On-line & Off-line Partial Evaluation:
Semantic Specifications and Correctness Proofs

Charles Consel and Siau Cheng Khoo
Research Report YALEU/DCS/RR-912
June 1992

This work is supported by the Darpa grant N00014-88-K-0573 and NSF CCR-8809919

On-line & Off-line Partial Evaluation: Semantic Specifications and Correctness Proofs *

Charles Consel Siau Cheng Khoo

Yale University
Department of Computer Science
New Haven, CT 06520
{consel, khoo}@cs.yale.edu

May 8, 1992

Abstract

This paper presents semantic specifications and correctness proofs for both on-line and off-line partial evaluation of strict functional programs. To do so, our strategy consists of defining a *core semantics* as a basis for the specification of three non-standard evaluations: instrumented evaluation, on-line and off-line partial evaluation. We then use the technique of *logical relation* to prove the correctness of both on-line and off-line partial evaluation semantics.

Our approach goes beyond previous work [Gom92, Lau90, Jon88b] in that

1. We provide a *uniform framework* to defining and proving correct both on-line and off-line partial evaluation.
2. This work required a formal specification of on-line partial evaluation with polyvariant specialization (which had never been done). We define criteria for its correctness with respect to an instrumented standard semantics. As a byproduct, on-line partial evaluation appears to be based on a fixpoint iteration process, just like binding-time analysis.
3. We show that binding-time analysis, the preprocessing phase of off-line partial evaluation, is an *abstraction* of on-line partial evaluation. Therefore, its correctness can be proven with respect to on-line partial evaluation, instead of with respect to the standard semantics, as is customarily done.
4. Based on the binding-time analysis, we formally *derive* the specialization semantics for off-line partial evaluation. This strategy ensures the correctness of the resulting semantics.

1 Introduction

Partial evaluation is the process of constructing a new program given some original program and a part of its input [Fut71]. It is considered a realization of the s_n^m theorem in recursive function theory [Kle52]. Therefore, a faithful partial evaluator must satisfy the following criterion:

*This research was supported in part by NSF and DARPA grants CCR-8809919 and N00014-91-J-4043, respectively. The second author was also supported by a National University of Singapore Overseas Graduate Scholarship.

Suppose that $P(x, y)$ is a program with two arguments, whose first argument has a known value c, but whose second argument is unknown. Partial evaluation of $P(c, y)$ with an unknown value for y should result in a specialized residual program $P_c(y)$ such that:

$$\forall y \in Y,\ P(c, y) = P_c(y). \tag{1}$$

In essence, a partial evaluator is a program specializer and is expected to produce more efficient programs [Jon90]. In practice, there are two different strategies of partial evaluation: *on-line* and *off-line*. An on-line partial evaluator processes a program in one single phase. This process can be viewed as a derivation from the standard evaluation [HM89]. An off-line partial evaluator performs some analyses before specializing the program; the main analysis performed is *binding-time analysis* [Jon88a]. Prior to specialization, this analysis determines the static and dynamic expressions of a program given a known/unknown division of its input. The static expressions are evaluated at partial-evaluation time, and the dynamic expressions are evaluated at run-time. As such, binding-time analysis can naturally be viewed as an abstraction of the on-line partial-evaluation process, but this has not been proven, not even stated formally.

1.1 Correctness of Partial Evaluation – An Overview

Regardless of the strategy used, partial evaluation is a non-trivial process, it involves numerous program transformations. Therefore, proving the correctness of this process must go beyond the extensional criterion given by Equation 1 (Section 1); it must be based on the semantics of partial evaluation. This approach should provide the user with a better understanding of the process.

Several works on proving the correctness of partial evaluation have appeared in the literature recently, all dedicated to off-line partial evaluation. In particular, Gomard in [Gom92] defines a denotational semantics of a specializer for lambda calculus,[1] together with its correctness proof. However, the specializer is limited to monovariant specialization (that is, every function in a program can have at most one specialized version created during specialization). In [Lau90], Launchbury defines in a denotational style a binding-time analysis and proves its correctness with respect to the standard semantics. He also shows that his result corresponds to the notion of *uniform congruence*, a restrictive version of the congruence criterion for binding-time analysis defined by Jones [Jon88b]. However, since the correctness proofs are done with respect to the standard semantics, they do not provide any insight as to how binding-time properties are related to the partial-evaluation process, and more specifically to that of on-line partial evaluation.

In this paper, we provide the semantic specifications and the correctness proofs for partial evaluation of first-order strict functional programs (an extension to higher-order programs is discussed in Section 6). This work is distinct from the existing ones in two aspects: First, it provides a correctness proof for *polyvariant* specialization (that is, a function in a program can have more than one specialized version created during specialization); second, it adopts a *uniform approach* for defining and proving the correctness of both the on-line and off-line partial evaluation.

[1] The binding-time information are provided by the user, and therefore its derivation is not included in the semantics.

The Structure of the Semantics

In polyvariant specialization, when a function call is suspended, a specialized version of the function is created; it is this function call that characterizes the specialized function. During partial evaluation, if all suspended function calls are collected, this information characterizes the residual program, and can actually be used to construct it.

This observation prompted us to specify the partial evaluation in terms of collecting interpretation, as described in [HY88] (the resulting semantics is also similar to the minimal function graph (MFG) semantics [JM86]). As a consequence, just like a collecting interpretation, the semantics consists of two functions: the *local semantic function* (or standard semantic function, using the terminology of [HY88]) describes the partial evaluation of expressions. The *global semantic function* (correspondingly, the collecting interpretation) describes the collection of call patterns.

Uniform Approach for Defining and Proving the Correctness of Partial Evaluation

A uniform approach to defining and proving the correctness of both on-line and off-line partial evaluation enables us to define the relationship between the two levels of partial evaluation. Furthermore, it provides a basis for applying techniques of one level to the other. The uniformity of our approach is based on the following two techniques:

1. **Factorized Semantics:** We define a *core semantic* [JM76, JN90] which consists of semantic rules, and uses some uninterpreted domain names and combinator names (Section 2). This semantics forms the basis for all the semantic specifications defined in the paper. In particular, we define an instrumented semantics that extends the standard semantics to capture all function applications performed during program execution (Section 3). Using other interpretations for domains and combinators, we define the on-line partial evaluation semantics (Section 4), the binding-time analysis and the specialization semantics (Section 5). The advantage of a factorized semantics is that different instances can be related at the level of domain definitions and combinator definitions.

2. **Logical Relations:** We use the technique of *logical relations* [Abr90, JN90, MS90] to prove the correctness of partial evaluation semantics. Logical relations are defined (1) to relate the on-line partial evaluation semantics to the instrumented semantics, and (2) to relate the binding-time analysis to the on-line semantics. Since all these semantic specifications are just different interpretations of the core semantics, their relations can be defined locally by relating their domains and combinators. The resulting proofs thus conform closely to our intuition about the relations between these semantics.

Our approach is summarized in Figure 1. Note that, the specializer for off-line partial evaluation can be systematically and correctly derived from its on-line counterpart, using the information collected by the binding-time analysis.

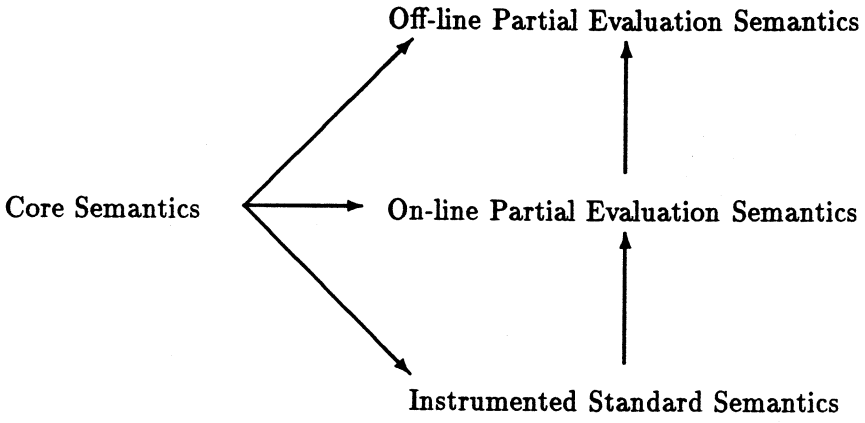

Figure 1: The Factorized Semantics and Logical Relations

Factorized semantics enables the instantiation of various semantics of interest from the core semantics. Logical relation relates two adjacent level of semantics.

1.2 Notation

Most of our notation is that of standard denotational semantics. A domain $\mathbf{D}$ is a *pointed cpo* — a chain-complete partial order with a least element $\perp_D$ (called "bottom"). As is customary, during a computation $\perp_D$ means "not yet calculated" [JN90]. A domain has a binary ordering relation denoted by $\sqsubseteq_D$. The infix least upper bound (lub) operator for the domain $\mathbf{D}$ is written $\sqcup_D$; its prefix form, which computes the lub of a set of elements, is denoted $\bigsqcup_D$. Thus we have that for all $d \in \mathbf{D}$, $\perp_D \sqsubseteq_D d$ and $\perp_D \sqcup_D d = d$. A domain is *flat* if all its elements apart from $\perp$ are incomparable with each other. Domain subscripts are often omitted, as in $\perp \sqcup d$, when they are clear from context.

The notation "$d \in \mathbf{D} = \cdots$" defines the domain (or set) $\mathbf{D}$ with "typical element" d, where $\cdots$ provides the domain specification usually via some combination of the following domain constructions: $\mathbf{D}_\perp$ denotes the domain $\mathbf{D}$ lifted with a new least element $\perp$. $\mathcal{P}(\mathbf{D})$ *denotes the powerset domain whose least element is the empty set, and whose partial-order relation is the subset inclusion.* $\mathbf{D}_1 \to \mathbf{D}_2$ denotes the domain of all *continuous functions* from $\mathbf{D}_1$ to $\mathbf{D}_2$. $\mathbf{D}_1 + \mathbf{D}_2$ and $\mathbf{D}_1 \times \mathbf{D}_2$ denote the separated sum and product, respectively, of the domains $\mathbf{D}_1$ and $\mathbf{D}_2$. All domain/subdomain coercions are omitted when clear from context.

The ordering on functions f, $f' \in \mathbf{D}_1 \to \mathbf{D}_2$ is defined in the standard way: $f \sqsubseteq f' \Leftrightarrow (\forall d \in \mathbf{D}_1)\ f(d) \sqsubseteq f'(d)$.. A function $f \in \mathbf{D}_1 \to \mathbf{D}_2$ is *monotonic* iff it satisfies $(\forall d, d' \in \mathbf{D}_1)$ $d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$; it is *continuous* if in addition it satisfies $f(\bigsqcup\{d_i\}) = \bigsqcup\{f(d_i)\}$ for any chain $\{d_i\} \subseteq \mathbf{D}_1$. A function $f \in \mathbf{D}_1 \to \mathbf{D}_2$ is said to be *strict* if $f(\perp_{D_1}) = \perp_{D_2}$. An element $d \in \mathbf{D}$ is a *fixpoint* of $f \in \mathbf{D} \to \mathbf{D}$ iff $f(d) = d$; it is the *least fixpoint* if for every other fixpoint d', we have that $d \sqsubseteq d'$. The composition of function $f \in \mathbf{D}_1 \to \mathbf{D}_2$ with $f' \in \mathbf{D}_2 \to \mathbf{D}_3$ is denoted by $f' \circ f$.

Angle brackets are used for tupling. If $d = \langle d_1, \ldots, d_n \rangle \in \mathbf{D}_1 \times \cdots \times \mathbf{D}_n$, then for all $i \in \{1, \ldots, n\}$, $d{\downarrow}i$ denotes the i-th element (that is, d_i) of d. For convenience, in the context of a smashed product, that is, $d \in \mathbf{D}_1 \otimes \cdots \otimes \mathbf{D}_n$, d^i denotes the i-th element of d. Syntactic objects

$$
\begin{array}{rcll}
c & \in & \textbf{Const} & \text{Constants} \\
x & \in & \textbf{Var} & \text{Variables} \\
p & \in & \textbf{Po} & \text{Primitive Functions} \\
f & \in & \textbf{Fn} & \text{Function Names} \\
e & \in & \textbf{Exp} & \text{Expressions} \\
e & ::= & c \mid x \mid p\ (e_1, \cdots, e_n) \mid f\ (e_1, \cdots, e_n) \mid \mathit{if}\ e_1\ e_2\ e_3 & \\
\textbf{Prog} & ::= & \{f_i(x_1, \cdots, x_n) = e_i\} \quad (f_1 \text{ is the main function}) &
\end{array}
$$

Figure 2: Syntactic Domains of the Subject Language

are consistently enclosed in double brackets, as in $[\![e]\!]$. Square brackets are used for environment update, as in $env[d/[\![x]\!]]$, which is equivalent to the function $\lambda v\ .\ \mathit{if}\ v = [\![x]\!]\ \ \mathit{then}\ d\ \mathit{else}\ env(v)$. The notation $env[d_i/[\![x_i]\!]]$ is shorthand for $env[d_1/[\![x_1]\!], \ldots, d_n/[\![x_n]\!]]$, where the subscript bounds are inferred form context. "New" environments are created by $\perp[d_i/[\![x_i]\!]]$. Similar notations are also used to denote cache, cache update and new cache respectively.

The paper describes three levels of evaluations: standard evaluation, on-line partial evaluation and off-line partial evaluation. A symbol s is noted $\hat{s}$ if it is used in on-line partial evaluation and $\tilde{s}$ in off-line partial evaluation. Symbols that refer to standard semantics are unannotated. For generality, any symbol used in either on-line or off-line partial evaluation is noted $\overline{s}$. Finally, an algebra is noted $[\mathbf{A}; \mathbf{O}]$ where $\mathbf{A}$ is the *carrier* of the algebra and $\mathbf{O}$ a set of functions operating on this domain. All operations of an algebra are assumed to be continuous.

2 Core Semantics

We begin the discussion of semantic specification of partial evaluation by presenting a core semantics. The subject language is a first-order functional language. Figure 2 defines its syntactic domains. The meaning of a program is the meaning of function f_1. We assume all functions (and primitive functions) have the same arity.

The core semantics is defined in Figure 3. It is used as a basis for all the other semantic specifications defined later, and it factors out the common components of those semantic specifications. This semantics is composed of two valuation functions: $\overline{\mathcal{E}}$ and $\overline{\mathcal{A}}$. Briefly, $\overline{\mathcal{E}}$ defines the standard/abstract semantics (called the *local* semantics) for the language constructs, while $\overline{\mathcal{A}}$ defines a process which collects information globally (called the *global* semantics). The structure of the core semantics is similar to that used in [HY88] for defining collecting interpretation. A similar structure is also used by Sestoft for defining binding-time analysis [Ses85].

The core semantics is defined by semantic rules. It uses some uninterpreted domain names and combinator names. A semantic specification is defined by providing an interpretation to these domains and combinators. As a result, the relation between two semantic specifications defined from a core semantics can simply be based on their domains and their combinators. In fact, all three semantic specifications presented in this paper are defined from the core semantic displayed in Figure 3. Also, to prove their correctness we use the relations defined between their domains and combinators. This is depicted in Figure 4.

1. $\overline{\mathcal{E}}$: **Exp** $\rightarrow$ *ECont* *where* *ECont* = $\overline{Env} \rightarrow Result_{\overline{\mathcal{E}}}$

$\overline{\mathcal{E}}[c] = Const_{\overline{\mathcal{E}}}\ [c]$

$\overline{\mathcal{E}}[x] = VarLookup_{\overline{\mathcal{E}}}\ [x]$

$\overline{\mathcal{E}}[p(e_1,\cdots,e_n)] = PrimOp_{\overline{\mathcal{E}}}\ [p](\overline{\mathcal{E}}[e_1],\ldots,\overline{\mathcal{E}}[e_n])$

$\overline{\mathcal{E}}[if\ e_1\ e_2\ e_3] = Cond_{\overline{\mathcal{E}}}\ (\overline{\mathcal{E}}[e_1],\ \overline{\mathcal{E}}[e_2],\ \overline{\mathcal{E}}[e_3])$

$\overline{\mathcal{E}}[f(e_1,\cdots,e_n)] = App_{\overline{\mathcal{E}}}\ [f](\overline{\mathcal{E}}[e_1],\ldots,\overline{\mathcal{E}}[e_n])$

where $Const_{\overline{\mathcal{E}}}$: **Const** $\rightarrow$ *ECont*

$VarLookup_{\overline{\mathcal{E}}}$: **Var** $\rightarrow$ *ECont*

$PrimOp_{\overline{\mathcal{E}}}$: **Po** $\rightarrow ECont^n \rightarrow$ *ECont*

$Cond_{\overline{\mathcal{E}}}$: $ECont^3 \rightarrow$ *ECont*

$App_{\overline{\mathcal{E}}}$: **Fn** $\rightarrow ECont^n \rightarrow$ *ECont*

2. $\overline{\mathcal{A}}$: **Exp** $\rightarrow$ *ACont* *where* *ACont* = $\overline{Env} \rightarrow Result_{\overline{\mathcal{A}}}$

$\overline{\mathcal{A}}[c] = Const_{\overline{\mathcal{A}}}\ [c]$

$\overline{\mathcal{A}}[x] = VarLookup_{\overline{\mathcal{A}}}\ [x]$

$\overline{\mathcal{A}}[p(e_1,\cdots,e_n)] = PrimOp_{\overline{\mathcal{A}}}\ [p]\ (\overline{\mathcal{A}}[e_1],\ldots,\ \overline{\mathcal{A}}[e_n])$

$\overline{\mathcal{A}}[if\ e_1\ e_2\ e_3] = Cond_{\overline{\mathcal{A}}}\ (\overline{\mathcal{A}}[e_1],\ \overline{\mathcal{A}}[e_2],\ \overline{\mathcal{A}}[e_3])\ (\overline{\mathcal{E}}[e_1])$

$\overline{\mathcal{A}}[f(e_1,\cdots,e_n)] = App_{\overline{\mathcal{A}}}\ [f]\ (\overline{\mathcal{A}}[e_1],\ldots,\ \overline{\mathcal{A}}[e_n])\ (\overline{\mathcal{E}}[e_1],\ldots,\ \overline{\mathcal{E}}[e_n])$

where $Const_{\overline{\mathcal{A}}}$: **Const** $\rightarrow$ *ACont*

$VarLookup_{\overline{\mathcal{A}}}$: **Var** $\rightarrow$ *ACont*

$PrimOp_{\overline{\mathcal{A}}}$: **Po** $\rightarrow ACont^n \rightarrow$ *ACont*

$Cond_{\overline{\mathcal{A}}}$: $ACont^3 \rightarrow ECont \rightarrow$ *ACont*

$App_{\overline{\mathcal{A}}}$: **Fn** $\rightarrow ACont^n \rightarrow ECont^n \rightarrow$ *ACont*

Figure 3: Core Semantics

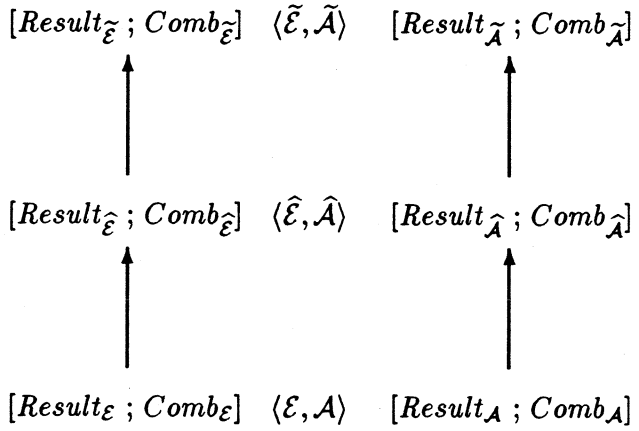

Figure 4: Relations between three levels of evaluation

$Result_{\overline{\mathcal{E}}}$ and $Result_{\overline{\mathcal{A}}}$ are the result domains used by semantic functions $\overline{\mathcal{E}}$ and $\overline{\mathcal{A}}$ respectively. $Comb_{\overline{\mathcal{E}}}$ and $Comb_{\overline{\mathcal{A}}}$ are their respective set of combinators.

3 Standard and Instrumented Semantics

3.1 The Semantics Specifications

In Figure 5 the core semantics is instantiated to define the standard semantics of the language. As is customary, we will omit summand projections and injections. Only interpretation of the valuation function $\overline{\mathcal{E}}$ is provided since the definition of standard semantics does not require collecting information globally. For a function f, "*strict* f" is a function just like f except that it is strict in all its arguments.

In order to investigate the relationship between the standard semantics and the partial evaluation semantics, the standard semantics is enriched to capture information about function applications. The enhanced semantics, called *instrumented* semantics, collects all function calls performed during the standard execution of a program. Function calls are recorded in a *cache*, which maps a function name to a set of *standard signatures*[2]. A standard signature consists of the value of the arguments to a function application. This is depicted in Figure 6.

- Semantic Domains
  $v \in Result_{\mathcal{E}} = \mathbf{Values} = \mathbf{Int} + \mathbf{Bool}$
  $\rho \in VarEnv = \mathbf{Var} \rightarrow \mathbf{Values}$
  $\phi \in FunEnv = \mathbf{Fn} \rightarrow \mathbf{Values}^n \rightarrow \mathbf{Values}$
  $Env = VarEnv \times FunEnv$
- Valuation Functions
  $\mathcal{E}_{Prog} : \mathbf{Prog} \rightarrow \mathbf{Values}^n \rightarrow \mathbf{Values}$
  $\mathcal{E}_{Prog}\,[\![\{ f_i(x_1, \cdots, x_n) = e_i\}]\!]\langle v_1, \ldots, v_n\rangle =$
  $\phi\,[\![f_1]\!]\,(v_1, \ldots, v_n)$ *whererec* $\phi = \perp[strict\,\{\lambda(v_1, \cdots, v_n)\,.\,\mathcal{E}[\![e_i]\!]\,((\perp[v_k/x_k]), \phi)\}/f_i]$
  $\mathcal{E} = \overline{\mathcal{E}}$
- Combinator Definitions
  $Const_{\mathcal{E}}\,[\![c]\!] = \lambda(\rho, \phi)\,.\,\mathcal{K}\,[\![c]\!]$
  $VarLookup_{\mathcal{E}}\,[\![x]\!] = \lambda(\rho, \phi)\,.\,\rho[\![x]\!]$
  $PrimOp_{\mathcal{E}}\,[\![p]\!]\,(k_1, \ldots, k_n) = \lambda(\rho, \phi)\,.\,\mathcal{K}_P[\![p]\!]\,(k_1(\rho, \phi), \ldots, k_n(\rho, \phi))$
  $Cond_{\mathcal{E}}\,(k_1,\ k_2,\ k_3) = \lambda(\rho, \phi)\,.\ k_1(\rho, \phi) \rightarrow k_2(\rho, \phi),\ k_3(\rho, \phi)$
  $App_{\mathcal{E}}\,[\![f]\!]\,(k_1, \ldots, k_n) = \lambda(\rho, \phi)\,.\,\phi[\![f]\!]\,(k_1(\rho, \phi), \ldots, k_n(\rho, \phi))$

Figure 5: Standard Semantics

3.2 Correctness of Instrumentation

Because the local semantics is exactly identical to the standard semantics, we only need to show that the instrumentation part of the instrumented semantics is correct. That is, the instrumented semantics captures (in the cache) all the calls performed during standard evaluation. Since the language we consider is strict, only those standard signatures that represent function calls with non-bottom argument values are collected in the cache. We shall refer to these function calls as *non-trivial* calls.

[2]Notice that powerset, instead of powerdomain, is used to model the content of the cache. This avoids some technical complication incurred in the correctness proof, as discussed in [HY88].

- Semantic Domains

  $v \in Result_{\mathcal{E}} = \mathbf{Values} = $ *as in Figure 5*

  $\rho \in VarEnv = $ *as in Figure 5*

  $\phi \in FunEnv = $ *as in Figure 5*

  $\sigma \in Result_{\mathcal{A}} = \mathbf{Cache}_{\mathcal{A}} = \mathbf{Fn} \rightarrow \mathcal{P}(\mathbf{Values}^n)$

- Valuation Functions

  $\mathcal{E}_{Prog} : \mathbf{Prog} \rightarrow \mathbf{Values}^n \rightarrow \mathbf{Cache}_{\mathcal{A}}$

  $\mathcal{E}_{Prog}\ [\![\{\ f_i(x_1,\cdots,x_n) = e_i\}]\!]\langle v_1,\ldots,v_n\rangle = h(\bot[\{\langle v_1,\ldots,v_n\rangle\}/f_1])$

  $\quad whererec \quad h(\sigma) = \sigma \sqcup h(\bigsqcup\{\mathcal{A}\ [\![e_i]\!]\ (\bot[v_k/x_k])\phi \mid \langle v_1,\ldots,v_n\rangle \in \sigma[\![f_i]\!],\ \forall [\![f_i]\!] \in Dom(\sigma)\})$

  $\quad\quad\quad\quad\ \ \phi = \bot[strict\ (\lambda(v_1,\cdots,v_n)\ .\ \mathcal{E}\ [\![e_i]\!]\ (\bot[v_k/x_k])\ \phi)/f_i]$

  $\mathcal{E} = \overline{\mathcal{E}}$

  $\mathcal{A} = \overline{\mathcal{A}}$

- Combinator Definitions

  $Const_{\mathcal{E}}\ [\![c]\!] = $ *as in Figure 5*

  $VarLookup_{\mathcal{E}}\ [\![x]\!] = $ *as in Figure 5*

  $PrimOp_{\mathcal{E}}\ [\![p]\!]\ (k_1,\ldots,k_n) = $ *as in Figure 5*

  $Cond_{\mathcal{E}}\ (k_1,\ k_2,\ k_3) = $ *as in Figure 5*

  $App_{\mathcal{E}}\ [\![f]\!]\ (k_1,\ldots,k_n) = $ *as in Figure 5*

  $Const_{\mathcal{A}}\ [\![c]\!] = \lambda(\rho,\phi)\ .\ (\lambda f\ .\ \{\})$

  $VarLookup_{\mathcal{A}}\ [\![x]\!] = \lambda(\rho,\phi)\ .\ (\lambda f\ .\ \{\})$

  $PrimOp_{\mathcal{A}}\ [\![p]\!]\ (a_1,\ldots,a_n) = \lambda(\rho,\phi)\ .\ \bigsqcup_{i=1}^{n} a_i(\rho,\phi)$

  $Cond_{\mathcal{A}}\ (a_1,\ a_2,\ a_3)\ k_1 = \lambda(\rho,\phi)\ .\ a_1(\rho,\phi) \sqcup (k_1(\rho,\phi) \rightarrow a_2(\rho,\phi),\ a_3(\rho,\phi))$

  $App_{\mathcal{A}}\ [\![f]\!]\ (a_1,\ldots,\ a_n)(\ k_1,\ldots,\ k_n) = \lambda(\rho,\phi)\ .\ \bigsqcup_{i=1}^{n} a_i(\rho,\phi)) \sqcup\ (\exists i \in \{1,\ldots,n\}\ s.t.\ v_i = \bot \rightarrow (\lambda f\ .\ \{\}),\ \bot[\{\langle v_1,\ldots,v_n\rangle\}/f])$

  $where\ v_i = k_i(\rho,\phi)\quad \forall\, i \in \{1,\cdots,n\}$

Figure 6: Instrumented Semantics capturing function calls

Theorem 1 (Correctness of Instrumentation) *Let P be a program evaluated with input $\langle v_1,\ldots,v_n\rangle$. For any user-defined function f in P, if f is called with non-bottom argument $\langle v'_1,\ldots,v'_n\rangle$ during the standard evaluation, then $\langle v'_1,\ldots,v'_n\rangle \in \sigma[\![f]\!]$.*

The proof is given in Appendix A.

4 On-Line Partial Evaluation Semantics

In this section, we instantiate the core semantics to on-line partial evaluation. Using logical relation, we then present the specification and correctness proof of on-line partial evaluation. But first, we present the notion of partial-evaluation algebra.

4.1 Partial-evaluation Algebra

At the standard evaluation level, primitive operations can be captured by the **algebra [Values; O]**; where **Values** is the domain of basic values and **O** is the set of primitive functions. At the on-

line partial evaluation level, because it is a program transformation process, primitives operate on constants instead of values. Also, because a program is processed with a partial input, a primitive might be invoked with some non-constant arguments, and thus return a non-constant value. These observations are captured by the *partial-evaluation algebra*: an abstraction of the standard algebra **[Values; O]**.

Definition 1 (Partial-Evaluation Algebra) *Let* **[Values; O]** *be an algebra consisting of the domain of basic values and a set of primitive functions, the* partial-evaluation algebra **[$\widehat{\text{Values}}$; $\widehat{\text{O}}$]** *consists of the following components:*

1. *The domain of basic values* **Values** *and the domain of constants* $\widehat{\text{Values}}$ *are related by the abstraction function* $\hat{\tau}$

$$\begin{array}{rcll} \hat{\tau} & : & \textbf{Values} \rightarrow \widehat{\textbf{Values}} & \\ \hat{\tau}(x) & = & \bot_{\widehat{Values}} & \textit{if } x = \bot_{Values} \\ & & \mathcal{K}^{-1}(x) & \textit{otherwise} \end{array}$$

2. $\forall\, p \in \mathbf{O}$ *of arity* n, *there exists a corresponding abstract version* $\hat{p} \in \hat{\mathbf{O}}$ *such that*

$$\hat{p} : \widehat{\textbf{Values}}^{n} \rightarrow \widehat{\textbf{Values}}$$

$$\hat{p} = \lambda\ (\hat{d}_1, \cdots, \hat{d}_n)\ .\ \exists i \in \{1, \cdots, n\}\ s.t.\ \hat{d}_i = \bot_{\widehat{Values}} \rightarrow \bot_{\widehat{Values}},$$

$$\bigwedge_{i=1}^{n} (\hat{d}_i \in \textbf{Const}) \rightarrow \hat{\tau}(\mathcal{K}_p[\![p]\!](d_1, \cdots, d_n)), \top_{\widehat{Values}}$$

$$where\ d_i\ =\ \mathcal{K}\,[\![\hat{d}_i]\!] \quad \forall\ i\ \in\ \{1, \cdots, n\}$$

where $\mathcal{K}^{-1}$ is a monotonic semantics function that converts a basic value to its textual representation. Because **Values** is the sum of basic domains, $\hat{\tau}$ is actually a family of abstraction functions indexed by the summands. Domain $\widehat{\text{Values}}$ — refered to as the *partial-evaluation domain* — is constructed by adding elements $\bot_{\widehat{Values}}$ and $\top_{\widehat{Values}}$ to the set of constants denoted by **Const**; $\bot_{\widehat{Values}}$ and $\top_{\widehat{Values}}$ are respectively weaker and stronger than all the elements of **Const**. Value $\bot_{\widehat{Values}}$ corresponds to $\bot_{Values}$, while value $\top_{\widehat{Values}}$ represents a non-constant value. If a primitive call partially evaluates to value $\top_{\widehat{Values}}$, a residual expression for this primitive call has to be constructed, so that it be performed at run-time, when the complete input is available. In fact, the partial evaluation algebra captures the primitive operations at partial-evaluation time. This provides a modular definition of on-line partial evaluation and it facilitates the proofs.

Notice that the abstract primitives defined in $\hat{\mathbf{O}}$ satisfies the following safety criterion:

$\forall v \in \textbf{Values}$, $\forall p \in \mathbf{O}$ and its corresponding abstract version $\hat{p} \in \hat{\mathbf{O}}$,

$$\hat{\tau} \circ p\ (v)\ \sqsubseteq_{\widehat{Values}}\ \hat{p} \circ \hat{\tau}\ (v)$$

The relation between **[Values; O]** and the partial-evaluation algebra can be succintly described by a logical relation ([Nie89, JN90]) $\sqsubseteq_{\hat{\tau}}$ defined as follows:

1. $\forall\, d\ \in \textbf{Values}$, $\forall\, \hat{d} \in \widehat{\textbf{Values}}$: $\quad d \sqsubseteq_{\hat{\tau}} \hat{d} \Leftrightarrow \hat{\tau}(d) \sqsubseteq_{\widehat{Values}} \hat{d}$.
2. $\forall\, p \in \mathbf{O}$ *and* $\hat{p} \in \hat{\mathbf{O}}$, $\quad p \sqsubseteq_{\hat{\tau}} \hat{p} \Leftrightarrow \forall\, d \in \textbf{Values}, \forall\, \hat{d} \in \widehat{\textbf{Values}} : d \sqsubseteq_{\hat{\tau}} \hat{d} \Rightarrow p(d) \sqsubseteq_{\hat{\tau}} \hat{p}(\hat{d})$

This logical relation forms the basis of the correctness proof of the on-line partial evaluation semantics.

Using the partial-evaluation algebra, we can go one step further and investigate the relation between on-line and off-line partial evaluation. Recall that off-line partial evaluation consists of a binding-time analysis and a specializer. For now, let us examine how the binding-time domain can be captured from the on-line partial evaluation domain.

Usually the binding-time domain, noted $\widetilde{\mathbf{Values}}$, is composed of the binding-time values *Static* and *Dynamic*, lifted with a least element[3] $\perp_{\widetilde{Values}}$. This domain forms a chain, with ordering $\perp_{\widetilde{Values}} \sqsubseteq$ *Static* $\sqsubseteq$ *Dynamic*. $\widehat{\mathbf{Values}}$ and $\widetilde{\mathbf{Values}}$ can be related by the abstraction function $\tilde{\tau}$ defined by

$$\begin{array}{lcll} \tilde{\tau} & : & \widehat{\mathbf{Values}} \rightarrow & \widetilde{\mathbf{Values}} \\ \tilde{\tau}(x) & = & \perp_{\widetilde{Values}} & if\ x = \perp_{\widehat{Values}} \\ & & Static & if\ x \in \mathbf{Const} \\ & & Dynamic & otherwise. \end{array}$$

Domain $\widetilde{\mathbf{Values}}$ is predominantly used in the binding-time analysis, which will be described in Section 5.

4.2 The Semantics Specification

The on-line partial evaluation semantics is displayed in figures 7 and 8. This semantics aims at partially evaluating a program with respect to a partially-known input. It returns a residual program consisting of the specialized functions.

Domain **Exp** is a flat domain of expressions. Besides using $[\![\ ]\!]$ to denote a syntactic fragment, we also use it to construct expressions. This operation is assumed to be strict in all its arguments (*i.e.*, the subexpressions).

The semantics consists of three valuation functions: $\hat{\mathcal{E}}$, $\hat{\mathcal{A}}$ and $\hat{\mathcal{E}}_{Prog}$. Function $\hat{\mathcal{E}}$ defines the partial evaluation of an expression. It produces a pair of values $\hat{v} \in \mathbf{Res} = \mathbf{Exp} \times \widehat{\mathbf{Values}}$, where the first component is a residual expression and the second component is a value in the partial-evaluation domain. **Res** is ordered component-wise.

One of the central issues in partial evaluation of functional programs is the treatment of function calls. Basically, there are two kinds of transformation performed in partially evaluating a function call: *unfold* and *specialization*. The latter includes suspending the call, and specializing the function with respect to the value of the known (static) arguments values. Exactly how a function call is to be treated can be determined by the user, or automatically by some termination analysis (*e.g.*, [Ses88]). To capture this piece of decision making, we introduce the notion of *filters*.

We associate a filter specification to each user-defined function in a subject program. A filter consists of a pair of *strict* and *continuous* functions. The first function determines how to transform a function call (unfold or specialize). The second function specifies how a called function is to be

[3]Note that this three-point domain refines the usual two-point domain {*Static*, *Dynamic*} in that it allows to detect functions in a program that are never invoked, and simple cases of non-terminating computations. Without the value $\perp_{\widehat{Values}}$, these cases would be considered as *Static*.

- Semantic Domains

$\hat{\delta} \in \widehat{\mathbf{Values}}$ $\qquad$ $\hat{\phi} \in \widehat{FunEnv} = \mathbf{Fn} \to \mathbf{Res}^n \to \mathbf{Res}$

$\hat{v} \in \widehat{Result_{\mathcal{E}}} = \mathbf{Res} = \mathbf{Exp} \times \widehat{\mathbf{Values}}$ $\qquad$ $\hat{\sigma} \in \widehat{Result_{\mathcal{A}}} = \widehat{\mathbf{Cache}_{\mathcal{A}}} = \mathbf{Fn} \to \mathcal{P}(\mathbf{Transf} \times \mathbf{Res}^n)$

$\hat{\rho} \in \widehat{VarEnv} = \mathbf{Var} \to \mathbf{Res}$ $\qquad$ $\widehat{Env} = \widehat{VarEnv} \times \widehat{FunEnv}$

- Valuation Functions

$\widehat{\mathcal{E}}_{Prog} : \mathbf{Prog} \to \mathbf{Res}^n \to \mathbf{Prog}_\perp$

$\widehat{\mathcal{E}}_{Prog}\ [\![\{f_i(x_1, \cdots, x_n) = e_i\}]\!]\ (\hat{v}_1, \cdots, \hat{v}_n) = MkProg\ (\hat{h}(\perp[\{\langle \mathbf{s}, \hat{v}_1, \ldots, \hat{v}_n\rangle\}/f_1]))\hat{\phi}$

$\quad whererec\ \hat{h}(\hat{\sigma}) = \hat{\sigma} \sqcup \hat{h}(\bigsqcup\{\widehat{\mathcal{A}}\ [\![e_i]\!]\ (\perp[\hat{v}'_k/x_k], \hat{\phi}) \mid \langle -, \hat{v}'_1, \ldots, \hat{v}'_n\rangle \in \hat{\sigma}[\![f_i]\!], \forall [\![f_i]\!] \in Dom(\hat{\sigma})\})$

$\qquad\qquad \hat{\phi} = \perp[strict\ (\lambda(\hat{v}_1, \cdots, \hat{v}_n)\ .\ \widehat{\mathcal{E}}\ [\![e_i]\!]\ (\perp[\hat{v}_k/x_k], \hat{\phi}))/f_i]$

$\widehat{\mathcal{E}} = \overline{\mathcal{E}}$

$\widehat{\mathcal{A}} = \overline{\mathcal{A}}$

- $MkProg$ Definition

$MkProg\ \hat{\sigma}\ \hat{\phi} = \{\ f_i^{sp}(x_1, \ldots, x_k) = \hat{v}\downarrow 1 \mid \forall \langle \mathbf{s}, \hat{v}_1, \ldots, \hat{v}_n\rangle \in \hat{\sigma}[\![f_i]\!],\ \forall [\![f_i]\!] \in Dom(\hat{\sigma})\}$

$\quad where\ f_i^{sp} = SpName([\![f_i]\!], \hat{v}_1, \ldots, \hat{v}_n)$

$\qquad\quad \hat{v} = \widehat{\mathcal{E}}\ [\![e_i]\!]\ (\perp[\hat{v}_k/x_k], \hat{\phi})$

$\qquad\quad \langle x_1, \ldots, x_k\rangle = ResidPars\ ([\![f_i]\!], \hat{v}_1\downarrow 1, \ldots, \hat{v}_n\downarrow 1)$

- Local Combinator Definitions

$Const_{\widehat{\mathcal{E}}}\ [\![c]\!] = \lambda(\hat{\rho}, \hat{\phi})\ .\ \widehat{\mathcal{K}}\ [\![c]\!]$

$VarLookup_{\widehat{\mathcal{E}}}\ [\![x]\!] = \lambda(\hat{\rho}, \hat{\phi})\ .\ \hat{\rho}\ [\![x]\!]$

$PrimOp_{\widehat{\mathcal{E}}}\ [\![p]\!]\ (\hat{k}_1, \ldots, \hat{k}_n) = \lambda(\hat{\rho}, \hat{\phi})\ .\ \widehat{\mathcal{K}_P}\ [\![p]\!]\ (\hat{k}_1(\hat{\rho}, \hat{\phi}), \ldots,\ \hat{k}_n(\hat{\rho}, \hat{\phi}))$

$Cond_{\widehat{\mathcal{E}}}\ (\hat{k}_1, \hat{k}_2, \hat{k}_3) = \lambda(\hat{\rho}, \hat{\phi})\ .\ (\hat{v}_1\downarrow 2 \in \mathbf{Const}) \to ((\mathcal{K}(\hat{v}_1\downarrow 2)) \to \hat{v}_2, \hat{v}_3),$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle [\![if\ \hat{v}_1\downarrow 1\ \hat{v}_2\downarrow 1\ \hat{v}_3\downarrow 1]\!], \hat{v}_2\downarrow 2\ \sqcup \hat{v}_3\downarrow 2\rangle$

$\qquad\qquad\qquad\qquad where\ \hat{v}_i = \hat{k}_i(\hat{\rho}, \hat{\phi})\ \ \forall i \in \{1,\ 2,\ 3\}$

$App_{\widehat{\mathcal{E}}}\ [\![f]\!]\ (\hat{k}_1, \ldots, \hat{k}_n) = \lambda(\hat{\rho}, \hat{\phi})\ .\ (Ft\ [\![f]\!])\downarrow 1\ (\widehat{bt}(\hat{v}_1), \ldots, \widehat{bt}(\hat{v}_n)) = \mathbf{u}$

$\qquad\qquad\qquad\qquad\qquad\qquad \to \hat{\phi}\ [\![f]\!]\ (\hat{v}_1, \ldots, \hat{v}_n),\ \langle [\![f_{sp}(e''_1, \ldots, e''_k)]\!], \top_{\widehat{Values}}\rangle$

$\qquad\qquad\qquad\qquad\qquad\qquad where\ \hat{v}_i = \hat{k}_i(\hat{\rho}, \hat{\phi})\ \ \forall i \in \{1, \ldots, n\}$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad f_{sp} = SpName([\![f]\!], \hat{v}'_1, \ldots, \hat{v}'_n)$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle e''_1, \ldots, e''_k\rangle = ResidArgs\ ([\![f]\!],\ \langle b_1, \ldots, b_n\rangle,\ \langle \hat{v}_1\downarrow 1, \ldots, \hat{v}_n\downarrow 1\rangle)$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle \hat{v}'_1, \ldots, \hat{v}'_n\rangle = SpPat\ ([\![f]\!], \langle \hat{v}_1, \ldots, \hat{v}_n\rangle,\ \langle b_1, \ldots, b_n\rangle)$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle b_1, \ldots, b_n\rangle = (Ft\ [\![f]\!])\downarrow 2\ (\widehat{bt}(\hat{v}_1), \ldots, \widehat{bt}(\hat{v}_n))$

- Primitive Functions

$\widehat{\mathcal{K}} : \mathbf{Const} \to \mathbf{Res}$

$\widehat{\mathcal{K}}\ [\![c]\!] = \langle [\![c]\!], [\![c]\!]\rangle$

$\widehat{\mathcal{K}} : \mathbf{Po} \to \mathbf{Res}^n \to \mathbf{Res}$

$\widehat{\mathcal{K}_P}\ [\![p]\!]\ (\langle e'_1, \hat{\delta}_1\rangle, \ldots, \langle e'_n, \hat{\delta}_n\rangle) = (\hat{\delta} = \perp_{\widehat{Values}} \to \langle \perp_{Exp}, \perp_{\widehat{Values}}\rangle,$

$\qquad\qquad\qquad\qquad\qquad\qquad (\hat{\delta} \in \mathbf{Const}) \to \langle \hat{\delta}, \hat{\delta}\rangle, \langle [\![p(e'_1, \cdots, e'_n)]\!], \hat{\delta}\rangle$

$\qquad\qquad\qquad\qquad\qquad\qquad where\ \hat{\delta} = \hat{p}(\hat{\delta}_1, \cdots, \hat{\delta}_n)$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad d = \mathcal{K}(\hat{\delta})$

Figure 7: On-Line Partial Evaluation Semantics – Part 1

- Global Combinator Definitions

$$Const_{\widehat{\mathcal{A}}}\,[c] = \lambda(\hat{\rho},\hat{\phi}) \,.\, (\lambda f \,.\, \{\})$$

$$VarLookup_{\widehat{\mathcal{A}}}\,[x] = \lambda(\hat{\rho},\hat{\phi}) \,.\, (\lambda f \,.\, \{\})$$

$$PrimOp_{\widehat{\mathcal{A}}}\,[p]\,(\hat{a}_1,\ldots,\hat{a}_n) = \lambda(\hat{\rho},\hat{\phi}) \,.\, \bigsqcup_{i=1}^{n} \hat{a}_i(\hat{\rho},\hat{\phi})$$

$$Cond_{\widehat{\mathcal{A}}}\,(\hat{a}_1,\hat{a}_2,\hat{a}_3)\,\hat{k}_1 = \lambda(\hat{\rho},\hat{\phi}) \,.\, \hat{a}_1(\hat{\rho},\hat{\phi}) \sqcup \hat{\delta}_1 \in \mathbf{Const} \rightarrow (\mathcal{K}(\hat{\delta}_1) \rightarrow \hat{a}_2(\hat{\rho},\hat{\phi}), \hat{a}_3(\hat{\rho},\hat{\phi})),\ \hat{a}_2(\hat{\rho},\hat{\phi}) \sqcup \hat{a}_3(\hat{\rho},\hat{\phi})$$
$$\text{where } \langle e_1,\hat{\delta}_1\rangle = \hat{k}_1(\hat{\rho},\hat{\phi})$$

$$App_{\widehat{\mathcal{A}}}\,[f]\,(\hat{a}_1,\ldots,\hat{a}_n)\,(\hat{k}_1,\cdots,\hat{k}_n) = \lambda(\hat{\rho},\hat{\phi}) \,.\, (\bigsqcup_{i=1}^{n} \hat{a}_i(\hat{\rho},\hat{\phi})) \sqcup \hat{\sigma}$$
$$\text{where } \hat{v}_i = \hat{k}_i(\hat{\rho},\hat{\phi}) \quad \forall i \in \{1,\cdots,n\}$$
$$\hat{\sigma} = ((Ft[\![f]\!]\downarrow 1(\widehat{bt}(\hat{v}_1),\ldots,\widehat{bt}(\hat{v}_n)) = \mathbf{u}) \rightarrow \bot[\{\langle \mathbf{u},\hat{v}_1,\ldots,\hat{v}_n\rangle\}/f],\ \bot[\{\langle \mathbf{s},\hat{v}'_1,\ldots,\hat{v}'_n\rangle\}/f]$$
$$\langle \hat{v}'_1,\ldots,\hat{v}'_n\rangle = SpPat\,([\![f]\!],\langle \hat{v}_1,\ldots,\hat{v}_n\rangle,\ \langle b_1,\ldots,b_n\rangle)$$
$$\langle b_1,\ldots,b_n\rangle = (Ft\,[\![f]\!])\downarrow 2\ (\widehat{bt}(\hat{v}_1),\ldots,\widehat{bt}(\hat{v}_n))$$

Figure 8: On-Line Partial Evaluation Semantics – Part 2

specialized (it is not used when the call is unfolded): it determines which argument values are to be propagated. (Only arguments with constant values are considered for propagation.) The functionality of a filter is $(\widetilde{\mathbf{Values}}^n \rightarrow \mathbf{T}) \times (\widetilde{\mathbf{Values}}^n \rightarrow \widetilde{\mathbf{Values}}^n)$ where $\widetilde{\mathbf{Values}}$ is the binding-time domain and domain $\mathbf{T}$ contains two values: $\mathbf{u}$ and $\mathbf{s}$, which stand for unfolding and specializing respectively. This strategy has been developed for the partial evaluator Schism [Con88, Con90].

Domain $\mathbf{T}$ is ordered as follows: $\mathbf{u} \sqsubseteq \mathbf{s}$. This ordering reflects our intuition about the termination behavior of these transformations: unfolding a function call will terminate less often than its specialization. This means that replacing the unfolding of a call by its suspension cannot cause non-termination; however, the converse is not true. A detailed discussion on the treatment of calls can be found in [Ses88], for example.

For a function f, the two components of its filter are denoted by $Ft[\![f]\!]\downarrow 1$ and $Ft[\![f]\!]\downarrow 2$ respectively. When a function call is suspended, a specialized function will be created. The specialized function name is denoted by f_i^{sp}. It is uniquely identified by two components: the name of the original function f_i and the specialization pattern.[4]

Function $\widehat{\mathcal{A}}$ collects *partial-evaluation signatures* associated with the user-defined functions. A partial-evaluation signature is created when a non-trivial function call is performed at partial-evaluation time. It consists of two components: A transformation tag indicating the transformation performed on the function, and the argument values of the application. For function specialization, the partial-evaluation signature is a specialization pattern.

All signatures are recorded in a *cache*. Formally, it is defined as

$$\mathbf{Cache}_{\widehat{\mathcal{A}}} = \mathbf{Fn} \rightarrow \mathcal{P}(\mathbf{Transf} \times \mathbf{Res}^n).$$

[4]The specialization pattern describes information about the arguments used in specializing the function. Each argument value belongs to **Res**. The expression component is either a constant (which is to be propagated at function specialization) or a parameter name (representing an unknown argument). Thus, the specialized pattern is defined as $\mathbf{Res}^n$.

The cache is updated using a l.u.b. operation equivalent to the set union. That is, $\forall \sigma_1, \sigma_2 \in \mathbf{Cache}_{\widehat{\mathcal{A}}}$, $\sigma_1 \sqcup \sigma_2 = \lambda f \,.\, (\sigma_1[\![f]\!] \cup \sigma_2[\![f]\!])$.

Lastly, it is worth noticing that, just like a binding-time analysis, $\widehat{\mathcal{E}}_{Prog}$ performs a fixpoint iteration to obtain a cache. Such fixpoint iteration can be viewed as a semantic specification of the pending list technique used in existing partial evaluators. The cache produced will be used by *MkProg* to generate the residual code for all the specialized functions.

The auxiliary functions used in the semantics are listed below. Note that all these functions are *continuous* by construction:

1. *Dom* returns elements in the domain of a function.
2. *SpName* produces a specialized function name from the original function name and the argument pattern. It has the functionality:

$$SpName \;:\; (\mathbf{Fn} \times \mathbf{Res}^n) \rightarrow \mathbf{SpFn}$$

   where **SpFn** is a flat domain of specialized function names.

3. $\widehat{bt} \;:\; \mathbf{Res} \rightarrow \widetilde{\mathbf{Values}}$ returns the binding time of a residual pair. It is defined as $\widehat{bt}(e, \hat{\delta}) = \tilde{\tau}(\hat{\delta})$.
4. If a function call is to be specialized, then
   (a) For those arguments that are *not* propagated at function specialization,
      - $ResidArgs \;:\; \mathbf{Fn} \times \widetilde{\mathbf{Values}}^n \times \mathbf{Exp}^n \rightarrow \mathbf{Exp}^m$ (for $m \leq n$) returns a tuple of residual arguments;
      - $ResidPars \;:\; \mathbf{Fn} \times \mathbf{Exp}^n \rightarrow \mathbf{Var}^m$ (for $m \leq n$) returns a tuple of parameters replacing these residual arguments in the partial-evaluation signature.

   (b) $SpPat \;:\; \mathbf{Fn} \times \mathbf{Res}^n \times \widetilde{\mathbf{Values}}^n \rightarrow \mathbf{Res}^n$ returns the specialization pattern.

$$\begin{aligned} SpPat \;=\; & \lambda(f, \langle \hat{v}_1, \ldots, \hat{v}_n \rangle, \langle b_1, \ldots, b_n \rangle) \,.\, \langle \hat{v}'_1, \ldots, \hat{v}'_n \rangle \\ & \textit{where } \forall\, i \in \{1, \ldots, n\}, \\ & \qquad \hat{v}'_i \;=\; b_i = Static \rightarrow \langle e_i, \hat{\delta}_i \rangle, \\ & \qquad\qquad\; b_i = Dynamic \rightarrow \langle x_i, \top_{\widehat{Values}} \rangle, \langle \bot_{Exp}, \bot_{\widehat{Values}} \rangle \\ & \qquad \hat{v}_i \;=\; \langle e_i, \hat{\delta}_i \rangle \end{aligned}$$

   where $x_1, \ldots, x_n$ are the parameters of function f.

We state here without proof the following two lemmas:

Lemma 1 *$\widehat{\mathcal{E}}$ is continuous in all its arguments.*

Lemma 2 *$\widehat{\mathcal{A}}$ is continuous in all its arguments.*

4.3 Correctness of Partial Evaluation Semantics

Let us first observe that any constant produced by partially evaluating a primitive call is always correct with respect to the standard semantics, modulo termination. This is formalized below, and can be proven from the definition of the partial-evaluation algebra.

Observation 1 *For any primitive function p, let $c = (\widehat{\mathcal{E}}[\![p(x_1, \cdots, x_n)]\!](\bot[(\langle[\![x_i]\!], \hat{\delta}_i\rangle/x_i], \bot))\downarrow 1$, and $v = \mathcal{E}[\![p(x_1, \cdots, x_n)]\!](\bot[d_i/x_i], \bot)$ where $d_i \sqsubseteq_{\hat{\tau}} \hat{\delta}_i$, for $i \in \{1, \ldots, n\}$. Then,*

$$(c \in \mathbf{Const}) \text{ and } v \neq \bot \;\Rightarrow\; c = \hat{\tau}(v)$$

Before proving the correctness of the semantics, we can already show that the partial evaluation semantics subsumes standard evaluation in the following sense:

Theorem 2 *Given a program P. Suppose that (1) the input to P is completely known at partial-evaluation time, and (2) all function calls in P are unfolded during partial evaluation, then for any expression e in P,*

$$\hat{\tau}(\mathcal{E}\ [\![e]\!](\rho, \phi)) \;=\; (\widehat{\mathcal{E}}\ [\![e]\!](\hat{\rho}, \hat{\phi}))\downarrow 1$$

where both $\phi \in FunEnv$ and $\hat{\phi} \in \widehat{FunEnv}$ are fixed for the program, $\rho \in VarEnv$, and $\hat{\rho} \in \widehat{VarEnv}$ is defined as:

$$\hat{\rho} \;=\; \lambda\, [\![x]\!]\, .\, \langle \hat{\tau}(\rho[\![x]\!]), \hat{\tau}(\rho[\![x]\!])\rangle \quad \text{for } (\rho[\![x]\!]) \in \mathbf{D}.$$

The proof is given in Appendix B.

Intuitively, we view the top element in $\widehat{\mathbf{Values}}$, $\top_{\widehat{Values}}$, as a representation for all the possible constant values. Thus, a partially known input $\langle \hat{v}_1, \ldots, \hat{v}_n\rangle$ to a program during partial evaluation represents a set of concrete inputs to that program. That is,

$$\langle \hat{v}_1, \ldots, \hat{v}_n\rangle \text{ represents the set } \{\langle v_1, \ldots, v_n\rangle \mid \hat{\tau}(v_i) \sqsubseteq_{\hat{\tau}} \hat{v}_i\downarrow 2,\ i \in \{1, \ldots, n\}\}$$

The safety criterion described in the begining of Section 1 (Equation 1) can be expressed in our semantic specification as follows: The partial evaluation of a program with input $\langle \hat{v}_1, \ldots, \hat{v}_n\rangle$ is correct if it produces a cache that captures all possible non-trivial calls performed during the execution of a program (under the instrumented semantics) with input taken from the set represented by $\langle \hat{v}_1, \ldots, \hat{v}_n\rangle$. This can be shown by relating the local and global semantics to their respective counterpart in the instrumented semantics. That is, we define a relation $\mathcal{R}^{\widehat{\mathcal{E}}}$ relating $\mathcal{E}$ and $\widehat{\mathcal{E}}$, and a logical relation $\mathcal{R}^{\widehat{\mathcal{A}}}$ relating $\mathcal{A}$ and $\widehat{\mathcal{A}}$. Notice that $\mathcal{R}^{\widehat{\mathcal{E}}}$ relates the results v and $\hat{v}$ computed by $\mathcal{E}$ and $\widehat{\mathcal{E}}$ respectively. Since $\hat{v} = \langle e, \hat{\delta}\rangle \in (\mathbf{Exp} \times \widehat{\mathbf{Values}})$, $\mathcal{R}^{\widehat{\mathcal{E}}}$ is composed of two relations, $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ and $\mathcal{R}^{\widehat{\mathcal{E}}_2}$, that relate a concrete value v to e and $\hat{\delta}$ respectively. It turns out that the correctness of $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ depends on that of $\mathcal{R}^{\widehat{\mathcal{A}}}$. At the same time, the correctness of $\mathcal{R}^{\widehat{\mathcal{A}}}$ depends on the correctness of $\mathcal{R}^{\widehat{\mathcal{E}}_2}$. Therefore, we shall prove the correctness of $\mathcal{R}^{\widehat{\mathcal{E}}_2}$, then that of $\mathcal{R}^{\widehat{\mathcal{A}}}$, and finally that of $\mathcal{R}^{\widehat{\mathcal{E}}_1}$. Lastly, we combine the result of $\mathcal{R}^{\widehat{\mathcal{E}}_2}$ and $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ to express the correctness of $\mathcal{R}^{\widehat{\mathcal{E}}}$.

4.3.1 Correctness of $\mathcal{R}^{\widehat{\mathcal{E}}_2}$

In this section, we define and prove the correctness of the relation $\mathcal{R}^{\widehat{\mathcal{E}}_2}$ between the result of $\mathcal{E}$ and the second component (*i.e.*, the partial-evaluation domain) of the result of $\widehat{\mathcal{E}}$.

Definition 2 (Relation $\mathcal{R}^{\widehat{\mathcal{E}}_2}$) *$\mathcal{R}^{\widehat{\mathcal{E}}_2}$ is a logical relation between domains of $\mathcal{E}$ and $\widehat{\mathcal{E}}$ defined by:*

$$
\begin{aligned}
& v\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v} \quad \Leftrightarrow \quad v \sqsubseteq_{\widehat{\tau}} \hat{v}{\downarrow}2 \\
& \rho\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{VarEnv}\ \hat{\rho} \quad \Leftrightarrow \quad \forall [\![x]\!] \in \mathbf{Var},\ \rho[\![x]\!]\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{Result_{\widehat{\mathcal{E}}}}\ \hat{\rho}[\![x]\!] \\
& \phi\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{FunEnv}\ \hat{\phi} \quad \Leftrightarrow \quad \forall [\![f]\!] \in \mathbf{Fn}, \forall i \in \{1,\ldots,n\}, \forall v_i \in \mathbf{Values}, \forall \hat{v}_i \in \mathbf{Res}, \\
& \qquad\qquad \bigwedge_{i=1}^{n} (v_i\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v}_i) \quad \Rightarrow \quad \phi[\![f]\!](v_1,\ldots,v_n)\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{Result_{\widehat{\mathcal{E}}}}\ \hat{\phi}[\![f]\!](\hat{v}_1,\ldots,\hat{v}_n) \\
& \langle d_1, d_2\rangle\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{D_1\times D_2}\ \langle \hat{d}_1, \hat{d}_2\rangle \quad \Leftrightarrow \quad d_1\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{D_1}\ \hat{d}_1 \ \wedge\ d_2\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{D_2}\ \hat{d}_2 \\
& f\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{D_1\to D_2}\ \hat{f} \quad \Leftrightarrow \quad \forall d \in D_1, \forall \hat{d} \in \widehat{D}_1,\ d\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{D_1}\ \hat{d} \quad \Rightarrow \quad f(d)\ \mathcal{R}^{\widehat{\mathcal{E}}_2}_{D_2}\ \hat{f}(\hat{d}).
\end{aligned}
$$

Lemma 3 *Given a program P. Let ϕ and $\hat{\phi}$ be the two function environments for P defined by the standard and the partial evaluation semantics respectively. Then $\phi\ \mathcal{R}^{\widehat{\mathcal{E}}_2}\ \hat{\phi}$.*

Proof : We need to show that $\forall [\![f]\!] \in \mathbf{Fn}, \forall i \in \{1,\ldots,n\}, \forall v_i \in \mathbf{Values}, \forall \hat{v}_i \in \mathbf{Res}$,

$$\bigwedge_{i=1}^{n} (v_i\ \mathcal{R}^{\widehat{\mathcal{E}}_2}\ \hat{v}_i) \quad \Rightarrow \quad \phi[\![f]\!](v_1,\ldots,v_n)\ \mathcal{R}^{\widehat{\mathcal{E}}_2}\ \hat{\phi}[\![f]\!](\hat{v}_1,\ldots,\hat{v}_n).$$

Since this involves the recursive function environments ϕ and $\hat{\phi}$, we prove the relation using fixpoint induction on Kleene's chain over ϕ and $\hat{\phi}$, with the least element (in this proof, i ranges over all user-defined functions):

$$
\begin{aligned}
\langle \phi_0, \hat{\phi}_0\rangle \ = \ \langle\ & \bot[(strict\ (\lambda(v_1,\ldots,v_n)\ .\ \bot_{Values})/f_i], \\
& \bot[(strict\ (\lambda(\hat{v}_1,\ldots,\hat{v}_n)\ .\ \langle \bot_{Exp}, \bot_{\widehat{Values}}\rangle)/f_i]\rangle.
\end{aligned}
$$

It is true trivially that $\phi_0\ \mathcal{R}^{\widehat{\mathcal{E}}_2}\ \hat{\phi}_0$.

Suppose that $\mathcal{R}^{\widehat{\mathcal{E}}_2}$ is true for some element $\langle \phi_n, \hat{\phi}_n\rangle$ in the ascending chain, we would like to prove that $\mathcal{R}^{\widehat{\mathcal{E}}_2}$ is true for $\langle \phi_{n+1}, \hat{\phi}_{n+1}\rangle$ where

$$
\begin{aligned}
\langle \phi_{n+1}, \hat{\phi}_{n+1}\rangle \ = \ \langle\ & \bot[(strict\ \{\lambda(v_1,\ldots,v_n)\ .\ \mathcal{E}[\![e_i]\!](\bot[v_k/x_k], \phi_n)\}/f_i], \\
& \bot[(strict\ \{\lambda(\hat{v}_1,\ldots,\hat{v}_n)\ .\ \widehat{\mathcal{E}}[\![e_i]\!](\bot[\hat{v}_k/x_k], \hat{\phi}_n)\}/f_i]\rangle.
\end{aligned}
$$

That is, we want to show that
$\forall [\![f]\!] \in \mathbf{Fn}, \forall j \in \{1,\ldots,n\}, \forall v_j \in \mathbf{Values}, \forall \hat{v}_j \in \mathbf{Res}$,

$$\bigwedge_{j=1}^{n} (v_j\ \mathcal{R}^{\widehat{\mathcal{E}}_2}\ \hat{v}_j) \quad \Rightarrow \quad \phi_{n+1}[\![f]\!](v_1,\ldots,v_n)\ \mathcal{R}^{\widehat{\mathcal{E}}_2}\ \hat{\phi}_{n+1}[\![f]\!](\hat{v}_1,\ldots,\hat{v}_n).$$

The proof is by structural induction on e. It suffices to show that $\mathcal{R}^{\widehat{\mathcal{E}}_2}$ holds for all the corresponding pairs of combinators used by $\mathcal{E}$ and $\widehat{\mathcal{E}}$ respectively.

- $Const_{\mathcal{E}}$: $\mathcal{R}^{\widehat{\mathcal{E}_2}}$ is true trivially by comparing $\mathcal{K}$ and $\widehat{\mathcal{K}}$.
- $VarLookup_{\mathcal{E}}$: by structural induction.
- $PrimOp_{\mathcal{E}}$: $PrimOp_{\mathcal{E}}\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ PrimOp_{\widehat{\mathcal{E}}}$ holds by structural induction and a case analysis over the values produced by $PrimOp_{\widehat{\mathcal{E}}}$. Proof is omitted.
- $Cond_{\mathcal{E}}$: $Cond_{\mathcal{E}}\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ Cond_{\widehat{\mathcal{E}}}$ holds by structural induction and a case analysis over the values produced by $\hat{k}_1(\hat{\rho}, \hat{\phi}_n)$.
- $App_{\mathcal{E}}$: For any user-defined function f, all the corresponding arguments of $App_{\mathcal{E}}$ and $App_{\widehat{\mathcal{E}}}$ are related by $\mathcal{R}^{\widehat{\mathcal{E}_2}}$ (by structural induction hypothesis).

  It is easy to show that $\mathcal{R}^{\widehat{\mathcal{E}_2}}$ holds when the function is specialized, since the top element $\top_{\widehat{Values}}$ is returned. For the case when the function is unfolded, $App_{\widehat{\mathcal{E}}}[\![f]\!]\ (\hat{k}_1, \ldots,\ \hat{k}_n)$ is reduced to $\hat{\phi}_n[\![f]\!]\ (\hat{v}_1, \ldots, \hat{v}_n)$, while $App_{\mathcal{E}}[\![f]\!]\ (k_1, \ldots,\ k_n)$ is reduced to $\phi_n[\![f]\!]\ (v_1, \ldots, v_n)$. Since $\phi_n\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ \hat{\phi}_n$ by fixpoint induction hypothesis, we have

  $$\phi_n[\![f]\!](v_1, \ldots, v_n)\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ \hat{\phi}_n[\![f]\!](\hat{v}_1, \ldots, \hat{v}_n).$$

  Hence, $App_{\mathcal{E}}\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ App_{\widehat{\mathcal{E}}}$.

Hence, $\phi\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ \hat{\phi}$. This concludes the proof. □

Theorem 3 (Correctness of Local Semantics – 2nd Component) $\mathcal{E}\ \mathcal{R}^{\widehat{\mathcal{E}_2}}\ \widehat{\mathcal{E}}$.

Proof : From Lemma 3. □

Before we close this section, let us make an observation about the relationship between the first and second components of a value produced by $\widehat{\mathcal{E}}$.

Observation 2 *During partial evaluation, all values* $\hat{v} \in$ **Res** *satisfy the following conditions:*

- $\hat{v}\downarrow 1 \in \mathbf{Const} \wedge \hat{v}\downarrow 2 \in \mathbf{Const} \Leftrightarrow \hat{v}\downarrow 1 = \hat{v}\downarrow 2$
- $\hat{v}\downarrow 1 = \perp_{Exp} \Leftrightarrow \hat{v}\downarrow 2 = \perp_{\widehat{Values}}$.

The above observation comes directly from the definition of $\widehat{\mathcal{K}}_P$ in Figure 7.

We say that a value $\hat{v} \in$ **Res** is $\mathcal{R}$-*consistent* if it satisfies one of the above conditions. This fact is used in the next section.

4.3.2 Correctness of the Global Semantics

In this section, we prove the correctness of the global partial evaluation semantics (1) by relating the semantics of $\widehat{\mathcal{A}}$ with $\mathcal{A}$ using logical relation $\mathcal{R}^{\widehat{\mathcal{A}}}$, and (2) by showing that all the non-trivial calls performed at standard evaluation are captured by $\widehat{\mathcal{A}}$.

Since the result of both $\mathcal{A}$ and $\widehat{\mathcal{A}}$ is a cache, $\mathcal{R}^{\widehat{\mathcal{A}}}$ should relate caches. That is, whenever a standard signature for a function is recorded in the cache produced by $\mathcal{A}$, there exists a logically related partial-evaluation signature for that function in the cache produced by $\widehat{\mathcal{A}}$. Formally,

Definition 3 (Relation $\mathcal{R}^{\widehat{\mathcal{A}}}$) *$\mathcal{R}^{\widehat{\mathcal{A}}}$ is a logical relation between domains of $\mathcal{A}$ and $\widehat{\mathcal{A}}$ defined by:*

$$v\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\varepsilon}}}\ \hat{v} \quad \Leftrightarrow \quad (\hat{v}\ \textit{is}\ \mathcal{R}\textit{-consistent}) \wedge (v \sqsubseteq_{\widehat{\tau}} \hat{v}\downarrow 2)$$

$$\langle v_1, \ldots, v_n \rangle\ \mathcal{R}^{\widehat{\mathcal{A}}}_{(Transf \times Res^n)}\ \langle t, \hat{v}_1, \ldots, \hat{v}_n \rangle \quad \Leftrightarrow \quad \bigwedge_{i=1}^{n} (v_i \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\varepsilon}}} \hat{v}_i)$$

$$\sigma\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{A}}}}\ \hat{\sigma} \quad \Leftrightarrow \quad \forall [\![f]\!] \in Dom(\sigma), \forall s \in \sigma[\![f]\!], \exists \hat{s} \in \hat{\sigma}[\![f]\!],\ s\ \mathcal{R}^{\widehat{\mathcal{A}}}_{(Transf \times Res^n)}\ \hat{s}$$

$$\rho\ \mathcal{R}^{\widehat{\mathcal{A}}}_{VarEnv}\ \hat{\rho} \quad \Leftrightarrow \quad \forall [\![x]\!] \in \mathbf{Var},\ \rho[\![x]\!]\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\varepsilon}}}\ \hat{\rho}[\![x]\!]$$

$$\phi\ \mathcal{R}^{\widehat{\mathcal{A}}}_{FunEnv}\ \hat{\phi} \quad \Leftrightarrow \quad \forall [\![f]\!] \in \mathbf{Fn}, \forall j \in \{1, \ldots, n\}, \forall v_j \in \mathbf{Values}, \forall \hat{v}_j \in \mathbf{Res},$$
$$\bigwedge_{j=1}^{n} (v_j\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\varepsilon}}}\ \hat{v}_j) \quad \Rightarrow \quad \phi[\![f]\!](v_1, \ldots, v_n)\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\varepsilon}}}\ \hat{\phi}[\![f]\!](\hat{v}_1, \ldots, \hat{v}_n)$$

$$\langle d_1, d_2 \rangle\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_1 \times D_2}\ \langle \hat{d}_1, \hat{d}_2 \rangle \quad \Leftrightarrow \quad d_1\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_1}\ \hat{d}_1 \wedge d_2\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_2}\ \hat{d}_2$$

$$f\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_1 \to D_2}\ \hat{f} \quad \Leftrightarrow \quad \forall d \in D_1, \forall \hat{d} \in \hat{D}_1,\ d\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_1}\ \hat{d} \quad \Rightarrow \quad f(d)\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_2}\ \hat{f}(\hat{d}).$$

Note that the $\mathcal{R}$-consistency (Observation 2) ensures that the first component of $\hat{v}$, the residual expression is consistent with the result of the partial-evaluation algebra. Observe that there is no value in the standard signature corresponding to the transformation tag of the partial evaluation signature. In fact, a transformation tag for a standard signature could have been obtained by performing filter computations at the standard semantics level. However, the transformation has no effect on standard evaluation. Furthermore, since filters are continuous, the transformation computed is guaranteed to be more precise or equal to that computed at the on-line level. Thus, we can ignore this information without compromising the correctness proof. Lastly, we note that the l.u.b. operation (which is the set-union operation) on caches is closed under $\mathcal{R}^{\widehat{\mathcal{A}}}$.

The next lemma shows that all the standard signatures recorded in the final cache produced by $\mathcal{A}$ are "captured" in the corresponding cache produced by $\widehat{\mathcal{A}}$ in the sense that they are related by $\mathcal{R}^{\widehat{\mathcal{A}}}$.

Notice that whenever $\widehat{\mathcal{A}}$ uses a value $\hat{v}$ in decision making (combinators $Cond_{\widehat{\mathcal{A}}}$ and $App_{\widehat{\mathcal{A}}}$), only the value of the partial-evaluation domain is used, as is manifested by the definition of functions $SpPat$, $\widehat{bt}$ and Ft. Therefore, only the second component of $\hat{v}$ is needed to show the correctness of $\widehat{\mathcal{A}}$. Although the first component of $\hat{v}$ (the expression) is modified by $\widehat{\mathcal{A}}$ when dealing with combinator $App_{\widehat{\mathcal{A}}}$, it should be noted that the modification is exactly identical to the one done in $\widehat{\mathcal{E}}$, and by Observation 2, the modified value is still $\mathcal{R}$-consistent.

Lemma 4 *Given a program P. For any $\widehat{\mathcal{E}}$ such that $\mathcal{E}\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \widehat{\mathcal{E}}$, let ϕ and $\hat{\phi}$ be two function environments for P defined by the standard and the partial evaluation semantics respectively. For any expression e in P, for any variable environments ρ and $\hat{\rho}$ such that $\rho \mathcal{R}^{\widehat{\mathcal{A}}} \hat{\rho}$,*

$$\mathcal{A}[\![e]\!](\rho, \phi)\ \ \mathcal{R}^{\widehat{\mathcal{A}}}\ \ \widehat{\mathcal{A}}[\![e]\!](\hat{\rho}, \hat{\phi}).$$

Proof : The proof is by structural induction on e. Firstly, notice that

$$\mathcal{E}\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \widehat{\mathcal{E}} \quad \Rightarrow \quad \phi\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{\phi}.$$

The predicate can be proved using Kleene's approximation over Φ, with the least element

$$\begin{aligned}\langle \phi_0, \hat{\phi}_0, \phi'_0 \rangle \; = \langle \; & \bot[(\mathit{strict}\ (\lambda(v_1, v_2)\ .\ \bot_{\mathit{Values}}))/f_i \mid \forall\ [\![f_i]\!] \in \mathbf{Fn}], \\ & \bot[(\mathit{strict}\ (\lambda(\hat{v}_1, \hat{v}_2)\ .\ \bot_{\mathit{Res}}))/f_i \mid \forall\ [\![f_i]\!] \in \mathbf{Fn}], \\ & \bot[(\mathit{strict}\ (\lambda(v_1, v_2)\ .\ \bot_{\mathit{Values}})/f^{sp} \mid \forall\ \mathit{specialized\ function}\ f^{sp}]\rangle\end{aligned}$$

and the predicate $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ over the $n+1^{st}$ approximation being

$$\begin{aligned}\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1} \equiv_{def} \; & \mathcal{R}^{\widehat{\mathcal{E}}_1}\langle \phi_{n+1}, \hat{\phi}_{n+1}, \phi'_{n+1}\rangle \\ = \; & \forall\ [\![f_i]\!] \in \mathbf{Fn},\ \forall v_1, v_2 \in \mathbf{Values},\ \forall \hat{v}_1, \hat{v}_2 \in \mathbf{Res},\ \forall \rho_d \in VarEnv, \\ & \bigwedge_{j=1}^{2} \rho_d\ \mathit{satisfies}\ \langle v_j, \hat{v}_j\rangle \;\Rightarrow\; \phi_{n+1}[\![f_i]\!](v_1, v_2) =_{\bot} \mathcal{E}[\![(\hat{\phi}_{n+1}[\![f_i]\!](\hat{v}_1, \hat{v}_2))\downarrow 1]\!](\rho_d, \phi'_{n+1}) \\ = \; & \forall\ [\![f_i]\!] \in \mathbf{Fn},\ \forall v_1, v_2 \in \mathbf{Values},\ \forall \hat{v}_1, \hat{v}_2 \in \mathbf{Res},\ \forall \rho_d \in VarEnv, \\ & \bigwedge_{j=1}^{2} \rho_d\ \mathit{satisfies}\ \langle v_j, \hat{v}_j\rangle \;\Rightarrow\; \mathcal{E}[\![e_i]\!](\bot[v_k/x_k], \phi_n) =_{\bot} \mathcal{E}[\![(\widehat{\mathcal{E}}[\![e_i]\!](\bot[\hat{v}_k/x_k], \hat{\phi}_n))\downarrow 1]\!](\rho_d, \phi'_{n+1})\end{aligned}$$

Notice that at any $i+1^{st}$ approximation, ϕ'_{i+1} is obtained from the residual program produced by $\widehat{\mathcal{A}}$ and $\widehat{\mathcal{E}}$, both having $\hat{\phi}_{i+1}$ as their function environment. Formally,

$$\phi'_{i+1} \;=\; \bot[(\mathit{strict}\{\lambda\ v\ .\ \mathcal{E}[\![e^{sp}]\!](\bot[v/x], \phi'_i)\})/f^{sp} \mid \forall\ \mathit{specialized\ function}\ f^{sp}\ \mathit{with\ body}\ e^{sp}]$$

ϕ' is derived from cache $\hat{\sigma}$ produced by $\widehat{\mathcal{A}}$ and ϕ'_i is derived from cache $\hat{\sigma}_i$ at the i^{th} approximation. Below are properties about $\hat{\sigma}_i$ and ϕ'_i.

Property 1 $\forall\ i \in \{0, 1, \ldots\},\ \hat{\sigma}_i \sqsubseteq_{Cache} \hat{\sigma}_{i+1}$.

Proof : From the result that $\hat{\sigma}_i$'s are the cache produced by $\widehat{\mathcal{A}}$ with function environment $\hat{\phi}_i$ and $\widehat{\mathcal{A}}$ is continuous in all its arguments. □

Property 2 $\forall\ i \in \{0, 1, \ldots\},\ \phi'_i \sqsubseteq_{FunEnv} \phi'_{i+1}$.

Proof : Since $\forall i \in \{0, 1, \ldots\}$, ϕ'_i is obtained from the residual program, which is the result of $\widehat{\mathcal{E}}_{prog}$. Inspecting the function definition of $\widehat{\mathcal{E}}_{prog}$ shows that it is continuous in all its arguments. In particular, since $\forall\ i \in \{0, 1, \ldots\},\ \hat{\sigma}_i \sqsubseteq_{Cache} \hat{\sigma}_{i+1}$, therefore $\phi'_i \sqsubseteq_{FunEnv} \phi'_{i+1}$. □

We prove the validity of $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ by fixpoint induction:

For the least element, $\langle \phi_0, \hat{\phi}_0, \phi'_0\rangle$, we have $\phi[\![f_i]\!](v_1, v_2) = \bot_{Values}$ and $\hat{\phi}[\![f_i]\!](\hat{v}_1, \hat{v}_2) = \bot_{Res}$. Thus, $\mathcal{R}^{\widehat{\mathcal{E}}_1}\langle \phi_0, \hat{\phi}_0, \phi'_0\rangle$ holds vacuously.

Suppose that $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ is true for some element $\langle \phi_n, \hat{\phi}_n, \phi'_n\rangle$ in the ascending chain, we want to prove that $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ is true for $\langle \phi_{n+1}, \hat{\phi}_{n+1}, \phi'_{n+1}\rangle \;=\; \Phi\langle \phi_n, \hat{\phi}_n, \phi'_n\rangle$.

For clarity, we introduce the following abbreviations:

1. $\bot[v_k/x_k]$ is abbreviated by ρ and $\bot[\hat{v}_k/x_k]$ by $\hat{\rho}$.
2. Given an expression e, we abbreviate $\mathcal{E}[\![e]\!](\rho, \phi_n)$ by $[\![e]\!]_{\mathcal{E}}$, and $\widehat{\mathcal{E}}[\![e]\!](\hat{\rho}, \hat{\phi}_n)$ by $[\![e]\!]_{\widehat{\mathcal{E}}}$.

The proof of $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ requires structural induction on e.

- If e is a constant or a variable, the proof is trivial, and thus omitted.
- e is a primitive call, $[\![p(e_1, \ldots, e_n)]\!]$. Let $v = [\![p(e_1, \ldots, e_n)]\!]\varepsilon$ and $\hat{v} = [\![p(e_1, \ldots, e_n)]\!]_{\widehat{\mathcal{E}}}$.
  - $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds trivially if $\hat{v} = \perp_{Res}$.
  - If $\hat{v}{\downarrow}1$ is a constant, then $\hat{v}{\downarrow}1 = \hat{\tau}(v)$ from Observation 1. Therefore, $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds in this case.
  - If $\hat{v}{\downarrow}1$ is not a constant, then the residual expression is of the form $[\![p(e''_1, \ldots, e''_n)]\!]$, where $e''_i = [\![e_i]\!]_{\widehat{\mathcal{E}}}\ \forall i \in \{1, \ldots, n\}$. By the structural induction hypothesis, $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds for all the arguments of the primitive call. Furthermore, since ρ and $\hat{\rho}$ contain all the bindings for free variables in e, they also contain the bindings for free variables of the arguments. We thus have:

$$
\begin{array}{lll}
\mathcal{E}[\![[\![p(e_1, \ldots, e_n)]\!]_{\widehat{\mathcal{E}}}]\!](\rho_d, \phi'_{n+1}) & & \\
= \quad \mathcal{E}[\![p(e''_1, \ldots, e''_n)]\!](\rho_d, \phi'_{n+1}) & & \\
= \quad \mathcal{K}_P[\![p]\!]((\mathcal{E}[\![e''_1]\!](\rho_d, \phi'_{n+1})), \ldots, (\mathcal{E}[\![e''_n]\!](\rho_d, \phi'_{n+1}))) & & \text{[from standard semantics]} \\
=_{\perp} \quad \mathcal{K}_P[\![p]\!]([\![e_1]\!]\varepsilon, \ldots, [\![e_n]\!]\varepsilon) & & \text{[structural induction hypothesis]} \\
= \quad [\![p(e_1, \ldots, e_n)]\!]\varepsilon & & \text{[from standard semantics]}
\end{array}
$$

  Therefore, $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds.

- e is a conditional expression, $[\![if\ e_1\ e_2\ e_3]\!]$. $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds trivially if e_1 partially evaluates to $\perp_{Res}$. If e_1 is partially evaluated to a constant, then the result of partially evaluating e is obtained from partially evaluating either e_2 or e_3. By the structural induction hypothesis, $\mathcal{R}_{n+1}$ holds.

  If e_1 partially evaluates to a residual expression, then the result of partially evaluating e has the form $[\![if\ e''_1\ e''_2\ e''_3]\!]$, where $e''_i = [\![e_i]\!]_{\widehat{\mathcal{E}}}\ \forall i \in \{1, \ldots, 3\}$. Therefore,

$$
\begin{array}{lll}
\mathcal{E}[\![[\![if\ e_1\ e_2\ e_3]\!]_{\widehat{\mathcal{E}}}]\!](\rho_d, \phi'_{n+1}) & & \\
= \quad \mathcal{E}[\![if\ e''_1\ e''_2\ e''_3]\!](\rho_d, \phi'_{n+1}) & & \\
= \quad (\mathcal{E}[\![e''_1]\!](\rho_d, \phi'_{n+1})) \rightarrow (\mathcal{E}[\![e''_2]\!](\rho_d, \phi'_{n+1})), (\mathcal{E}[\![e''_3]\!](\rho_d, \phi'_{n+1})) & & \text{[from standard semantics]} \\
=_{\perp} \quad [\![e_1]\!]\varepsilon \rightarrow [\![e_2]\!]\varepsilon, [\![e_3]\!]\varepsilon & & \text{[structural induction hypothesis]} \\
= \quad [\![if\ e_1\ e_2\ e_3]\!]\varepsilon & & \text{[from standard semantics]}
\end{array}
$$

  Thus, $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds.

- e is a function application, $[\![f_i(e_1, e_2)]\!]$. Partially evaluating e may result in the application being either unfolded or specialized. Suppose that the application is specialized, without loss of generality, we assume that the first argument of the application is static and propagated, whereas the second argument is dynamic. Then $[\![f_i(e_1, e_2)]\!]_{\widehat{\mathcal{E}}}$ becomes $[\![f^{sp}_i([\![e_2]\!]_{\widehat{\mathcal{E}}})]\!]$ where f^{sp}_i is the specialized function. The partial-evaluation signature obtained from this application is (by the definition of $\widehat{\mathcal{A}}$)

$$
\langle \mathbf{s}, [\![e_1]\!]_{\widehat{\mathcal{E}}}, \langle [\![x_2]\!], \hat{\delta}' \rangle \rangle \quad \textit{where} \quad \begin{array}{l} \langle \hat{v}'_1, \langle [\![x_2]\!], \hat{\delta}' \rangle \rangle = SpPat([\![f_i]\!], \langle [\![e_1]\!]_{\widehat{\mathcal{E}}}, [\![e_2]\!]_{\widehat{\mathcal{E}}} \rangle, \langle Static, Dynamic \rangle) \\ \langle -, \hat{\delta}' \rangle = [\![e_2]\!]_{\widehat{\mathcal{E}}} \end{array}
$$

  Notice from the definition of $SpPat$ that $\hat{v}'_1 = [\![e_1]\!]_{\widehat{\mathcal{E}}}$ and $\hat{\delta}' = \top_{\widehat{Values}}$. The specialized function f^{sp}_i is included in the residual program produced; its definition is as follows.

$$
\begin{array}{rcl}
f^{sp}_i(x_2) & = & [\![(\hat{\phi}_n[\![f_i]\!]([\![e_1]\!]_{\widehat{\mathcal{E}}}, \langle [\![x_2]\!], \hat{\delta}' \rangle))\downarrow 1]\!] \\
& = & [\![(\hat{\phi}_n[\![f_i]\!](\hat{\tau}([\![e_1]\!]\varepsilon), \langle [\![x_2]\!], \hat{\delta}' \rangle))\downarrow 1]\!]
\end{array}
$$

  The last equality holds by structural induction hypothesis and by the fact that only constants are allowed to be propagated for a function specialization. The corresponding entry of f^{sp}_i in ϕ'_{n+1} is

$$
\textit{strict}(\lambda v\ .\ \mathcal{E}[\![(\hat{\phi}_n[\![f_i]\!](\hat{\tau}([\![e_1]\!]\varepsilon), \langle [\![x_2]\!], \hat{\delta}' \rangle))\downarrow 1]\!](\perp[v/x_2], \phi'_n)) \qquad (2)
$$

Thus, we have

$$
\begin{array}{lll}
& \mathcal{E}[\![f_i^{sp}([\![e_2]\!]_{\widehat{\mathcal{E}}})]\!](\rho_d, \phi'_{n+1}) & \\
= & \phi'_{n+1}[\![f_i^{sp}]\!](\mathcal{E}[\![e_2]\!]_{\widehat{\mathcal{E}}}(\rho_d, \phi'_{n+1})) & \\
=_\perp & \phi'_{n+1}[\![f_i^{sp}]\!]([\![e_2]\!]\varepsilon) & \text{[structural induction hypothesis]} \\
= & \mathcal{E}[\![(\hat{\phi}_n[\![f_i]\!](\hat{\tau}([\![e_1]\!]\varepsilon), \langle [\![x_2]\!], \hat{\delta}'\rangle))\downarrow 1]\!](\perp[[\![e_2]\!]\varepsilon/x_2], \phi'_n) & \\
=_\perp & \phi_n[\![f_i]\!]([\![e_1]\!]\varepsilon, [\![e_2]\!]\varepsilon) & \text{[fixpoint induction hypothesis]} \\
= & [\![f_i(e_1, e_2)]\!]\varepsilon &
\end{array}
$$

In the derivation above, the fourth equality is valid based on an instance of our fixpoint induction hypothesis. This is because $(\perp[[\![e_2]\!]\varepsilon/x_2])$ is the only environment that satisfies both the pairs $\langle [\![e_1]\!]\varepsilon, \hat{\tau}([\![e_1]\!]\varepsilon)\rangle$ and $\langle [\![e_2]\!]\varepsilon, \langle [\![x_2]\!], \hat{\delta}'\rangle\rangle$. Therefore, $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds for the application.

On the other hand, consider the case where the application is unfolded. The equality in $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ becomes

$$\phi_n[\![f_i]\!]([\![e_1]\!]\varepsilon, [\![e_2]\!]\varepsilon) \;=_\perp\; \mathcal{E}[\![(\hat{\phi}_n[\![f_i]\!]([\![e_1]\!]_{\widehat{\mathcal{E}}}, [\![e_2]\!]_{\widehat{\mathcal{E}}}))\downarrow 1]\!]\;(\rho_d,\; \phi'_{n+1}). \tag{3}$$

For Equation 3 to hold, ρ_d must satisfy both pairs $\langle [\![e_1]\!]\varepsilon, [\![e_1]\!]_{\widehat{\mathcal{E}}}\rangle$ and $\langle [\![e_2]\!]\varepsilon, [\![e_2]\!]_{\widehat{\mathcal{E}}}\rangle$. That is, $\forall i \in \{1, 2\}$,

$$v_i =_\perp \mathcal{E}[\![\hat{v}_i\downarrow 1]\!](\rho_d, \phi'_{n+1}) \;\wedge\; v_i \sqsubseteq_{\widehat{\tau}} \hat{v}_i\downarrow 2.$$

This is true by the structural induction hypothesis, Property 2 about ϕ'_{n+1}, and Theorem 3.
Using ρ_d, the fixpoint induction hypothesis is

$$\phi_n[\![f_i]\!]([\![e_1]\!]\varepsilon, [\![e_2]\!]\varepsilon) \;=_\perp\; \mathcal{E}[\![(\hat{\phi}_n[\![f_i]\!]([\![e_1]\!]_{\widehat{\mathcal{E}}}, [\![e_2]\!]_{\widehat{\mathcal{E}}}))\downarrow 1]\!](\rho_d, \phi'_n),$$

Notice that the only difference between the hypothesis and Equation 3 is the usage of ϕ'_n and ϕ'_{n+1}. Let $e' = (\hat{\phi}_n[\![f_i]\!](([\![e_1]\!]_{\widehat{\mathcal{E}}}), ([\![e_2]\!]_{\widehat{\mathcal{E}}})))\downarrow 1$. Since the domain **Exp** is flat, the only case where Equation 3 may have failed to hold would be when standard evaluation of e' made references to specialized functions defined in ϕ'_{n+1}. Suppose that f^{sp} were such a function, and its call in e' were $[\![f^{sp}(r'_2)]\!]$. This residual call would be the result of partially evaluating a function call. Let the function call be $[\![f(r_1, r_2)]\!]$. Then, it would be the case that at the n^{th} approximation, we had

$$[\![f(r_1, r_2)]\!]\varepsilon \;=_\perp\; \mathcal{E}[\![[\![f(r_1, r_2)]\!]_{\widehat{\mathcal{E}}}]\!](\rho_d, \phi'_n) \;=\; \mathcal{E}[\![f^{sp}(r'_k)]\!](\rho_d, \phi'_n)$$

be true vacuously (by the hypothesis), but at the $n+1^{st}$ approximation, the equation

$$[\![f(r_1, r_2)]\!]\varepsilon \;=_\perp\; \mathcal{E}[\![f^{sp}(r'_k)]\!](\rho_d, \phi'_{n+1}) \tag{4}$$

became false. However, Equation 4 is the result of function specialization, and we have already proved its validity. Thus, we arrive at a contradiction, and Equation 3 must therefore hold. Hence, $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds.

Hence, $\mathcal{R}^{\widehat{\mathcal{E}}_1}\langle \phi, \hat{\phi}, \phi'\rangle$ holds. This concludes the proof. □

4.3.4 Correctness of $\mathcal{R}^{\widehat{\mathcal{E}}}$

Now, we are ready to define the relation between $\mathcal{E}$ and $\widehat{\mathcal{E}}$. This is defined in terms of the result of Theorem 3 and Lemma 5. Firstly, since both $\mathcal{E}$ and $\widehat{\mathcal{E}}$ take variable environments as their arguments, we need to relate these environments. To do so, we extend the notion of satisfiability to define the relationship between variable environments, instead of pairs of related values. This is a variant of the notion of *agreeability* as defined by Gomard in [Gom92].

Definition 5 (Agreeability) *Let P be a program. Suppose ϕ' is the function environment, defined by the standard semantics, for a specialized version of P. Also, let $\rho, \rho_d \in VarEnv$ be two variable environments defined by the standard semantics, and $\hat{\rho} \in \widehat{VarEnv}$ be a variable environment defined by the partial evaluation semantics. For any expression, e in P, ρ, $\hat{\rho}$ and ρ_d agree on e at ϕ' if*

$$\forall [\![x]\!] \in FV(e),\ \rho[\![x]\!] =_\perp \mathcal{E}[\![(\hat{\rho}[\![x]\!])\downarrow 1]\!](\rho_d, \phi') \wedge \rho[\![x]\!] \sqsubseteq_{\hat{\tau}} (\hat{\rho}[\![x]\!])\downarrow 2.$$

The notion of satisfiability can then be expressed in terms of agreeability as follows.

Observation 3 *Given that ρ_d satisfies all the pairs in the set $\{\langle v_1, \hat{v}_1\rangle, \ldots, \langle v_n, \hat{v}_n\rangle\}$. Let $\rho = \perp[v_1/x_1, \ldots, v_n/x_n]$, and $\hat{\rho} = \perp[\hat{v}_1/x_1, \ldots, \hat{v}_n/x_n]$. Then, for any expression e in P with $FV(e) = \{x_1, \ldots, x_n\}$, we must have ρ, $\hat{\rho}$ and ρ_d agree on e.*

Notice that ρ and $\hat{\rho}$ as defined in Observation 3 represent how all the variable environments used in standard and partial evaluation semantics are constructed. Therefore, the result of Lemma 5 can be expressed in terms of an arbitrary expression in program P as follows.

Corollary 1 *Given a program P. Let ϕ and $\hat{\phi}$ be the two function environments for P defined by the standard and the partial evaluation semantics respectively. Let ϕ' be the function environment, defined by the standard semantics, for a a specialized version of program P. Then, for any expression e in P, $\forall \rho, \hat{\rho} \in VarEnv$ and $\rho_d \in \widehat{VarEnv}$ that agree on e at ϕ', we have*

$$\mathcal{E}[\![e]\!](\rho, \phi) =_\perp \mathcal{E}\,[\![(\hat{\mathcal{E}}[\![e]\!](\hat{\rho}, \hat{\phi}))\downarrow 1]\!](\rho_d, \phi').$$

Correctness of the local partial evaluation semantics can be stated as follows:

Theorem 5 (Correctness of Local Partial Evaluation Semantics) *Given a program P. Let ϕ and $\hat{\phi}$ be the two function environments for P defined by the standard and the partial evaluation semantics respectively. Let ϕ' be the function environment, defined by the standard semantics, for a specialized version of program P. Then, for any expression e in P, $\forall\ \rho, \hat{\rho} \in VarEnv$ and $\rho_d \in \widehat{VarEnv}$ that agree on e at ϕ', we have*

$$\mathcal{E}[\![e]\!](\rho, \phi) =_\perp \mathcal{E}\,[\![(\hat{\mathcal{E}}[\![e]\!](\hat{\rho}, \hat{\phi}))\downarrow 1]\!](\rho_d, \phi')$$

and

$$\mathcal{E}[\![e]\!](\rho, \phi) \sqsubseteq_{\hat{\tau}} (\hat{\mathcal{E}}[\![e]\!](\hat{\rho}, \hat{\phi}))\downarrow 2.$$

Proof : From Theorem 3 and Corollary 1. □

5 Off-Line Partial Evaluation Semantics

Off-line partial evaluation consists of two phases: *binding-time analysis* and *specialization*. In this section, we provide an interpretation of the core semantics (Section 2) that defines binding-time analysis. We then use the technique of logical relation to define and prove the correctness of binding-time analysis. This formally demonstrates the intuition that binding-time analysis is an abstraction of on-line partial evaluation. Finally, we describe a systematic way of deriving a specializer from on-line partial evaluation using the result of binding-time analysis, and we list some optimizations that can be performed to improve the efficiency of the specializer.

5.1 Binding-time Algebra

Just as the partial-evaluation behavior of the primitives is captured by the partial-evaluation algebra, the binding-time behavior of the primitives can similarly be captured by the notion of *binding-time algebra*. The binding-time algebra defines primitive operations over the binding-time domain $\widetilde{\mathbf{Values}}$ (defined in Section 4.1). Formally,

Definition 6 (Binding-Time Algebra) *The binding-time algebra* $[\widetilde{\mathbf{Values}}; \tilde{\mathbf{O}}]$ *is an abstraction of a partial-evaluation algebra* $[\widehat{\mathbf{Values}}; \hat{\mathbf{O}}]$*; it consists of the following components:*

1. *The domain* $\widehat{\mathbf{Values}}$ *and the binding-time domain* $\widetilde{\mathbf{Values}}$ *are related by the abstraction function* $\tilde{\tau}$ *defined in Section 4.1.*
2. $\forall\, \hat{p} \in \hat{\mathbf{O}}$ *of arity* n*, there exists a corresponding abstract version* $\tilde{p} \in \tilde{\mathbf{O}}$ *such that*
   $\tilde{p} : \widetilde{\mathbf{Values}}^n \rightarrow \widetilde{\mathbf{Values}}$
   $\tilde{p} = \lambda\, (\tilde{d}_1, \cdots, \tilde{d}_n)\ .\ \exists j \in \{1, \ldots, n\}$ *s.t.* $\tilde{d}_j = \bot_{\widetilde{Values}} \rightarrow \bot_{\widetilde{Values}}$,
   $\bigwedge_{i=1}^{n} (\tilde{d}_i = \mathit{Static}) \rightarrow \mathit{Static},\ \mathit{Dynamic}$

Like the partial-evaluation algebra, we notice that the abstract primitives defined in $\tilde{\mathbf{O}}$ satisfies the following safety criterion:

$\forall \hat{v} \in \widehat{\mathbf{Values}}$, $\forall \hat{p} \in \hat{\mathbf{O}}$ and its corresponding abstract version $\tilde{p} \in \tilde{\mathbf{O}}$,

$$\tilde{\tau} \circ \hat{p}\ (\hat{v}) \ \sqsubseteq_{\widetilde{Values}}\ \tilde{p} \circ \tilde{\tau}\ (\hat{v})$$

The relation between partial-evaluation algebra and binding-time algebra can also be succintly described by a logical relation $\sqsubseteq_{\tilde{\tau}}$. The definition is similar to that defined in Section 4.1, and is omitted here.

5.2 Specification of Binding-time Analysis

Figure 9 displays the binding-time analysis for our language. The analysis aims at collecting binding-time information for each function in a given program; this forms the *binding-time signature*

of the function. More precisely, a binding-time signature in domain **Sig** is created when a function call is analyzed by the binding-time analysis. It consists of two components: A transformation tag similar to that used in the partial-evaluation signature, and the argument values of the application in $\widetilde{\mathbf{Values}}^n$.

The valuation function $\widetilde{\mathcal{E}}$ is used to define abstract version of each user-defined function. The resulting abstract functions are then used by the valuation function $\widetilde{\mathcal{A}}$ to compute the binding-time signatures. These signatures are recorded in a cache (from domain $\mathbf{Cache}_{\widetilde{\mathcal{A}}}$). As usual, computation is accomplished via fixpoint iteration. Functions $\widetilde{\mathcal{K}}$ and $\widetilde{\mathcal{K}}_P$ perform the abstract computation on constants and primitive operators respectively.

The analysis is *monovariant*: each user-defined function is associated with *one* binding-time signature. Various binding-time signatures associated with a function at different call sites are folded into one signature using the l.u.b. operation. This operation is defined as

$$\begin{array}{l}
\forall \tilde{\sigma}_1, \tilde{\sigma}_2 \in \mathbf{Cache}_{\widetilde{\mathcal{A}}},\ \tilde{\sigma}_1 \sqcup \tilde{\sigma}_2 \ = \ \bot[\langle t, \tilde{\delta}_1, \ldots, \tilde{\delta}_n\rangle / f \mid \forall [\![f]\!] \in Dom(\tilde{\sigma}_1) \cup Dom(\tilde{\sigma}_2)] \\
\quad \textit{where}\ \langle t, \tilde{\delta}_1, \ldots, \tilde{\delta}_n\rangle \ = \ ([\![f]\!] \in (Dom(\tilde{\sigma}_1) \cap Dom(\tilde{\sigma}_2))) \ \rightarrow\ \langle t' \sqcup t'',\ \tilde{\delta}'_1 \sqcup \tilde{\delta}''_1, \ldots, \tilde{\delta}'_n \sqcup \tilde{\delta}''_n\rangle, \\
\qquad\qquad [\![f]\!] \in Dom(\tilde{\sigma}_1) \ \rightarrow\ \tilde{\sigma}_1[\![f]\!],\ \tilde{\sigma}_2[\![f]\!] \\
\qquad \langle t', \tilde{\delta}'_1, \ldots, \tilde{\delta}'_n\rangle \ = \ \tilde{\sigma}_1[\![f]\!] \\
\qquad \langle t'', \tilde{\delta}''_1, \ldots, \tilde{\delta}''_n\rangle \ = \ \tilde{\sigma}_2[\![f]\!]
\end{array}$$

5.3 Correctness of Binding-time Analysis

The initial input to the binding-time analysis is an abstraction of the initial input of on-line partial evaluation. The analysis is correct if its final cache ($\mathbf{Cache}_{\widetilde{\mathcal{A}}}$) contains the abstraction to all the partial-evaluation signatures of the on-line partial evaluation. The correctness is shown by relating the local and global semantics to their respective counterpart in the on-line partial evaluation semantics. That is, we define a logical relation $\mathcal{R}^{\widetilde{\mathcal{E}}}$ that relates $\widehat{\mathcal{E}}$ and $\widetilde{\mathcal{E}}$, and a logical relation $\mathcal{R}^{\widetilde{\mathcal{A}}}$ that relates $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$. We first show the correctness of the local semantics defined by $\widetilde{\mathcal{E}}$, and then that of the global semantics defined by $\widetilde{\mathcal{A}}$.

5.3.1 Correctness of $\widetilde{\mathcal{E}}$

To relate $\widetilde{\mathcal{E}}$ and $\widehat{\mathcal{E}}$, it is sufficient to relate the binding-time values produced at analysis to the partial-evaluation values produced at on-line. This relation forms the basis of the logical relation, as defined below.

Definition 7 (Relation $\mathcal{R}^{\widetilde{\mathcal{E}}}$) *$\mathcal{R}^{\widetilde{\mathcal{E}}}$ is a logical relation between domains of $\widehat{\mathcal{E}}$ and $\widetilde{\mathcal{E}}$ defined by:*

$$\begin{array}{l}
\hat{v}\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{Result_{\widetilde{\varepsilon}}}\ \tilde{\delta} \ \Leftrightarrow\ \hat{v}{\downarrow}2 \sqsubseteq_{\widetilde{\tau}} \tilde{\delta} \\
\hat{\rho}\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{VarEnv}\ \tilde{\rho} \ \Leftrightarrow\ \forall [\![x]\!] \in \mathbf{Var},\ \hat{\rho}[\![x]\!]\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{Result_{\widetilde{\varepsilon}}}\ \tilde{\rho}[\![x]\!] \\
\hat{\phi}\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{FunEnv}\ \tilde{\phi} \ \Leftrightarrow\ \forall [\![f]\!] \in \mathbf{Fn}, \forall i \in \{1, \ldots, n\}, \forall \hat{v}_i \in \mathbf{Res}, \forall \tilde{\delta}_i \in \widetilde{\mathbf{Values}}, \\
\qquad\qquad \bigwedge_{i=1}^{n} (\hat{v}_i\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{Result_{\widetilde{\varepsilon}}}\ \tilde{\delta}_i) \ \Rightarrow\ \hat{\phi}[\![f]\!](\hat{v}_1, \ldots, \hat{v}_n)\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{Result_{\widetilde{\varepsilon}}}\ \tilde{\phi}[\![f]\!](\tilde{\delta}_1, \ldots, \tilde{\delta}_n) \\
\langle \hat{d}_1, \hat{d}_2\rangle\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{D_1 \times D_2}\ \langle \tilde{d}_1, \tilde{d}_2\rangle \ \Leftrightarrow\ \hat{d}_1\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{D_1}\ \tilde{d}_1 \wedge \hat{d}_2\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{D_2}\ \tilde{d}_2 \\
\hat{f}\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{D_1 \rightarrow D_2}\ \tilde{f} \ \Leftrightarrow\ \forall \hat{d} \in \hat{D}_1, \forall \tilde{d} \in \tilde{D}_1,\ \hat{d}\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{D_1}\ \tilde{d} \ \Rightarrow\ \hat{f}(\hat{d})\ \mathcal{R}^{\widetilde{\mathcal{E}}}_{D_2}\ \tilde{f}(\tilde{d}).
\end{array}$$

- Semantic Domains

$\tilde{\delta} \in \mathit{Result}_{\tilde{\mathcal{E}}} = \widetilde{\mathbf{Values}}$

$\tilde{\rho} \in \widetilde{\mathit{VarEnv}} = \mathbf{Var} \to \widetilde{\mathbf{Values}}$

$\tilde{\phi} \in \widetilde{\mathit{FunEnv}} = \mathbf{FEnv} = \mathbf{Fn} \to \widetilde{\mathbf{Values}}^{n} \to \widetilde{\mathbf{Values}}$

$\widetilde{\mathit{Env}} = \widetilde{\mathit{VarEnv}} \times \widetilde{\mathit{FunEnv}}$

$\tilde{s} \in \mathbf{Sig} = (\mathbf{Transf} \times \widetilde{\mathbf{Values}}^{n})$

$\tilde{\sigma} \in \mathit{Result}_{\tilde{\mathcal{A}}} = \mathbf{Cache}_{\tilde{\mathcal{A}}} = \mathbf{Fn} \to \mathbf{Sig}$

- Valuation Functions

$\tilde{\mathcal{E}}_{\mathit{Prog}} : \mathbf{Program} \to \widetilde{\mathbf{Values}}^{n} \to \mathbf{Cache}_{\tilde{\mathcal{A}}}$

$\tilde{\mathcal{E}}_{\mathit{Prog}}\,[\![\{f_i(x_1,\cdots,x_n) = e_i\}]\!]\,\langle\tilde{\delta}_1,\cdots,\tilde{\delta}_n\rangle = \tilde{h}(\bot[\langle \mathbf{s},\tilde{\delta}_1,\cdots,\tilde{\delta}_n\rangle/f_1])$

$\quad \mathit{whererec}\quad \tilde{h}(\tilde{\sigma}) = \tilde{\sigma} \sqcup \tilde{h}(\bigsqcup\{\tilde{\mathcal{A}}\,[\![e_i]\!]\,(\bot[\tilde{\delta}_k/x_k],\tilde{\phi}) \mid \langle -,\tilde{\delta}_1,\ldots,\tilde{\delta}_n\rangle = \tilde{\sigma}[\![f_i]\!],\ \forall [\![f_i]\!] \in \mathit{Dom}(\tilde{\sigma})\})$

$\qquad\qquad \tilde{\phi} = \bot[\{\lambda(\tilde{\delta}_1,\ldots,\tilde{\delta}_n)\,.\,\tilde{\mathcal{E}}\,[\![e_i]\!](\bot[\tilde{\delta}_k/x_k],\tilde{\phi})\}/f_i]$

$\widetilde{\mathcal{E}} = \overline{\mathcal{E}}$

$\widetilde{\mathcal{A}} = \overline{\mathcal{A}}$

- Combinator Definitions

$\mathit{Const}_{\tilde{\mathcal{E}}}\,[\![c]\!] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,\widetilde{\mathcal{K}}\,[\![c]\!]$

$\mathit{VarLookup}_{\tilde{\mathcal{E}}}\,[\![x]\!] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,\tilde{\rho}[\![x]\!]$

$\mathit{PrimOp}_{\tilde{\mathcal{E}}}\,[\![p]\!]\,(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\,.\,\widetilde{\mathcal{K}_P}[\![p]\!]\,(\tilde{k}_1(\tilde{\rho},\tilde{\phi}),\ldots,\tilde{k}_n(\tilde{\rho},\tilde{\phi}))$

$\mathit{Cond}_{\tilde{\mathcal{E}}}\,(\tilde{k}_1,\tilde{k}_2,\tilde{k}_3) = \lambda(\tilde{\rho},\tilde{\phi})\,.\ \tilde{\delta}_1 = \bot_{\widetilde{\mathit{Values}}} \to \bot_{\widetilde{\mathit{Values}}},\ \tilde{\delta}_1 = \mathit{Static} \to \tilde{\delta}_2 \sqcup \tilde{\delta}_3,\ \top_{\widetilde{\mathit{Values}}}$

$\qquad \mathit{where}\ \tilde{\delta}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi})\quad \forall i \in \{1,2,3\}$

$\mathit{App}_{\tilde{\mathcal{E}}}\,[\![f]\!]\,(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(\mathit{Ft}[\![f]\!])\!\downarrow\!1\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n) = \mathrm{u} \to \tilde{\phi}\,[\![f]\!]\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n),\ \top_{\widetilde{\mathit{Values}}}$

$\qquad \mathit{where}\ \tilde{\delta}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi})\quad \forall i \in \{1,\ldots,n\}$

$\mathit{Const}_{\tilde{\mathcal{A}}}\,[\![c]\!] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(\lambda f\,.\,\bot_{\mathit{Sig}})$

$\mathit{VarLookup}_{\tilde{\mathcal{A}}}\,[\![x]\!] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(\lambda f\,.\,\bot_{\mathit{Sig}})$

$\mathit{PrimOp}_{\tilde{\mathcal{A}}}\,[\![p]\!]\,(\tilde{a}_1,\ldots,\tilde{a}_n) = \lambda(\tilde{\rho},\tilde{\phi})\,.\,\bigsqcup_{i=1}^{n} \tilde{a}_i(\tilde{\rho},\tilde{\phi})$

$\mathit{Cond}_{\tilde{\mathcal{A}}}\,(\tilde{a}_1,\tilde{a}_2,\tilde{a}_3)\,\tilde{k}_1 = \lambda(\tilde{\rho},\tilde{\phi})\,.\ \tilde{a}_1(\tilde{\rho},\tilde{\phi}) \sqcup \tilde{a}_2(\tilde{\rho},\tilde{\phi}) \sqcup \tilde{a}_3(\tilde{\rho},\tilde{\phi})$

$\mathit{App}_{\tilde{\mathcal{A}}}\,[\![f]\!]\,(\tilde{a}_1,\ldots,\tilde{a}_n)\,(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(\bigsqcup_{i=1}^{n} \tilde{a}_i(\tilde{\rho},\tilde{\phi})) \sqcup \tilde{\sigma}'$

$\qquad \mathit{where}\ \tilde{\sigma}' = (\mathit{Ft}[\![f]\!])\!\downarrow\!1\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n) = \mathrm{u}$

$\qquad\qquad \to \bot[\langle \mathrm{u},\tilde{\delta}_1,\ldots,\tilde{\delta}_n\rangle/f],\ \bot[\langle \mathbf{s},\tilde{\delta}'_1,\ldots,\tilde{\delta}'_n\rangle/f]$

$\qquad\qquad \langle\tilde{\delta}'_1,\ldots,\tilde{\delta}'_n\rangle = (\mathit{Ft}[\![f]\!])\!\downarrow\!2\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n)$

$\qquad\qquad \tilde{\delta}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi})\quad \forall i \in \{1,\ldots,n\}$

- Primitive Functions

$\widetilde{\mathcal{K}} : \mathbf{Const} \to \widetilde{\mathbf{Values}}$

$\widetilde{\mathcal{K}}\,[\![c]\!] = \tilde{\tau}([\![c]\!])$

$\widetilde{\mathcal{K}_P} : \mathbf{Po} \to \widetilde{\mathbf{Values}}^{n} \to \widetilde{\mathbf{Values}}$

$\widetilde{\mathcal{K}_P}\,[\![p]\!]\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n) = \tilde{p}(\tilde{\delta}_1,\ldots,\tilde{\delta}_n)$

Figure 9: Binding-time Analysis

Lemma 6 *Given a program P. Let $\hat{\phi}$ and $\tilde{\phi}$ be the two function environments for P defined by the partial evaluation semantics and the binding-time analysis respectively. Then $\hat{\phi}\ \mathcal{R}^{\tilde{\mathcal{E}}}\ \tilde{\phi}$.*

Proof : The proof is similar to the proof for Lemma 3, and is thus omitted. □

Theorem 6 (Correctness of Local Binding-Time Analysis) $\hat{\mathcal{E}}\ \mathcal{R}^{\tilde{\mathcal{E}}}\ \tilde{\mathcal{E}}$.

Proof : From Lemma 6. □

Corollary 2 *Given a program P. For any expression e in P, and $\forall \hat{\rho} \in \widehat{VarEnv}$,*

$$(\tilde{\mathcal{E}}\ [\![e]\!](\tilde{\rho},\tilde{\phi}))\downarrow 1 \ = \ Static \ \Rightarrow \ (\hat{\mathcal{E}}\ [\![e]\!](\hat{\rho},\hat{\phi}))\downarrow 1 \in \mathbf{Const} \cup \{\perp_{Exp}\}$$

where both $\hat{\phi} \in \widehat{FunEnv}$ and $\tilde{\phi} \in \widetilde{FunEnv}$ are fixed for the program, and $\tilde{\rho} \in \widetilde{VarEnv}$ is defined such that $\hat{\rho}\ \mathcal{R}^{\tilde{\mathcal{E}}}\ \tilde{\rho}$.

5.3.2 Correctness of the Global Analysis

We prove the correctness of the global analysis (1) by relating the semantics of $\tilde{\mathcal{A}}$ with that of $\hat{\mathcal{A}}$ using the logical relation $\mathcal{R}^{\tilde{\mathcal{A}}}$, and (2) by showing that all the non-trivial calls that are recorded by $\hat{\mathcal{A}}$ are captured in the cache produced by $\tilde{\mathcal{A}}$.

Definition 8 (Relation $\mathcal{R}^{\tilde{\mathcal{A}}}$) *$\mathcal{R}^{\tilde{\mathcal{A}}}$ is a logical relation between the domains of $\hat{\mathcal{A}}$ and $\tilde{\mathcal{A}}$ defined by extending relation $\mathcal{R}^{\tilde{\mathcal{E}}}$ to include the relation between $\hat{\sigma}$ and $\tilde{\sigma}$ produced by $\hat{\mathcal{A}}$ and $\tilde{\mathcal{A}}$ respectively:*

$$\langle \hat{t}, \hat{v}_1, \ldots, \hat{v}_n \rangle\ \mathcal{R}^{\tilde{\mathcal{A}}}_{Sig}\ \langle \tilde{t}, \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle \ \Leftrightarrow \ (\hat{t} \sqsubseteq_{Transf} \tilde{t}) \wedge \bigwedge_{i=1}^{n} (\hat{v}_i \mathcal{R}^{\tilde{\mathcal{E}}}_{Result_{\tilde{\mathcal{E}}}} \tilde{\delta}_i)$$

$$\hat{\sigma}\ \mathcal{R}^{\tilde{\mathcal{A}}}_{Result_{\tilde{\mathcal{A}}}}\ \tilde{\sigma} \ \Leftrightarrow \ \forall\ [\![f]\!] \in Dom(\sigma), \forall \hat{s} \in \hat{\sigma}[\![f]\!], \exists \tilde{s} \in \tilde{\sigma}[\![f]\!] \text{ such that } \hat{s}\ \mathcal{R}^{\tilde{\mathcal{A}}}_{Sig}\ \tilde{s}$$

We note that the l.u.b. operations defined on both caches are closed under $\mathcal{R}^{\tilde{\mathcal{A}}}_{Result_{\tilde{\mathcal{A}}}}$. With this relation, the next lemma shows that all the partial-evaluation signatures recorded in the final cache produced by $\hat{\mathcal{A}}$ are captured in the corresponding cache produced by $\tilde{\mathcal{A}}$ in the sense that they are related by $\mathcal{R}^{\tilde{\mathcal{A}}}$.

Lemma 7 *Given a program P. Let $\hat{\phi}$ and $\tilde{\phi}$ be two function environments for P defined by the partial evaluation and the binding-time analysis respectively. For any expression e in P, for any $\hat{\rho}$, $\tilde{\rho}$ such that $\hat{\rho}\mathcal{R}^{\tilde{\mathcal{A}}}\tilde{\rho}$,*

$$\hat{\mathcal{A}}\ [\![e]\!](\hat{\rho},\hat{\phi}) \ \ \mathcal{R}^{\tilde{\mathcal{A}}} \ \ \tilde{\mathcal{A}}\ [\![e]\!](\tilde{\rho},\tilde{\phi}).$$

Proof : The proof is by structural induction over an expression. Firstly, notice that $\hat{\phi}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\phi}$. It then suffices to show that $\mathcal{R}^{\widetilde{\mathcal{A}}}$ holds for all the corresponding pairs of combinators used by $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ respectively. By structural induction, it is easy to see that $\mathcal{R}^{\widetilde{\mathcal{A}}}$ holds for constant, variable and primitive calls. We show below that $\mathcal{R}^{\widetilde{\mathcal{A}}}$ holds for the case of conditional expressions and function applications.

1. $Cond_{\widehat{\mathcal{A}}}$: By the structural induction hypothesis, all the corresponding pairs of arguments are related by $\mathcal{R}^{\widetilde{\mathcal{A}}}$. Since the result of $Cond_{\widetilde{\mathcal{A}}}$ is the l.u.b. of the caches produced at all the arguments, whereas the result of $Cond_{\widehat{\mathcal{A}}}$ is the l.u.b. of the caches produced at some of the arguments, $\mathcal{R}^{\widetilde{\mathcal{A}}}$ must hold.

2. $App_{\widehat{\mathcal{A}}}$: By the structural induction hypothesis, $\mathcal{R}^{\widetilde{\mathcal{A}}}$ holds for all the arguments to the application.
   Let $\hat{\sigma} = \bot[\{\langle \hat{t}, \hat{v}_1'', \ldots, \hat{v}_n'' \rangle\}/f]$ and $\tilde{\sigma} = \bot[\langle \tilde{t}, \tilde{\delta}_1'', \ldots, \tilde{\delta}_n'' \rangle/f]$, we need to show that $\hat{\sigma}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\sigma}$.
   We consider the cases with different transformation values produced at the binding-time analysis level.

   - If $\tilde{t} = \mathbf{u}$, then $\hat{t} = \mathbf{u}$ by the monotonicity of filters. Thus, $\forall\ i \in \{1, \ldots, n\}$,

     $$\begin{array}{rcll} \hat{v}_i'' & = & \hat{a}_i(\hat{\rho}, \hat{\phi}_n) & \text{[by definition]} \\ & \mathcal{R}^{\widetilde{\mathcal{A}}} & \tilde{a}_i(\tilde{\rho}, \tilde{\phi}_n) & \text{[structural induction hypothesis]} \\ & = & \tilde{\delta}_i''. & \text{[by definition]} \end{array}$$

     Therefore, $\hat{\sigma}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\sigma}$.

   - If $\tilde{t} = \mathbf{s}$, then $\hat{t} \sqsubseteq_{Transf} \tilde{t}$ by monotonicity of the filter.
     Let $\langle \hat{v}_1, \ldots, \hat{v}_n \rangle$ and $\langle \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle$ be the initial arguments computed for the application. By the monotonicity of filter,

     $$\hat{b}_i \ \sqsubseteq\ \tilde{\delta}_i' \quad \forall\ i \in \{1, \ldots, n\}$$

     where $\langle \hat{b}_1, \ldots, \hat{b}_n \rangle = (Ft[\![f]\!])\downarrow 2\ (\widehat{bt}(\hat{v}_1), \ldots, \widehat{bt}(\hat{v}_n))$ *and*
     $\langle \tilde{\delta}_1', \ldots, \tilde{\delta}_n' \rangle = (Ft[\![f]\!])\downarrow 2\ (\tilde{\delta}_1, \ldots, \tilde{\delta}_n)$
     Let $\langle \hat{v}_1'', \ldots, \hat{v}_n'' \rangle = SpPat\ ([\![f]\!], \langle \hat{v}_1, \ldots, \hat{v}_n \rangle, \langle \hat{b}_1, \ldots, \hat{b}_n \rangle)$. From the definition of $SpPat$, we have

     $$SpPat\ ([\![f]\!], \langle \hat{v}_1, \ldots, \hat{v}_n \rangle, \langle \hat{b}_1, \ldots, \hat{b}_n \rangle)\ \ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \ \langle \tilde{\delta}_1', \ldots, \tilde{\delta}_n' \rangle$$

     Since $\langle \tilde{t}, \tilde{\delta}_1', \ldots, \tilde{\delta}_n' \rangle$ is the binding-time signature produced for the application, we therefore have $\hat{\sigma}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\sigma}$.

Thus, $(\bigsqcup_{i=1}^{n} \hat{a}_i(\hat{\rho}, \hat{\phi}_n) \sqcup \hat{\sigma})\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ (\bigsqcup_{i=1}^{n} \tilde{a}_i(\tilde{\rho}, \tilde{\phi}_n) \sqcup \tilde{\sigma})$. Hence, $App_{\widehat{\mathcal{A}}}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ App_{\widetilde{\mathcal{A}}}$.

Hence, $\mathcal{R}^{\widetilde{\mathcal{A}}}$ holds in general. This concludes the proof. □

Theorem 7 (Correctness of Global Binding-Time Analysis) *Given a program P. Let $\langle \hat{v}_1, \ldots, \hat{v}_n \rangle$ and $\langle \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle$ be initial inputs to P for on-line partial evaluation and binding-time analysis respectively, such that $\hat{v}_i\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\delta}_i$, $\forall i \in \{1, \ldots, n\}$. If $\hat{\sigma}$ and $\tilde{\sigma}$ are the final caches produced by $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ respectively, then $\hat{\sigma}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\sigma}$.*

Proof: Firstly, we notice from the definition of $\widetilde{\mathcal{E}}_{Prog}$ that $\langle \mathbf{s}, \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle$ is the corresponding binding-time signature for f_1 in $\tilde{\sigma}$. Therefore, $\langle \mathbf{s}, \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle \sqsubseteq \tilde{\sigma}[\![f_1]\!]$. This captures the initial call to the on-line partial evaluation: $\langle \mathbf{s}, \hat{v}_1, \ldots, \hat{v}_n \rangle \in \hat{\sigma}[\![f_1]\!]$. Next $\tilde{h}$ in $\widetilde{\mathcal{E}}_{Prog}$ applies $\widetilde{\mathcal{A}}$ to each binding-time signature in the cache, like function $\hat{h}$ in $\widehat{\mathcal{E}}_{Prog}$. Since l.u.b. operation is closed under $\mathcal{R}^{\widetilde{\mathcal{A}}}$, $\hat{\sigma}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\sigma}$. □

5.4 Deriving the Specialization Semantics

We now describe the derivation of the specialization semantics (for off-line partial evaluation) from its on-line counterpart. This derivation is based on the observation that, prior to on-line partial evaluation, the binding-time analysis has determined the invariants of this process. Indeed, the result of the on-line partial-evaluation computations has been approximated and is available statically. Thus, the aim of this derivation is to transform the on-line partial evaluation semantics so that it makes use of binding-time information as much as possible. The uses of binding-time information are listed below.

1. Predicates testing whether an expression partially evaluates to a constant can safely be replaced by a predicate testing whether this expression returns *Static* at binding-time analysis.
2. Filter computation for a function call can safely be replaced by an access to the function's binding-time signature; it contains the call transformation to be performed.

The use of binding-time information collected for an expression requires that this information be bound to the expression. That is, each expression in a program should be annotated with the information computed by the binding-time analysis. We achieve this annotation by assigning a unique label to each expression in a program and binding this label to the corresponding binding-time information. A cache, noted $\tilde{\psi}$, maps each label of an expression to its binding-time information. For a label l, we write $(\tilde{\psi}\ l)_v$ to denote the binding-time value corresponding to l. If l is the label of a function call, then $(\tilde{\psi}\ l)_t$ refers to its transformation (*i.e.*, unfolding or suspension).

5.4.1 Specification of the Specializer

Note that this annotation strategy only requires a minor change to the core semantics. Namely, the labels of an expression must be passed to the semantic combinator.[5] For example, in specializing a labeled conditional expression $[\![(if\ e_1^{l_1}\ e_2^{l_2}\ e_3^{l_3})^l]\!]$, the combinator $Cond_{\widehat{\mathcal{E}}}$ takes as an additional argument $\langle l, l_1, l_2, l_3\rangle$. Besides passing labels to combinators, we extend the usual pair of environments to include the cache (*i.e.*, $\tilde{\psi} \in$ **AtCache**).

Figures 10 and 11 depict the detailed specification of the specialization process. Each interpreted combinator is similar to that of on-line partial evaluation, except in the following cases:

1. For both $Cond_{\widehat{\mathcal{E}}}$ and $Cond_{\widehat{\mathcal{A}}}$, the predicate that determines whether the conditional test evaluates to a constant has been replaced by a predicate that tests the staticity of its binding-time value.
2. For primitive call, the predicate testing whether the result of the operation is a constant has been replaced by a predicate testing the staticity of the resulting binding-time value.
3. For both $App_{\widehat{\mathcal{E}}}$ and $App_{\widehat{\mathcal{A}}}$, filter computation has been replaced by an access to the static information about the function call: binding-time value of the arguments and function call transformation.

[5] Note that for simplicity we did not introduce labels in the core semantics presented on page 6. Indeed, labels are only used for the specialization semantics.

- Semantic Domains

$l \in$ **Labels**

$\widehat{\delta} \in \widehat{\textbf{Values}}$ = as in On−Line Sem.

$\langle \tilde{t}, \tilde{\delta} \rangle \in$ **Att** = (**Transf** × $\widetilde{\textbf{Values}}$)

$\tilde{\psi} \in$ **AtCache** = **Labels** → **Att**

$\hat{\rho} \in \widehat{VarEnv}$ = as in On−Line Sem.

$\hat{v} \in \widehat{Result_{\mathcal{E}}}$ = **Res** = as in On−Line Sem.

$\tilde{\delta} \in \widetilde{\textbf{Values}}$ = as in Off−Line Sem.

$\hat{\sigma} \in \widehat{Result_{\mathcal{A}}}$ = as in On−Line Sem.

$\hat{\phi} \in \widehat{FunEnv}$ = as in On−Line Sem.

$\widehat{Env} = \widehat{VarEnv} \times \widehat{FunEnv} \times$ **AtCache**

- Valuation Functions

$\widehat{\mathcal{E}_{Prog}}$: **Prog** → **Res**^{n} → **AtCache** → **Prog**$_\perp$

$\widehat{\mathcal{E}_{Prog}}\ [\![\{f_i(x_1,\cdots,x_n) = e_i\}]\!]\ \langle \hat{v}_1,\ldots,\hat{v}_n\rangle\ \tilde{\psi} = MkProg\ (\hat{h}(\perp[\{\langle \mathbf{s}, \hat{v}_1,\ldots,\hat{v}_n\rangle\}/f_1]))\tilde{\psi}\hat{\phi}$

$\quad$ *whererec* $\hat{h}(\hat{\sigma}) = \hat{\sigma} \sqcup \hat{h}(\bigsqcup\{\mathcal{A}_{\widetilde{SA}}\ [\![e_i]\!](\perp[\hat{v}'_k/x_k], \hat{\phi}, \tilde{\psi}) \mid \langle -, \hat{v}'_1,\ldots,\hat{v}'_n\rangle \in \hat{\sigma}[\![f_i]\!],\ \forall [\![f_i]\!] \in Dom(\hat{\sigma})\})$

$\qquad\qquad \hat{\phi} = \perp[strict\ \{\lambda(\hat{v}_1,\cdots,\hat{v}_n)\ .\ \mathcal{E}_{\widetilde{SE}}\ [\![e_i]\!](\perp[\hat{v}_k/x_k], \hat{\phi}, \tilde{\psi})\}/f_i]$

- *MkProg* Definition

$MkProg\ \hat{\sigma}\ \tilde{\psi}\ \hat{\phi} = \{\ f_i^{sp}(x_1,\ldots,x_k) = \hat{v}\downarrow 1 \mid \forall \langle \mathbf{s}, \hat{v}_1,\ldots,\hat{v}_n\rangle \in \hat{\sigma}[\![f_i]\!],\ \forall [\![f_i]\!] \in Dom(\hat{\sigma})\}$

$\quad$ *where* $f_i^{sp} = SpName([\![f_i]\!], \hat{v}_1,\ldots,\hat{v}_n)$

$\qquad \hat{v}' = \mathcal{E}_{\widetilde{SE}}\ [\![e_i]\!](\perp[\hat{v}_k/x_k], \hat{\phi}, \tilde{\psi})$

$\qquad \langle x_1,\ldots,x_k\rangle = ResidPars\ ([\![f_i]\!], \hat{v}_1\downarrow 1,\ldots,\hat{v}_n\downarrow 1)$

- Local Combinator Definitions

$Const_{\widetilde{SE}}\ [\![c]\!]\ \langle l\rangle = \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ \hat{\mathcal{K}}\ [\![c]\!]$

$Var_{\widetilde{SE}}\ [\![x]\!]\ \langle l\rangle = \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ \hat{\rho}\ [\![x]\!]$

$PrimOp_{\widetilde{SE}}\ [\![p]\!]\ (\hat{k}_1,\ldots,\hat{k}_n)\ \tilde{\delta}\ \langle l, l_1,\ldots,l_n\rangle = \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ \hat{\mathcal{K}}_{SP}\ [\![p]\!]\ (\hat{k}_1(\hat{\rho},\hat{\phi},\tilde{\psi}),\ldots,\ \hat{k}_n(\hat{\rho},\hat{\phi},\tilde{\psi}))\ (\tilde{\psi}\ l)_v$

$Cond_{\widetilde{SE}}\ (\hat{k}_1,\hat{k}_2,\hat{k}_3)\ \langle l, l_1, l_2, l_3\rangle =$

$\lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ (\tilde{\psi}\ l_1)_v = Static \rightarrow\ (\mathcal{K}(\hat{v}_1\downarrow 1) \rightarrow \hat{v}_2,\ \hat{v}_3),\ \langle [\![if\ \hat{v}_1\downarrow 1\ \hat{v}_2\downarrow 1\ \hat{v}_3\downarrow 1]\!], \hat{v}_2\downarrow 2 \sqcup \hat{v}_3\downarrow 2\rangle$

$\quad$ *where* $\hat{v}_i = \hat{k}_i(\hat{\rho},\hat{\phi},\tilde{\psi})\quad \forall i \in \{1,\ 2,\ 3\}$

$App_{\widetilde{SE}}\ [\![f]\!]\ (\hat{k}_1,\ldots,\hat{k}_n)\ \langle l, l_1,\ldots,l_n\rangle = \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ (\tilde{\psi}\ l)_t = \mathrm{u} \rightarrow\ \hat{\phi}\ [\![f]\!]\ (\hat{v}_1,\ldots,\hat{v}_n),$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle [\![f_{sp}(e''_1,\ldots,e''_k)]\!], \top_{\widehat{Values}}\rangle$

$\quad$ *where* $\hat{v}_i = \hat{k}_i(\hat{\rho},\hat{\phi},\tilde{\psi})\quad \forall i \in \{1,\ldots,n\}$

$\qquad f_{sp} = SpName([\![f]\!], \hat{v}'_1,\ldots,\hat{v}'_n)$

$\qquad \langle e''_1,\ldots,e''_k\rangle = ResidArgs\ ([\![f]\!],\ \langle \tilde{\delta}'_1,\ldots,\tilde{\delta}'_n\rangle,\ \langle \hat{v}_1\downarrow 1,\ldots,\hat{v}_n\downarrow 1\rangle)$

$\qquad \langle \hat{v}'_1,\ldots,\hat{v}'_n\rangle = SpPat\ ([\![f]\!], \langle \hat{v}_1,\ldots,\hat{v}_n\rangle,\ \langle \tilde{\delta}'_1,\ldots,\tilde{\delta}'_n\rangle)$

$\qquad \tilde{\delta}'_i = (\tilde{\psi}\ l_i)_v\quad \forall i \in \{1,\ldots,n\}$

- Primitive Functions

$\hat{\mathcal{K}}$: **Const** → **Res**

$\hat{\mathcal{K}}\ [\![c]\!]$ = (as in On−Line Semantics)

$\hat{\mathcal{K}}_{SP}$: **Po** → **Res**^{n} → $\widetilde{\textbf{Values}}$ → **Res**

$\hat{\mathcal{K}}_{SP}\ [\![p]\!]\ (\langle \hat{e}'_1, \hat{\delta}_1\rangle,\ldots,\langle \hat{e}'_n, \hat{\delta}_n\rangle)\ \tilde{\delta} =$

$\quad (\tilde{\delta} = \perp_{\widetilde{Values}}) \rightarrow \langle \perp_{Exp}, \perp_{\widehat{Values}}\rangle,\ (\tilde{\delta} = Static) \rightarrow \langle \hat{\delta}, \hat{\delta}\rangle,\ \langle [\![p(\hat{e}'_1,\cdots,\hat{e}'_n)]\!], \hat{\delta}\rangle$

$\quad$ *where* $\hat{\delta} = \hat{p}(\hat{\delta}_1,\cdots,\hat{\delta}_n)$

Figure 10: Specialization Semantics – Part 1

- Global Combinator Definitions

$$Const_{\widetilde{SA}}\ [c]\ \langle l\rangle\ =\ \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ \lambda l\ .\ \perp_{Att}$$

$$Var_{\widetilde{SA}}\ [x]\ \langle l\rangle\ =\ \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ \lambda l\ .\ \perp_{Att}$$

$$PrimOp_{\widetilde{SA}}\ [p]\ (\hat{a}_1,\ldots,\hat{a}_n)\ \langle l,l_1,\ldots,l_n\rangle\ =\ \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ \bigsqcup_{i=1}^{n}\hat{a}_i(\hat{\rho},\hat{\phi},\tilde{\psi})$$

$$Cond_{\widetilde{SA}}\ (\hat{a}_1,\hat{a}_2,\hat{a}_3)\ \hat{k}_1\ \langle l,l_1,l_2,l_3\rangle\ =$$
$$\lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ (\hat{a}_1(\hat{\rho},\hat{\phi},\tilde{\psi}))\sqcup\ ((\tilde{\psi}\ l)_v = Static)\ \rightarrow\ \ \hat{\mathcal{K}}(\hat{\delta}_1)\rightarrow \hat{a}_2(\hat{\rho},\hat{\phi},\tilde{\psi}),\hat{a}_3(\hat{\rho},\hat{\phi},\tilde{\psi}),$$
$$\hat{a}_2(\hat{\rho},\hat{\phi},\tilde{\psi})\sqcup\hat{a}_3(\hat{\rho},\hat{\phi},\tilde{\psi}))$$

$$App_{\widetilde{SA}}\ [f]\ (\hat{a}_1,\ldots,\ \hat{a}_n)\ (\hat{k}_1,\ldots,\ \hat{k}_n)\ \langle l,l_1,\ldots,l_n\rangle\ =\ \lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\ .\ (\bigsqcup_{i=1}^{n}\hat{a}_i(\hat{\rho},\hat{\phi},\tilde{\psi}))\sqcup\hat{\sigma}$$

$$\text{where}\ \hat{v}_i\ =\ \hat{k}_i(\hat{\rho},\hat{\phi},\tilde{\psi})\quad\forall i\in\{1,\cdots,n\}$$
$$\hat{\sigma}\ =\ (\tilde{\psi}\ l)_t = \mathbf{u}\rightarrow\ \perp[\{\langle \mathbf{u},\hat{v}_1,\ldots,\hat{v}_n\rangle\}/f],\ \perp[\{\langle \mathbf{s},\hat{v}'_1,\ldots,\hat{v}'_n\rangle\}/f]$$
$$\langle\hat{v}'_1,\ldots,\hat{v}'_n\rangle\ =\ SpPat\ ([f],\langle\hat{v}_1,\ldots,\hat{v}_n\rangle,\ \langle\hat{\delta}_1,\ldots,\hat{\delta}_n\rangle)$$
$$\hat{\delta}_i\ =\ (\tilde{\psi}\ l_i)_v\quad\forall i\in\{1,\ldots,n\}$$

Figure 11: Specialization Semantics – Part 2

5.4.2 Optimization of Specialization

At this point it is important to determine whether the specialization semantics that we derived indeed describes a specialization process. In fact, as mentioned in [BJMS88, JSS89], binding-time analysis was introduced for practical reasons. Namely, by taking advantage of binding-time information, the partial-evaluation process can be simplified and its efficiency improved. This is a key point for successful self-application [JSS89].

Thus, the off-line strategy aims at lifting as many computations as possible from specialization by exploiting static information. In other terms, there exists a wide range of specializers for a given language; each possible specializer reflects how much has been computed in the preprocessing phase.

As a matter of fact, the specialization semantics derived in the previous section may be used as a basis to introduce many optimizations. In particular, it is possible to infer statically the actions to be performed by the specializer. The basic actions of a specializer consists of reducing or rebuilding an expression. Such actions can be determined using the binding-time value of an expression. This technique has been used in off-line partial evaluation [Con89, CD90].

6 Conclusion

Based on the technique of factorized semantics, we provide semantic specifications and correctness proofs for both on-line and off-line partial evaluation of functional programs. Using the technique of collecting interpretation and the introduction of partial-evaluation algebra, we are able to prove the correctness of polyvariant specialization.

Furthermore, this paper addresses and solves a series of open issues in partial evaluation such as relating on-line partial evaluation to standard semantics, showing that binding-time analysis is

an abstraction of the on-line partial-evaluation process, and formally defining the specialization semantics.

As such, this work should improve the understanding of partial evaluation. Also, it should provide a basis for implementation. In fact, the specifications presented in this paper have been generalized by the authors to the specification of *parameterized partial evaluation* – a generic form of partial evaluation aimed at specializing programs not only with respect to concrete values, but also with respect to static properties [CK91, CK92]. Parameterized partial evaluation has already been successfully implemented at CMU [CL91] and at Yale [Kho92].

Future work includes extending the results of this paper to higher-order programs. Our preliminary studies in this direction indicate that such an extension should be minor since it would be based on an existing framework for abstract interpretation of higher-order programs, such as [HY88, Jon91].

References

[Abr90] A. Abramsky. Abstract interpretation, logical relations and Kan extensions. *Logic and Computation*, 1(1):5–40, 1990.

[BJMS88] A. Bondorf, N. D. Jones, T. Mogensen, and P. Sestoft. Binding time analysis and the taming of self-application. Diku report, University of Copenhagen, Copenhagen, Denmark, 1988.

[CD90] C. Consel and O. Danvy. From interpreting to compiling binding times. In N. D. Jones, editor, *ESOP'90, 3rd European Symposium on Programming*, volume 432 of *Lecture Notes in Computer Science*, pages 88–105. Springer-Verlag, 1990.

[CK91] C. Consel and S. C. Khoo. Parameterized partial evaluation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 92–106, 1991.

[CK92] C. Consel and S.C. Khoo. Parameterized partial evaluation: Semantic specifications and correctness proofs. Research Report 896, Yale University, New Haven, Connecticut,. USA, 1992.

[CL91] C. Colby and P. Lee. An implementation of parameterized partial evaluation. *Bigre Journal*, 74:82–89, 1991.

[Con88] C. Consel. New insights into partial evaluation: the Schism experiment. In H. Ganzinger, editor, *ESOP'88, 2nd European Symposium on Programming*, volume 300 of *Lecture Notes in Computer Science*, pages 236–246. Springer-Verlag, 1988.

[Con89] C. Consel. *Analyse de Programmes, Evaluation Partielle et Génération de Compilateurs.* PhD thesis, Université de Paris VI, Paris, France, June 1989.

[Con90] C. Consel. *The Schism Manual.* Yale University, New Haven, Connecticut, USA, 1990. Version 1.0.

[Fut71] Y. Futamura. Partial evaluation of computation process – an approach to a compiler-compiler. *Systems, Computers, Controls 2, 5*, pages 45–50, 1971.

[GJ85] H. Ganzinger and N. D. Jones, editors. *Programs as Data Objects*, volume 217 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

[Gom92] C. K. Gomard. A self-applicable partial evaluator for the lambda-calculus: Correctness and pragmatics. *ACM Transactions on Programming Languages and Systems*, 14(2):147–172, 1992.

[HM89] J. Hannan and D. Miller. Deriving mixed evaluation from standard evaluation for a simple functional language. Technical Report MS-CIS-89-28, University of Pennsylvania, Philadelphia, Pennsylvania, 1989.

[HY88] P. Hudak and J. Young. A collecting interpretation of expressions (without Powerdomains). In *ACM Symposium on Principles of Programming Languages*, pages 107–118, 1988.

[JM76] N. D. Jones and S. S. Muchnick. Some thoughts towards the design of an ideal language. In *ACM Conference on Principles of Programming Languages*, pages 77–94, 1976.

[JM86] N. D. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs. In *ACM Symposium on Principles of Programming Languages*, 1986.

[JN90] N. D Jones and F. Nielson. Abstract interpretation: a semantics-based tool for program analysis. Technical report, University of Copenhagen and Aarhus University, Copenhagen, Denmark, 1990.

[Jon88a] N. D Jones. Binding time analysis and static semantics (extended abstract). Diku report, University of Copenhagen, Copenhagen, Denmark, 1988.

[Jon88b] N.D. Jones. Automatic program specialization: A re-examination from basic principles. In D. Bjørner, A. P. Ershov, and N. D. Jones, editors, *Partial Evaluation and Mixed Computation*, pages 225–282. North-Holland, 1988.

[Jon90] N. D. Jones. Partial evaluation, self-application and types. In M.S. Paterson, editor, *17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 639–659. Springer-Verlag, 1990.

[Jon91] N. D Jones. A minimal function graph semantics as a basis for abstract interpretation of higher order programs, 1991. Presented at the 1991 Workshop on Static Analysis of Equational, Functional and Logic Programs.

[JSS89] N. D. Jones, P. Sestoft, and H. Søndergaard. Mix: a self-applicable partial evaluator for experiments in compiler generation. *LISP and Symbolic Computation*, 2(1):9–50, 1989.

[Kho92] S. C. Khoo. *Parameterized Partial Evaluation: Principle and Practice.* PhD thesis, Yale University, 1992. Forthcoming.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* Van Nostrand, 1952.

[Lau90] J. Launchbury. *Projection Factorisation in Partial Evaluation.* PhD thesis, Department of Computing Science, University of Glasgow, Scotland, 1990.

[MS90] M. Mizuno and D. Schmidt. A security flow control algorithm and its denotational semantics correctness proof. Technical Report CS-90-21, Kansas State University, Manhattan, Kansas, 1990.

[Nie89] F. Nielson. Two-level semantics and abstract interpretation. *Theoretical Computer Science*, 69:117–242, 1989.

[Ses85] P. Sestoft. The structure of a self-applicable partial evaluator. In [GJ85], pages 236–256, 1985.

[Ses88] P. Sestoft. Automatic call unfolding in a partial evaluator. In D. Bjørner, A. P. Ershov, and N. D. Jones, editors, *Partial Evaluation and Mixed Computation.* North-Holland, 1988.

A Correctness of Instrumentation

Lemma 8 *Given a program P. Let ϕ be the function environment for P defined by the instrumented semantics. If the standard evaluation of P with input $\langle v_1, \ldots, v_n \rangle$ terminates, and σ is the cache computed for P by $\mathcal{A}$, then*

1. *For any expression e in P, if a non-trivial function call occurring in e is performed when e is evaluated, then $\mathcal{A}$ records the call in the cache.*
2. *For any function definition in P of the form*

$$f_i(x_1, \ldots, x_n) = \cdots f_j(e'_1, \ldots, e'_n) \cdots$$

Let $\langle v'_1, \ldots, v'_n \rangle \in \sigma[\![f_i]\!]$. If evaluating f_i with argument $\langle v'_1, \ldots, v'_n \rangle$ results in a call to f_j with $\langle v''_1, \ldots, v''_n \rangle$, where $v''_i = \mathcal{E}[\![e'_i]\!](\perp[v'_k/x_k], \phi)\ \forall i \in \{1, \ldots, n\}$, then $\langle v''_1, \ldots, v''_n \rangle \in \sigma[\![f_j]\!]$, provided $v''_i \neq \perp, \forall i \in \{1, \ldots, n\}$.

Proof (Sketch):

1. We want to show that the predicate "if a non-trivial function call occurring in e is performed when evaluating e, then the call is recorded in the cache produced by $\mathcal{A}$" is true. The proof is done by structural induction over e.
2. The second part of the lemma is shown by examining local function h in function $\mathcal{E}_{Prog}$. If $\langle v'_1, \ldots, v'_n \rangle \in \sigma[\![f_i]\!]$, then $\mathcal{A}$ will be called to collect non-trivial calls in the body of f_i. Using the first result of this lemma we know that $\langle v''_1, \ldots, v''_n \rangle \in \sigma[\![f_j]\!]$, provided $v''_i \neq \perp, \forall i \in \{1, \ldots, n\}$. □

Theorem 1 (Correctness of Instrumented Semantics) *Let P be a program evaluated with input $\langle v_1, \ldots, v_n \rangle$. For any user-defined function f in P, if f is called with non-bottom argument $\langle v'_1, \ldots, v'_n \rangle$ during the standard evaluation, then $\langle v'_1, \ldots, v'_n \rangle \in \sigma[\![f]\!]$.*

Proof : From Lemma 8, and noticing that, since none of the initial input should be bottom, the initial call to f_1 is captured in the cache (by the definition of $\mathcal{E}_{Prog}$) . □

B Partial Evaluation Semantics Subsumes Standard Semantics

Theorem 2 *Given a program P. Suppose that (1) the input to P is completely known at partial-evaluation time, and (2) all function calls in P are unfolded during partial evaluation, then for any expression e in P,*

$$\hat{\tau}(\mathcal{E}\ [\![e]\!](\rho,\phi)) \;=\; (\widehat{\mathcal{E}}\ [\![e]\!](\hat{\rho},\hat{\phi}))\downarrow 1$$

where both $\phi \in FunEnv$ and $\hat{\phi} \in \widehat{FunEnv}$ are fixed for the program, $\rho \in VarEnv$, and $\hat{\rho} \in \widehat{VarEnv}$ is defined as:

$$\hat{\rho} \;=\; \lambda\, [\![x]\!]\ .\ \langle \hat{\tau}(\rho[\![x]\!]), \hat{\tau}(\rho[\![x]\!])\rangle \quad \textit{for } (\rho[\![x]\!]) \in \mathbf{D}.$$

Proof : The proof is by induction on the structure of expression, proving $\mathcal{E}[\![e]\!]\ \widehat{\mathcal{R}}\ \widehat{\mathcal{E}}[\![e]\!]$, for the logical relation $\widehat{\mathcal{R}}$ between domains of $\mathcal{E}$ and $\widehat{\mathcal{E}}$ defined by:

$$\begin{aligned}
&v\ \widehat{\mathcal{R}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v} \;\Leftrightarrow\; \hat{\tau}(v) \;=\; \hat{v}\downarrow 1 \;=\; \hat{v}\downarrow 2 \;\; \textit{where } v \in \mathbf{D}\\
&\rho\ \widehat{\mathcal{R}}_{VarEnv}\ \hat{\rho} \;\Leftrightarrow\; \forall [\![x]\!] \in \mathbf{Var},\ \rho[\![x]\!]\ \widehat{\mathcal{R}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{\rho}[\![x]\!]\\
&\phi\ \widehat{\mathcal{R}}_{FunEnv}\ \hat{\phi} \;\Leftrightarrow\; \forall [\![f]\!] \in \mathbf{Fn}, \forall i \in \{1,\ldots,n\}, \forall v_i \in \mathbf{Values}, \forall \hat{v}_i \in \mathbf{Res},\\
&\qquad\qquad \bigwedge_{i=1}^{n} (v_i\ \widehat{\mathcal{R}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v}_i) \;\Rightarrow\; \phi[\![f]\!](v_1,\ldots,v_n)\ \widehat{\mathcal{R}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{\phi}[\![f]\!](\hat{v}_1,\ldots,\hat{v}_n)\\
&\langle d_1, d_2\rangle\ \widehat{\mathcal{R}}_{D_1\times D_2}\ \langle \hat{d}_1, \hat{d}_2\rangle \;\Leftrightarrow\; d_1\ \widehat{\mathcal{R}}_{D_1}\ \hat{d}_1 \wedge d_2\ \widehat{\mathcal{R}}_{D_2}\ \hat{d}_2\\
&f\ \widehat{\mathcal{R}}_{D_1\to D_2}\ \hat{f} \;\Leftrightarrow\; \forall d \in D_1, \forall \hat{d} \in \hat{D}_1,\ d\ \widehat{\mathcal{R}}_{D_1}\ \hat{d} \;\Rightarrow\; f(d)\ \widehat{\mathcal{R}}_{D_2}\ \hat{f}(\hat{d})
\end{aligned}$$

It suffices to show that $\widehat{\mathcal{R}}$ holds for all the corresponding pair of combinators used by $\mathcal{E}$ and $\widehat{\mathcal{E}}$.

- The proofs for $Const_{\mathcal{E}}$ and $VarLookup_{\mathcal{E}}$ are easy, and thus ignored.
- $PrimOp_{\mathcal{E}}$: This is done by structural induction and a case analysis over all the possible argument values of the primitive.
- $Cond_{\mathcal{E}}$: This is done by structural induction and a case analysis over the possible values produced by $\hat{k}_1(\hat{\rho},\hat{\phi})$.
- $App_{\mathcal{E}}$: For any user-defined function f, suppose that all corresponding arguments of $App_{\mathcal{E}}$ and $App_{\widehat{\mathcal{E}}}$ are related by $\widehat{\mathcal{R}}$. $\forall i \in \{1,\ldots,n\}$, let $v_i = k_i(\rho,\phi)$ and $\hat{v}_i = \hat{k}_i(\hat{\rho},\hat{\phi})$. We have $\forall i \in \{1,\ldots,n\}$, $v_i\ \widehat{\mathcal{R}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v}_i$. Since both ϕ and $\hat{\phi}$ contain only strict functions, $\widehat{\mathcal{R}}$ holds when some of the arguments is bottom. On the other hand, under the condition that all function applications are unfolded, $App_{\widehat{\mathcal{E}}}[\![f]\!](\hat{k}_1,\ldots,\hat{k}_n)$ is reduced to $\hat{\phi}\ [\![f]\!]\ (\hat{v}_1,\ldots,\hat{v}_n)$. From the supposition that $\phi\ \widehat{\mathcal{R}}_{FunVar}\ \hat{\phi}$, we have

$$\phi\ \widehat{\mathcal{R}}_{FunVar}\ \hat{\phi} \;\Rightarrow\; \phi[\![f]\!](v_1,\ldots,v_n)\ \widehat{\mathcal{R}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{\phi}[\![f]\!](\hat{v}_1,\ldots,\hat{v}_n).$$

  Hence, $App_{\mathcal{E}}\ \widehat{\mathcal{R}}\ App_{\widehat{\mathcal{E}}}$.

This concludes the proof. □