induction hypotheses and definition of tail and local positions.
- Invariant 5: By the induction hypotheses, Invariant 3 (x is to appear in the new closure if and only if $f = h_i$).
- Invariant 6: Lemma 2.36 (p. 27).

The other invariants hold trivially.

By the induction hypotheses, we get

$$(b)_*^{\,s} \xRightarrow[(\mathcal{F}')_*]{\rho_T|\rho} v^{\,s'}.$$

If $f \neq h_i$,

$$(\textbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \textbf{in}\ \ b)_* = \textbf{letrec}\ \ f(x_1 \ldots x_n) = (a)_*\ \ \textbf{in}\ \ (b)_*$$

hence, by definition of $(\mathcal{F}')_*$,

$$\text{(LETREC)}\ \frac{(b)_*^{\,s} \xRightarrow[(\mathcal{F}')_*]{\rho_T|\rho} v^{\,s'} \qquad \rho' = \rho_T \cdot \rho|_{\mathrm{dom}(\rho_T\cdot\rho)\setminus\{x_1\ldots x_n\}} \qquad (\mathcal{F}')_* = (\mathcal{F})_* + \{f \mapsto [\lambda x_1 \ldots x_n.\,(a)_*, \rho', F]\}}{(\textbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \textbf{in}\ \ b)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v^{\,s'}}$$

On the other hand, if $f = h_i$,

$$(\textbf{letrec}\ \ f(x_1 \ldots x_n) = a\ \ \textbf{in}\ \ b)_* = \textbf{letrec}\ \ f(x_1 \ldots x_n x) = (a)_*\ \ \textbf{in}\ \ (b)_*$$

hence, by definition of $(\mathcal{F}')_*$,

$$\text{(LETREC)}\ \frac{(b)_*^{\,s} \xRightarrow[(\mathcal{F}')_*]{\rho_T|\rho} v^{\,s'} \qquad \rho' = \rho_T \cdot \rho|_{\mathrm{dom}(\rho_T\cdot\rho)\setminus\{x_1\ldots x_n x\}} \qquad (\mathcal{F}')_* = (\mathcal{F})_* + \{h_i \mapsto [\lambda x_1 \ldots x_n x.\,(a)_*, \rho', F]\}}{(\textbf{letrec}\ \ h_i(x_1 \ldots x_n) = a\ \ \textbf{in}\ \ b)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v^{\,s'}}$$

(val) $(v)_* = v$ so

$$\text{(VAL)}\ \frac{}{(v)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v^{\,s\setminus\rho_T}}$$

(var) $(y)_* = y$ so

$$(\textsc{var})\ \frac{\rho_T \cdot \rho\ y = l \in \mathrm{dom}\ \ s}{(y)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} s\ l\ ^{s\backslash\rho_T}}$$

(assign) The parameter x is liftable in $(y \coloneqq a, \mathcal{F}, \rho_T, \rho)$ so in $(a, \mathcal{F}, \varepsilon, \rho_T \cdot \rho)$ too.

Indeed, the invariants of Definition 2.25 hold:

– Invariant 6: Lemma 2.34, p. 26.

The other invariants hold trivially.

By the induction hypotheses, we get

$$(a)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{|\rho_T\cdot\rho} v\ ^{s'}.$$

Moreover

$$(y \coloneqq a)_* = y \coloneqq (a)_*,$$

so :

$$(\textsc{assign})\ \frac{(a)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{|\rho_T\cdot\rho} v\ ^{s'} \qquad \rho_T \cdot \rho\ y = l \in \mathrm{dom}\ \ s'}{(y \coloneqq a)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} \mathbf{1}\ ^{s'+\{l\mapsto v\}\backslash\rho_T}}$$

(seq) The parameter x is liftable in $(a\ ;\ b, \mathcal{F}, \rho_T, \rho)$. If x is not defined in a or $\mathcal{F}$, then $()_*$ is the identity function and can trivially be applied to the reduction of a. Otherwise, x is a liftable parameter in $(a, \mathcal{F}, \varepsilon, \rho_T \cdot \rho)$.

Indeed, the invariants of Definition 2.25 hold:

– Invariant 6: Lemma 2.34, p. 26.

The other invariants hold trivially.

If x is not defined in b or $\mathcal{F}$, then $()_*$ is the identity function and can trivially be applied to the reduction of b. Otherwise, x is a liftable parameter in $(b, \mathcal{F}, \rho_T, \rho)$. Indeed, the invariants of Definition 2.25 hold trivially.

By the induction hypotheses, we get $(a)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{|\rho_T\cdot\rho} v\ ^{s'}$ and $(b)_*\ ^{s'} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v'\ ^{s''}$.

Moreover,

$$(a\ ;\ b)_* = (a)_*\ ;\ (b)_*,$$

hence:

$$(\textsc{seq})\ \frac{(a)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{|\rho_T\cdot\rho} v\ ^{s'} \qquad (b)_*\ ^{s'} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v'\ ^{s''}}{(a\ ;\ b)_*\ ^{s} \xRightarrow[(\mathcal{F})_*]{\rho_T|\rho} v'\ ^{s''}}$$

(if-true) and (if-false) are proved similarly to (seq).

3 CPS conversion

In this section, we prove the correctness of the CPS-conversion performed by the CPC translator. This conversion is defined only on a subset of C programs that we call *CPS-convertible terms* (Section 3.1). We first show that the *early evaluation* of function parameters in CPS-convertible terms is correct (Section 3.2). To simplify the proof of correctness of CPS-conversion, we then introduce small-step reduction rules featuring contexts and early evaluation (Section 3.3).

In Section 3.4, we define *CPS terms*, with the `push` and `invoke` operators to build and execute continuations, and the associated reduction rules. Since the syntax of CPS-terms does not ensure a correct reduction, we also define *well-formed* CPS-terms, which are the image of CPS-convertible terms by CPS-conversion.

The proof of correctness of CPS-conversion is finally carried out in Section 3.5. It consists merely in checking that the reduction rules for CPS-convertible terms and well-formed CPS-terms execute in lock-step.

3.1 CPS-convertible form

CPS conversion is not defined for every C function; instead, we restrict ourselves to a subset of functions, which we call the *CPS-convertible* subset. The CPS-convertible form restricts the calls to cps functions to make it straightforward to capture their continuation. In CPS-convertible form, a call to a cps function `f` is either in tail position, or followed by a tail call to another cps function whose parameters are *non-shared* variables that cannot be modified by `f`.

In the C language, we define the CPS-convertible form as follows:

Definition 3.1 (CPS-convertible form)**.** *A function* `h` *is in* CPS-convertible form *if every call to a cps function that it contains matches one of the following patterns, where both* `f` *and* `g` *are cps functions,* $e_1, \ldots, e_n$ *are any C expressions and* $x, y_1, \ldots, y_n$ *are distinct, non-shared variables:*

```
                                  return f(e1, ..., en);                (1)
          x = f(e1, ..., en); return g(x, y1, ..., yn);                 (2)
              f(e1, ..., en); return g(x, y1, ..., yn);                 (3)
                                f(e1, ..., en); return;                 (4)
             f(e1, ..., en); g(x, y1, ..., yn); return;                 (5)
        x = f(e1, ..., en); g(x, y1, ..., yn); return;                  (6)
```

Note the use of `return` to explicitly mark calls in tail position. The forms (3) to (6) are only necessary to handle the cases where `f` and `g` return `void`; in the rest of the proof, we ignore these cases that are a syntactical detail of the C language, and focus on the essential cases (1) and (2).

To prove the correctness of CPS-conversion, we need to express this definition in our small imperative language. This is done by defining CPS-convertible terms, which are a subset of the terms introduced in Definition 2.1 (Section 2.1). A program in CPS-convertible form consists of a set of mutually-recursive functions with no free variables, the body of each of which is a CPS-convertible term.

A CPS-convertible term has two parts: the head and the tail. The head is a (possibly empty) sequence of assignments, possibly embedded within conditional statements. The tail is a (possibly empty) sequence of function calls in a highly restricted form: their parameters are (side-effect free) expressions, except possibly for the last one, which can be another function call of the same form. Values and expressions are left unchanged.

Definition 3.2 (CPS-convertible terms)**.**

$$
\begin{aligned}
v &::= \quad \mathbf{1} \mid \mathbf{true} \mid \mathbf{false} \mid n \in \mathbf{N} && \text{(values)}\\
expr &::= \quad v \mid x \mid \ldots && \text{(expressions)}\\
F &::= f(expr, \ldots, expr) \mid f(expr, \ldots, expr, F) && \text{(nested function calls)}\\
Q &::= \epsilon \mid Q \ ; \ F && \text{(tail)}\\
T &::= expr \mid x \coloneqq expr \ ; \ T \mid \mathbf{if} \ e \ \mathbf{then} \ T \ \mathbf{else} \ T \mid Q && \text{(head)}
\end{aligned}
$$

The essential property of CPS-convertible terms, which makes their CPS conversion immediate to perform, is the guarantee that there is no cps call outside of the tails. It makes continuations easy to represent as a series of function calls (tails) and separates them clearly from imperative blocks (heads), which are not modified by the CPC translator.

The tails are a generalisation of Definition 3.1, which will be useful for the proof of correctness of CPS-conversion. Note that `x = f(e1, ..., en); return g(x, y1, ..., yn)` is represented by $g(f(e_1 \ldots e_n), y_1 \ldots y_n)$: this translation is correct because, contrary to C, our language guarantees a left-to-right evaluation of function parameters.

Also noteworthy are the facts that:

- there is no letrec construct anymore since every function is defined at top-level,
- assignments, conditions and function parameters of f are restricted to expressions, to ensure that function calls only appear in tail position,
- there is no need to forbid shared variables in the parameters of g because they are ruled out of our language by design.

3.2 Early evaluation

In this section, we prove that correctness of *early evaluation*, ie. evaluating the expressions $expr$ before F when reducing $f(expr, \ldots, expr, F)$ in a tail. This result is necessary to show the correctness of the CPS-conversion, because function parameters are evaluated before any function call when building continuations.

The reduction rules may be simplified somewhat for CPS-convertible terms. We do not need to keep an explicit environment of functions since there are no inner functions any more; for the same reason, the (letrec) rule disappears. Instead, we use a constant environment $\mathcal{F}$ holding every function used in the reduced term M. To account for the absence of free variables, the closures in $\mathcal{F}$ need not carry an environment. As a result, in the (call) rule, $\rho' = \varepsilon$ and $\mathcal{F}' = \mathcal{F}$.

Early evaluation is correct for lifted terms because a lifted term can never modify the variables that are not in its environment, since it cannot access them through closures.

Lemma 3.3. *Let M be a lambda-lifted term. Then,*

$$M^{\,s} \xrightarrow[\mathcal{F}]{\rho} v^{\,s'}$$

implies

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s'|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$

Proof. By induction on the structure of the reduction. The key points are the use of $\rho' = \varepsilon$ in the (call) case, and the absence of (letrec) rules.

(val) and (var) Trivial ($s = s'$).

(assign) By the induction hypotheses,

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s'|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} \text{ and } l \in \mathrm{Im}(\rho),$$

hence

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = (s' + \{l \mapsto v\})|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$

(seq) By the induction hypotheses,

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s'|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} \text{ and } s'|_{\mathrm{dom}(s')\setminus\mathrm{Im}(\rho)} = s''|_{\mathrm{dom}(s')\setminus\mathrm{Im}(\rho)}.$$

Since, $\mathrm{dom}(s) \subset \mathrm{dom}(s')$, the second equality can be restricted to

$$s'|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s''|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$

Hence,

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s''|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$

(if-true) and (if-false) are proved similarly to (seq).

(letrec) doesn't occur since M is lambda-lifted.

(call) By the induction hypotheses,

$$(s_{n+1} + \{l_i \mapsto v_i\})|_{\mathrm{dom}(s_{n+1}+\{l_i\mapsto v_i\})\setminus\mathrm{Im}(\rho''\cdot\rho')} = s'|_{\mathrm{dom}(s_{n+1}+\{l_i\mapsto v_i\})\setminus\mathrm{Im}(\rho''\cdot\rho')}$$

Since $\rho' = \varepsilon$, $\mathrm{Im}(\rho'') = \{l_i\}$ and $\mathrm{dom}(s_{n+1}) \cap \{l_i\} = \emptyset$ (by freshness),

$$(s_{n+1} + \{l_i \mapsto v_i\})|_{\mathrm{dom}(s_{n+1})} = s'|_{\mathrm{dom}(s_{n+1})}$$

so $s_{n+1} = s'|_{\mathrm{dom}(s_{n+1})}$.
Since $\mathrm{dom}(s) \setminus \mathrm{Im}(\rho) \subset \mathrm{dom}(s) \subset \mathrm{dom}(s_{n+1})$,

$$s_{n+1}|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s'|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$

Finally, we can prove similarly to the (seq) case that

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s_{n+1}|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$

Hence,

$$s|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)} = s'|_{\mathrm{dom}(s)\setminus\mathrm{Im}(\rho)}.$$ □

As a consequence, a tail of function calls cannot modify the current store, only extend it with the parameters of the called functions.

Corollary 3.4. *For every tail Q,*

$$Q\,{}^{s} \xrightarrow[\mathcal{F}]{\rho} v\,{}^{s'} \text{ implies } s \sqsubseteq s'.$$

Proof. We prove the corollary by induction on the structure of a tail. First remember that *store extension* (written $\sqsubseteq$) is a partial order over stores (Property 2.18), defined in Section 2.3.2 as follows: $s \sqsubseteq s' \quad \text{iff} \quad s'|_{\mathrm{dom}(s)} = s$.

The case ϵ is trivial. The case $Q\ ;\ F$ is immediate by induction ((seq) rule), since $\sqsubseteq$ is transitive. Similarly, it is pretty clear that $f(expr, \ldots, expr, F)$ follows by induction and transitivity from $f(expr, \ldots, expr)$ ((call) rule). We focus on this last case.

Lemma 3.3 implies:

$$(s_{n+1} + \{l_i \mapsto v_i\})|_{\mathrm{dom}(s_{n+1}+\{l_i \mapsto v_i\}) \setminus \mathrm{Im}(\rho'' \cdot \rho')} = s'|_{\mathrm{dom}(s_{n+1}+\{l_i \mapsto v_i\}) \setminus \mathrm{Im}(\rho'' \cdot \rho')}.$$

Since $\rho' = \varepsilon$, $\mathrm{Im}(\rho'') = \{l_i\}$ and $\mathrm{dom}(s_{n+1}) \cap \{l_i\} = \emptyset$ (by freshness),

$$(s_{n+1} + \{l_i \mapsto v_i\})|_{\mathrm{dom}(s_{n+1})} = s'|_{\mathrm{dom}(s_{n+1})}$$

so $s_{n+1} = s'|_{\mathrm{dom}(s_{n+1})}$.

The evaluation of *expr* parameters do not change the store: $s_{n+1} = s$. The expected result follows: $s = s'|_{\mathrm{dom}(s)}$, hence $s \sqsubseteq s'$. □

This leads to the correctness of early evaluation.

Theorem 3.5 (Early evaluation)**.** *For every tail* Q, $Q\,^{s} \xrightarrow[\mathcal{F}]{\rho} v\,^{s'}$ *implies* $Q[x \backslash s(\rho\ x)]\,^{s} \xrightarrow[\mathcal{F}]{\rho} v\,^{s'}$ *(provided* $x \in \mathrm{dom}(\rho)$ *and* $\rho\ x \in \mathrm{dom}(s)$*).*

Proof. Immediate induction on the structure of tails and expressions: Corollary 3.4 implies that $s \sqsubseteq s''$ and $\rho\ x \in \mathrm{dom}(s)$ ensures that $s(\rho\ x) = s''(\rho\ x)$ in the relevant cases (namely the (seq) rule for $Q\ ;\ F$ and the (call) rule for $f(expr, \ldots, expr, F)$). □

3.3 Small-step reduction

We define the semantics of CPS-convertible terms through a set of small-step reduction rules. We distinguish three kinds of reductions: $\rightarrow_T$ to reduce the head of terms, $\rightarrow_Q$ to reduce the tail, and $\rightarrow_e$ to evaluate expressions.

These rules describe a stack machine with a store σ to keep the value of variables. Since free and shared variables have been eliminated in earlier passes, there is a direct correspondence at any point in the program between variable names and locations, with no need to dynamically maintain an extra environment.

We use contexts as a compact representation for stacks. The head rules $\rightarrow_T$ reduce triples made of a term, a context and a store: $\langle T, C[\,], \sigma \rangle$. The tail rules $\rightarrow_Q$, which merely unfold tails with no need of a store, reduce couples of a tail and a context: $\langle Q, C[\,], \rangle$. The expression rules do not need context to reduce, thus operating on couples made of an expression and a store: $\langle e, \sigma \rangle$.

Contexts Contexts are sequences of function calls. In those sequences, function parameters shall be already evaluated: constant expressions are allowed, but not variables. As a special case, the last parameter might be a "hole" instead, written $\ominus$, to be filled with the return value of the next, nested function.

Definition 3.6 (Contexts)**.** *Contexts are defined inductively:*

$$C ::= [\,] \mid C[[\,]\ ;\ f(v, \ldots, v)] \mid C[[\,]\ ;\ f(v, \ldots, v, \ominus)]$$

Definition 3.7 (CPS-convertible reduction rules)**.**

$$\langle x \coloneqq expr\ ;\ T, C[\ ], \sigma\rangle \to_T \langle T, C[\ ], \sigma[x \mapsto v]\rangle \quad (7)$$
$$\textit{when } \langle expr, \sigma\rangle \to_e^\star v$$
$$\langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, C[\ ], \sigma\rangle \to_T \langle T_1, C[\ ], \sigma\rangle \quad (8)$$
$$\textit{when } \langle expr, \sigma\rangle \to_e^\star \textbf{true}$$
$$\langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, C[\ ], \sigma\rangle \to_T \langle T_2, C[\ ], \sigma\rangle \quad (9)$$
$$\textit{when } \langle expr, \sigma\rangle \to_e^\star \textbf{false}$$
$$\langle expr, C[[\ ]\ ;\ f(v_1, \ldots, v_n)], \sigma\rangle \to_T \langle \epsilon, C[[\ ]\ ;\ f(v_1, \ldots, v_n)]\rangle \quad (10)$$
$$\langle expr, C[[\ ]\ ;\ f(v_1, \ldots, v_n, \ominus)], \sigma\rangle \to_T \langle \epsilon, C[[\ ]\ ;\ f(v_1, \ldots, v_n, v)]\rangle \quad (11)$$
$$\textit{when } \langle expr, \sigma\rangle \to_e^\star v$$
$$\langle expr, [\ ], \sigma\rangle \to_T v \quad \textit{when } \langle expr, \sigma\rangle \to_e^\star v$$
$$\langle Q, C[\ ], \sigma\rangle \to_T \langle Q[x_i \setminus \sigma\ x_i], C[\ ]\rangle \quad (12)$$
$$\textit{for every } x_i \textit{ in } \mathrm{dom}(\sigma)$$

$$\langle Q\ ;\ f(v_1, \ldots, v_n), C[\ ]\rangle \to_Q \langle Q, C[[\ ]\ ;\ f(v_1, \ldots, v_n)]\rangle \quad (13)$$
$$\langle Q\ ;\ f(v_1, \ldots, v_n, F), C[\ ]\rangle \to_Q \langle Q\ ;\ F, C[[\ ]\ ;\ f(v_1, \ldots, v_n, \ominus)]\rangle \quad (14)$$
$$\langle \epsilon, C[[\ ]\ ;\ f(v_1, \ldots, v_n)]\rangle \to_Q \langle T, C[\ ], \sigma\rangle \quad (15)$$
$$\textit{when } f(x_1, \ldots, x_n) = T \textit{ and } \sigma = \{x_i \mapsto v_i\}$$

We do not detail the rules for $\to_e$, which simply looks for variables in σ and evaluates arithmetical and boolean operators.

Early evaluation Note that Rule 12 evaluates every function parameter in a tail before the evaluation of the tail itself. This is precisely the early evaluation process described above, which is correct by Theorem 3.5. We introduce early evaluation directly in the reduction rules rather than using it as a lemma to simplify the proof of correctess of the CPS-conversion.

3.4 CPS terms

Unlike classical CPS conversion techniques [5], our CPS terms are not continuations, but a procedure which builds and executes the continuation of a term. Construction is performed by **push**, which adds a function to the current continuation, and execution by **invoke**, which calls the first function of the continuation, optionally passing it the return value of the current function.

Definition 3.8 (CPS terms)**.**

$$v ::= \mathbf{1} \mid \textbf{true} \mid \textbf{false} \mid n \in \mathbf{N} \quad \text{(values)}$$
$$expr ::= v \mid x \mid \ldots \quad \text{(expressions)}$$
$$Q ::= \textbf{invoke} \mid \textbf{push } f(expr, \ldots, expr)\ ;\ Q \mid \textbf{push } f(expr, \ldots, expr, \boxdot)\ ;\ Q \quad \text{(tail)}$$
$$T ::= \textbf{invoke } expr \mid x \coloneqq expr\ ;\ T \mid \textbf{if } e \textbf{ then } T \textbf{ else } T \mid Q \quad \text{(head)}$$

Continuations and reduction rules A continuation is a sequence of function calls to be performed, with already evaluated parameters. We write $\cdot$ for appending a function to a continuation, and $\boxdot$ for a "hole", i.e. an unknown parameter.

Definition 3.9 (Continuations)**.**

$$\mathcal{C} ::= \varepsilon \mid \ f(v, \ldots, v) \cdot \mathcal{C} \mid \ f(v, \ldots, v, \boxdot) \cdot \mathcal{C}$$

The reduction rules for CPS terms are isomorphic to the rules for CPS-convertible terms, except that they use continuations instead of contexts.

Definition 3.10 (CPS reduction rules)**.**

$$\langle x \coloneqq expr \ ; \ T, \mathcal{C}, \sigma\rangle \to_T \langle T, \mathcal{C}, \sigma[x \mapsto v]\rangle \quad (16)$$
$$\textit{when } \langle expr, \sigma\rangle \to_e^\star v$$
$$\langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, \mathcal{C}, \sigma\rangle \to_T \langle T_1, \mathcal{C}, \sigma\rangle \quad (17)$$
$$\textit{if } \langle expr, \sigma\rangle \to_e^\star \textbf{true}$$
$$\langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, \mathcal{C}, \sigma\rangle \to_T \langle T_2, \mathcal{C}, \sigma\rangle \quad (18)$$
$$\textit{if } \langle expr, \sigma\rangle \to_e^\star \textbf{false}$$
$$\langle \textbf{invoke } expr, f(v_1, \ldots, v_n) \cdot \mathcal{C}, \sigma\rangle \to_T \langle \textbf{invoke}, f(v_1, \ldots, v_n) \cdot \mathcal{C}\rangle \quad (19)$$
$$\langle \textbf{invoke } expr, f(v_1, \ldots, v_n, \boxdot) \cdot \mathcal{C}, \sigma\rangle \to_T \langle \textbf{invoke}, f(v_1, \ldots, v_n, v) \cdot \mathcal{C}\rangle \quad (20)$$
$$\textit{when } \langle expr, \sigma\rangle \to_e^\star v$$
$$\langle \textbf{invoke } expr, \varepsilon, \sigma\rangle \to_T v \quad \textit{when } \langle expr, \sigma\rangle \to_e^\star v$$
$$\langle Q, \mathcal{C}, \sigma\rangle \to_T \langle Q[x_i \setminus \sigma \ x_i], \mathcal{C}\rangle \quad (21)$$
$$\textit{for every } x_i \textit{ in } \mathrm{dom}(\sigma)$$

$$\langle \textbf{push } \ f(v_1, \ldots, v_n) \ ; \ Q, \mathcal{C}\rangle \to_Q \langle Q, f(v_1, \ldots, v_n) \cdot \mathcal{C}\rangle \quad (22)$$
$$\langle \textbf{push } \ f(v_1, \ldots, v_n, \boxdot) \ ; \ Q, \mathcal{C}\rangle \to_Q \langle Q, f(v_1, \ldots, v_n, \boxdot) \cdot \mathcal{C}\rangle \quad (23)$$
$$\langle \textbf{invoke}, f(v_1, \ldots, v_n) \cdot \mathcal{C}\rangle \to_Q \langle T, \mathcal{C}, \sigma\rangle \quad (24)$$
$$\textit{when } f(x_1, \ldots, x_n) = T \textit{ and } \sigma = \{x_i \mapsto v_i\}$$

Well-formed terms Not all CPS term will lead to a correct reduction. If we **push** a function expecting the result of another function and **invoke** it immediately, the reduction blocks:

$$\langle \textbf{push } \ f(v_1, \ldots, v_n, \boxdot) \ ; \ \textbf{invoke}, \mathcal{C}, \sigma\rangle \to \langle \textbf{invoke}, f(v_1, \ldots, v_n, \boxdot) \cdot \mathcal{C}, \sigma\rangle \not\to$$

Well-formed terms avoid this behaviour.

Definition 3.11 (Well-formed term)**.** *A continuation queue is* well-formed *if it does not end with:*

$$\textbf{push } \ f(expr, \ldots, expr, \boxdot) \ ; \ \textbf{invoke}\,.$$

A term is well-formed *if every continuation queue in this term is well-formed.*

3.5 Correctess of the CPS-conversion

We define the CPS conversion as a mapping from CPS-convertible terms to CPS terms.

Definition 3.12 (CPS conversion)**.**

$$
\begin{aligned}
(Q\ ;\ f(expr,\ldots,expr))^{\blacktriangle} &= \textbf{push}\ \ f(expr,\ldots,expr)\ ;\ Q^{\blacktriangle} \\
(Q\ ;\ f(expr,\ldots,expr,F))^{\blacktriangle} &= \textbf{push}\ \ f(expr,\ldots,expr,\boxdot)\ ;\ (Q\ ;\ F)^{\blacktriangle} \\
\epsilon^{\blacktriangle} &= \textbf{invoke} \\
(x \coloneqq expr\ ;\ T)^{\blacktriangle} &= x \coloneqq expr\ ;\ T^{\blacktriangle} \\
(\textbf{if}\ \ expr\ \ \textbf{then}\ \ T_1\ \ \textbf{else}\ \ T_2)^{\blacktriangle} &= \textbf{if}\ \ expr\ \ \textbf{then}\ \ T_1^{\blacktriangle}\ \ \textbf{else}\ \ T_2^{\blacktriangle} \\
expr^{\blacktriangle} &= \textbf{invoke}\ \ expr
\end{aligned}
$$

In the rest of this section, we prove that this mapping yields an isomorphism between the reduction rules of CPS-convertible terms and well-formed CPS terms, whence the correctness of our CPS conversion (Theorem 3.17).

We first prove two lemmas to show that $^{\blacktriangle}$ yields only well-formed CPS terms. This leads to a third lemma to show that $^{\blacktriangle}$ is a bijection between CPS-convertible terms and well-formed CPS terms.

CPS-convertible terms have been carefully designed to make CPS conversion as simple as possible. Accordingly, the following three proofs, while long and tedious, are fairly trivial.

Lemma 3.13. *Let Q be a continuation queue. Then $Q^{\blacktriangle}$ is well-formed.*

Proof. By induction on the structure of a tail.

$$\epsilon^{\blacktriangle} = \textbf{invoke}$$

and

$$(\epsilon\ ;\ f(expr,\ldots,expr))^{\blacktriangle} = \textbf{push}\ \ f(expr,\ldots,expr)\ ;\ \textbf{invoke}$$

are well-formed by definition.

$$((Q\ ;\ F)\ ;\ f(expr,\ldots,expr))^{\blacktriangle} = \textbf{push}\ \ f(expr,\ldots,expr)\ ;\ (Q\ ;\ F)^{\blacktriangle}$$

and

$$(Q\ ;\ f(expr,\ldots,expr,F))^{\blacktriangle} = \textbf{push}\ \ f(expr,\ldots,expr,\boxdot)\ ;\ (Q\ ;\ F)^{\blacktriangle}$$

are well-formed by induction. □

Lemma 3.14. *Let T be a CPS-convertible term. Then $T^{\blacktriangle}$ is well-formed.*

Proof. Induction on the structure of T, using the above lemma. □

Lemma 3.15. *The $^{\blacktriangle}$ relation is a bijection between CPS-convertible terms and well-formed CPS terms.*

Proof. Consider the following mapping from well-formed CPS terms to CPS-convertible terms:

$$
\begin{aligned}
(\textbf{push}\ \ f(expr,\ldots,expr)\ ;\ Q)^{\blacktriangledown} &= Q^{\blacktriangledown}\ ;\ f(expr,\ldots,expr) \\
(\textbf{push}\ \ f(expr,\ldots,expr,\boxdot)\ ;\ Q)^{\blacktriangledown} &= Q'\ ;\ f(expr,\ldots,expr,F) \\
&\qquad \text{with } Q^{\blacktriangledown} = Q'\ ;\ F \qquad (*) \\
\textbf{invoke}^{\blacktriangledown} &= \epsilon \\
(x \coloneqq expr\ ;\ T)^{\blacktriangledown} &= x \coloneqq expr\ ;\ T^{\blacktriangledown} \\
\textbf{if}\ \ expr\ \ \textbf{then}\ \ T_1\ \ \textbf{else}\ \ T_2^{\blacktriangledown} &= \textbf{if}\ \ expr\ \ \textbf{then}\ \ T_1^{\blacktriangledown}\ \ \textbf{else}\ \ T_2^{\blacktriangledown} \\
(\textbf{invoke}\ \ expr)^{\blacktriangledown} &= expr
\end{aligned}
$$

(*) The existence of Q' is guaranteed by well-formedness:

– $\forall T,\ T^{\blacktriangledown} = \epsilon \ \Rightarrow\ T = \textbf{invoke}$ (by disjunction on the definition of $^{\blacktriangledown}$),
– here, $Q \neq \textbf{invoke}$ because ($\textbf{push}\ \ f(expr, \ldots, expr, \boxdot)\ ;\ Q$) is well-formed,
– hence $Q^{\blacktriangledown} \neq \epsilon$.

One checks easily that $(T^{\blacktriangledown})^{\blacktriangle} = T$ and $(T^{\blacktriangle})^{\blacktriangledown} = T$. □

To conclude the proof of isomorphism, we also need an (obviously bijective) mapping from contexts to continuations:

Definition 3.16 (Conversion of contexts)**.**

$$\begin{aligned}
([\,])^{\triangle} &= \varepsilon \\
(C[[\,]\ ;\ f(v_1, \ldots, v_n)])^{\triangle} &= f(v_1, \ldots, v_n) \cdot \mathcal{C} \\
&\quad \textit{with } (C[\,])^{\triangle} = \mathcal{C} \\
(C[[\,]\ ;\ f(v_1, \ldots, v_n, \ominus)])^{\triangle} &= f(v_1, \ldots, v_n, \boxdot) \cdot \mathcal{C} \\
&\quad \textit{with } (C[\,])^{\triangle} = \mathcal{C}
\end{aligned}$$

The correctness theorem follows:

Theorem 3.17 (Correctness of CPS conversion)**.** *The $^{\blacktriangle}$ and $^{\triangle}$ mappings are two bijections, the inverses of which are written $^{\blacktriangledown}$ and $^{\triangledown}$. They yield an isomorphism between reduction rules of CPS-convertible terms and CPS terms.*

Proof. Lemma 3.15 ensures that $^{\blacktriangle}$ is a bijection between CPS-convertible terms and well-formed CPS terms. Moreover, $^{\triangle}$ is an obvious bijection between contexts and continuations.

To complete the proof, we only need to apply $^{\blacktriangle}$, $^{\triangle}$, $^{\blacktriangledown}$ and $^{\triangledown}$ to CPS-convertible terms, contexts, well-formed CPS terms and continuations (respectively) in every reduction rule and check that we get a valid rule in the dual reduction system. The result is summarized in Figure 4. □

$$\langle x \coloneqq expr\ ;\ T, C[\,], \sigma\rangle \to_T \langle T, C[\,], \sigma[x \mapsto v]\rangle \quad \Leftrightarrow \quad \langle x \coloneqq expr\ ;\ T, \mathcal{C}, \sigma\rangle \to_T \langle T, \mathcal{C}, \sigma[x \mapsto v]\rangle \text{ when } \langle expr, \sigma\rangle \to_e^\star v$$

$$\langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, C[\,], \sigma\rangle \to_T \langle T_1, C[\,], \sigma\rangle \quad \Leftrightarrow \quad \langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, \mathcal{C}, \sigma\rangle \to_T \langle T_1, \mathcal{C}, \sigma\rangle \text{ if } \langle expr, \sigma\rangle \to_e^\star \textbf{true}$$

$$\langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, C[\,], \sigma\rangle \to_T \langle T_2, C[\,], \sigma\rangle \quad \Leftrightarrow \quad \langle \textbf{if } expr \textbf{ then } T_1 \textbf{ else } T_2, \mathcal{C}, \sigma\rangle \to_T \langle T_2, \mathcal{C}, \sigma\rangle \text{ if } \langle expr, \sigma\rangle \to_e^\star \textbf{false}$$

$$\langle expr, C[[\,]\ ;\ f(v_1, \ldots, v_n)], \sigma\rangle \to_T \langle \epsilon, C[[\,]\ ;\ f(v_1, \ldots, v_n)]\rangle \quad \Leftrightarrow \quad \langle \textbf{invoke } expr, f(v_1, \ldots, v_n) \cdot \mathcal{C}, \sigma\rangle \to_T \langle \textbf{invoke}, f(v_1, \ldots, v_n) \cdot \mathcal{C}\rangle$$

$$\langle expr, C[[\,]\ ;\ f(v_1, \ldots, v_n, \ominus)], \sigma\rangle \to_T \langle \epsilon, C[[\,]\ ;\ f(v_1, \ldots, v_n, v)]\rangle \quad \Leftrightarrow \quad \langle \textbf{invoke } expr, f(v_1, \ldots, v_n, \boxdot) \cdot \mathcal{C}, \sigma\rangle \to_T \langle \textbf{invoke}, f(v_1, \ldots, v_n, v) \cdot \mathcal{C}\rangle \text{ when } \langle expr, \sigma\rangle \to_e^\star v$$

$$\langle expr, [\,], \sigma\rangle \to_T v \quad \Leftrightarrow \quad \langle \textbf{invoke } expr, \varepsilon, \sigma\rangle \to_T v \text{ when } \langle expr, \sigma\rangle \to_e^\star v$$

$$\langle Q, C[\,], \sigma\rangle \to_T \langle Q[x_i \setminus \sigma\ x_i], C[\,]\rangle \quad \Leftrightarrow \quad \langle Q, \mathcal{C}, \sigma\rangle \to_T \langle Q[x_i \setminus \sigma\ x_i], \mathcal{C}\rangle \text{ for every } x_i \text{ in } \mathrm{dom}(\sigma)$$

$$\langle Q\ ;\ f(v_1, \ldots, v_n), C[\,]\rangle \to_Q \langle Q, C[[\,]\ ;\ f(v_1, \ldots, v_n)]\rangle \quad \Leftrightarrow \quad \langle \textbf{push } f(v_1, \ldots, v_n)\ ;\ Q, \mathcal{C}\rangle \to_Q \langle Q, f(v_1, \ldots, v_n) \cdot \mathcal{C}\rangle$$

$$\langle Q\ ;\ f(v_1, \ldots, v_n, F), C[\,]\rangle \to_Q \langle Q\ ;\ F, C[[\,]\ ;\ f(v_1, \ldots, v_n, \ominus)]\rangle \quad \Leftrightarrow \quad \langle \textbf{push } f(v_1, \ldots, v_n, \boxdot)\ ;\ Q', \mathcal{C}\rangle \to_Q \langle Q', f(v_1, \ldots, v_n, \boxdot) \cdot \mathcal{C}\rangle \text{ when } Q' = (Q\ ;\ F)^{\blacktriangle}$$

$$\langle \epsilon, C[[\,]\ ;\ f(v_1, \ldots, v_n)]\rangle \to_Q \langle T, C[\,], \sigma\rangle \quad \Leftrightarrow \quad \langle \textbf{invoke}, f(v_1, \ldots, v_n) \cdot \mathcal{C}\rangle \to_Q \langle T, \mathcal{C}, \sigma\rangle \text{ when } f(x_1, \ldots, x_n) = T \text{ and } \sigma = \{x_i \mapsto v_i\}$$

Figure 4: Isomorphism between reduction rules

References

[1] William D. Clinger. Proper tail recursion and space efficiency. In *Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation*, PLDI '98, pages 174–185, New York, NY, USA, 1998. ACM.

[2] Olivier Danvy and Ulrik Schultz. Lambda-lifting in quadratic time. In *Functional and Logic Programming*, volume 2441 of *Lecture Notes in Computer Science*, pages 134–151. Springer-Verlag, Berlin, Germany, 2002.

[3] International Organization for Standardization. ISO/IEC 9899:1999 "Programming Languages – C", December 1999.

[4] Gabriel Kerneis and Juliusz Chroboczek. Continuation-Passing C: from threads to events through continuations. January 2012. Submitted for publication.

[5] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1(2):125–159, December 1975.