J. Functional Programming 5 (4): 461–500, October 1995 © 1995 Cambridge University Press

On-line and off-line partial evaluation: semantic specifications and correctness proofs[†]

CHARLES CONSEL AND SIAU CHENG KHOO

Department of Computer Science, Yale University [‡]
New Haven, CT 06520, USA

Abstract

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

The contributions of this work are as follows:

1. We provide a *uniform framework* to defining and proving correct both on-line and off-line partial evaluation.
2. This work required a formal specification of on-line partial evaluation with polyvariant specialization. We define criteria for its correctness with respect to an instrumented standard semantics. As a by-product, on-line partial evaluation appears to be based on a fixpoint iteration process, just like binding-time analysis.
3. We show that binding-time analysis, the preprocessing phase of off-line partial evaluation, is an *abstraction* of on-line partial evaluation. Therefore, its correctness can be proved with respect to on-line partial evaluation, instead of with respect to the standard semantics, as is customarily done.
4. Based on the binding-time analysis, we formally *derive* the specialization semantics for off-line partial evaluation. This strategy ensures the correctness of the resulting semantics.

Capsule Review

Partial evaluation is the process of generating a residual program, given a program and parts of its input. When the residual program is applied to the rest of the input, the same result as by partial evaluation of the original program is obtained. Partial evaluation has a variety of application areas such as program optimization, and even automatic compiler generation.

[†] 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.

[‡] Authors' current address: C. Consel, University of Rennes/Irisa Campus Universitaire de Beaulieu 35042 Rennes, France (e-mail: `consel@irisa.fr`); S.C. Khoo, Department of Information Systems and Computer Science, National University of Singapore, 10 Kent Ridge Crescent, Singapore 0511, Republic of Singapore (e-mail: `khoosc@iscs.nus.sg`).

This paper formally defines a core semantics of a first-order strict functional language which is subsequently instantiated to a standard semantics, an on-line poly-variant partial evaluation semantics, and finally, a mono-variant binding time analysis. The partial evaluation semantics is proven correct by means of the logical relations to an instrumented version of the standard semantics, and the binding time analysis with respect to the on-line partial evaluator. From the formal specifications of the on-line partial evaluator and the binding time analysis, an off-line partial evaluator is derived. The paper is reasonably self-contained, but for full benefit previous knowledge of on-line and off-line partial evaluation is an advantage.

1 Introduction

Partial evaluation is the process of constructing a new program given some original program and a part of its input (Futamura, 1971). It is considered a realization of the S_n^m theorem in recursive function theory (Kleene, 1952). Therefore, a faithful partial evaluator must satisfy the following criterion:

Suppose $P(x, y)$ 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:

$$\forall y \in Y,\ P(c, y) = P_c(y) \qquad (1)$$

In essence, a partial evaluator is a program specializer and is expected to produce more efficient programs (Jones, 1990). 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 (Hannan and Miller, 1989). An off-line partial evaluator performs some analyses before specializing the program; the main analysis performed is *binding-time analysis* (Jones, 1988b). 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 proved until this paper, not even stated formally.

Splitting the partial evaluation process into two phases (binding-time analysis and specialization) makes it possible to shift computations away from the program transformation phase. The specialization becomes simpler and its efficiency is improved (Jones, 1988b; Consel and Danvy, 1993).

In off-line partial evaluation, the specialization phase is primarily driven by the binding-time information, not by concrete values as in on-line partial evaluation. Because binding-time analysis operates on abstract values, it sometimes approximates the binding-time properties of a program. Consequently, off-line partial evaluation may not specialize programs as much as on-line partial evaluation.

An on-line partial evaluator determines the treatment of a program as it gets processed, depending on the available concrete values. Thus, program transformations have to be performed at each specialization of a program. This process can be expensive because it may involve numerous symbolic values and program transformations. In contrast, in off-line partial evaluation, static and dynamic expressions are determined prior to the specialization phase. This information is valid as long as the program is specialized with respect to values that correspond to the description of the input of the program provided to the binding-time analysis. A more detailed comparison between off-line and on-line partial evaluation can be found elsewhere (Consel and Khoo, 1993; Khoo, 1992; Consel and Danvy, 1993).

1.1 Related work

Several works on proving the correctness of partial evaluation have appeared in the literature recently, mostly dedicated to off-line partial evaluation. Nielson and Nielson (1988a, 1988b, 1992) present an algorithm for performing binding-time analysis for a monotyped λ-calculus. This work is based on a non-standard type inference. The correctness of the analysis is proved independently of a given optimization which would use the resulting binding-time information.

Gomard (1992) describes a self-applicable partial evaluator for the untyped λ-calculus. The binding-time analysis is based on non-standard type inference. A monovariant specializer[†] is defined in a denotational setting. A proof of correctness for the partial evaluator is given; it is based on the standard semantics of the language.

Wand (1993) presents a binding-time analysis based on Mogensen's specification of a monovariant specializer for the pure λ-calculus (Mogensen, 1992). The binding-time analysis is proved correct with respect to this specializer. On-line partial evaluation is not addressed.

Launchbury (1990) 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 (1988a). However, since the correctness proof is done with respect to the standard semantics, it provides little insight as to how binding-time properties are related to the partial-evaluation process, and more specifically to that of on-line partial evaluation.

Holst (1989) describes an on-line partial evaluation semantics for a first-order functional language[‡]. This work is based on factorized semantics and abstract interpretation. Holst shows that a partial evaluation semantics is an interpretation of this factorized semantics. Like Holst, we use a factorized semantics to introduce various non-standard semantics. However our work addresses both on-line and off-line partial evaluation. Also, the non-standard semantic specifications we introduce are proved correct.

[†] A specializer is monovariant when each function in a program can have at most one specialized version.
[‡] Thanks to the referee for pointing out Holst's Masters Thesis.

1.2 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.

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

1.2.1 The structure of the semantics

In polyvariant specialization, when a function call is suspended, the function must be specialized. The function call signature characterizes the specialized version of the function. Given all the call signatures, the residual program can be constructed.

This observation prompted us to specify the partial evaluation in terms of collecting interpretation, as described by Hudak and Young (1988) (the resulting semantics is also similar to the minimal function graph (MFG) semantics (Jones and Mycroft, 1986)). As a consequence, just like a collecting interpretation, the semantics consists of two functions. The *local semantic function* (or standard semantic function, using the terminology of Hudak and Young (1988)) describes the partial evaluation of expressions. The *global semantic function* (correspondingly, the collecting interpretation) describes the collection of call signatures.

1.2.2 Uniform approach for defining and proving the correctness of partial evaluation

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

1. **Factorized semantics:** We define a *core semantic* (Jones and Muchnick, 1976; Jones and Nielson, 1990) which consists of semantic rules, and uses some uninterpreted domain names and combinator names (Section 2). This semantics forms the basis for all the semantic specifications defined in the paper. In particular, we define an instrumented semantics that extends the standard semantics to capture all function applications performed during program execution (Section 3). Using other interpretations for domains and combinators, we define the on-line partial evaluation semantics (Section 4), the binding-time analysis and the specialization semantics (Section 5). The advantage of

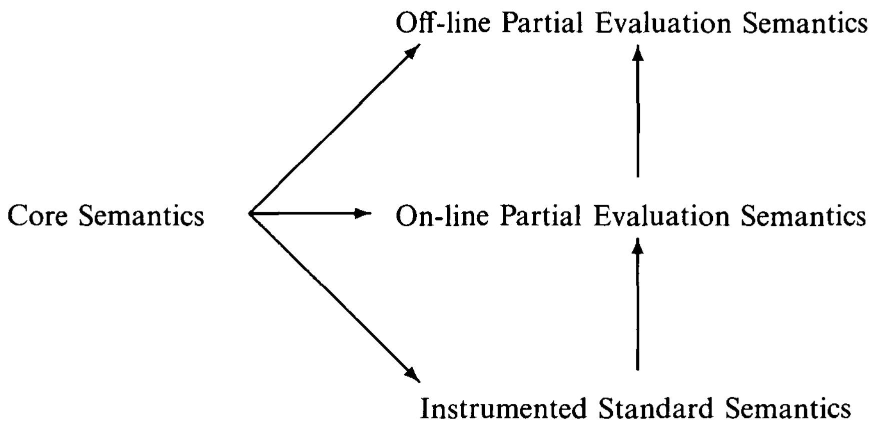
Off-line Partial Evaluation Semantics
Core Semantics
On-line Partial Evaluation Semantics
Instrumented Standard Semantics

Fig. 1. Factorized semantics and logical relations. Factorized semantics enables the instantiation of various semantics of interest from the core semantics. Logical relations relate two adjacent levels of semantics.

a factorized semantics is that different instances can be related at the level of domain definitions and combinator definitions.

2. **Logical relations:** We use the technique of *logical relations* (Abramsky, 1990; Jones and Nielson, 1990; Mizuno and Schmidt, 1990) to prove the correctness of partial evaluation semantics. Logical relations are defined (1) to relate the on-line partial evaluation semantics to the instrumented semantics, and (2) to relate the binding-time analysis to the on-line semantics. Since all these semantic specifications are just different interpretations of the core semantics, their relations can be defined locally by relating their domains and combinators. The resulting proofs thus closely conform to our intuition about the relations between these semantics.

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

1.3 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 $\bot_D$ (called 'bottom'). As is customary, during a computation $\bot_D$ means 'not yet calculated' (Jones and Nielson, 1990). A domain has a binary ordering relation denoted by $\sqsubseteq_D$. The infix least upper bound (l.u.b.) operator for the domain **D** is written $\sqcup_D$; its prefix form, which computes the l.u.b. of a set of elements, is denoted $\bigsqcup_D$. Thus we have that for all $d \in \mathbf{D}$, $\bot_D \sqsubseteq_D d$ and $\bot_D \sqcup_D d = d$. A domain is *flat* if all its elements apart from $\bot$ are incomparable with each other. Domain subscripts are often omitted, as in $\bot \sqcup d$, when they are clear from context.

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

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

A conditional expression is noted '$e_1 \ \to\ e_2\ [\!]\ e_3$', where e_1 is the test, e_2 the consequent and e_3 the alternative.

Angle brackets are used for tupling. If $d = \langle d_1, \ldots, d_n \rangle \in \mathbf{D}_1 \times \cdots \times \mathbf{D}_n$, then for all $i \in \{1, \ldots, n\}$, $d\!\downarrow\! i$ denotes the i-th element (that is, d_i) of d. For convenience, in the context of a smashed product, that is, $d \in \mathbf{D}_1 \otimes \cdots \otimes \mathbf{D}_n$, d^i denotes the i-th element of d. Syntactic objects are consistently enclosed in double brackets, as in $[\![e]\!]$. Square brackets are used for environment updates, as in $env[d/[\![x]\!]]$, which is equivalent to the function $\lambda v\ .\ v = [\![x]\!] \ \to\ d\ [\!]\ env(v)$. 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 from the context. 'New' environments are created by $\perp[d_i/[\![x_i]\!]]$. Similar notations are also used to denote caches, cache updates and new caches, 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 $\tilde{s}$ if it is used in the binding-time analysis of 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 Core semantics

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

The core semantics is defined in Figure 3. It is used as the basis for all the other semantic specifications defined later, and it factors out the common components

$$
\begin{array}{lcll}
c & \in & \textbf{Const} & \text{Constants} \\
x & \in & \textbf{Var} & \text{Variables} \\
p & \in & \textbf{Po} & \text{Primitive Functions} \\
f & \in & \textbf{Fn} & \text{Function Names} \\
e & \in & \textbf{Exp} & \text{Expressions}
\end{array}
$$

$$
\begin{array}{rcl}
e & ::= & c \mid x \mid p\,(e_1,\cdots,e_n) \mid f\,(e_1,\cdots,e_n) \mid \mathit{if}\ e_1\ e_2\ e_3 \\
\textbf{Prog} & ::= & \{f_1(x_1,\cdots,x_n) = e_1\ \ldots\ f_m(x_1,\cdots,x_n) = e_m\,\} \qquad (f_1 \text{ is the main function})
\end{array}
$$

Fig. 2. Syntactic domains of the subject language.œ

of those semantic specifications. This semantics is composed of two valuation functions: $\overline{\mathscr{E}}$ and $\overline{\mathscr{A}}$. Briefly, $\overline{\mathscr{E}}$ defines the standard/abstract semantics (called the *local* semantics) for the language constructs, while $\overline{\mathscr{A}}$ defines a process that globally collects information (called the *global* semantics). The structure of the core semantics is similar to that used in Hudak and Young (1988) for defining collecting interpretation. A similar structure is also used by Sestoft (1985) for defining binding-time analysis.

$$
\begin{array}{ll}
1. & \overline{\mathscr{E}} : \textbf{Exp} \to \mathit{ECont} \quad \mathit{where}\ \ \mathit{ECont} = \overline{\mathit{Env}} \to \mathit{Result}_{\overline{\mathscr{E}}} \\
& \overline{\mathscr{E}}[\![c]\!] = \mathit{Const}_{\overline{\mathscr{E}}}\ [\![c]\!] \\
& \overline{\mathscr{E}}[\![x]\!] = \mathit{VarLookup}_{\overline{\mathscr{E}}}\ [\![x]\!] \\
& \overline{\mathscr{E}}[\![p(e_1,\ldots,e_n)]\!] = \mathit{PrimOp}_{\overline{\mathscr{E}}}\ [\![p]\!](\overline{\mathscr{E}}[\![e_1]\!],\ldots,\overline{\mathscr{E}}[\![e_n]\!]) \\
& \overline{\mathscr{E}}[\![\mathit{if}\ e_1\ e_2\ e_3]\!] = \mathit{Cond}_{\overline{\mathscr{E}}}\ (\overline{\mathscr{E}}[\![e_1]\!],\ \overline{\mathscr{E}}[\![e_2]\!],\ \overline{\mathscr{E}}[\![e_3]\!]) \\
& \overline{\mathscr{E}}[\![f(e_1,\ldots,e_n)]\!] = \mathit{App}_{\overline{\mathscr{E}}}\ [\![f]\!](\overline{\mathscr{E}}[\![e_1]\!],\ldots,\overline{\mathscr{E}}[\![e_n]\!]) \\
& \mathit{where}\ \ \mathit{Const}_{\overline{\mathscr{E}}} : \textbf{Const} \to \mathit{ECont} \\
& \qquad\quad\ \mathit{VarLookup}_{\overline{\mathscr{E}}} : \textbf{Var} \to \mathit{ECont} \\
& \qquad\quad\ \mathit{PrimOp}_{\overline{\mathscr{E}}} : \textbf{Po} \to \mathit{ECont}^n \to \mathit{ECont} \\
& \qquad\quad\ \mathit{Cond}_{\overline{\mathscr{E}}} : \mathit{ECont}^3 \to \mathit{ECont} \\
& \qquad\quad\ \mathit{App}_{\overline{\mathscr{E}}} : \textbf{Fn} \to \mathit{ECont}^n \to \mathit{ECont} \\
2. & \overline{\mathscr{A}} : \textbf{Exp} \to \mathit{ACont} \quad \mathit{where}\ \ \mathit{ACont} = \overline{\mathit{Env}} \to \mathit{Result}_{\overline{\mathscr{A}}} \\
& \overline{\mathscr{A}}[\![c]\!] = \mathit{Const}_{\overline{\mathscr{A}}}\ [\![c]\!] \\
& \overline{\mathscr{A}}[\![x]\!] = \mathit{VarLookup}_{\overline{\mathscr{A}}}\ [\![x]\!] \\
& \overline{\mathscr{A}}[\![p(e_1,\ldots,e_n)]\!] = \mathit{PrimOp}_{\overline{\mathscr{A}}}\ [\![p]\!]\ (\overline{\mathscr{A}}[\![e_1]\!],\ldots,\ \overline{\mathscr{A}}[\![e_n]\!]) \\
& \overline{\mathscr{A}}[\![\mathit{if}\ e_1\ e_2\ e_3]\!] = \mathit{Cond}_{\overline{\mathscr{A}}}\ (\overline{\mathscr{A}}[\![e_1]\!],\ \overline{\mathscr{A}}[\![e_2]\!],\ \overline{\mathscr{A}}[\![e_3]\!])\ (\overline{\mathscr{E}}[\![e_1]\!]) \\
& \overline{\mathscr{A}}[\![f(e_1,\ldots,e_n)]\!] = \mathit{App}_{\overline{\mathscr{A}}}\ [\![f]\!]\ (\overline{\mathscr{A}}[\![e_1]\!],\ldots,\ \overline{\mathscr{A}}[\![e_n]\!])\ (\overline{\mathscr{E}}[\![e_1]\!],\ldots,\ \overline{\mathscr{E}}[\![e_n]\!]) \\
& \mathit{where}\ \ \mathit{Const}_{\overline{\mathscr{A}}} : \textbf{Const} \to \mathit{ACont} \\
& \qquad\quad\ \mathit{VarLookup}_{\overline{\mathscr{A}}} : \textbf{Var} \to \mathit{ACont} \\
& \qquad\quad\ \mathit{PrimOp}_{\overline{\mathscr{A}}} : \textbf{Po} \to \mathit{ACont}^n \to \mathit{ACont} \\
& \qquad\quad\ \mathit{Cond}_{\overline{\mathscr{A}}} : \mathit{ACont}^3 \to \mathit{ECont} \to \mathit{ACont} \\
& \qquad\quad\ \mathit{App}_{\overline{\mathscr{A}}} : \textbf{Fn} \to \mathit{ACont}^n \to \mathit{ECont}^n \to \mathit{ACont}
\end{array}
$$

Fig. 3. Core semantics.

The core semantics is defined by semantic rules. It uses some uninterpreted domain names and combinator names. A semantic specification is defined by providing an interpretation to these domains and combinators. The interpreted domains and combinators for a local semantics $\overline{\mathscr{E}}$ are noted $[\mathit{Result}_{\overline{\mathscr{E}}}\ ;\mathit{Comb}_{\overline{\mathscr{E}}}]$; those for a global semantics $\overline{\mathscr{A}}$ are noted $[\mathit{Result}_{\overline{\mathscr{A}}}\ ;\mathit{Comb}_{\overline{\mathscr{A}}}]$. Figure 4 provides an overview of three levels of semantic specifications, namely, a standard semantics with instrumentation ($\langle\mathscr{E},\mathscr{A}\rangle$), an on-line partial-evaluation semantics ($\langle\widehat{\mathscr{E}},\widehat{\mathscr{A}}\rangle$), and a binding-time analy-

$$\begin{tikzcd}
{[Result_{\tilde{\mathcal{E}}} \; ; \; Comb_{\tilde{\mathcal{E}}}]} & \langle \tilde{\mathcal{E}}, \tilde{\mathcal{A}} \rangle & {[Result_{\tilde{\mathcal{A}}} \; ; \; Comb_{\tilde{\mathcal{A}}}]} \\
{[Result_{\hat{\mathcal{E}}} \; ; \; Comb_{\hat{\mathcal{E}}}]} \arrow[u] & \langle \hat{\mathcal{E}}, \hat{\mathcal{A}} \rangle & {[Result_{\hat{\mathcal{A}}} \; ; \; Comb_{\hat{\mathcal{A}}}]} \arrow[u] \\
{[Result_{\mathcal{E}} \; ; \; Comb_{\mathcal{E}}]} \arrow[u] & \langle \mathcal{E}, \mathcal{A} \rangle & {[Result_{\mathcal{A}} \; ; \; Comb_{\mathcal{A}}]} \arrow[u]
\end{tikzcd}$$

Fig. 4. Relations between three levels of evaluation.

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

sis $(\langle \tilde{\mathcal{E}}, \tilde{\mathcal{A}} \rangle)$. These semantic specifications are obtained via instantiation of the core semantics described in Figure 3. We can relate two adjacent semantic specifications simply by relating their domains and combinators. These relations represent the key component for proving the correctness of these specifications. In the following sections, we examine how each semantic specification can be instantiated from the core semantics. We then define the relation between two adjacent specifications, and use it to prove their correctness.

3 Standard and instrumented semantics

3.1 Semantic specifications

In Figure 5 the core semantics is instantiated to define the standard semantics of the language. As is customary, we will omit summand projections and injections. Only interpretation of the valuation function $\overline{\mathcal{E}}$ is provided since the definition of standard semantics does not require collecting information globally. For a function f, '*strict* f' is a function just like f except that it is strict in all its arguments. Function $\mathcal{K}$ converts a syntactic constant into its corresponding denotable value (i.e. a basic value in domain **Values**). Function $\mathcal{K}_P$ defines the semantics of primitive operations. We require these primitive operations to be continuous, while leaving the detailed definitions to the actual implementation.

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* that 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 6.

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

Fig. 5. Standard semantics.

To illustrate the functionality of the instrumented semantics, we apply $\mathscr{A}$ to the following exponentiation function:

```
f(b, e) = if (e > 0) then (b * f(b, e - 1)) else 1
```

Function `f` raises the value `b` to the power of `e`. For a function call `f(2, 3)`, the function $\mathscr{A}$ computes the following caches until a fixpoint is reached.

Iteration #	Cache
1	$[\{\langle 2,3\rangle\}/f]$
2	$[\{\langle 2,3\rangle,\langle 2,2\rangle\}/f]$
3	$[\{\langle 2,3\rangle,\langle 2,2\rangle,\langle 2,1\rangle\}/f]$
4	$[\{\langle 2,3\rangle,\langle 2,2\rangle,\langle 2,1\rangle,\langle 2,0\rangle\}/f]$
5	$[\{\langle 2,3\rangle,\langle 2,2\rangle,\langle 2,1\rangle,\langle 2,0\rangle\}/f]$

[§] Notice that powerset, instead of powerdomain, is used to model the content of the cache. This avoids some technical complication incurred in the correctness proof, as discussed in Hudak and Young (1988).

- Semantic Domains

$$\begin{array}{lll} v & \in & \mathit{Result}_{\mathscr{E}} = \mathbf{Values} = \mathit{as\ in\ Figure\ 5} \\ \rho & \in & \mathit{VarEnv} = \mathit{as\ in\ Figure\ 5} \\ \phi & \in & \mathit{FunEnv} = \mathit{as\ in\ Figure\ 5} \\ \sigma & \in & \mathit{Result}_{\mathscr{A}} = \mathbf{Cache}_{\mathscr{A}} = \mathbf{Fn} \rightarrow \mathscr{P}(\mathbf{Values}^n) \end{array}$$

- Valuation Functions

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

$$\begin{array}{llcl} \multicolumn{4}{l}{\mathscr{E}_{Prog}\ [\![\{\ f_i(x_1,\ldots,x_n) = e_i\}]\!]\langle v_1,\ldots,v_n\rangle = h(\bot[\{\langle v_1,\ldots,v_n\rangle\}/f_1])} \\ \mathit{whererec} & h(\sigma) & = & \sigma \sqcup h(\bigsqcup\{\mathscr{A}\ [\![e_i]\!]\ (\bot[v_k/x_k])\phi \mid \langle v_1,\ldots,v_n\rangle \in \sigma[\![f_i]\!], \\ & & & \qquad \forall [\![f_i]\!] \in \mathit{Dom}(\sigma)\}) \\ & \phi & = & \bot[\mathit{strict}\ (\lambda(v_1,\ldots,v_n)\ .\ \mathscr{E}\ [\![e_i]\!]\ (\bot[v_k/x_k])\ \phi)/f_i] \end{array}$$

$$\mathscr{E} = \overline{\mathscr{E}}$$

$$\mathscr{A} = \overline{\mathscr{A}}$$

- Combinator Definitions

$$\begin{array}{l} \mathit{Const}_{\mathscr{E}}\ [\![c]\!] = \mathit{as\ in\ Figure\ 5} \\ \mathit{VarLookup}_{\mathscr{E}}\ [\![x]\!] = \mathit{as\ in\ Figure\ 5} \\ \mathit{PrimOp}_{\mathscr{E}}\ [\![p]\!]\ (k_1,\ldots,k_n) = \mathit{as\ in\ Figure\ 5} \\ \mathit{Cond}_{\mathscr{E}}\ (k_1,\ k_2,\ k_3) = \mathit{as\ in\ Figure\ 5} \\ \mathit{App}_{\mathscr{E}}\ [\![f]\!]\ (k_1,\ldots,k_n) = \mathit{as\ in\ Figure\ 5} \end{array}$$

$$\begin{array}{l} \mathit{Const}_{\mathscr{A}}\ [\![c]\!] = \lambda(\rho,\phi)\ .\ (\lambda f\ .\ \{\}) \\ \mathit{VarLookup}_{\mathscr{A}}\ [\![x]\!] = \lambda(\rho,\phi)\ .\ (\lambda f\ .\ \{\}) \\ \mathit{PrimOp}_{\mathscr{A}}\ [\![p]\!]\ (a_1,\ldots,a_n) = \lambda(\rho,\phi)\ .\ \bigsqcup_{i=1}^{n} a_i(\rho,\phi) \\ \mathit{Cond}_{\mathscr{A}}\ (a_1,\ a_2,\ a_3)\ k_1 = \lambda(\rho,\phi)\ .\ \ a_1(\rho,\phi)\ \sqcup\ (k_1(\rho,\phi)\ \rightarrow\ a_2(\rho,\phi)\ [\!]\ a_3(\rho,\phi)) \\ \mathit{App}_{\mathscr{A}}\ [\![f]\!]\ (a_1,\ldots,\ a_n)(\ k_1,\ldots,\ k_n) = \lambda(\rho,\phi)\ .\ \ \bigsqcup_{i=1}^{n} a_i(\rho,\phi) \sqcup\ \ (\exists i \in \{1,\ldots,n\}\ \mathit{s.t.}\ v_i = \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \bot \rightarrow (\lambda f\ .\ \{\}) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\!]\ \bot[\{\langle v_1,\ldots,v_n\rangle\}/f]) \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathit{where}\ \ v_i = k_i(\rho,\phi)\quad \forall\ i\ \in\ \{1,\cdots,n\} \end{array}$$

Fig. 6. Instrumented semantics capturing function calls.

3.2 *Correctness of instrumentation*

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

Lemma 1

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

1. For any expression e in P, if a non-trivial function call occurring in e is performed when e is evaluated, then $\mathscr{A}$ 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 $v''_i = \mathscr{E}[\![e'_i]\!](\bot[v'_k/x_k],\phi)\ \forall i \in \{1,\ldots,n\}$, then $\langle v''_1,\ldots,v''_n\rangle \in \sigma[\![f_j]\!]$, provided $v''_i \neq \bot, \forall i \in \{1,\ldots,n\}$.

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 $\mathscr{A}$' 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 $\mathscr{E}_{Prog}$. If $\langle v'_1, \ldots, v'_n \rangle \in \sigma[\![f_i]\!]$, then $\mathscr{A}$ will be called to collect the 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 $v''_i \neq \bot, \forall i \in \{1, \ldots, n\}$. □

Using this lemma, the following theorem formalizes the correctness of instrumentation.

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

Proof : From Lemma 1, and by noticing that since none of the initial input is bottom, the initial call to f_1 is captured in the cache (by the definition of $\mathscr{E}_{Prog}$) . □

4 On-line partial evaluation semantics

In this section, we instantiate the core semantics to obtain on-line partial evaluation. A key component for this instantiation is the partial-evaluation algebra defined in Section 4.1. On-line partial evaluation semantics is presented in Section 4.2. Finally, Section 4.3 defines a relation between this semantics and the instrumented semantics. This makes it possible to state and prove the correctness of the on-line partial evaluation semantics.

4.1 Partial-evaluation algebra

At the standard evaluation level, primitive operations can be captured by the algebra [**Values**; **O**] where **Values** is the domain of basic values and **O** is the set of primitive functions. At the on-line partial evaluation level, because it is a program transformation process, primitives operate on syntactic constants instead of values. We denote the set of syntactic constants by **Const**. Furthermore, because a program is processed with a partial input, a primitive might be invoked with some non-constant arguments (i.e. an expression which is not a constant), and thus yield a non-constant result. These observations are captured by the *partial-evaluation algebra*: an abstraction of the standard algebra [**Values**; **O**].

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

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

$$\begin{array}{lcl} \hat{\tau} & : & \mathbf{Values} \rightarrow \widehat{\mathbf{Values}} \\ \hat{\tau}(x) & = & (x = \bot_{Values}) \rightarrow \bot_{\widehat{Values}} \; [] \; \mathscr{K}^{-1}(x) \end{array}$$

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

$$\begin{array}{l} \hat{p} : \widehat{\mathbf{Values}}^{n} \rightarrow \widehat{\mathbf{Values}} \\ \hat{p} = \lambda\ (\hat{d}_1, \cdots, \hat{d}_n)\ .\ \exists i \in \{1, \ldots, n\}\ s.t.\ \hat{d}_i = \bot_{\widehat{Values}} \rightarrow \bot_{\widehat{Values}} \\ \qquad\qquad [] \; (\bigwedge_{i=1}^{n} (\hat{d}_i \in \mathbf{Const}) \rightarrow \hat{\tau}(\mathscr{K}_p[\![p]\!](d_1, \cdots, d_n)) \; [] \; \top_{\widehat{Values}}) \\ \qquad where\ d_i = \mathscr{K}\ [\![\hat{d}_i]\!] \quad \forall\ i\ \in\ \{1, \ldots, n\} \end{array}$$

where $\mathscr{K}^{-1}$ is the monotonic semantics function that converts a basic value to its textual representation (i.e. a syntactic constant). Because **Values** is the sum of basic domains, $\hat{\tau}$ is actually a family of abstraction functions indexed by the summands. Domain $\widehat{\mathbf{Values}}$ – refered to as the *partial-evaluation domain* – is constructed by adding elements $\bot_{\widehat{Values}}$ and $\top_{\widehat{Values}}$ to the set of syntactic constants **Const** such that $\bot_{\widehat{Values}}$ and $\top_{\widehat{Values}}$ are respectively weaker and stronger than all the elements of **Const**. Value $\bot_{\widehat{Values}}$ corresponds to $\bot_{Values}$, while the value $\top_{\widehat{Values}}$ represents a non-constant value.

Operators in the partial-evaluation algebra define the partial-evaluation semantics of primitive operations. This semantics is an abstraction of the standard semantics. Indeed, when called with constant values, the partial-evaluation semantics of a primitive operation corresponds to the standard semantics of this primitive operation. However, if some of the arguments in the primitive call are non-constant at partial-evaluation time, the value $\top_{\widehat{Values}}$ is produced. This represents a value unknown at partial-evaluation time. The abstract primitive operations satisfy the following safety criterion:

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

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

The relation between [**Values**; **O**] and the partial-evaluation algebra can be succinctly described by a logical relation (Nielson, 1989; Jones and Nielson, 1990) $\sqsubseteq_{\hat{\tau}}$ defined as follows:

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

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

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

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

$$\begin{array}{lcl} \tilde{\tau} & : & \widehat{\textbf{Values}} \rightarrow \widetilde{\textbf{Values}} \\ \tilde{\tau}(x) & = & x = \perp_{\widehat{Values}} \rightarrow \perp_{\widetilde{Values}} \\ & & \quad [\!] \; (x \in \textbf{Const} \rightarrow Static \; [\!] \; Dynamic). \end{array}$$

Notice that, not only is the domain $\widetilde{\textbf{Values}}$ used in the binding-time analysis (Section 5), but binding-time values are also used to drive the transformation of function calls (Page 473).

4.2 *Semantic specification*

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

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

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

This structure is similar to the notion of *symbolic values* used in the on-line partial evaluator FUSE (Weise and Ruf, 1990). In Consel and Khoo (1991), this representation is generalized to a tuple of values that captures user-defined static properties, in addition to a residual expression and a value in the partial-evaluation domain.

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 unfolding analysis (e.g. Sestoft, 1988). To capture this piece of decision making, we introduce the notion of *filters*.

[¶] 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*.

- Semantic Domains

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

- Valuation Functions

$$\begin{aligned}
&\widehat{\mathscr{E}}_{Prog} : \mathbf{Prog} \to \mathbf{Res}^n \to \mathbf{Prog}_\perp \\
&\quad \widehat{\mathscr{E}}_{Prog}\, [\![\{f_i(x_1,\ldots,x_n)=e_i\}]\!]\, (\hat{r}_1,\ldots,\hat{r}_n) = \mathit{MkProg}\ (\hat{h}(\perp[\{\langle \mathrm{s},\hat{r}_1,\ldots,\hat{r}_n\rangle\}/f_1]))\hat{\phi} \\
&\qquad \mathit{whererec}\ \ \hat{h}(\hat{\sigma}) = \hat{\sigma} \sqcup \hat{h}(\textstyle\bigsqcup\{\widehat{\mathscr{A}}\ [\![e_i]\!]\ (\perp[\hat{r}'_k/x_k],\hat{\phi}) \mid \langle -,\hat{r}'_1,\ldots,\hat{r}'_n\rangle \in \hat{\sigma}[\![f_i]\!], \forall [\![f_i]\!] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \in \mathit{Dom}(\hat{\sigma})\}) \\
&\qquad\qquad\qquad \hat{\phi} = \perp[\mathit{strict}\ (\lambda(\hat{r}_1,\ldots,\hat{r}_n)\,.\,\widehat{\mathscr{E}}\ [\![e_i]\!]\ (\perp[\hat{r}_k/x_k],\hat{\phi}))/f_i] \\
&\widehat{\mathscr{E}} = \overline{\mathscr{E}} \\
&\widehat{\mathscr{A}} = \overline{\mathscr{A}}
\end{aligned}$$

- *MkProg* Definition

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

- Local Combinator Definitions

$$\begin{aligned}
&\mathit{Const}_{\widehat{\mathscr{E}}}\ [\![c]\!] = \lambda(\hat{\rho},\hat{\phi})\,.\,\widehat{\mathscr{K}}\ [\![c]\!] \\
&\mathit{VarLookup}_{\widehat{\mathscr{E}}}\ [\![x]\!] = \lambda(\hat{\rho},\hat{\phi})\,.\,\hat{\rho}\ [\![x]\!] \\
&\mathit{PrimOp}_{\widehat{\mathscr{E}}}\ [\![p]\!]\ (\hat{k}_1,\ldots,\hat{k}_n) = \lambda(\hat{\rho},\hat{\phi})\,.\,\widehat{\mathscr{K}}_P\ [\![p]\!]\ (\hat{k}_1(\hat{\rho},\hat{\phi}),\ldots,\ \hat{k}_n(\hat{\rho},\hat{\phi})) \\
&\mathit{Cond}_{\widehat{\mathscr{E}}}\ (\hat{k}_1,\hat{k}_2,\hat{k}_3) = \lambda(\hat{\rho},\hat{\phi})\,.\ \ (\hat{r}_1{\downarrow}2 \in \mathbf{Const}) \ \to\ (\mathscr{K}(\hat{r}_1{\downarrow}2) \to \hat{r}_2,\hat{r}_3) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\!]\ \langle [\![\mathit{if}\ \hat{r}_1{\downarrow}1\ \hat{r}_2{\downarrow}1\ \hat{r}_3{\downarrow}1]\!], \hat{r}_2{\downarrow}2 \sqcup \hat{r}_3{\downarrow}2\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathit{where}\ \ \hat{r}_i = \hat{k}_i(\hat{\rho},\hat{\phi}) \quad \forall i \in \{1,\ 2,\ 3\} \\
&\mathit{App}_{\widehat{\mathscr{E}}}\ [\![f]\!]\ (\hat{k}_1,\ldots,\hat{k}_n) = \lambda(\hat{\rho},\hat{\phi})\,.\ \ (\mathit{Ft}\ [\![f]\!]){\downarrow}1\ \ (\widehat{bt}(\hat{r}_1),\ldots,\widehat{bt}(\hat{r}_n)) = \mathrm{u} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to\ \hat{\phi}\ [\![f]\!]\ (\hat{r}_1,\ldots,\hat{r}_n)\ [\!]\ \langle [\![f_{sp}(e''_1,\ldots,e''_k)]\!], \top_{\widehat{Values}}\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{where}\ \ \hat{r}_i = \hat{k}_i(\hat{\rho},\hat{\phi}) \quad \forall i \in \{1,\ldots,n\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad f_{sp} = \mathit{SpName}([\![f]\!],\hat{r}'_1,\ldots,\hat{r}'_n) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \langle e''_1,\ldots,e''_k\rangle = \mathit{ResidArgs}\ ([\![f]\!],\ \langle b_1,\ldots,b_n\rangle, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle \hat{r}_1{\downarrow}1,\ldots,\hat{r}_n{\downarrow}1\rangle) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \langle \hat{r}'_1,\ldots,\hat{r}'_n\rangle = \mathit{SpPat}\ ([\![f]\!],\langle \hat{r}_1,\ldots,\hat{r}_n\rangle, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \langle b_1,\ldots,b_n\rangle) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \langle b_1,\ldots,b_n\rangle = (\mathit{Ft}\ [\![f]\!]){\downarrow}2\ (\widehat{bt}(\hat{r}_1),\ldots,\widehat{bt}(\hat{r}_n))
\end{aligned}$$

- Primitive Functions

$$\begin{aligned}
&\widehat{\mathscr{K}}\ :\ \mathbf{Const} \to \mathbf{Res} \\
&\quad \widehat{\mathscr{K}}\ [\![c]\!] = \langle [\![c]\!], [\![c]\!]\rangle \\[1em]
&\widehat{\mathscr{K}}\ :\ \mathbf{Po} \to \mathbf{Res}^n \to \mathbf{Res} \\
&\quad \widehat{\mathscr{K}}_P\ [\![p]\!]\ (\langle e'_1,\hat{v}_1\rangle,\ldots,\langle e'_n,\hat{v}_n\rangle) = \ \hat{v} = \perp_{\widehat{Values}} \to \langle \perp_{Exp}, \perp_{\widehat{Values}}\rangle \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad [\!]\ (\hat{v} \in \widehat{\mathbf{Const}} \to \langle \hat{v},\hat{v}\rangle\ [\!]\ \langle [\![p(e'_1,\cdots,e'_n)]\!],\hat{v}\rangle) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{where}\ \ \hat{v} = \hat{p}(\hat{v}_1,\cdots,\hat{v}_n)
\end{aligned}$$

Fig. 7. On-line partial evaluation semantics – Part 1.

Global Combinator Definitions

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

Fig. 8. On-line partial evaluation semantics – Part 2

A filter specification is associated to each user-defined function in a subject program. A filter consists of a pair of *strict* and *continuous* functions. Both functions are passed the binding-time value of the arguments in a function call. 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 constant arguments are considered for propagation. The functionality of a filter is $(\widehat{\mathbf{Values}}^n \rightarrow \mathbf{T}) \times (\widehat{\mathbf{Values}}^n \rightarrow \widehat{\mathbf{Values}}^n)$ where $\widehat{\mathbf{Values}}$ is the binding-time domain and the domain **T** contains two values: u and s, which stand for unfolding and specializing respectively. This strategy has been developed for the partial evaluator Schism (Consel, 1988, 1993b).

Domain **T** is ordered as follows: $\mathrm{u} \sqsubset \mathrm{s}$. This ordering reflects our intuition about the termination behaviour 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 Sestoft (1988), 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. As an example of a filter, consider the exponentiation function given in Section 3.1. Assume that we want to unconditionally suspend calls to this function, and that it should only be specialized with respect to the second argument (when it is a constant). Such conditions can be expressed by the following filter.

$$Ft\,[\![\mathrm{f}]\!] =\ \langle\ \lambda\langle b_1,b_2\rangle.\mathrm{s},\ \ \lambda\langle b_1,b_2\rangle.\langle Dynamic, b_2\rangle\ \rangle.$$

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.[‖]

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

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

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

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

Lastly, it is worth noticing that, just like a binding-time analysis, $\widehat{\mathscr{E}}_{Prog}$ 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 (e.g. Sestoft, 1985). The cache produced will be used by *MkProg* to generate the residual code for all the specialized functions. The co-domain of this function (and Function $\widehat{\mathscr{E}}_{Prog}$) is lifted to account for the fact that partial evaluation may not terminate and thus not produce a residual program.

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

1. *SpName* produces a specialized function name from the original function name and the argument signature. It has the functionality:

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

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

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

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

$$\begin{aligned} SpPat = \ & \lambda(f, \langle \hat{r}_1, \ldots, \hat{r}_n \rangle, \langle b_1, \ldots, b_n \rangle) \,.\, \langle \hat{r}'_1, \ldots, \hat{r}'_n \rangle \\ & \textit{where } \forall\, i \in \{1, \ldots, n\}, \\ & \quad r'_i = \ b_i = Static \ \rightarrow \hat{r}_i \\ & \qquad\quad [\!]\ (b_i = Dynamic \ \rightarrow \ \langle x_i, \top_{\widehat{Values}} \rangle\ [\!]\ \langle \perp_{Exp}, \perp_{\widehat{Values}} \rangle) \end{aligned}$$

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

We state here without proof the following two lemmas:

Lemma 2
$\widehat{\mathscr{E}}$ is continuous in all its arguments.

Lemma 3
$\widehat{\mathscr{A}}$ is continuous in all its arguments.

As an example of a cache produced by $\widehat{\mathscr{A}}$, consider the exponentiation function given in Section 3.1. Assume that f includes the following filter (previously discussed),

$$Ft\,[\![\mathtt{f}]\!] = \ \langle\ \lambda\langle b_1, b_2 \rangle.\mathtt{s},\ \ \lambda\langle b_1, b_2 \rangle.\langle Dynamic, b_2 \rangle\ \rangle.$$

Specializing function f with respect to $\langle\langle \mathtt{x}, \top_{\widehat{Values}} \rangle, \langle [\![3]\!], 3 \rangle\rangle$ produces the following caches until a fixpoint is reached.

Iteration #	Cache
1	$[\{\langle \mathtt{s}, \langle [\![\mathtt{x}_1]\!], \top_{\widehat{Values}} \rangle, \langle [\![3]\!], 3 \rangle \rangle\}/f]$
2	$[\{\langle \mathtt{s}, \langle [\![\mathtt{x}_1]\!], \top_{\widehat{Values}} \rangle, \langle [\![3]\!], 3 \rangle \rangle, \langle \mathtt{s}, \langle [\![\mathtt{x}_2]\!], \top_{\widehat{Values}} \rangle, \langle [\![2]\!], 2 \rangle \rangle\}/f]$
3	$[\{\langle \mathtt{s}, \langle [\![\mathtt{x}_1]\!], \top_{\widehat{Values}} \rangle, \langle [\![3]\!], 3 \rangle \rangle,$ $\langle \mathtt{s}, \langle [\![\mathtt{x}_2]\!], \top_{\widehat{Values}} \rangle, \langle [\![2]\!], 2 \rangle \rangle, \langle \mathtt{s}, \langle [\![\mathtt{x}_3]\!], \top_{\widehat{Values}} \rangle, \langle [\![1]\!], 1 \rangle \rangle\}/f]$
4	$[\{\langle \mathtt{s}, \langle [\![\mathtt{x}_1]\!], \top_{\widehat{Values}} \rangle, \langle [\![3]\!], 3 \rangle \rangle, \langle \mathtt{s}, \langle [\![\mathtt{x}_2]\!], \top_{\widehat{Values}} \rangle, \langle [\![2]\!], 2 \rangle \rangle,$ $\langle \mathtt{s}, \langle [\![\mathtt{x}_3]\!], \top_{\widehat{Values}} \rangle, \langle [\![1]\!], 1 \rangle \rangle, \langle \mathtt{s}, \langle [\![\mathtt{x}_4]\!], \top_{\widehat{Values}} \rangle, \langle [\![0]\!], 0 \rangle \rangle\}/f]$
5	$[\{\langle \mathtt{s}, \langle [\![\mathtt{x}_1]\!], \top_{\widehat{Values}} \rangle, \langle [\![3]\!], 3 \rangle \rangle, \langle \mathtt{s}, \langle [\![\mathtt{x}_2]\!], \top_{\widehat{Values}} \rangle, \langle [\![2]\!], 2 \rangle \rangle,$ $\langle \mathtt{s}, \langle [\![\mathtt{x}_3]\!], \top_{\widehat{Values}} \rangle, \langle [\![1]\!], 1 \rangle \rangle, \langle \mathtt{s}, \langle [\![\mathtt{x}_4]\!], \top_{\widehat{Values}} \rangle, \langle [\![0]\!], 0 \rangle \rangle\}/f]$

where the variables $\mathtt{x}_1, \ldots, \mathtt{x}_4$ are new variables generated by the function *SpPat* during the four call suspensions. Using the final cache, the function *MkProg* will generate the following four specialized functions:

$$\begin{aligned} \mathtt{f}_1(\mathtt{x}_1) &= \mathtt{x}_1 * \mathtt{f}_2(\mathtt{x}_1 - 1) \\ \mathtt{f}_2(\mathtt{x}_2) &= \mathtt{x}_2 * \mathtt{f}_3(\mathtt{x}_2 - 1) \\ \mathtt{f}_3(\mathtt{x}_3) &= \mathtt{x}_3 * \mathtt{f}_4(\mathtt{x}_3 - 1) \\ \mathtt{f}_4(\mathtt{x}_4) &= 1 \end{aligned}$$

where $\mathtt{f}_1, \ldots, \mathtt{f}_4$ are the specialized-function names generated by function *SpName*.

4.3 Correctness of partial evaluation semantics

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

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

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

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

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

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

where $\phi \in FunEnv$ and $\hat{\phi} \in \widehat{FunEnv}$ are two function environments for P defined by the standard semantics and the partial evaluation semantics, respectively. Also, $\rho \in VarEnv$, and $\hat{\rho} \in \widehat{VarEnv}$ is defined as:

$$\hat{\rho} = \lambda\ [\![x]\!]\ .\ \langle\hat{\tau}(\rho[\![x]\!]), \hat{\tau}(\rho[\![x]\!])\rangle$$

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

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

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

- The proofs for $Const_{\mathscr{E}}$ and $VarLookup_{\mathscr{E}}$ are easy, and thus omitted.
- $PrimOp_{\mathscr{E}}$: This is done by structural induction and a case analysis over all the possible argument values of the primitive.
- $Cond_{\mathscr{E}}$: This is done by structural induction and a case analysis over the possible values produced by $\hat{k}_1(\hat{\rho}, \hat{\phi})$.

- $App_{\mathscr{E}}$: For any user-defined function f, suppose that all corresponding arguments of $App_{\mathscr{E}}$ and $App_{\widehat{\mathscr{E}}}$ are related by $\widehat{\mathscr{R}}$. Let $v_i = k_i(\rho, \phi)$ and $\hat{r}_i = \hat{k}_i(\hat{\rho}, \hat{\phi})$ $\forall i \in \{1, \ldots, n\}$. We have $\forall i \in \{1, \ldots, n\}$, $v_i \; \widehat{\mathscr{R}}_{Result_{\widehat{\mathscr{E}}}} \; \hat{r}_i$. Since both ϕ and $\hat{\phi}$ only contain strict functions, $\widehat{\mathscr{R}}$ also holds when some of the arguments are bottom. On the other hand, under the condition that all function applications are unfolded, $App_{\widehat{\mathscr{E}}}[\![f]\!](\hat{k}_1, \ldots, \hat{k}_n)$ is reduced to $\hat{\phi}\ [\![f]\!]\ (\hat{r}_1, \ldots, \hat{r}_n)$. From the supposition that $\phi \; \widehat{\mathscr{R}}_{FunEnv} \; \hat{\phi}$, we have

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

Hence, $App_{\mathscr{E}} \; \widehat{\mathscr{R}} \; App_{\widehat{\mathscr{E}}}$.

This concluses the proof. □

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

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

The safety criterion described in the beginning of Section 1 (Equation 1) can be expressed for our semantic specification as follows: Partial evaluation of a program with input $\langle \hat{r}_1, \ldots, \hat{r}_n \rangle$ 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 $\langle \hat{r}_1, \ldots, \hat{r}_n \rangle$. This assumes that the function that generates the residual program from the cache is correct.

Hence, the correctness of the partial-evaluation semantics can be shown by relating the local and global semantics to their respective counterpart in the instrumented semantics. That is, we define a relation $\widehat{\mathscr{R}^{\mathscr{E}}}$ relating $\mathscr{E}$ and $\widehat{\mathscr{E}}$, and a logical relation $\widehat{\mathscr{R}^{\mathscr{A}}}$ relating $\mathscr{A}$ and $\widehat{\mathscr{A}}$. Notice that $\widehat{\mathscr{R}^{\mathscr{E}}}$ relates the results v and r computed by $\mathscr{E}$ and $\widehat{\mathscr{E}}$ respectively. Since $\hat{r} = \langle e, \hat{v} \rangle \in (\mathbf{Exp} \times \widehat{\mathbf{Values}})$, $\widehat{\mathscr{R}^{\mathscr{E}}}$ is composed of two relations, $\widehat{\mathscr{R}^{\mathscr{E}_1}}$ and $\widehat{\mathscr{R}^{\mathscr{E}_2}}$, that relate a concrete value v to e and $\hat{v}$ respectively. It turns out that the correctness of $\widehat{\mathscr{R}^{\mathscr{E}_1}}$ depends on that of $\widehat{\mathscr{R}^{\mathscr{A}}}$. At the same time, the correctness of $\widehat{\mathscr{R}^{\mathscr{A}}}$ depends on the correctness of $\widehat{\mathscr{R}^{\mathscr{E}_2}}$. Therefore, we shall prove the correctness of $\widehat{\mathscr{R}^{\mathscr{E}_2}}$, then that of $\widehat{\mathscr{R}^{\mathscr{A}}}$, and finally that of $\widehat{\mathscr{R}^{\mathscr{E}_1}}$. Lastly, we combine the result of $\widehat{\mathscr{R}^{\mathscr{E}_2}}$ and $\widehat{\mathscr{R}^{\mathscr{E}_1}}$ to express the correctness of $\widehat{\mathscr{R}^{\mathscr{E}}}$.

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

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

https://doi.org/10.1017/S0956796800001453 Published online by Cambridge University Press

Definition 2 (Relation $\widehat{\mathscr{R}^{\mathscr{E}_2}}$)

$\widehat{\mathscr{R}^{\mathscr{E}_2}}$ is a logical relation between domains of $\mathscr{E}$ and $\widehat{\mathscr{E}}$ defined by:

$$v \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{\widehat{Result_{\mathscr{E}}}} \; \hat{r} \Leftrightarrow v \sqsubseteq_{\widehat{\tau}} \hat{r}{\downarrow}2$$

$$\rho \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{VarEnv} \; \hat{\rho} \Leftrightarrow Dom(\rho) = Dom(\hat{\rho}) \wedge \forall [\![x]\!] \in Dom(\rho),\; \rho[\![x]\!] \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{\widehat{Result_{\mathscr{E}}}} \; \hat{\rho}[\![x]\!]$$

$$\phi \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{FunEnv} \; \hat{\phi} \Leftrightarrow Dom(\phi) = Dom(\hat{\phi}) \wedge \forall [\![f]\!] \in Dom(\phi),\; \forall i \in \{1,\ldots,n\}, \forall v_i \in \textbf{Values}, \forall \hat{r}_i \in \textbf{Res},$$
$$\bigwedge_{i=1}^{n} (v_i \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{\widehat{Result_{\mathscr{E}}}} \; \hat{r}_i) \;\Rightarrow\; \phi[\![f]\!](v_1,\ldots,v_n) \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{\widehat{Result_{\mathscr{E}}}} \; \hat{\phi}[\![f]\!](\hat{r}_1,\ldots,\hat{r}_n)$$

$$\langle d_1, d_2 \rangle \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{D_1 \times D_2} \; \langle \hat{d}_1, \hat{d}_2 \rangle \Leftrightarrow d_1 \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{D_1} \; \hat{d}_1 \;\wedge\; d_2 \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{D_2} \; \hat{d}_2$$

$$f \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{D_1 \to D_2} \; \hat{f} \Leftrightarrow \forall d \in D_1, \forall \hat{d} \in \widehat{D}_1,\; d \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{D_1} \; \hat{d} \Rightarrow f(d) \; \widehat{\mathscr{R}}^{\mathscr{E}_2}_{D_2} \; \hat{f}(\hat{d}).$$

Lemma 4

Given a program P, let ϕ and $\hat{\phi}$ be the two function environments for P defined by the standard and the partial evaluation semantics, respectively. Then $\phi \; \widehat{\mathscr{R}^{\mathscr{E}_2}} \; \hat{\phi}$.

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

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

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):

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

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

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

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

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

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

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

- $Const_{\mathscr{E}}$: $\widehat{\mathscr{R}^{\mathscr{E}_2}}$ is true trivially by comparing $\mathscr{K}$ and $\widehat{\mathscr{K}}$.
- $VarLookup_{\mathscr{E}}$: $\widehat{\mathscr{R}^{\mathscr{E}_2}}$ is true since $\rho \; \widehat{\mathscr{R}^{\mathscr{E}_2}} \; \hat{\rho}$.
- $PrimOp_{\mathscr{E}}$: $PrimOp_{\mathscr{E}} \; \widehat{\mathscr{R}^{\mathscr{E}_2}} \; PrimOp_{\widehat{\mathscr{E}}}$ holds by structural induction and a case analysis over the values produced by $PrimOp_{\widehat{\mathscr{E}}}$. Proof is omitted.
- $Cond_{\mathscr{E}}$: $Cond_{\mathscr{E}} \; \widehat{\mathscr{R}^{\mathscr{E}_2}} \; Cond_{\widehat{\mathscr{E}}}$ holds by structural induction and a case analysis over the values produced by $\hat{k}_1(\hat{\rho}, \hat{\phi}_n)$.

$App_{\mathscr{E}}$: For any user-defined function f, all the corresponding arguments of $App_{\mathscr{E}}$ and $App_{\widehat{\mathscr{E}}}$ are related by $\widehat{\mathscr{R}^{\mathscr{E}_2}}$ (by structural induction hypothesis). It is easy to show that $\widehat{\mathscr{R}^{\mathscr{E}_2}}$ holds when the function is specialized, since the top element $\top_{\widehat{Values}}$ is returned. For the case when the function is unfolded, $App_{\widehat{\mathscr{E}}}[\![f]\!]\ (\hat{k}_1,\dots,\hat{k}_n)$ is reduced to $\hat{\phi}_n[\![f]\!]\ (\hat{r}_1,\dots,\hat{r}_n)$, while $App_{\mathscr{E}}[\![f]\!]\ (k_1,\dots,k_n)$ is reduced to $\phi_n[\![f]\!]\ (v_1,\dots,v_n)$. Since $\phi_n\ \widehat{\mathscr{R}^{\mathscr{E}_2}}\ \hat{\phi}_n$ by fixpoint induction hypothesis, we have

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

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

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

Theorem 3 (Correctness of local semantics – 2nd component)
$\mathscr{E}\ \widehat{\mathscr{R}^{\mathscr{E}_2}}\ \widehat{\mathscr{E}}$.

Proof: From Lemma 4. □

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

Observation 2

During partial evaluation, all values $\hat{r} \in$ **Res** produced by $\widehat{\mathscr{E}}$ satisfy the following conditions:

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

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

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

4.3.2 Correctness of the global semantics

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

Since the result of both $\mathscr{A}$ and $\widehat{\mathscr{A}}$ is a cache, $\widehat{\mathscr{R}^{\mathscr{A}}}$ should relate caches. That is, whenever a standard signature for a function is recorded in the cache produced by $\mathscr{A}$, there exists a logically related partial-evaluation signature for that function in the cache produced by $\widehat{\mathscr{A}}$. Formally, *Definition 3 (Relation $\widehat{\mathcal{R}^{\mathcal{A}}}$)*

$\widehat{\mathcal{R}^{\mathcal{A}}}$ is a logical relation between domains of $\mathcal{A}$ and $\widehat{\mathcal{A}}$ defined by:

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

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

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

$$\rho \; \widehat{\mathcal{R}^{\mathcal{A}}_{VarEnv}} \; \hat{\rho} \Leftrightarrow Dom(\rho) = Dom(\hat{\rho}) \wedge \forall [\![x]\!] \in Dom(\rho), \; \rho[\![x]\!] \; \widehat{\mathcal{R}^{\mathcal{A}}_{Result_{\widehat{\mathcal{E}}}}} \; \hat{\rho}[\![x]\!]$$

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

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

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

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

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 $\widehat{\mathcal{R}^{\mathcal{A}}}$.

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

Lemma 5

Given a program P, for any $\widehat{\mathcal{E}}$ such that $\mathcal{E} \; \widehat{\mathcal{R}^{\mathcal{A}}} \; \widehat{\mathcal{E}}$, 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 $\rho \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{\rho}$,

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

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

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

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

1. $Cond_{\mathscr{A}}$: By structural induction hypothesis, $\widehat{\mathscr{R}^{\mathscr{A}}}$ holds for the test expression.
   (a) If $\hat{v}_1 \in$ **Const**, since $k_1(\rho, \phi) \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{k}_1(\hat{\rho}, \hat{\phi})$, the branch chosen will be the same for both $\mathscr{A}$ and $\widehat{\mathscr{A}}$, and by structural induction hypothesis, $\widehat{\mathscr{R}^{\mathscr{A}}}$ holds.
   (b) If $\hat{v}_1 = \top_{\widehat{Values}}$, then all non-trivial calls in both branches are recorded by $\widehat{\mathscr{A}}$. Again, by structural induction hypothesis, $\widehat{\mathscr{R}^{\mathscr{A}}}$ holds.
2. $App_{\mathscr{A}}$: By structural induction hypothesis, $\widehat{\mathscr{R}^{\mathscr{A}}}$ holds for all the arguments to the application. As for the application itself, if it is recorded by $\mathscr{A}$, then it is non-trivial. By structural induction on the arguments, the application is also recorded by $\widehat{\mathscr{A}}$. Its transformation tag is either u or s. It is easy to see that $\widehat{\mathscr{R}^{\mathscr{A}}}$ holds when the transformation tag is u. If it is s, $\widehat{\mathscr{R}^{\mathscr{A}}}$ holds if $\forall i \in \{1, \ldots, n\}$, $\hat{r}_i{\downarrow}2 \sqsubseteq_{\widehat{Values}} \hat{r}'_i{\downarrow}2$, where $\langle \hat{r}'_1, \cdots, \hat{r}'_n \rangle$ is the result of applying $SpPat$ in $\widehat{\mathscr{A}}$. This is true by the definition of $\widehat{bt}$, $SpPat$ and $Ft[\![f]\!]{\downarrow}2$.

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

Notice that, for a value $\hat{r}$, there may be more than one value v such that $v \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{r}$. Therefore, the above lemma shows that given an expression e, $\widehat{\mathscr{A}}$ captures all calls within e that may be invoked under different initial value v with $v \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{r}$. The following theorem uses Lemma 5 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 4 (Correctness of global partial evaluation semantics)
Given a program P, let $\widehat{\mathscr{E}}$ be a valuation function of the partial evaluation semantics such that $\mathscr{E} \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \widehat{\mathscr{E}}$. Let $\langle v_1, \ldots, v_n \rangle$ and $\langle \hat{r}_1, \ldots, \hat{r}_n \rangle$ be initial inputs to P for standard and partial evaluation semantics, respectively, such that $v_i \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{r}_i$, $\forall i \in \{1, \ldots, n\}$. If σ and $\hat{\sigma}$ are the final caches produced by $\mathscr{A}$ and $\widehat{\mathscr{A}}$ respectively, then $\sigma \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{\sigma}$.

Proof: First, we notice from the definition of $\widehat{\mathscr{E}}_{Prog}$ that $\langle \mathrm{s}, \hat{r}_1, \ldots, \hat{r}_n \rangle \in \hat{\sigma}[\![f_1]\!]$, just like $\langle v_1, \ldots, v_n \rangle \in \sigma[\![f_1]\!]$. Next, $\hat{h}$ in $\widehat{\mathscr{E}}_{Prog}$ applies $\widehat{\mathscr{A}}$ to each partial-evaluation signature in the cache, and combines the result using the l.u.b. operation. This is similar to function h in $\mathscr{E}_{Prog}$. By Lemma 5, the computation of both $\mathscr{A}$ and $\widehat{\mathscr{A}}$ are related by $\widehat{\mathscr{R}^{\mathscr{A}}}$. Since l.u.b. operation is closed under $\widehat{\mathscr{R}^{\mathscr{A}}}$, $\sigma \ \widehat{\mathscr{R}^{\mathscr{A}}} \ \hat{\sigma}$. □

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

4.3.3 Correctness of $\widehat{\mathscr{R}^{\mathscr{E}_1}}$

We now prove the correctness of the residual expression produced by $\widehat{\mathscr{E}}$ using the relation $\widehat{\mathscr{R}^{\mathscr{E}_1}}$ which relates a residual expression to a concrete value produced by $\mathscr{E}$. 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, $\widehat{\mathscr{R}^{\mathscr{E}_1}}$ 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 (1992). 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 4 (Satisfiability)
Let P be a program. Let $\rho_d \in \mathit{VarEnv}$ be a variable environment for a residual expression such that $\mathit{Dom}(\rho_d) = \mathit{FV}(\hat{r}{\downarrow}1)$. We say ρ_d satisfies the pair $\langle v, \hat{r}\rangle$ if

$$v =_\perp \mathscr{E}[\![\hat{r}{\downarrow}1]\!](\rho_d, \phi') \;\wedge\; v \sqsubseteq_{\hat{\tau}} \hat{r}{\downarrow}2,$$

where $v \in$ **Values**, $\hat{r} \in$ **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 takes two parameters (x_1 and x_2). To show the relationship between $\mathscr{E}$ and $\widehat{\mathscr{E}}$, we must first show that the function environments they take as arguments are related. The following lemma clarifies this relation.

Lemma 6
Given a program P, 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 $\langle v_1, \hat{r}_1\rangle$ and $\langle v_2, \hat{r}_2\rangle$ with $v_1, v_2 \in$ **Values** and $\hat{r}_1, \hat{r}_2 \in$ **Res**. Then,

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

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 Gomard (1992). 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 the correctness proof of polyvariant specialization.

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

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

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

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

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

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

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

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

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

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

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

ϕ' is derived from cache $\hat{\sigma}$ produced by $\widehat{\mathscr{A}}$ 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 $\widehat{\mathscr{A}}$ with function environment $\hat{\phi}_i$ and $\widehat{\mathscr{A}}$ is continuous in all its arguments. □

Property 2
$\forall\ i \in \{0,1,\ldots\},\ \phi'_i \sqsubseteq_{FunEnv} \phi'_{i+1}$.
Proof: Since $\forall i \in \{0,1,\ldots\}$, ϕ'_i is obtained from the residual program, which is the result of $\widehat{\mathscr{E}_{prog}}$. Inspecting the function definition of $\widehat{\mathscr{E}_{prog}}$ shows that it is continuous in all its arguments. In particular, since $\forall\ i \in \{0,1,\ldots\}$, $\hat{\sigma}_i \sqsubseteq_{Cache} \hat{\sigma}_{i+1}$, therefore $\phi'_i \sqsubseteq_{FunEnv} \phi'_{i+1}$. □

We prove the validity of $\widehat{\mathscr{R}^{\mathscr{E}_1}}$ 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 $\hat{\phi}[\![f_i]\!](\hat{r}_1,\hat{r}_2) = \perp_{Res}$. Thus, $\widehat{\mathscr{R}^{\mathscr{E}_1}}\langle\phi_0,\hat{\phi}_0,\phi'_0\rangle$ holds vacuously.

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

For clarity, we introduce the following abbreviations:

1. $\bot[v_k/x_k]$ is abbreviated by ρ and $\bot[\hat{r}_k/x_k]$ by $\hat{\rho}$.
2. Given an expression e, we abbreviate $\mathscr{E}\llbracket e \rrbracket(\rho, \phi_n)$ by $\llbracket e \rrbracket_{\mathscr{E}}$, and $\widehat{\mathscr{E}}\llbracket e \rrbracket(\hat{\rho}, \hat{\phi}_n)$ by $\llbracket e \rrbracket_{\widehat{\mathscr{E}}}$.

The proof of $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ 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 $v = \llbracket p(e_1, \ldots, e_n) \rrbracket_{\mathscr{E}}$ and $\hat{r} = \llbracket p(e_1, \ldots, e_n) \rrbracket_{\widehat{\mathscr{E}}}$.
  - — $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ holds trivially if $\hat{r} = \bot_{Res}$.
  - — If $\hat{r}\!\downarrow\!1$ is a constant, then $\hat{r}\!\downarrow\!1 = \hat{\tau}(v)$ from Observation 1. Therefore, $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ holds in this case.
  - — If $\hat{r} \downarrow 1$ is not a constant, then the residual expression is of the form $\llbracket p(e''_1, \ldots, e''_n) \rrbracket$, where $e''_i = \llbracket e_i \rrbracket_{\widehat{\mathscr{E}}}$ $\forall i \in \{1, \ldots, n\}$. By the structural induction hypothesis, $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ 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:

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

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

- e is a conditional expression, $\llbracket if\ e_1\ e_2\ e_3 \rrbracket$. $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ holds trivially if e_1 partially evaluates to $\bot_{Res}$. 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, $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ 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 $e''_i = \llbracket e_i \rrbracket_{\widehat{\mathscr{E}}}$ $\forall i \in \{1, \ldots, 3\}$. Therefore,

$$
\begin{array}{ll}
\mathscr{E}\llbracket \llbracket if\ e_1\ e_2\ e_3 \rrbracket_{\widehat{\mathscr{E}}} \rrbracket(\rho_d, \phi'_{n+1}) & \\
\quad = \ \mathscr{E}\llbracket if\ e''_1\ e''_2\ e''_3 \rrbracket(\rho_d, \phi'_{n+1}) & \\
\quad = \ (\mathscr{E}\llbracket e''_1 \rrbracket(\rho_d, \phi'_{n+1})) \rightarrow (\mathscr{E}\llbracket e''_2 \rrbracket(\rho_d, \phi'_{n+1})) \rrbracket (\mathscr{E}\llbracket e''_3 \rrbracket(\rho_d, \phi'_{n+1})) & \text{[from standard semantics]} \\
\quad =_{\bot} \llbracket e_1 \rrbracket_{\mathscr{E}} \rightarrow \llbracket e_2 \rrbracket_{\mathscr{E}} \rrbracket \llbracket e_3 \rrbracket_{\mathscr{E}} & \text{[structural induction hypothesis]} \\
\quad = \ \llbracket if\ e_1\ e_2\ e_3 \rrbracket_{\mathscr{E}} & \text{[from standard semantics]}
\end{array}
$$

  Thus, $\mathscr{R}^{\widehat{\mathscr{E}_1}}_{n+1}$ 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 $\widehat{\mathcal{A}}$)

$$\begin{aligned} \langle \mathsf{s}, \llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \langle \llbracket x_2\rrbracket, \hat{v}'\rangle\rangle \quad \textit{where}\ \langle \hat{r}_1', \langle \llbracket x_2\rrbracket, \hat{v}'\rangle\rangle &= \mathit{SpPat}(\llbracket f_i\rrbracket, \langle \llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}, \llbracket e_2\rrbracket_{\widehat{\mathcal{E}}}\rangle, \\ &\qquad \langle \mathit{Static}, \mathit{Dynamic}\rangle)\langle -, \hat{v}'\rangle = \llbracket e_2\rrbracket_{\widehat{\mathcal{E}}} \end{aligned}$$

Notice from the definition of *SpPat* that $\hat{r}_1' = \llbracket e_1\rrbracket_{\widehat{\mathcal{E}}}$ and $\hat{v}' = \top_{\widehat{Values}}$. The specialized function f_i^{sp} is included in the residual program produced; its definition is as follows.

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

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

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

Thus, we have

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

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

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

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

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, $\forall i \in \{1, 2\}$,

$$v_i =_\bot \mathcal{E}\llbracket \hat{r}_i\downarrow 1\rrbracket(\rho_d, \phi_{n+1}') \ \wedge\ v_i \sqsubseteq_{\hat{\tau}} \hat{r}_i\downarrow 2.$$

This is true by the structural induction hypothesis, Property 2 about ϕ_{n+1}', and Theorem 3.

Using ρ_d, the fixpoint induction hypothesis is

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

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 $[\![f^{sp}(r'_2)]\!]$. This residual call would be the result of partially evaluating a function call. Let the function call be $[\![f(r_1, r_2)]\!]$. Then, it would be the case that at the n^{th} approximation, we had

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

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

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

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

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

4.3.4 *Correctness of* $\widehat{\mathscr{R}}^{\mathscr{E}}$

Now, we are ready to define the relation between $\mathscr{E}$ and $\widehat{\mathscr{E}}$. This is defined in terms of the result of Theorem 3 and Lemma 6. Firstly, since both $\mathscr{E}$ and $\widehat{\mathscr{E}}$ 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 Gomard (1992).

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

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

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

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

Notice that ρ and $\hat{\rho}$ as defined in Observation 3 represent how all the variable environments used in standard and partial evaluation semantics are constructed. Therefore, the result of Lemma 6 can be expressed in terms of an arbitrary expression in program P as follows.

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

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

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

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

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

and

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

Proof: From Theorem 3 and Corollary 1. □

5 Off-line partial evaluation semantics

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

5.1 Binding-time algebra

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

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

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

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

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

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

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

5.2 *Specification of binding-time analysis*

Figure 9 displays the binding-time analysis for our language. The analysis aims at collecting binding-time information for each function in a given program; this forms the *binding-time signature* of the function. More precisely, a binding-time signature in domain **Sig** is created when a function call is analyzed by the binding-time analysis. It consists of two components: A transformation tag similar to that used in the partial-evaluation signature, and the argument values of the application in $\widetilde{\mathbf{Values}}^{n}$.

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

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

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

As an example of a cache produced by $\widetilde{\mathscr{A}}$, consider the exponentiation function f defined in Section 3.1 (with the filter discussed in Section 4.2). When function f is analyzed with respect to the binding-time description $\langle Dynamic, Static\rangle$, the following caches are produced until a fixpoint is reached.

Iteration #	Cache
1	$[\langle \mathtt{s}, Dynamic, Static\rangle / f]$
2	$[\langle \mathtt{s}, Dynamic, Static\rangle / f]$

The final cache indicates that when Function f is partially evaluated with respect to an input of binding-time value $\langle Dynamic, Static\rangle$, all calls to Function f are suspended (thus the transformation value s), and the second argument is propagated.

5.3 Correctness of binding-time analysis

The initial input to the binding-time analysis is an abstraction of the initial input of on-line partial evaluation. The analysis is correct if its final cache (**Cache**$_{\widetilde{\mathscr{A}}}$) 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 $\widetilde{\mathscr{R}^{\mathscr{E}}}$ that relates $\widehat{\mathscr{E}}$ and $\widetilde{\mathscr{E}}$, and a logical relation $\widetilde{\mathscr{R}^{\mathscr{A}}}$ that relates $\widehat{\mathscr{A}}$ and $\widetilde{\mathscr{A}}$. We first show the correctness of the local semantics defined by $\widetilde{\mathscr{E}}$, and then that of the global semantics defined by $\widetilde{\mathscr{A}}$.

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

To relate $\widetilde{\mathscr{E}}$ and $\widehat{\mathscr{E}}$, it is sufficient to relate the binding-time values to the partial-evaluation values. This relation forms the basis of the logical relation defined below.

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

$$\hat{r}\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{Result_{\tilde{\mathscr{E}}}}\,\tilde{v} \Leftrightarrow \hat{r}{\downarrow}2 \sqsubseteq_{\tilde{\tau}} \tilde{v}$$
$$\hat{\rho}\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{VarEnv}\,\tilde{\rho} \Leftrightarrow Dom(\hat{\rho}) = Dom(\tilde{\rho}) \wedge \forall [\![x]\!] \in Dom(\hat{\rho}), \hat{\rho}[\![x]\!]\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{Result_{\tilde{\mathscr{E}}}}\,\tilde{\rho}[\![x]\!]$$
$$\hat{\phi}\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{FunEnv}\,\tilde{\phi} \Leftrightarrow Dom(\hat{\phi}) = Dom(\tilde{\phi}) \wedge \forall [\![f]\!] \in Dom(\hat{\phi}), \forall i \in \{1,\ldots,n\}, \forall \hat{r}_i \in \mathbf{Res}, \forall \tilde{v}_i \in \widetilde{\mathbf{Values}},$$
$$\bigwedge_{i=1}^{n} (\hat{r}_i\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{Result_{\tilde{\mathscr{E}}}}\,\tilde{v}_i) \Rightarrow \hat{\phi}[\![f]\!](\hat{r}_1,\ldots,\hat{r}_n)\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{Result_{\tilde{\mathscr{E}}}}\,\tilde{\phi}[\![f]\!](\tilde{v}_1,\ldots,\tilde{v}_n)$$
$$\langle \hat{d}_1, \hat{d}_2\rangle\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{D_1\times D_2}\,\langle \tilde{d}_1, \tilde{d}_2\rangle \Leftrightarrow \hat{d}_1\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{D_1}\,\tilde{d}_1 \wedge \hat{d}_2\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{D_2}\,\tilde{d}_2$$
$$\hat{f}\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{D_1\to D_2}\,\tilde{f} \Leftrightarrow \forall \hat{d} \in \widehat{D}_1, \forall \tilde{d} \in \widetilde{D}_1, \hat{d}\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{D_1}\,\tilde{d} \Rightarrow \hat{f}(\hat{d})\,\widetilde{\mathscr{R}^{\mathscr{E}}}_{D_2}\,\tilde{f}(\tilde{d}).$$

- Semantic Domains

  $\tilde{v} \in \mathit{Result}_{\tilde{\mathscr{E}}} = \widetilde{\mathbf{Values}}$
  $\tilde{\rho} \in \widetilde{\mathit{VarEnv}} = \mathbf{Var} \rightarrow \widetilde{\mathbf{Values}}$
  $\tilde{\phi} \in \widetilde{\mathit{FunEnv}} = \mathbf{FEnv} = \mathbf{Fn} \rightarrow \widetilde{\mathbf{Values}}^{n} \rightarrow \widetilde{\mathbf{Values}}$
  $\widetilde{\mathit{Env}} = \widetilde{\mathit{VarEnv}} \times \widetilde{\mathit{FunEnv}}$
  $\tilde{s} \in \mathbf{Sig} = (\mathbf{Transf} \times \widetilde{\mathbf{Values}}^{n})$
  $\tilde{\sigma} \in \mathit{Result}_{\tilde{\mathscr{A}}} = \mathbf{Cache}_{\tilde{\mathscr{A}}} = \mathbf{Fn} \rightarrow \mathbf{Sig}$

- Valuation Functions

  $\tilde{\mathscr{E}}_{Prog} : \mathbf{Program} \rightarrow \widetilde{\mathbf{Values}}^{n} \rightarrow \mathbf{Cache}_{\tilde{\mathscr{A}}}$
  $\tilde{\mathscr{E}}_{Prog}\llbracket\{f_i(x_1,\cdots,x_n) = e_i\}\rrbracket\langle\tilde{v}_1,\cdots,\tilde{v}_n\rangle = \tilde{h}(\bot[\langle \mathrm{s},\tilde{v}_1,\cdots,\tilde{v}_n\rangle/f_1])$
  $\quad \mathit{whererec}\ \tilde{h}(\tilde{\sigma}) = \tilde{\sigma} \sqcup \tilde{h}(\bigsqcup\{\tilde{\mathscr{A}}\llbracket e_i\rrbracket(\bot[\tilde{v}_k/x_k],\tilde{\phi}) \mid \langle -,\tilde{v}_1,\ldots,\tilde{v}_n\rangle$
  $\qquad = \tilde{\sigma}\llbracket f_i\rrbracket,\ \forall\llbracket f_i\rrbracket \in \mathit{Dom}(\tilde{\sigma})\})$
  $\qquad \tilde{\phi} = \bot[\{\lambda(\tilde{v}_1,\ldots,\tilde{v}_n).\tilde{\mathscr{E}}\llbracket e_i\rrbracket(\bot[\tilde{v}_k/x_k],\tilde{\phi})\}/f_i]$
  $\tilde{\mathscr{E}} = \overline{\mathscr{E}}$
  $\tilde{\mathscr{A}} = \overline{\mathscr{A}}$

- Combinator Definitions

  $\mathit{Const}_{\tilde{\mathscr{E}}}\llbracket c\rrbracket = \lambda(\tilde{\rho},\tilde{\phi})\ .\ \tilde{\mathscr{K}}\llbracket c\rrbracket$
  $\mathit{VarLookup}_{\tilde{\mathscr{E}}}\llbracket x\rrbracket = \lambda(\tilde{\rho},\tilde{\phi})\ .\ \tilde{\rho}\llbracket x\rrbracket$
  $\mathit{PrimOp}_{\tilde{\mathscr{E}}}\llbracket p\rrbracket(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\ .\ \tilde{\mathscr{K}}_P\llbracket p\rrbracket(\tilde{k}_1(\tilde{\rho},\tilde{\phi}),\ldots,\tilde{k}_n(\tilde{\rho},\tilde{\phi}))$
  $\mathit{Cond}_{\tilde{\mathscr{E}}}(\tilde{k}_1,\tilde{k}_2,\tilde{k}_3) = \lambda(\tilde{\rho},\tilde{\phi})\ .\ \tilde{v}_1 = \bot_{\widetilde{Values}} \rightarrow \bot_{\widetilde{Values}}$
  $\qquad [\!](\tilde{v}_1 = \mathit{Static} \rightarrow \tilde{v}_2 \sqcup \tilde{v}_3\,[\!]\,\mathit{Dynamic})$
  $\qquad \mathit{where}\ \tilde{v}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi})\forall i \in \{1,2,3\}$
  $\mathit{App}_{\tilde{\mathscr{E}}}\llbracket f\rrbracket(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\ .\ (\mathit{Ft}\llbracket f\rrbracket)\downarrow 1(\tilde{v}_1,\ldots,\tilde{v}_n) = \mathrm{u} \rightarrow\ \tilde{\phi}\llbracket f\rrbracket(\tilde{v}_1,\ldots,\tilde{v}_n)\,[\!]\,\mathit{Dynamic}$
  $\qquad \mathit{where}\ \tilde{v}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi})\forall i \in \{1,\ldots,n\}$
  $\mathit{Const}_{\tilde{\mathscr{A}}}\llbracket c\rrbracket = \lambda(\tilde{\rho},\tilde{\phi})\ .\ (\lambda f\ .\ \bot_{Sig})$
  $\mathit{VarLookup}_{\tilde{\mathscr{A}}}\llbracket x\rrbracket = \lambda(\tilde{\rho},\tilde{\phi})\ .\ (\lambda f\ .\ \bot_{Sig})$
  $\mathit{PrimOp}_{\tilde{\mathscr{A}}}\llbracket p\rrbracket(\tilde{a}_1,\ldots,\tilde{a}_n) = \lambda(\tilde{\rho},\tilde{\phi})\ .\ \bigsqcup_{i=1}^{n}\tilde{a}_i(\tilde{\rho},\tilde{\phi})$
  $\mathit{Cond}_{\tilde{\mathscr{A}}}(\tilde{a}_1,\tilde{a}_2,\tilde{a}_3)\tilde{k}_1 = \lambda(\tilde{\rho},\tilde{\phi})\ .\ \tilde{a}_1(\tilde{\rho},\tilde{\phi}) \sqcup \tilde{a}_2(\tilde{\rho},\tilde{\phi}) \sqcup \tilde{a}_3(\tilde{\rho},\tilde{\phi})$
  $\mathit{App}_{\tilde{\mathscr{A}}}\llbracket f\rrbracket(\tilde{a}_1,\ldots,\tilde{a}_n)(\tilde{k}_1,\ldots,\tilde{k}_n) = \lambda(\tilde{\rho},\tilde{\phi})\ .\ (\bigsqcup_{i=1}^{n}\tilde{a}_i(\tilde{\rho},\tilde{\phi})) \sqcup \tilde{\sigma}$
  $\qquad \mathit{where}\tilde{\sigma} = (\mathit{Ft}\llbracket f\rrbracket)\downarrow 1(\tilde{v}_1,\ldots,\tilde{v}_n) = \mathrm{u}$
  $\qquad\quad \rightarrow \bot[\langle \mathrm{u},\tilde{v}_1,\ldots,\tilde{v}_n\rangle/f]\,[\!]\,\bot[\langle \mathrm{s},\tilde{v}'_1,\ldots,\tilde{v}'_n\rangle/f]$
  $\qquad \langle\tilde{v}'_1,\ldots,\tilde{v}'_n\rangle = (\mathit{Ft}\llbracket f\rrbracket)\downarrow 2(\tilde{v}_1,\ldots,\tilde{v}_n)$
  $\qquad \tilde{v}_i = \tilde{k}_i(\tilde{\rho},\tilde{\phi})\forall i \in \{1,\ldots,n\}$

- Primitive Functions

  $\tilde{\mathscr{K}} : \mathbf{Const} \rightarrow \widetilde{\mathbf{Values}}$
  $\tilde{\mathscr{K}}\llbracket c\rrbracket = \tilde{\tau}(\llbracket c\rrbracket)$
  $\tilde{\mathscr{K}}_P : \mathbf{Po} \rightarrow \widetilde{\mathbf{Values}}^{n} \rightarrow \widetilde{\mathbf{Values}}$
  $\tilde{\mathscr{K}}_P\llbracket p\rrbracket(\tilde{v}_1,\ldots,\tilde{v}_n) = \tilde{p}(\tilde{v}_1,\ldots,\tilde{v}_n)$

Fig. 9. Binding-time analysis.

Lemma 7
Given a program P, let $\hat{\phi}$ and $\tilde{\phi}$ be the two function environments for P defined by the partial evaluation semantics and the binding-time analysis respectively. Then $\hat{\phi}\ \widetilde{\mathscr{R}^{\mathscr{E}}}\ \tilde{\phi}$.

Proof: The proof is similar to the proof for Lemma 4, and is thus omitted. □

Theorem 6 (Correctness of local binding-Time analysis)
$\widehat{\mathscr{E}}\ \widetilde{\mathscr{R}^{\mathscr{E}}}\ \widetilde{\mathscr{E}}$.

Proof: From Lemma 7. □

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

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

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

5.3.2 *Correctness of the global analysis*

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

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

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

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

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

Lemma 8
Given a program P, let $\hat{\phi}$ and $\tilde{\phi}$ be two function environments for P defined by the partial evaluation and the binding-time analysis respectively. For any expression e in P, for any $\hat{\rho}$, $\tilde{\rho}$ such that $\hat{\rho}\widetilde{\mathscr{R}^{\mathscr{A}}}\tilde{\rho}$,

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

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

1. $Cond_{\widetilde{\mathscr{A}}}$: By the structural induction hypothesis, all the corresponding pairs of arguments are related by $\widetilde{\mathscr{R}^{\mathscr{A}}}$. Since the result of $Cond_{\widetilde{\mathscr{A}}}$ is the l.u.b. of the caches produced at all the arguments, whereas the result of $Cond_{\widehat{\mathscr{A}}}$ is the l.u.b. of the caches produced at some of the arguments, $\widetilde{\mathscr{R}^{\mathscr{A}}}$ must hold.
2. $App_{\widetilde{\mathscr{A}}}$: As seen from the two semantic specifications in figures 8 and 9, we need to show that the results of function applications at the two specifications are related by $\widetilde{\mathscr{R}^{\mathscr{A}}}$. That is,

$$(\bigsqcup_{i=1}^{n} \hat{a}_i(\hat{\rho},\hat{\phi}) \sqcup \hat{\sigma})\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ (\bigsqcup_{i=1}^{n} \tilde{a}_i(\tilde{\rho},\tilde{\phi}) \sqcup \tilde{\sigma}).$$

By the structural induction hypothesis and the fact that l.u.b. operations are closed under $\widetilde{\mathscr{R}^{\mathscr{A}}}$, we have

$$\bigsqcup_{i=1}^{n} \hat{a}_i(\hat{\rho},\hat{\phi})\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ \bigsqcup_{i=1}^{n} \tilde{a}_i(\tilde{\rho},\tilde{\phi}).$$

It suffices to show that $\hat{\sigma}\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ \tilde{\sigma}$. We refer the reader to figures 8 and 9 for the notations used in the following proof, in which we consider the cases with different transformation values $\tilde{t}$ produced at the binding-time analysis level.

- If $\tilde{t} = \text{u}$, then $\hat{t} = \text{u}$ by the monotonicity of filters. From figures 8 and 9, we have $\hat{\sigma} = \bot[\{\langle \text{u}, \hat{r}_1, \ldots, \hat{r}_n\rangle\}/f]$ and $\tilde{\sigma} = \bot[\langle \text{u}, \tilde{v}_1, \ldots, \tilde{v}_n\rangle/f]$. Notice that, $\forall\ i \in \{1,\ldots,n\}$,

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

  Therefore, $\hat{\sigma}\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ \tilde{\sigma}$.
- If $\tilde{t} = \text{s}$, then $\hat{t} \sqsubseteq_{Transf} \tilde{t}$ by monotonicity of filters. From Figure 9, we have $\tilde{\sigma} = \bot[\langle \text{s}, \tilde{v}'_1, \ldots, \tilde{v}'_n\rangle/f]$. From Figure 8, $\hat{\sigma}$ can either be $\bot[\{\text{u}, \hat{r}_1, \ldots, \hat{r}_n\}/f]$ or $\bot[\{\text{s}, \hat{r}'_1, \ldots, \hat{r}'_n\}/f]$. However, from the functionality of the filter and the definition of $SpPat$ at the on-line level, we have $\forall i \in \{1,\ldots,n\}$, $\hat{r}_i{\downarrow}2 \sqsubseteq_{\widehat{Values}} \hat{r}'_i{\downarrow}2$. Therefore, it suffices to show that $\bot[\{\langle \text{s}, \hat{r}'_1, \ldots, \hat{r}'_n\rangle\}/f]\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ \bot[\langle \text{s}, \tilde{v}'_1, \ldots, \tilde{v}'_n\rangle/f]$. The proof can be further reduced to showing that $\forall i \in \{1,\ldots,n\}$, $\hat{r}'_i\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ \tilde{v}'_i$, which can be derived from the following two observations:

  (a) $\forall i \in \{1,\ldots,n\}$, if $\hat{r}_i\ \widetilde{\mathscr{R}^{\mathscr{A}}}\ \tilde{v}_i$, then from the monotonicity of the filter, we must have $\forall j \in \{1,\ldots,n\}$,

$$((Ft\,[\![f]\!]){\downarrow}2\ (\widehat{bt}(\hat{r}_1),\ldots,\widehat{bt}(\hat{r}_n)))_j \sqsubseteq_{\widetilde{Values}} ((Ft\,[\![f]\!]){\downarrow}2\ (\tilde{v}_1,\ldots,\tilde{v}_n))_j.$$

(b) Given $\hat{r}_1,\ldots,\hat{r}_n$, if $\langle b_1,\ldots,b_n\rangle = (Ft[\![f]\!]) \downarrow 2(\widehat{bt}(\hat{r}_1),\ldots,\widehat{bt}(\hat{r}_n))$, and $\langle \hat{r}'_1,\ldots,\hat{r}'_n\rangle = SpPat([\![f]\!],\langle \hat{r}_1,\ldots,\hat{r}_n\rangle, \langle b_1,\ldots,b_n\rangle)$. Then, from the definition of *SpPat*, we have $\forall j \in \{1,\ldots,n\}$,

$$\hat{r}_j{\downarrow}2 \sqsubseteq_{\widehat{Values}} \hat{r}'_j{\downarrow}2 \textit{ and } \widehat{bt}(\hat{r}'_j) = b_j.$$

These two observations together ensure that $\forall i \in \{1,\ldots,n\}$, if $\hat{r}'_i$ is taken from $\bot[\{\langle \mathrm{s},\hat{r}'_1,\ldots,\hat{r}'_n\rangle\}/f]$, and $\tilde{v}'_i$ is taken from $\bot[\langle \mathrm{s},\tilde{v}'_1,\ldots,\tilde{v}'_n\rangle/f]$, then we must have

$$\widehat{bt}(\hat{r}'_i) \sqsubseteq_{\widetilde{Values}} \tilde{v}'_i.$$

That is, $\hat{r}'_i{\downarrow}2 \sqsubseteq_{\tilde{\tau}} \tilde{v}'_i$. Hence, we have $\hat{r}'_i \; \widetilde{\mathscr{R}^{\mathscr{A}}} \; \tilde{v}'_i$.

Thus, we have $\hat{\sigma} \; \widetilde{\mathscr{R}^{\mathscr{A}}} \; \tilde{\sigma}$, and therefore, $App_{\widehat{\mathscr{A}}} \; \widetilde{\mathscr{R}^{\mathscr{A}}} \; App_{\widetilde{\mathscr{A}}}$.

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

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

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

5.4 Deriving the specialization semantics

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

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

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

5.4.1 *Specification of the specializer*

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

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

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

5.4.2 *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 Bondorf *et al.* (1988) and Jones *et al.* (1989), 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 (Jones *et al.*, 1989).

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. In fact, the specialization

** Note that for simplicity we did not introduce labels in the core semantics presented on page 467. Indeed, labels are only used for the specialization semantics.

- Semantic Domains

$$\begin{array}{rcl@{\qquad}rcl}
l & \in & \textbf{Labels} & \hat{r} & \in & \mathit{Result}_{\widetilde{\mathscr{S}}} = \textbf{Res} = \text{as in On–Line Sem.} \\
\hat{v} & \in & \widehat{\textbf{Values}} = \text{as in On–Line Sem.} & \tilde{v} & \in & \widetilde{\textbf{Values}} = \text{as in Off–Line Sem.} \\
\langle \tilde{t}, \tilde{v} \rangle & \in & \textbf{Att} = (\textbf{Transf} \times \widetilde{\textbf{Values}}) & \hat{\sigma} & \in & \mathit{Result}_{\widetilde{\mathscr{A}}} = \text{as in On–Line Sem.} \\
\tilde{\psi} & \in & \textbf{AtCache} = \textbf{Labels} \rightarrow \textbf{Att} & \hat{\phi} & \in & \widehat{\mathit{FunEnv}} = \text{as in On–Line Sem.} \\
\hat{\rho} & \in & \widehat{\mathit{VarEnv}} = \text{as in On–Line Sem.} & & & \widehat{\mathit{Env}} = \widehat{\mathit{VarEnv}} \times \widehat{\mathit{FunEnv}} \times \textbf{AtCache}
\end{array}$$

- Valuation Functions

$$\begin{array}{l}
\widehat{\mathscr{E}}_{Prog} \ : \textbf{Prog} \rightarrow \textbf{Res}^n \rightarrow \textbf{AtCache} \rightarrow \textbf{Prog}_{\perp} \\
\widehat{\mathscr{E}}_{Prog} \llbracket \{ f_i(x_1, \cdots, x_n) = e_i \} \rrbracket \langle \hat{r}_1, \ldots, \hat{r}_n \rangle \tilde{\psi} = \mathit{MkProg}(\hat{h}(\perp[\{\langle \mathrm{s}, \hat{r}_1, \ldots, \hat{r}_n \rangle\}/f_1]))\tilde{\psi}\hat{\phi} \\
\quad \mathit{whererec} \ \ \hat{h}(\hat{\sigma}) = \hat{\sigma} \sqcup \hat{h}(\bigsqcup \{ \mathscr{A}_{\widetilde{\mathscr{S}\mathscr{A}}} \llbracket e_i \rrbracket (\perp[\hat{r}'_k/x_k], \hat{\phi}, \tilde{\psi}) \mid \langle -, \hat{r}'_1, \ldots, \hat{r}'_n \rangle \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \in \hat{\sigma}\llbracket f_i \rrbracket, \ \forall \llbracket f_i \rrbracket \in \mathit{Dom}(\hat{\sigma}) \}) \\
\qquad\qquad\qquad \hat{\phi} = \perp[\mathit{strict}\{\lambda(\hat{r}_1, \cdots, \hat{r}_n) . \mathscr{E}_{\widetilde{\mathscr{S}\mathscr{E}}} \llbracket e_i \rrbracket (\perp[\hat{r}_k/x_k], \hat{\phi}, \tilde{\psi})\}/f_i]
\end{array}$$

- *MkProg* Definition

$$\begin{array}{l}
\mathit{MkProg}\,\hat{\sigma}\tilde{\psi}\hat{\phi} = \{ f_i^{sp}(x_1, \ldots, x_k) = \hat{r}'{\downarrow}1 \mid \forall \langle \mathrm{s}, \hat{r}_1, \ldots, \hat{r}_n \rangle \in \hat{\sigma}\llbracket f_i \rrbracket, \forall \llbracket f_i \rrbracket \in \mathit{Dom}(\hat{\sigma}) \} \\
\quad \mathit{where} \ \ f_i^{sp} = \mathit{SpName}(\llbracket f_i \rrbracket, \hat{r}_1, \ldots, \hat{r}_n) \\
\qquad\qquad \hat{r}' = \mathscr{E}_{\widetilde{\mathscr{S}\mathscr{E}}} \llbracket e_i \rrbracket (\perp[\hat{r}_k/x_k], \hat{\phi}, \tilde{\psi}) \\
\qquad\qquad \langle x_1, \ldots, x_k \rangle = \mathit{ResidPars}(\llbracket f_i \rrbracket, \hat{r}_1{\downarrow}1, \ldots, \hat{r}_n{\downarrow}1)
\end{array}$$

- Local Combinator Definitions

$$\begin{array}{l}
\mathit{Const}_{\widetilde{\mathscr{S}\mathscr{E}}} \llbracket c \rrbracket \langle l \rangle = \lambda(\hat{\rho}, \hat{\phi}, \tilde{\psi}) \ . \ \widehat{\mathscr{K}} \llbracket c \rrbracket \\
\mathit{Var}_{\widetilde{\mathscr{S}\mathscr{E}}} \llbracket x \rrbracket \langle l \rangle = \lambda(\hat{\rho}, \hat{\phi}, \tilde{\psi}) \ . \ \hat{\rho}\llbracket x \rrbracket \\
\mathit{PrimOp}_{\widetilde{\mathscr{S}\mathscr{E}}} \llbracket p \rrbracket (\hat{k}_1, \ldots, \hat{k}_n) \tilde{v} \langle l, l_1, \ldots, l_n \rangle = \lambda(\hat{\rho}, \hat{\phi}, \tilde{\psi}) \ . \ \widehat{\mathscr{K}}_{SP} \llbracket p \rrbracket (\hat{k}_1(\hat{\rho}, \hat{\phi}, \tilde{\psi}), \ldots, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \hat{k}_n(\hat{\rho}, \hat{\phi}, \tilde{\psi}))(\tilde{\psi} l)_v \\
\mathit{Cond}_{\widetilde{\mathscr{S}\mathscr{E}}} (\hat{k}_1, \hat{k}_2, \hat{k}_3) \langle l, l_1, l_2, l_3 \rangle = \\
\quad \lambda(\hat{\rho}, \hat{\phi}, \tilde{\psi}) \ . \ (\tilde{\psi} l_1)_v = \mathit{Static} \rightarrow \ (\mathscr{K}(\hat{r}_1{\downarrow}1) \rightarrow \hat{r}_2 [\!] \hat{r}_3) [\!] \langle \llbracket \mathit{if}\, \hat{r}_1{\downarrow}1 \hat{r}_2{\downarrow}1 \hat{r}_3{\downarrow}1 \rrbracket, \hat{r}_2{\downarrow}2 \sqcup \hat{r}_3{\downarrow}2 \rangle \\
\quad \mathit{where}\, \hat{r}_i = \hat{k}_i(\hat{\rho}, \hat{\phi}, \tilde{\psi}) \forall i \in \{1, 2, 3\} \\
\mathit{App}_{\widetilde{\mathscr{S}\mathscr{E}}} \llbracket f \rrbracket (\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}\llbracket f \rrbracket (\hat{r}_1, \ldots, \hat{r}_n) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\!] \langle \llbracket f_{sp}(e''_1, \ldots, e''_k) \rrbracket, \top_{\widehat{Values}} \rangle \\
\quad \mathit{where} \ \ \hat{r}_i = \hat{k}_i(\hat{\rho}, \hat{\phi}, \tilde{\psi}) \forall i \in \{1, \ldots, n\} \\
\qquad\qquad f_{sp} = \mathit{SpName}(\llbracket f \rrbracket, \hat{r}'_1, \ldots, \hat{r}'_n) \\
\qquad\qquad \langle e''_1, \ldots, e''_k \rangle = \mathit{ResidArgs}(\llbracket f \rrbracket, \langle \tilde{v}'_1, \ldots, \tilde{v}'_n \rangle, \langle \hat{r}_1{\downarrow}1, \ldots, \hat{r}_n{\downarrow}1 \rangle) \\
\qquad\qquad \langle \hat{r}'_1, \ldots, \hat{r}'_n \rangle = \mathit{SpPat}(\llbracket f \rrbracket, \langle \hat{r}_1, \ldots, \hat{r}_n \rangle, \langle \tilde{v}'_1, \ldots, \tilde{v}'_n \rangle) \\
\qquad\qquad \tilde{v}'_i = (\tilde{\psi} l_i)_v \forall i \in \{1, \ldots, n\}
\end{array}$$

- Primitive Functions

$$\begin{array}{l}
\widehat{\mathscr{K}} \ : \textbf{Const} \rightarrow \textbf{Res} \\
\ \widehat{\mathscr{K}} \llbracket c \rrbracket = (\text{as in On–Line Semantics}) \\
\widehat{\mathscr{K}}_{SP} \ : \textbf{Po} \rightarrow \textbf{Res}^n \rightarrow \widetilde{\textbf{Values}} \rightarrow \textbf{Res} \\
\ \widehat{\mathscr{K}}_{SP} \llbracket p \rrbracket (\langle e'_1, \hat{v}_1 \rangle, \ldots, \langle e'_n, \hat{v}_n \rangle) \tilde{v} = \\
\quad (\hat{v} = \perp_{\widehat{Values}}) \rightarrow \langle \perp_{Exp}, \perp_{\widehat{Values}} \rangle [\!] (\tilde{v} = \mathit{Static} \rightarrow \langle \hat{v}, \hat{v} \rangle [\!] \langle \llbracket p(e'_1, \cdots, e'_n) \rrbracket, \hat{v} \rangle) \\
\quad \mathit{where} \ \ \hat{v} = \hat{p}(\hat{v}_1, \cdots, \hat{v}_n)
\end{array}$$

Fig. 10. Specialization semantics – Part 1.

- Global Combinator Definitions

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

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

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

$$\begin{aligned}&Cond_{\widetilde{\mathscr{SA}}}(\hat{a}_1,\hat{a}_2,\hat{a}_3)\hat{k}_1\langle l,l_1,l_2,l_3\rangle = \\ &\quad\lambda(\hat{\rho},\hat{\phi},\tilde{\psi})\,.\,(\hat{a}_1(\hat{\rho},\hat{\phi},\tilde{\psi}))\sqcup((\tilde{\psi}l)_v = Static \rightarrow \; (\mathscr{K}(\hat{v}_1)\rightarrow \hat{a}_2(\hat{\rho},\hat{\phi},\tilde{\psi})[\!]\hat{a}_3(\hat{\rho},\hat{\phi},\tilde{\psi})) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\!]\hat{a}_2(\hat{\rho},\hat{\phi},\tilde{\psi})\sqcup\hat{a}_3(\hat{\rho},\hat{\phi},\tilde{\psi}))\end{aligned}$$

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

$$\begin{aligned}\textit{where}\;\; &\hat{r}_i = \hat{k}_i(\hat{\rho},\hat{\phi},\tilde{\psi})\forall i\in\{1,\cdots,n\} \\ &\hat{\sigma} = \;(\tilde{\psi}l)_t = \mathrm{u}\rightarrow\bot[\{\langle \mathrm{u},\hat{r}_1,\ldots,\hat{r}_n\rangle\}/f][\!]\bot[\{\langle \mathrm{s},\hat{r}'_1,\ldots,\hat{r}'_n\rangle\}/f] \\ &\langle\hat{r}'_1,\ldots,\hat{r}'_n\rangle = SpPat([\![f]\!],\langle\hat{r}_1,\ldots,\hat{r}_n\rangle,\langle\tilde{v}_1,\ldots,\tilde{v}_n\rangle) \\ &\tilde{v}_i = (\tilde{\psi}l_i)_v\forall i\in\{1,\ldots,n\}\end{aligned}$$

Fig. 11. Specialization semantics – Part 2.

semantics derived in the previous section may be used as a basis to introduce many optimizations. In particular, it is possible to infer statically the actions to be performed by the specializer. The basic actions of a specializer consists of reducing or rebuilding an expression. Such actions can be determined using the binding-time value of an expression. This technique has been used in off-line partial evaluation (Consel, 1993b; Consel and Danvy, 1990).

6 Conclusion

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

This paper should improve the understanding of partial evaluation in that it addresses such open issues as showing that binding-time analysis is an abstraction of the on-line partial-evaluation process, and formally defining the specialization semantics.

Also, this work should provide a basis for implementation. In fact, the specifications presented in this paper have been generalized by the authors to the specification of *parameterized partial evaluation* – a generic form of partial evaluation aimed at specializing programs not only with respect to concrete values, but also with respect to static properties (Consel and Khoo, 1991, 1993). Parameterized partial evaluation has already been successfully implemented at CMU (Colby and Lee, 1991) and at Yale (Khoo, 1992).

We are exploring ways of extending the current work to higher-order programs. To do so, we are formulating existing on-line and off-line higher-order partial evaluators (e.g. Bondorf, 1991; Consel, 1993a; Weise and Ruf, 1990) in a higher-order abstract interpretation setting like Jones's (1991).

Acknowledgments

We wish to thank Olivier Danvy, Pierre Jouvelot and the referees for their helpful remarks and their careful reading of earlier versions of this paper. Thanks are also due to Paul Hudak and the Yale Haskell Group for their encouragement.

References

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

Bondorf, A. (1991) Automatic autoprojection of higher-order recursive equations. *Science of Computer Programming*, **17**:3–34.

Bondorf, A., Jones, N. D., Mogensen, T. and Sestoft, P. (1988) *Binding Time Analysis and the Taming of Self-Application.* DIKU Report, University of Copenhagen, Denmark.

Colby, C. and Lee, P. (1991) An implementation of parameterized partial evaluation. In: *Workshop on Static Analysis*, pp. 82–89, Bigre Journal.

Consel, C. (1988) New insights into partial evaluation: the Schism experiment. In: H. Ganzinger (ed.), *ESOP'88: 2nd European Symposium on Programming*, pp. 236–246. Springer-Verlag.

Consel, C. (1993a) Polyvariant binding-time analysis for higher-order, applicative languages. In: *ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pap. 145–154.

Consel, C. (1993b) A tour of Schism: a partial evaluation system for higher-order applicative languages. In: *ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pp. 66–77.

Consel, C. and Danvy, D. (1990) From interpreting to compiling binding times. In: N. D. Jones (ed.), *ESOP'90: 3rd European Symposium on Programming*, pp. 88–105. Springer-Verlag.

Consel, C. and Danvy, D. (1993) Tutorial notes on partial evaluation. In: *ACM Symposium on Principles of Programming Languages*, pp. 493–501.

Consel, C. and Khoo, S. C. (1991) Parameterized partial evaluation. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 92–106.

Consel, C. and Khoo, S. C. (1993) Parameterized partial evaluation. *ACM Transactions on Programming Languages and Systems*, **15**(3):463–493. (Extended version of Consel and Khoo (1991).)

Futamura, Y. (1971) Partial evaluation of computation process – an approach to a compiler-compiler. *Systems, Computers, Controls*, **2**(5):45–50.

Ganzinger, H. and Jones, N. D. (eds.) (1985) *Programs as Data Objects. Lecture Notes in Computer Science 217.* Springer-Verlag.

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

Hannan, J. and Miller, D. (1989) *Deriving Mixed Evaluation From Standard Evaluation For a Simple Functional Language.* Technical Report MS-CIS-89-28, University of Pennsylvania, Philadelphia, PA.

Holst, C. K. (1989) *Program Specialization for Compiler Generation.* Master's thesis, University of Copenhagen, DIKU, Denmark.

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

Jones, N. D. (1988a) Automatic program specialization: a re-examination from basic principles. In: D. Bjørner, A. P. Ershov and N. D. Jones (eds.), *Partial Evaluation and Mixed Computation*, pp. 225–282. North-Holland.

Jones, N. D. (1988b) *Binding Time Analysis and Static Semantics (extended abstract).* DIKU Report, University of Copenhagen, Denmark.

Jones, N. D. (1990) Partial evaluation, self-application and types. In: M.S. Paterson (ed.), *17th International Colloquium on Automata, Languages and Programming*, pp. 639–659. Springer-Verlag.

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

Jones, N. D. and Muchnick, S. S. (1976) Some thoughts towards the design of an ideal language. In: *ACM Conference on Principles of Programming Languages*, pp. 77–94.

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

Jones, N. D. and Nielson, F. (1990) *Abstract Interpretation: a Semantics-Based Tool for Program Analysis.* Technical Report, University of Copenhagen and Aarhus University, Denmark.

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

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

Khoo, S. C. (1992) *Parameterized Partial Evaluation: Theory and Practice.* PhD thesis, Yale University. (Research Report 926.)

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

Mogensen, T. (1992) Self-applicable partial evaluation for pure Lambda Calculus. In: C. Consel (ed.), *ACM Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, pp 116–121. Yale University. (Research Report 909.)

Mizuno, M. and Schmidt, D. (1990) *A Security Flow Control Algorithm and Its Denotational Semantics Correctness Proof.* Technical Report CS-90-21, Kansas State University, Manhattan, KS.

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

Nielson, H. R. and Nielson, F. (1988a) Automatic binding time analysis for a typed λ-calculus. *Science of Computer Programming*, **10**:139–176.

Nielson, H. R. and Nielson, F. (1988b) Automatic binding time analysis for a typed λ-calculus. In: *ACM Symposium on Principles of Programming Languages*, pp. 98–106.

Nielson, F. and Nielson, H. R. (1992) *Two-Level Functional Languages. Cambridge Tracts in Theoretical Computer Science 34.* Cambridge University Press.

Sestoft, P. (1985) The structure of a self-applicable partial evaluator. In: Ganzinger, H. and Jones, N. D. (eds.) *Programs as Data Objects. Lecture Notes in Computer Science 217*, pp. 236–256. Springer-Verlag.

Sestoft, P. (1988) Automatic call unfolding in a partial evaluator. In: D. Bjørner, A. P. Ershov and N. D. Jones (eds.), *Partial Evaluation and Mixed Computation.* North-Holland.

Wand, M. (1993) Specifying the correctness of binding-time analysis. In: *ACM Symposium on Principles of Programming Languages*, pp. 137–143.

Weise, D. and Ruf, E. (1990) *Computing Types During Program Specialization.* Technical Report 441, Stanford University, CA.