Parameterized Partial Evaluation: Semantic Specifications and Correctness Proofs

Charles Consel and Siau Cheng Khoo

Research Report YALEU/DCS/RR-896
March, 1991

This work is supported by NSF grant CCR-8809919 and DARPA N000014-91-J-4043

Parameterized 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

December 11, 91

Abstract

Parameterized partial evaluation is a uniform approach for specializing programs not only with respect to concrete values, but also with respect to abstract values such as signs, ranges and types. This paper presents semantic specifications and correctness proofs for both on-line and off-line parameterized partial evaluation of strict functional programs.

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 parameterized partial evaluation. We then use the technique of *logical relation* to prove the correctness of both on-line and off-line parameterized partial evaluation semantics. The correctness of conventional partial evaluation follows as a corollary because it is a particular case of parameterized partial evaluation.

The main contributions of this work can be summarized as follows.

1. We provide a *uniform approach* 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 (which had never been done). We define criteria for its correctness with respect to the 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 facet 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 static semantics of partial evaluation defined by the facet analysis, we formally derive the specialization semantics for off-line parameterized partial evaluation. This strategy ensures the correctness of the resulting semantics.

[*]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.

1 Introduction

Partial Evaluation

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:

> 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.

Parameterized partial evaluation

Parameterized partial evaluation [CK91a, CK91b] aims at specializing programs with respect to concrete values as well as abstract values. We develop an algebraic framework to enable modular definition of static properties. More specifically, from a concrete algebra, an abstract algebra called a *facet* is defined; it is composed of an abstract domain — capturing the properties of interest — and a set of abstract primitives that operate on this domain. The safety criteria of this abstraction are captured by the notion of *facet mapping* (see Section 2).

A facet allows one to introduce static properties at the on-line partial evaluation level. By considering an algebra whose domain is syntactic terms and operations are primitive functions, it is possible to capture the partial-evaluation behavior of primitive functions as a facet.

Facet mapping is also general enough to capture off-line partial evaluation. Just as a binding-time analysis is used to compute the static/dynamic property, we introduce a *facet analysis* to statically compute user-defined static properties (including the binding-time property itself). A specializer can then use the result of a facet analysis in the same way as it used the result of binding-time analysis previously to perform the static computations of a given program.

In summary, not only does the facet mapping extend traditional partial evaluation to include specialization of programs with respect to user-defined static properties, it also provides a uniform framework for relating three levels of evaluation: standard evaluation, on-line and off-line partial evaluation. Examples of facets are given in [CK91a, CK91b].

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 conventional partial evaluation have appeared in the literature recently, all dedicated to off-line partial evaluation. In particular, Gomard in [Gom89] 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 parameterized partial evaluation of first-order strict functional programs (an extension to higher-order programs is discussed in Section 7). This work is distinct from the existing ones in that it adopts a *uniform approach* for both defining and proving the on-line and off-line parameterized partial evaluation. More specifically,

1. We define a *core semantics* [JM76, JN90] which consists of semantic rules, and uses some uninterpreted domain names and combinator names (Section 3). 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 the program execution (Section 4). In essence, this semantics corresponds to a minimal function graph semantics [JM86]. Using other interpretations for domains and combinators, we define the on-line parameterized partial evaluation semantics (Section 5), the facet analysis and the specialization semantics (Section 6). The advantage of a factorized semantics is that different instances can be related at the level of domain definitions and combinator definitions.

2. We use the technique of *logical relations* [Abr90, JN90, MS90] to prove the correctness of parameterized partial evaluation semantics. Logical relations are defined (1) to relate the on-line parameterized partial evaluation semantics to the instrumented semantics, and (2) to relate the facet 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. To do so, we use facet mapping to define the logical relation between the basic domains of two specifications. We then extend the definition to specify the relationship between the interpreted combinators. The resulting proofs thus conform closely to our intuition about the relations between these semantics.

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

3. We show how the specializer for off-line parameterized partial evaluation can be systematically and correctly derived from its on-line counterpart, using the information collected by the facet analysis.

Lastly, we note that since conventional, on-line and off-line partial evaluation is subsumed by parameterized partial evaluation, its correctness directly follows from our results.

Notation

Most of our notation is that of standard denotational semantics. A domain 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 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 $d \in \mathrm{D}$, $\perp_D \sqsubseteq_D d$ and $\perp_D \sqcup_D d = d$. Domain subscripts are often omitted, as in $\perp \sqcup d$, when they are clear from context.

A domain D is a *lattice* if for all x, $y \in \mathrm{D}$, $x \sqcup y$ and $x \sqcap y$ exists, where $\sqcap$ is the infix greatest lower bound (glb) operator for D. Any lattice D has a maximum element $\top$ (called "top") such that for all $d \in \mathrm{D}$, $d \sqsubseteq_D \top_D$ and $\top_D \sqcap d = d$. A lattice D is *complete* if $\bigsqcup X$ and $\bigsqcap X$ exist for every subset $X \subseteq \mathrm{D}$. A domain is *flat* if all its elements apart from $\perp$ are incomparable with each other. Analogously, a lattice is *flat* if all its elements apart from $\perp$ and $\top$ are incomparable with each other.

The notation "$d \in \mathrm{D} = \cdots$" defines the domain (or set) D with "typical element" d, where $\cdots$ provides the domain specification usually via some combination of the following domain constructions: $\mathrm{D}_\perp$ denotes the domain D lifted with a new least element $\perp$. $\mathcal{P}(\mathrm{D})$ *denotes the powerset domain whose least element is the empty set, and whose partial-order relation is the subset inclusion.* $\mathrm{D}_1 \rightarrow \mathrm{D}_2$ denotes the domain of all *continuous functions* from D_1 to D_2. $\mathrm{D}_1 + \mathrm{D}_2$ and $\mathrm{D}_1 \times \mathrm{D}_2$ denote the separated sum and product, respectively, of the domains D_1 and D_2. $\mathrm{D}_1 \otimes \mathrm{D}_2$ denotes the *smashed product* of the domains D_1 and D_2; its elements are defined by the function, *smashed*, such that:

$$\begin{array}{lcll} smashed & : & \mathrm{D}_1 \times \mathrm{D}_2 \rightarrow \mathrm{D}_1 \otimes \mathrm{D}_2 & \\ smashed(d,e) & = & \langle d_1, d_2 \rangle & if\ (d_1 \neq \perp_{D_1})\ and\ (d_2 \neq \perp_{D_2}) \\ & & \perp_{D_1 \otimes D_2} & otherwise \end{array}$$

All domain/sub-domain coercions are omitted when clear from context.

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

Angle brackets are used for tupling. If $d = \langle d_1, \ldots, d_n \rangle \in \mathrm{D}_1 \times \cdots \times \mathrm{D}_n$, 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 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 $\bot[d_i/[\![x_i]\!]]$. 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 $\bar{s}$. 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 The Abstraction Methodology

As mentioned in the introduction, parameterized partial evaluation aims at specializing programs with respect to both concrete values and abstract values. Abstract values denote static properties of actual values occurring at run-time. Computations over these abstract values are defined by abstract versions of primitive functions operating on abstract domains.

To ensure the correctness of the abstract computations, it is necessary to relate an abstract algebra to the one from which it is abstracted. Because we address both on-line and off-line partial evaluation, we need to relate algebras defined at three different levels — listed in increasing abstractness: standard semantics, on-line partial evaluation and off-line partial evaluation. Algebras defined at these levels are respectively called semantic (or concrete) algebras, facets and abstract facets. In this section, we describe the abstraction methodology used to relate these algebras. Then, in Section 5.1 and Section 6.1, we instantiate this methodology to define algebras used in on-line and off-line partial evaluation, respectively.

Relating an algebra and its abstraction consists of relating their domains and relating their operators. Using abstract interpretation [AH87, JN90], a domain and its abstraction can be related by an *abstraction function*. This function is both strict, monotonic and $\bot$-reflecting. (A function $f : \mathbf{A} \rightarrow \mathbf{B}$ is $\bot$-reflecting if $fa \ = \bot_B \Rightarrow a \ = \bot_A$). Relating two operators consists of relating their respective domains and codomains. To do so, it is convenient to classify the operators as follows: An operator is *closed* if its codomain is the carrier of the algebra. It is *open* otherwise. Intuitively, an abstraction of a closed operator is passed abstract values and computes new ones, whereas an abstraction of an open operator uses the abstract values to yield a constant.

In standard semantics, actual computations are performed in the domain **Values** containing the basic values. In on-line partial evaluation, actual computations are performed using constants, *i.e.*, textual representation of the basic values. The domain of constants is noted $\widehat{\mathbf{Values}}$. It 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**. In off-line partial evaluation, actual computations are performed over the binding-time domain $\widetilde{\mathbf{Values}}$, which is composed of the binding-time values *Static* and *Dynamic*, lifted with a least element[2] $\perp_{\widetilde{Values}}$. This domain forms a chain, with ordering $\perp_{\widetilde{Values}} \sqsubseteq Static \sqsubseteq Dynamic$.

Domains **Values**, $\widehat{\mathbf{Values}}$ and $\widetilde{\mathbf{Values}}$ are related by the abstraction functions $\hat{\tau}$ and $\tilde{\tau}$ defined as follows:

$$\begin{array}{lcll} \hat{\tau} & : & \mathbf{Values} \to \widehat{\mathbf{Values}} & \\ \hat{\tau}(x) & = & \perp_{\widehat{Values}} & if\ x = \perp_{Values} \\ & & \mathcal{K}^{-1}(x) & otherwise \end{array} \qquad \begin{array}{lcll} \tilde{\tau} & : & \widehat{\mathbf{Values}} \to \widetilde{\mathbf{Values}} & \\ \tilde{\tau}(x) & = & \perp_{\widetilde{Values}} & if\ x = \perp_{\widehat{Values}} \\ & & Static & if\ x \in \mathbf{Const} \\ & & Dynamic & otherwise \end{array}$$

where $\mathcal{K}^{-1}$ is a monotonic semantics function that converts a basic value to its textual representation. Because both the domains of $\hat{\tau}$ and $\tilde{\tau}$ are sum of some basic domains, these abstraction functions are actually families of abstraction functions indexed by their summands.

Let $\alpha' = \{\alpha_{B'_i} : \mathbf{B}_i \to \mathbf{B'}_i\}$ be a family of abstraction functions, $[\mathbf{A};\mathbf{O}]$ and $[\mathbf{A'};\mathbf{O'}]$ be two algebras and $\alpha_{A'} : \mathbf{A} \to \mathbf{A'}$ be an abstraction function. Then, $\alpha_{A'} : [\mathbf{A};\mathbf{O}] \to [\mathbf{A'};\mathbf{O'}]$ is called a *facet mapping*, and is defined as follows:

Definition 1 (Facet Mapping) $\alpha_{A'} : [\mathbf{A};\mathbf{O}] \to [\mathbf{A'};\mathbf{O'}]$ *is a facet mapping with respect to* $\alpha' = \{\alpha_{B'_i} : \mathbf{B}_i \to \mathbf{B'}_i\}$ *if and only if*

1. $\mathbf{A'}$ *is a complete lattice of finite height.*
2. $\forall p' \in \mathbf{O'}$, p' *is monotonic.*
3. *If* $p \in \mathbf{O}$ *is a closed operator, then* $p' : \mathbf{A'} \to \mathbf{A'}$ *is its corresponding abstract version.*
4. *If* $p \in \mathbf{O}$ *is an open operator with functionality* $\mathbf{A} \to \mathbf{B}_i$, *where* $\mathbf{B}_i$ *is some domain different from* $\mathbf{A}$, *then* $p' : \mathbf{A'} \to \mathbf{B'}_i$ *is its corresponding abstract version.*
5. $\forall p \in \mathbf{O}$ *and its corresponding abstract version* $p' \in \mathbf{O'}$

$$\begin{array}{lll} \alpha_{A'} \circ p \sqsubseteq p' \circ \alpha_{A'} & if\ p\ is\ a\ closed\ operator \\ \alpha' \circ p \sqsubseteq p' \circ \alpha_{A'} & if\ p\ is\ an\ open\ operator\ with \\ & functionality\ \mathbf{A} \to \mathbf{B}_i \end{array}$$

Note that for simplicity, abstraction function α' will not be indexed by a codomain when the context is clear.

A facet mapping $\alpha_{A'} : [\mathbf{A};\mathbf{O}] \to [\mathbf{A'};\mathbf{O'}]$ induces a logical relation ([Nie89, JN90]) $\sqsubseteq_{\alpha_{A'}}$ defined as follows:

1. $\forall\, a \in \mathbf{A}, \forall\, a' \in \mathbf{A'}$: $a \sqsubseteq_{\alpha_{A'}} a' \Leftrightarrow \alpha_{A'}(a) \sqsubseteq_{A'} a'$.
2. *Let* $p \in \mathbf{O}$ *and* $p' \in \mathbf{O'}$ *be closed operators. Then,*
   $p \sqsubseteq_{\alpha_{A'}} p' \Leftrightarrow \forall\, a \in \mathbf{A}, \forall\, a' \in \mathbf{A'} : a \sqsubseteq_{\alpha_{A'}} a' \Rightarrow p(a) \sqsubseteq_{\alpha_{A'}} p'(a')$
3. *Let* $p \in \mathbf{O}$ *and* $p' \in \mathbf{O'}$ *be open operators and* $p : \mathbf{A} \to \mathbf{B}_i$ *for some domain* $\mathbf{B}_i$. *Then,*
   $p \sqsubseteq_{\alpha_{A'}} p' \Leftrightarrow \forall\, a \in \mathbf{A}, \forall\, a' \in \mathbf{A'} : a \sqsubseteq_{\alpha_{A'}} a' \Rightarrow p(a) \sqsubseteq_{\alpha'} p'(a')$
   where $\forall\, b \in \mathbf{B}_i, \forall\, b' \in \mathbf{B}'_i$: $b \sqsubseteq_{\alpha'} b' \Leftrightarrow \alpha_{B'_i}(b) \sqsubseteq_{B'_i} b'$.

[2]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 $\perp_{\widetilde{Values}}$, these cases would be considered as *Static*.

c	$\in$	**Const**	Constants
x	$\in$	**Var**	Variables
p	$\in$	**Po**	Primitive Operators
f	$\in$	**Fn**	Function Names
e	$\in$	**Exp**	Expressions

$$\begin{array}{rcl} e & ::= & c \mid x \mid p\,(e_1,\cdots,e_n) \mid f\,(e_1,\cdots,e_n) \mid \mathit{if}\ e_1\ e_2\ e_3 \\ \text{Prog} & ::= & \{f_i(x_1,\cdots,x_n)\ =\ e_i\} \quad (\text{f}_1 \text{ is the main function}) \end{array}$$

Figure 1: Syntactic Domains of the Subject Language

3 Core Semantics

We begin the discussion of semantic specification of parameterized partial evaluation by presenting a core semantics. The subject language is a first-order functional language. Figure 1 defines its syntactic domains. The meaning of a program is the meaning of function f_1. We assume all functions (and primitive operations) have the same arity.

The core semantics is defined in Figure 2. 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 defined by Sestoft for binding-time analysis [Ses85]. A similar structure is also used in [HY88] for defining collecting interpretation.

The core semantics is defined by semantic rules. It uses some uninterpreted domain names and combinator names. A particular semantics is thus defined by interpreting these domains and combinators appropriately, as we shall see in the remainder of this paper.

4 Standard and Instrumented Semantics

4.1 The Semantics Specifications

In Figure 3 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*. A standard signature consists of the value of the arguments to a function application. 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 2: Core Semantics

- Semantic Domains

  $v \in Result_{\mathcal{E}} = \text{Values} = \text{Int} + \text{Bool}$

  $\rho \in VarEnv = \text{Var} \rightarrow \text{Values}$

  $\phi \in FunEnv = \text{Fn} \rightarrow \text{Values}^n \rightarrow \text{Values}$

  $Env = VarEnv \times FunEnv$

- Valuation Functions

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

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

  $\phi\ [f_1]\ (v_1,\ldots,v_n)$ *wherered* $\phi = \bot[strict\ \{\lambda(v_1,\cdots,v_n)\ .\ \mathcal{E}[e_i]\ ((\bot[v_k/x_k]),\phi)\}/f_i]$

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

- 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 3: Standard Semantics

- Semantic Domains

$$\begin{array}{lll} v & \in & \mathit{Result}_{\mathcal{E}} = \text{Values} = \textit{as in Figure 3} \\ \rho & \in & \mathit{VarEnv} = \textit{as in Figure 3} \\ \phi & \in & \mathit{FunEnv} = \textit{as in Figure 3} \\ \sigma & \in & \mathit{Result}_{\mathcal{A}} = \text{Cache}_{\mathcal{A}} = \text{Fn} \rightarrow \mathcal{P}(\text{Values}^n) \end{array}$$

- Valuation Functions

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

$$\begin{array}{lll} \mathcal{E}_{Prog}\, [\{\, f_i(x_1,\cdots,x_n) = e_i\}]\langle v_1,\ldots,v_n\rangle & = & h(\bot[\{\langle v_1,\ldots,v_n\rangle\}/f_1]) \\ \quad \textit{whererec} \quad h(\sigma) & = & \sigma \sqcup h(\bigsqcup\{\mathcal{A}\,[e_i]\,(\bot[v_k/x_k])\phi \mid \langle v_1,\ldots,v_n\rangle \in \sigma[f_i],\ \forall [f_i] \in Dom(\sigma)\}) \\ \qquad\qquad\quad \phi & = & \bot[\mathit{strict}\,(\lambda(v_1,\cdots,v_n)\,.\,\mathcal{E}\,[e_i]\,(\bot[v_k/x_k])\,\phi)/f_i] \end{array}$$

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

- Combinator Definitions

$$\begin{array}{l} \mathit{Const}_{\mathcal{E}}\,[c] = \textit{as in Figure 3} \\ \mathit{VarLookup}_{\mathcal{E}}\,[x] = \textit{as in Figure 3} \\ \mathit{PrimOp}_{\mathcal{E}}\,[p]\,(k_1,\ldots,k_n) = \textit{as in Figure 3} \\ \mathit{Cond}_{\mathcal{E}}\,(k_1,\ k_2,\ k_3) = \textit{as in Figure 3} \\ \mathit{App}_{\mathcal{E}}\,[f]\,(k_1,\ldots,k_n) = \textit{as in Figure 3} \end{array}$$

$$\begin{array}{l} \mathit{Const}_{\mathcal{A}}\,[c] = \lambda(\rho,\phi)\,.\,(\lambda f\,.\,\{\}) \\ \mathit{VarLookup}_{\mathcal{A}}\,[x] = \lambda(\rho,\phi)\,.\,(\lambda f\,.\,\{\}) \end{array}$$

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

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

$$\begin{array}{l} \mathit{App}_{\mathcal{A}}\,[f]\,(a_1,\ldots,\ a_n)(\ k_1,\ldots,\ k_n) = \lambda(\rho,\phi)\,.\ \bigsqcup_{i=1}^{n} a_i(\rho,\phi)) \sqcup\ (\exists i \in \{1,\ldots,n\}\ \textit{s.t.}\ v_i = \bot \rightarrow (\lambda f\,.\,\{\}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \bot[\{\langle v_1,\ldots,v_n\rangle\}/f]) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{where}\ v_i = k_i(\rho,\phi)\ \ \forall\, i \in \{1,\cdots,n\} \end{array}$$

Figure 4: Instrumented Semantics capturing function calls

4.2 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.

Theorem 1 (Correctness of Instrumentation) *Given a program P in our first-order language, let P be 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.

5 On-Line Parameterized Partial Evaluation Semantics

In this section, we instantiate the facet mapping to on-line partial evaluation. Using logical relation, we then present the specification and correctness proof of on-line parameterized partial evaluation.

5.1 Facets and Product of Facets

An abstract algebra used in on-line parameterized partial evaluation is called a *facet*. Formally,

Definition 2 (Facet) *A facet for a semantic algebra* $[\mathrm{D};\mathrm{O}]$ *is an algebra* $[\widehat{\mathrm{D}};\widehat{\mathrm{O}}]$ *defined by a facet mapping* $\hat{\alpha}_{\widehat{D}} : [\mathrm{D};\mathrm{O}] \rightarrow [\widehat{\mathrm{D}};\widehat{\mathrm{O}}]$ *with respect to* $\hat{\tau}$.

(Abstraction function $\hat{\tau}$ is defined on page 6.) This definition implies that when an open facet operator yields a constant for some abstract values, this constant is the textual representation of the value produced by the concrete operator called with the corresponding concrete values, modulo termination (see [CK91b]).

Multiple facets can be defined for a concrete algebra; each facet captures specific static property (*i.e.*, signs, types, ranges, *etc.*). They are bundled together to form a *product of facets* defined as follows:[3]

Definition 3 (Product of Facets) *Let* $\hat{\alpha}_i : [\mathrm{D};\mathrm{O}] \rightarrow [\widehat{\mathrm{D}}^i;\widehat{\mathrm{O}}^i]$ *for* $i \in \{1,\ldots,m\}$ *be the set of facet mappings defined for a semantic algebra* $[\mathrm{D};\mathrm{O}]$. *Its product of facets, noted* $[\widehat{\mathcal{D}};\widehat{\Omega}]$, *consists of two components:*

1. *A domain* $\widehat{\mathcal{D}} = \widehat{\mathrm{D}}^1 \otimes \cdots \otimes \widehat{\mathrm{D}}^m \cong \prod_{i=1}^{m} \widehat{\mathrm{D}}^i$ *; it is the smashed product of the facet domains;*

2. *A set of product operators* $\widehat{\Omega}$ *such that* $\forall p \in \mathrm{O}$ *and its corresponding product operator* $\hat{\omega}_p \in \widehat{\Omega}$,

   (a) *if* $p \in \mathrm{O}$ *is a closed operator, then*
   $p : \mathrm{D}^n \rightarrow \mathrm{D}$, *and*
   $\hat{\omega}_p : \widehat{\mathcal{D}}^n \rightarrow \widehat{\mathcal{D}}$
   $\hat{\omega}_p = \lambda(\hat{\delta}_1,\cdots,\hat{\delta}_n) \,.\, \prod_{i=1}^{m} \hat{p}_i(\hat{\delta}^i_1,\cdots,\hat{\delta}^i_n)$

   (b) *otherwise,* $p \in \mathrm{O}$ *is an open operator*
   $p : \mathrm{D}^n \rightarrow \mathrm{D}'$ *for some domain* D', *and*
   $\hat{\omega}_p : \widehat{\mathcal{D}}^n \rightarrow \widehat{\mathrm{Values}}$
   $$\begin{aligned}\hat{\omega}_p = \ & \lambda(\hat{\delta}_1,\cdots,\hat{\delta}_n) \,.\, (\exists j \in \{1,\cdots,m\} \text{ s.t. } \hat{d}_j = \bot_{\widehat{Values}}) \rightarrow \bot_{\widehat{Values}}, \\ & \qquad (\exists j \in \{1,\cdots,m\} \text{ s.t. } \hat{d}_j \in \mathrm{Const}) \rightarrow \hat{d}_j, \top_{\widehat{Values}} \\ & \quad \textit{where } \ \hat{d} = \langle \hat{p}_1(\hat{\delta}^1_1,\cdots,\hat{\delta}^1_n),\ldots,\hat{p}_m(\hat{\delta}^m_1,\cdots,\hat{\delta}^m_n)\rangle\end{aligned}$$

[3]This is not a "product of facets" in the algebraic sense, since the result of an operation performed at one facet may have an effect on other facets in the product.

Domain $\widehat{\mathcal{D}}$ is partially ordered component-wise. The smashed product construction is used for this domain to ensure the notion of consistency explained below. Furthermore, we notice that all operators defined in the product of facets are monotonic [CK91b].

Although facets of a product are defined independently, the facet values with respect to which a program is specialized must have some consistency. This notion of consistency can be motivated by the following example. Suppose that two facets are defined for the integer algebra: one facet describes the sign of an integer value, and the other one indicates whether the value is odd or even. Then, a value such as $\langle zero,\ odd\rangle$ should not be considered as a valid facet value since *zero* is an even number. Formally,

Definition 4 *Let $[\widehat{\mathcal{D}};\widehat{\Omega}]$ be a product of facets of an algebra $[\mathrm{D};\mathrm{O}]$; $\hat{\delta} \in \widehat{\mathcal{D}}$ is consistent if and only if*

$$\bigcap_{i=1}^{m}\{d \in \mathrm{D} \mid d \sqsubseteq_{\widehat{\alpha}_i} \hat{\delta}^i\} \textit{ is neither the empty set nor } \{\bot\}.$$

In essence, the above definition implies that a consistent product of facet values represents an actual subdomain of D. We assume that all product of facet values provided to the partial-evaluation process are consistent. Technically, note that the smashed product construction is used to conveniently eliminate inconsistent values such as $\langle \bot,\ odd\rangle$.

The notion of facet can also be used to capture the traditional partial-evaluation behavior of primitives. It is called the *partial-evaluation facet.* More specifically, for a given semantic algebra, the corresponding partial-evaluation facet will define its standard semantics whenever it is passed constant arguments. The partial-evaluation facet is defined as follows:

Definition 5 (Partial-Evaluation Facet) *The partial-evaluation facet of a semantic algebra $[\mathrm{D};\mathrm{O}]$ is defined by the facet mapping $\widehat{\alpha}_{\widehat{Values}}\ :\ [\mathrm{D};\mathrm{O}] \rightarrow [\widehat{\mathrm{Values}};\widehat{\mathrm{O}}]$*

1. $\widehat{\alpha}_{\widehat{Values}}\ :\ \mathrm{D} \rightarrow \widehat{\mathrm{Values}}$
   $\widehat{\alpha}_{\widehat{Values}}\ \equiv\ \widehat{\tau}$ (as defined on page 6)

2. $\forall\, \hat{p} \in \widehat{\mathrm{O}}$ *of arity n*

$$\hat{p} : \widehat{\mathrm{Values}}^{n} \rightarrow \widehat{\mathrm{Values}}$$
$$\hat{p} = \lambda\ (\hat{d}_1,\cdots,\hat{d}_n)\ .\ \exists i \in \{1,\cdots,n\}\ s.t.\ \hat{d}_i = \bot_{\widehat{Values}} \rightarrow \bot_{\widehat{Values}},$$
$$\bigwedge_{i=1}^{n}(\hat{d}_i \in \mathrm{Const}) \rightarrow \widehat{\tau}(\mathcal{K}_p[\![p]\!](d_1,\cdots,d_n)),\ \top_{\widehat{Values}}$$
$$\textit{where } d_i\ =\ \mathcal{K}\,[\![\hat{d}_i]\!]\quad \forall\, i\ \in\ \{1,\cdots,n\}$$

5.2 The Semantics Specification

Figures 5 and 6 display the on-line parameterized partial evaluation semantics. The semantics aims at partially evaluating a program with respect to a set of static properties (including constants). It returns a residual program consisting of the specialized functions created at partial-evaluation time. We assume that the partial-evaluation facet always exists during partial evaluation.

- Semantic Domains

$$\widehat{\delta} \;\in\; \widehat{\mathcal{SD}} \;=\; \sum_{j=1}^{s} \widehat{\mathcal{D}}_j \quad \textit{where } \widehat{\mathcal{D}}_j \;=\; (\widehat{\mathrm{D}}_j^1 \otimes \cdots \otimes \widehat{\mathrm{D}}_j^m) \;\; \textit{and } \; s \textit{ is the number of basic domains}$$

$$\begin{array}{lll} \hat{v} & \in & \mathit{Result}_{\widehat{\mathcal{E}}} = \mathbf{Res} = \mathbf{Exp} \times \widehat{\mathcal{SD}} \\ \hat{\rho} & \in & \widehat{\mathit{VarEnv}} = \mathbf{Var} \rightarrow \mathbf{Res} \\ & & \widehat{\mathit{Env}} = \widehat{\mathit{VarEnv}} \times \widehat{\mathit{FunEnv}} \end{array} \qquad \begin{array}{lll} \hat{\phi} & \in & \widehat{\mathit{FunEnv}} = \mathbf{Fn} \rightarrow \mathbf{Res}^n \rightarrow \mathbf{Res} \\ \hat{\sigma} & \in & \mathit{Result}_{\widehat{\mathcal{A}}} = \mathbf{Cache}_{\widehat{\mathcal{A}}} = \mathbf{Fn} \rightarrow \mathcal{P}(\mathbf{Transf} \times \mathbf{Res}^n) \end{array}$$

- Valuation Functions

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

$$\begin{array}{l} \widehat{\mathcal{E}}_{Prog} \; [\![\{f_i(x_1, \cdots, x_n) \;=\; e_i\}]\!] \, (\hat{v}_1, \cdots, \hat{v}_n) \;=\; \mathit{MkProg} \; (\hat{h}(\perp[\{\langle \mathrm{s}, \hat{v}_1, \ldots, \hat{v}_n \rangle\}/f_1]))\hat{\phi} \\ \quad \textit{whererec } \hat{h}(\hat{\sigma}) \;=\; \hat{\sigma} \sqcup \hat{h}(\bigsqcup\{\widehat{\mathcal{A}} \; [\![e_i]\!] \; (\perp[\hat{v}'_k/x_k], \hat{\phi}) \mid \langle -, \hat{v}'_1, \ldots, \hat{v}'_n \rangle \in \hat{\sigma}[\![f_i]\!], \forall [\![f_i]\!] \in \mathit{Dom}(\hat{\sigma})\}) \\ \qquad\qquad\qquad \hat{\phi} \;=\; \perp[\mathit{strict} \; (\lambda(\hat{v}_1, \cdots, \hat{v}_n) \, . \, \widehat{\mathcal{E}} \; [\![e_i]\!] \; (\perp[\hat{v}_k/x_k], \hat{\phi}))/f_i] \end{array}$$

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

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

- Local Combinator Definitions

$$\begin{array}{l} \mathit{Const}_{\widehat{\mathcal{E}}} \, [\![c]\!] \;=\; \lambda(\hat{\rho}, \hat{\phi}) \, . \, \widehat{\mathcal{K}} \, [\![c]\!] \\ \mathit{VarLookup}_{\widehat{\mathcal{E}}} \, [\![x]\!] \;=\; \lambda(\hat{\rho}, \hat{\phi}) \, . \, \hat{\rho} \, [\![x]\!] \\ \mathit{PrimOp}_{\widehat{\mathcal{E}}} \, [\![p]\!] \, (\hat{k}_1, \ldots, \hat{k}_n) \;=\; \lambda(\hat{\rho}, \hat{\phi}) \, . \, \widehat{\mathcal{K}}_P \; [\![p]\!] \, (\hat{k}_1(\hat{\rho}, \hat{\phi}), \ldots, \; \hat{k}_n(\hat{\rho}, \hat{\phi})) \\ \mathit{Cond}_{\widehat{\mathcal{E}}} \, (\hat{k}_1, \hat{k}_2, \hat{k}_3) \;=\; \lambda(\hat{\rho}, \hat{\phi}) \, . \;\; (\hat{v}_1{\downarrow}1 \in \mathbf{Const}) \;\rightarrow ((\mathcal{K}(\hat{v}_1{\downarrow}1)) \rightarrow \hat{v}_2, \hat{v}_3), \; \langle [\![\mathit{if} \; \hat{v}_1{\downarrow}1 \;\, \hat{v}_2{\downarrow}1 \;\, \hat{v}_3{\downarrow}1]\!], \hat{v}_2{\downarrow}2 \sqcup \hat{v}_3{\downarrow}2 \rangle \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{where } \; \hat{v}_i \;=\; \hat{k}_i(\hat{\rho}, \hat{\phi}) \quad \forall i \in \{1, \; 2, \; 3\} \end{array}$$

$$\begin{array}{l} \mathit{App}_{\widehat{\mathcal{E}}} \, [\![f]\!] \, (\hat{k}_1, \ldots, \hat{k}_n) \;=\; \lambda(\hat{\rho}, \hat{\phi}) \, . \;\; (\mathit{Ft} \; [\![f]\!]){\downarrow}1 \; (\widehat{bt}(\hat{v}_1), \ldots, \widehat{bt}(\hat{v}_n)) \;=\; \mathrm{u} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \; \hat{\phi} \; [\![f]\!] \; (\hat{v}_1, \ldots, \hat{v}_n), \; \langle [\![f_{sp}(e''_1, \ldots, e''_k)]\!], \top_{\widehat{\mathcal{SD}}} \rangle \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{where } \; \hat{v}_i \;=\; \hat{k}_i(\hat{\rho}, \hat{\phi}) \quad \forall i \in \{1, \ldots, n\} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad f_{sp} \;=\; \mathit{SpName}([\![f]\!], \hat{v}'_1, \ldots, \hat{v}'_n) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle e''_1, \ldots, e''_k \rangle \;=\; \mathit{ResidArgs} \; ([\![f]\!], \; \langle b_1, \ldots, b_n \rangle, \; \langle \hat{v}_1{\downarrow}1, \ldots, \hat{v}_n{\downarrow}1 \rangle) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle \hat{v}'_1, \ldots, \hat{v}'_n \rangle \;=\; \mathit{SpPat} \; ([\![f]\!], \langle \hat{v}_1, \ldots, \hat{v}_n \rangle, \; \langle b_1, \ldots, b_n \rangle) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle b_1, \ldots, b_n \rangle \;=\; (\mathit{Ft} \; [\![f]\!]){\downarrow}2 \; (\widehat{bt}(\hat{v}_1), \ldots, \widehat{bt}(\hat{v}_n)) \end{array}$$

- Primitive Functions

$$\widehat{\mathcal{K}} \; : \; \mathbf{Const} \;\rightarrow\; \mathbf{Res}$$

$$\widehat{\mathcal{K}} \; [\![c]\!] \;=\; \langle [\![c]\!], \langle \widehat{\alpha}_{\widehat{\mathcal{D}}1}(d), \; \cdots, \widehat{\alpha}_{\widehat{\mathcal{D}}m}(d) \rangle \rangle \;\; \textit{where } d \;=\; \mathcal{K} \; [\![c]\!] \in \; \mathbf{D}$$

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

$$\begin{array}{lcl} \widehat{\mathcal{K}}_P \; [\![p^c]\!] \; (\langle e'_1, \widehat{\delta}_1 \rangle, \ldots, \langle e'_n, \widehat{\delta}_n \rangle) & = & (\widehat{\delta} \;=\; \perp_{\widehat{\mathcal{D}}}) \rightarrow \langle \perp_{Exp}, \perp_{\widehat{\mathcal{SD}}} \rangle, \\ & & (\widehat{\delta}^1 \in \; \mathbf{Const}) \rightarrow \langle \widehat{\delta}^1, \langle \widehat{\alpha}_{\widehat{\mathcal{D}}1}(d), \cdots, \; \widehat{\alpha}_{\widehat{\mathcal{D}}m}(d) \rangle \rangle, \; \langle [\![p^c(e'_1, \cdots, e'_n)]\!], \widehat{\delta} \rangle \\ & & \textit{where } \; p^c \; : \; \mathbf{D}^n \;\rightarrow\; \mathbf{D} \\ & & \qquad\quad \widehat{\delta} \;=\; \widehat{\omega}_{p^c}(\widehat{\delta}_1, \cdots, \widehat{\delta}_n) \\ & & \qquad\quad d \;=\; \mathcal{K}(\widehat{\delta}^1) \\ \widehat{\mathcal{K}}_P \; [\![p^o]\!] \; (\langle e'_1, \widehat{\delta}_1 \rangle, \ldots, \langle e'_n, \widehat{\delta}_n \rangle) & = & (\hat{d} \;=\; \perp_{\widehat{Values}}) \rightarrow \langle \perp_{Exp}, \perp_{\widehat{\mathcal{SD}}} \rangle, \\ & & \hat{d} \in \; \mathbf{Const} \rightarrow \;\; \langle \hat{d}, \langle \widehat{\alpha}_{\widehat{\mathcal{D}'}1}(d), \cdots, \widehat{\alpha}_{\widehat{\mathcal{D}'}m}(d) \rangle \rangle, \\ & & \qquad\qquad\qquad \langle [\![p^o(e'_1, \cdots, e'_n)]\!], \langle \top_{\widehat{\mathcal{D}'}1}, \cdots, \; \top_{\widehat{\mathcal{D}'}m} \rangle \rangle \\ & & \textit{where } \; p^o \; : \; \mathbf{D}^n \;\rightarrow\; \mathbf{D}' \\ & & \qquad\quad \hat{d} \;=\; \widehat{\omega}_{p^o}(\widehat{\delta}_1, \cdots, \widehat{\delta}_n) \\ & & \qquad\quad d \;=\; \mathcal{K}(\hat{d}) \end{array}$$

Figure 5: On-Line Parameterized Partial Evaluation Semantics – Part 1

- Global Combinator Definitions

$$
\begin{aligned}
&Const_{\widehat{\mathcal{A}}}\,[\![c]\!] = \lambda(\hat{\rho},\hat{\phi})\,.\,(\lambda f\,.\,\{\})\\
&VarLookup_{\widehat{\mathcal{A}}}\,[\![x]\!] = \lambda(\hat{\rho},\hat{\phi})\,.\,(\lambda f\,.\,\{\})\\
&PrimOp_{\widehat{\mathcal{A}}}\,[\![p]\!]\,(\hat{a}_1,\ldots,\hat{a}_n) = \lambda(\hat{\rho},\hat{\phi})\,.\,\bigsqcup_{i=1}^{n}\hat{a}_i(\hat{\rho},\hat{\phi})\\
&Cond_{\widehat{\mathcal{A}}}\,(\hat{a}_1,\hat{a}_2,\hat{a}_3)\,\hat{k}_1 = \lambda(\hat{\rho},\hat{\phi})\,.\,\hat{a}_1(\hat{\rho},\hat{\phi})\sqcup\hat{\delta}_1^1\in \mathrm{Const}\rightarrow\ (\mathcal{K}(\hat{\delta}_1^1)\rightarrow\hat{a}_2(\hat{\rho},\hat{\phi}),\hat{a}_3(\hat{\rho},\hat{\phi})),\ \hat{a}_2(\hat{\rho},\hat{\phi})\sqcup\hat{a}_3(\hat{\rho},\hat{\phi})\\
&\qquad\qquad where\ \langle e_1,\langle\hat{\delta}_1^1,\ldots,\hat{\delta}_1^m\rangle\rangle = \hat{k}_1(\hat{\rho},\hat{\phi})\\
&App_{\widehat{\mathcal{A}}}\,[\![f]\!]\,(\hat{a}_1,\ldots,\hat{a}_n)\,(\hat{k}_1,\cdots,\hat{k}_n) = \lambda(\hat{\rho},\hat{\phi})\,.\,(\bigsqcup_{i=1}^{n}\hat{a}_i(\hat{\rho},\hat{\phi}))\sqcup\hat{\sigma}\\
&\qquad where\ \hat{v}_i = \hat{k}_i(\hat{\rho},\hat{\phi})\quad\forall i\in\{1,\cdots,n\}\\
&\qquad\quad \hat{\sigma} = ((Ft[\![f]\!])\downarrow 1(\widehat{bt}(\hat{v}_1),\ldots,\widehat{bt}(\hat{v}_n)) = \mathrm{u})\rightarrow\\
&\qquad\qquad \bot[\{\langle \mathrm{u},\hat{v}_1,\ldots,\hat{v}_n\rangle\}/f],\ \bot[\{\langle \mathrm{s},\hat{v}'_1,\ldots,\hat{v}'_n\rangle\}/f]\\
&\qquad\quad \langle\hat{v}'_1,\ldots,\hat{v}'_n\rangle = SpPat\,([\![f]\!],\langle\hat{v}_1,\ldots,\hat{v}_n\rangle,\ \langle b_1,\ldots,b_n\rangle)\\
&\qquad\quad \langle b_1,\ldots,b_n\rangle = (Ft\,[\![f]\!])\downarrow 2\,(\widehat{bt}(\hat{v}_1),\ldots,\widehat{bt}(\hat{v}_n))
\end{aligned}
$$

- *MkProg* Definition

$$
\begin{aligned}
&MkProg\ \hat{\sigma}\ \hat{\phi} = \{\ f_i^{sp}(x_1,\ldots,x_k) = \hat{v}\downarrow 1 \mid \forall\langle \mathrm{s},\hat{v}_1,\ldots,\hat{v}_n\rangle\in\hat{\sigma}[\![f_i]\!],\ \forall[\![f_i]\!]\in Dom(\hat{\sigma})\}\\
&\quad where\ f_i^{sp} = SpName([\![f_i]\!],\hat{v}_1,\ldots,\hat{v}_n)\\
&\qquad \hat{v} = \widehat{\mathcal{E}}\,[\![e_i]\!]\,(\bot[\hat{v}_k/x_k],\hat{\phi})\\
&\qquad \langle x_1,\ldots,x_k\rangle = ResidPars\,([\![f_i]\!],\hat{v}_1\downarrow 1,\ldots,\hat{v}_n\downarrow 1)
\end{aligned}
$$

Figure 6: On-Line Parameterized Partial Evaluation Semantics – Part 2

Thus, because a partial-evaluation facet is defined for each semantic domain, it will be assigned to the first component of every product of facets. A sum of these products of facets is noted $\widehat{\mathcal{SD}}$; each summand corresponds to a semantic algebra. For brevity, We write $\top_{\widehat{\mathcal{SD}}}$ to represent the maximum value of any summand of $\widehat{\mathcal{SD}}$.

Domain **Exp** is a flat domain of expressions. Domain $Result_{\widehat{\mathcal{E}}}$ is ordered component-wise.

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: $\widehat{\mathcal{E}}$, $\widehat{\mathcal{A}}$ and $\widehat{\mathcal{E}}_{Prog}$. Function $\widehat{\mathcal{E}}$ defines the partial evaluation of an expression. It produces a pair of values $\hat{v}\in$ **Res** $=$ **Exp** $\times\ \widehat{\mathcal{SD}}$, where the first component is a residual expression and the second component is a product of facet values. The partial-evaluation facet is assumed to be the first component of a product of facets.

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 (*eg.*, [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 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: u and 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: $\mathtt{u} \sqsubseteq \mathtt{s}$. 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

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

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

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*:

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 $\mathbf{SpFn}$ is a flat domain of specialized function names.

[4]The specialization pattern describes information about the arguments used in specializing the function. Each argument value is represented in the pattern by an expression and a product of facet values. The expression 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: $(\mathbf{Exp} \times \widehat{\mathcal{SD}})^n$, or simply, $\mathbf{Res}^n$.

3. $\widehat{bt}$: **Res** $\rightarrow$ $\widetilde{\textbf{Values}}$ returns the binding time of a residual pair. It is defined as $\widehat{bt}(e, \hat{\delta}) = \hat{\tau}(\hat{\delta}^1)$.

4. If a function call is to be specialized, then

   (a) For those arguments that are *not* propagated at function specialization,

   - *ResidArgs* : $\textbf{Fn} \times \widetilde{\textbf{Values}}^n \times \textbf{Exp}^n \rightarrow \textbf{Exp}^m$ (for $m \leq n$) returns a tuple of residual arguments;
   - *ResidPars* : $\textbf{Fn} \times \textbf{Exp}^n \rightarrow \textbf{Var}^m$ (for $m \leq n$) returns a tuple of parameters replacing these residual arguments in the partial-evaluation signature.

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

$$\begin{array}{rcl}
SpPat & = & \lambda(f, \langle \hat{v}_1, \ldots, \hat{v}_n \rangle, \langle b_1, \ldots, b_n \rangle) \,.\, \langle \hat{v}'_1, \ldots, \hat{v}'_n \rangle \\
& & \textit{where } \forall\, i \in \{1, \ldots, n\}, \\
& & \qquad \hat{v}'_i \;=\; \langle e'_i, \langle \hat{d}_i, \hat{\delta}_i^2, \ldots, \hat{\delta}_i^m \rangle \rangle \\
& & \qquad \langle e'_i, \hat{d}_i \rangle \;=\; \begin{array}{lcl} b_i = \textit{Static} & \rightarrow & \langle e_i, \hat{\delta}_i^1 \rangle, \\ b_i = \textit{Dynamic} & \rightarrow & \langle x_i, \top_{\widehat{Values}} \rangle, \; \langle \bot_{Exp}, \bot_{\widehat{Values}} \rangle \end{array} \\
& & \qquad \hat{v}_i \;=\; \langle e_i, \langle \hat{\delta}_i^1, \hat{\delta}_i^2, \ldots, \hat{\delta}_i^m \rangle \rangle
\end{array}$$

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.*

5.3 Correctness of Partial Evaluation Semantics

Let us first restate a theorem from [CK91b], which asserts that any constant produced by partially evaluating a primitive call is always correct with respect to the standard semantics, modulo termination.

Theorem 2 *Let $[\widehat{\mathcal{D}}; \widehat{\Omega}]$ be a product of facets (including the partial-evaluation facet) for an algebra* $[\mathbf{D}; \mathbf{O}]$. *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 $d_i \in \bigcap_{j=1}^{m} \{d \in \mathbf{D} \mid d \sqsubseteq_{\hat{\alpha}_{\widehat{\mathcal{D}}_j}} \hat{\delta}_i^j\}$, for $i \in \{1, \ldots, n\}$. Then,*

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

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

Theorem 3 *Given a program P in our first-order language. Suppose that (1) the input to this program is completely known at partial-evaluation time, and (2) all function calls in this program 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:

$$\hat{\rho} \;=\; \lambda\ [\![x]\!]\ .\ \langle \hat{\tau}(\rho[\![x]\!]), \langle \hat{\alpha}_{\widehat{\mathcal{D}}^1}(\rho[\![x]\!]), \ldots, \hat{\alpha}_{\widehat{\mathcal{D}}^m}(\rho[\![x]\!])\rangle\rangle \quad for\ (\rho[\![x]\!]) \in \mathbf{D}.$$

The proof is given in Appendix B.

Since an abstract value used during partial evaluation represents a set of concrete values, 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,

$$\langle \hat{v}_1, \ldots, \hat{v}_n\rangle \ \textit{represents the set}\ \{\langle v_1, \ldots, v_n\rangle \mid \hat{\alpha}_{\widehat{\mathcal{D}}_i}(v_i) = \hat{v}_i,\ i \in \{1, \ldots, n\}\}$$

where, for each $v_i \in \mathbf{D}_i$, the corresponding abstraction function is $\hat{\alpha}_{\widehat{\mathcal{D}}_i}$ (for $i \in \{1, \ldots, n\}$). The safety criterion described in 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 logical 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 $\hat{v} = \langle e, \hat{\delta}\rangle \in (\mathbf{Exp} \times \widehat{\mathcal{SD}})$, $\mathcal{R}^{\widehat{\mathcal{E}}}$ 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 partly 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 $\mathcal{R}^{\widehat{\mathcal{E}}}$.

5.3.1 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 (the product of facets) of the result of $\widehat{\mathcal{E}}$.

Definition 6 (Relation $\sqsubseteq_{\hat{\alpha}_{\widehat{\mathcal{SD}}}}$) *For any value $v \in \mathbf{D}$ and $\hat{\delta} \in \widehat{\mathcal{SD}}$ with $\hat{\delta} = \langle \hat{\delta}^1, \ldots, \hat{\delta}^m\rangle$,*

$$v \sqsubseteq_{\hat{\alpha}_{\widehat{\mathcal{SD}}}} \hat{\delta} \Leftrightarrow \forall i \in \{1, \ldots, m\},\ v \sqsubseteq_{\hat{\alpha}_{\widehat{\mathcal{D}}^i}} \hat{\delta}^i.$$

Since $(v \sqsubseteq_{\hat{\alpha}_{\widehat{SD}}} \hat{\delta}) \equiv (\bigwedge_{i=1}^{m}(v \sqsubseteq_{\hat{\alpha}_{\widehat{D}^i}} \hat{\delta}^i))$, $\sqsubseteq_{\hat{\alpha}_{\widehat{SD}}}$ is a logical relation between Values and $\widehat{SD}$ (assuming that the values have been injected in their respective sum domain).

Definition 7 (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:*

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

Lemma 3 *Given a program P in our first-order language. 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 $\forall [\![f]\!] \in \mathbf{Fn}, \forall i \in \{1,\ldots,n\}, \forall v_i \in \mathbf{Values}, \forall \hat{v}_i \in \mathbf{Res}$,

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

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):

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

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

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

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

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

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 maximal element of domain $\widehat{\mathcal{SD}}$ 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 4 (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 1 *During partial evaluation, all values* $\hat{v} \in$ Res *satisfy the following conditions:*

- $\hat{v}\downarrow 1 \in \text{Const} \wedge (\hat{v}\downarrow 2)\downarrow 1 \in \text{Const} \iff \hat{v}\downarrow 1 = (\hat{v}\downarrow 2)\downarrow 1$
- $\hat{v}\downarrow 1 = \perp_{Exp} \iff (\hat{v}\downarrow 2)\downarrow 1 = \perp_{\widehat{Values}}$.

The above observation comes directly from the definition of $\widehat{\mathcal{K}}_P$ in Figure 5.

We say that a value $\hat{v} \in$ Res is *$\mathcal{R}$-consistent* if (1) it satisfies one of the above conditions and (2) its second component (product of facets) is *consistent*, as defined in Definition 4. This fact is used in the next section.

5.3.2 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 8 (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:*

$$
\begin{array}{lcl}
v\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v} & \Leftrightarrow & (\hat{v} \textit{ is } \mathcal{R}\textit{-consistent}) \wedge (v \sqsubseteq_{\hat{\alpha}_{\widehat{SD}}} \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 & \Leftrightarrow & \bigwedge_{i=1}^{n}(v_i \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{E}}}} \hat{v}_i) \\
\sigma\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{A}}}}\ \hat{\sigma} & \Leftrightarrow & \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} & \Leftrightarrow & \forall[\![x]\!] \in \mathbf{Var},\ \rho[\![x]\!]\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{\rho}[\![x]\!] \\
\phi\ \mathcal{R}^{\widehat{\mathcal{A}}}_{FunEnv}\ \hat{\phi} & \Leftrightarrow & \forall[\![f]\!] \in \mathbf{Fn}, \forall j \in \{1,\ldots,n\}, \forall v_j \in \mathbf{Values}, \forall \hat{v}_j \in \mathbf{Res}, \\
 & & \quad \bigwedge_{j=1}^{n}(v_j\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{E}}}}\ \hat{v}_j) \Rightarrow \phi[\![f]\!](v_1,\ldots,v_n)\ \mathcal{R}^{\widehat{\mathcal{A}}}_{Result_{\widehat{\mathcal{E}}}}\ \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 & \Leftrightarrow & 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} & \Leftrightarrow & \forall d \in D_1, \forall \hat{d} \in \widehat{D}_1,\ d\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_1}\ \hat{d} \Rightarrow f(d)\ \mathcal{R}^{\widehat{\mathcal{A}}}_{D_2}\ \hat{f}(\hat{d}).
\end{array}
$$

Note that the $\mathcal{R}$-consistency (Observation 1) ensures that the first component of $\hat{v}$, the residual expression is consistent with the result of the partial-evaluation facet. 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 facet 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 1, the modified value is still $\mathcal{R}$-consistent.

Lemma 4 *Given a program P in our first-order language. 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

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

It suffices to show that $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds for all the corresponding pairs of combinators used by $\mathcal{A}$ and $\widehat{\mathcal{A}}$ respectively. By structural induction, it is easy to see that $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds for constants, variables and primitive calls.

1. $Cond_{\mathcal{A}}$: By structural induction hypothesis, $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds for the test expression.
   (a) If $\hat{\delta}_1^1 \in$ Const, since $k(\rho,\phi)\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{k}(\hat{\rho},\hat{\phi})$, the branch chosen will be the same for both $\mathcal{A}$ and $\widehat{\mathcal{A}}$, and by structural induction hypothesis, $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds.
   (b) If $\hat{\delta}_1^1 = \top_{\widehat{Values}}$, then all non-trivial calls in both branches are recorded by $\widehat{\mathcal{A}}$. Again, by structural induction hypothesis, $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds.
2. $App_{\mathcal{A}}$: By structural induction hypothesis, $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds for all the arguments to the application. As for the application itself, if it is recorded by $\mathcal{A}$, then it is non-trivial. By structural induction on the arguments, the application is also recorded by $\widehat{\mathcal{A}}$. Its transformation tag is either u or s. It is easy to see that $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds when the transformation tag is u. If it is s, $\mathcal{R}^{\widehat{\mathcal{A}}}$ holds if $\forall i \in \{1,\ldots,n\}$, $\hat{v}_i{\downarrow}2 \sqsubseteq \hat{v}'_i$, where $\langle \hat{v}'_1,\cdots,\hat{v}'_n\rangle$ is the result of applying $SpPat$ in $\widehat{\mathcal{A}}$. This is true by the definition of $\widehat{bt}$, $SpPat$ and $Ft[\![f]\!]{\downarrow}2$.

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

Notice that, for a value $\hat{v}$, there may be more than one value v such that $v\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{v}$. Therefore, the above lemma shows that given an expression e, $\widehat{\mathcal{A}}$ captures all calls within e that may be invoked under different initial value v with $v\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{v}$. The following theorem uses Lemma 3 to prove that the final cache produced by the global semantics is "complete" in the sense that it captures all the non-trivial calls performed during standard evaluation.

Theorem 5 (Correctness of Global Partial Evaluation Semantics) *Given a program P in our first-order language. Let $\widehat{\mathcal{E}}$ be a valuation function of the partial evaluation semantics such that $\mathcal{E}\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \widehat{\mathcal{E}}$. Let $\langle v_1,\ldots,v_n\rangle$ and $\langle \hat{v}_1,\ldots,\hat{v}_n\rangle$ be initial inputs to P for standard and partial evaluation semantics respectively, such that $v_i\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{v}_i$, $\forall i \in \{1,\ldots,n\}$. If σ and $\hat{\sigma}$ are the final caches produced by $\mathcal{A}$ and $\widehat{\mathcal{A}}$ respectively, then $\sigma\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{\sigma}$.*

Proof : Firstly, we notice from the definition of $\widehat{\mathcal{E}}_{Prog}$ that $\langle \mathbf{s},\hat{v}_1,\ldots,\hat{v}_n\rangle \in \hat{\sigma}[f_1]$, just like $\langle v_1,\ldots,v_n\rangle \in \sigma[f_1]$. Next, $\hat{h}$ in $\widehat{\mathcal{E}}_{Prog}$ applies $\widehat{\mathcal{A}}$ to each partial-evaluation signature in the cache, similar to function h in $\mathcal{E}_{Prog}$. Since l.u.b. operation is closed under $\mathcal{R}^{\widehat{\mathcal{A}}}$, $\sigma\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{\sigma}$. □

By Theorem 1, we know that σ contains all the non-trivial calls performed at the standard evaluation. Since $\sigma\ \mathcal{R}^{\widehat{\mathcal{A}}}\ \hat{\sigma}$, all these calls must be captured by $\hat{\sigma}$.

5.3.3 Correctness of $\mathcal{R}^{\widehat{\mathcal{E}}_1}$

We now prove the correctness of the residual expression produced by $\widehat{\mathcal{E}}$ using the relation $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ which relates a residual expression to a concrete value produced by $\mathcal{E}$. Intuitively, a residual expression and a concrete value are related if the former evaluates (under standard evaluation) to the latter. This requires "post-evaluation" of the residual expression. Therefore, $\mathcal{R}^{\widehat{\mathcal{E}}_1}$ is not simply a relation between a value and a residual expression; it is a relation between the value, the residual expression and its "post-evaluated" value. In the following definition, we introduce the notion of *satisfiability* to aid in formulating this relation. This notion is similar, though simpler, to the definition of *agreeability* used by Gomard in [Gom89]. For clarity, $a =_\perp b$ denotes the equality between a and b, provided both of them terminate. Of Course, $a = b \Rightarrow a =_\perp b$.

Definition 9 (Satisfiability) *Let P be a program in our first-order language. Let $\rho_d \in VarEnv$ be a variable environment for a residual expression such that $Dom(\rho_d) = FV(\hat{v}\downarrow 1)$. We say ρ_d satisfies the pair $\langle v, \hat{v}\rangle$ if*

$$v =_\perp \mathcal{E}[\![\hat{v}\downarrow 1]\!](\rho_d, \phi') \ \wedge \ v \sqsubseteq_{\hat{\alpha}_{\widehat{SD}}} \hat{v}\downarrow 2,$$

where $v \in$ Values, $\hat{v} \in$ Res, and ϕ' is the function environment, defined by the standard semantics, for the specialized version of program P.

Without loss of generality, we assume that every user-defined function consists of two parameters (x_1 and x_2). To show the relationship between $\mathcal{E}$ and $\widehat{\mathcal{E}}$, we must first show that the function environments they take as arguments are related. The following lemma clarifies this relation.

Lemma 5 *Given a program P in our first-order language. Let ϕ and $\hat{\phi}$ be the two function environments for P defined by the standard semantics and the partial evaluation semantics respectively. For any user-defined function f, let ρ_d be a variable environment that satisfies both the pairs $\langle v_1, \hat{v}_1\rangle$ and $\langle v_2, \hat{v}_2\rangle$ with $v_1, v_2 \in$ Values and $\hat{v}_1, \hat{v}_2 \in$ Res. Then,*

$$\phi[\![f]\!]\ (v_1, v_2) \ =_\perp \ \mathcal{E}\ [\![(\hat{\phi}[\![f]\!]\ (\hat{v}_1, \hat{v}_2))\downarrow 1]\!]\ (\rho_d, \phi')$$

where ϕ' is the function environment, defined by the standard semantics, for the specialized version of P.

Proof : The proof is similar to that described in [Gom89]. The major difference lies in the the fact that the property of a cache σ is used to show the correctness of partial evaluation of function application. This enables to show the correctness of multiple instances of specialized functions.

Since the lemma involves three functions $FunEnv$: ϕ, $\hat{\phi}$ for program P, and ϕ' for the residual program, we define the functional Φ as

$$\begin{aligned}\Phi\ \langle\phi_a, \hat{\phi}_a, \phi'_a\rangle \ = \ \langle\ & \perp[\{\lambda(v_1, v_2)\ .\ \mathcal{E}[\![e_i]\!](\perp[v_k/x_k,], \phi_a)\}/f_i \mid \forall\ [\![f_i]\!] \in \mathbf{Fn}],\\ & \perp[\{\lambda(\hat{v}_1, \hat{v}_2)\ .\ \widehat{\mathcal{E}}[\![e_i]\!](\perp[\hat{v}_k/x_k], \hat{\phi}_a)\}/f_i \mid \forall\ [\![f_i]\!] \in \mathbf{Fn}],\\ & \perp[\{\lambda(v)\ .\ \mathcal{E}[\![e^{sp}]\!](\perp[v/x], \phi'_a)\}/f^{sp} \mid \forall\ \textit{specialized function } f^{sp}]\rangle\end{aligned}$$

In this proof, i ranges over all user-defined functions.

Let $\mathcal{R}^{\widehat{\mathcal{E}_1}}$ be the predicate over Φ such that:

$$\begin{aligned}
\mathcal{R}^{\widehat{\mathcal{E}_1}}\Phi &= \mathcal{R}^{\widehat{\mathcal{E}_1}}\langle \phi, \hat{\phi}, \phi' \rangle \\
&= \forall\, [\![f_i]\!] \in \mathbf{Fn},\ \forall v_1, v_2 \in \mathbf{Values},\ \forall \hat{v}_1, \hat{v}_2 \in \mathbf{Res},\ \forall \rho_d \in VarEnv, \\
&\quad \bigwedge_{j=1}^{2} (\rho_d\ satisfies\ \langle v_j, \hat{v}_j \rangle) \ \Rightarrow\ \phi[\![f_i]\!](v_1, v_2) =_{\perp} \mathcal{E}[\![(\hat{\phi}[\![f_i]\!](\hat{v}_1, \hat{v}_2)) \downarrow 1]\!](\rho_d, \phi')
\end{aligned}$$

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

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

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

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

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,

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

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 over 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 over 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 $\phi[\![f_i]\!](v_1, v_2) = \perp_{Values}$ and $\hat{\phi}[\![f_i]\!](\hat{v}_1, \hat{v}_2) = \perp_{Res}$. 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. $\perp[v_k/x_k]$ is abbreviated by ρ and $\perp[\hat{v}_k/x_k]$ by $\hat{\rho}$.
2. Given an expression e, we abbreviate $\mathcal{E}\llbracket e\rrbracket(\rho, \phi_n)$ by $\llbracket e\rrbracket_{\mathcal{E}}$, and $\widehat{\mathcal{E}}\llbracket e\rrbracket(\hat{\rho}, \hat{\phi}_n)$ by $\llbracket e\rrbracket_{\widehat{\mathcal{E}}}$.

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, $\llbracket p(e_1, \ldots, e_n)\rrbracket$. Let $v = \llbracket p(e_1, \ldots, e_n)\rrbracket_{\mathcal{E}}$ and $\hat{v} = \llbracket p(e_1, \ldots, e_n)\rrbracket_{\widehat{\mathcal{E}}}$.

  - $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ holds trivially if $\hat{v} = \perp_{Res}$.
  - If $\hat{v}\downarrow 1$ is a constant, then $\hat{v}\downarrow 1 = \hat{\tau}(v)$ from Theorem 2. 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 $\llbracket p(e''_1, \ldots, e''_n)\rrbracket$, where $e''_i = \llbracket e_i\rrbracket_{\widehat{\mathcal{E}}}\ \forall i \in \{1, \ldots, n\}$. 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:

$$\begin{array}{lll}
\multicolumn{2}{l}{\mathcal{E}\llbracket\llbracket p(e_1, \ldots, e_n)\rrbracket_{\widehat{\mathcal{E}}}\rrbracket(\rho_d, \phi'_{n+1})} & \\
= & \mathcal{E}\llbracket p(e''_1, \ldots, e''_n)\rrbracket(\rho_d, \phi'_{n+1}) & \\
= & \mathcal{K}_P\llbracket p\rrbracket((\mathcal{E}\llbracket e''_1\rrbracket(\rho_d, \phi'_{n+1})), \ldots, (\mathcal{E}\llbracket e''_n\rrbracket(\rho_d, \phi'_{n+1}))) & \text{[from standard semantics]} \\
=_\perp & \mathcal{K}_P\llbracket p\rrbracket(\llbracket e_1\rrbracket_{\mathcal{E}}, \ldots, \llbracket e_n\rrbracket_{\mathcal{E}}) & \text{[structural induction hypothesis]} \\
= & \llbracket p(e_1, \ldots, e_n)\rrbracket_{\mathcal{E}} & \text{[from standard semantics]}
\end{array}$$

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

- e is a conditional expression, $\llbracket if\ e_1\ e_2\ e_3\rrbracket$. $\mathcal{R}^{\widehat{\mathcal{E}}_1}_{n+1}$ 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 $\llbracket if\ e''_1\ e''_2\ e''_3\rrbracket$, where $e''_i = \llbracket e_i\rrbracket_{\widehat{\mathcal{E}}}\ \forall i \in \{1, \ldots, 3\}$. Therefore,

$$\begin{array}{lll}
\multicolumn{2}{l}{\mathcal{E}\llbracket\llbracket if\ e_1\ e_2\ e_3\rrbracket_{\widehat{\mathcal{E}}}\rrbracket(\rho_d, \phi'_{n+1})} & \\
= & \mathcal{E}\llbracket if\ e''_1\ e''_2\ e''_3\rrbracket(\rho_d, \phi'_{n+1}) & \\
= & (\mathcal{E}\llbracket e''_1\rrbracket(\rho_d, \phi'_{n+1})) \to (\mathcal{E}\llbracket e''_2\rrbracket(\rho_d, \phi'_{n+1})), (\mathcal{E}\llbracket e''_3\rrbracket(\rho_d, \phi'_{n+1})) & \text{[from standard semantics]} \\
=_\perp & \llbracket e_1\rrbracket_{\mathcal{E}} \to \llbracket e_2\rrbracket_{\mathcal{E}}, \llbracket e_3\rrbracket_{\mathcal{E}} & \text{[structural induction hypothesis]} \\
= & \llbracket if\ e_1\ e_2\ e_3\rrbracket_{\mathcal{E}} & \text{[from standard semantics]}
\end{array}$$

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

- e is a function application, $\llbracket f_i(e_1, e_2)\rrbracket$. 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 $\llbracket f_i(e_1, e_2)\rrbracket_{\widehat{\mathcal{E}}}$ becomes $\llbracket f_i^{sp}(\llbracket e_2\rrbracket_{\widehat{\mathcal{E}}})\rrbracket$ where f_i^{sp} is the specialized function. The partial-evaluation signature obtained from this application is (by the definition of $\widehat{\mathcal{A}}$)

$$\begin{array}{lrcl}
\langle s, \llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \langle\llbracket x_2\rrbracket, \hat{\delta}'\rangle\rangle & \textit{where}\ \hat{\delta}' & = & \langle\top_{\widehat{Values}}, \hat{\delta}^2, \cdots, \hat{\delta}^m\rangle \\
& \langle\llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \langle\llbracket x_2\rrbracket, \hat{\delta}'\rangle\rangle & = & SpPat(\llbracket f_i\rrbracket, \langle\llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \llbracket e_2\rrbracket_{\widehat{\mathcal{E}}}\rangle, \langle Static, Dynamic\rangle) \\
& \langle -, \langle\hat{\delta}^1, \hat{\delta}^2, \cdots, \hat{\delta}^n\rangle\rangle & = & \llbracket e_2\rrbracket_{\widehat{\mathcal{E}}}
\end{array}$$

Thus, the specialized function f_i^{sp} is included in the residual program produced; its definition is as follows.

$$\begin{aligned} f_i^{sp}(x_2) &= \llbracket(\hat{\phi}_n\llbracket f_i\rrbracket(\llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \langle\llbracket x_2\rrbracket, \hat{\delta}'\rangle))\downarrow 1\rrbracket \\ &= \llbracket(\hat{\phi}_n\llbracket f_i\rrbracket(\hat{\tau}(\llbracket e_1\rrbracket_{\mathcal{E}}), \langle\llbracket x_2\rrbracket, \hat{\delta}'\rangle))\downarrow 1\rrbracket \end{aligned}$$

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_i^{sp} in ϕ'_{n+1} is

$$strict(\lambda v\ .\ \mathcal{E}\llbracket(\hat{\phi}_n\llbracket f_i\rrbracket(\hat{\tau}(\llbracket e_1\rrbracket_{\mathcal{E}}), \langle\llbracket x_2\rrbracket, \hat{\delta}'\rangle))\downarrow 1\rrbracket(\perp[v/x_2], \phi'_n)) \tag{2}$$

Thus, we have

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

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

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

$$\phi_n\llbracket f_i\rrbracket(\llbracket e_1\rrbracket_{\mathcal{E}}, \llbracket e_2\rrbracket_{\mathcal{E}}) =_\perp \mathcal{E}\llbracket(\hat{\phi}_n\llbracket f_i\rrbracket(\llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \llbracket e_2\rrbracket_{\widehat{\mathcal{E}}}))\downarrow 1\rrbracket\ (\rho_d,\ \phi'_{n+1}). \tag{3}$$

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

$$v_i =_\perp \mathcal{E}\llbracket\hat{v}_i\downarrow 1\rrbracket(\rho_d, \phi'_{n+1})\ \wedge\ v_i \sqsubseteq_{\hat{\alpha}_{\widehat{SD}}} \hat{v}_i\downarrow 2.$$

This is true by the structural induction hypothesis, Property 2 about ϕ'_{n+1}, and Theorem 4.

Using ρ_d, the fixpoint induction hypothesis is

$$\phi_n\llbracket f_i\rrbracket(\llbracket e_1\rrbracket_{\mathcal{E}}, \llbracket e_2\rrbracket_{\mathcal{E}}) =_\perp \mathcal{E}\llbracket(\hat{\phi}_n\llbracket f_i\rrbracket(\llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \llbracket e_2\rrbracket_{\widehat{\mathcal{E}}}))\downarrow 1\rrbracket(\rho_d, \phi'_n),$$

Notice that the only difference between the hypothesis and Equation 3 is the usage of ϕ'_n and ϕ'_{n+1}. Let $e' = (\hat{\phi}_n\llbracket f_i\rrbracket((\llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}), (\llbracket e_2\rrbracket_{\widehat{\mathcal{E}}})))\downarrow 1$. 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 $\llbracket f^{sp}(r'_2)\rrbracket$. This residual call would be the result of partially evaluating a function call. Let the function call be $\llbracket f(r_1, r_2)\rrbracket$. Then, it would be the case that at the n^{th} approximation, we had

$$\llbracket f(r_1, r_2)\rrbracket_{\mathcal{E}} =_\perp \mathcal{E}\llbracket\llbracket f(r_1, r_2)\rrbracket_{\widehat{\mathcal{E}}}\rrbracket(\rho_d, \phi'_n) = \mathcal{E}\llbracket f^{sp}(r'_k)\rrbracket(\rho_d, \phi'_n)$$

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

$$\llbracket f(r_1, r_2)\rrbracket_{\mathcal{E}} =_\perp \mathcal{E}\llbracket f^{sp}(r'_k)\rrbracket(\rho_d, \phi'_{n+1}) \tag{4}$$

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

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

5.3.4 Correctness of $\mathcal{R}^{\hat{\mathcal{E}}}$

Now, we are ready to define the relation between $\mathcal{E}$ and $\hat{\mathcal{E}}$. This is defined in terms of the result of Theorem 4 and Lemma 5. Firstly, since both $\mathcal{E}$ and $\hat{\mathcal{E}}$ 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 [Gom89].

Definition 10 (Agreeability) *Let P be a program in our first-order language. 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*

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

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

Observation 2 *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 2 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 in our first-order language. 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 6 (Correctness of Local Partial Evaluation Semantics) *Given a program P in our first-order language. 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

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

Proof : From Theorem 4 and Corollary 1. □

6 Off-Line Parameterized Partial Evaluation Semantics

Off-Line parameterized partial evaluation of a program consists of two phases: the preprocessing phase called the *facet analysis*, and the *specialization* phase. In this section, the abstraction methodology described in Section 2 is instantiated for facet analysis. We then use the technique of logical relation to define and prove the correctness of facet analysis. This conforms to our intuition that facet analysis is an abstraction of on-line parameterized partial evaluation. Lastly, we describe a systematic way of deriving the specializer from on-line partial evaluation using the result of facet analysis, and list some optimizations that can be performed to improve the efficiency of the specializer.

6.1 Abstract Facets and Product of Abstract Facets

An abstract algebra used at the analysis level is called an *abstract facet.* Formally,

Definition 11 (Abstract Facet) *An abstract facet* $[\widetilde{\mathrm{D}};\widetilde{\mathrm{O}}]$ *of a facet* $[\widehat{\mathrm{D}};\widehat{\mathrm{O}}]$ *is defined by a facet mapping* $\widetilde{\alpha}_{\widetilde{D}} : [\widehat{\mathrm{D}};\widehat{\mathrm{O}}] \rightarrow [\widetilde{\mathrm{D}};\widetilde{\mathrm{O}}]$ *with respect to* $\widetilde{\tau}$.

(Abstraction function $\widetilde{\tau}$ is defined on page 6.) The use of facet mapping in the definition ensures that, if an open operator of an abstract facet maps some properties into the value *Static*, then the open operator of the corresponding facet will yield a constant at specialization-time, modulo termination.

Just like the on-line parameterized partial evaluation, multiple abstract facets of an algebra are bundled together to form a *product of abstract facets* defined as follows:

Definition 12 (Product of Abstract Facets) *Let* $\widetilde{\alpha}_i : [\widehat{\mathrm{D}}^i;\widehat{\mathrm{O}}^i] \rightarrow [\widetilde{\mathrm{D}}^i;\widetilde{\mathrm{O}}^i]$ *for* $i \in \{1,\ldots,m\}$ *be the set of Facet mappings defined for the facets of a semantic algebra* $[\mathrm{D};\mathrm{O}]$. *Its product of abstract facets, noted* $[\widetilde{\mathcal{D}},\widetilde{\Omega}]$, *consists of two components:*

1. *A domain* $\widetilde{\mathcal{D}} = \prod_{i=1}^{m} \widetilde{\mathrm{D}}^i$ *is the smashed product of the abstract facet domains;*
2. *A set of product operators* $\widetilde{\Omega}$ *such that* $\forall p \in \mathrm{O}$ *and its corresponding product operator* $\widetilde{\omega}_p \in \widetilde{\Omega}$,
   (a) *if* $\widetilde{p}$ *is a closed operator, then*
   $p : \mathrm{D}^n \rightarrow \mathrm{D}$, *and*
   $\widetilde{\omega}_p : \widetilde{\mathcal{D}}^n \rightarrow \widetilde{\mathcal{D}}$
   $\widetilde{\omega}_p = \lambda\ (\widetilde{\delta}_1,\cdots,\widetilde{\delta}_n).\ \prod_{i=1}^{m} \widetilde{p}_i(\widetilde{\delta}_1^i,\cdots,\widetilde{\delta}_n^i)$
   (b) *otherwise,* $\widetilde{p} \in \widetilde{\mathrm{O}}$ *is an open operator, and*
   $p : \mathrm{D}^n \rightarrow \mathrm{D}'$ *for some domain* D', *and*
   $\widetilde{\omega}_p : \widetilde{\mathcal{D}}^n \rightarrow \widetilde{\mathrm{Values}}$
   $\widetilde{\omega}_p = \lambda(\widetilde{\delta}_1,\cdots,\widetilde{\delta}_n)\ .\ (\exists j \in \{1,\cdots,m\}\ s.t.\ \widetilde{d}_j = \bot_{\widetilde{Values}}) \rightarrow \bot_{\widetilde{Values}},$
   $(\exists j \in \{1,\cdots,m\}\ s.t.\ \widetilde{d}_j = \mathit{Static}) \rightarrow \mathit{Static}, \mathit{Dynamic}$
   where $\widetilde{d} = \langle \widetilde{p}_1(\widetilde{\delta}_1^1,\cdots,\widetilde{\delta}_n^1),\ldots,\widetilde{p}_m(\widetilde{\delta}_1^m,\cdots,\widetilde{\delta}_n^m)\rangle$

Domain $\widetilde{\mathcal{D}}$ is partially ordered component-wise, and all operators defined in the product $[\widetilde{\mathcal{D}};\widetilde{\Omega}]$, are monotonic [CK91b].

Just as the partial evaluation semantics of algebraic operators is captured by a facet, the computation of their binding-time values can similarly be captured by the notion of abstract facet. Such an abstract facet is called a *binding-time facet.*

Definition 13 (Binding-Time Facet) *The binding-time facet of a partial-evaluation facet* $[\widehat{\text{Values}};\hat{\text{O}}]$ *is defined by the facet mapping* $\tilde{\alpha}_{\widehat{Values}} : [\widehat{\text{Values}};\hat{\text{O}}] \to [\widetilde{\text{Values}};\tilde{\text{O}}]$

1. $$\begin{aligned} \tilde{\alpha}_{\widehat{Values}} &: \widehat{\text{Values}} \to \widetilde{\text{Values}} \\ \tilde{\alpha}_{\widehat{Values}} &\equiv \tilde{\tau} \end{aligned}$$
2. $\forall\, \tilde{o} \in \tilde{\text{O}}$ *of arity* n
   $$\begin{aligned} &\tilde{o} : \widetilde{\text{Values}}^n \to \widetilde{\text{Values}} \\ &\tilde{o} = \lambda\, (\tilde{d}_1, \cdots, \tilde{d}_n)\, .\, \exists j \in \{1, \ldots, n\} \text{ s.t. } \tilde{d}_j = \bot_{\widetilde{Values}} \to \bot_{\widetilde{Values}}, \\ &\qquad \bigwedge_{i=1}^{n} (\tilde{d}_i = \mathit{Static}) \to \mathit{Static},\ \mathit{Dynamic} \end{aligned}$$

6.2 Specification of Facet Analysis

Analogous to the definition of on-line parameterized partial evaluation, we assume the binding-time facet to be always defined in a facet analysis. The main semantic domain used by the analysis is noted $\widetilde{\mathcal{SD}}$. It is a sum of products of abstract facets. The binding-time facet is assigned to the first component of each product. For brevity, we write $\top_{\widetilde{\mathcal{SD}}}$ to represent the maximum value of any summand of $\widetilde{\mathcal{SD}}$.

Figure 7 displays the facet analysis for a first-order language. The analysis aims at collecting facet information for each function in a given program; this forms the *facet signature* of the function. More precisely, a facet signature in domain Sig is created when a function call processed by the facet 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 $\widetilde{\mathcal{SD}}^n$.

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

The analysis is *monovariant*: each user-defined function is associated with *one* facet signature. Various facet 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

$$\begin{aligned} \forall \tilde{\sigma}_1, \tilde{\sigma}_2 \in \text{Cache}_{\tilde{\mathcal{A}}},\ \tilde{\sigma}_1 \sqcup \tilde{\sigma}_2 &= \bot[\langle t, \tilde{\delta}_1, \ldots, \tilde{\delta}_n\rangle / f \mid \forall [\![f]\!] \in \mathit{Dom}(\tilde{\sigma}_1) \cup \mathit{Dom}(\tilde{\sigma}_2)] \\ \mathit{where}\ \langle t, \tilde{\delta}_1, \ldots, \tilde{\delta}_n\rangle &= ([\![f]\!] \in (\mathit{Dom}(\tilde{\sigma}_1) \cap \mathit{Dom}(\tilde{\sigma}_2))) \to \langle t' \sqcup t'',\ \tilde{\delta}'_1 \sqcup \tilde{\delta}''_1, \ldots, \tilde{\delta}'_n \sqcup \tilde{\delta}''_n\rangle, \\ &\qquad [\![f]\!] \in \mathit{Dom}(\tilde{\sigma}_1) \to \tilde{\sigma}_1[\![f]\!],\ \tilde{\sigma}_2[\![f]\!] \\ \langle t', \tilde{\delta}'_1, \ldots, \tilde{\delta}'_n\rangle &= \tilde{\sigma}_1[\![f]\!] \\ \langle t'', \tilde{\delta}''_1, \ldots, \tilde{\delta}''_n\rangle &= \tilde{\sigma}_2[\![f]\!] \end{aligned}$$

- Semantic Domains

$$\begin{array}{lcl}
\tilde{\delta} & \in & Result_{\widetilde{\mathcal{E}}} = \widetilde{\mathcal{SD}} = \sum_{j=1}^{s} \widetilde{\mathcal{D}}_j \;\; \textit{where } \widetilde{\mathcal{D}}_j = (\widetilde{\mathrm{D}}_{j1} \otimes \cdots \otimes \widetilde{\mathrm{D}}_{jm}) \textit{ and } s \textit{ is the number of basic domains} \\
\tilde{\rho} & \in & \widetilde{VarEnv} = \mathrm{Var} \rightarrow \widetilde{\mathcal{SD}} \\
\tilde{\phi} & \in & \widetilde{FunEnv} = \mathrm{FEnv} = \mathrm{Fn} \rightarrow \widetilde{\mathcal{SD}}^n \rightarrow \widetilde{\mathcal{SD}} \\
 & & \widetilde{Env} = \widetilde{VarEnv} \times \widetilde{FunEnv} \\
\tilde{s} & \in & \mathrm{Sig} = (\mathrm{Transf} \times \widetilde{\mathcal{SD}}^n) \\
\tilde{\sigma} & \in & Result_{\widetilde{\mathcal{A}}} = \mathrm{Cache}_{\widetilde{\mathcal{A}}} = \mathrm{Fn} \rightarrow \mathrm{Sig}
\end{array}$$

- Valuation Functions

$$\begin{array}{l}
\widetilde{\mathcal{E}}_{Prog} : \mathrm{Program} \rightarrow \widetilde{\mathcal{SD}}^n \rightarrow \mathrm{Cache}_{\widetilde{\mathcal{A}}} \\
\quad \widetilde{\mathcal{E}}_{Prog}\,[\{f_i(x_1,\cdots,x_n) = e_i\}]\,\langle\tilde{\delta}_1,\cdots,\tilde{\delta}_n\rangle = \tilde{h}(\bot[\langle \mathrm{s},\tilde{\delta}_1,\cdots,\tilde{\delta}_n\rangle/f_1]) \\
\qquad \textit{whererec} \quad \tilde{h}(\tilde{\sigma}) = \tilde{\sigma} \sqcup \tilde{h}(\bigsqcup\{\widetilde{\mathcal{A}}\,[e_i]\,(\bot[\tilde{\delta}_k/x_k],\tilde{\phi}) \mid \langle -,\tilde{\delta}_1,\ldots,\tilde{\delta}_n\rangle = \tilde{\sigma}[f_i],\ \forall[f_i] \in Dom(\tilde{\sigma})\}) \\
\qquad\qquad\qquad\quad \tilde{\phi} = \bot[\{\lambda(\tilde{\delta}_1,\ldots,\tilde{\delta}_n)\,.\,\widetilde{\mathcal{E}}\,[e_i](\bot[\tilde{\delta}_k/x_k],\tilde{\phi})\}/f_i] \\
\widetilde{\mathcal{E}} = \overline{\mathcal{E}} \\
\widetilde{\mathcal{A}} = \overline{\mathcal{A}}
\end{array}$$

- Combinator Definitions

$$\begin{array}{l}
Const_{\widetilde{\mathcal{E}}}\,[c] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,\widetilde{\mathcal{K}}\,[c] \\
VarLookup_{\widetilde{\mathcal{E}}}\,[x] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,\tilde{\rho}[x] \\
PrimOp_{\widetilde{\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})) \\
Cond_{\widetilde{\mathcal{E}}}\,(\tilde{k}_1,\tilde{k}_2,\tilde{k}_3) = \lambda(\tilde{\rho},\tilde{\phi})\,.\ \tilde{\delta}_1 = \bot_{\widetilde{\mathcal{SD}}} \rightarrow \bot_{\widetilde{\mathcal{SD}}}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \tilde{\delta}_1^1 = Static \rightarrow \tilde{\delta}_2 \sqcup \tilde{\delta}_3,\ \top_{\widetilde{\mathcal{SD}}} \\
\qquad\qquad\qquad\qquad \textit{where } \tilde{\delta}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi}) \quad \forall i \in \{1,2,3\} \\
App_{\widetilde{\mathcal{E}}}\,[f]\,(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(Ft[f])\!\downarrow\!1\,(\tilde{\delta}_1^1,\ldots,\tilde{\delta}_n^1) = \mathrm{u} \rightarrow \tilde{\phi}\,[f]\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n),\ \top_{\widetilde{\mathcal{SD}}} \\
\qquad\qquad\qquad\qquad\qquad \textit{where } \tilde{\delta}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi}) \quad \forall i \in \{1,\ldots,n\} \\
Const_{\widetilde{\mathcal{A}}}\,[c] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(\lambda f\,.\,\bot_{Sig}) \\
VarLookup_{\widetilde{\mathcal{A}}}\,[x] = \lambda(\tilde{\rho},\tilde{\phi})\,.\,(\lambda f\,.\,\bot_{Sig}) \\
PrimOp_{\widetilde{\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}) \\
Cond_{\widetilde{\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}) \\
App_{\widetilde{\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\qquad\qquad\qquad\qquad\qquad \textit{where } \tilde{\sigma}' = (Ft[f])\!\downarrow\!1\,(\tilde{\delta}_1^1,\ldots,\tilde{\delta}_n^1) = \mathrm{u} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \rightarrow \bot[\langle \mathrm{u},\tilde{\delta}_1,\ldots,\tilde{\delta}_n\rangle/f],\ \bot[\langle \mathrm{s},\tilde{\delta}_1',\ldots,\tilde{\delta}_n'\rangle/f] \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \tilde{\delta}_i' = \langle b_i,\tilde{\delta}_i^2,\ldots,\tilde{\delta}_i^m\rangle \quad \forall i \in \{1,\ldots,n\} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle b_1,\ldots,b_n\rangle = (Ft[f])\!\downarrow\!2\,(\tilde{\delta}_1^1,\ldots,\tilde{\delta}_n^1) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \tilde{\delta}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi}) \quad \forall i \in \{1,\ldots,n\}
\end{array}$$

- Primitive Functions

$$\begin{array}{l}
\widetilde{\mathcal{K}} : \mathrm{Const} \rightarrow \widetilde{\mathcal{SD}} \\
\quad \widetilde{\mathcal{K}}\,[c] = \langle \widetilde{\Gamma}^1(d),\cdots,\widetilde{\Gamma}^m(d)\rangle \;\; \textit{where } \widetilde{\Gamma}^i = \widetilde{\alpha}_{\widetilde{\mathcal{D}}^i} \circ \widehat{\alpha}_{\widehat{\mathcal{D}}^i} \textit{ and } d = \mathcal{K}\,[c] \\
\widetilde{\mathcal{K}}_P : \mathrm{Po} \rightarrow \widetilde{\mathcal{SD}}^n \rightarrow \widetilde{\mathcal{SD}} \\
\quad \widetilde{\mathcal{K}}_P\,[p^c]\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n) = \widetilde{\omega}_{p^c}(\tilde{\delta}_1,\cdots,\tilde{\delta}_n) \;\; \textit{where } p^c : \mathrm{D}^n \rightarrow \mathrm{D} \\
\quad \widetilde{\mathcal{K}}_P\,[p^o]\,(\tilde{\delta}_1,\ldots,\tilde{\delta}_n) = \tilde{b} = \bot_{\widetilde{Values}} \rightarrow \bot_{\widetilde{\mathcal{SD}}},\langle \tilde{b},\top_{\widetilde{\mathcal{D}}'_2},\cdots,\top_{\widetilde{\mathcal{D}}'_m}\rangle \\
\qquad\qquad\qquad\qquad\qquad \textit{where } p^o : \mathrm{D}^n \rightarrow \mathrm{D}' \\
\qquad\qquad\qquad\qquad\qquad\qquad \tilde{b} = \widetilde{\omega}_{p^o}(\tilde{\delta}_1,\cdots,\tilde{\delta}_n)
\end{array}$$

Figure 7: Facet Analysis

6.3 Correctness of Facet Analysis

The initial input to the facet analysis is an abstraction of the initial input of on-line partial evaluation. The facet 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 $\widetilde{\mathcal{A}}$.

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

We begin by defining a logical relation between product of facets and product of abstract facets.

Definition 14 (Relation $\sqsubseteq_{\tilde{\alpha}_{\widetilde{\mathcal{SD}}}}$) *For any value $\hat{\delta} \in \widehat{\mathcal{SD}}$ and $\tilde{\delta} \in \widetilde{\mathcal{SD}}$,*

$$\hat{\delta} \sqsubseteq_{\tilde{\alpha}_{\widetilde{\mathcal{SD}}}} \tilde{\delta} \Leftrightarrow \forall i \in \{1, \ldots, m\},\ \hat{\delta}^i \sqsubseteq_{\tilde{\alpha}_{\widetilde{D}^i}} \tilde{\delta}^i.$$

where $\sqsubseteq_{\tilde{\alpha}_{\widetilde{D}^i}}$ is the logical relation induced from the facet mapping from $\widehat{\mathbf{D}}^i$ to $\widetilde{\mathbf{D}}^i$.

Since $(\hat{\delta} \sqsubseteq_{\tilde{\alpha}_{\widetilde{\mathcal{SD}}}} \tilde{\delta}) \equiv (\bigwedge_{i=1}^{m} (\hat{\delta}^i \sqsubseteq_{\tilde{\alpha}_{\widetilde{D}^i}} \tilde{\delta}^i))$, $\sqsubseteq_{\tilde{\alpha}_{\widetilde{\mathcal{SD}}}}$ is a logical relation between $\widehat{\mathcal{SD}}$ and $\widetilde{\mathcal{SD}}$.

Definition 15 (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:*

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

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

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

Theorem 7 (Correctness of Local Facet Analysis) $\widehat{\mathcal{E}}\ \mathcal{R}^{\widetilde{\mathcal{E}}}\ \widetilde{\mathcal{E}}$.

Proof : From Lemma 6. □

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

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

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 $\hat{\rho}\ \mathcal{R}^{\widetilde{\mathcal{E}}}\ \tilde{\rho}$.

6.3.2 Correctness of the Global Analysis

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

Definition 16 (Relation $\mathcal{R}^{\widetilde{\mathcal{A}}}$) *$\mathcal{R}^{\widetilde{\mathcal{A}}}$ is a logical relation between the domains of $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ defined by extending relation $\mathcal{R}^{\widetilde{\mathcal{E}}}$ to include the relation between $\hat{\sigma}$ and $\tilde{\sigma}$ produced by $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ respectively:*

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

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

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

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

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

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. $Cond_{\widetilde{\mathcal{A}}}$: 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.

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

The use of facet 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 facet analysis. We achieve this annotation by assigning a unique label to each expression in a program and binding this label to the corresponding facet information. A cache, noted $\bar{\psi}$, maps each label of an expression to its facet information. For a label l, we write $(\bar{\psi}\ l)_v$ to denote the product of abstract facet value corresponding to l. If l is the label of a function call, then $(\bar{\psi}\ l)_t$ refers to its transformation (*i.e.*, unfolding or suspension).

6.4.1 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.*, $\bar{\psi} \in \mathbf{AtCache}$).

Figures 8 and 9 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 facet 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 facet 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: facet value of the arguments and function call transformation.

6.4.2 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].

[5]Note that for simplicity we did not introduce labels in the core semantics presented on page 8. Indeed, labels are only used for the specialization semantics.

2. $App_{\widetilde{\mathcal{A}}}$: 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 facet analysis level.

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

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

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

   - If $\tilde{t} = \mathrm{s}$, 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,

     $$\langle \hat{b}_1, \ldots, \hat{b}_n \rangle \ \sqsubseteq\ \langle \tilde{b}_1, \ldots, \tilde{b}_n \rangle$$

     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*
     $\langle \tilde{b}_1, \ldots, \tilde{b}_n \rangle = (Ft[\![f]\!])\!\downarrow\! 2\, (\tilde{\delta}_1^1, \ldots, \tilde{\delta}_n^1)$
     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

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

     where $\forall i \in \{1, \ldots, n\}$, $\tilde{\delta}_i' = \langle \tilde{b}_i, \tilde{\delta}_i^2, \ldots, \tilde{\delta}_i^m \rangle$. Since $\langle \tilde{t}, \tilde{\delta}_1', \ldots, \tilde{\delta}_n' \rangle$ is the facet 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 8 (Correctness of Global Facet Analysis) *Given a program P in our first-order language. 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 facet 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 $\langle \mathrm{s}, \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle$ is the corresponding facet signature for f_1 in $\tilde{\sigma}$. Therefore, $\langle \mathrm{s}, \tilde{\delta}_1, \ldots, \tilde{\delta}_n \rangle \sqsubseteq \tilde{\sigma}[\![f_1]\!]$. This captures the initial call to the on-line partial evaluation: $\langle \mathrm{s}, \hat{v}_1, \ldots, \hat{v}_n \rangle \in \hat{\sigma}[\![f_1]\!]$. Next $\tilde{h}$ in $\widetilde{\mathcal{E}}_{Prog}$ applies $\widetilde{\mathcal{A}}$ to each facet signature in the cache, like function $\hat{h}$ in $\widehat{\mathcal{E}}_{Prog}$. Since l.u.b. operation is closed under $\mathcal{R}^{\widetilde{\mathcal{A}}}$, $\hat{\sigma}\ \mathcal{R}^{\widetilde{\mathcal{A}}}\ \tilde{\sigma}$. □

6.4 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 facet 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 facet information as much as possible. The uses of facet information are listed below.

- Semantic Domains

$$\begin{array}{rcl} l & \in & \text{Labels} \\ \hat{\delta} & \in & \widehat{\mathcal{SD}} \;=\; \text{as in On-Line Sem.} \\ \langle \tilde{t}, \tilde{\delta} \rangle & \in & \text{Att} \;=\; (\text{Transf} \times \widetilde{\mathcal{SD}}) \\ \tilde{\psi} & \in & \text{AtCache} \;=\; \text{Labels} \rightarrow \text{Att} \\ \hat{\rho} & \in & \widehat{VarEnv} \;=\; \text{as in On-Line Sem.} \end{array} \qquad \begin{array}{rcl} \hat{v} & \in & Result_{\widehat{\mathcal{E}}} \;=\; \text{Res} \;=\; \text{as in On-Line Sem.} \\ \tilde{\delta} & \in & \widetilde{\mathcal{SD}} \;=\; \text{as in Off-Line Sem.} \\ \hat{\sigma} & \in & Result_{\widehat{\mathcal{A}}} \;=\; \text{as in On-Line Sem.} \\ \hat{\phi} & \in & \widehat{FunEnv} \;=\; \text{as in On-Line Sem.} \\ & & \widehat{Env} = \widehat{VarEnv} \times \widehat{FunEnv} \times \text{AtCache} \end{array}$$

- Valuation Functions

$$\begin{array}{l} \widehat{\mathcal{E}}_{Prog} : \text{Prog} \rightarrow \text{Res}^n \rightarrow \text{AtCache} \rightarrow \text{Prog}_\bot \\ \widehat{\mathcal{E}}_{Prog}\, [\{f_i(x_1, \cdots, x_n) \;=\; e_i\}]\, \langle \hat{v}_1, \ldots, \hat{v}_n \rangle\, \tilde{\psi} \;=\; MkProg\, (\hat{h}(\bot[\{\langle s, \hat{v}_1, \ldots, \hat{v}_n \rangle\}/f_1]))\tilde{\psi}\hat{\phi} \\ \quad \textit{whererec}\ \hat{h}(\hat{\sigma}) \;=\; \hat{\sigma} \sqcup \hat{h}(\bigsqcup\{\mathcal{A}_{\widetilde{\mathcal{SA}}}\, [e_i](\bot[\hat{v}'_k/x_k], \hat{\phi}, \tilde{\psi}) \mid \langle -, \hat{v}'_1, \ldots, \hat{v}'_n \rangle \in \hat{\sigma}[f_i],\ \forall [f_i] \in Dom(\hat{\sigma})\}) \\ \qquad\qquad\quad \hat{\phi} \;=\; \bot[\textit{strict}\ \{\lambda(\hat{v}_1, \cdots, \hat{v}_n)\, .\, \mathcal{E}_{\widetilde{\mathcal{SE}}}\, [e_i](\bot[\hat{v}_k/x_k], \hat{\phi}, \tilde{\psi})\}/f_i] \end{array}$$

- *MkProg* Definition

$$\begin{array}{l} MkProg\ \hat{\sigma}\ \tilde{\psi}\ \hat{\phi} \;=\; \{\, f_i^{sp}(x_1, \ldots, x_k) \;=\; \hat{v}\!\downarrow\!1 \mid \forall \langle s, \hat{v}_1, \ldots, \hat{v}_n \rangle \in \hat{\sigma}[f_i],\ \forall [f_i] \in Dom(\hat{\sigma})\} \\ \quad \textit{where}\ f_i^{sp} \;=\; SpName([f_i], \hat{v}_1, \ldots, \hat{v}_n) \\ \qquad\qquad \hat{v}' \;=\; \mathcal{E}_{\widetilde{\mathcal{SE}}}\, [e_i](\bot[\hat{v}_k/x_k], \hat{\phi}, \tilde{\psi}) \\ \qquad\qquad \langle x_1, \ldots, x_k \rangle \;=\; ResidPars\, ([f_i], \hat{v}_1\!\downarrow\!1, \ldots, \hat{v}_n\!\downarrow\!1) \end{array}$$

Figure 8: Specialization Semantics – Part 1

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 this section, we briefly mention some of them.

Eliminating Useless Facet Computations

Facet analysis allows to determine statically which facet computations will produce a constant value at specialization-time. We can exploit this information to eliminate, prior to specialization, the facet computations that do not produce static values.

More precisely, for each static expression in a program, we can determine the abstract facets which actually produce the static value (there can be more than one). It is then possible to determine a minimal set of these abstract facets, needed to produce all the static values in the program. In doing so, we eliminate the time spent on computing irrelevant facet information.

Determining Specialization Actions Statically

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 facet value of an expression. This technique has been used in traditional off-line partial evaluation [Con89, CD90].

- Local Combinator Definitions

$$Const_{\widetilde{\mathcal{SE}}}\ [c]\ \langle l\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ \widehat{\mathcal{K}}\ [c]$$
$$Var_{\widetilde{\mathcal{SE}}}\ [x]\ \langle l\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ \hat\rho\ [x]$$
$$PrimOp_{\widetilde{\mathcal{SE}}}\ [p]\ (\hat k_1,\ldots,\hat k_n)\ \bar\delta\ \langle l,l_1,\ldots,l_n\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ \widehat{\mathcal{K}}_{SP}\ [p]\ (\hat k_1(\hat\rho,\hat\phi,\tilde\psi),\ldots,\ \hat k_n(\hat\rho,\hat\phi,\tilde\psi))\ (\tilde\psi\ l)_v$$
$$Cond_{\widetilde{\mathcal{SE}}}\ (\hat k_1,\hat k_2,\hat k_3)\ \langle l,l_1,l_2,l_3\rangle \ =$$
$$\lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ (\tilde\psi\ l_1)^1_v = Static\ \rightarrow\ (\mathcal{K}(\hat v_1{\downarrow}1)\ \rightarrow \hat v_2,\ \hat v_3),\ \langle[if\ \hat v_1{\downarrow}1\ \hat v_2{\downarrow}1\ \hat v_3{\downarrow}1], \hat v_2{\downarrow}2\ \sqcup \hat v_3{\downarrow}2\rangle$$
$$where\ \ \hat v_i \ = \ \hat k_i(\hat\rho,\hat\phi,\tilde\psi)\quad \forall i\in\{1,\ 2,\ 3\}$$
$$App_{\widetilde{\mathcal{SE}}}\ [f]\ (\hat k_1,\ldots,\hat k_n)\ \langle l,l_1,\ldots,l_n\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ (\tilde\psi\ l)_t = \mathrm{u}\rightarrow\ \hat\phi\ [f]\ (\hat v_1,\ldots,\hat v_n),$$
$$\langle[f_{sp}(e''_1,\ldots,e''_k)], \top_{\widehat{\mathcal{SD}}}\rangle$$
$$\begin{aligned} where\ \ \hat v_i \ &= \ \hat k_i(\hat\rho,\hat\phi,\tilde\psi)\quad \forall i\in\{1,\ldots,n\}\\ f_{sp} \ &= \ SpName([f],\hat v'_1,\ldots,\hat v'_n)\\ \langle e''_1,\ldots,e''_k\rangle \ &= \ ResidArgs\ ([f],\ \langle b_1,\ldots,b_n\rangle,\ \langle\hat v_1{\downarrow}1,\ldots,\hat v_n{\downarrow}1\rangle)\\ \langle\hat v'_1,\ldots,\hat v'_n\rangle \ &= \ SpPat\ ([f],\langle\hat v_1,\ldots,\hat v_n\rangle,\ \langle b_1,\ldots,b_n\rangle)\\ \langle b_1,\ldots,b_n\rangle \ &= \ \langle\tilde\delta'^1_1,\ldots,\tilde\delta'^1_n\rangle\\ \tilde\delta'_i &= (\tilde\psi\ l_i)_v\quad \forall i\in\{1,\ldots,n\}\end{aligned}$$

- Primitive Functions

$$\widehat{\mathcal{K}}\ :\ \mathrm{Const}\ \rightarrow\ \mathrm{Res}$$
$\widehat{\mathcal{K}}$ [c] = (as in On–Line Semantics)

$$\widehat{\mathcal{K}}_{SP}\ :\ \mathrm{Po}\ \rightarrow\ \mathrm{Res}^n\ \rightarrow\ \widetilde{\mathcal{SD}}\ \rightarrow\ \mathrm{Res}$$
$$\widehat{\mathcal{K}}_{SP}\ [p^c]\ (\langle e'_1,\widehat\delta_1\rangle,\ldots,\langle e'_n,\widehat\delta_n\rangle)\ \tilde\delta\ =$$
$$(\widehat\delta\ =\ \bot_{\widehat{\mathcal{D}}})\rightarrow\langle\bot_{Exp},\bot_{\widehat{\mathcal{SD}}}\rangle,\ (\tilde\delta^1 = Static)\rightarrow\langle\widehat\delta^1,\langle\widehat\alpha_{\widehat{\mathcal{D}}1}(d),\cdots,\ \widehat\alpha_{\widehat{\mathcal{D}}m}(d)\rangle\rangle,\ \langle[p^c(e'_1,\cdots,e'_n)],\widehat\delta\rangle$$
$$where\ \ p^c\ :\ \mathrm{D}^n\ \rightarrow\ \mathrm{D}\ ;\ \ \widehat\delta\ =\ \widehat\omega_{p^c}(\widehat\delta_1,\cdots,\widehat\delta_n)\ ;\ \ d\ =\ \mathcal{K}(\widehat\delta^1)$$
$$\widehat{\mathcal{K}}_{SP}\ [p^o]\ (\langle e'_1,\widehat\delta_1\rangle,\ldots,\langle e'_n,\widehat\delta_n\rangle)\ \tilde\delta\ =$$
$$(\hat d\ =\ \bot_{\widehat{Values}})\rightarrow\langle\bot_{Exp},\bot_{\widehat{\mathcal{SD}}}\rangle,\ \tilde\delta^1 = Static\rightarrow\ \langle\hat d,\langle\widehat\alpha_{\widehat{\mathcal{D}}'1}(d),\cdots,\widehat\alpha_{\widehat{\mathcal{D}}'m}(d)\rangle\rangle,$$
$$\langle[p^o(e'_1,\cdots,e'_n)],\langle\top_{\widehat{\mathcal{D}}'1},\cdots,\ \top_{\widehat{\mathcal{D}}'m}\rangle\rangle$$
$$where\ \ p^o\ :\ \mathrm{D}^n\ \rightarrow\ \mathrm{D}'\ ;\ \ \hat d\ =\ \widehat\omega_{p^o}(\widehat\delta_1,\cdots,\widehat\delta_n)\ ;\ \ d\ =\ \mathcal{K}(\hat d)$$

- Global Combinator Definitions

$$Const_{\widetilde{\mathcal{SA}}}\ [c]\ \langle l\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ \lambda l\ .\ \bot_{Att}$$
$$Var_{\widetilde{\mathcal{SA}}}\ [x]\ \langle l\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ \lambda l\ .\ \bot_{Att}$$
$$PrimOp_{\widetilde{\mathcal{SA}}}\ [p]\ (\hat a_1,\ldots,\hat a_n)\ \langle l,l_1,\ldots,l_n\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ \bigsqcup_{i=1}^{n}\hat a_i(\hat\rho,\hat\phi,\tilde\psi)$$
$$Cond_{\widetilde{\mathcal{SA}}}\ (\hat a_1,\hat a_2,\hat a_3)\ \hat k_1\ \langle l,l_1,l_2,l_3\rangle \ =$$
$$\lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ (\hat a_1(\hat\rho,\hat\phi,\tilde\psi))\sqcup\ ((\tilde\psi\ l)^1_v = Static)\ \rightarrow\ \ \mathcal{K}(\hat\delta^1_1)\rightarrow\hat a_2(\hat\rho,\hat\phi,\tilde\psi),\hat a_3(\hat\rho,\hat\phi,\tilde\psi),$$
$$\hat a_2(\hat\rho,\hat\phi,\tilde\psi)\sqcup\hat a_3(\hat\rho,\hat\phi,\tilde\psi))$$
$$App_{\widetilde{\mathcal{SA}}}\ [f]\ (\hat a_1,\ldots,\ \hat a_n)\ (\hat k_1,\ldots,\ \hat k_n)\ \langle l,l_1,\ldots,l_n\rangle \ = \ \lambda(\hat\rho,\hat\phi,\tilde\psi)\ .\ (\bigsqcup_{i=1}^{n}\hat a_i(\hat\rho,\hat\phi,\tilde\psi))\sqcup\hat\sigma$$
$$\begin{aligned} where\ \ \hat v_i \ &= \ \hat k_i(\hat\rho,\hat\phi,\tilde\psi)\quad \forall i\in\{1,\cdots,n\}\\ \hat\sigma \ &= \ \ (\tilde\psi\ l)_t = \mathrm{u}\rightarrow\ \bot[\{\langle\mathrm{u},\hat v_1,\ldots,\hat v_n\rangle\}/f],\ \bot[\{\langle\mathrm{s},\hat v'_1,\ldots,\hat v'_n\rangle\}/f]\\ \langle\hat v'_1,\ldots,\hat v'_n\rangle \ &= \ SpPat\ ([f],\langle\hat v_1,\ldots,\hat v_n\rangle,\ \langle b_1,\ldots,b_n\rangle)\\ \langle b_1,\ldots,b_n\rangle \ &= \ \langle\tilde\delta'^1_1,\ldots,\tilde\delta'^1_n\rangle\\ \tilde\delta_i \ &= \ (\tilde\psi\ l_i)_v\quad \forall i\in\{1,\ldots,n\}\end{aligned}$$

Figure 9: Specialization Semantics – Part 2

7 Conclusion

In [CK91a, CK91b] we presented a generic form of partial evaluation, parameterized with respect to user-defined static properties. Based on this work and the technique of factorized semantics, this paper provides semantic specifications and correctness proofs for both on-line and off-line partial evaluation of functional programs.

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 facet analysis (and thus 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 implemented and shown to work using SML.

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 [Jon91].

Currently, we are exploring various analyses aimed at taking advantage of facet information to optimize the specialization process. These optimizations (mentioned in Section 6.4) include eliminating useless facet computations and determining specialization actions statically.

References

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

[AH87] S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis Horwood, 1987.

[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.

[CK91a] C. Consel and S. C. Khoo. Parameterized partial evaluation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 92–106, 1991.

[CK91b] C. Consel and S. C. Khoo. Parameterized partial evaluation. Research Report 865, Yale University, New Haven, Connecticut, USA, 1991. Extended version.

[Con88] C. Consel. New insights into partial evaluation: the Schism experiment. In H. Ganzinger, editor, *ESOP'88, 2^{nd} 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.

[Gom89] C. K. Gomard. Higher order partial evaluation – hope for the lambda calculus. Master's thesis, DIKU, University of Copenhagen, Copenhagen, Denmark, 1989.

[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.

[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 in our first-order language. 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 $v''_i \neq \perp, \forall i \in \{1, \ldots, n\}$. □