Systematic Design of Program Transformation Frameworks by Abstract Interpretation*

Patrick Cousot and Radhia Cousot

Département d'informatique
École normale supérieure, 45 rue d'Ulm
75230 Paris cedex 05, France
cousot@ens.fr

Laboratoire d'informatique
École polytechnique
91128 Palaiseau cedex, France
rcousot@lix.polytechnique.fr

ABSTRACT

We introduce a general uniform language-independent framework for designing online and offline source-to-source program transformations by abstract interpretation of program semantics. Iterative source-to-source program transformations are designed constructively by composition of source-to-semantics, semantics-to-transformed semantics and semantics-to-source abstractions applied to fixpoint trace semantics. The correctness of the transformations is expressed through observational and performance abstractions. The framework is illustrated on three examples: constant propagation, program specialization by online and offline partial evaluation and static program monitoring.

1. INTRODUCTION

A program transformation is a meaning-preserving mapping defined on a programming language [21]. The program transformation methodology provides *thinking tools* for the development of programs from specifications (such as the fold/unfold transformation [24]) and program verification (such as temporal verification [9]). The program transformation techniques provide *mechanical tools* for program optimization (such as cache optimization [12], call-by-name to call-by-value transformation [26], constant propagation [18], continuation passing style transformation [23], deforestation [29], finite differencing [20], partial evaluation [1, 14], transition compression [16]), software customization (such as security policy enforcement [10, 25], reverse engineering [31], slicing [30]) and compilation [22].

The objective of this paper is to introduce a *general uniform language-independent framework for reasoning on semantics-based program transformation.* The formalization is based on abstract interpretation [4, 6] which accounts for:

- the static program analyses that are used to justify transformations (such as binding time analysis [28] for partial evaluation [13, 15]);
- the correctness of transformations which should preserve the semantics, at some level of observation or abstraction of irrelevant details (this aspect is also standard and similar to the use of abstract interpretation to hierarchically organize programming language semantics [3]);
- the efficiency of transformations which should improve program performances as measured by an abstraction of the semantics (the use of abstract interpretation for performance analysis is more recent, see e.g. [19]);
- the formalization of syntactic program transformations as abstractions of program semantics transformations.

This last point is the main novelty of our approach which leads to a *constructive language-independent program transformation design methodology* where the syntactic transformation is constructed systematically by approximation of a semantic transformation which is easily shown to be correct and efficient. As noticed by [21], "Transformational systems may have the power to perform sophisticated program analysis and to generate software at breakneck speed, but to date they are not sound. Lacking from them is a convenient mechanical facility to prove that each transformation preserves semantics. In order to create confidence in the products of transformational systems we need to prove correctness of specifications and transformations. Currently, this is too labor-intensive to be practical." The proposed uniform framework to formalize semantics-based program manipulation will hopefully be a useful step in that direction.

2. A FEW BASIC ELEMENTS OF ABSTRACT INTERPRETATION

Abstract interpretation [4, 6] formalizes the approximation correspondence between the concrete semantics $\mathsf{S}[\![\mathtt{P}]\!]$ of a syntactically correct program $\mathtt{P} \in \mathbb{P}$, chosen in a given programming language $\mathbb{P}$, and an abstract semantics $\overline{\mathsf{S}}[\![\mathtt{P}]\!]$ which is a safe/conservative approximation of the concrete semantics $\mathsf{S}[\![\mathtt{P}]\!]$.

The concrete semantics belongs to a concrete semantic domain $\mathfrak{D}$ which is a poset $\mathtt{po}\langle\mathfrak{D}; \sqsubseteq\rangle$ when partially ordered by the approximation ordering $\sqsubseteq$ formalizing the loss of information (e.g. the logical implication). The abstract semantics is also a poset $\mathtt{po}\langle\overline{\mathfrak{D}}; \overline{\sqsubseteq}\rangle$ which is ordered by the abstract version $\overline{\sqsubseteq}$ of the concrete approximation ordering $\sqsubseteq$. The concrete and abstract semantic domains often enjoy stronger properties, such as being complete partial orderings or complete lattices.

*This work was supported in part by the RTD project IST-1999-20527 DAEDALUS of the european FP5 programme.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

POPL '02, Jan. 16-18, 2002 Portland, OR USA
© 2002 ACM ISBN 1-58113-450-9/02/01...$5.00 .

The correspondence between the concrete and the abstract semantic domains is given by a pair of maps α, which is the abstraction, and γ, which is the concretization. The concretization $\gamma(\overline{\mathsf{S}}[\![\mathtt{P}]\!])$ of the abstract semantics $\overline{\mathsf{S}}[\![\mathtt{P}]\!]$ expresses the abstract information available about program execution in concrete terms. It should be a sound approximation of the concrete semantics in that $\mathsf{S}[\![\mathtt{P}]\!] \sqsubseteq \gamma(\overline{\mathsf{S}}[\![\mathtt{P}]\!])$.

If any element $\mathcal{S}$ of the concrete domain $\mathsf{po}\langle\mathfrak{D}; \sqsubseteq\rangle$ has a best approximation (i.e. $\overline{\sqsubseteq}$-most precise) in the abstract domain $\mathsf{po}\langle\overline{\mathfrak{D}}; \overline{\sqsubseteq}\rangle$ given by $\alpha(\mathcal{S})$, then the pair $\langle\alpha, \gamma\rangle$ is a Galois connection which is written $\mathsf{po}\langle\mathfrak{D}; \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathsf{po}\langle\overline{\mathfrak{D}}; \overline{\sqsubseteq}\rangle$. By definition, this means that $\forall \mathcal{X} \in \mathfrak{D}$: $\forall \mathcal{Y} \in \overline{\mathfrak{D}}$: $\alpha(\mathcal{X}) \mathrel{\overline{\sqsubseteq}} \mathcal{Y} \Leftrightarrow \mathcal{X} \sqsubseteq \gamma(\mathcal{Y})$. A Galois insertion $\mathsf{po}\langle\mathfrak{D}; \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\rightarrow}} \mathsf{po}\langle\overline{\mathfrak{D}}; \overline{\sqsubseteq}\rangle$ is a Galois connection with α surjective.

One interest of the abstract interpretation theory is to constructively derive the *exact* (resp. *approximate*) abstract semantics $\overline{\mathsf{S}}[\![\mathtt{P}]\!]$ from the given concrete semantics $\mathsf{S}[\![\mathtt{P}]\!]$ by refining the specification $\alpha(\mathsf{S}[\![\mathtt{P}]\!]) = \overline{\mathsf{S}}[\![\mathtt{P}]\!]$ (resp. $\alpha(\mathsf{S}[\![\mathtt{P}]\!]) \mathrel{\overline{\sqsubseteq}} \overline{\mathsf{S}}[\![\mathtt{P}]\!]$). If e.g. the concrete semantics is given in fixpoint form $\mathsf{S}[\![\mathtt{P}]\!] = \mathsf{lfp}^{\sqsubseteq}\mathsf{F}[\![\mathtt{P}]\!]$ where the semantic transformer $\mathsf{F}[\![\mathtt{P}]\!]$ is monotonic, then the abstract semantics can be chosen as $\mathsf{lfp}^{\overline{\sqsubseteq}}\overline{\mathsf{F}}[\![\mathtt{P}]\!]$ where the abstract semantic transformer $\overline{\mathsf{F}}[\![\mathtt{P}]\!]$ is designed using the *local commutation conditions* $\alpha \circ \mathsf{F}[\![\mathtt{P}]\!] \circ \gamma = \overline{\mathsf{F}}[\![\mathtt{P}]\!]$, $\alpha \circ \mathsf{F}[\![\mathtt{P}]\!] = \overline{\mathsf{F}}[\![\mathtt{P}]\!] \circ \alpha$ (resp. $\dot{\overline{\sqsubseteq}}$ i.e. $\overline{\sqsubseteq}$ pointwise for *semi-commutation*) or $\mathsf{F}[\![\mathtt{P}]\!] \circ \gamma = \gamma \circ \overline{\mathsf{F}}[\![\mathtt{P}]\!]$ (resp. $\dot{\sqsubseteq}$). Several *fixpoint transfer* theorems e.g. [6, Th. 7.1.0.4(3)], [3, Th. 2.1], [8, Cor. 2.4] (resp. *fixpoint upper approximation* theorems e.g. [6, Th. 7.1.0.4(2)], [8, Th. 2.5]) guarantee that:

$$\alpha(\mathsf{lfp}^{\sqsubseteq}\mathsf{F}[\![\mathtt{P}]\!]) = \mathsf{lfp}^{\overline{\sqsubseteq}}\overline{\mathsf{F}}[\![\mathtt{P}]\!] \quad \big(\text{resp. } \alpha(\mathsf{lfp}^{\sqsubseteq}\mathsf{F}[\![\mathtt{P}]\!]) \mathrel{\overline{\sqsubseteq}} \mathsf{lfp}^{\overline{\sqsubseteq}}\overline{\mathsf{F}}[\![\mathtt{P}]\!]\big) \quad (1)$$

and their duals (with $\sqsupseteq$, $\overline{\sqsupseteq}$, γ substituted for $\sqsubseteq$, $\overline{\sqsubseteq}$ and α).

3. PRINCIPLE OF PROGRAM TRANSFORMATION

In this section we introduce the principle of our formalization of program transformations by abstract interpretation.

3.1 Syntactic Program Transformation

A syntactic program transformer $\mathtt{t}$ takes as input a subject program $\mathtt{P} \in \mathbb{P}$ and, upon termination, produces as output a transformed program $\mathtt{t}[\![\mathtt{P}]\!] \in \mathbb{P}'$ (for short $\mathbb{P}' = \mathbb{P}$):

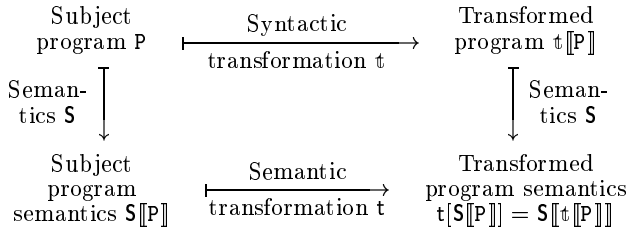
Subject program $\mathtt{P} \in \mathbb{P}$ $\quad\longmapsto$ (Syntactic transformation $\mathtt{t}$) $\quad\longrightarrow$ Transformed program $\mathtt{t}[\![\mathtt{P}]\!] \in \mathbb{P}$

However, program transformations seldom rely on syntactic criteria only, so we now introduce the semantic foundations which are especially needed to determine meaning preservedness of program transformations. We first consider *online program transformations* directly referring to program executions and next *offline transformations* based upon a preliminary static program analysis.

3.2 Semantics-Based Online Program Transformation

Online transformations refer to program executions. For example, online partial evaluation makes use of the concrete values of the static input variables so that, according to [16, Def. 4.6], the concrete values computed during program specialization can affect the choice of action taken.

Formally, an online transformation can be understood as making use of the program semantics $\mathsf{S}[\![\mathtt{P}]\!] \in \mathfrak{D}$. From this formal point of view, any program transformer $\mathtt{t} \in \mathbb{P} \longmapsto \mathbb{P}$ on the program syntax induces a corresponding semantic transformer $\mathtt{t} \in \mathfrak{D} \longmapsto \mathfrak{D}$ taking as input the semantics $\mathsf{S}[\![\mathtt{P}]\!]$ of the subject program $\mathtt{P}$ and producing the semantics $\mathsf{S}[\![\mathtt{t}[\![\mathtt{P}]\!]]\!]$ of the transformed program $\mathtt{t}[\![\mathtt{P}]\!]$. A strong equivalence requirement is that $\mathsf{S}[\![\mathtt{t}[\![\mathtt{P}]\!]]\!] = \mathtt{t}[\mathsf{S}[\![\mathtt{P}]\!]]$ stating that the semantics of the syntactically transformed program is precisely the semantic transformation of the semantics of the subject program:



A generalization consists in considering $\mathtt{P}$ as a tuple of programs, see Sec. 8. We now study in more details the correspondences between the various elements of this diagram.

3.3 Correspondence Between Syntax and Semantics of Programs

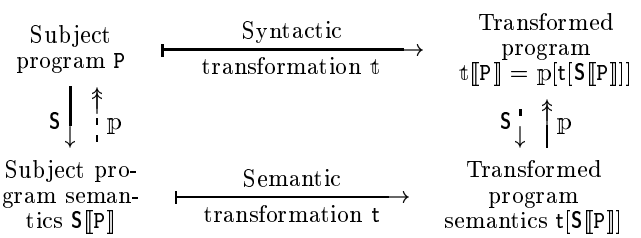
Programs can be considered as an abstraction of their semantics. For example the syntax of programs records the existence of variables and maybe their type but not the sequence of their successive values during execution, as defined by the semantics. Usually programs record the chaining of actions but not their exact sequences of execution. The same way, program performances are completely abstracted in the program syntax although execution time might be included in the operational semantics. Formally:

$$\mathsf{po}\langle\mathfrak{D}; \sqsubseteq\rangle \underset{\mathbb{p}}{\overset{\mathsf{S}}{\leftrightarrows\!\!\!\!\rightarrow}} \mathsf{po}\langle\mathbb{P}/_{\equiv}; \sqsubseteq\rangle \quad (2)$$

where $\mathsf{S}[\![\mathtt{P}]\!]$ is the semantics of program $\mathtt{P} \in \mathbb{P}$ while $\mathbb{p}[\mathcal{S}]$ is the simplest program whose semantics upper-approximates $\mathcal{S} \in \mathfrak{D}$. Programs are considered up to a *syntactic equivalence* $\mathtt{P} \equiv \mathtt{Q} \triangleq (\mathsf{S}[\![\mathtt{P}]\!] = \mathsf{S}[\![\mathtt{Q}]\!])$ ($\triangleq$ means "is defined as"). The *syntactic refinement* is $\mathtt{P} \sqsubseteq \mathtt{Q} \triangleq (\mathsf{S}[\![\mathtt{P}]\!] \sqsubseteq \mathsf{S}[\![\mathtt{Q}]\!])$. In practice, neither $\sqsubseteq$ nor $\equiv$ are computable and they must be approximated (e.g. by $=$ and $\subseteq$ in Sec. 6.6).

3.4 Syntactic Program Transformations as Abstractions of Semantics-Based Program Transformations

Thanks to the above correspondence between the syntax and semantics of programs, the transformed program $\mathtt{t}[\![\mathtt{P}]\!]$ can be viewed as the decompilation $\mathbb{p}[\mathtt{t}[\mathsf{S}[\![\mathtt{P}]\!]]]$ of its semantics $\mathtt{t}[\mathsf{S}[\![\mathtt{P}]\!]]$. Using this correspondence $\langle\mathsf{S}, \mathbb{p}\rangle$ between the syntax and the semantics of a program as well as the semantic form $\mathtt{t}$ of the program syntactic transformation $\mathtt{t}$, we get the following commuting schema (dashed arrows are unused in the explanation):



This schema leads to another view of online program transformation. A syntactic program transformation algorithm $\mathtt{t}[\![\mathtt{P}]\!] = \mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$ is derived by abstraction of its semantic specification $\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]$. In practice the syntactic transformation $\mathtt{t}[\![\mathtt{P}]\!]$ may be weaker, that is more restricted or less effective, than the ideal semantics-based but undecidable transformation $\mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$ so that $\mathtt{t}[\![\mathtt{P}]\!] \sqsupseteq \mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$.

3.5 Correspondence Between Syntactic and Semantic Program Transformations

Formally, the *semantic transformation* $\mathtt{t} \in \mathfrak{D} \longmapsto \mathfrak{D}$ *induced by a syntactic transformation* $\mathbb{t} \in \mathbb{P} \longmapsto \mathbb{P}$ is:

$$\mathtt{t}[\mathcal{S}] \quad \triangleq \quad \mathtt{S}[\![\mathbb{t}[\![\mathbb{p}[\mathcal{S}]]\!]]\!] \ .$$

Conversely, the *syntactic transformation* $\mathbb{t} \in \mathbb{P} \longmapsto \mathbb{P}$ *induced by a semantic transformation* $\mathtt{t} \in \mathfrak{D} \longmapsto \mathfrak{D}$ is:

$$\mathbb{t}[\![\mathtt{P}]\!] \quad \triangleq \quad \mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]] \ . \tag{3}$$

We use (3) as a basis for designing the syntactic transformation $\mathbb{t}[\![\mathtt{P}]\!]$ formally from the semantic transformation $\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]$. Observe that $\mathsf{po}\langle\mathfrak{D};\sqsubseteq\rangle \underset{\mathbb{p}}{\overset{\mathtt{S}}{\leftleftarrows}} \mathsf{po}\langle\mathbb{P}/_{\equiv};\sqsubseteq\rangle$ implies

$$\mathsf{po}\langle\mathfrak{D} \overset{\mathrm{m}}{\longmapsto} \mathfrak{D};\dot{\sqsubseteq}\rangle \underset{\lambda\,\mathtt{t}\cdot\lambda\,\mathtt{P}\cdot\mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]}{\overset{\lambda\,\mathtt{t}\cdot\lambda\,\mathcal{T}\cdot\mathtt{S}[\![\mathbb{t}[\![\mathbb{p}[\mathcal{T}]]\!]]\!]}{\leftleftarrows}} \mathsf{po}\langle\mathbb{P}/_{\equiv} \overset{\mathrm{m}}{\longmapsto} \mathbb{P}/_{\equiv};\dot{\sqsubseteq}\rangle$$

so that for $\sqsubseteq$-monotonic transformations, the source-to-source syntactic transformation is a functional abstraction of the semantic transformation. However, because of undecidability, we can often only compute an effective approximation $\mathbb{t}[\![\mathtt{P}]\!] \sqsupseteq \mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$.

3.6 Correspondence Between the Subject and Transformed Program Semantics

A program transformation corresponds to a loss of information on the subject program whence its semantics. For example, in partial evaluation, the transformed program is specialized for given static values so that the information on how these static values are computed in the subject program is lost in the specialization process. In the abstract interpretation framework, this can be formalized as:

$$\mathsf{po}\langle\mathfrak{D};\sqsubseteq\rangle \quad \underset{\mathtt{t}}{\overset{\gamma_\mathtt{t}}{\leftrightarrows}} \quad \mathsf{po}\langle\mathfrak{D};\sqsubseteq\rangle \ . \tag{4}$$

By composition of the Galois connections (2) and (4), by definition (3) and by $\mathbb{p} \circ \mathtt{S}[\![\mathtt{P}]\!] \equiv \mathtt{P}$ from (2), we have:

$$\mathsf{po}\langle\mathbb{P}/_{\equiv};\sqsubseteq\rangle \underset{\mathbb{t}}{\overset{\gamma_\mathtt{t}}{\leftrightarrows}} \mathsf{po}\langle\mathbb{P}/_{\equiv};\sqsubseteq\rangle \ . \tag{5}$$

3.7 Design of a Program Transformation Algorithm by Abstraction of the Program Fixpoint Semantics

The semantics $\mathtt{S}[\![\mathtt{P}]\!]$ of program $\mathtt{P}$ can often be expressed in least fixpoint form as $\mathsf{lfp}\,\mathtt{F}[\![\mathtt{P}]\!]$ (or dually as $\mathsf{gfp}\,\mathtt{F}[\![\mathtt{P}]\!]$) [3]. From this fixpoint semantic definition $\mathtt{S}[\![\mathtt{P}]\!] = \mathsf{lfp}\ \mathtt{F}[\![\mathtt{P}]\!]$, we constructively derive the semantic and then the syntactic transformations in fixpoint form using the fixpoint transfer theorems (1) successively applied with the abstraction (4) and then (5), as follows:

$\mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$

$= \mathbb{p}[\mathtt{t}[\mathsf{lfp}\,\mathtt{F}[\![\mathtt{P}]\!]]]$ by the fixpoint definition $\mathtt{S}[\![\mathtt{P}]\!] = \mathsf{lfp}\,\mathtt{F}[\![\mathtt{P}]\!]$ of the semantics,

$= \mathbb{p}[\mathsf{lfp}\,\mathtt{F}[\![\mathtt{P}]\!]]$ where $\mathtt{F}[\![\mathtt{P}]\!] \in \mathfrak{D} \overset{\mathrm{m}}{\longmapsto} \mathfrak{D}$ is designed using (1) with abstraction (4),

$= \mathsf{lfp}\,\mathbb{F}[\![\mathtt{P}]\!]$ by (1) with (5),

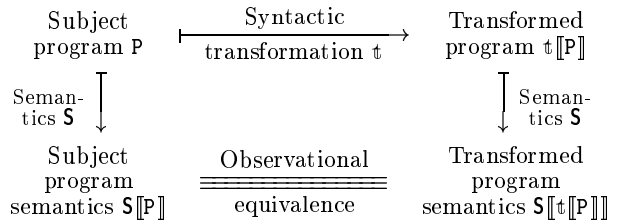
$\triangleq \mathbb{t}[\![\mathtt{P}]\!]$ (resp. $\mathbb{U}$ for approximations).

When the fixpoint $\mathsf{lfp}\,\mathbb{F}[\![\mathtt{P}]\!]$ is defined on posets satisfying the ascending chain condition, this fixpoint characterization $\mathbb{t}[\![\mathtt{P}]\!] = \mathsf{lfp}\,\mathbb{F}[\![\mathtt{P}]\!]$ of the syntactic program transformation $\mathbb{t}[\![\mathtt{P}]\!]$ directly leads to an iterative algorithm.

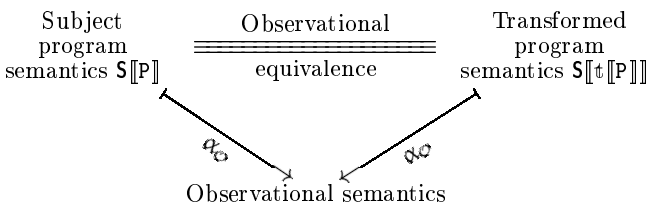
However, the semantic transformation $\mathtt{t}$ can depend on undecidable results on the program semantics so that the syntactic transformation algorithm $\mathbb{t}[\![\mathtt{P}]\!]$ may not terminate or, even worse, fixpoint transfer theorems (1) may not be applicable. In this case, weaker approximate transformations must be considered which depend upon decidable criteria only. This leads to the idea of offline program transformation considered in Sec. 3.10. But before, we study semantics-based correctness criteria of program transformation.

3.8 Correctness of Online Program Transformation: Observational Abstraction

Another advantage of understanding syntactic program transformation as an abstraction of a semantics transformation is that it naturally leads to a simple notion of correctness of the program transformation. A transformation is correct iff, at some level of abstraction, the observation of the execution of the subject program is equivalent to the observation of the execution of the transformed program:



Observational equivalence can be formalized in the abstract interpretation framework by requiring the abstraction of the semantics of the subject and of the transformed programs to be identical. The specification of the observational abstraction $\alpha_\mathcal{O}$ should be considered part of the problematics. For example,BT ignoring the termination problem [16, Ch. 4, note on p. 69] can be formalized by an observational abstraction ignoring all infinite program behaviors. Schematically:



Formally, a source-to-source program transformation $\mathbb{t} \in \mathbb{P} \longmapsto \mathbb{P}$ is said to be *correct with respect to an observational abstraction*:

$$\mathsf{po}\langle\mathfrak{D};\sqsubseteq\rangle \quad \underset{\alpha_\mathcal{O}}{\overset{\gamma_\mathcal{O}}{\leftrightarrows}} \quad \mathsf{po}\langle\mathcal{D}_\mathcal{O};\sqsubseteq_\mathcal{O}\rangle \tag{6}$$

if and only if for all programs $\mathtt{P} \in \mathbb{P}$, $\alpha_\mathcal{O}(\mathtt{S}[\![\mathtt{P}]\!]) = \alpha_\mathcal{O}(\mathtt{S}[\![\mathbb{t}[\![\mathtt{P}]\!]]\!])$, as shown in diagramed form, in **Fig. 1**. More generally, programs $\mathtt{P}$ and $\mathtt{Q}$ are said to be $\alpha_\mathcal{O}$-*observationally equivalent*, written $\mathtt{P} \equiv_\mathcal{O} \mathtt{Q}$, if and only if:

$$\alpha_\mathcal{O}(\mathtt{S}[\![\mathtt{P}]\!]) \quad = \quad \alpha_\mathcal{O}(\mathtt{S}[\![\mathtt{Q}]\!]) \ . \tag{7}$$

3.9 Principle of Online Program Transformation

Summarizing this point of view on online program transformation, we get the schematic diagram of **Fig. 2** including a formalization of the transformation correctness through a

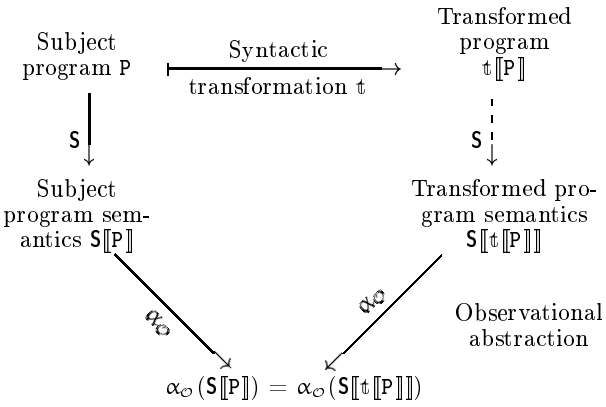


Figure 1: Correctness of a syntactic transformation

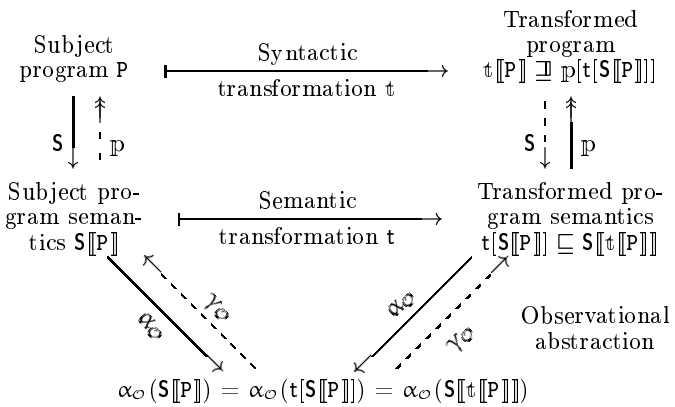


Figure 2: Online program transformation

semantics observational abstraction $\alpha_{\mathcal{O}}$. According to this diagram, the syntactic transformation $\mathbb{t}[\![\mathtt{P}]\!]$ is designed as an upper-approximation of the best possible one $\mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$. This consists in simplifying the term $\mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]]$ in order to get rid of all semantic subterms in $\mathtt{S}$ and $\mathtt{t}$.

By definition of the Galois connection (2), the correctness condition $\mathbb{p}[\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]] \mathbb{\sqsubseteq} \mathbb{t}[\![\mathtt{P}]\!]$ is equivalent to $\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]] \sqsubseteq \mathtt{S}[\![\mathbb{t}[\![\mathtt{P}]\!]]\!]$. This equivalence leads to an alternative method for designing the syntactic transformation $\mathbb{t}$. Starting from the given semantic transformation $\mathtt{t}$, the term $\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]$ is simplified by pushing out $\mathtt{S}$ in order to rewrite it in the form $\mathtt{S}[\![\mathbb{t}[\![\mathtt{P}]\!]]\!]$ so as to extract the definition of $\mathbb{t}[\![\mathtt{P}]\!]$. This methodology, which is quite common in abstract interpretation (e.g. [3]), is illustrated in Sec. 7.1.4.

In both cases, the transformation is an upper-approximation and so must be proved correct. To do so we prove that $\mathtt{S}[\![\mathtt{P}]\!] \sqsubseteq \mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]$ and $\alpha_{\mathcal{O}}(\mathtt{S}[\![\mathtt{P}]\!]) = \alpha_{\mathcal{O}}(\mathtt{S}[\![\mathbb{t}[\![\mathtt{P}]\!]]\!])$. Then, by (6), $\alpha_{\mathcal{O}}$ is monotonic so $\alpha_{\mathcal{O}}(\mathtt{S}[\![\mathtt{P}]\!]) \sqsubseteq_{\mathcal{O}} \alpha_{\mathcal{O}}(\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]) \sqsubseteq_{\mathcal{O}} \alpha_{\mathcal{O}}(\mathtt{S}[\![\mathbb{t}[\![\mathtt{P}]\!]]\!]) = \alpha_{\mathcal{O}}(\mathtt{S}[\![\mathtt{P}]\!])$. By antisymmetry, we conclude that $\alpha_{\mathcal{O}}(\mathtt{S}[\![\mathtt{P}]\!]) = \alpha_{\mathcal{O}}(\mathtt{t}[\mathtt{S}[\![\mathtt{P}]\!]]) = \alpha_{\mathcal{O}}(\mathtt{S}[\![\mathbb{t}[\![\mathtt{P}]\!]]\!])$.

3.10 Principle of Offline Program Transformation

Although *offline* program transformation does not directly use the values of variables during program execution, it nevertheless uses some information on program execution, which is obtained by a preliminary static program analysis. For example, in partial evaluation, a preliminary binding time analysis is used to compute a division of program variables into static and dynamic ones, the transformation being only applied to static ones [16]. From a syntactic point of view, the preliminary static analysis phase is used to add annotations to the program and then the transformation is applied to the annotated program. This leads to the schema of **Fig. 3**, which is the schema given for online program transformation in Sec. 3.9, but for the fact that it is applied to an annotated program $\underline{\mathtt{P}}$ derived from the subject program $\mathtt{P}$ by a preliminary static analysis based annotation algorithm. Although the preliminary syntactic annotation phase can be very useful as a user interface, one can isolate the preliminary program static phase as an abstract interpretation of the program semantics as specified by a static analysis Galois connection

$$\mathtt{po}\langle\mathfrak{D};\sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathtt{po}\langle\overline{\mathfrak{D}};\overline{\sqsubseteq}\rangle \tag{8}$$

and consider syntactic transformations acting on the program $\mathtt{P}$ given its abstract semantics $\overline{\mathtt{S}}[\![\mathtt{P}]\!]$, as shown in **Fig. 4**. We now exemplify this framework on elementary program transformations of a simple imperative programming language.

3.11 Transformation Combinations

Since the composition of Galois connections is a Galois connection, the transformation diagrams of **Fig. 2**, **Fig. 3** and **Fig. 4** can be combined serially or in parallel (for multi-programs transformations as examplified in Sec. 8) to explain complex combinations of transformations. For example, the reduced product [6] of constant propagation (Sec. 6) and online partial evaluation (Sec. 7.1) leads to an offline partial evaluator where the values of some of the static variables is detected by the preliminary constant detection analysis (Sec. 6.2).

4. SYNTAX AND SEMANTICS OF THE EXAMPLE PROGRAMMING LANGUAGE

Let us consider imperative iterative programs acting on global variables such as e.g.

```
X := ?; while X > 0 do X := X + 1 od
```

that would be written:

$\mathtt{a} : \mathtt{X} := ? \to \mathtt{b};$	$\mathtt{c} : \mathtt{X} := \mathtt{X} + 1 \to \mathtt{d};$
$\mathtt{b} : (\mathtt{X} > 0) \to \mathtt{c};$	$\mathtt{d} : \mathtt{skip} \to \mathtt{b};$
$\mathtt{b} : \neg(\mathtt{X} > 0) \to \mathtt{e};$	$\mathtt{e} : \mathtt{stop};$

in the example language $\mathbb{P}$. If execution is at some label $\mathtt{L}$ then one of the transitions $\mathtt{L} : \mathtt{A} \to \mathtt{L}';$ labeled with $\mathtt{L}$ is executed, provided the action $\mathtt{A}$ is not blocking and the execution can go on by branching to the next label $\mathtt{L}'$. Programs are nondeterministic since several actions can be referenced by the same label. If no action is labelled $\mathtt{L}'$, the execution is blocked at $\mathtt{L}$, which is the case for the `stop` command $\mathtt{L} : \mathtt{stop};$ which is a shorthand for $\mathtt{L} : \mathtt{skip} \to \mathtt{l};$ where $\mathtt{l}$ is the undefined label. The `skip` command $\mathtt{L} : \mathtt{skip} \to \mathtt{L};$ is itself a shorthand for the boolean test $\mathtt{L} : \mathtt{true} \to \mathtt{L};$.

Formally, programs are <u>not</u> restricted to be finite. This is useful to discuss program transformations such as partial evaluation which may not terminate. However, in practice, program transformations are required to be effective so as to produce finite transformed programs out of finite subject programs.

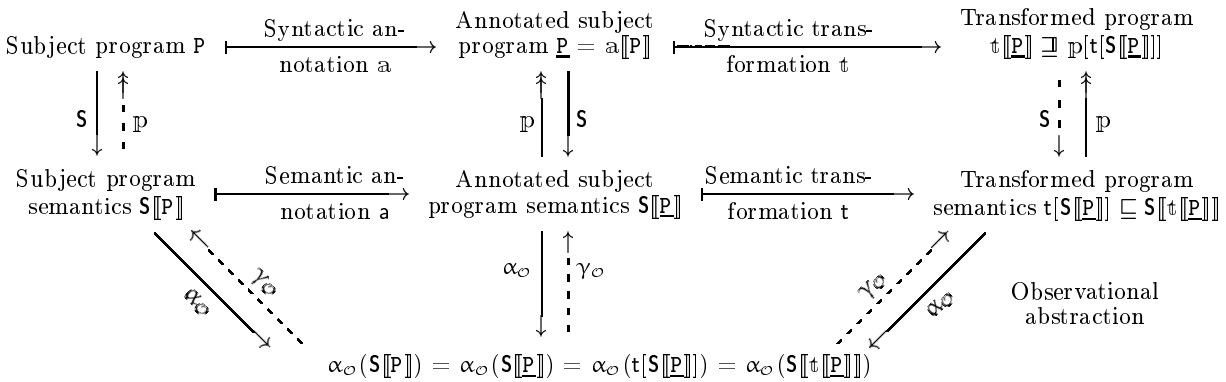


Figure 3: Offline program transformation with annotations

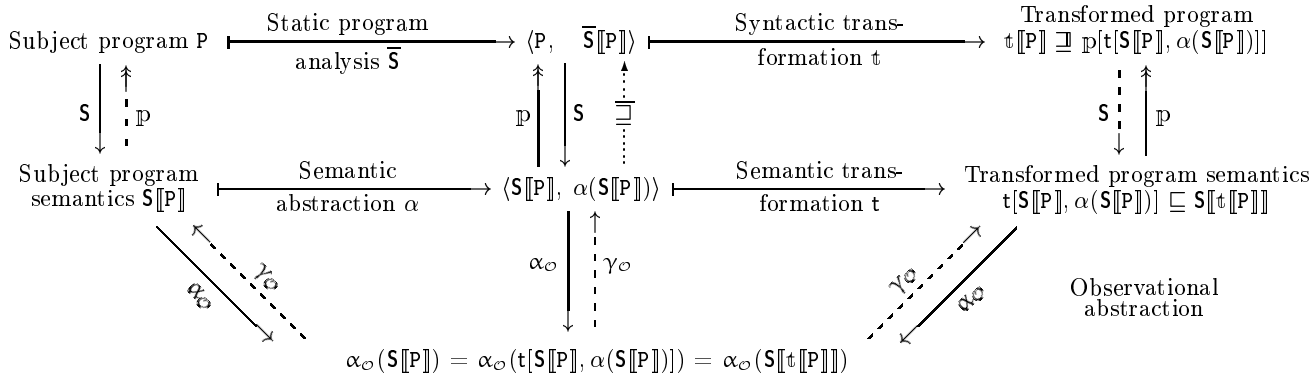


Figure 4: Offline abstract program transformation

4.1 Abstract Syntax of Programs

The abstract syntax of programs is defined in **Fig. 5**. We let $\mathtt{var}[\![\mathtt{D}]\!]$ be the set of variables of an expression or action $\mathtt{D} \in \mathbb{E} \cup \mathbb{B} \cup \mathbb{A}$ and define:

$$\mathtt{lab}[\![\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2\,;]\!] \triangleq \mathtt{L}_1, \quad \mathtt{var}[\![\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2\,;]\!] \triangleq \mathtt{var}[\![\mathtt{A}]\!],$$
$$\mathtt{act}[\![\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2\,;]\!] \triangleq \mathtt{A}, \quad \mathtt{lab}[\![\mathtt{P}]\!] \triangleq \{\mathtt{lab}[\![\mathtt{C}]\!] \mid \mathtt{C} \in \mathtt{P}\},$$
$$\mathtt{suc}[\![\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2\,;]\!] \triangleq \mathtt{L}_2, \quad \mathtt{var}[\![\mathtt{P}]\!] \triangleq \bigcup_{\mathtt{C} \in \mathtt{P}} \mathtt{var}[\![\mathtt{C}]\!] .$$

A `stop` command is $\mathtt{L} : \mathtt{stop}\,; \Box\, \mathtt{L} : \mathtt{skip} \to \mathtt{l}\,;$ and a `skip` command is $\mathtt{L} : \mathtt{skip} \to \mathtt{L}\,; \Box\, \mathtt{L} : \mathtt{true} \to \mathtt{L}\,;$.

4.2 Environments

The commands of a program P act on global variables $\mathtt{X} \in \mathtt{var}[\![\mathtt{P}]\!]$. The program variables take their value in the semantic domain $\mathfrak{V}$. This value can be the uninitialized or undefined value $\mho \notin \mathfrak{V}$ so $\mathfrak{V}_\mho \triangleq \mathfrak{V} \cup \{\mho\}$ [1]. An *environment* $\rho \in \mathfrak{E}$ maps variables $\mathtt{X} \in \mathbf{dom}[\rho]$ to their value $\rho(\mathtt{X})$ so $\mathfrak{E} \triangleq \bigcup_{\mathcal{X} \subseteq \mathbb{X}} \mathfrak{E}[\![\mathcal{X}]\!]$ where $\mathfrak{E}[\![\mathcal{X}]\!] \triangleq \mathcal{X} \longmapsto \mathfrak{V}_\mho$ is the subset of environments ρ with *domain* $\mathbf{dom}[\rho] \triangleq \mathcal{X}$. In particular the *empty environment* $\mathfrak{d}$ is the only environment with empty domain $\mathbf{dom}[\mathfrak{d}] = \emptyset$ so that $\mathfrak{E}[\![\emptyset]\!] \triangleq \emptyset \longmapsto \mathfrak{V}_\mho = \{\mathfrak{d}\}$. $\mathfrak{E}[\![\mathtt{P}]\!]$ is the set of environments of a program P whose domain is the set of program variables: $\mathfrak{E}[\![\mathtt{P}]\!] \triangleq \mathfrak{E}[\![\mathtt{var}[\![\mathtt{P}]\!]]\!]$.

$\rho|_\mathcal{X}$ where $\mathcal{X} \subseteq \mathbb{X}$ is the restriction of environment ρ to the domain $\mathbf{dom}[\rho] \cap \mathcal{X}$. We write $\rho \setminus \mathcal{X}$ for the restriction of environment ρ to the variables not in $\mathcal{X}$. Therefore $\rho \setminus \mathcal{X}$ is the environment ρ' such that $\mathbf{dom}[\rho'] = \{\mathtt{X} \in \mathbf{dom}[\rho] \mid \mathtt{X} \notin \mathcal{X}\}$ and $\forall \mathtt{X} \in \mathbf{dom}[\rho'] : \rho'(\mathtt{X}) = \rho(\mathtt{X})$. For short we write $\rho \setminus \mathtt{X}$ for $\rho \setminus \{\mathtt{X}\}$ when $\mathtt{X} \in \mathbb{X}$ is a program variable and $\mathcal{R} \setminus \mathcal{X} \triangleq \{\rho \setminus \mathcal{X} \mid \rho \in \mathcal{R}\}$ when $\mathcal{R}$ is a set of environments. Let us write $\rho \Subset \rho'$ if ρ is identical to ρ' on its domain that is $\mathbf{dom}[\rho] \subseteq \mathbf{dom}[\rho']$ and $\forall \mathtt{X} \in \mathbf{dom}[\rho] : \rho(\mathtt{X}) = \rho'(\mathtt{X})$.

4.3 Semantics of Program Actions

The semantics $\mathbf{A}[\![\mathtt{E}]\!]$ of an arithmetic expression E is defined inductively[2]:

$$\mathbf{A}[\![\mathtt{n}]\!]\rho \triangleq \mathtt{n}, \qquad \mathbf{A}[\![\mathtt{X}]\!]\rho \triangleq \rho(\mathtt{X}),$$
$$\mathbf{A}[\![\mathtt{E}_1 - \mathtt{E}_2]\!]\rho \triangleq \mathbf{A}[\![\mathtt{E}_1]\!]\rho - \mathbf{A}[\![\mathtt{E}_2]\!]\rho$$

where $-$ is extended to the undefined value $\mho$ as $\mho - \mho \triangleq \mho - n \triangleq n - \mho \triangleq \mho$. $\mathbf{A}[\![\mathtt{E}]\!] \in \mathfrak{E}[\![\mathcal{X}]\!] \longmapsto \mathfrak{V}_\mho$ is well-defined when $\mathtt{var}[\![\mathtt{E}]\!] \subseteq \mathcal{X}$ whence $\mathbf{A}[\![\mathtt{E}]\!] \in \mathfrak{E}[\![\mathtt{P}]\!] \longmapsto \mathfrak{V}_\mho$ is well-defined when the arithmetic expression E belongs to a program P. Similarly, we define the semantics $\mathbf{B}[\![\mathtt{B}]\!] \in$

[1] Instead of using a special undefined value $\mho$ raising error exceptions when used at runtime, we could have chosen to have variables initialized to a specific value (like 0) or to a random value (in practice often depending upon previous memory states). Formally, these last choices can be enforced by adding (random) initialization assignment commands to the program.

[2] For simplicity, here and afterwards, we do not distinguish between values $n = \mathtt{valofstr}(\mathtt{n})$ and their denotations $\mathtt{n} = \mathtt{strofval}(n)$ (including for booleans so `true` denotes `true`). In particular the denotation of the undefined value $\mho$ is also writen $\mho$ but should be some uninitialized variable `Undefined`.

$\mathtt{n}$	:	$\mathbb{Z}$		Integer numbers
$\mathtt{X}$	:	$\mathbb{X}$		Program variables
$\mathtt{E}$	:	$\mathbb{E}$		Arithmetic expressions
$\mathtt{E}$	::=	$\mathtt{n}$		Integer
	|	$\mathtt{X}$		Variable
	|	$\mathtt{E}_1 - \mathtt{E}_2$		Difference
$\mathtt{B}$	:	$\mathbb{B}$		Boolean expressions
$\mathtt{B}$	::=	$\mathtt{E}_1 < \mathtt{E}_2$		Comparison
	|	$\mathtt{B}_1 \vee \mathtt{B}_2$		Disjunction
	|	$\neg\mathtt{B}_1$		Negation
	|	$\mathtt{true} \mid \mathtt{false}$		Truth/Falsity
$\mathtt{A}$	:	$\mathbb{A}$		Program actions
$\mathtt{A}$	::=	$\mathtt{X} := \mathtt{E}$		Assignment
	|	$\mathtt{X} := ?$		Random assignment
	|	$\mathtt{B}$		Test
$\mathtt{L}$	:	$\mathbb{L}$		Program labels
$\mathfrak{l}$	$\notin$	$\mathbb{L}$		Undefined label
$\mathbb{L}$	:	$\mathbb{L} \triangleq \mathbb{L} \cup \{\mathfrak{l}\}$		Colabels
$\mathtt{C}$	:	$\mathbb{C}$		Commands
$\mathtt{C}$	::=	$\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2;$		Transition command
$\mathtt{P}$	:	$\mathbb{P} \triangleq \wp(\mathbb{C})$		Programs

Figure 5: Abstract syntax of programs

$\mathfrak{E}[\![\mathcal{X}]\!] \longmapsto \mathfrak{B}_\upsilon$ of a boolean expression $\mathtt{B}$ with $\mathsf{var}[\![\mathtt{B}]\!] \subseteq \mathcal{X}$, $\mathfrak{B} \triangleq \{\mathtt{true}, \mathtt{false}\}$ and $\mathfrak{B}_\upsilon \triangleq \mathfrak{B} \cup \{\upsilon\}$:

$$\mathsf{B}[\![\mathtt{E}_1 < \mathtt{E}_2]\!]\rho \triangleq \mathsf{A}[\![\mathtt{E}_1]\!]\rho < \mathsf{A}[\![\mathtt{E}_2]\!]\rho, \qquad \mathsf{B}[\![\neg\mathtt{B}]\!]\rho \triangleq \neg\mathsf{B}[\![\mathtt{B}]\!]\rho,$$

$$\mathsf{B}[\![\mathtt{B}_1 \vee \mathtt{B}_2]\!]\rho \triangleq \mathsf{B}[\![\mathtt{B}_1]\!]\rho \vee \mathsf{B}[\![\mathtt{B}_2]\!]\rho, \qquad \mathsf{B}[\![\mathtt{true}]\!]\rho \triangleq \mathtt{true},$$

where $\neg\mathtt{true} = \mathtt{false}$, $\neg\mathtt{false} = \mathtt{true}$ and the undefined value υ is propagated as in $\upsilon < \upsilon \triangleq \upsilon < n \triangleq n < \upsilon \triangleq \upsilon$, $\neg\upsilon = \upsilon$, etc. The *semantics* $\mathsf{S}[\![\mathtt{A}]\!]\rho$ of an action $\mathtt{A}$ in a program $\mathtt{P}$ defines the effect of executing this action on the environment ρ. Because of nondeterministic executions, we define $\mathsf{S} \in \mathbb{A} \longmapsto (\mathfrak{E}[\![\mathcal{X}]\!] \longmapsto \wp(\mathfrak{E}[\![\mathcal{X}]\!]))$ where $\mathsf{S}[\![\mathtt{A}]\!]\rho$ is well-defined when $\mathsf{var}[\![\mathtt{A}]\!] \subseteq \mathsf{dom}[\rho]$:

$$\mathsf{S}[\![\mathtt{B}]\!]\rho \triangleq \{\rho' \mid \mathsf{B}[\![\mathtt{B}]\!]\rho' = \mathtt{true} \wedge \rho' = \rho\},$$

$$\mathsf{S}[\![\mathtt{X} := ?]\!]\rho \triangleq \{\rho' \mid \exists z \in \mathbb{Z} : \rho' = \rho[\mathtt{X} := z]\},$$

$$\mathsf{S}[\![\mathtt{X} := \mathtt{E}]\!]\rho \triangleq \{\rho[\mathtt{X} := \mathsf{A}[\![\mathtt{E}]\!]\rho]\}, \quad \mathsf{S}[\![\mathtt{true}]\!]\rho = \mathsf{S}[\![\mathtt{skip}]\!]\rho = \{\rho\} .$$

4.4 Small-Step Operational Semantics of Programs

A *state* $s \in \mathfrak{S} \triangleq \mathfrak{E} \times \mathbb{C}$ is a pair $s = \langle \rho, \mathtt{C} \rangle$ where the environment ρ records the values of variables while $\mathtt{C}$ is the next command to be executed. The set of states $\mathfrak{S}[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ is defined as:

$$\mathfrak{S}[\![\mathtt{P}]\!] \triangleq \mathfrak{E}[\![\mathtt{P}]\!] \times \mathtt{P} .$$

The *transition relation* $\mathsf{S} \in \mathfrak{S} \longmapsto \wp(\mathfrak{S})$ specifies which successor states s' can follow a given state s:

$$\mathsf{S}(\langle \rho, \mathtt{C} \rangle) \triangleq \{\langle \rho', \mathtt{C}' \rangle \mid \rho' \in \mathsf{S}[\![\mathsf{act}[\![\mathtt{C}]\!]]\!]\rho \wedge \mathsf{suc}[\![\mathtt{C}]\!] = \mathsf{lab}[\![\mathtt{C}']\!]\} .$$

The *transitional semantics* $\mathsf{S}[\![\mathtt{P}]\!] \in \mathfrak{S}[\![\mathtt{P}]\!] \longmapsto \wp(\mathfrak{S}[\![\mathtt{P}]\!])$ of a program $\mathtt{P} \in \mathbb{P}$ restricts the transition relation to program commands:

$$\mathsf{S}[\![\mathtt{P}]\!]\langle \rho, \mathtt{C} \rangle \triangleq \{\langle \rho', \mathtt{C}' \rangle \in \mathsf{S}(\langle \rho, \mathtt{C} \rangle) \mid \rho, \rho' \in \mathfrak{E}[\![\mathtt{P}]\!] \wedge \mathtt{C}' \in \mathtt{P}\} .$$

4.5 Partial Trace Semantics of Programs

We let $\mathfrak{D}^*$ be the set of *finite partial traces*. $\mathfrak{D}^*$ is therefore defined as the set of all sequences σ of states of length $\#\sigma \geq 0$ such that any state σ_i, $i \in [1, \#\sigma[$ in the trace is a possible successor of the previous state σ_{i-1}: $\sigma_i \in \mathsf{S}(\sigma_{i-1})$ [3]. The set $\mathsf{S}^*[\![\mathtt{P}]\!] \subseteq \mathfrak{D}^*$ of finite partial traces of a program $\mathtt{P}$ is, in fixpoint form, $\mathsf{S}^*[\![\mathtt{P}]\!] = \mathsf{lfp}^{\subseteq} \mathsf{F}^*[\![\mathtt{P}]\!]$ where, for the backward case:

$$\mathsf{F}^*[\![\mathtt{P}]\!]\mathcal{T} \triangleq \mathfrak{S}[\![\mathtt{P}]\!] \cup \{ss'\sigma \mid s' \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge s'\sigma \in \mathcal{T}\},$$

and similarly for the forward case. If we are only interested in those executions of a program $\mathtt{P}$ starting from a given set $\mathfrak{L}[\![\mathtt{P}]\!]$ of entry points so that $\mathfrak{I}[\![\mathtt{P}]\!] \triangleq \{\langle \rho, \mathtt{C} \rangle \mid \rho \in \mathfrak{E} \wedge \mathsf{var}[\![\mathtt{P}]\!] \subseteq \mathsf{dom}[\rho] \wedge \mathtt{C} \in \mathtt{P} \wedge \mathsf{lab}[\![\mathtt{C}]\!] \in \mathfrak{L}[\![\mathtt{P}]\!]\}$ is the set of initial states, we can consider the partial trace semantics $\mathsf{S}_\iota^*[\![\mathtt{P}]\!] \subseteq \mathfrak{D}^*$ of $\mathtt{P}$ which is the set of partial traces $\sigma \in \mathfrak{D}^*$ starting from an initial state $\sigma_0 \in \mathfrak{I}[\![\mathtt{P}]\!]$. The partial trace semantics $\mathsf{S}_\iota^*[\![\mathtt{P}]\!]$ can be expressed in fixpoint form as $\mathsf{lfp}^{\subseteq} \mathsf{F}_\iota^*[\![\mathtt{P}]\!]$ where:

$$\mathsf{F}_\iota^*[\![\mathtt{P}]\!]\mathcal{T} \triangleq \mathfrak{I}[\![\mathtt{P}]\!] \cup \{\sigma ss' \mid \sigma s \in \mathcal{T} \wedge s' \in \mathsf{S}[\![\mathtt{P}]\!]s\} .$$

4.6 Correspondence Between Program Syntax and Semantics

The trace semantics maps programs to sets of traces. Conversely, we map sets of traces to programs by collecting commands executed along traces so $\mathbb{p}^* \in \wp(\mathfrak{D}^*) \longmapsto \mathbb{P}$ is:

$$\mathbb{p}^*[\mathcal{T}] \triangleq \{\mathtt{C} \mid \exists \sigma \in \mathcal{T} : \exists i \in [0, \#\sigma[: \exists \rho \in \mathfrak{E} : \sigma_i = \langle \rho, \mathtt{C} \rangle\} .$$

Following (2), we have a Galois connection:

$$\mathsf{po}\langle \wp(\mathfrak{D}^*); \subseteq \rangle \underset{\mathbb{p}^*}{\overset{\mathsf{S}_\iota^*}{\leftrightarrows}} \mathsf{po}\langle \mathbb{P}/\!\equiv; \sqsubseteq \rangle$$

where $\mathbb{P}/\!\equiv$ is the quotient of $\mathbb{P}$ by the syntactic equivalence $\mathtt{P} \equiv \mathtt{Q}$ and $\mathtt{P} \sqsubseteq \mathtt{Q}$ is the syntactic refinement. The Galois connection follows from $\mathbb{p}^*$ and S_ι^* are monotonic, $\mathbb{p}^* \circ \mathsf{S}_\iota^*[\![\mathtt{P}]\!]$ is $\mathtt{P}$ up to dead code elimination and equivalences such as $\mathtt{L} : \mathtt{stop}; \equiv \mathtt{L} : \mathtt{skip} \to \mathfrak{l};$ so $\mathbb{p}^* \circ \mathsf{S}_\iota^*[\![\mathtt{P}]\!] \sqsubseteq \mathtt{P}$ and $\mathcal{T} \subseteq \mathsf{S}_\iota^* \circ \mathbb{p}^*[\mathcal{T}]$ since all commands along the traces of $\mathcal{T}$ are collected.

5. ACTION SPECIALIZATION

5.1 Definition of Expression Specialization

The *residual* of an arithmetic or boolean expression $\mathtt{E}$ in a given (so-called "static") environment ρ (assigning "static" values to the "static" variables $\mathsf{dom}[\rho]$) is the expression $\mathsf{R}[\![\mathtt{E}]\!]\rho$ resulting from the specialization of that expression $\mathtt{E}$ to that environment ρ. Expression specialization is defined in **Fig. 6** (where values and their denotations are once again confounded).

An expression $\mathtt{D} \in \mathbb{E} \cup \mathbb{B}$ is *static* in the (so-called "static") environment ρ (written $\mathsf{static}[\![\mathtt{D}]\!]\rho$) if and only if it can be fully evaluated in this environment ρ, that is $\mathsf{var}[\![\mathtt{D}]\!] \subseteq \mathsf{dom}[\rho]$. Otherwise $\mathsf{var}[\![\mathtt{D}]\!] \not\subseteq \mathsf{dom}[\rho]$ and the expression $\mathtt{D}$ is *dynamic* in the environment ρ.

The specialization of an arithmetic expression $\mathtt{E} \in \mathbb{E}$ which is static in an environment ρ always yields a static value (i.e. a constant): $\mathsf{static}[\![\mathtt{E}]\!]\rho = (\mathsf{R}[\![\mathtt{E}]\!]\rho \in \mathfrak{V}_\upsilon)$ and similarly for boolean expressions $\mathtt{B} \in \mathbb{B}$: $\mathsf{static}[\![\mathtt{B}]\!]\rho = (\mathsf{R}[\![\mathtt{B}]\!]\rho \in \mathfrak{B}_\upsilon)$.

[3] For short we exclude infinite traces. This is not a problem for the considered safety-preserving example program transformations because if sets of prefix-closed finite traces are observationally equivalent then so is their limit. Otherwise, see [8].

$$
\begin{array}{rcl}
\mathsf{R} & \in & \mathbb{E} \longmapsto \mathfrak{E} \longmapsto \mathbb{E} \\
\mathsf{R}[\![\mathtt{n}]\!]\rho & \triangleq & \mathtt{n} \\
\mathsf{R}[\![\mathtt{X}]\!]\rho & \triangleq & \text{if } \mathtt{X} \in \mathsf{dom}[\rho] \text{ then } \rho(\mathtt{X}) \text{ else } \mathtt{X} \\
\mathsf{R}[\![\mathtt{E}_1 - \mathtt{E}_2]\!]\rho & \triangleq & \text{let } \mathtt{E}_1' = \mathsf{R}[\![\mathtt{E}_1]\!]\rho \text{ and } \mathtt{E}_2' = \mathsf{R}[\![\mathtt{E}_2]\!]\rho \text{ in} \\
 & & \quad \text{if } \mathtt{E}_1' = \mho \text{ or } \mathtt{E}_2' = \mho \text{ then } \mho \\
 & & \quad \text{elsif } \mathtt{E}_1' = \mathtt{n}_1 \text{ and } \mathtt{E}_2' = \mathtt{n}_2 \text{ and } \mathtt{n} = \mathtt{n}_1 - \mathtt{n}_2 \\
 & & \quad\quad \text{then } \mathtt{n} \text{ else } \mathtt{E}_1' - \mathtt{E}_2' \\
\mathsf{R} & \in & \mathbb{B} \longmapsto \mathfrak{E} \longmapsto \mathbb{B} \\
\mathsf{R}[\![\mathtt{E}_1 < \mathtt{E}_2]\!]\rho & \triangleq & \text{let } \mathtt{E}_1' = \mathsf{R}[\![\mathtt{E}_1]\!]\rho \text{ and } \mathtt{E}_2' = \mathsf{R}[\![\mathtt{E}_2]\!]\rho \text{ in} \\
 & & \quad \text{if } \mathtt{E}_1' = \mho \text{ or } \mathtt{E}_2' = \mho \text{ then } \mho \\
 & & \quad \text{elsif } \mathtt{E}_1' = \mathtt{n}_1 \text{ and } \mathtt{E}_2' = \mathtt{n}_2 \text{ and } \mathtt{b} = \mathtt{n}_1 < \mathtt{n}_2 \\
 & & \quad\quad \text{then } \mathtt{b} \text{ else } \mathtt{E}_1' < \mathtt{E}_2' \\
\mathsf{R}[\![\mathtt{B}_1 \vee \mathtt{B}_2]\!]\rho & \triangleq & \text{let } \mathtt{B}_1' = \mathsf{R}[\![\mathtt{B}_1]\!]\rho \text{ and } \mathtt{B}_2' = \mathsf{R}[\![\mathtt{B}_2]\!]\rho \text{ in} \\
 & & \quad \text{if } \mathtt{B}_1' = \mho \text{ or } \mathtt{B}_2' = \mho \text{ then } \mho \\
 & & \quad \text{elsif } \mathtt{B}_1' = \mathtt{true} \text{ or } \mathtt{B}_2' = \mathtt{true} \text{ then } \mathtt{true} \\
 & & \quad \text{elsif } \mathtt{B}_1' = \mathtt{false} \text{ then } \mathtt{B}_2' \\
 & & \quad \text{elsif } \mathtt{B}_2' = \mathtt{false} \text{ then } \mathtt{B}_1' \\
 & & \quad \text{else } \mathtt{B}_1' \vee \mathtt{B}_2' \\
\mathsf{R}[\![\neg \mathtt{B}]\!]\rho & \triangleq & \text{let } \mathtt{B}' = \mathsf{R}[\![\mathtt{B}]\!]\rho \text{ in} \\
 & & \quad \text{if } \mathtt{B}' = \mho \text{ then } \mho \\
 & & \quad \text{elsif } \mathtt{B}' = \mathtt{true} \text{ then } \mathtt{false} \\
 & & \quad \text{elsif } \mathtt{B}' = \mathtt{false} \text{ then } \mathtt{true} \\
 & & \quad \text{else } \neg \mathtt{B}' \\
\mathsf{R}[\![\mathtt{true}]\!]\rho & \triangleq & \mathtt{true} \qquad \mathsf{R}[\![\mathtt{false}]\!]\rho \triangleq \mathtt{false}
\end{array}
$$

Figure 6: Expression specialization

5.2 Correctness of Expression Specialization

Intuitively, expression specialization is correct in that the semantics of the residual expression restricted to dynamic variables is equivalent to the semantics of the subject expression with the same static variables. Formally, for any arithmetic expression $\mathtt{E}$, any (so-called "static") environment ρ and any (so-called "dynamic") environment ρ' such that $\rho \Subset \rho'$, we have $\mathsf{A}[\![\mathsf{R}[\![\mathtt{E}]\!]\rho]\!]\rho' = \mathsf{A}[\![\mathtt{E}]\!]\rho'$. Moreover the evaluation of the residual only depends on the dynamic variables in that $\mathsf{A}[\![\mathsf{R}[\![\mathtt{E}]\!]\rho]\!]\rho' = \mathsf{A}[\![\mathsf{R}[\![\mathtt{E}]\!]\rho]\!](\rho' \setminus \mathsf{dom}[\rho])$.

In particular, for any static expression $\mathtt{E}$ in the environment ρ (such that $\mathsf{static}[\![\mathtt{E}]\!]\rho$), the residual is equal (up to the value/denotation correspondence) to the classical evaluation of Sec. 4.3: $\mathsf{R}[\![\mathtt{E}]\!]\rho = \mathsf{A}[\![\mathtt{E}]\!]\rho$. Similar results hold for boolean expressions $\mathtt{B}$.

5.3 Definition of Action Specialization

The specialization $\mathsf{R}[\![\mathtt{A}]\!]\rho$ of an action $\mathtt{A}$ in an environment ρ is defined in **Fig. 7**. Action specialization produces both a residual environment and a residual action, the residual environment possibly recording the value of static variables (upon initialization or e.g. after assignment of a constant) or no longer recording the value of dynamic variables (e.g. after a random assignment):

Action specialization is an abstraction in that (representing properties as usual by the set of elements having this property):

$$
\mathsf{R} \in \mathbb{A} \longmapsto \mathfrak{E} \longmapsto (\mathfrak{E} \times \mathbb{A})
$$

$$
\begin{array}{rcl}
\mathsf{R}[\![\mathtt{B}]\!]\rho & \triangleq & \langle \rho,\ \mathsf{R}[\![\mathtt{B}]\!]\rho \rangle \\
\mathsf{R}[\![\mathtt{X} := ?]\!]\rho & \triangleq & \langle \rho \setminus \mathtt{X},\ \mathtt{X} := ? \rangle \\
\mathsf{R}[\![\mathtt{X} := \mathtt{E}]\!]\rho & \triangleq & \text{if } \mathsf{static}[\![\mathtt{E}]\!]\rho \text{ then } \langle \rho[\mathtt{X} := \mathsf{R}[\![\mathtt{E}]\!]\rho],\ \mathtt{skip} \rangle \\
 & & \text{else } \langle \rho \setminus \mathtt{X},\ \mathtt{X} := \mathsf{R}[\![\mathtt{E}]\!]\rho \rangle \ .
\end{array}
$$

Figure 7: Action specialization

$$
\mathsf{po}\langle \wp(\mathfrak{E} \times \mathbb{A});\ \subseteq \rangle \underset{\alpha_{\mathrm{R}}}{\overset{\gamma_{\mathrm{R}}}{\leftrightarrows}} \mathsf{po}\langle \wp(\mathfrak{E} \times \mathbb{A});\ \subseteq \rangle
$$

where $\alpha_{\mathrm{R}}(X) \triangleq \{\mathsf{R}[\![\mathtt{A}]\!]\rho \mid \langle \rho,\ \mathtt{A} \rangle \in X\}$.

5.4 Correctness of Action Specialization

The specialization $\langle \rho_r,\ \mathtt{A}_r \rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0$ of an action $\mathtt{A}$ in a (so-called "static") environment ρ_0 is correct since the residual action $\mathtt{A}_r$ and subject action $\mathtt{A}$ have the same semantics in environments with identical static variables, ρ_r is the new static environment after execution of action $\mathtt{A}$ and the execution of the residual action $\mathtt{A}_r$ depends on dynamic variables only. Formally, for all actions $\mathtt{A}$, if $\langle \rho_r,\ \mathtt{A}_r \rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0$ and $\rho_0 \Subset \rho'$ then $\mathsf{S}[\![\mathtt{A}_r]\!]\rho' = \mathsf{S}[\![\mathtt{A}]\!]\rho'$, $\forall \rho'' \in \mathsf{S}[\![\mathtt{A}]\!]\rho' : \rho_r \Subset \rho''$ and $(\mathsf{S}[\![\mathtt{A}_r]\!]\rho') \setminus \mathsf{dom}[\rho_r] = (\mathsf{S}[\![\mathtt{A}_r]\!](\rho' \setminus \mathsf{dom}[\rho_0])) \setminus \mathsf{dom}[\rho_r]$.

6. EXAMPLE 1: CONSTANT PROPAGATION

6.1 Observational Abstraction

The observational abstraction for constant propagation gets rid of commands but preserves the sequence of environments observed along a partial trace:

$$
\alpha^{\mathrm{c}}_{\mathcal{O}}(\mathcal{T}) \triangleq \{\alpha^{\mathrm{c}}_{\mathcal{O}}(\sigma) \mid \sigma \in \mathcal{T}\},
$$

$$
\alpha^{\mathrm{c}}_{\mathcal{O}}(\sigma) \triangleq \boldsymbol{\lambda} i \cdot \alpha^{\mathrm{c}}_{\mathcal{O}}(\sigma_i), \qquad \alpha^{\mathrm{c}}_{\mathcal{O}}(\langle \rho,\ \mathtt{C} \rangle) \triangleq \rho \ .
$$

It is therefore insensible to the modification of the program actions, relabelling (contrary to $\equiv$) and dead code elimination (like $\equiv$). We have:

$$
\mathsf{po}\langle \wp(\mathfrak{D}^*);\ \subseteq \rangle \underset{\alpha^{\mathrm{c}}_{\mathcal{O}}}{\overset{\gamma^{\mathrm{c}}_{\mathcal{O}}}{\leftrightarrows}} \mathsf{po}\langle \wp(\mathfrak{R}^*);\ \subseteq \rangle
$$

where $\mathfrak{R}^*$ is the set of finite sequences of environments.

6.2 Constant Detection Analysis

Constant detection static analysis $\mathsf{S}^{\mathrm{c}}[\![\mathtt{P}]\!]$ of a program $\mathtt{P}$ [18] is a sound abstract interpretation $\alpha^{\mathrm{c}}(\mathsf{S}^*_{\iota}[\![\mathtt{P}]\!]) \ddot{\sqsubseteq} \mathsf{S}^{\mathrm{c}}[\![\mathtt{P}]\!]$ of the program partial trace semantics $\mathsf{S}^*_{\iota}[\![\mathtt{P}]\!]$ [6] for the following abstraction (which is the upper adjoint of the Galois connection (8)):

$$
\begin{array}{rcl}
\alpha^{\mathrm{c}}(\mathcal{T}) & \triangleq & \boldsymbol{\lambda}\mathtt{L} \cdot \boldsymbol{\lambda}\mathtt{X} \cdot \ddot{\bigsqcup} \{\rho(\mathtt{X}) \mid \exists \sigma \in \mathcal{T} : \exists \mathtt{C} \in \mathbb{C} : \exists i : \\
 & & \qquad\qquad \sigma_i = \langle \rho,\ \mathtt{C} \rangle \wedge \mathsf{lab}[\![\mathtt{C}]\!] = \mathtt{L}\}
\end{array}
$$

where $\ddot{\bigsqcup}$ is the pointwise extension of the least upper bound $\bigsqcup$ in the complete lattice $\mathfrak{D}^{\mathrm{c}} \triangleq \mathfrak{V}_{\mho} \cup \{\bot, \top\}$ partially ordered by $\forall x \in \mathfrak{D}^{\mathrm{c}} : \bot \sqsubseteq x \sqsubseteq x \sqsubseteq \top$.

6.3 Offline Semantic Constant Propagation

Let $\mathcal{T}^c = \mathsf{S}^c[\![\mathtt{P}]\!] \ddot{\sqsupseteq} \alpha^c(\mathsf{S}^*_\iota[\![\mathtt{P}]\!])$ be the result of a preliminary constant detection algorithm. The semantics transformer propagates constants along commands appearing within traces:

$$\mathsf{t}^c[\mathcal{T}, \mathcal{T}^c] \triangleq \{\mathsf{t}^c[\sigma, \mathcal{T}^c] \mid \sigma \in \mathcal{T}\}, \quad \mathsf{t}^c[\sigma, \mathcal{T}^c] \triangleq \boldsymbol{\lambda} i \cdot \mathsf{t}^c[\sigma_i, \mathcal{T}^c],$$
$$\mathsf{t}^c[\langle \rho,\ \mathtt{C}\rangle, \mathcal{T}^c] \triangleq \langle \rho,\ \mathsf{t}^c[\mathtt{C}, \mathcal{T}^c(\mathsf{lab}[\![\mathtt{C}]\!])]\rangle \ .$$

This relies on the following command specialization algorithm:

$$\mathsf{t}^c[\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2\,;, \rho^c] \triangleq \mathtt{L}_1 : \mathsf{t}^c[\mathtt{A}, \rho^c] \to \mathtt{L}_2\,;$$
$$\mathsf{t}^c[\mathtt{A}, \rho^c] \triangleq \text{let } \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!](\rho^c|_{\{\mathtt{X} \in \mathbb{X} \mid \rho^c(\mathtt{X}) \in \mathfrak{V}_\upsilon\}}) \text{ in } \mathtt{A}_r \ .$$

6.4 Semantic Correctness of the Semantic Constant Propagation Transformation

The first aspect of semantic correctness is to prove that the semantic transformation produces valid traces (belonging to $\mathfrak{D}^*$). The proof relies on the fact that the execution of subject and transformed actions are equal:

$$\mathsf{S}[\![\mathtt{A}]\!]\rho \quad = \quad \mathsf{S}[\![\mathsf{t}^c[\mathtt{A}, \rho^c]]\!]\rho \quad \text{whenever} \quad \rho \in \gamma^c(\rho^c) \ . \qquad (9)$$

The second aspect of semantic correctness is that the observational abstraction α^c of the subject and transformed partial trace semantics are identical. This is trivial since environments are left untouched in the transformation.

6.5 Performance Correctness of the Semantic Constant Propagation Transformation

For performance correctness, the length of maximal traces is left unchanged while the evaluation of the transformed actions takes fewer elementary steps. Formally, the weight of a finite trace is:

$$\begin{array}{rcl@{\qquad}rcl}
\varpi[\sigma] & \triangleq & \Sigma_{i=0}^{\#\sigma-1} \varpi[\![\sigma_i]\!] & \varpi[\![\mathtt{skip}]\!] & \triangleq & 1 \\
\varpi[\![\mathtt{L}_1 : \mathtt{A} \to \mathtt{L}_2\,;]\!] & \triangleq & \varpi[\![\mathtt{A}]\!] & \varpi[\![\mathtt{n}]\!] & \triangleq & 0 \\
\varpi[\![\mathtt{X} := \mathtt{E}]\!] & \triangleq & 1 + \varpi[\![\mathtt{E}]\!] & \varpi[\![\mathtt{X}]\!] & \triangleq & 1 \\
\varpi[\![\mathtt{E}_1 - \mathtt{E}_2]\!] & \triangleq & 1 + \varpi[\![\mathtt{E}_1]\!] + \varpi[\![\mathtt{E}_2]\!] & \cdots & \cdots & \cdots
\end{array}$$

The weight of a set of traces is a mapping of trace observations to the maximal weight of the concrete traces with such observation:

$$\varpi[\mathcal{T}] \quad \triangleq \quad \boldsymbol{\lambda} \varsigma \in \alpha^c_{\mathcal{O}}(\mathcal{T}) \cdot \max\{\varpi[\sigma] \mid \alpha^c_{\mathcal{O}}(\sigma) = \varsigma\} \ .$$

This is an abstraction:

$$\mathsf{po}\langle \mathfrak{D}^*;\ \subseteq\rangle \quad \underset{\varpi}{\overset{\gamma_\varpi}{\leftrightarrows}} \quad \mathsf{po}\langle \mathfrak{R}^* \longmapsto \mathbb{N};\ \dot{\leq}\rangle$$

where $\dot{\leq}$ is the pointwise extension of $\leq$. The performance correctness of semantic constant propagation follows from $\varpi[\mathsf{t}^c[\mathcal{T}, \mathcal{T}^c]] \dot{\leq} \varpi[\mathcal{T}]$.

6.6 Offline Syntactic Constant Propagation

The constant propagation algorithm $\mathbb{t}^c[\![\mathtt{P}, \mathcal{T}^c]\!]$ is finally derived from the partial trace semantics $\mathsf{lfp}^{\subseteq} \mathsf{F}^*_\iota[\![\mathtt{P}]\!]$ by the abstraction $\boldsymbol{\lambda} \mathcal{T} \cdot \mathbb{p}^* \circ \mathsf{t}^c[\mathcal{T}, \mathcal{T}^c]$, (where $\mathcal{T}^c = \mathsf{S}^c[\![\mathtt{P}]\!]$ is the constant detection algorithm) using the fixpoint approximation theorem (1): $\mathbb{p}^* \circ \mathsf{t}^c[\mathsf{lfp}^{\subseteq} \mathsf{F}^*_\iota[\![\mathtt{P}]\!], \mathcal{T}^c] \mathbin{\mathbb{\sqsubseteq}} \mathsf{lfp}^{\mathbb{\sqsubseteq}} \mathsf{F}^c[\![\mathtt{P}]\!]$ where

$$\begin{array}{rcl}
\mathsf{F}^c[\![\mathtt{P}]\!]X & \triangleq & \{\mathsf{t}^c[\mathtt{C}, \mathcal{T}^c(\mathsf{lab}[\![\mathtt{C}]\!])] \mid \mathtt{C} \in \mathtt{P} \wedge \mathsf{lab}[\![\mathtt{C}]\!] \in \mathfrak{L}[\![\mathtt{P}]\!]\} \\
& & \cup\ \{\mathsf{t}^c[\mathtt{C}', \mathcal{T}^c(\mathsf{lab}[\![\mathtt{C}']\!])] \mid \exists \mathtt{C} \in X : \mathsf{act}[\![\mathtt{C}]\!] \neq \mathtt{false} \\
& & \qquad \wedge\ \mathsf{suc}[\![\mathtt{C}]\!] = \mathsf{lab}[\![\mathtt{C}']\!] \wedge \mathtt{C}' \in \mathtt{P}\} \ .
\end{array}$$

As is classical in abstract interpretation [6], $\mathsf{F}^c[\![\mathtt{P}]\!]$ is formally derived from $\mathsf{F}^*_\iota[\![\mathtt{P}]\!]$ [4] by the commutation condition $\mathbb{p}^* \circ \mathsf{t}^c[\mathsf{F}^*_\iota[\![\mathtt{P}]\!]\mathcal{T}, \mathcal{T}^c] \mathbin{\mathbb{\sqsubseteq}} \mathsf{F}^c[\![\mathtt{P}]\!]\mathbb{p}^* \circ \mathsf{t}^c[\mathcal{T}, \mathcal{T}^c]$. In practice, $\mathbb{\sqsubseteq}$ is not computable so we compute $\mathsf{lfp}^{\subseteq} \mathsf{F}^c[\![\mathtt{P}]\!]$ such that $\mathsf{lfp}^{\mathbb{\sqsubseteq}} \mathsf{F}^c[\![\mathtt{P}]\!] \mathbin{\mathbb{\sqsubseteq}} \mathsf{lfp}^{\subseteq} \mathsf{F}^c[\![\mathtt{P}]\!]$ whence $\mathbb{p}^* \circ \mathsf{t}^c[\mathsf{lfp}^{\subseteq} \mathsf{F}^*_\iota[\![\mathtt{P}]\!], \mathcal{T}^c] \mathbin{\mathbb{\sqsubseteq}} \mathsf{lfp}^{\subseteq} \mathsf{F}^c[\![\mathtt{P}]\!]$. Since the subject program is finite $\mathsf{lfp}^{\subseteq} \mathsf{F}^c[\![\mathtt{P}]\!]$ immediately leads to an iterative constant propagation algorithm (where dead code is also partially eliminated).

6.7 Correctness of Offline Syntactic Constant Propagation

Finally, (9) implies that the semantics of program actions is unchanged by constant propagation which implies $\alpha^c_{\mathcal{O}}(\mathsf{S}^*_\iota[\![\mathsf{t}^c[\![\mathtt{P}, \mathcal{T}^c]\!]]\!]) = \alpha^c_{\mathcal{O}}(\mathsf{t}^c[\mathsf{S}^*_\iota[\![\mathtt{P}]\!], \mathcal{T}^c])$ whence the correctness of syntactic constant propagation.

7. EXAMPLE 2: PARTIAL EVALUATION

It is now shown that online partial evaluation and binding time analysis based offline partial evaluation [1, 13, 14, 15] can be captured in the program transformation framework introduced in Sec. 3. This is applied to the imperative language of Sec. 4 which is similar to the one considered in [16, Ch. 4]. It is shown that the widening operation [4, 7] can be used in practice to enforce termination of the transformation expressed in fixpoint form which leads to a terminating iterative algorithm. Moreover widenings offer a continuum between online and offline partial evaluation as required in mixline partial evaluation [16, p. 153].

7.1 Online Partial Evaluation

Applying the framework of Sec. 3.9, we understand partial evaluation of a subject program (a syntactic transformation) as an abstract interpretation of a partial evaluation of its semantics (a semantic transformation which is itself an abstraction of the subject semantics). Following [16, p. 78], "the idea of program point specialization is to incorporate the values of the static variables into the control point" so the set $\mathbb{L}$ of program labels is assumed to be of the form:

$$\mathbb{L} \quad \triangleq \quad \mathbb{N} \times \mathfrak{E}$$

where $\mathbb{N}$ is the set of naturals and $\mathfrak{E}$ is the set of environments. We assume that all labels in the subject program belong to $\mathbb{N} \times \{\mathfrak{d}\}$ (for short to $\mathbb{N}$ up to an isomorphism).

7.1.1 *Semantic Online Partial Evaluation*

We define the environment and the label of the command of the first state in the non-empty trace σ respectively as $\mathsf{env}[\sigma] \triangleq \mathsf{env}[\sigma_0]$ where $\mathsf{env}[\langle \rho,\ \mathtt{C}\rangle] \triangleq \rho$ and $\mathsf{lab}[\sigma] \triangleq \mathsf{lab}[\sigma_0]$ where $\mathsf{lab}[\langle \rho,\ \mathtt{C}\rangle] \triangleq \mathsf{lab}[\![\mathtt{C}]\!]$. Similarly $\mathsf{suc}[\langle \rho,\ \mathtt{C}\rangle] \triangleq \mathsf{suc}[\![\mathtt{C}]\!]$ and $\mathsf{act}[\langle \rho,\ \mathtt{C}\rangle] \triangleq \mathsf{act}[\![\mathtt{C}]\!]$.

The partial evaluation of a set of non-empty traces $\mathcal{T}$ is the partial evaluation of the traces σ in that set starting at a given program label $\mathtt{L}_0$ for a given static environment ρ_0:

$$\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle \mathtt{L}_0,\ \rho_0\rangle \quad \triangleq \quad \{\mathsf{PE}_{\mathrm{on}}[\sigma]\langle \mathtt{L}_0,\ \rho_0\rangle \mid \sigma \in \mathcal{T} \wedge \mathsf{lab}[\sigma] = \mathtt{L}_0 \wedge \rho_0 \Subset \mathsf{env}[\sigma]\}$$

Observe that the semantic online partial evaluation is a functional abstraction:

[4] For short, this computation is not shown here. A similar one is detailed in the following Sec. 7.1.4.

$$\mathsf{po}\langle\wp(\mathfrak{D}^*);\subseteq\rangle \underset{\alpha^{\mathrm{PE}}_{\mathrm{on}}}{\overset{\gamma^{\mathrm{PE}}_{\mathrm{on}}}{\leftrightarrows}} \mathsf{po}\langle(\mathbb{L}\times\mathfrak{E})\longmapsto\wp(\mathfrak{D}^*);\dot{\subseteq}\rangle \quad (10)$$

The partial evaluation of a non-empty trace σ starting with an environment ρ specifies the residual/specialized computation when knowing the given static values $\rho_0 \Subset \rho$ (we write $\langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle \triangleq$ *if* $\mathtt{L}_1 = \mathtt{l}$ *then* $\mathtt{l}$ *else* $\langle\mathtt{L}_1,\ \rho_r\rangle$):

$$\begin{array}{l}\mathsf{PE}_{\mathrm{on}}[\langle\rho,\ \mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\rangle\sigma]\langle\mathtt{L}_0,\ \rho_0\rangle \triangleq \\ \quad \text{let } \langle\rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0 \text{ and } \mathtt{L}_0' = \langle\mathtt{L}_0,\ \rho_0\rangle \text{ in} \\ \quad\quad \text{let } \mathtt{L}_1' = \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle \text{ in} \\ \quad\quad\quad \langle\rho\setminus\mathsf{dom}[\rho_0],\ \mathtt{L}_0' : \mathtt{A}_r\to\mathtt{L}_1';\rangle\mathsf{PE}_{\mathrm{on}}[\sigma]\langle\mathtt{L}_1,\ \rho_r\rangle\ .\end{array} \quad (11)$$

We define $\mathsf{PE}_{\mathrm{on}}[\sigma]\langle\mathtt{l},\ \rho_0\rangle \triangleq \vec{\epsilon}$ to handle `stop` commands and $\mathsf{PE}_{\mathrm{on}}[\vec{\epsilon}]\langle\mathtt{L}_0,\ \rho_0\rangle \triangleq \vec{\epsilon}$ to cover the case of the empty trace.

7.1.2 *Observational Abstraction*

The observational abstraction of a set $\mathcal{T}\subseteq\mathfrak{D}^*$ of traces gets rid of those traces in $\mathcal{T}$ not starting at the given program label $\mathtt{L}_0$ with the given static environment ρ_0:

$$\begin{array}{rcl}\alpha^{\mathrm{PE}}_{\mathcal{O}}[\mathcal{T}]\langle\mathtt{L}_0,\ \rho_0\rangle &\triangleq& \{\alpha^{\mathrm{PE}}_{\mathcal{O}}[\sigma](\rho_0) \mid \sigma\in\mathcal{T}\wedge(\sigma\neq\vec{\epsilon})\Rightarrow \\ && \qquad (\mathsf{lab}[\sigma]=\mathtt{L}_0\wedge\rho_0\Subset\mathsf{env}[\sigma])\}\end{array}$$

The observation $\alpha^{\mathrm{PE}}_{\mathcal{O}}[\sigma](\rho_0)$ of a trace σ records only the value of dynamic variables (not in $\mathsf{dom}[\rho_0]$):

$$\begin{array}{rcl}\alpha^{\mathrm{PE}}_{\mathcal{O}}[\vec{\epsilon}](\rho_0) &\triangleq& \vec{\epsilon},\\ \alpha^{\mathrm{PE}}_{\mathcal{O}}[\langle\rho,\ \mathtt{C}\rangle\sigma](\rho_0) &\triangleq& \text{let } \langle\rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathsf{act}[\![\mathtt{C}]\!]]\!]\rho_0 \text{ in} \\ && \qquad (\rho\setminus\mathsf{dom}[\rho_0])\cdot\alpha^{\mathrm{PE}}_{\mathcal{O}}[\sigma](\rho_r)\ .\end{array}$$

By defining the concretization:

$$\begin{array}{rcl}\gamma^{\mathrm{PE}}_{\mathcal{O}}[\mathcal{R}]\langle\mathtt{L}_0,\ \rho_0\rangle &\triangleq& \{\sigma\in\mathfrak{D}^* \mid (\sigma=\vec{\epsilon}\vee(\mathsf{lab}[\sigma]=\mathtt{L}_0\wedge \\ && \qquad \rho_0\Subset\mathsf{env}[\sigma]))\Rightarrow(\alpha^{\mathrm{PE}}_{\mathcal{O}}[\sigma](\rho_0)\in\mathcal{R})\},\end{array}$$

we have the following observational abstraction ($\mathfrak{R}^*$ is the set of finite sequences of environments):

$$\mathsf{po}\langle\wp(\mathfrak{D}^*);\subseteq\rangle \underset{\alpha^{\mathrm{PE}}_{\mathcal{O}}}{\overset{\gamma^{\mathrm{PE}}_{\mathcal{O}}}{\leftrightarrows}} \mathsf{po}\langle(\mathbb{L}\times\mathfrak{E})\longmapsto\wp(\mathfrak{R}^*);\dot{\subseteq}\rangle\ .$$

7.1.3 *Semantic and Performance Correctness of the Semantic Transformation*

The semantic correctness of the semantic partial evaluation follows from the fact that the subject and specialized semantics have the same observed environments up to static variables:

$$\alpha^{\mathrm{PE}}_{\mathcal{O}}[\mathcal{T}]\langle\mathtt{L}_0,\ \rho_0\rangle = \alpha^{\mathrm{PE}}_{\mathcal{O}}[\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle\mathtt{L}_0,\ \rho_0\rangle]\langle\mathtt{L}_0,\ \rho_0\rangle$$

The performance correctness of partial evaluation can be expressed by the performance abstraction introduced in Sec. 6.5 in that $\varpi[\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle\mathtt{L}_0,\ \rho_0\rangle] \dot{\leq} \varpi[\mathcal{T}]$.

7.1.4 *Fixpoint Online Partial Evaluation Semantics*

We now compute the abstraction by the online partial evaluation abstraction $\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle\mathtt{L}_0,\ \rho_0\rangle$ defined in Sec. 7.1.1 of the partial trace semantics $\mathsf{S}^*[\![\mathtt{P}]\!]$ expressed in the fixpoint form $\mathsf{lfp}^{\subseteq}\mathsf{F}^*[\![\mathtt{P}]\!]$ of Sec. 4.5. We first establish the local commutation property necessary for fixpoint transfer (1).

$$\begin{array}{ll} & \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathsf{F}^*[\![\mathtt{P}]\!]\mathcal{T}]\langle\mathtt{L}_0,\ \rho_0\rangle \\ = & \quad \wr\text{def. } \mathsf{F}^*[\![\mathtt{P}]\!]\wr \\ & \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathfrak{S}[\![\mathtt{P}]\!]\cup\{ss'\sigma \mid s'\in\mathsf{S}[\![\mathtt{P}]\!]s\wedge s'\sigma\in\mathcal{T}\}]\langle\mathtt{L}_0,\ \rho_0\rangle \\ = & \quad \wr\text{By (10) in Sec. 7.1.1 so that } \alpha^{\mathrm{PE}}_{\mathrm{on}} \text{ is a complete} \\ & \quad\ \dot{\cup}\text{-join morphism}\wr \\ & \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathfrak{S}[\![\mathtt{P}]\!]]\langle\mathtt{L}_0,\ \rho_0\rangle\cup \\ & \quad \alpha^{\mathrm{PE}}_{\mathrm{on}}[\{ss'\sigma \mid s'\in\mathsf{S}[\![\mathtt{P}]\!]s\wedge s'\sigma\in\mathcal{T}\}]\langle\mathtt{L}_0,\ \rho_0\rangle\end{array}$$

We consider the two terms separately. For the first term, we have:

$$\begin{array}{ll} & \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathfrak{S}[\![\mathtt{P}]\!]]\langle\mathtt{L}_0,\ \rho_0\rangle \\ = & \quad \wr\text{def. } \alpha^{\mathrm{PE}}_{\mathrm{on}}\langle\mathtt{L}_0,\ \rho_0\rangle \text{ and } \vec{\epsilon}\notin\mathfrak{S}[\![\mathtt{P}]\!]\wr \\ & \{\mathsf{PE}_{\mathrm{on}}[\sigma]\langle\mathtt{L}_0,\ \rho_0\rangle \mid \sigma\in\mathfrak{S}[\![\mathtt{P}]\!]\wedge\mathsf{lab}[\sigma]=\mathtt{L}_0\wedge\rho_0\Subset \\ & \mathsf{env}[\sigma]\} \\ = & \quad \wr\text{def. } \mathfrak{S}[\![\mathtt{P}]\!]\wr \\ & \{\mathsf{PE}_{\mathrm{on}}[\langle\rho,\ \mathtt{L}_0 :\ \mathtt{A}\ \to\ \mathtt{L}_1;\rangle]\langle\mathtt{L}_0,\ \rho_0\rangle \mid \rho\ \in\ \mathfrak{E}[\![\mathtt{P}]\!]\wedge \\ & \mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\in\mathtt{P}\wedge\rho_0\Subset\rho\} \\ = & \quad \wr\text{def. (11) of } \mathsf{PE}_{\mathrm{on}}[s]\langle\mathtt{L}_0,\ \rho_0\rangle\wr \\ & \{\langle\rho\backslash\mathsf{dom}[\rho_0],\langle\mathtt{L}_0,\rho_0\rangle : \mathtt{A}_r\to\langle\!|\mathtt{L}_1,\rho_r|\!\rangle;\rangle \mid \rho\in\mathfrak{E}[\![\mathtt{P}]\!]\wedge \\ & \rho_0\Subset\rho\wedge\mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\in\mathtt{P}\wedge\langle\rho_r,\ \mathtt{A}_r\rangle=\mathsf{R}[\![\mathtt{A}]\!]\rho_0\}\end{array}$$

For the second term, we have:

$$\begin{array}{ll} & \alpha^{\mathrm{PE}}_{\mathrm{on}}[\{ss'\sigma \mid s'\in\mathsf{S}[\![\mathtt{P}]\!]s\wedge s'\sigma\in\mathcal{T}\}]\langle\mathtt{L}_0,\ \rho_0\rangle \\ = & \quad \wr\text{def. } \alpha^{\mathrm{PE}}_{\mathrm{on}}\langle\mathtt{L}_0,\ \rho_0\rangle\wr \\ & \{\mathsf{PE}_{\mathrm{on}}[ss'\sigma]\langle\mathtt{L}_0,\ \rho_0\rangle \mid s'\ \in\ \mathsf{S}[\![\mathtt{P}]\!]s\wedge s'\sigma\ \in\ \mathcal{T}\wedge \\ & \mathsf{lab}[ss'\sigma]=\mathtt{L}_0\wedge\rho_0\Subset\mathsf{env}[ss'\sigma]\} \\ = & \quad \wr\text{def. (11) of } \mathsf{PE}_{\mathrm{on}}[\langle\rho,\ \mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\rangle\sigma]\langle\mathtt{L}_0,\ \rho_0\rangle\wr \\ & \{\langle\rho\ \setminus\ \mathsf{dom}[\rho_0],\ \langle\mathtt{L}_0,\ \rho_0\rangle :\ \ \mathtt{A}_r\ \ \to\ \ \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle;\rangle \\ & \mathsf{PE}_{\mathrm{on}}[s'\sigma]\langle\mathtt{L}_1,\rho_r\rangle \mid s'\in\mathsf{S}[\![\mathtt{P}]\!]\langle\rho,\ \mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\rangle\wedge\langle\rho_r, \\ & \mathtt{A}_r\rangle=\mathsf{R}[\![\mathtt{A}]\!]\rho_0\wedge s'\sigma\in\mathcal{T}\wedge\rho_0\Subset\rho\} \\ = & \quad \wr\text{letting } s'=\langle\rho',\mathtt{C}'\rangle, \text{def. } \mathsf{S}[\![\mathtt{P}]\!]\langle\rho,\ \mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\rangle \\ & \quad\ \text{in Sec. 4.4 and def. } \mathsf{S}[\![\mathtt{A}]\!]\rho \text{ so that } \rho'\in\mathfrak{E}[\![\mathtt{P}]\!] \text{ iff} \\ & \quad\ \rho\in\mathfrak{E}[\![\mathtt{P}]\!]\wr \\ & \{\langle\rho\ \setminus\ \mathsf{dom}[\rho_0],\ \langle\mathtt{L}_0,\ \rho_0\rangle :\ \ \mathtt{A}_r\ \ \to\ \ \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle;\rangle \\ & \mathsf{PE}_{\mathrm{on}}[\langle\rho',\ \mathtt{C}'\rangle\sigma]\langle\mathtt{L}_1,\ \rho_r\rangle \mid \rho\ \in\ \mathfrak{E}[\![\mathtt{P}]\!]\wedge\rho_0\ \Subset\ \rho\wedge \\ & \mathtt{L}_0 :\ \mathtt{A}\ \to\ \mathtt{L}_1;\ \in\ \mathtt{P}\wedge\langle\rho_r,\ \mathtt{A}_r\rangle\ =\ \mathsf{R}[\![\mathtt{A}]\!]\rho_0\wedge\rho'\ \in \\ & \mathsf{S}[\![\mathtt{A}]\!]\rho\wedge\mathtt{L}_1=\mathsf{lab}[\![\mathtt{C}']\!]\wedge\langle\rho',\ \mathtt{C}'\rangle\sigma\in\mathcal{T}\} \\ = & \quad \wr\langle\rho_r,\ \mathtt{A}_r\rangle=\mathsf{R}[\![\mathtt{A}]\!]\rho_0 \text{ implies } \rho'\in\mathsf{S}[\![\mathtt{A}]\!]\rho \text{ iff } (\rho'\setminus \\ & \quad\ \mathsf{dom}[\rho_r])\in\mathsf{S}[\![\mathtt{A}_r]\!](\rho\setminus\mathsf{dom}[\rho_0]) \text{ and } \rho_r\Subset\rho' \text{ as} \\ & \quad\ \text{observed in Sec. 5.4}\wr \\ & \{\langle\rho\ \setminus\ \mathsf{dom}[\rho_0],\ \langle\mathtt{L}_0,\ \rho_0\rangle :\ \ \mathtt{A}_r\ \ \to\ \ \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle;\rangle \\ & \mathsf{PE}_{\mathrm{on}}[\langle\rho',\ \mathtt{C}'\rangle\sigma]\langle\mathtt{L}_1,\ \rho_r\rangle \mid \rho\ \in\ \mathfrak{E}[\![\mathtt{P}]\!]\wedge\rho_0\ \Subset\ \rho\wedge \\ & \mathtt{L}_0 :\ \mathtt{A}\ \to\ \mathtt{L}_1;\ \in\ \mathtt{P}\wedge\langle\rho_r,\ \mathtt{A}_r\rangle\ =\ \mathsf{R}[\![\mathtt{A}]\!]\rho_0\wedge\mathsf{env}[\langle\rho', \\ & \mathtt{C}'\rangle\sigma]\setminus\mathsf{dom}[\rho_r]\in\mathsf{S}[\![\mathtt{A}_r]\!]\rho\setminus\mathsf{dom}[\rho_0]\wedge\mathsf{lab}[\langle\rho',\ \mathtt{C}'\rangle\sigma]= \\ & \mathtt{L}_1\wedge\langle\rho',\ \mathtt{C}'\rangle\sigma\in\mathcal{T}\wedge\rho_r\Subset\mathsf{env}[\langle\rho',\ \mathtt{C}'\rangle\sigma]\} \\ = & \quad \wr\text{letting } \sigma'\ =\ \langle\rho',\ \mathtt{C}'\rangle\sigma \text{ so that } (\mathsf{env}[\sigma']\ \setminus \\ & \quad\ \mathsf{dom}[\rho_r])=\mathsf{env}[\mathsf{PE}_{\mathrm{on}}[\sigma']\langle\mathtt{L}_1,\ \rho_r\rangle]\wr \\ & \{\langle\rho\backslash\mathsf{dom}[\rho_0],\langle\mathtt{L}_0,\rho_0\rangle : \mathtt{A}_r\to\langle\!|\mathtt{L}_1,\rho_r|\!\rangle;\rangle\mathsf{PE}_{\mathrm{on}}[\sigma']\langle\mathtt{L}_1, \\ & \rho_r\rangle \mid \rho\in\mathfrak{E}[\![\mathtt{P}]\!]\wedge\rho_0\Subset\rho\wedge\mathtt{L}_0 : \mathtt{A}\to\mathtt{L}_1;\in\mathtt{P}\wedge\langle\rho_r, \\ & \mathtt{A}_r\rangle\ =\ \mathsf{R}[\![\mathtt{A}]\!]\rho_0\wedge\mathsf{env}[\mathsf{PE}_{\mathrm{on}}[\sigma']\langle\mathtt{L}_1,\ \rho_r\rangle]\ \in\ \mathsf{S}[\![\mathtt{A}_r]\!](\rho\ \setminus \\ & \mathsf{dom}[\rho_0])\wedge\mathsf{lab}[\sigma']=\mathtt{L}_1\wedge\sigma'\in\mathcal{T}\wedge\rho_r\Subset\mathsf{env}[\sigma']\} \\ = & \quad \wr\sigma=\mathsf{PE}_{\mathrm{on}}[\sigma']\langle\mathtt{L}_1,\ \rho_r\rangle \text{ and def. } \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle\mathtt{L}_1,\ \rho_r\rangle \\ & \quad\ \text{in Sec. 7.1.1}\wr\end{array}$$

$\{\langle \rho \setminus \mathbf{dom}[\rho_0],\ \langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\rangle\sigma \mid \rho \in \mathfrak{E}[\![\mathtt{P}]\!] \wedge \rho_0 \Subset \rho \wedge \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0 \wedge \mathsf{env}[\sigma] \in \mathsf{S}[\![\mathtt{A}_r]\!](\rho \setminus \mathbf{dom}[\rho_0]) \wedge \sigma \in \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle \mathtt{L}_1,\ \rho_r\rangle\}$

Grouping the two terms together, we have:

$$\begin{array}{ll} & \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathsf{F}^*[\![\mathtt{P}]\!]\mathcal{T}]\langle \mathtt{L}_0,\ \rho_0\rangle \\ = & \{\langle \rho \setminus \mathbf{dom}[\rho_0],\ \langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\rangle\sigma \mid \rho \in \mathfrak{E}[\![\mathtt{P}]\!] \wedge \rho_0 \Subset \rho \wedge \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0 \wedge (\sigma = \vec{\epsilon} \vee \mathsf{env}[\sigma] \in \mathsf{S}[\![\mathtt{A}_r]\!](\rho \setminus \mathbf{dom}[\rho_0]) \wedge \sigma \in \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]\langle \mathtt{L}_1,\ \rho_r\rangle)\} \\ = & \mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]]\langle \mathtt{L}_0,\ \rho_0\rangle \end{array}$$

by defining (again, to avoid a particular case for `stop` commands we assume, for short, that $\mathcal{T}_\iota\langle \mathtt{l},\ \rho_r\rangle = \vec{\epsilon}$):

$$\begin{array}{l} \mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!] \quad \in \quad ((\mathbb{L} \times \mathfrak{E}) \longmapsto \wp(\mathfrak{D}^*)) \longmapsto ((\mathbb{L} \times \mathfrak{E}) \longmapsto \wp(\mathfrak{D}^*)) \\ \mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\mathcal{T}_\iota]\langle \mathtt{L}_0,\ \rho_0\rangle \quad \triangleq \quad \{\langle \rho \setminus \mathbf{dom}[\rho_0], \\ \quad \langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\rangle\sigma \mid \rho \in \mathfrak{E}[\![\mathtt{P}]\!] \wedge \\ \quad \rho_0 \Subset \rho \wedge \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0 \wedge (\sigma = \vec{\epsilon} \vee \\ \quad \mathsf{env}[\sigma] \in \mathsf{S}[\![\mathtt{A}_r]\!](\rho \setminus \mathbf{dom}[\rho_0]) \wedge \sigma \in \mathcal{T}_\iota\langle \mathtt{L}_1,\ \rho_r\rangle)\} \end{array}$$

By (10), the above local commutation property $\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathsf{F}^*[\![\mathtt{P}]\!]\mathcal{T}] = \mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathcal{T}]]$ and fixpoint transfer (1), we conclude that $\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathsf{S}^*[\![\mathtt{P}]\!]] = \alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathsf{lfp}^{\subseteq}\mathsf{F}^*[\![\mathtt{P}]\!]] = \mathsf{lfp}^{\subseteq}\mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!]$.

7.1.5 *Syntactic Source-to-Source Online Partial Evaluation*

We now apply the semantics-to-syntax abstraction of Sec. 4.6 extended to

$$\mathsf{po}\langle(\mathbb{L}\times\mathfrak{E})\longmapsto\wp(\mathfrak{D}^*);\ \dot{\subseteq}\rangle \underset{\dot{\mathbb{p}}^*}{\overset{\dot{\mathsf{S}}^*}{\leftleftarrows}} \mathsf{po}\langle(\mathbb{L}\times\mathfrak{E})\longmapsto\mathbb{P};\ \sqsubseteq\rangle \quad (12)$$

where $\dot{\mathbb{p}}^*[\mathcal{T}_\iota]\langle \mathtt{L}_0,\ \rho_0\rangle \triangleq \mathbb{p}^*[\mathcal{T}_\iota\langle \mathtt{L}_0,\ \rho_0\rangle]$ and $\mathbb{p}^*$ collects commands along a set of partial traces as defined in Sec. 4.6. The local semi-commutation property is computed for $\subseteq$ as an approximation to $\sqsubseteq$ (and similarly for fixpoints as in Sec. 6.6):

$$\begin{array}{ll} & \dot{\mathbb{p}}^*[\mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\mathcal{T}_\iota]]\langle \mathtt{L}_0,\ \rho_0\rangle \\ = & \quad \wr\text{def. } \dot{\mathbb{p}}^*\wr \\ & \mathbb{p}^*[\mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\mathcal{T}_\iota]\langle \mathtt{L}_0,\ \rho_0\rangle] \\ = & \quad \wr\text{def. } \mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!]\wr \\ & \mathbb{p}^*[\{\langle \rho \setminus \mathbf{dom}[\rho_0], \langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\rangle\sigma \mid \rho \in \mathfrak{E}[\![\mathtt{P}]\!] \wedge \rho_0 \Subset \rho \wedge \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0 \wedge (\sigma = \vec{\epsilon} \vee \mathsf{env}[\sigma] \in \mathsf{S}[\![\mathtt{A}_r]\!](\rho \setminus \mathbf{dom}[\rho_0]) \wedge \sigma \in \mathcal{T}_\iota\langle \mathtt{L}_1,\ \rho_r\rangle)\}] \\ \subseteq & \quad \wr\text{ignoring the values of dynamic variables in } \rho \setminus \mathbf{dom}[\rho_0]\wr \\ & \mathbb{p}^*[\{\langle \rho \setminus \mathbf{dom}[\rho_0], \langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\rangle\sigma \mid \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0 \wedge \sigma \in \{\vec{\epsilon}\} \cup \mathcal{T}_\iota\langle \mathtt{L}_1,\ \rho_r\rangle\}] \\ = & \quad \wr\text{def. } \mathbb{p}^* \text{ in Sec. 4.6}\wr \\ & \bigcup\{\{\langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\} \cup \mathbb{p}^*[\mathcal{T}_\iota\langle \mathtt{L}_1,\ \rho_r\rangle] \mid \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0\} \\ = & \quad \wr\text{def. } \dot{\mathbb{p}}^*\wr \\ & \bigcup\{\{\langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\} \cup \dot{\mathbb{p}}^*[\mathcal{T}_\iota]\langle \mathtt{L}_1,\ \rho_r\rangle \mid \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0\} \\ = & \quad \mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\dot{\mathbb{p}}^*[\mathcal{T}_\iota]]\langle \mathtt{L}_0,\ \rho_0\rangle \end{array}$$

by defining (again, to avoid a particular case for `stop` commands we assume, for short, that $\mathcal{T}_\iota\langle \mathtt{l},\ \rho_r\rangle = \emptyset$):

$$\begin{array}{l} \mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!] \quad \in \quad ((\mathbb{L} \times \mathfrak{E}) \longmapsto \mathbb{P}) \longmapsto ((\mathbb{L} \times \mathfrak{E}) \longmapsto \mathbb{P}) \\ \mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\mathcal{T}_\iota]\langle \mathtt{L}_0,\ \rho_0\rangle \quad \triangleq \quad \bigcup\{\{\langle \mathtt{L}_0,\ \rho_0\rangle : \mathtt{A}_r \to \langle\!|\mathtt{L}_1,\ \rho_r|\!\rangle\mathtt{;}\} \\ \quad \cup\, \mathcal{T}_\iota\langle \mathtt{L}_1,\ \rho_r\rangle \mid \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1\mathtt{;} \in \mathtt{P} \wedge \langle \rho_r,\ \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho_0\}\ . \end{array}$$

By (12), the above semi-commutation property $\dot{\mathbb{p}}^*[\mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\mathcal{T}_\iota]] \dot{\subseteq} \mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\dot{\mathbb{p}}^*[\mathcal{T}_\iota]]$ and fixpoint approximation (1), we conclude that $\dot{\mathbb{p}}^*[\alpha^{\mathrm{PE}}_{\mathrm{on}}[\mathsf{S}^*[\![\mathtt{P}]\!]]] = \dot{\mathbb{p}}^*[\mathsf{lfp}^{\subseteq}\mathsf{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!]] \dot{\subseteq} \mathsf{lfp}^{\subseteq}\mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!]$. We can have a strict approximation since e.g. dead code whose uselessness can be established only thanks to the dynamic variable cannot be discovered using the values of the static variables and the program source only. Consequently the correctness of this approximation must be established by proving:

$$\alpha^{\mathrm{PE}}_{\mathcal{O}}[\mathsf{S}^*[\![\mathtt{P}]\!]] \quad = \quad \alpha^{\mathrm{PE}}_{\mathcal{O}}[\mathsf{lfp}^{\subseteq}\mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!]]\ .$$

7.1.6 *Online Partial Evaluation Semi-Algorithm*

The definition of the source-to-source partial evaluation in Sec. 7.1.5 is functional in that $\mathsf{lfp}^{\subseteq}\mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!] \in (\mathbb{L}\times\mathfrak{E})\longmapsto\mathbb{P}$. In practice, partial evaluation algorithms consider a given specific value $\langle \mathtt{L}_0, \rho_0\rangle$ of the initial argument only. The semi-algorithm to compute $(\mathsf{lfp}^{\subseteq}\mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!])\langle \mathtt{L}_0,\ \rho_0\rangle$ follows from the first-order chaotic iterations of [5] popularized as minimal function graph [17]. We obtain the semi-algorithm of [16, Fig. 4.6, p. 80] using a working list `WL`:

```
specialization(P, L0, ρ0) =
  Q := ∅; WL := {⟨L0, ρ0⟩};
  while WL contains an unmarked ⟨L, ρ⟩ do
    mark ⟨L, ρ⟩;
    forall L : A → L1; ∈ P do
      ⟨ρr, Ar⟩ := R[[A]]ρ;
      if L1 ≠ l then
        WL := WL ∪ {⟨L1, ρr⟩};
        Q := Q ∪ {⟨L, ρ⟩ : Ar → ⟨L1, ρr⟩;}
      else
        Q := Q ∪ {⟨L, ρ⟩ : Ar → l;}
  end end end;
  return Q.
```

Figure 8: A simple online specialization algorithm

7.2 Online Partial Evaluation with Widening

specialization is a semi-algorithm since it operates on an infinite complete lattice. As suggested in [5, 7], we can enforce convergence using widenings ∇ and compute:

$$\begin{array}{l} \mathsf{lfp}^{\subseteq}\boldsymbol{\lambda}\,\mathcal{T}_\iota \cdot \boldsymbol{\lambda}\langle \mathtt{L},\ \rho\rangle \cdot \mathcal{T}_\iota\langle \mathtt{L},\ \rho\rangle\ \nabla_1 \\ \qquad \mathbb{F}^{\mathrm{PE}}_{\mathrm{on}}[\![\mathtt{P}]\!][\boldsymbol{\lambda}\,\langle \mathtt{L}',\ \rho'\rangle \cdot \mathcal{T}_\iota(\langle \mathtt{L},\ \rho\rangle\ \nabla_2\ \langle \mathtt{L}',\ \rho'\rangle)]\langle \mathtt{L},\ \rho\rangle. \end{array}$$

The first widening ∇_1 is to avoid an infinite iteration for the function body called with a given parameter (which is useless here since there are finitely many $\langle \mathtt{L},\ \rho\rangle$, $\mathtt{L} \in \mathsf{lab}[\![\mathtt{P}]\!]$ for a given ρ) and the second widening ∇_2 is used to avoid infinitely many calls of the function with different parameters

(which is possible here, see an example in [16, p. 83]). The notion of *generalization* in the partial evaluation literature (see [16, p. 83]) is an example of ∇_2.

A very simple form of widening consists in replacing $\mathtt{WL} \cup \{\langle \mathtt{L}_1, \rho_r\rangle\}$ by $\mathtt{WL} \nabla \{\langle \mathtt{L}_1, \rho_r\rangle\}$ in **Fig. 8** and to use a threshold n for the size of WL as in $\mathtt{WL} \nabla \{\langle \mathtt{L}_1, \rho_r\rangle\} \triangleq$ *if* $|\mathtt{WL}| < n$ *then* $\mathtt{WL} \cup \{\langle \mathtt{L}_1, \rho_r\rangle\}$ *else* $\mathtt{WL} \cup \{\langle \mathtt{L}_1, \partial\rangle\}$. When the threshold n is overrun, $\mathtt{Q} := \mathtt{Q} \cup \{\langle \mathtt{L}, \rho\rangle : \mathtt{A}_r \to \langle \mathtt{L}_1, \rho_r\rangle;\}$ must be replaced by $\mathtt{Q} := \mathtt{Q} \cup \{\langle \mathtt{L}, \rho\rangle : \mathtt{A}_r \to \mathtt{L}_1;\}$ (where $\mathtt{L}_1$ is a shorthand for $\langle \mathtt{L}_1, \partial\rangle$ so that a potentially diverging specialization is cancelled over the threshold and all subject commands syntactically reachable from $\mathtt{L}_1$ in P will be added to the specialized program Q) [5]. Many alternative widenings are considered or referenced in [16, Ch. 14]. Since the widening provides a $\sqsubseteq$ over-approximation its observational correctness must be checked.

7.3 Binding Time Analysis

Binding time analysis (BTA for short) is used in offline partial evaluation in order to determine which variables are static at each program point. It is a static program analysis by abstract interpretation [16, Sec. 15.1.4], [28]. Formally, a *pointwise division* $\delta \in \mathbb{L} \longmapsto \wp(\mathbb{X})$ is a binding time information associating a set of static variables to each program point [16, Sec. 4.9.1] [6]. The meaning $\gamma^{\mathrm{BTA}}\langle \mathtt{L}_0, \mathcal{X}_0\rangle[\delta]$ of a division δ is relative to a given initial program point $\mathtt{L}_0$ and a given corresponding set $\mathcal{X}_0$ of variables which are assumed to be static. $\gamma^{\mathrm{BTA}}\langle \mathtt{L}_0, \mathcal{X}_0\rangle[\delta]$ is the set of program execution traces σ for which, when starting at program point $\mathsf{lab}[\sigma] = \mathtt{L}_0$ with $\delta[\mathtt{L}_0] \subseteq \mathcal{X}_0$, the values $\mathsf{env}[\sigma_{i+1}]|_{\delta[\mathsf{lab}[\sigma_{i+1}]]}$ of the static variables $\delta[\mathsf{lab}[\sigma_{i+1}]]$ at a given program $\mathsf{lab}[\sigma_{i+1}]$ can be statically computed in terms of the values $\mathsf{env}[\sigma_i]|_{\delta[\mathsf{lab}[\sigma_i]]}$ of the static variables $\delta[\mathsf{lab}[\sigma_i]]$ at the predecessor program points $\mathsf{lab}[\sigma_i]$ only. If $\langle \rho_r, \mathtt{A}_r\rangle = \mathsf{R}[\![\mathsf{act}[\sigma_i]]\!](\mathsf{env}[\sigma_i]|_{\delta[\mathsf{lab}[\sigma_i]]})$ is the specialization of the action $\mathsf{act}[\sigma_i]$ of state σ_i to the static part $\mathsf{env}[\sigma_i]|_{\delta[\mathsf{lab}[\sigma_i]]}$ of the environment $\mathsf{env}[\sigma_i]$ of state σ_i then $\mathsf{dom}[\rho_r]$ is the maximal set of static variables after executing the action of state σ_i with static variables $\delta[\mathsf{lab}[\sigma_i]]$ so the division should provide a subset $\delta[\mathsf{lab}[\sigma_{i+1}]]$ of this maximal set after execution of this action ($\delta = \boldsymbol{\lambda}\mathtt{L}{\cdot}\emptyset$ being therefore always correct). Formally:

$$\begin{array}{rcl}\gamma^{\mathrm{BTA}}\langle \mathtt{L}_0, \mathcal{X}_0\rangle[\delta] & \triangleq & \{\sigma \mid (\mathsf{lab}[\sigma] = \mathtt{L}_0) \Rightarrow (\delta[\mathtt{L}_0] \subseteq \mathcal{X}_0 \wedge \\ & & \forall i \in [0, \#\sigma - 1[: \mathsf{suc}[\sigma_i] = \mathsf{lab}[\sigma_{i+1}] \wedge \langle \rho_r, \mathtt{A}_r\rangle = \\ & & \mathsf{R}[\![\mathsf{act}[\sigma_i]]\!](\mathsf{env}[\sigma_i]|_{\delta[\mathsf{lab}[\sigma_i]]}) \wedge \delta[\mathsf{lab}[\sigma_{i+1}]] \subseteq \mathsf{dom}[\rho_r])\}\ .\end{array}$$

We have the abstraction:

$$\mathsf{po}\langle \wp(\mathfrak{D}^*); \subseteq\rangle \underset{\alpha^{\mathrm{BTA}}}{\overset{\gamma^{\mathrm{BTA}}}{\leftrightarrows}} \mathsf{po}\langle (\mathbb{L} \times \mathbb{X}) \longmapsto \mathbb{L} \longmapsto \wp(\mathbb{X});\ \dot{\supseteq}\rangle$$

We prefer to understand BTA as an abstraction of the program semantics rather than of the online partial evaluation semantics of Sec. 7.1.4 in order to be able to design an offline partial evaluation semantics without having to first design an online partial evaluation semantics (as in [2]).

[5] Formally, the widening ∇_2 is therefore on both Q and WL.

[6] This set of static variables is often encoded by its characteristic function as a map from the program variables to $\{S, D\}$ where S stands for "belongs to the set" (static) and D stands for "does not belong to the set" (dynamic). It is very confusing to try to understand S and D themselves as "abstract values" (see [16, p. 314]).

In the following we assume that a correct division $\delta[\mathtt{P}]$ is available for the program P, that is for all $\mathtt{L}_0 \in \mathbb{L}$ and all $\mathcal{X}_0 \subseteq \mathbb{X}$, $\mathsf{S}^*[\![\mathtt{P}]\!] \subseteq \gamma^{\mathrm{BTA}}\langle \mathtt{L}_0, \mathcal{X}_0\rangle[\delta[\mathtt{P}]]$. In order to avoid a particular case for `stop` commands, we extend $\delta \in \mathbb{L} \longmapsto \wp(\mathbb{X})$ to $\delta \in \mathbb{L} \longmapsto \wp(\mathbb{X})$ by $\delta[\mathtt{l}] \triangleq \emptyset$.

7.4 Offline Partial Evaluation

Offline partial evaluation [16, Ch. 7] uses a division δ specifying which inputs are static and initial values $\rho_0|_{\delta[\mathtt{L}_0]}$ of the static variables at the program entry point $\mathtt{L}_0$. It considers only those execution traces σ starting at $\mathtt{L}_0$ with values $\mathsf{env}[\sigma]$ of the variables which are those assumed for the initial static variables (as specified by the condition $\rho_0|_{\delta[\mathtt{L}_0]} \Subset \mathsf{env}[\sigma]$):

$$\begin{array}{rcl}\alpha^{\mathrm{PE}}_{\mathrm{off}}[\delta, \mathcal{T}]\langle \mathtt{L}_0, \rho_0\rangle & \triangleq & \{\mathsf{PE}_{\mathrm{off}}[\delta, \sigma]\langle \mathtt{L}_0, \rho_0\rangle \mid \sigma \in \mathcal{T} \wedge (\sigma \neq \vec{\epsilon}) \\ & & \Rightarrow (\mathsf{lab}[\sigma] = \mathtt{L}_0 \wedge \rho_0|_{\delta[\mathtt{L}_0]} \Subset \mathsf{env}[\sigma])\}\end{array}$$

The partial evaluation of a trace computes static values and specializes actions for dynamic values as in the case of online partial evaluation (11) but for the fact that the considered static values are only those specified by the division:

$$\begin{array}{rcll}\mathsf{PE}_{\mathrm{off}}[\delta, \langle \rho, \mathtt{L}_0 : \mathtt{A} \to \mathtt{L}_1;\rangle\sigma]\langle \mathtt{L}_0, \rho_0\rangle & \triangleq & & (13) \\ \quad \text{let } \rho'_0 = \rho_0|_{\delta[\mathtt{L}_0]} \text{ and } \mathtt{L}'_0 = \langle \mathtt{L}_0, \rho'_0\rangle \text{ and } \langle \rho_r, \mathtt{A}_r\rangle = \mathsf{R}[\![\mathtt{A}]\!]\rho'_0 & & \\ \quad\quad \text{and } \rho'_r = \rho_r|_{\delta[\mathtt{L}_1]} \text{ and } \mathtt{L}'_1 = \langle \mathtt{L}_1, \rho'_r\rangle \text{ in} & & \\ \quad\quad\quad \langle \rho \setminus \delta[\mathtt{L}_0], \mathtt{L}'_0 : \mathtt{A}_r \to \mathtt{L}'_1;\rangle\mathsf{PE}_{\mathrm{off}}[\delta, \sigma]\langle \mathtt{L}_1, \rho'_r\rangle & &\end{array}$$

(Note that by induction on the length of traces and correctness of the BTA, we have $\delta[\mathtt{L}_0] \subseteq \mathsf{dom}[\rho_0]$ and $\delta[\mathtt{L}_1] \subseteq \mathsf{dom}[\rho_r]$.) The design of the offline specialization algorithm is then similar to that of online specialization as shown in Sec. 7.1.1. The resulting algorithm is therefore similar to **Fig. 8** but for the fact that static environments (i.e. ρ_0, ρ and ρ_r) are restricted to the division specified by the preliminary BTA (i.e. respectively $\rho_0|_{\delta[\mathtt{L}_0]}$, $\rho|_{\delta[\mathtt{L}]}$ and $\rho_r|_{\delta[\mathtt{L}_1]}$).

7.5 Mixline Partial Evaluation

Widenings offer a continuum between online and offline partial evaluation to achieve mixline partial evaluation. A simple example would consist in using the online partial evaluator of **Fig. 8** with a widening, as considered in Sec. 7.2, but with a history-based one that computes for each label $\mathtt{L}_1$ a division $\delta[\mathtt{L}_1]$ which is initially $\mathsf{var}[\![\mathtt{P}]\!]$ and is progressively restricted during the partial evaluation by cumulating the intersection of the domains of all the static environments ρ_r computed for that label $\mathtt{L}_1$. The specialization $\mathsf{R}[\![\mathtt{A}]\!]$ would be restricted to that division $\delta[\mathtt{L}_1]$ as in the offline partial evaluation of Sec. 7.4. With this widening, a BTA is performed *during* the mixline partial evaluation.

8. EXAMPLE 3: STATIC PROGRAM MONITORING

Program monitoring consists in restricting the possible executions $\mathsf{S}^*_t[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ in order to enforce a safety property. An example is the insertion of run-time checks for checking errors such as division by zero and out of bound array indexing. Another example is the enforcement of security policies by modifying object code for a target system before that system is executed so as to halt that system whenever it is about to violate some security policy of concern [10, 25]. This is similar to the idea of *observers*

in synchronous programming, i.e. a program that observes the behavior of the subject program and decides whether it is correct [11].

8.1 Correctness of the Monitoring Transformation

Let M be the abstract specification of the program property to be enforced. The semantics of the transformed program $\mathtt{t}^m_M[\![\mathtt{P}]\!]$ should, up to an observational abstraction $\alpha^m_{\mathcal{O}}$, be a subset of the executions of P (so that $\mathtt{t}^m_M[\![\mathtt{P}]\!]$ refines P) and satisfy the abstract monitoring specification M. Formally, $\alpha^m_{\mathcal{O}}(\mathtt{S}^*_i[\![\mathtt{t}^m_M[\![\mathtt{P}]\!]]\!]) \subseteq \alpha^m_{\mathcal{O}}(\mathtt{S}^*_i[\![\mathtt{P}]\!]) \cap \alpha^m_{\mathcal{O}}(\gamma(M))$ (where equality is preferred since most precise and $\gamma(M)$ is the semantic meaning of the abstract specification M as defined below).

8.2 Observational Abstraction

We choose the abstraction $\alpha^m_{\mathcal{O}}$ to only observe the modifications to the successive environments during execution of P that is $\alpha^m_{\mathcal{O}}(\mathcal{T}) \triangleq \{\alpha^m_{\mathcal{O}}(\sigma) \mid \sigma \in \mathcal{T}\}$ and for traces $\alpha^m_{\mathcal{O}}(\vec{\epsilon}) \triangleq \vec{\epsilon}$, $\alpha^m_{\mathcal{O}}(\langle\rho,\, \mathtt{C}\rangle\langle\rho,\, \mathtt{C}'\rangle\sigma) \triangleq \alpha^m_{\mathcal{O}}(\langle\rho,\, \mathtt{C}'\rangle\sigma)$ and $\alpha^m_{\mathcal{O}}(\langle\rho,\, \mathtt{C}\rangle\langle\rho',\, \mathtt{C}'\rangle\sigma) \triangleq \rho|_{\mathsf{var}[\![\mathtt{P}]\!]}\alpha^m_{\mathcal{O}}(\langle\rho',\, \mathtt{C}'\rangle\sigma)$ when $\rho \neq \rho'$.

8.3 Reference Monitor

We choose the abstract monitoring specification M to be provided as a program $\mathtt{M} \in \mathbb{P}$ called the *reference monitor* (as usual in the particular context of security policy enforcement [10, 25]), $\gamma(M)$ being now its semantics $\mathtt{S}^*_i[\![\mathtt{M}]\!]$. (Any other mean for specifying finite sequences, such as automata [10, 25], grammars, safety temporal formula, etc. would do as well for M and could be handled equally well in the framework provided their semantics can be expressed in fixpoint form). The subject program P is assumed to have its labels in $\mathbb{L}_{\mathtt{P}}$ and its actions in $\mathbb{A}_{\mathtt{P}}$ so its execution traces belong to $\mathfrak{D}^*[\![\mathbb{L}_{\mathtt{P}}, \mathbb{A}_{\mathtt{P}}]\!]$ where $\mathfrak{D}^*[\![\mathcal{L}, \mathcal{A}]\!]$ is the set of finite partial execution traces with labels in $\mathcal{L}$ and actions in $\mathcal{A}$. The reference monitor M is assumed to have its labels in $\mathbb{L}_{\mathtt{M}}$ and its actions in $\mathbb{A}_{\mathtt{P}} \cup \mathbb{A}_{\mathtt{M}}$ with $\mathbb{A}_{\mathtt{P}} \cap \mathbb{A}_{\mathtt{M}} = \emptyset$ so its execution traces belong to $\mathfrak{D}^*[\![\mathbb{L}_{\mathtt{M}}, \mathbb{A}]\!]$. Actions in $\mathbb{A}_{\mathtt{M}}$ are specific to the monitor while actions in $\mathbb{A}_{\mathtt{P}}$ specify which actions of P are allowed (in practice one would prefer abstract actions in M designating a set of concrete actions in P such as $\neg\mathtt{A}$ standing for all actions in P but A).

8.4 Semantic Monitoring Transformation

The transformed program $\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$, which incorporates both the subject program P and the reference monitor M has its labels in $\mathbb{L}_{\mathtt{P}} \times \mathbb{L}_{\mathtt{M}}$ (so $\mathbb{L} = \mathbb{L}_{\mathtt{P}} \cup \mathbb{L}_{\mathtt{M}} \cup (\mathbb{L}_{\mathtt{P}} \times \mathbb{L}_{\mathtt{M}})$) and its actions in $\mathbb{A} = \mathbb{A}_{\mathtt{P}} \cup \mathbb{A}_{\mathtt{M}}$. A state $\langle\rho,\, \langle \mathtt{L},\, \mathtt{L}'\rangle : \mathtt{A} \to \langle \mathtt{L},\, \mathtt{L}'\rangle;\rangle$ in a transformed trace must have $\mathtt{L} = \mathtt{L}$ when $\mathtt{A} \notin \mathbb{A}_{\mathtt{P}}$ (in which case the monitor M makes a step and the subject program P makes no progress). To avoid a particular case for `stop` commands, we set $\mathtt{l} \cong \langle \mathtt{l},\, \mathtt{L}\rangle \cong \langle \mathtt{L},\, \mathtt{l}\rangle$. The set of such partial execution traces is written $\mathfrak{D}^*[\![\mathbb{L}_{\mathtt{P}} \times \mathbb{L}_{\mathtt{M}}, \mathbb{A}]\!]$. Intuitively, $\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$ is equivalent to P but blocks when stopped by the reference monitor M.

The projection of a trace $\sigma \in \mathfrak{D}^*[\![\mathbb{L}_{\mathtt{P}} \times \mathbb{L}_{\mathtt{M}}, \mathbb{A}]\!]$ according to P and M is defined as (starting with the projection of states):

$$
\begin{array}{rcll}
\langle\rho,\, \langle \mathtt{L},\, \mathtt{L}'\rangle : \mathtt{A} \to \langle \mathtt{L},\, \mathtt{L}'\rangle;\rangle\backslash_{\mathtt{P}} & & & \\
\quad \triangleq & \langle\rho|_{\mathsf{var}[\![\mathtt{P}]\!]},\, \mathtt{L} : \mathtt{A} \to \mathtt{L};\rangle, & & \text{if } \mathtt{A} \in \mathbb{A}_{\mathtt{P}}; \\
\langle\rho,\, \langle \mathtt{L},\, \mathtt{L}'\rangle : \mathtt{A} \to \langle \mathtt{L},\, \mathtt{L}'\rangle;\rangle\backslash_{\mathtt{M}} & \triangleq & \langle\rho|_{\mathsf{var}[\![\mathtt{M}]\!]},\, \mathtt{L}' : \mathtt{A} \to \mathtt{L}';\rangle; &
\end{array}
$$

$$
\begin{array}{rcll}
\vec{\epsilon}\backslash_{\mathtt{P}} & \triangleq & \vec{\epsilon}\backslash_{\mathtt{M}} \triangleq \vec{\epsilon}; & \\
(s\sigma)\backslash_{\mathtt{P}} & \triangleq & (s\backslash_{\mathtt{P}})(\sigma\backslash_{\mathtt{P}}), & \text{if } \mathsf{act}[s] \in \mathbb{A}_{\mathtt{P}}; \\
& \triangleq & \sigma\backslash_{\mathtt{P}}, & \text{if } \mathsf{act}[s] \notin \mathbb{A}_{\mathtt{P}}; \\
(s\sigma)\backslash_{\mathtt{M}} & \triangleq & (s\backslash_{\mathtt{M}})(\sigma\backslash_{\mathtt{M}}) \ . &
\end{array}
$$

The program monitoring semantic transformation can now be defined as $\mathtt{t}^m[\mathcal{T}_{\mathtt{P}}, \mathcal{T}_{\mathtt{M}}] \triangleq \{\sigma \in \mathfrak{D}^*[\![\mathbb{L}_{\mathtt{P}} \times \mathbb{L}_{\mathtt{M}}, \mathbb{A}]\!] \mid \sigma\backslash_{\mathtt{P}} \in \mathcal{T}_{\mathtt{P}} \wedge \sigma\backslash_{\mathtt{M}} \in \mathcal{T}_{\mathtt{M}}\}$. This is an abstraction $\mathsf{po}\langle\mathfrak{D}^*[\![\mathbb{L}_{\mathtt{P}}, \mathbb{A}_{\mathtt{P}}]\!] \times \mathfrak{D}^*[\![\mathbb{L}_{\mathtt{M}}, \mathbb{A}]\!];\ \subseteq^2\rangle \xrightarrow[\mathtt{t}^m]{\gamma_{\mathtt{t}^m}} \mathsf{po}\langle\mathfrak{D}^*[\![\mathbb{L}_{\mathtt{P}} \times \mathbb{L}_{\mathtt{M}}, \mathbb{A}]\!];\ \subseteq\rangle$. Its correctness follows from the fact that $\alpha^m_{\mathcal{O}}(\mathtt{t}^m[\mathtt{S}^*_i[\![\mathtt{P}]\!], \mathtt{S}^*_i[\![\mathtt{M}]\!]]) = \alpha^m_{\mathcal{O}}(\mathtt{S}^*_i[\![\mathtt{P}]\!]) \cap \alpha^m_{\mathcal{O}}(\mathtt{S}^*_i[\![\mathtt{M}]\!])$.

8.5 Monitored Fixpoint Semantics

The monitored semantics $\mathtt{t}^m[\mathtt{S}^*_i[\![\mathtt{P}]\!], \mathtt{S}^*_i[\![\mathtt{M}]\!]]$ can now be expressed in fixpoint form. In order to use (1), the pair of semantics of P and Q is first expressed in fixpoint form as $\mathsf{lfp}^{\subseteq^2}\, \boldsymbol{\lambda}\langle\mathcal{T},\, \mathcal{T}'\rangle \cdot \langle \mathtt{F}^*_i[\![\mathtt{P}]\!]\mathcal{T},\ \mathtt{F}^*_i[\![\mathtt{M}]\!]\mathcal{T}'\rangle$. Then it is abstracted by the monitoring semantic transformation $\mathtt{t}^m$ as $\mathsf{lfp}^{\subseteq}\, \boldsymbol{\lambda}\mathcal{T} \cdot \mathit{Init} \cup \mathit{Next}(\mathcal{T})$ where the term $\mathit{Init} = \{s_1 \ldots s_n \mid n > 0 \wedge \forall i \in [1, n[\colon \mathsf{act}[s_i] \notin \mathbb{A}_{\mathtt{P}} \wedge \forall i \in [1, n[\colon \mathsf{lab}[s_i\backslash_{\mathtt{P}}] = \mathsf{suc}[s_i\backslash_{\mathtt{P}}] = \mathsf{lab}[s_{i+1}\backslash_{\mathtt{P}}] \wedge \mathsf{act}[s_n] \in \mathbb{A}_{\mathtt{P}} \wedge s_n\backslash_{\mathtt{P}} \in \mathfrak{I}[\![\mathtt{P}]\!] \wedge s_1\backslash_{\mathtt{M}} \in \mathfrak{I}[\![\mathtt{M}]\!] \wedge \forall i \in [2, n] : s_n\backslash_{\mathtt{M}} \in \mathtt{S}[\![\mathtt{M}]\!]s_{n-1}\backslash_{\mathtt{M}}\}$ corresponds to the initialization of the reference monitor M making its own computations before initializing the source program P and the term $\mathit{Next}(\mathcal{T}) = \{\sigma s_1 \ldots s_n \mid \sigma s_1 \in \mathcal{T} \wedge \mathsf{act}[s_1] \in \mathbb{A}_{\mathtt{P}} \wedge \forall i \in [2, n[\colon \mathsf{act}[i] \notin \mathbb{A}_{\mathtt{P}} \wedge \mathsf{act}[s_n] \in \mathbb{A}_{\mathtt{P}} \wedge \forall i \in [1, n[\colon \mathsf{suc}[s_i\backslash_{\mathtt{P}}] = \mathsf{lab}[s_{i+1}\backslash_{\mathtt{P}}] = \mathsf{suc}[s_{i+1}\backslash_{\mathtt{P}}] \wedge s_n\backslash_{\mathtt{P}} \in \mathtt{S}[\![\mathtt{P}]\!]s_{n-1}\backslash_{\mathtt{P}} \wedge \forall i \in [1, n[\colon s_{i+1}\backslash_{\mathtt{M}} \in \mathtt{S}[\![\mathtt{M}]\!]s_i\backslash_{\mathtt{P}}\}$ corresponds to a step of the source program P controlled by several steps of the monitor M.

8.6 Iterative Sources-to-Source Monitoring Transformation

The transformed monitored program $\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$ can now be expressed in fixpoint form using the semantics-to-syntax abstraction $\mathbb{p}^*$ defined in Sec. 4.6. Solving the fixpoint equation by chaotic iterations, we get the iterative transformation algorithm of **Fig. 9**. The algorithm is simpler than the

monitoring(P, $\mathfrak{L}[\![\mathtt{P}]\!]$, M, $\mathfrak{L}[\![\mathtt{M}]\!]$) =
  Q := $\emptyset$; WL := $\{\langle \mathtt{L},\, \mathtt{L}'\rangle \mid \mathtt{L} \in \mathfrak{L}[\![\mathtt{P}]\!] \wedge \mathtt{L}' \in \mathfrak{L}[\![\mathtt{M}]\!]\}$;
  while WL contains an unmarked pair $\langle \mathtt{L},\, \mathtt{L}'\rangle \not\cong \mathtt{l}$ **do**
    mark $\langle \mathtt{L},\, \mathtt{L}'\rangle$ in WL;
    forall $\mathtt{L} : \mathtt{A} \to \mathtt{L}; \in \mathtt{P}$ **and** $\mathtt{L}' : \mathtt{A}' \to \mathtt{L}'; \in \mathtt{M}$ **do**
      if $\mathtt{A} = \mathtt{A}'$ **then** — *step* A *in source and monitor*
        Q := Q $\cup$ $\{\langle \mathtt{L},\, \mathtt{L}'\rangle : \mathtt{A} \to \langle \mathtt{L},\, \mathtt{L}'\rangle;\}$;
        WL := WL $\cup$ $\{\langle \mathtt{L},\, \mathtt{L}'\rangle\}$
      elsif $\mathtt{A}' \notin \mathbb{A}_{\mathtt{P}}$ **then** — *control step* A′ *in monitor*
        Q := Q $\cup$ $\{\langle \mathtt{L},\, \mathtt{L}'\rangle : \mathtt{A}' \to \langle \mathtt{L},\, \mathtt{L}'\rangle;\}$;
        WL := WL $\cup$ $\{\langle \mathtt{L},\, \mathtt{L}'\rangle\}$
  end end end;
  return Q.

Figure 9: A simple program monitoring algorithm

copying and simplification of the reference monitor automaton at each program point in [10].

8.7 Program Proof by Transformation

Finally, program monitoring can be used as a proof method. If the semantics of the transformed program $\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$ (maybe after static analysis and optimization) is empty (e.g. when

$\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$ itself is empty), the program P does not satisfy the specification M. An example is the contrapositive automata-theoretic based model-checking [27].

When the observational abstraction of the semantics of the subject and transformed programs P and $\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$ are equal (e.g. when, maybe after further optimizations such as partial evaluation, P and $\mathtt{t}^m_{\mathtt{M}}[\![\mathtt{P}]\!]$ are isomorphic up to label renaming), then P does satisfy the specification M. An example is the introduction of run-time tests in programs and their subsequent elimination by program analysis [4]. Another example is that of observers in synchronous programming where the verification consists in checking, by traversing the finite automaton built by the compiler, that the parallel composition of the subject program and its observer never causes the observer to complain [11].

This method of proving program properties by program transformation remains to be fully explored.

9. CONCLUSION

We have shown that program transformation can be formalized within abstract interpretation theory. This leads to a new construction of program transformations as syntactic approximations of provably correct semantic transformations. This has been applied to the simple case of constant propagation, to online and offline partial evaluation which is certainly the most widely applicable and practical classical program transformation and to program monitoring (such as security policy enforcement). The framework unifies the static analysis and transformation of programs within solid semantic foundations. For offline transformation we use a specification of the static analysis algorithm as an abstraction of the program semantics thus making the transformation correctness proof independent of the particular static program analysis algorithm which is used. By using widening techniques as well as abstract domain combination techniques this leads to formal methods for combining static analyses and transformations more intimately. Although illustated on a low level imperative language, the formalization is language independent. It is indeed applicable to any computational process, for example to database search. Our formalization of program transformation by abstract interpretation makes very few hypotheses on the considered transformations and programming languages (only required to have some well-defined operational semantics) so that the model should be of very general scope.

Acknowledgments: B. Blanchet, J. Feret, N. Jones, F. Logozzo, L. Mauborgne, A. Miné & X. Rival for comments.

10. REFERENCES

[1] C. Consel and C. Danvy. Tutorial notes on partial evaluation. *20th POPL*, 493–501, 1993, ACM Press.

[2] C. Consel and S. Khoo. On-line and off-line partial evaluation: Semantics specifications and correctness proofs. *J. Func. Prog.*, 5(4):461–500, 1995.

[3] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *ENTCS*, 6, 25 p., 1997. `http://www.elsevier.nl/locate/entcs/volume6.html`.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *4th POPL*, 238–252, 1977, ACM Press.

[5] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. *IFIP Conf. on Formal Description of Programming Concepts*, 237–277, 1977, North-Holland.

[6] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *6th POPL*, 269–282, 1979, ACM Press.

[7] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. *PLILP '92*, LNCS 631, 269–295, 1992, Springer.

[8] P. Cousot and R. Cousot. A case study in abstract interpretation based program transformation: Blocking command elimination. *ENTCS*, 45, 2001. `http://www.elsevier.nl/locate/entcs/volume45.html`, 23 p.

[9] L. de Alfaro and Z. Manna. Temporal verification by diagram transformations. *CAV '96*, LNCS 1102, 288–299, 1996, Springer.

[10] Ú. Erlingsson and F. Schneider. SASI enforcement of security policies: a retrospective. *NSPW '99*, 87–95, 1999, ACM Press.

[11] N. Halbwachs, F. Lagnier, and P. Raymond. Synchronous observers and the verification of reactive systems. *AMAST '93*, Workshops in Comp., 83–96, 1994, Springer.

[12] G. Jin, Z. Li, and F. Chen. A theoretical foundation for program transformations to reduce cache thrashing due to true data sharing. *Theoret. Comput. Sci.*, 255(1–2):449–481, 2001.

[13] N. Jones. Abstract interpretation and partial evaluation in functional and logic programming. *ISLP '94*, 17–22, 1994, MIT Press.

[14] N. Jones. An introduction to partial evaluation. *ACM Comput. Surv.*, 28(3):480–504, 1996.

[15] N. Jones. Combining abstract interpretation and partial evaluation (brief overview). *SAS '97*, LNCS 1302, 396–405, 1997, Springer.

[16] N. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, 1993.

[17] N. Jones and A. Mycroft. Data flow analysis of applicative programs using minimal function graphs: abridged version. *13th POPL*, 296–306, 1986, ACM Press.

[18] G. Kildall. A unified approach to global program optimization. *1st POPL*, 194–206, 1973, ACMpress.

[19] D. Monniaux. Abstract interpretation of probabilistic semantics. *SAS'2000*, LNCS 1824, 322–339, 2000, Springer.

[20] R. Paige. Symbolic finite differencing – part I. *ESOP '90*, LNCS 432, 36–56, 1990, Springer.

[21] R. Paige. Future directions in program transformations. *ACM SIGPLAN Not.*, 32(1):94–97, 1997.

[22] A. Pnueli, O. Shtrichman, and M. Siegel. The code validation tool CVT: Automatic verification of a compilation process. *STTT*, 2(2):192–201, 1998.

[23] J. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–248, 1993.

[24] D. Sands. Proving the correctness of recursion-based automatic program transformations. *Theoret. Comput. Sci.*, 167(1–2):193–233, 1996.

[25] F. Schneider. Enforceable security policies. *TISSEC*, 3(1):30–50, 2000.

[26] P. Steckler and M. Wand. Selective thunkification. *SAS '94*, LNCS 864, 162–178. Springer, 1994.

[27] M. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. System Sci.*, 32(2):183–221, 1986.

[28] F. Védrine. Binding-time analysis and strictness analysis by abstract interpretation. *SAS '95*, LNCS 983, 400–417, 1995, Springer.

[29] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoret. Comput. Sci.*, 73(2):231–248, 1990.

[30] M. Weiser. Program slicing. *IEEE Trans. Software Engrg.*, SE-10(4):352–357, 1984.

[31] H. Yang and Y. Sun. Reverse engineering and reusing COBOL programs: A program transformation approach. *IWFM '97*, Electronic Workshops in Computing, 1997. `http://ewic.org.uk/ewic/workshop/view.cfm/IWFM-97`.