INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Abstracting Call-Stacks for Interprocedural Verification of Imperative Programs

Bertrand JEANNET and Wendelin SERWE

N°4904

Juillet 2003

THÈMES 1 et 2

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4904--FR+ENG

INRIA
RENNES

Abstracting Call-Stacks for Interprocedural Verification of Imperative Programs

Bertrand Jeannet * and Wendelin Serwe †

Thèmes 1 et 2 — Réseaux et systèmes — Génie logiciel
et calcul symbolique
Projets VerTeCs & Lande, Action de Recherche Coopérative ModoCop

Rapport de recherche n° 4904 — Juillet 2003 — 51 pages

Abstract: We address in this paper the verification of imperative programs with recursion. Our approach consists in using abstract interpretation to relate the standard semantics of imperative programs to an abstract semantics, by the mean of a Galois connection, and then to resort to *intraprocedural* techniques, which can be applied on the abstract semantics. This approach allows the reuse of classical intraprocedural techniques with few modifications, generalises existing approaches to interprocedural analysis and offers additional flexibility, as it keeps substantial information on the call-stack of the analysed program.

Key-words: interprocedural verification, abstract interpretation

(Résumé : tsvp)

* Bertrand.Jeannet@irisa.fr
† Wendelin.Serwe@inria.fr, INRIA Rhône-Alpes, ZIRST, 655 Avenue de l'Europe, Montbonnot, 38334 Saint Ismier cedex, France

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Abstraction de pile pour la vérification interprocédurale de programmes impératifs

Résumé : Nous nous intéressons à la vérification de programmes impératifs récursifs. Notre approche consiste à utiliser le cadre théorique de l'interprétation abstraite pour relier la sémantique standard des programmes impératifs à une sémantique abstraite, au moyen d'une connexion de Galois, puis de recourir à des techniques d'analyse *intraprocédurale* qui peuvent être directement appliquée à la sémantique abstraite. Cette approche permet de réutiliser des techniques interprocédurales classiques avec peu de modifications, généralise des méthodes existantes pour l'analyse interprocédurale, et offre une souplesse accrue en maintenant des informations précises sur la pile d'appel du programme analysé.

Mots-clé : vérification interprocédurale, interprétation abstraite

1 Introduction

We consider the interprocedural analysis and verification of an important class of imperative programs, taking into account the values of local and global variables, using the framework of abstract interpretation.

The main difficulty we address is the abstraction of the call stack which contains both, control points and values of local variables. Thus, the structure of the state space of a program is quite complex. In order to reuse standard abstract interpretation techniques, we suggest a more abstract semantics, which still keeps precise information on stack contents, and which can be efficiently implemented using standard intraprocedural abstract interpretation for data values. We give both correctness and optimality results of the proposed abstract semantcis w.r.t. the standard semantics.

Verification versus Data Flow Analysis. Not so many papers are explicitly devoted to *verification* of interprocedural programs, which contrasts with the important literature available on interprocedural data flow analysis for compilation applications. Compilation oriented data flow analysis and verification by algorithmic techniques actually share the same formal framework: in both cases, the aim is to compute for each control point of a program a property P that holds at that point, with P belonging to a lattice of properties (or predicates) L. However from a practical point of view data flow analysis and verification differ in two aspects: (i) in verification one tries to verify an user-defined property, whereas data flow analysis is less often driven by a precise property; (ii) the lattices of properties considered in verification are usually more complex than in the case of classical data flow analysis. The main difference actually lies in this second point: verification requires very precise information to be able to conclude to satisfaction (or non satisfaction) of the specified property, whereas in compilation unprecise information just restricts the possibilities for optimisation, without making the analysis useless. Typically in verification, one needs *flow-sensitive*[1], *attribute-dependent*[2], *infinite* or even *infinite height* lattices. Due to these requirements, several interprocedural techniques designed for application in compilation are not directly applicable to verification, although they can of course be used to perform very useful preprocessing analyses.

In this paper, our choices are oriented towards precise verification of programs taking into account variables values.

Two Main Approaches to Interprocedural Analysis. One can distinguish two main approaches to interprocedural static analysis. The first one, the *functional approach* (after the name given by [SP81]), consists in using a denotational semantics of the analysed program and in proceeding in two steps. The first step consists in computing the predicate transformers associated to the procedures of the program, which implies solving fixpoint equations on predicate transformers. The second step then consists in applying to some input predicate the composition of predicate transformers along a program path to get the predicate holding at the end of the path. [CC77b, SP81, KS92] apply this approach to different classes of programs.

The second approach, which we call the *operational approach*, consists in adopting an operational semantics for programs. Here, like in many intraprocedural verification techniques, the predicates are propagated along the control points of the program, using only the predicate transformers associated to *elementary* program statements, until a fixpoint is reached. The analysis can be viewed as a symbolic execution of the program where values are just replaced by properties, without any computation of predicates transformers associated to blocks of instructions or procedures. However, as a real execution of the program is simulated, there is the necessity of taking into account the call stack of the program. Indeed, when a procedure returns to its caller, the call site and the local environment of the caller should be popped from the stack stored in order to jump back to the effective caller and to restore its

[1] That is to say, depending on values of variables and/or taking into account conditionals of the program.

[2] That is to say, attributes (or properties) attached to variables are inter-related.

local environment before the procedure call. The "call-strings approach" of [SP81], later generalized by [JM82], belong to this approach, as well as techniques based on pushdown systems [EK99]. Also, a recent extension [RSJ03] lies in-between the two approaches. Related work is discussed in more detail in Section 7, as well as in Sections 3.5 and 5.3.

Some Advantages of the Operational Approach. The functional approach is certainly more flexible, as the predicate transformers solution of the fixpoint equations can be applied to any new input predicate, whereas the operational approach requires a new fixpoint for each different input predicate a new fixpoint[3]. But the operational approach presents several advantages:

1. It requires the computation of a system of fixpoint equations in the domain L, *which is simpler* than the domain $L \to L$, especially if L is a complex lattice[4]. This comparison should however be attenuated, as in the operational approach the call stack has to be taken into account, so that the real domain of computation looks more like $\mathit{Stack} \times L$ where *Stack* is some (possibly simple) stack lattice.

2. The computations are *context-sensitive*: instead of computing predicate transformers valid for any input, which have to handle all possible cases, the operational approach takes into account only the cases encountered during the fixpoint computation. A typical example is a conditional instruction, the condition of which always evaluates to true (or false) during the symbolic execution, such that the analysis needs only to take into account one of the two branches. Computations are in some way "partially evaluated" by the input predicate, and so likely to be less expensive.

3. When L *is not distributive*, which is often the case when abstract interpretation is used, the previous observation makes the analysis also more precise.

Let us illustrate these three points by a simple example, in the intraprocedural case.

Example 1 *Consider the statement* `if x>=5 then x := x-5` *and the input predicate* $I = 0 \leq x \leq 1$. *We suppose that we use the lattice of convex polyhedra as the set of predicates, and that predicate transformers are represented relationally as polyhedra defined on unprimed and primed variable. The predicate transformer of the statement is*

$$\tau(x, x') = (x \leq 5 \sqcap x' = x) \sqcup (x \geq 5 \sqcap x' = x - 5) = (x - 5 \leq x' \leq x)$$

where $\sqcap$ *and* $\sqcup$ *are the meet and join operators in the convex polyhedra lattice. The application of* τ *to the predicate* I *yields the result*

$$\exists x : I(x) \wedge \tau(x, x') \equiv \exists x : (x - 1 \leq x' \leq x + 1) \sqcap (0 \leq x \leq 1) \equiv (-5 \leq x' \leq 1)$$

Now, if we apply the operational method, we have

$$\big(I(x) \sqcap (x \leq 5)\big) \sqcup \big((\lambda x.x - 5)(I(x) \sqcap (x \geq 5))\big) \equiv I(x) \sqcup (\lambda x.x - 5)(\bot) \equiv I(x) \equiv (0 \leq x \leq 1)$$

In the operational method, we manipulate one dimensional intervals instead of two dimensional polygons, we save one join operation and we get a noticeably more accurate result.

[3] We assume here that solving fixpoint equations is more expensive than applying predicate transformers.
[4] For instance, the domain $\mathbb{N} \to \mathbb{N}$ is not countable, whereas $\mathbb{N}$ is.

Our Method. The motivation of our work is to design an analysis using the operational approach, in order to take advantage of these three points. We address here the case of programs with global and local variables, using call-by-value parameter passing, so that there is no aliasing of variables in the stack. We also aim at a method allowing the verification of invariance properties, where the considered properties are equivalent to sets of states and the lattice of properties L is the powerset of states $\wp(S)$. The main challenge here is to take into account the call stack of the program. We tackle indeed a sensibly more complicated case than the call-string approach developed in [SP81], where only parameterless procedures are considered, so that stacks contain only call-sites. [JM82] describes a more general approach that we partially follow. Let us write the state space of such programs. If we consider an intraprocedural program with n (global or local) variables, its state space is

$$S_{\text{intra}} = \mathit{Ctrl} \times D^n \tag{1}$$

where *Ctrl* is the finite set of control points of the program and D the domain of values of the variables. If we consider an interprocedural program with n global variables and m local variables in each procedure, the state space becomes

$$S_{\text{inter}} = (\mathit{Ctrl} \times D^m)^+ \times D^n \tag{2}$$

where $+$ denotes the Kleene operator (for any set E, $E^+ \stackrel{\text{def}}{=} \bigcup_{i\geq 1} E^i$). The set $\mathit{Ctrl} \times D^m$ is the set of *activation records*, that store the local state of a procedure, and the set $(\mathit{Ctrl} \times D^m)^+$ is the stack of activation records, the top of which being the current activation record.

Because of undecidability problems, properties in $\wp(S)$ are usually approximated using abstract interpretation theory. While several abstract domains have been suggested for the domain $\wp(S_{\text{intra}})$ when D is the set of integers $\mathbb{Z}$ for instance, designing directly an abstract domain for $\wp(S_{\text{inter}})$ in the interprocedural case seems to us so challenging that we didn't even try: how to represent for instance $\mathbb{Z}^+$, even approximatively ? Of course, simple solutions exists, like drastically abstracting the stack of activation records by its top element, but we are looking for more precise solutions.

Our method proceeds thus in two steps:

1. We define a more abstract semantics, that enjoys a simpler state space than S_{inter} yet keeps substantial information on the call stack. More precisely, in this semantics the concrete set of properties $\wp\big(D^n \times (\mathit{Ctrl} \times D^m)^+\big)$ is replaced by an abstract set of properties $\big(\wp(\mathit{Ctrl}^+ \times D^{n+m})\big)^2$, in which in the Kleene operator does not apply any more to the data domain but only to the *finite* set of control points.

2. We resort then to more classical abstract interpretation techniques to design a effectively computable lattice approximating $\big(\wp(\mathit{Ctrl}^+ \times D^{n+m})\big)^2$.

Let us precise that although the first abstraction step is quite rigid, there is a full scale of solutions for the second abstraction step.

[JM82] attacks the very same problem quite similarly, by abstracting a stack by a pair composed of a *token* abstracting all but the top activation record, and an (abstract) environment for the top activation record. However, the set of tokens is not given a lattice structure, tokens are just enumerated, so that it cannot lead directly to an abstract lattice which is both implementable and precise enough to maintain information on the values of variables (of infinite domain) in the stack. A more precise comparison is given in in Section 7.

Contribution of this work. We describe a systematic way to analyse interprocedural programs, by directly abstracting stacks used by the standard operational semantics of imperative programs with an abstract domain through a Galois connection. The proposed stack abstraction is completely separated from a possible data abstraction which may follow, and allows to describe properties on stacks. Moreover this straightforward approach allow us to relate the result of the analysis w.r.t. the operational semantics of the analysed programs and to give some optimality results. The use of the abstract interpretation framework also enables us to benefit from classical techniques such as the combination of forward and backward analysis or state space partitioning, without requiring new correctness proofs. We also analyse practical issues about efficient implementation of the method.

Outline. The paper is organised as follows. We present first the program model we consider together with its operational and collecting semantics. We also study some properties of this standard semantics. We present in Section 3 a first stack abstraction, in the case of programs without global variables. We give correctness results as well as an optimality result under some additional but very common assumptions. This optimality result corresponds to well-known results already available in the literature, but is formulated and proved in a quite different way. In Section 4, we refine the previous abstraction by using call-strings in order to increase the precision of the analysis and to be able to reconstruct stack contents more precisely. In Section 5, we present an original technique for taking into account global variables, which generalises other techniques in the sense that it enables a finer tuning of the precision of the analysis. Section 6 describes the second step of our approach, giving hints how to abstract further the stack abstraction in order to obtain effective analysis methods. We compare our approach with related work in Section 7. Section 8 concludes.

2 Program Model and Standard Semantics

Our analysis considers programs written in a simple imperative programming language with non-nested procedure definitions, using a value parameter passing scheme[5]. Besides a fixed set of global variables, each procedure has its own fixed set of local variables. We do not precise the types of variables. In fact, variables can rather be thought of as dimensions and do not correspond necessarily to a syntactic variable in the program. For instance, a special global variable could represent the heap manipulated by the program, if dynamic allocation is considered. We consider programs as a set of procedures $P_0, \dots, P_p$, defined by their *intraprocedural Control Flow Graphs* (CFG for short). The main restrictions with respect to more complex programming languages are the absence of exceptions or non-local jumps, pointers to local variables (as it happens with reference parameter passing), pointers to procedures or procedural parameters[6].

We make the important assumption that formal parameters of procedures *are not modified* in the procedure. This allows to represent a relation between environments at the entry of the procedure and environments at any control point of the procedure, similarly as in [SP81, KS92]. It is trivial to add new local variables to ensure this assumption on an existing program.

2.1 Program Syntax

The syntactic domains we use are summarised in Table 1.

Program. A program $Prog = (GVar, (P_i)_{0 \leq i \leq p})$ is defined by a set $GVar$ of global variables, the vector of which is noted $\mathbf{g}$, and a set $(P_i)_{0 \leq i \leq p}$ of procedures. We do not have any main procedure, as we specify separately what are the initial states from which to start an analysis.

Procedures. A procedure $P_i = (LVar_i, FPar_i, FRet_i, G_i)$ is defined by its set of local variables $LVar_i$, its set of formal *call parameters* $FPar_i \subseteq LVar_i$, its set of formal *return parameters* $FRet_i \subseteq LVar_i$, and its intraprocedural CFG G_i. Considered as vectors, these sets are written respectively $\mathbf{l}_i$, $\mathbf{fp}_i = \langle \mathbf{fp}_i^{(1)}, \dots, \mathbf{fp}_i^{(m)} \rangle$ and $\mathbf{fr}_i = \langle \mathbf{fr}_i^{(1)}, \dots, \mathbf{fr}_i^{(n)} \rangle$.

[5] This convention is common in recent programming languages, like JAVA or ML.

[6] Dealing with this features could presently be done, but only by doing drastic abstractions.

Var	Variables: $\mathrm{Var} = GVar \cup \bigcup_i LVar_i$	
Instr	Instructions	
$GVar$, $\mathbf{g}$	Global variables of the program	
$LVar_i$, $\mathbf{l}_i$	Set of local variables of procedure P_i	
$FPar_i$, $\mathbf{fp}_i$	Tuple of formal call parameters of procedure P_i	$FPar_i \subseteq LVar_i$
$FRet_i$, $\mathbf{fr}_i$	Tuple of formal return parameters of procedure P_i	$FRet_i \subseteq LVar_i$
$G_i = \langle Ctrl_i, I_i \rangle$	Flow graph of the procedure P_i	
$Ctrl_i$	Control points of the procedure P_i	
$s_i, e_i \in Ctrl_i$	entry and exit points of procedure P_i	
$I_i : Ctrl_i \times Ctrl_i \to \mathrm{Instr}$	Instructions labelling edges of G_i	
$G = \langle Ctrl, I \rangle$	Interprocedural flow graph of the program	
$Ctrl = \bigcup_{0 \leq i \leq p} Ctrl_i$	Control points of the program	
$I : Ctrl \times Ctrl \to \mathrm{Instr}^{\mathsf{inter}}$	Instructions labelling edges of G	

Table 1: Syntactic domains

Intraprocedural CFG. An intraprocedural CFG is a graph $G_i = (Ctrl_i, I_i)$ where $Ctrl_i$ is the set of *control points* of procedure P_i, containing a unique entry control point s_i and a unique exit control point e_i. $I_i : Ctrl_i \times Ctrl_i \to \mathrm{Instr}$ is function labelling edges between control points with *instructions* belonging to Instr. The instructions we consider in this paper are either intraprocedural instructions or procedure invocations:

$$\begin{array}{lll} \mathrm{Instr} & ::= \langle R \rangle & \text{intraprocedural instruction, with } R \subseteq (GEnv \times LEnv)^2 \\ & \mid \ \langle \mathbf{y} := P_j(\mathbf{x}) \rangle & \text{procedure invocation, where the } \mathbf{x}^{(k)}\text{'s and } \mathbf{y}^{(l)}\text{'s are variables} \end{array}$$

Intraprocedural instructions are specified in a general way as a relation R describing how the global and top local environment are transformed by the instruction. Obviously, any assignment of a variable by an expression or any conditional instruction can be specified that way. Empty relations are used to specify that there is no instruction between two control points. The assumption that formal call parameters are not modified is reflected on relations by the assumption that $R(e_1, e_2)$ implies $e_1(\mathbf{fp}_j^{(k)}) = e_2(\mathbf{fp}_j^{(k)})$, and on procedure calls by the assumption that $\mathbf{fp}_i \cap \mathbf{y} = \emptyset$, where i is the number of the considered (calling) procedure. In the examples we will use the notations $\langle x := e \rangle$ for the assignment of variable x by the expression e, and $\langle e? \rangle$ for the conditional jump guarded by the condition "e is true".

Interprocedural CFG. The set of intraprocedural CFG's (containing procedure invocations) allows to define the *interprocedural CFG*

$$G = (Ctrl = \bigcup_i Ctrl_i, I)$$

where $I : Ctrl \times Ctrl \to \mathrm{Instr}^{\mathsf{inter}}$ and where $\mathrm{Instr}^{\mathsf{inter}}$ is the set of interprocedural instructions:

$$\begin{array}{lll} \mathrm{Instr}^{\mathsf{inter}} & ::= \langle R \rangle & \text{intraprocedural instruction, with } R \subseteq (GEnv \times LEnv)^2 \\ & \mid \ \langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle & \text{procedure calls, where the } \mathbf{x}^{(k)}\text{'s and } \mathbf{y}^{(l)}\text{'s are variables} \\ & \mid \ \langle \mathtt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle & \text{procedure returns, where the } \mathbf{x}^{(k)}\text{'s and } \mathbf{y}^{(l)}\text{'s are variables} \end{array}$$

I is defined as the "union" $\bigcup_i I_i$, where furthermore each edge labelled by a procedure invocation is removed and replaced by two edges usually called *call-to-start* and *exit-to-return* edges:

$$\frac{I_i(c, c') \neq \langle \mathbf{y} := P_j(\mathbf{x}) \rangle}{I(c, c') = I_i(c, c')} \qquad \frac{I_i(c, c') = \langle \mathbf{y} := P_j(\mathbf{x}) \rangle}{\begin{array}{c} I(c, s_j) = \langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle \\ I(e_j, c') = \langle \mathtt{ret}\, \mathbf{y} := P_j(\mathbf{x}) \rangle \end{array}}$$

As an example, Figure 1 depicts the interprocedural CFG of the recursive implementation of the factorial function f.

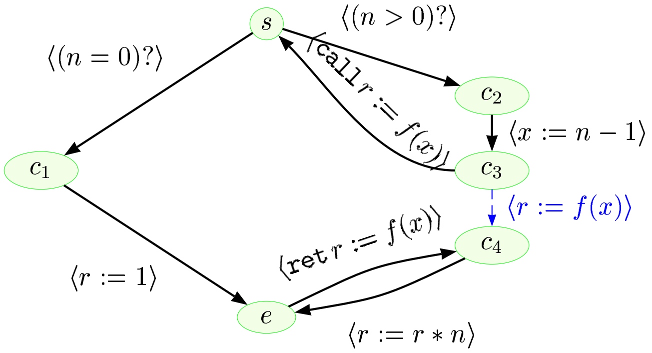

Figure 1: Interprocedural CFG for the Factorial Function

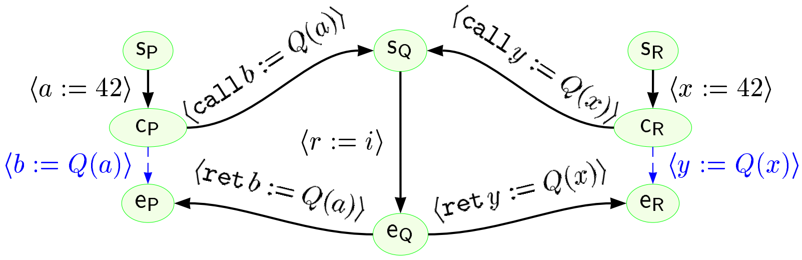

Figure 2: Interprocedural CFG of Prog

Notations. For $c \in \mathit{Ctrl}$, $\mathsf{proc}(c)$ will denote the index j such that $c \in \mathit{Ctrl}_j$. For any interprocedural transition $\tau = c \xrightarrow{\langle \mathbf{y}:=P_j(\mathbf{x})\rangle} c'$, we define $\mathsf{ret}(c) \stackrel{\text{def}}{=} c'$ and its inverse $\mathsf{call}(c') \stackrel{\text{def}}{=} c$. We call a control point $c \in \mathit{Ctrl}$ with an outgoing edge labelled with a procedure call (i.e., $\exists c' : I(c,c') = \langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x})\rangle$) a *call-site* (to procedure j).

Example 2 *Consider the following simplistic program consisting of global variables x and y and three procedures P, Q and R.*

<table>
<tr><td colspan="3" align="center"><code>var x,y : int;</code></td></tr>
<tr>
<td><pre>proc P() : () is
var a,b : int;
begin (s_P)
   a := 42;
   (c_P)
   b := Q(a);
end (e_P)</pre></td>
<td><pre>proc Q(i : int) : (r : int) is
begin (s_Q)
   r := i;
end (e_Q)</pre></td>
<td><pre>proc R() : () is
begin (s_R)
   x := 42;
   (c_R)
   y := Q(x);
end (e_R)</pre></td>
</tr>
</table>

Procedure Q implements the identify function, returning its argument i without modification as its result r. The parameterless procedures P and R call Q, P with the local variable a, and R with the global variable x.

Figure 2 depicts the interprocedural CFG of this program, where the dashed lines represent the transitions of the intraprocedural CFG that have been removed. The intraprocedural CFG's of Prog *are the three "vertical components" of Figure 2.*

2.2 Operational Semantics

The operational semantics of programs is given by a *transition system* $(\mathit{State}, \rightarrow)$.

v	$\in$ *Value*	:	values of expressions and variables
ϵ_i	$\in LEnv_i = LVar_i \to Value$	:	local environments for procedure/function P_i
ϵ	$\in LEnv = \bigcup_i LEnv_i$	:	local environments for any procedure/function
σ	$\in GEnv = GVar \to Value$	:	global environments
$\langle c, \epsilon\rangle$	$\in Act = Ctrl \times LEnv$	:	activation record
Γ	$\in Act^*$	:	stacks (sequences) of activation records[8]
$\langle \sigma, \Gamma\rangle$	$\in State = GEnv \times Act^+$	:	program states

Table 2: Semantic domains

$$\frac{I(c,c') = \langle R\rangle \qquad R(\langle\sigma,\epsilon\rangle,\langle\sigma',\epsilon'\rangle)}{\big\langle \sigma,\ \Gamma\cdot\langle c,\epsilon\rangle\big\rangle \to \big\langle \sigma',\ \Gamma\cdot\langle c',\epsilon'\rangle\big\rangle} \quad \text{(Intra)}$$

$$\frac{I(c,s_j) = \langle \mathtt{call}\,\mathbf{y} := P_j(\mathbf{x})\rangle \qquad \epsilon_j(\mathbf{fp}_j^{(k)}) = (\sigma\oplus\epsilon)(\mathbf{x}^{(k)})}{\big\langle \sigma,\ \Gamma\cdot\langle c,\epsilon\rangle\big\rangle \to \big\langle \sigma,\ \Gamma\cdot\langle c,\epsilon\rangle\cdot\langle s_j,\epsilon_j\rangle\big\rangle} \quad \text{(Call)}$$

$$\frac{\begin{array}{c} I(e_j,c) = \langle \mathtt{ret}\,\mathbf{y} := P_j(\mathbf{x})\rangle \\ \sigma' = \sigma[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \in GVar] \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \in LVar] \end{array}}{\big\langle \sigma,\ \Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle\cdot\langle e_j,\epsilon_j\rangle\big\rangle \to \big\langle \sigma',\ \Gamma\cdot\langle c,\epsilon'\rangle\big\rangle} \quad \text{(Return)}$$

Figure 3: SOS Rules Defining the Transition Relation $\to$ of the Operational Semantics

The *states* of a program are pairs $\langle\sigma,\Gamma\rangle$ composed of the global environment σ and the stack[7] $\Gamma = \langle c_0,\epsilon_0\rangle\cdot\ldots\cdot\langle c_n,\epsilon_n\rangle$ of activation records of the calling functions, with on its top the current activation record $\langle c_n,\epsilon_n\rangle$ (i.e., the current control point c_n and the current local environment ϵ_n). We call *tail* of Γ the stack Γ without its top element, i.e., the stack $\langle c_0,\epsilon_0\rangle\cdot\ldots\cdot\langle c_{n-1},\epsilon_{n-1}\rangle$ Environments map variables to values. They can be concatenated with the $\oplus$ operator, and updated with the notation $\sigma[x\mapsto v]$. The semantic domains are summarised in Table 2.

The transition relation $\to\ \subseteq State \times State$ is defined (in SOS-style) by the set of inference rules shown in Figure 3. An important remark is that we do not use a special value to denote the fact that the value of a variable is undefined. Instead, as long as a variable hasn't been specified, the variable is supposed to hold undeterministically any value in its domain. This choice appears in the rule (Call), where the local variables which are not formal parameters can contain any value at the entry point fo the procedure. $\to^*$ will denote, as usual, the reflexive and transitive closure of $\to$.

Example 3 *Consider the program* Prog *of Example 2. Starting execution from the state* $\big\langle ?,\ \langle \mathsf{s_P}, ?\rangle\big\rangle$, *where ? signifies that there are no constraints on the values of the variables, we get the following execution path (also shown in Figure 4):*

$$\big\langle ?,\ \langle \mathsf{s_P}, ?\rangle\big\rangle \to \big\langle ?,\ \langle \mathsf{c_P}, a{=}42\rangle\big\rangle \to \big\langle ?,\ \langle \mathsf{c_P}, a{=}42\rangle\cdot\langle \mathsf{s_Q}, i{=}42\rangle\big\rangle \to \big\langle ?,\ \langle \mathsf{c_P}, a{=}42\rangle\cdot\langle \mathsf{e_Q}, i{=}r{=}42\rangle\big\rangle \to \big\langle ?,\ \langle \mathsf{e_P}, a{=}b{=}42\rangle\big\rangle$$

Another possible execution (starting from state $\big\langle ?,\ \langle \mathsf{s_R}\rangle\big\rangle$*) is:*

[7]Clearly, in this context, the indexes of the local environments ϵ_i do not indicate the number of the procedure, but just the position in the stack.

[8]For any set E, $E^+ \stackrel{\text{def}}{=} \bigcup_{i\geq 0} E^i$.

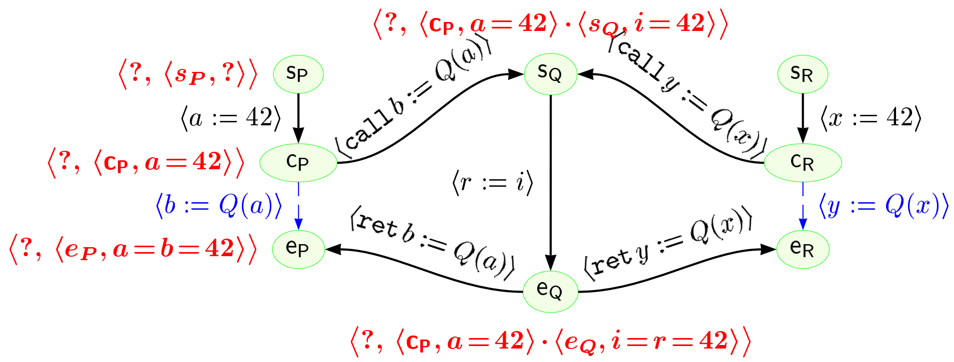

Figure 4: Execution-Path of Prog, starting from $\mathsf{s_P}$

$$\langle ?, \langle \mathsf{s_R} \rangle \rangle \to \langle x{=}42, \langle \mathsf{c_R} \rangle \rangle \to \langle x{=}42, \langle \mathsf{c_R} \rangle \cdot \langle \mathsf{s_Q}, i{=}42 \rangle \rangle \to \langle x{=}42, \langle \mathsf{c_R} \rangle \cdot \langle \mathsf{e_Q}, i{=}r{=}42 \rangle \rangle \to \langle x{=}y{=}42, \langle \mathsf{e_R} \rangle \rangle$$

2.3 Standard Forward Collecting Semantics

The forward collecting semantics of a program defines the meaning of a program by its set of reachable states. It is the natural semantics for expressing and verifying invariance properties. It is derived from its trace semantics by collecting the states belonging to traces (executions) of the program.

Let $\mathit{reach} : \wp(\mathit{State}) \to \wp(\mathit{State})$ be the function returning the set of states reachable from a set of *initial states* given as argument. *reach* is defined as

$$\mathit{reach}(X_0) \stackrel{\text{def}}{=} \{q \mid \exists q^{\mathit{init}} \in X_0 \;.\; q^{\mathit{init}} \to^* q\}$$

It is also the least fix-point solution of

$$X = X_0 \cup \mathit{post}(X)$$

where $\mathit{post} : \begin{array}{rcl} \wp(\mathit{State}) & \to & \wp(\mathit{State}) \\ X & \mapsto & \{q' \mid \exists q \in X : q \to q'\} \end{array}$ is the forward transfer function associated to the program. We will actually decompose it into operators associated to the transitions of the interprocedural flow graph G:

$$\mathit{post}(X) = \bigcup_{(c,c') \in \mathit{Ctrl} \times \mathit{Ctrl}} \mathit{post}(c \xrightarrow{I(c,c')} c')(X)$$

We have, by straightforward application of the inferences rules defining the operational semantics, the equations given in Table 3.

In the sequel, we call $F[X_0]$ the function defined by

$$F[X_0](X) \stackrel{\text{def}}{=} X_0 \cup \mathit{post}(X) \tag{4}$$

where $X_0, X \subseteq S$. Since $F[X_0]$ is obviously monotone and continuous, we have by the fix-point theorem of Kleene that we can compute the forward collecting semantics by an iterated application of $F[X_0]$ starting from $\bot$:

$$\mathit{reach}(X_0) = \mathrm{lfp}(F[X_0]) = \bigcup_{n \geq 0} (F[X_0])^n(\bot)$$

Example 4 *Consider the program of Example 2. We have* $\mathit{reach}\big(\{\langle ?, \langle \mathsf{s_P}, ? \rangle \rangle\}\big)$ *depicted below (cf. Example 3 and Figure 4):*

$$post(c \xrightarrow{\langle R\rangle} c')(X) = \left\{ \langle \sigma', \Gamma\cdot\langle c', \epsilon'\rangle\rangle \;\middle|\; \begin{array}{l} \langle \sigma, \Gamma\cdot\langle c,\epsilon\rangle\rangle \in X \\ R(\langle\sigma,\epsilon\rangle, \langle\sigma',\epsilon'\rangle) \end{array} \right\} \tag{3a}$$

$$post(c \xrightarrow{\langle \texttt{call } \mathbf{y}:=P_j(\mathbf{x})\rangle} c')(X) = \left\{ \langle \sigma, \Gamma\cdot\langle c,\epsilon\rangle\cdot\langle c', \epsilon'\rangle\rangle \;\middle|\; \begin{array}{l} \langle \sigma, \Gamma\cdot\langle c,\epsilon\rangle\rangle \in X \\ \epsilon'(\mathbf{fp}_j^{(k)}) = (\sigma\oplus\epsilon)(\mathbf{x}^{(k)}) \end{array} \right\} \tag{3b}$$

$$post(c \xrightarrow{\langle \texttt{ret } \mathbf{y}:=P_j(\mathbf{x})\rangle} c')(X) = \left\{ \langle \sigma', \Gamma\cdot\langle c', \epsilon'\rangle\rangle \;\middle|\; \begin{array}{l} \langle \sigma, \Gamma\cdot\langle \mathsf{call}(c'),\epsilon\rangle\cdot\langle c,\epsilon_j\rangle\rangle \in X \\ \sigma' = \sigma[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \in GVar] \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \notin GVar] \end{array} \right\} \tag{3c}$$

Table 3: Concrete Postcondition Operators

`proc P() : () is`
`var a,b : int;`
`begin` ($\mathsf{s_P}$) $\{\langle ?, \langle s_P, ?\rangle\rangle\}$
`a := 42;`
($\mathsf{c_P}$) $\{\langle ?, \langle \mathsf{c_P}, a{=}42\rangle\rangle\}$
`b := Q(a);`
`end` ($\mathsf{e_P}$) $\{\langle ?, \langle e_P, a{=}b{=}42\rangle\rangle\}$
	`proc Q(i : int) : (r : int) is`
	`begin`
	($\mathsf{s_Q}$) $\{\langle ?, \langle \mathsf{c_P}, a{=}42\rangle\cdot\langle s_Q, i{=}42\rangle\rangle\}$
	`r := i;`
	`end`
	($\mathsf{e_Q}$) $\{\langle ?, \langle \mathsf{c_P}, a{=}42\rangle\cdot\langle e_Q, i{=}r{=}42\rangle\rangle\}$
		`proc R() : () is`
		`begin` ($\mathsf{s_R}$) $\emptyset$
		`x := 42;`
		($\mathsf{c_R}$) $\emptyset$
		`y := Q(x);`
		`end` ($\mathsf{e_R}$) $\emptyset$

2.4 Standard Backward Collecting Semantics

The collecting semantics can also be considered backward, in order to yield the set of states X from which a given set of *final states* X_0 is reachable. In this case we call X the set of co-reachable states of X_0. Let *coreach* : $\wp(State) \to \wp(State)$ be the function describing the set of co-reachable states. $coreach(X_0)$ is the least solution of the equation $X = X_0 \cup pre(X)$ where *pre* is defined as the inverse of *post*, i.e.,

$$pre(X) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } X = \emptyset \\ post^{-1}(X) & \text{otherwise} \end{cases} \tag{5}$$

We define the monotone and continuous function

$$G[X_0](X) = X_0 \cup pre(X)$$

and as for *reach*, we can compute iteratively the fixpoint

$$coreach(X_0) = \mathrm{lfp}(G[X_0]) = \bigcup_{n\geq 0} (G[X_0])^n(\bot)$$

.

Example 5 *Consider the program of Example 2. We have coreach* $\left(\{\langle x{=}y{=}42, \langle e_R\rangle\rangle\}\right)$ *depicted below (cf. also Example 3) :*

`proc P() : () is`
`var a,b : int;`
`begin` ($\mathsf{s_P}$) $\emptyset$
`a := 42;`
($\mathsf{c_P}$) $\emptyset$
`b := Q(a);`
`end` ($\mathsf{e_P}$) $\emptyset$
	`proc Q(i : int) : (r : int) is`
	`begin`
	($\mathsf{s_Q}$) $\{\langle x{=}42, \langle \mathsf{c_P}, a{=}42\rangle\cdot\langle \mathsf{s_Q}, i{=}42\rangle\}\rangle$
	`r := i;`
	`end`
	($\mathsf{e_Q}$) $\{\langle x{=}42, \langle \mathsf{c_P}, a{=}42\rangle\cdot\langle \mathsf{e_Q}, i{=}r{=}42\rangle\rangle\}$
		`proc R() : () is`
		`begin` ($\mathsf{s_R}$) $\langle ?, \langle \mathsf{s_R}\rangle\rangle$
		`x := 42;`
		($\mathsf{c_R}$) $\langle x{=}42, \langle \mathsf{c_R}\rangle\rangle$
		`y := Q(x);`
		`end` ($\mathsf{e_R}$) $\langle x{=}y{=}42, \langle \mathsf{e_R}\rangle\rangle$

2.5 Properties of the standard semantics

We gather here some properties of the standard semantics (in the operational or the collecting versions) that will be useful to prove optimality results w.r.t. the precision of the abstract semantics we will present later on in Sections 3 and 4.

2.5.1 Properties of Executions

First, observing that the inference rules defining the standard operational semantics need only the top *two* activation records, it is easy to see that we have the following property: for any states $\langle\sigma,\Gamma\rangle$ and $\langle\sigma',\Gamma'\rangle$ and for any stack $\Upsilon \in Act^*$,

$$\langle\sigma,\Gamma\rangle \to \langle\sigma',\Gamma'\rangle \implies \langle\sigma,\Upsilon\cdot\Gamma\rangle \to \langle\sigma',\Upsilon\cdot\Gamma'\rangle \tag{6}$$

Another intuitive property is that the execution of a procedure depends only on the global environment and the value of actual parameters at the call points of the procedure. This is formalised below;

Proposition 1 (determinism of procedures) *Let $q_1, q_2 \in State$ such that*

$$\begin{cases} q_1 = \big\langle\sigma,\, \Gamma_1\cdot\langle c_1,\epsilon_1\rangle\big\rangle \ ,\ q_2 = \big\langle\sigma,\, \Gamma_2\cdot\langle c_2,\epsilon_2\rangle\big\rangle \ ,\ \textit{with}\ \Gamma_1,\Gamma_2\in Act^* \\ I(c_1,s_j) = \langle \mathbf{y}_1 := \texttt{call}\ P_j(\mathbf{x}_1)\rangle \\ I(c_2,s_j) = \langle \mathbf{y}_2 := \texttt{call}\ P_j(\mathbf{x}_2)\rangle \\ (\sigma\oplus\epsilon_1)(\mathbf{x}_1^{(k)}) = (\sigma\oplus\epsilon_2)(\mathbf{x}_2^{(k)}) \end{cases}$$

$$\begin{array}{ll} \textit{Then} & q_1 \to \big\langle\sigma_1,\, \Gamma_1\cdot\langle c_1,\epsilon_1\rangle\cdot\Upsilon_1\big\rangle \to \ldots \to \big\langle\sigma_n,\, \Gamma_1\cdot\langle c_1,\epsilon_1\rangle\cdot\Upsilon_n\big\rangle \\ \textit{implies} & q_2 \to \big\langle\sigma_1,\, \Gamma_2\cdot\langle c_2,\epsilon_2\rangle\cdot\Upsilon_1\big\rangle \to \ldots \to \big\langle\sigma_n,\, \Gamma_2\cdot\langle c_2,\epsilon_2\rangle\cdot\Upsilon_n\big\rangle \end{array}$$

for any $n \geq 1$ and $\Upsilon_i \in Act^+$.

In other words, the effect of procedures is deterministic in the forward collecting semantics and depends only on the global environment and the actual parameters on the top activation record that are given to the procedure. If at a call point two states agree on these, they can reach the same activation records in the callee and above, if the callee in turn calls other procedures.

Proof. We prove Proposition 1 by induction on the length n of the execution.

Base case. The base case is $n = 1$. For $i = 1,2$, we have $q_i \to \big\langle\sigma,\, \Gamma_i\cdot\langle c_i,\epsilon_i\rangle\cdot\langle s_j,\epsilon'\rangle\big\rangle$ for any ϵ' such that $\epsilon'(\mathbf{fp}_j^{(k)}) = (\sigma\oplus\epsilon_i)(\mathbf{x}_i^{(k)})$. As $\sigma\oplus\epsilon_1$ and $\sigma\oplus\epsilon_2$ agree on actual parameters $\mathbf{x}_1^{(k)}$ and $\mathbf{x}_2^{(k)}$, the property holds.

Induction case. Suppose we have

$$q_1 \to \big\langle\sigma_1,\, \Gamma_1\cdot\langle c_1,\epsilon_1\rangle\cdot\Upsilon_1\big\rangle \to \ldots \to \underbrace{\big\langle\sigma_{n-1},\, \Gamma_1\cdot\langle c_1,\epsilon_1\rangle\cdot\Upsilon_{n-1}\big\rangle}_{q_1'} \to \underbrace{\big\langle\sigma_n,\, \Gamma_1\cdot\langle c_1,\epsilon_1\rangle\cdot\Upsilon_n\big\rangle}_{q_1''}$$

with $n \geq 2$ and Υ_i of length at least 1. By induction, we have

$$q_2 \to \big\langle\sigma_1,\, \Gamma_2\cdot\langle c_2,\epsilon_2\rangle\cdot\Upsilon_1\big\rangle \to \ldots \to \underbrace{\big\langle\sigma_{n-1},\, \Gamma_2\cdot\langle c_2,\epsilon_2\rangle\cdot\Upsilon_{n-1}\big\rangle}_{q_2'}$$

Now a single transition depends at most on the two top activation records (see also the motivation of (6)). If the transition $q_1' \to q_1''$ is a procedure return, then the length of Υ_{n-1} is at least 2, because the length of Υ_n is at least 1. So the transition $\big\langle\sigma_{n-1},\, \Upsilon_{n-1}\big\rangle \to \big\langle\sigma_n,\, \Upsilon_n\big\rangle$ is possible, hence $q_2' \to \big\langle\sigma_n,\, \Gamma_2\cdot\langle c_2,\epsilon_2\rangle\cdot\Upsilon_n\big\rangle$. If the transition $q_1' \to q_1''$ is not a procedure return, the transition depends only on the top activation record of Υ_{n-1} and we have again $\big\langle\sigma_{n-1},\, \Upsilon_{n-1}\big\rangle \to \big\langle\sigma_n,\, \Upsilon_n\big\rangle$, so we conclude the same way.

□

2.5.2 Properties of the Collecting Semantics

In the forward collecting semantics, it is possible that a state $q = \langle \sigma, \Gamma\cdot\langle c,\epsilon\rangle\rangle$ is reachable, but that no state of the form $\langle \sigma', \Gamma\rangle$ is reachable. This can happen for instance when initial states have stacks with more than one activation record and the state q is reached from initial states by the mean of procedure returns, instead of procedure calls. We capture the absence of this phenomenon by the following notion of prefix-closure for sets of states.

Definition 1 (Prefix-closed set of states) *A set $X \subseteq State$ is said to be* prefix-closed *whenever $\langle \sigma,\ \Gamma\cdot\langle c,\epsilon\rangle\rangle \in X$ (with $\Gamma \in Act^+$) implies that there exists an execution*

$$\langle \sigma_n,\ \Gamma\rangle \to \langle \sigma_{n-1},\ \Gamma\cdot\Upsilon_{n-1}\rangle \to \ldots \to \langle \sigma_0,\ \Gamma\cdot\Upsilon_0\rangle \to \langle \sigma,\ \Gamma\cdot\langle c,\epsilon\rangle\rangle \tag{7}$$

with $n \geq 0$, $\langle \sigma_i,\ \Gamma\cdot\Upsilon_i\rangle \in X$, and $\Upsilon_i \in Act^+$.

A set of states X is prefixed closed if, for any state in $q = \langle \sigma, \Gamma\cdot\langle c,\epsilon\rangle\rangle \in X$, if the top activation record is popped from its stack, there exists a state $q' = \langle \sigma, \Gamma\rangle \in X$ with the popped stack in X from which q is reachable by an execution belonging to X, i.e., $q' \to^* q$ such that all intermediate states also belong to X. We have the following quite obvious property:

Proposition 2 *Let $X \in \wp(State)$ a prefix-closed set. Then $X \cup post(X)$ and $reach(X)$ are prefix-closed.*

Proof. Let $X \in \wp(State)$ which is prefix-closed, and let $q \in X$. We just to show that for any q' such that $q \to q'$, $X \cup \{q'\}$ is prefix-closed, which implies Proposition 2. We distinguish the three types of transitions.

$q \xrightarrow{\langle R\rangle} q'$**:** Obvious.

$q \xrightarrow{\langle \mathtt{y{:=}call}\ P_j(\mathtt{x})\rangle} q'$**:** Obvious (we have (7) with $n = 0$).

$q \xrightarrow{\langle \mathtt{y{:=}ret}\ P_j(\mathtt{x})\rangle} q'$**:** q and q' can be written as $q = \langle \sigma,\ \Gamma\cdot\langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle\rangle$ and $q' = \langle \sigma',\ \Gamma\cdot\langle c'',\epsilon''\rangle\rangle$. If Γ is empty, $X \cup \{q'\}$ is obviously prefix-closed. Otherwise, since $q \in X$ and X is prefix-closed, there exists $q'' = \langle \sigma'',\ \Gamma\cdot\langle c,\epsilon\rangle\rangle \in X$ with an execution $q'' \to^* q$ satisfying (7). As $q'' \in X$, there exists $q''' = \langle \sigma''',\ \Gamma\rangle \in X$ with an execution $q''' \to^* q''$ satisfying (7). By combining these executions with $q \to q'$, we get an execution $q''' = \langle \sigma''',\ \Gamma\rangle \to^* q' = \langle \sigma',\ \Gamma\cdot\langle c'',\epsilon''\rangle\rangle$ satisfying (7). □

It might be difficult to ensure that a set X is prefix-closed, because it involves the transition relation, and we assume it to be a not computable relation[9]. Here is a trivial sufficient condition:

Proposition 3 *Let $X \in \wp(State)$. If $X \in \wp(GEnv \times Act)$, then X is prefix-closed.*

Indeed, if states in X have only one-element stacks, the condition of Definition 1 are trivially satisfied. An easy but interesting property is also the following one:

Proposition 4 *Let $X \in \wp(State)$ be prefix-closed, and let $X_0 = X \cap \wp(GEnv \times Act)$. Then $X \subseteq reach(X_0)$ and $reach(X) = reach(X_0)$.*

This means that for reachability analysis, it is not more restrictive to consider as initial states one-element stack states instead of prefix-closed sets of states.

[9]Remember that our goal is to design a *simpler* abstract semantics for the standard semantics. So we do not want to rely on the standard semantics for applying the previous proposition.

2.5.3 Properties Induced by Parameter Passing Convention

We assumed that formal parameters are read-only variables. This assumption allows us to get a necessary condition for a state $q = \langle \sigma, \Gamma \cdot c, \epsilon \rangle$ to lead to $q' = \langle \sigma', \Gamma \cdot \langle c, \epsilon \rangle \cdot \langle c', \epsilon' \rangle \rangle$, when c is a call site to procedure $\mathsf{proc}(c')$. Indeed, if $q \to^* q'$, the values of actual parameters in $\sigma \oplus \epsilon$ should match those of the formal parameters in ϵ'.

Definition 2 (valid calling context) *Given a* context $\langle \sigma, \langle c, \epsilon \rangle \rangle \in GEnv \times Act$ *and an activation record* $\langle c', \epsilon' \rangle \in Act$, $\langle \sigma, \langle c, \epsilon \rangle \rangle$ *is a* valid calling context *for* $\langle c', \epsilon' \rangle$ *if,*

1. *c is a call site to a procedure P_j:*
   $\exists j : c' = s_j \ \wedge \ I(c, s_j) = \langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$, *and*
2. *equality between actual and formal parameters holds:*
   $\forall k : (\sigma \oplus \epsilon)(\mathbf{x}^{(k)}) = \epsilon'(\mathbf{fp}_j^{(k)})$.

In the sequel, we will use as a short-hand that a context is *valid* for an activation record. The following property formalises the intuition of the previous definition.

Proposition 5 *For any stack $\Gamma \in Act^*$, global environments σ, σ', and activation records $\langle c, \epsilon \rangle$, $\langle c', \epsilon' \rangle$ and $\langle c'', \epsilon'' \rangle$,*

$$\langle \sigma, \Gamma \cdot \langle c, \epsilon \rangle \rangle \to \langle \sigma, \Gamma \cdot \langle c, \epsilon \rangle \cdot \langle c', \epsilon' \rangle \rangle \to^* \langle \sigma', \Gamma \cdot \langle c, \epsilon \rangle \cdot \langle c'', \epsilon'' \rangle \rangle$$

implies that $\langle \sigma, \langle c, \epsilon \rangle \rangle$ *is* valid *for* $\langle c', \epsilon' \rangle$ *and* $\langle c'', \epsilon'' \rangle$.

If we do not want to involve the global environment in the constraints induced by the parameter passing convention, we can weaken Definition 2 as follows:

Definition 3 (valid calling activation record) $\langle c, \epsilon \rangle$ *is a* valid calling activation record *for* $\langle c', \epsilon' \rangle$ *if,*

1. *c is a call site to a procedure P_j:*
   $\exists j : c' = s_j \ \wedge \ I(c, s_j) = \langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle$, *and*
2. *equality between local actual parameters and corresponding formal parameters holds:*
   $\forall \mathbf{x}^{(k)} \in LVar : \epsilon(\mathbf{x}^{(k)}) = \epsilon'(\mathbf{fp}_j^{(k)})$.

We will actually focus only on states which might be reachable from one-element stacks (with only one activation record). We use then the previous property for defining *consistent stacks* (and states).

Definition 4 (consistent stack and state) *A stack* $\Gamma = \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle$ *is* consistent *if, for any* $0 \leq i < n$, $\langle c_i, \epsilon_i \rangle$ *is a* valid calling activation record *for* $\langle c_{i+1}, \epsilon_{i+1} \rangle$. *A state* $q = \langle \sigma, \Gamma \rangle$ *is* consistent *if its stack Γ is consistent.*

From now on, we focus on consistent states and we restrict *State* to its consistent subset. In particular, we implicitly intersect (for any $X \subseteq S$) $pre(X)$, which was defined as $post^{-1}(X)$, with the sets of consistent states.

3 A Simple Stack Abstraction for Functional Programs

We propose here an abstract semantics for programs without global variables, to which standard abstraction and representation techniques can be applied. This abstract semantics is actually defined by an abstraction of the program's stacks into simpler values. This approach belongs to the *operational approach* (according to the definition in the introduction), which contrasts with other solutions suggested for similar programs. We are indeed still able to talk about stacks in the abstract semantics. We give correctness results for forward and backward analyses, as well as an optimality result w.r.t. the precision of the forward analysis.

3.1 Stack Abstraction

In the absence of global variables, we have $State = Act^+ = (Ctrl \times LEnv)^+$, implicitly restricted to its subset of consistent states. Notice that Definitions 2 and 3 become equivalent.

We decide here to forget all about sequences of call sites in the stack, roughly keeping information only on activation records. This abstraction will be refined in Section 4. We can nevertheless get accurate results in such a setting, as it has been observed in a different context [KS92]. Basically, our abstraction consists in abstracting a state $q = \langle c_0, \epsilon_0\rangle \dots \langle c_n, \epsilon_n\rangle$ by the pair of sets $\big\langle \{\langle c_n, \epsilon_n\rangle\},\ \{\langle c_i, \epsilon_i\rangle \mid i < n\}\big\rangle$.

Abstract Domain. An abstract state $Y = \langle \overline{Y}, \widetilde{Y}\rangle$ is composed of two sets of activation records. The first set $\overline{Y}$ represents top activation records, whereas the second set $\widetilde{Y}$ is the collapse of all the activation records belonging to tails of stacks.

This leads to the following definition of the abstract domain:

$$A_f \stackrel{\text{def}}{=} \wp(Act) \times \wp(Act) \tag{8}$$

which is equipped with the standard lattice structure $A_f(\sqsubseteq, \sqcup, \sqcap, \top, \bot)$ of a Cartesian product of lattices, defined by the order:

$$\forall Y_1, Y_2 \in A_f : \langle \overline{Y_1}, \widetilde{Y_1}\rangle \sqsubseteq \langle \overline{Y_2}, \widetilde{Y_2}\rangle \iff \overline{Y_1} \subseteq \overline{Y_2} \wedge \widetilde{Y_1} \subseteq \widetilde{Y_2} \tag{9}$$

We implicitly collapse all elements of the form $\langle x, \emptyset\rangle$ and $\langle \emptyset, x\rangle$ with $\bot$. For a value $Y \in A_f$, we will always decompose it as $\langle \overline{Y}, \widetilde{Y}\rangle$. We will do the same for functions returning abstract states.

Abstraction and Concretisation Functions. A_f is connected to the state space $\wp(State) = \wp(Act^+)$ by the Galois connection $\wp(State) \underset{\alpha_f}{\overset{\gamma_f}{\leftrightarrows}} A_f$, where the abstraction function $\alpha_f = \langle \overline{\alpha_f}, \widetilde{\alpha_f}\rangle$ is defined by:

$$\begin{array}{rcrcl} \forall q = \langle c_0, \epsilon_0\rangle \dots \langle c_n, \epsilon_n\rangle \in State & : & \alpha_f(\{q\}) & = & \left\langle \begin{array}{c} \{\langle c_n, \epsilon_n\rangle\}, \\ \{\langle c_i, \epsilon_i\rangle \mid 0 \le i < n\} \end{array} \right\rangle \\ \forall X \in \wp(State) & : & \alpha_f(X) & = & \bigsqcup_{q \in X} \alpha_f(\{q\}) \end{array} \tag{10}$$

and the concretisation function γ_f, for any $Y = \langle \overline{Y}, \widetilde{Y}\rangle \in A_f$, by:

$$q = \langle c_0, \epsilon_0\rangle \dots \langle c_n, \epsilon_n\rangle \in \gamma_f(Y) \iff \left\{ \begin{array}{l} \langle c_n, \epsilon_n\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_i, \epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0, \epsilon_0\rangle \dots \langle c_n, \epsilon_n\rangle \text{ is a consistent stack} \end{array} \right. \tag{11}$$

As we told, $\overline{\alpha_f}$ just gathers the top activation records of sets of stacks, whereas $\widetilde{\alpha_f}$ collects all the other activation records. To rebuild a stack from a an abstract state, the concretisation function γ_f uses the notion of consistent stacks. Notice that $\langle c, \epsilon\rangle \in \widetilde{Y}$ implies that c is a call site.

Proposition 6 $\wp(State) \underset{\alpha_f}{\overset{\gamma_f}{\leftrightarrows}} A_f$ *is a Galois connection. It follows that* $\alpha_f \circ \gamma_f \sqsubseteq id$ *and* $\gamma_f \circ \alpha_f \sqsupseteq \mathsf{id}$. [10]

Proof. We defined γ_f as $\gamma_f(Y) = \bigcup\{X \in \wp(State) \mid \alpha_f(X) \sqsubseteq Y\}$, see Proposition 7 of [CC92a]. □

[10] *id* denotes the identify function.

Because of the consistency condition in (11), γ_f is not injective, and two abstract elements may have the same meaning. It would be desirable to obtain a *Galois insertion* [CC92a], with γ_f injective and $\alpha_f \circ \gamma_f = id$. This can be achieved by restricting A_f to *canonical abstract values* verifying $\alpha_f \circ \gamma_f(Y) = Y$. However, the product meet operator in A_f, $Y_1 \sqcap Y_2 \stackrel{\text{def}}{=} \langle \overline{Y_1} \cap \overline{Y_2}, \widetilde{Y_1} \cap \widetilde{Y_2} \rangle$, *is not* the meet operator in the restricted lattice, so we will stay in A_f.

The loss of information of our abstraction is given by the extensive function $\gamma_f \circ \alpha_f$. We can have for instance

$$\begin{array}{rcl} \langle c_0, \epsilon_0 \rangle \cdot \langle c_1, \epsilon'_1 \rangle & \in & (\gamma_f \circ \alpha_f)\Big(\{ \ \langle c_0, \epsilon_0 \rangle \cdot \langle c_1, \epsilon_1 \rangle \ , \ \langle c_1, \epsilon'_1 \rangle \ \} \Big) \\ & \notin & \qquad\quad \{ \ \langle c_0, \epsilon_0 \rangle \cdot \langle c_1, \epsilon_1 \rangle \ , \ \langle c_1, \epsilon'_1 \rangle \ \} \end{array} \tag{12}$$

if $\langle c_0, \epsilon_0 \rangle$ is a valid calling activation record for $\langle c_1, \epsilon'_1 \rangle$ and $\epsilon_1 \neq \epsilon'_1$.

However, it is clear that $\overline{\alpha_f}(X)$ keeps *exact information* on *top* activation records of stacks, which is the only information computed by the functional approaches. This is precisely the reason why we did not collapse completely the stacks but rather separate top activation records from activation records lying below them in the stacks.

3.2 Forward and Backward Analysis

We have still to compute reachable and coreachable states in the abstract semantics. One might think that collapsing tails of stacks of activation records to sets of activation records leads to an important approximation for these computations, even if at the end we just want to obtain information on top activation records. Fortunately, this is not the case thanks to the use of the notion of valid calling activation record.

3.2.1 Abstract Transfer Functions.

Computing sets of reachable and coreachable states in the abstract domain requires the definition of postcondition and precondition operators acting on the abstract domain.

Abstract Postcondition $post^{\mathsf{a}}_f$**.** We define $post^{\mathsf{a}}_f = \bigcup_{c \xrightarrow{I(c,c')} c'} post^{\mathsf{a}}_f(c \xrightarrow{I(c,c')} c')$, where $post^{\mathsf{a}}_f(c \xrightarrow{I(c,c')} c')$ is defined on Table 4.

The interesting case is the procedure return for $\overline{post^{\mathsf{a}}_f}$, *cf.* (13c) and Figure 5, the others being straightforward. How to recover the stack contents before the call at the return point ? This corresponds to the function *combine* introduced [SP81, JM82], the aim of which is to combine the environment of the caller at the call point with the environement of the callee at its exit point. In these papers however, this function is not fully defined. Let us define it in the abstract domain A_f. According to Proposition 5, $\Gamma \cdot \langle \mathsf{call}(c), \epsilon \rangle \cdot \langle e_j, \epsilon_j \rangle$ can be a successor of $\Gamma \cdot \langle \mathsf{call}(c), \epsilon \rangle$ only if $\langle c, \epsilon \rangle$ is a valid calling activation record for $\langle e_j, \epsilon_j \rangle$. So if we have $\langle e_j, \epsilon_j \rangle \in \overline{Y}$, we select as the possible calling activation records all the activation records $\langle \mathsf{call}(c), \epsilon \rangle \in \widetilde{Y}$ that are valid for it. Once this is done, we just have to perform the assignment of actual returning parameters $\mathbf{y}$, and we get the reachable activation record(s) at point c.

Seen from another angle, at the exit point e_j of a procedure P_j, the top of the stack (in $\overline{Y}$) contains a relation $\phi_j(\mathbf{fp}_j, \mathbf{fr}_j)$ between *reachable* formal call parameter and formal return parameters:

$$\phi_j = \{ (\mathbf{x}, \mathbf{y}) \mid \exists \langle e_j, \epsilon \rangle \in \overline{Y} : \mathbf{x} = \epsilon(\mathbf{fp}_j) \wedge \mathbf{y} = \epsilon(\mathbf{fr}_j) \} \tag{15}$$

This relation is the predicate transformer of the procedure specialised on the reachable inputs. Hence, our approach for procedure return can be seen as applying this predicate transformer of the callee to the valid calling activation records that are reachable in $\widetilde{Y}$ at the calling point of the caller.

$$c \xrightarrow{\langle R\rangle} c' \qquad \overline{post^{\mathsf{a}}_f}(\tau)(Y) = \{\langle c',\epsilon'\rangle \mid \langle c,\epsilon\rangle \in \overline{Y} \wedge R(\epsilon,\epsilon')\} \tag{13a}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} s_j \qquad \overline{post^{\mathsf{a}}_f}(\tau)(Y) = \left\{ \langle s_j,\epsilon_j\rangle \;\middle|\; \begin{array}{l} \langle c,\epsilon\rangle \in \overline{Y} \\ \epsilon_j(\mathbf{fp}_j^{(k)}) = \epsilon(\mathbf{x}^{(k)}) \end{array} \right\} \tag{13b}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} c \qquad \overline{post^{\mathsf{a}}_f}(\tau)(Y) = \left\{ \langle c,\epsilon'\rangle \;\middle|\; \begin{array}{l} \langle e_j,\epsilon_j\rangle \in \overline{Y} \\ \langle \mathsf{call}(c),\epsilon\rangle \in \widetilde{Y} \\ \epsilon(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}_j^{(k)}) \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}_j^{(k)})] \end{array} \right\} \tag{13c}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} s_j \qquad \widetilde{post^{\mathsf{a}}_f}(\tau)(Y) = \widetilde{Y} \;\cup\; \{\langle c,\epsilon\rangle \in \overline{Y}\} \tag{13d}$$

$$c \xrightarrow{i} c' \qquad \widetilde{post^{\mathsf{a}}_f}(\tau)(Y) = \widetilde{Y} \quad \text{otherwise} \tag{13e}$$

τ denotes a transition, and $i \in \mathrm{Instr}^{\mathsf{inter}}$ an interprocedural instruction.

Table 4: Equations Defining $post^{\mathsf{a}}_f$

$$c \xrightarrow{\langle R\rangle} c' \qquad \overline{pre^{\mathsf{a}}_f}(\tau)(Y) = \{\langle c,\epsilon\rangle \mid \langle c',\epsilon'\rangle \in \overline{Y} \wedge R(\epsilon,\epsilon')\} \tag{14a}$$

$$c_i \xrightarrow{\langle \mathtt{call}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} s_j \qquad \overline{pre^{\mathsf{a}}_f}(\tau)(Y) = \left\{ \langle c,\epsilon\rangle \;\middle|\; \begin{array}{l} \langle c,\epsilon\rangle \in \widetilde{Y} \\ \langle s_j,\epsilon_j\rangle \in \overline{Y} \\ \epsilon(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}^{(k)}) \end{array} \right\} \tag{14b}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} c \qquad \overline{pre^{\mathsf{a}}_f}(\tau)(Y) = \left\{ \langle e_j,\epsilon_j\rangle \;\middle|\; \begin{array}{ll} \langle c,\epsilon\rangle \in \overline{Y} & \\ & \epsilon_j(\mathbf{fr}_j^{(k)}) = \epsilon(\mathbf{y}^{(k)}) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : & \epsilon_j(\mathbf{fp}_j^{(k)}) = \epsilon(\mathbf{x}^{(k)}) \end{array} \right\} \tag{14c}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} c \qquad \widetilde{pre^{\mathsf{a}}_f}(\tau)(Y) = \widetilde{Y} \;\cup\; \left\{ \langle \mathsf{call}(c),\epsilon\rangle \;\middle|\; \begin{array}{l} \langle c,\epsilon'\rangle \in \overline{Y} \\ \forall z \notin \mathbf{y} : \epsilon(z) = \epsilon'(z) \end{array} \right\} \tag{14d}$$

$$c \xrightarrow{i} c' \qquad \widetilde{pre^{\mathsf{a}}_f}(\tau)(Y) = \widetilde{Y} \quad \text{otherwise} \tag{14e}$$

Table 5: Equations Defining pre^{a}_f

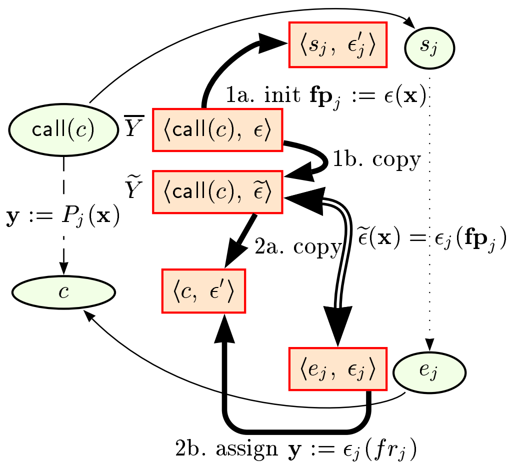

Figure 5: Abstract Postcondition for Procedure Call (1) and Return (2)

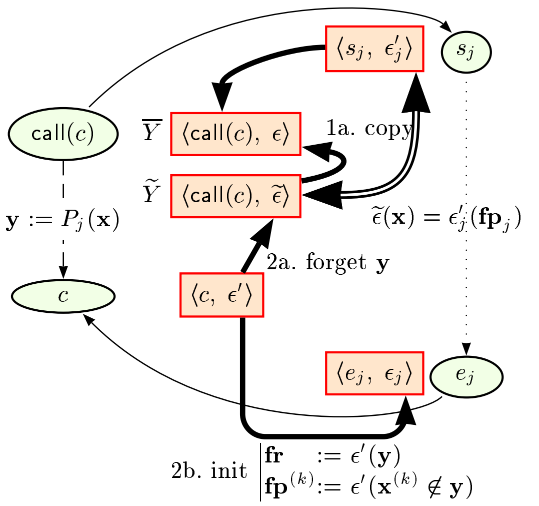

Figure 6: Abstract Precondition for Procedure Call (1) and Return (2)

Abstract Precondition pre_f^{a}. We define $pre_f^{\mathsf{a}} = \bigcup_{c \xrightarrow{I(c,c')} c'} pre_f^{\mathsf{a}}(c \xrightarrow{I(c,c')} c')$, where $pre_f^{\mathsf{a}}(c \xrightarrow{I(c,c')} c')$ is defined on Table 5.

The interesting cases are procedure calls and returns, *cf.* (14b) and (14c), and Fig. 6, the others being straightforward. We use the same kind of technique as for the forward analysis to recover information on stacks, although our abstract states are "flat".

Here, we do not have the dual of equation (15) at the entry point of the procedure, because formal return parameters are not constant during the execution of a procedure.

Correctness of the Abstract Transfer Functions. The abstract operators defined above are correct approximations of their concrete counterpart.

Proposition 7 ($post_f^{\mathsf{a}}, pre_f^{\mathsf{a}}$ are correct approximations of $post, pre$)

$$post_f^{\mathsf{a}} \;\sqsupseteq\; \alpha_f \circ post \circ \gamma_f \tag{16}$$

$$pre_f^{\mathsf{a}} \;\sqsupseteq\; \alpha_f \circ post \circ \gamma_f \tag{17}$$

Proof. Appendix A.1. □

Actually, if we filter the result of $post_f^{\mathsf{a}}$ and pre_f^{a} by applying $\alpha_f \circ \gamma_f$, we could even show the equalities $\alpha_f \circ \gamma_f \circ post_f^{\mathsf{a}} = \alpha_f \circ post \circ \gamma_f$ and $\alpha_f \circ \gamma_f \circ pre_f^{\mathsf{a}} = \alpha_f \circ pre \circ \gamma_f$. We do not really need it, and it is quite cumbersome to show, but this means that $post_f^{\mathsf{a}}$ and pre_f^{a} are *best correct approximations* modulo canonicity of abstract values.

3.2.2 Fixpoint Equations and Correctness

We can transpose the equations defining reachable and coreachable states into the abstract lattice A_f. Let us define for any $Y_0 \in A_f$,

$$\begin{array}{rclrcl} F_f^{\mathsf{a}}[Y_0](Y) & \stackrel{\text{def}}{=} & Y_0 \sqcup post_f^{\mathsf{a}}(Y) & \qquad reach_f^{\mathsf{a}}(Y_0) & \stackrel{\text{def}}{=} & \mathrm{lfp}(F_f^{\mathsf{a}}[Y_0]) \\ G_f^{\mathsf{a}}[Y_0](Y) & \stackrel{\text{def}}{=} & Y_0 \sqcup pre_f^{\mathsf{a}}(Y) & \qquad coreach_f^{\mathsf{a}}(Y_0) & \stackrel{\text{def}}{=} & \mathrm{lfp}(G_f^{\mathsf{a}}[Y_0]) \end{array}$$

We deduce from Proposition 7 and standard abstract interpretation theory that we compute a correct approximation of reachable and coreachable states, for any set of initial or final states.

Theorem 1 (Soundness of the abstract forward and backward analysis)

$$reach_f^{\mathsf{a}} \;\sqsupseteq\; \alpha_f \circ reach \circ \gamma_f \tag{18}$$

$$coreach_f^{\mathsf{a}} \;\sqsupseteq\; \alpha_f \circ coreach \circ \gamma_f \tag{19}$$

Proof. Let us show (18). Let $Y_0 \in A_f$ and $X_0 = \gamma_f(Y_0)$. We first show that $F_f^{\mathsf{a}}[Y_0]$ is a correct approximation of $F[X_0]$:

$$\begin{array}{rcll} F_f^{\mathsf{a}}[Y_0](Y) & = & Y_0 \sqcup post_f^{\mathsf{a}}(Y) & \text{(by definition)} \\ & \sqsupseteq & \alpha_f \circ \gamma_f(Y_0) \sqcup \alpha_f \circ post \circ \gamma_f(Y) & \text{(by Proposition 7)} \\ & = & \alpha_f\big(\gamma_f(Y_0) \sqcup post \circ \gamma_f(Y)\big) & \text{(by definition of } \alpha_f) \\ & = & \alpha_f \circ F[X_0] \circ \gamma_f & \text{(by definition)} \end{array}$$

Now, by using Proposition 27 of [CC92a], we get $reach_f^{\mathsf{a}}(Y_0) = \mathrm{lfp}(F_f^{\mathsf{a}}[Y_0]) \sqsupseteq \alpha_f \circ \mathrm{lfp}(F[X_0]) = \alpha_f \circ reach(X_0)$ and we deduce (18). The proof for (18) is identical. □

3.3 An Optimality Result for the Forward Analysis

The previous result is standard, once the abstract postcondition and precondition operators have been shown to be correct approximations, *cf.* Proposition 7. Could we obtain a better result? Ideally, we would like to have

$$\mathit{reach}^{\mathsf{a}}_f \circ \alpha_f = \alpha_f \circ \mathit{reach} \tag{20}$$

Intuitively, the idea that we build incrementally at the exit point of a procedure its associated predicate transformer (*cf.* Section 3.2) suggests that such an equality holds, maybe under some more assumptions.

We tried first to prove it inductively, by finding a stronger relation between post and $\mathit{post}^{\mathsf{a}}_f$ than the inequality (16) and to prove (20) using conditions similar to those given in [CC92a]. However, all attempts we made in this direction failed. The deep reason is that the propagation of information is not the same in the concrete and the abstract semantics. Consider the forward analysis. In the concrete semantics, the reachable values from the return point of a procedure call depends only on the exit point of the callee. However, in the abstract semantics, it also depends directly on the call point of the procedure call, which propagates stack contents, *cf.* Fig. 5. As a result, information may be propagated faster in the abstract semantics than in the concrete one, and it is not possible to match step by step the propagation of information in the two semantics.

So we will prove a exactness result in a more straightforward way, which is less elegant but which is required by the above observations. We already have $\mathit{reach}^{\mathsf{a}}_f \circ \alpha_f \sqsupseteq \alpha_f \circ \mathit{reach}$ by Theorem 1, so we just have to prove the inverse inclusion. A way to achieve this is to show that $Y = \alpha_f \circ \mathit{reach}(X_0)$ is a pre-fixpoint of $\mathit{post}^{\mathsf{a}}_f$, i.e., that $\mathit{post}^{\mathsf{a}}_f(Y) \sqsubseteq Y$. However, this is not true in general, and some additional assumptions on X_0 are required. In the abstract semantics indeed, procedure returns are approximated, because these transitions depends on two activation records on the stack, and in the abstract lattice we maintain only partial relations between two activation records belonging to the same stack.

Here is the technical proposition.

Proposition 8 *Let $X_0 \in \wp(\mathit{State})$ such that*

1. *X_0 is prefix-closed;*
2. *$q = \langle c_0, \epsilon_0 \rangle \cdot \ldots \cdot \langle c_n, \epsilon_n \rangle \in X_0$ implies that procedure $\mathsf{proc}(c_0)$ is never called by any procedure (this is property of the interprocedural CFG).*

Let $X = \mathit{reach}(X_0)$ and $Y = \alpha_f(X)$. Then $\mathit{post}^{\mathsf{a}}_f(Y) \sqsubseteq Y$.

Intuitively, if X_0 satisfies the two hypotheses, by combining Propositions 1 and 2, we will be able to ensure the existence of paths between the entry and the exit points of a procedure for a given valuation of the formal parameters, when these points are both reachable.

Proof. Let X_0 satisfying the hypothesis, $X = \mathit{reach}(X_0)$ and $Y = \alpha_f(X)$. As X_0 is prefix-closed, so is X by Proposition 2. We will prove that for any transition τ, $\mathit{post}^{\mathsf{a}}_f(\tau)(Y) \sqsubseteq Y$. The difficult case is the procedure return, for which the hypotheses are used.

$\tau = c \xrightarrow{R} c'$**:** According to (13e), $\widetilde{\mathit{post}^{\mathsf{a}}_f}(\tau)(Y) = \widetilde{Y}$. According to (13a) and (Intra), $\overline{\mathit{post}^{\mathsf{a}}_f}(\tau)(Y) = \{\langle c', \epsilon' \rangle \mid \langle c, \epsilon \rangle \in \overline{Y} \wedge R(\epsilon, \epsilon')\} = \{\langle c', \epsilon' \rangle \mid \langle c, \epsilon \rangle \in \overline{Y} \wedge \langle c, \epsilon \rangle \xrightarrow{\tau} \langle c', \epsilon' \rangle\}$. Now, since $Y = \alpha_f(X)$, : $\langle c, \epsilon \rangle \in \overline{Y}$ implies $\exists \Gamma : \Gamma \cdot \langle c, \epsilon \rangle \in X$, and by (6) : $\langle c, \epsilon \rangle \rightarrow \langle c', \epsilon' \rangle$ implies $\Gamma \cdot \langle c, \epsilon \rangle \rightarrow \Gamma \cdot \langle c', \epsilon' \rangle$, so by definition of X: $\Gamma \cdot \langle c', \epsilon' \rangle \in X$. We conclude that $\langle c', \epsilon' \rangle \in \overline{Y}$ by definition of α_f. So $\mathit{post}^{\mathsf{a}}_f(\tau)(Y) \sqsubseteq Y$.

$\tau = c \xrightarrow{\langle \mathtt{call}\, \mathtt{y} := P_j(\mathtt{x}) \rangle} c'$**:** According to (13b) and (13d),

$$\mathit{post}^{\mathsf{a}}_f(\tau)(Y) \quad = \quad \left\langle \begin{array}{c} \{\langle c', \epsilon' \rangle \mid \langle c, \epsilon \rangle \in \overline{Y} \wedge \epsilon'(\mathbf{fp}_j^{(k)}) = \epsilon(\mathbf{x}^{(k)})\}, \\ \widetilde{Y} \cup \{\langle c, \epsilon \rangle \in \overline{Y}\} \end{array} \right\rangle$$

$$= \left\langle \begin{array}{c} \{\langle c',\epsilon'\rangle \mid \langle c,\epsilon\rangle \in \overline{Y} \wedge \langle c,\epsilon\rangle \to \langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle\}, \\ \widetilde{Y} \cup \{\langle c,\epsilon\rangle \in \overline{Y}\} \end{array} \right\rangle$$

Consider $\langle c,\epsilon\rangle \in \overline{Y}$. As before, this implies $\exists\Gamma : \Gamma\cdot\langle c,\epsilon\rangle \in X$, and we have $\Gamma\cdot\langle c,\epsilon\rangle \to \Gamma\cdot\langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle$. So $\Gamma\cdot\langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle \in X$, which implies $\langle c',\epsilon'\rangle \in \overline{Y}$ and $\langle c,\epsilon\rangle \in \widetilde{Y}$.

$\tau = e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c$**:** According to (13e), $\widetilde{post^{\mathsf{a}}_f}(\tau)(Y) = \widetilde{Y}$. According to (13c),

$$\overline{post^{\mathsf{a}}_f}(\tau)(Y) \quad = \quad \left\{ \langle c,\epsilon'\rangle \;\middle|\; \begin{array}{l} \langle e_j,\epsilon_j\rangle \in \overline{Y} \\ \langle \mathsf{call}(c),\epsilon\rangle \in \widetilde{Y} \\ \epsilon(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}_j^{(k)}) \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}_j^{(k)})] \end{array} \right\}$$

- $\langle \mathsf{call}(c),\epsilon\rangle \in \widetilde{Y}$ implies $\exists\Gamma,\Gamma' : \Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle\cdot\Gamma' \in X$. As X is prefix-closed, then $\Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle \in X$.
- $\langle e_j,\epsilon_j\rangle \in \overline{Y}$ implies $\exists\Upsilon : \Upsilon\cdot\langle e_j,\epsilon_j\rangle \in X$. If Υ is empty, then e_j belongs to an initial procedure, i.e., a procedure j such that $\exists\langle c,\epsilon\rangle \in X_0 : \mathsf{proc}(c) = j$. But then, τ cannot exist because by hypothesis procedure j is never called, and there is a contradiction. So Υ contains at least one activation record and we have, with a new Υ, $\Upsilon\cdot\langle c_l,\epsilon_l\rangle\cdot\langle e_j,\epsilon_j\rangle \in X$, with $\langle c_l,\epsilon_l\rangle$ valid for $\langle e_j,\epsilon_j\rangle$. As X is prefix-closed, it follows that $\Upsilon\cdot\langle c_l,\epsilon_l\rangle \in X$ and $\Upsilon\cdot\langle c_l,\epsilon_l\rangle \to^* \Upsilon\cdot\langle c_l,\epsilon_l\rangle\cdot\langle e_j,\epsilon_j\rangle$ as in equation (7) of Definition 1.

$\Upsilon\cdot\langle c_l,\epsilon_l\rangle$ and $\Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle$ satisfy the hypothesis of Proposition 1, so we have $\Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle \to^* \Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle\cdot\langle e_j,\epsilon_j\rangle$. Then we have $\Gamma\cdot\langle \mathsf{call}(c),\epsilon\rangle\cdot\langle e_j,\epsilon_j\rangle \to \Gamma\cdot\langle c,\epsilon'\rangle$ by rule (Return), and by definition of *reach*: $\Gamma\cdot\langle c,\epsilon'\rangle \in X$. So $\langle c,\epsilon'\rangle \in \overline{Y}$ and $\overline{post^{\mathsf{a}}_f}(Y) \subseteq \overline{Y}$.

□

We can now state the following *optimality* theorem, w.r.t. the precision of the forward analysis.

Theorem 2 (Optimality of the forward analysis) *Let $X_0 \in \wp(State)$ such that $q \in X_0$ implies that $q = \langle c,\epsilon\rangle$ and that the procedure* $\mathsf{proc}(c)$ *is never called by any procedure. Then*

$$reach^{\mathsf{a}}_f \circ \alpha_f(X_0) = \alpha_f \circ reach(X_0) \tag{21}$$

This implies that

$$\overline{reach^{\mathsf{a}}_f} \circ \alpha_f(X_0) \quad = \quad \{\langle c,\epsilon\rangle \mid \exists\Gamma : \Gamma\cdot\langle c,\epsilon\rangle \in reach(X_0)\} \tag{22}$$

$$\widetilde{reach^{\mathsf{a}}_f} \circ \alpha_f(X_0) \quad = \quad \{\langle c,\epsilon\rangle \mid \exists\Gamma,\Gamma' : \Gamma\cdot\langle c,\epsilon\rangle\cdot\Gamma' \in reach(X_0)\} \tag{23}$$

Proof. Under the hypotheses, $Y = \alpha_f \circ reach(X_0)$ is a pre-fixpoint of $post^{\mathsf{a}}_f$, according to Proposition 8. $X_0 \subseteq reach(X_0)$, so $\alpha_f(X_0) \sqsubseteq Y$ (α_f is monotone), so $F^{\mathsf{a}}_f[\alpha_f(X_0)](Y) = \alpha_f(X_0) \sqcup post^{\mathsf{a}}_f(Y) \sqsubseteq Y \sqcup post^{\mathsf{a}}_f(Y) \sqsubseteq Y$. Thus Y is a pre-fixpoint of $F^{\mathsf{a}}_f[\alpha_f(X_0)]$ and $reach^{\mathsf{a}}_f \circ \alpha_f(X_0) = \mathrm{lfp}(F^{\mathsf{a}}_f[\alpha_f(X_0)]) \sqsubseteq Y$. From Theorem 1 and (18), we have $reach^{\mathsf{a}}_f \circ \alpha_f(X_0) \sqsupseteq \alpha_f \circ reach \circ \gamma_f \circ \alpha_f(X_0) \sqsupseteq \alpha_f \circ reach(X_0)$. By double inclusion we get (21). □

Whereas our analysis looses information on stack contents, we get exact results if we are just interested in the values that variables can hold at some control points, which is the information of interest for many applications, *as long as* we start from one-element initial stacks and such that the initial procedure is never called.

The two hypotheses seem necessary for proving Proposition 8 and ultimately Theorem 2, as shown by examples 6 and 7. The second hypothesis on X_0 is rather odd, but necessary even when the first one is satisfied. Intuitively, in the concrete semantics, one cannot return from a procedure when there is a single activation record on the stack, whereas this is possible in the abstract semantics, whenever a matching activation record of the calling procedure can be found in the second component of the abstract state.

```
proc P(x:int):(y:int) is
var z:int;
begin
   (sP) {⟨sP, x=3⟩}
   z := 42;
   (cP) {⟨cP, x=3 ∧ z=42⟩}
   y := Q(x);
   (eP) {⟨eP, x=y=3 ∧ z=42⟩}
end
```

```
proc Q(x:int):(y:int) is
begin
   (sQ) {⟨cP, x=3 ∧ z=42⟩·⟨sQ, x=3⟩}
   if ? then
      (c1Q) {⟨cP, x=3 ∧ z=42⟩·⟨sQ, x=3⟩}
      y := R(x);
      (c2Q) ∅
   else
      y:=x;
      (c3Q) {⟨cP, x=3 ∧ z=42⟩·⟨c3Q, x=y=3⟩}
   endif
   (eQ) {⟨cP, x=3 ∧ z=42⟩·⟨eQ, x=y=3⟩}
end
```

```
proc R(x:int):(y:int) is
begin
   (sR) {⟨cP, x=3 ∧ z=0⟩·⟨c1Q, x=3⟩·⟨sR, x=3⟩}
   halt;
   ...
   end
```

Figure 7: Example illustrating the role of Hypothesis 1 of Proposition 8

```
proc P(x:int):(y:int) is begin
   (sP) {⟨sP, x=3⟩}
   y := Q(x);
   (eP)
end
```

```
proc Q(x:int):(y:int) is begin
   (sQ) {⟨sP, x=3⟩·⟨sQ, x=3⟩}
   halt;
   (cQ) ∅
   y := x;
   (eQ) {⟨eQ, x=y=3⟩}
end
```

Figure 8: Example illustrating the role of Hypothesis 2 of Proposition 8

Example 6 *Consider the program depicted in Fig 7, where the set X_0 of initial states is defined with plain boxes, and the set reach$(X_0) \setminus X_0$ with dashed boxes. The conditional* `if ?` *is supposed to be a non-deterministic choice. Notice that X_0 is not prefix-closed, but still "well-formed" in many aspects. For instance, $\langle \mathsf{s_P}, x=3 \wedge z=0\rangle \cdot \langle \mathsf{c^1_Q}, x=3\rangle \cdot \langle \mathsf{s_R}, x=3\rangle$ is reachable from $\langle \mathsf{s_P}, x=3 \wedge z=0\rangle$. However, the conclusion of Proposition 8 does not hold.*

Let $X = reach(X_0)$ and $Y = \alpha_f(X)$. Let us show that $post^{\mathsf{a}}_f(Y) \not\subseteq Y$. If we compute $Z = post^{\mathsf{a}}_f(\mathsf{e_Q} \xrightarrow{\langle \mathsf{ret}\ y{:=}Q(x)\rangle} \mathsf{e_P})(Y)$, we obtain that $\langle \mathsf{e_P}, x=y=3 \wedge z=0\rangle \in \overline{Z}$, and thus $Z \not\subseteq Y$, because $\langle \mathsf{s_P}, x=3 \wedge z=0\rangle \in \widetilde{Y}$ is associated with $\langle \mathsf{e_Q}, x=y=3\rangle \in \overline{Y}$.

Example 7 *Consider the program of Fig 8, where the set X_0 of initial states is defined with plain boxes, and the set reach$(X_0) \setminus X_0$ is shown with dashed boxes.*

Let $X = reach(X_0)$ and $Y = \alpha_f(X)$. If we compute $Z = post^{\mathsf{a}}_f(\mathsf{e_Q} \xrightarrow{\langle \mathsf{ret}\ y{:=}Q(x)\rangle} \mathsf{e_P})(Y)$, we will obtain that $\langle \mathsf{e_P}, x=y=3\rangle \in \overline{Z}$, and thus $Z \not\subseteq Y$, because $\langle \mathsf{s_P}, x=3\rangle \in \widetilde{Y}$ is associated with $\langle \mathsf{e_Q}, x=y=3\rangle \in \overline{Y}$. It seems in the abstract semantics, that $\langle \mathsf{s_P}, a=3\rangle \cdot \langle \mathsf{s_Q}, x=3\rangle$ can lead to $\langle \mathsf{s_P}, x=3\rangle \cdot \langle \mathsf{e_Q}, x=y=3\rangle$, which is false of course because of the `halt` *instruction.*

3.4 Non Optimality of the Backward Analysis

For the backward analysis, we cannot have a result similar to Theorem 2, since this theorem relies on Proposition 1, but there is no equivalent property for backward analysis. The reason is that whereas formal call parameters are frozen during the forward execution of a procedure, this is not the case for formal return parameters during a backward execution. Consequently, we cannot establish relationship between the values at the exit point of a procedure with the value at the entry point. The following example shows that under the same assumptions as the hypothesis of theorem 2, we do not get a best correct approximation of *coreach*(X_0).

Example 8 *Consider the program of Example 2, and let us perform a backward analysis starting from* $X_0 = \big\{\langle \mathsf{e_P}, a=42 \wedge b=43\rangle, \langle \mathsf{e_R}, x=y=42\rangle\big\}$. *We get the following results for the abstract backward analysis*[11]*:*

`proc P():() is`
`var a,b : int;`
`begin` $(\mathsf{s_P})$ $\{\langle \mathsf{s_P}, a=42\rangle\}$
`a := 42;`
$(\mathsf{c_P})$ $\left\langle \begin{array}{l}\{\langle \mathsf{c_P}, a=42\rangle\},\\ \{\langle \mathsf{c_P}, a=42\rangle\}\end{array}\right\rangle$
`b := Q(a);`
`end`
$(\mathsf{e_P})$ $\boxed{\{\langle \mathsf{e_P}, a=b-1=42\rangle\}}$
	`proc Q(i:int):(r:int) is`
	`begin`
	$(\mathsf{s_Q})$ $\{\langle \mathsf{s_Q}, i=42\rangle\}$
	`r := i;`
	`end`
	$(\mathsf{e_Q})$ $\{\langle \mathsf{e_Q}, i=42 \wedge 42\leq r\leq 43\rangle\}$
		`proc R() : () is`
		`begin` $(\mathsf{s_R})$ $\{\langle \mathsf{s_R}, x=42\rangle\}$
		`x := 42;`
		$(\mathsf{c_R})$ $\left\langle \begin{array}{l}\{\langle \mathsf{c_R}, x=42\rangle\},\\ \{\langle \mathsf{c_R}, x=42\rangle\}\end{array}\right\rangle$
		`y := Q(x);`
		`end`
		$(\mathsf{e_R})$ $\boxed{\{\langle \mathsf{e_R}, x=y=42\rangle\}}$

In the abstract analysis, $\mathsf{s_P}$ *is coreachable, but it the concrete one it is not, because in procedure* Q $\langle \mathsf{e_Q}, i=42 \wedge r=43\rangle$ *does not have any predecessor. So the abstract backward analysis delivers in this case a strict upper-approximation of the best correct approximation of coreach*(X_0).

We think that to obtain a better precision of the abstract analysis, we would need a different semantics enjoying duality properties w.r.t. the abstract forward semantics. Formal return parameters should be copied into frozen variables upon procedure returns, during a backward execution, and $\widetilde{Y}$ should keep track of activation records at the return point of the procedure call, instead of activation records at the call point of the procedure call as in A_f.

This requires among others the use of distinct Galois connections for forward and backward analysis and has the drawback that forward and backward analysis cannot be combined any more by simply intersecting them.

3.5 Discussion

The abstract semantics we presented in this section needs to be abstracted in the case where variables are of infinite type. But if we are interested only on activation records reachable or coreachable in the stack, it is a better starting point than the standard operational semantics, especially for the forward analysis where Theorem 2 applies. The point is that now there is no Kleene operator in the definition of the abstract domain, so the many classical abstract interpretation techniques for intraprocedural problems can be applied. Indeed, if we note D the union of all data domains in which variables takes their values, the abstract lattice has the structure $\big(\wp(\mathit{Ctrl} \times D^n)\big)^2$, which is similar to the structure of the state-space of intraprocedural programs, apart for the square exponent. Section 6 will give more details on these aspects.

Another point is that we do not address programs with global variables and side-effects. However this can be easily addressed by transferring back and forth all global variables during procedure calls and returns, as suggested by many authors using the functional approach [CC77b, SP81, KS92].

[11] For simplicity, we omit $\widetilde{Y}$ for non-call-sites.

Theorems 1 and 2 are actually very similar to the results obtained in the functional approach of [SP81] and in [KS92], even if they are formulated in a quite different way, because these papers use the functional approach. Both papers shows that if the postcondition function(s) defined on activation records (and not on stacks as in our case) are distributive, then the so-called *Meet Over all Valid Paths* solution coincides with the *Least Fix Point* solution.[12] As we relate our abstract semantics directly to the standard semantics, Theorem 2 says that we get the *Meet Over all Valid Paths* solution, even if we did not make an explicit use of the notion of interprocedural valid paths. Another interesting observation is that they have equivalent hypotheses than ours: they start the analysis with one-element stacks, and from a main procedure which cannot be called, even if this latter hypothesis is left implicit.

Our contribution gives however a more flexible method, as we can start the analysis from any set of states, possibly having non-empty stack tails, even if the analysis is not optimal any more. We are also able to recover information on stack contents.

4 A Refined Stack Abstraction for Functional Programs

Our motivation to improve the precision of the previous stack abstraction (*cf.* Section 3) is twofold:

1. The previous abstraction may give poor results if we want to recover information on stack contents, and not only about activation records present in the reachable stacks;
2. A second abstract interpretation step is required anyway. So if some aspects of the abstract semantics are improved, the second abstract interpretation step can benefit from it, since it can use more information to perform further abstraction.

Typically, for debugging applications, precise knowledge about the stacks would allow to ask questions like:

> If I am in procedure P with an environment ϵ_P, and if I assume that I have been called successively by Q and R, am I able to enter procedure S ?

Formally, this amounts to asking whether an execution path of the following form exists:

$$\langle c_R, ?\rangle \cdot \langle c_Q, ?\rangle \cdot \langle c_P, \epsilon_P\rangle \to^* \Gamma \cdot \langle s_S, ?\rangle$$

4.1 New Stack Abstraction

We suggest to replace activation records $\langle c_n, \epsilon_n\rangle$ used in the previous semantics by objects of the form $\langle c_0 \dots c_n, \epsilon_n\rangle$, which we will call *extended activation records*. This corresponds to replacing single control points c_n labelling environments by sequences $c_0 \dots c_n$, which are called *call strings* in [SP81]. Basically, our abstraction consists then in abstracting a state $q = \langle c_0, \epsilon_0\rangle \dots \langle c_n, \epsilon_n\rangle$ by the pair of sets $\big\langle \{\langle c_0 \dots c_n, \epsilon_n\rangle\},\ \{\langle c_0 \dots c_i, \epsilon_i\rangle \mid i < n\}\big\rangle$. The abstract semantics proposed in this section can be seen as a synthesis of the two distinct methods described in [SP81].

Let us note *EAct* the set of extended activation records:

$$\mathit{EAct} \stackrel{\text{def}}{=} \mathit{Ctrl}^+ \times \mathit{LEnv}$$

We can extend the notion of valid calling activation record (Definition 3) to extended activation records: $\langle \omega, \epsilon\rangle$ is valid for $\langle \omega', \epsilon'\rangle$ if $\omega = \omega_0{\cdot}c$, $\omega' = \omega_0{\cdot}c{\cdot}c'$, and $\langle c, \epsilon\rangle$ is a valid calling activation record for $\langle c', \epsilon'\rangle$. The better precision of the new abstract semantics is due to the additional condition (on equality of call-strings) in the definition of valid calling (extended) activation records.

[12] They actually talk about maximal fixpoints, following the usual convention in data flow analysis, but it can be transposed by duality to the convention used in abstract interpretation.

Abstract Domain. We extend the domain A_f defined in section 3.1, (8) by replacing activation records by extended activation records:

$$A_s \stackrel{\text{def}}{=} \wp(EAct) \times \wp(EAct) \tag{24}$$

A_s is equipped with the standard lattice structure $A_s(\sqsubseteq, \sqcup, \sqcap, \top, \bot)$ of a Cartesian product of lattices.

Abstraction and Concretisation Functions. A_s is connected to the state space $\wp(State)$ by the following Galois connection $\wp(State) \underset{\alpha_s}{\overset{\gamma_s}{\leftrightarrows}} A_s$, where the abstraction function $\alpha_s = \langle \overline{\alpha_s}, \widetilde{\alpha_s} \rangle$ is defined by:

$$\begin{array}{lcll} \forall q = \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \in State & : & \alpha_s(\{q\}) & = \left\langle \begin{array}{c} \{\langle c_0 \ldots c_n, \epsilon_n \rangle\}, \\ \{\langle c_0 \ldots c_i, \epsilon_i \rangle \mid 0 \leq i < n\} \end{array} \right\rangle \\ \forall X \in \wp(State) & : & \alpha_s(X) & = \bigsqcup_{q \in X} \alpha_s(\{q\}) \end{array} \tag{25}$$

and the concretisation function γ_s by, for any $Y = \langle \overline{Y}, \widetilde{Y} \rangle \in A_s$:

$$q = \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \in \gamma_s(Y) \iff \left\{ \begin{array}{l} \langle c_0 \ldots c_n, \epsilon_n \rangle \in \overline{Y} \\ \forall 0 \leq i < n : \langle c_0 \ldots c_i, \epsilon_i \rangle \in \widetilde{Y} \\ \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \text{ is a consistent stack} \end{array} \right. \tag{26}$$

Proposition 9 *$\wp(S) \underset{\alpha_s}{\overset{\gamma_s}{\leftrightarrows}} A_s$ is a Galois connection. It follows that $\alpha_s \circ \gamma_s \sqsubseteq id$ and $\gamma_s \circ \alpha_s \sqsupseteq id$.*

Proof. We defined γ_s as $\gamma_s(Y) = \bigcup\{X \mid \alpha_s(X) \sqsubseteq Y\}$, see Proposition 7 of [CC92a]. □

Again, we can define *canonical abstract values* as those abstract values $Y \in A_s$ such that $\alpha_s \circ \gamma_s(Y) = Y$. But since the set of canonical abstract states is not closed by the meet operator $\sqcap$ of A_s, we do not restrict our attention to canonical values (which would be more convenient).

Using A_s looses less information than using A_f. For instance, thanks to call-strings, we cannot have any more the problem of (12). However we have a slightly subtler phenomenon: it is possible to have

$$\begin{array}{lll} \langle c_0, \epsilon_0 \rangle \cdot \langle c_1, \epsilon_1' \rangle & \in & (\gamma_s \circ \alpha_s)\Big(\{\ \langle c_0, \epsilon_0 \rangle \cdot \langle c_1, \epsilon_1 \rangle\ ,\ \langle c_0, \epsilon_0' \rangle \cdot \langle c_1, \epsilon_1' \rangle\ \}\Big) \\ & \notin & \qquad\quad\ \{\ \langle c_0, \epsilon_0 \rangle \cdot \langle c_1, \epsilon_1 \rangle\ ,\ \langle c_0, \epsilon_0' \rangle \cdot \langle c_1, \epsilon_1' \rangle\ \} \end{array} \tag{27}$$

if $\langle c_0, \epsilon_0 \rangle$ is a valid calling activation record for $\langle c_1, \epsilon_1' \rangle$ and $\epsilon_1 \neq \epsilon_1'$. That means that while the abstraction keeps the length of the stack exact, it can still induce some "cross-over" of activation records belonging to different concrete stacks. However, the constraints on valid calling contexts representing the parameter passing conventions will rule out most of the states in excess.

Example 9 *For instance, considering the program of Example 2, we have*

$$\begin{array}{l} (\gamma \circ \alpha) \left(\left\{ \begin{array}{l} \langle \mathsf{c_P},\ a{=}42 \rangle \cdot \langle \mathsf{s_Q},\ i_Q{=}42 \rangle, \\ \langle \mathsf{c_P}, a{=}17 \rangle \cdot \langle \mathsf{s_Q},\ i_Q{=}17 \rangle \end{array} \right\} \right) \\ \quad = \gamma \left(\left\langle \left\{ \begin{array}{l} \langle \mathsf{c_P},\ a{=}42 \rangle, \\ \langle \mathsf{c_P},\ a{=}17 \rangle \end{array} \right\}, \left\{ \begin{array}{l} \langle \mathsf{c_P} \cdot \mathsf{s_Q},\ i_Q{=}42 \rangle, \\ \langle \mathsf{c_P} \cdot \mathsf{s_Q},\ i_Q{=}17 \rangle \end{array} \right\} \right\rangle \right) \\ \quad = \left\{ \begin{array}{l} \langle \mathsf{c_P},\ a{=}42 \rangle \cdot \langle \mathsf{s_Q},\ i_Q{=}42 \rangle, \\ \langle \mathsf{c_P}, a{=}17 \rangle \cdot \langle \mathsf{s_Q},\ i_Q{=}17 \rangle \end{array} \right\} \end{array}$$

since $\langle \mathsf{c_P}, \text{true}, a{=}42 \rangle$ *is not a valid calling context for* $\langle \mathsf{c_P} \cdot \mathsf{s_Q}, \text{true}, i_Q{=}17 \rangle$*, nor is* $\langle \mathsf{c_P}, \text{true}, a{=}17 \rangle$ *a valid calling context for* $\langle \mathsf{c_P} \cdot \mathsf{s_Q}, \text{true}, i_Q{=}42 \rangle$*.*

Actually we can *prove* that this second abstraction is more precise than the previous one by exhibiting the Galois connection $A_s \underset{\alpha_{s\to f}}{\overset{\gamma_{s\leftarrow f}}{\leftrightarrows}} A_f$ relating them:

$$\alpha_{s\to f}(Y) = \left\langle \begin{array}{l} \{\langle c_n, \epsilon_n\rangle \mid \langle c_0 \dots c_n, \epsilon_n\rangle \in \overline{Y}\}, \\ \{\langle c_n, \epsilon_n\rangle \mid \langle c_0 \dots c_n, \epsilon_n\rangle \in \widetilde{Y}\} \end{array} \right\rangle \tag{28}$$

$$\gamma_{s\leftarrow f}(Y) = \sqcup_s\{X \in A_s \mid \alpha_{s\to f}(X) \sqsubseteq_f Y\} \tag{29}$$

4.2 Forward and Backward Analysis

The transfer functions $post^{\mathsf{a}}_s$ and pre^{a}_s are defined in Tables 6 and 7 for the different transitions. These definitions are based on the same principles as the definitions in Tables 4 and 5, but now we can use call strings to perform a more accurate "matching" of possible (extended) calling contexts with top contexts. Furthermore, we can now mirror more precisely the behaviour of the concrete transfer functions in the case of procedure returns (for forward analysis) or procedure calls (for backward analysis).

Recall from (15) that we were able to extract after convergence of the forward analysis the predicate transformer of any procedure P_j specialised on the reachable inputs of P_j. Now, we can partition this predicate transformer using call strings:

$$\begin{array}{rcl} \phi_j &=& \bigcup_{\omega\in Ctrl^*} \phi_j(\omega) \\ \phi_j(\omega) &=& \big\{(\mathbf{x},\mathbf{y}) \;\big|\; \exists\langle \omega\cdot e_j, \epsilon\rangle \in \overline{Y} : \mathbf{x} = \epsilon(\mathbf{fp}_j) \wedge \mathbf{y} = \epsilon(\mathbf{fr}_j)\big\} \end{array} \tag{32}$$

Also, the predicate transformer defined by (15) is exact if fixpoint computations starts with a initial set of states satisfying the hypotheses of Theorem 2. Here, we will show a similar result, but with a *less restrictive assumption* about initial states.

First, we have to check that the operators we just defined are correct approximations.

Proposition 10 ($post^{\mathsf{a}}_s, pre^{\mathsf{a}}_s$ are correct approximations of $post, pre$)

$$post^{\mathsf{a}}_s \sqsupseteq \alpha_s \circ post \circ \gamma_s \tag{33}$$

$$pre^{\mathsf{a}}_s \sqsupseteq \alpha_s \circ post \circ \gamma_s \tag{34}$$

Proof. Appendix A.2. □

Here again, we *could* prove that $post^{\mathsf{a}}_s$ and pre^{a}_s are *best correct approximations* if we make their result canonical (by applying $\alpha_s \circ \gamma_s$).

Let us define for any $Y_0 \in A_s$,

$$\begin{array}{rclrcl} F^{\mathsf{a}}_s[Y_0](Y) &\stackrel{\text{def}}{=}& Y_0 \sqcup post^{\mathsf{a}}_s(Y) & reach^{\mathsf{a}}_s(Y_0) &\stackrel{\text{def}}{=}& \text{lfp}(F^{\mathsf{a}}_s[Y_0]) \\ G^{\mathsf{a}}_s[Y_0](Y) &\stackrel{\text{def}}{=}& Y_0 \sqcup pre^{\mathsf{a}}_s(Y) & coreach^{\mathsf{a}}_s(Y_0) &\stackrel{\text{def}}{=}& \text{lfp}(G^{\mathsf{a}}_s[Y_0]) \end{array}$$

As in Section 3.2.2, we deduce from Proposition 10 and standard abstract interpretation theory that we compute a correct approximation of reachable states.

Theorem 3 (Soundness of the abstract forward and backward analysis)

$$reach^{\mathsf{a}}_s \sqsupseteq \alpha_s \circ reach \circ \gamma_s \tag{35}$$

$$coreach^{\mathsf{a}}_s \sqsupseteq \alpha_s \circ coreach \circ \gamma_s \tag{36}$$

Proof. Identical to the proof of Theorem 1. □

$$c \xrightarrow{\langle R\rangle} c' \qquad \overline{post^{\mathsf{a}}_s}(\tau)(Y) = \big\{\langle \omega\cdot c', \epsilon'\rangle \mid \langle \omega\cdot c, \epsilon\rangle \in \overline{Y} \wedge R(\epsilon,\epsilon')\big\} \tag{30a}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} s_j \qquad \overline{post^{\mathsf{a}}_s}(\tau)(Y) = \left\{ \langle \omega\cdot c\cdot s_j, \epsilon_j\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot c,\epsilon\rangle \in \overline{Y} \\ \epsilon_j(\mathbf{fp}^{(k)}_j) = \epsilon(\mathbf{x}^{(k)}) \end{array} \right\} \tag{30b}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} c \qquad \overline{post^{\mathsf{a}}_s}(\tau)(Y) = \left\{ \langle \omega\cdot c, \epsilon'\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot \mathsf{call}(c)\cdot e_j, \epsilon_j\rangle \in \overline{Y} \\ \langle \omega\cdot \mathsf{call}(c), \epsilon\rangle \in \widetilde{Y} \\ \epsilon(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}^{(k)}_j) \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon_j(\mathbf{fr}^{(k)}_j)] \end{array} \right\} \tag{30c}$$

$$c \xrightarrow{\langle R\rangle} c' \qquad \widetilde{post^{\mathsf{a}}_s}(\tau)(Y) = \widetilde{Y} \tag{30d}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} s_j \qquad \widetilde{post^{\mathsf{a}}_s}(\tau)(Y) = \widetilde{Y} \;\cup\; \big\{\langle \omega\cdot c, \epsilon\rangle \in \overline{Y}\big\} \tag{30e}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} c \qquad \widetilde{post^{\mathsf{a}}_s}(\tau)(Y) = \big\{\langle \omega, \epsilon\rangle \in \widetilde{Y} \;\big|\; \langle \omega\cdot\omega'\cdot \mathsf{call}(c), \epsilon'\rangle \in \widetilde{Y}\big\} \tag{30f}$$

Table 6: Equations Defining $post^{\mathsf{a}}_s$

$$c \xrightarrow{\langle R\rangle} c' \qquad \overline{pre^{\mathsf{a}}_s}(\tau)(Y) = \big\{\langle \omega\cdot c, \epsilon\rangle \mid \langle \omega\cdot c', \epsilon'\rangle \in \overline{Y} \wedge R(\epsilon,\epsilon')\big\} \tag{31a}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} s_j \qquad \overline{pre^{\mathsf{a}}_s}(\tau)(Y) = \left\{ \langle \omega\cdot c, \epsilon\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot c\cdot s_j,\, \epsilon_j\rangle \in \overline{Y} \\ \langle \omega\cdot c,\, \epsilon\rangle \in \widetilde{Y} \\ \epsilon(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}^{(k)}) \end{array} \right\} \tag{31b}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} c \qquad \overline{pre^{\mathsf{a}}_s}(\tau)(Y) = \left\{ \langle \omega\cdot \mathsf{call}(c)\cdot e_j, \epsilon_j\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot c, \epsilon\rangle \in \overline{Y} \\ \qquad \epsilon_j(\mathbf{fr}^{(k)}_j) = \epsilon(\mathbf{y}^{(k)}) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_j(\mathbf{fp}^{(k)}_j) = \epsilon(\mathbf{x}^{(k)}) \end{array} \right\} \tag{31c}$$

$$c \xrightarrow{\langle R\rangle} c' \qquad \widetilde{pre^{\mathsf{a}}_s}(\tau)(Y) = \widetilde{Y} \tag{31d}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} s_j \qquad \widetilde{pre^{\mathsf{a}}_s}(\tau)(Y) = \big\{\langle \omega, \epsilon\rangle \in \widetilde{Y} \;\big|\; \langle \omega\cdot\omega'\cdot c, \epsilon'\rangle \in \widetilde{Y}\big\} \tag{31e}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}{:=}P_j(\mathbf{x})\rangle} c \qquad \widetilde{pre^{\mathsf{a}}_s}(\tau)(Y) = \widetilde{Y} \;\cup\; \left\{ \langle \omega\cdot \mathsf{call}(c), \epsilon\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot c, \epsilon'\rangle \in \overline{Y} \\ \forall z \notin \mathbf{y} : \epsilon(z) = \epsilon'(z) \end{array} \right\} \tag{31f}$$

Table 7: Equations Defining pre^{a}_s

4.3 Optimality Results for Forward Analysis

As already mentioned, A_s is strictly more precise than A_f, and we have the Galois connections depicted besides. It is also quite clear that the abstract postcondition and precondition operators in A_s are more precise than in A_f. Hence we can inherit from Theorem 2 in A_s, whenever we start from a set of initial states satisfying the hypotheses of Theorem 2. This means that we keep exact information on top activation records:

$$
\begin{tikzcd}
\wp(\mathit{State}) \arrow[r, shift left, "\alpha_s"'] \arrow[rr, bend right, "\alpha_f"'] & A_s \arrow[l, shift left, "\gamma_s"'] \arrow[r, shift left, "\alpha_{s\to f}"'] & A_f \arrow[l, shift left, "\gamma_{s\leftarrow f}"'] \arrow[ll, bend right, "\gamma_f"']
\end{tikzcd}
$$

Proposition 11 (Exact computation of top activation records) *Let $X_0 \in \wp(\mathit{State})$ satisfying the hypotheses of Theorem 2. Then*[13]

$$\overline{\alpha_{s\to f}} \circ \mathit{reach}_s^{\mathsf{a}} \circ \alpha_s(X_0) \;=\; \overline{\alpha_f} \circ \mathit{reach}(X_0) \qquad = \{\langle c,\epsilon\rangle \mid \Gamma\cdot\langle c,\epsilon\rangle \in \mathit{reach}(X_0)\} \tag{37a}$$

$$\widetilde{\alpha_{s\to f}} \circ \mathit{reach}_s^{\mathsf{a}} \circ \alpha_s(X_0) \;=\; \widetilde{\alpha_f} \circ \mathit{reach}(X_0) \qquad = \{\langle c,\epsilon\rangle \mid \Gamma\cdot\langle c,\epsilon\rangle\cdot\Gamma' \in \mathit{reach}(X_0)\} \tag{37b}$$

The intuition suggests that we can get an even better result, in two directions: we can now say something on extended activation records, instead of just activation records, and we can relax some assumptions on the set of initial states. It is clear that the second hypothesis of Proposition 8 is not needed any more, because we know that an extended activation record can return to a caller only when its call-string component is of length at least 2. However, the condition that the set of initial states is prefix-closed is still necessary as shown by the following example.

Example 10 *Let us take again the program of Fig. 7 (page 21), where X_0 (depicted with plain boxes) is not prefix-closed. Let $X = \mathit{reach}(X_0)$, $Y = \alpha_s(X)$ and let us show that $\mathit{post}_s^{\mathsf{a}}(Y) \not\in Y$. If we compute $Z = \mathit{post}_s^{\mathsf{a}}(\mathsf{e_Q} \xrightarrow{\langle \mathsf{ret}\ y:=Q(x)\rangle} \mathsf{e_P})(Y)$, we will obtain that $\langle \mathsf{e_P}, x=y=3 \wedge z=0\rangle \in \overline{Z}$, and thus $Z \not\sqsubseteq Y$, because $\langle \mathsf{s_P}, x=3 \wedge z=0\rangle \in \widetilde{Y}$ is associated with $\langle \mathsf{s_P}\cdot\mathsf{e_Q}, x=y=3\rangle \in \overline{Y}$.*

By Proposition 4, this basically means that the initial states can only have one-element stacks, if we want to have the optimality result.

Proposition 12 *Let $X_0 \in \wp(\mathit{State})$ be a prefix-closed set of states. Let $X = \mathit{reach}(X_0)$ and $Y = \alpha_f(X)$. Then $\mathit{post}_s^{\mathsf{a}}(Y) \sqsubseteq Y$.*

Proof. Similar to the proof of Proposition 8, the main difference is in the case of procedure returns.

Let $X_0 \in \wp(\mathit{State})$ prefix-closed, $X = \mathit{reach}(X_0)$ and $Y = \alpha_s(X)$. By Proposition 4, X is prefix-closed. We will prove that for any transition τ, $\mathit{post}_s^{\mathsf{a}}(\tau)(Y) \sqsubseteq Y$.

$\tau = c \xrightarrow{R} c'$**:** According to (30d), $\widetilde{\mathit{post}_s^{\mathsf{a}}}(\tau)(Y) = \widetilde{Y}$. According to (30a) and (Intra), $\overline{\mathit{post}_s^{\mathsf{a}}}(\tau)(Y) = \{\langle \omega\cdot c', \epsilon'\rangle \mid \langle \omega\cdot c, \epsilon\rangle \in \overline{Y} \wedge R(\epsilon,\epsilon')\} = \{\langle \omega\cdot c', \epsilon'\rangle \mid \langle \omega\cdot c, \epsilon\rangle \in \overline{Y} \wedge \langle c,\epsilon\rangle \to \langle c',\epsilon'\rangle\}$. Now,

(i) Since $Y = \alpha_s(X)$, $\langle \omega\cdot c,\epsilon\rangle \in \overline{Y}$ implies $\exists\Gamma = \langle c_0,\epsilon_0\rangle\ldots\langle c_{n-1},\epsilon_{n-1}\rangle : \Gamma\cdot\langle c,\epsilon\rangle \in X$, with $\omega = c_0\ldots c_{n-1}$;

(ii) $\langle c,\epsilon\rangle \to \langle c',\epsilon'\rangle$ implies $\Gamma\cdot\langle c,\epsilon\rangle \to \Gamma\cdot\langle c',\epsilon'\rangle$ (see (6)).

Since $\Gamma\cdot\langle c',\epsilon'\rangle \in X$ (by definition of X), we conclude that $\langle \omega\cdot c',\epsilon'\rangle \in \overline{Y}$. Hence, $\mathit{post}_s^{\mathsf{a}}(\tau)(Y) \sqsubseteq Y$.

$\tau = c \xrightarrow{\langle \mathtt{call}\ \mathtt{y}:=P_j(\mathtt{x})\rangle} c'$**:** According to (30b) and (30e),

$$\begin{aligned}
\mathit{post}_s^{\mathsf{a}}(\tau)(Y) &= \left\langle \begin{array}{c} \{\langle \omega\cdot c\cdot c',\epsilon'\rangle \mid \langle \omega\cdot c,\epsilon\rangle \in \overline{Y} \wedge \epsilon'(\mathbf{fp}_j^{(k)}) = \epsilon(\mathbf{x}^{(k)})\}, \\ \widetilde{Y} \cup \{\langle \omega\cdot c,\epsilon\rangle \in \overline{Y}\} \end{array} \right\rangle \\
&= \left\langle \begin{array}{c} \{\langle \omega\cdot c\cdot c',\epsilon'\rangle \mid \langle \omega\cdot c,\epsilon\rangle \in \overline{Y} \wedge \langle c,\epsilon\rangle \xrightarrow{\tau} \langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle\}, \\ \widetilde{Y} \cup \{\langle \omega\cdot c,\epsilon\rangle \in \overline{Y}\} \end{array} \right\rangle
\end{aligned}$$

Consider $\langle \omega\cdot c,\epsilon\rangle \in \overline{Y}$. As before, this implies $\exists\Gamma = \langle c_0,\epsilon_0\rangle\ldots\langle c_{n-1},\epsilon_{n-1}\rangle : \Gamma\cdot\langle c,\epsilon\rangle \in X$, with $\omega = c_0\ldots c_{n-1}$. We have $\Gamma\cdot\langle c,\epsilon\rangle \to \Gamma\cdot\langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle$. So $\Gamma\cdot\langle c,\epsilon\rangle\cdot\langle c',\epsilon'\rangle \in X$, which implies $\langle \omega\cdot c\cdot c',\epsilon'\rangle \in \overline{Y}$ and consequently $\langle \omega\cdot c,\epsilon\rangle \in \widetilde{Y}$.

[13] As before, we note $\alpha_{s\to f} = \langle \overline{\alpha_{s\to f}}, \widetilde{\alpha_{s\to f}}\rangle$.

$\tau = c_{n+1} \xrightarrow{\langle \mathbf{ret}\, \mathbf{y}{:=}P_j(\mathbf{x})\rangle} \mathsf{ret}(c_n)$**:** This is the interesting case. According to (30f), $\widetilde{post_s^{\mathsf{a}}}(\tau)(Y) \sqsubseteq \widetilde{Y}$. According to (30c),

$$\overline{post_s^{\mathsf{a}}}(\tau)(Y) \quad = \quad \left\{ \langle \omega \cdot \mathsf{ret}(c_n), \epsilon''_n \rangle \;\middle|\; \begin{array}{l} \langle \omega \cdot c_n \cdot c_{n+1}, \epsilon_{n+1} \rangle \in \overline{Y} \\ \langle \omega \cdot c_n, \epsilon'_n \rangle \in \widetilde{Y} \\ \epsilon'_n(x^{(k)}) = \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \\ \epsilon''_n = \epsilon'_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}_j^{(k)})] \end{array} \right\}$$

- $\langle \omega{\cdot}c_n, \epsilon'_n \rangle \in \widetilde{Y}$ implies $\exists \Gamma = \langle c_0, \epsilon'_0 \rangle \ldots \langle c_{n-1}, \epsilon'_{n-1} \rangle, \exists \Gamma' : \Gamma{\cdot}\langle c_n, \epsilon'_n \rangle{\cdot}\Gamma' \in X$, with $\omega = c_0 \ldots c_{n-1}$. As X is prefix-closed, $\Gamma \cdot \langle c_n, \epsilon'_n \rangle \in X$.
- $\langle \omega \cdot c_n \cdot c_{n+1}, \epsilon_{n+1} \rangle \in \overline{Y}$ implies $\exists \Upsilon = \langle c_0, \epsilon_0 \rangle \ldots \langle c_{n-1}, \epsilon_{n-1} \rangle, \exists \epsilon_n : \Upsilon \cdot \langle c_n, \epsilon_n \rangle \cdot \langle c_{n+1}, \epsilon_{n+1} \rangle \in X$. As X is prefix-closed, $\Upsilon \cdot \langle c_n, \epsilon_n \rangle \in X$ and $\Upsilon \cdot \langle c_n, \epsilon_n \rangle \to^* \Upsilon \cdot \langle c_n, \epsilon_n \rangle \cdot \langle c_{n+1}, \epsilon_{n+1} \rangle$ as in (7) (see Definition 1).

$\Upsilon \cdot \langle c_n, \epsilon_n \rangle$ and $\Gamma \cdot \langle c_n, \epsilon'_n \rangle$ satisfy the hypotheses of Proposition 1. Thus we have $\Gamma \cdot \langle c_n, \epsilon'_n \rangle \to^* \Gamma{\cdot}\langle c_n, \epsilon'_n \rangle{\cdot}\langle c_{n+1}, \epsilon_{n+1} \rangle$, and consequently $\Gamma{\cdot}\langle c_n, \epsilon'_n \rangle{\cdot}\langle c_{n+1}, \epsilon_{n+1} \rangle \in X$. So by rule (Return) we conclude that $\Gamma \cdot \langle \mathsf{ret}(c_n), \epsilon''_n \rangle \in X$ and (by definition of Y) $\langle c_0 \ldots c_{n-1} \cdot \mathsf{ret}(c_n), \epsilon''_n \rangle \in \overline{Y}$, so $\overline{post_s^{\mathsf{a}}}(Y) \sqsubseteq \overline{Y}$.

□

We can now state the following *optimality* theorem, w.r.t. the precision of the forward analysis.

Theorem 4 (Optimality of the forward analysis) *Let $X_0 \in \wp(State)$ be prefix-closed. Then*

$$reach_s^{\mathsf{a}} \circ \alpha_s(X_0) = \alpha_s \circ reach(X_0) \tag{38}$$

This implies that

$$\begin{aligned} \overline{reach_s^{\mathsf{a}}} \circ \alpha_f(X_0) &= \{\langle c_0 \ldots c_n, \epsilon_n \rangle \mid \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \in reach(X_0)\} \\ \widetilde{reach_s^{\mathsf{a}}} \circ \alpha_f(X_0) &= \{\langle c_0 \ldots c_n, \epsilon_n \rangle \mid \exists \Gamma' \in Act^+ : \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \cdot \Gamma' \in reach(X_0)\} \end{aligned}$$

This abstraction can be viewed as a synthesis and a generalisation of the two approaches described in [SP81], merging their functional and their call string approach. It is much more general, because in their call string approach, no local variables are considered, which simplifies abstracting the stack.

5 Handling Global Variables

There are basically two solutions to add global variables to the previous stack abstractions. The first one, which has been widely used in the functional approach, consists in passing forth and back global variables to called procedures, in order to use predicate transformers associated to procedures to model the evolution of global variables. This is equivalent to transforming the initial program with global variables into a program with local variables only. Using this technique, our two abstractions A_f and A_s can be applied to programs with global variables. In this case, we keep our accuracy results. However, this technique has the drawback, that it requires to duplicate and even to triple[14] the number of variables we have to deal with, and one of our motivations to use an operational approach is precisely to limit as much as possible the complexity of the lattice on which fixpoints are computed, and among others to limit the number of variables to be manipulated at the same time.

Therefore, we suggest here another solution, which consists in extending the previously presented stack abstractions with global variables. We propose to generalise the abstract semantics A_f and A_s to take into account global variables, by roughly keeping their information flow in the standard operational semantics. While this implies stronger approximations than in the previous sections, we keep the original number of variables.

[14] For every global variable, we would have to introduce a formal call and return parameter in every procedure, and we need at return points to combine the state of two procedures.

Program P		First iteration	Second iteration
`var g;`			
`proc succ(a):(b) is`			
`begin`		$g=a=0$	$0\leq g\leq 1 \wedge a=g$
`   b := a + 1;`			
`end`	$\overline{X}$	$g=0 \wedge a=0 \wedge b=a+1$	$0\leq g\leq 1 \wedge a=g \wedge b=a+1$
`proc P():() is`			
`var i,j;`			
`begin`			
`   g := 0; j := 0;`			
`   for i=0 to 10 do`	$\overline{Y}$	$g=i=j=0$	$0\leq g=i=j\leq 1$
	$\widetilde{Y}$	$i=j=0$	$0\leq i=j\leq 1$
`      j := succ(g);`	$\overline{Z}$	$g=i=0 \wedge j=1$	$0\leq g\leq 1 \wedge 0\leq i\leq 1 \wedge j=g+1$
`      g := g+1;`		$g=j=1 \wedge i=0$	$1\leq g\leq 2 \wedge 0\leq i\leq 1 \wedge j=g$
`   done;`			
`end`		$\emptyset$	$\emptyset$

Figure 9: Loss of Information whenever Global Actual Parameters are ignored

Following this idea, we could simply superimpose the information flow for global variables to our abstract semantics, thus replacing for instance

$$\begin{array}{lrlll} & A_s = & \wp(EAct) & \times & \wp(EAct) \\ \text{by} & & \wp(EAct) & \times & \wp(GEnv \times EAct) \end{array}$$

which allows to consider global variables for the current activation record, and to represent the tail of the call stack as in Section 4 by sets of extended activation record.

However, if an actual parameter of a procedure is a global variable, we are unable to exploit the equality between actual and formal parameters, which can lead to a important loss of precision upon return, as the following example shows.

Example 11 *Consider the program of Fig 9, where all variables are implicitly integers, and its partial forward analysis starting from the entry point of P and following the execution flow.*

Notice that the procedure `succ` *does not produce any side-effect. At the end of the* `for`*-loop we should find that $g=j=i+1$. However, in the second iteration, as the* global *actual parameter g is not taken into account in $\widetilde{Y}$, we combine any local environment in $\widetilde{Y}$ at the call site with any global and local environment $\overline{X}$ at the end of procedure* `succ`*, and then we assign j. We obtain $\overline{Z}$ just after the call to* `succ`*, where the equality $j=i+1$ is lost because of the assignment of j. We just keep $j=g+1$, because we had in $\overline{X}$ that $b=g+1$. This is quite dramatic, because i and j are disconnected and the analysis will converge with just $g\geq 1 \wedge i=10 \wedge j=g$ at the exit point of P.*

We can do better than in the example above if we memorise the values of the global parameters passed as arguments, and use them in order to better reconstruct the corresponding local environment. In order to formalise this idea, we suggest to replace the activation records of Sections 3 and 4 by contexts, i.e., pairs of a global environment and an activation record. This approach to take into account global variables can be applied to both of our previous abstractions. We choose here to extend the most precise one, using extended activation records, knowing that we can later abstract call strings by their head to obtain the first abstraction.

5.1 Stack Abstraction with Global Variables

The idea is to put $\big\langle\sigma, \langle\omega,\epsilon\rangle\big\rangle$ into the set $\widetilde{Y}$ representing tails of stacks, instead of ordinary activation records. However, in such triplets we need only to know the values of global variables which are passed as parameters, and the others can be forgotten. So we will project global environments to subsets of global variables. Let us note $\widehat{GEnv}$ the set of environments $\widehat{sigma}$ which are projections of global environments $\sigma \in GEnv$. For any $\sigma \in GEnv$, and $V \subseteq GVar$, we note $\sigma_{|V}$ rather $\widehat{\sigma}$ for the projection of σ on the subset V. Notice that given the activation record $\langle c_{n+1}, \epsilon_{n+1} \rangle$, saying that $\langle \widehat{\sigma}, \langle c_n, \epsilon_n \rangle \rangle$ is valid for $\langle c_{n+1}, \epsilon_{n+1} \rangle$ (according to Definition 2) defines completely $\widehat{\sigma}$, which is not the case for ϵ_n.

From now on, we will note $\langle \omega, \sigma, \epsilon \rangle$ elements of $GEnv \times EAct$, instead of noting them $\big\langle \sigma, \langle \omega, \epsilon \rangle \big\rangle$.

Abstract Domain. The abstract domain is

$$A_g \stackrel{\text{def}}{=} \wp\big(\widehat{GEnv} \times EAct\big) \times \wp\big(GEnv \times EAct\big) \tag{39}$$

which is equipped with the standard lattice structure $A_g(\sqsubseteq, \sqcup, \sqcap, \top, \bot)$ of a Cartesian product of lattices.

Abstraction and Concretisation Functions. A_g is connected to the state space $\wp(State)$ by the following Galois connection $\wp(State) \underset{\alpha_g}{\overset{\gamma_g}{\leftrightarrows}} A_g$, where the abstraction function $\alpha_g = \langle \overline{\alpha_g}, \widetilde{\alpha_g} \rangle$ is defined by:

$$\begin{array}{l}
\forall q = \big\langle \sigma_n,\ \langle c_0, \epsilon_0 \rangle \dots \langle c_n, \epsilon_n \rangle \big\rangle \in State : \\
\quad \alpha_g(\{q\}) = \\
\quad \left\langle \begin{array}{l} \big\{ \langle c_0 \dots c_n, \sigma_n, \epsilon_n \rangle \big\}, \\ \left\{ \langle c_0 \dots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \;\middle|\; 0 \le i < n : \begin{array}{l} I(c_i, s_{\mathsf{proc}(c_{i+1})}) = \langle \mathtt{call}\ \mathbf{y}_i := P_{\mathsf{proc}(c_{i+1})}(\mathbf{x}_i) \rangle \\ \widehat{\sigma}_i = \big[\mathbf{x}_i^{(k)} \mapsto \epsilon_{i+1}(\mathbf{fp}^{(k)}_{\mathsf{proc}(c_{i+1})}) \big| \mathbf{x}_i^{(k)} \in \mathbf{g} \big] \end{array} \right\} \end{array} \right\rangle \\
\forall X \in \wp(State) : \\
\quad \alpha_g(X) = \bigsqcup_{q \in X} \alpha_g(\{q\})
\end{array} \tag{40}$$

and the concretisation function γ_g, for any $Y = \langle \overline{Y}, \widetilde{Y} \rangle \in A_g$, by:

$$\begin{array}{c}
q = \big\langle \sigma_n,\ \langle c_0, \epsilon_0 \rangle \dots \langle c_n, \epsilon_n \rangle \big\rangle \in \gamma_f(Y) \\
\Leftrightarrow \\
\left\{ \begin{array}{ll} \langle c_0 \dots c_n, \sigma_n, \epsilon_n \rangle \in \overline{Y} & \\ \forall 0 \le i < n : & \langle c_0 \dots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \\ & \langle c_i, \widehat{\sigma}_i, \epsilon_i \rangle \text{ is valid for } \langle c_{i+1}, \epsilon_{i+1} \rangle \end{array} \right.
\end{array} \tag{41}$$

Proposition 13 $\wp(S) \underset{\alpha_g}{\overset{\gamma_g}{\leftrightarrows}} A_g$ *is a Galois connection. It follows that* $\alpha_g \circ \gamma_g \sqsubseteq id$ *and* $\gamma_g \circ \alpha_g \sqsupseteq id$.

Proof. We defined γ_g as $\gamma_g(Y) = \bigcup \{X \mid \alpha_g(X) \sqsubseteq Y\}$, see Proposition 7 of [CC92a]. □

5.2 Forward and Backward Analysis

$post_g^{\mathsf{a}}$ and pre_g^{a} are defined in Tables 8 and 9 for the different instructions. These definitions are based on the same principles as those in Tables 4, 5, 6 and 7, but here now we use both, call strings as well as global variables, to unify possible calling contexts with top contexts. Figures 10 and 11 illustrate the most interesting cases, namely procedure calls and returns.

We have to check that the operators we just defined are correct approximations.

Proposition 14 ($post_g^{\mathsf{a}}, pre_g^{\mathsf{a}}$ are correct approximations of $post, pre$)

$$post_g^{\mathsf{a}} \ \sqsupseteq\ \alpha_g \circ post \circ \gamma_g \tag{44}$$

$$pre_g^{\mathsf{a}} \ \sqsupseteq\ \alpha_g \circ post \circ \gamma_g \tag{45}$$

Proof. Appendix A.3. □

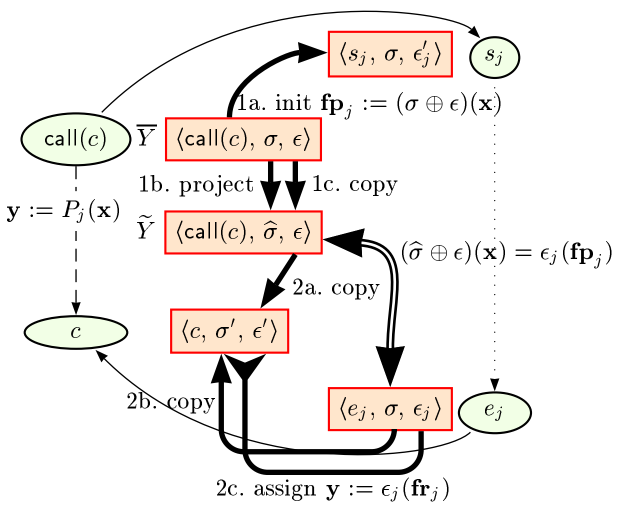

Figure 10: Abstract Postcondition for Procedure Call (1) and Return (2)

$$c \xrightarrow{\langle R \rangle} c' \qquad \overline{post^{\mathsf{a}}_g}(\tau)(Y) = \left\{ \langle \omega \cdot c', \sigma', \epsilon' \rangle \mid \langle \omega \cdot c, \epsilon \rangle \in \overline{Y} \wedge R(\langle \sigma, \epsilon \rangle, \langle \sigma', \epsilon' \rangle) \right\} \tag{42a}$$

$$c \xrightarrow{\langle \mathtt{call}\ \mathbf{y}:=P_j(\mathbf{x}) \rangle} s_j \qquad \overline{post^{\mathsf{a}}_g}(\tau)(Y) = \left\{ \langle \omega \cdot c \cdot s_j, \sigma, \epsilon_j \rangle \;\middle|\; \begin{array}{l} \langle \omega \cdot c, \sigma, \epsilon \rangle \in \overline{Y} \\ \epsilon_j(\mathbf{fp}_j^{(k)}) = (\sigma \oplus \epsilon)(\mathbf{x}^{(k)}) \end{array} \right\} \tag{42b}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x}) \rangle} c \qquad \overline{post^{\mathsf{a}}_g}(\tau)(Y) = \left\{ \langle \omega \cdot c, \sigma', \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle \omega \cdot \mathsf{call}(c) \cdot e_j, \sigma, \epsilon_j \rangle \in \overline{Y} \\ \langle \omega \cdot \mathsf{call}(c), \widehat{\sigma}, \epsilon \rangle \in \widetilde{Y} \\ (\widehat{\sigma} \oplus \epsilon)(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}_j^{(k)}) \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \notin \mathbf{g}] \\ \sigma' = \sigma[\mathbf{y}^{(k)} \mapsto \epsilon(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \in \mathbf{g}] \end{array} \right\} \tag{42c}$$

$$c \xrightarrow{\langle R \rangle} c' \qquad \widetilde{post^{\mathsf{a}}_g}(\tau)(Y) = \widetilde{Y} \tag{42d}$$

$$c \xrightarrow{\langle \mathtt{call}\ \mathbf{y}:=P_j(\mathbf{x}) \rangle} s_j \qquad \widetilde{post^{\mathsf{a}}_g}(\tau)(Y) = \widetilde{Y} \;\cup\; \left\{ \langle \omega \cdot c, \sigma_{|\mathbf{x} \cap GVar}, \epsilon \rangle \mid \langle \omega \cdot c, \sigma, \epsilon \rangle \in \overline{Y} \right\} \tag{42e}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x}) \rangle} c \qquad \widetilde{post^{\mathsf{a}}_g}(\tau)(Y) = \left\{ \langle \omega, \widehat{\sigma}, \epsilon \rangle \in \widetilde{Y} \mid \langle \omega \cdot \omega' \cdot \mathsf{call}(c), \widehat{\sigma}', \epsilon' \rangle \in \widetilde{Y} \right\} \tag{42f}$$

Table 8: Equations Defining $post^{\mathsf{a}}_g$

We get the usual correctness results. Let us define, for any $Y_0, Y \in A_g$:

$$\begin{array}{rclrcl} F^{\mathsf{a}}_g[Y_0](Y) & \stackrel{\text{def}}{=} & Y_0 \sqcup post^{\mathsf{a}}_g(Y) & reach^{\mathsf{a}}_g(Y_0) & \stackrel{\text{def}}{=} & \mathrm{lfp}(F^{\mathsf{a}}_g[Y_0]) \\ G^{\mathsf{a}}_g[Y_0](Y) & \stackrel{\text{def}}{=} & Y_0 \sqcup pre^{\mathsf{a}}_g(Y) & coreach^{\mathsf{a}}_g(Y_0) & \stackrel{\text{def}}{=} & \mathrm{lfp}(G^{\mathsf{a}}_g[Y_0]) \end{array}$$

Theorem 5 (Soundness of the abstract forward and backward analysis)

$$reach^{\mathsf{a}}_g \;\sqsupseteq\; \alpha_g \circ reach \circ \gamma_g \tag{46}$$

$$coreach^{\mathsf{a}}_g \;\sqsupseteq\; \alpha_g \circ coreach \circ \gamma_g \tag{47}$$

Proof. Identical to the proof of Theorem 1. □

5.3 Discussion

In the case with global variables, we cannot have any optimality result. The reason is that the semantics is not distributive any more w.r.t. to contexts. Indeed, we do not model accurately how global variables are transformed by a procedure. We are able to model the relationships between the values of global

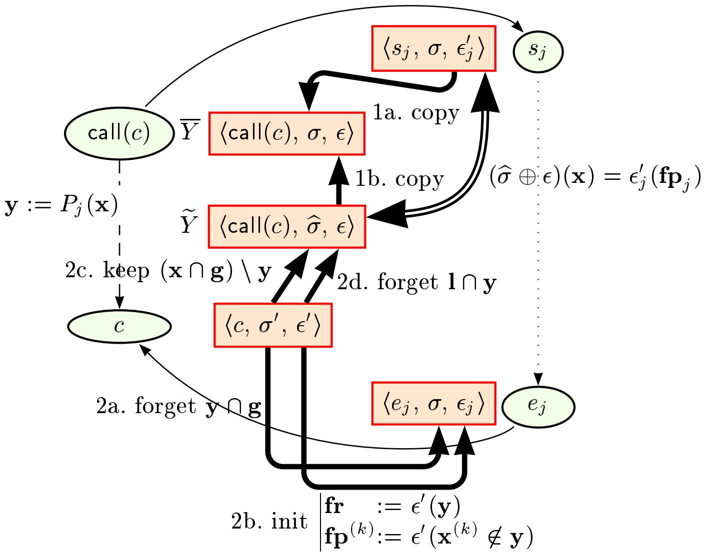

Figure 11: Abstract precondition for procedure call (1) and return (2)

$$c \xrightarrow{\langle R\rangle} c' \qquad \overline{pre^{\mathsf{a}}_g}(\tau)(Y) = \big\{\langle \omega\cdot c, \sigma, \epsilon\rangle \mid \langle \omega\cdot c', \sigma', \epsilon'\rangle \in \overline{Y} \wedge R(\langle\sigma,\epsilon\rangle, \langle\sigma',\epsilon'\rangle)\big\} \tag{43a}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} s_j \qquad \overline{pre^{\mathsf{a}}_g}(\tau)(Y) = \left\{ \langle \omega\cdot c, \widehat{\sigma}, \epsilon\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot c\cdot s_j,\, \sigma, \epsilon_j\rangle \in \overline{Y} \\ \langle \omega\cdot c,\, \widehat{\sigma}, \epsilon\rangle \in \widetilde{Y} \\ (\widehat{\sigma}\oplus\epsilon)(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}^{(k)}) \end{array} \right\} \tag{43b}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} c \qquad \overline{pre^{\mathsf{a}}_g}(\tau)(Y) = \left\{ \begin{array}{l} \langle \omega\cdot \mathsf{call}(c)\cdot e_j, \sigma', \epsilon_j\rangle \\ \quad \left| \begin{array}{lll} \langle \omega\cdot c, \sigma, \epsilon\rangle \in \overline{Y} & & \\ & & \epsilon_j(\mathbf{fr}_j^{(k)}) = (\sigma\oplus\epsilon)(\mathbf{y}^{(k)}) \\ \forall \mathbf{x}^{(k)} \in \mathbf{l}\setminus\mathbf{y}: & & \epsilon_j(\mathbf{fp}_j^{(k)}) = \epsilon(\mathbf{x}^{(k)}) \\ \forall \mathbf{x}^{(k)} \in \mathbf{g}\setminus\mathbf{y}: & & \epsilon_j(\mathbf{fp}_j^{(k)}) = \sigma(\mathbf{x}^{(k)}) \\ \forall z \in \mathbf{g}\setminus\mathbf{y}: & & \sigma'(z) = \sigma(z) \end{array} \right. \end{array} \right\} \tag{43c}$$

$$c \xrightarrow{\langle R\rangle} c' \qquad \widetilde{pre^{\mathsf{a}}_g}(\tau)(Y) = \widetilde{Y} \tag{43d}$$

$$c \xrightarrow{\langle \mathtt{call}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} s_j \qquad \widetilde{pre^{\mathsf{a}}_g}(\tau)(Y) = \big\{\langle \omega, \widehat{\sigma}, \epsilon\rangle \in \widetilde{Y} \mid \langle \omega\cdot\omega'\cdot c, \widehat{\sigma}', \epsilon'\rangle \in \widetilde{Y}\big\} \tag{43e}$$

$$e_j \xrightarrow{\langle \mathtt{ret}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} c \qquad \widetilde{pre^{\mathsf{a}}_g}(\tau)(Y) = \widetilde{Y} \cup \left\{ \langle \omega\cdot \mathsf{call}(c), \widehat{\sigma}, \epsilon\rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot c, \epsilon'\rangle \in \overline{Y} \\ \forall z \notin \mathbf{y} : \epsilon(z) = \epsilon'(z) \\ \forall z \in (\mathbf{x}\cap\mathbf{g})\setminus\mathbf{y} : \widehat{\sigma}(z) = \sigma(z) \end{array} \right\} \tag{43f}$$

Table 9: Equations Defining pre^{a}_g

var g:int	

`proc P():() is`
`var i:int;`
`begin`
$(\mathsf{s_P})$ $\boxed{\{\langle \mathsf{s_P}, ?\rangle\}}$
`g := 0;`
$(\mathsf{c^1_P})$ $\{\langle \mathsf{c^1_P}, g=0\rangle\}$
`for i=0 to 10 do`
$(\mathsf{c^2_P})$ $\{\langle \mathsf{c^2_P}, 0 \leq g \leq 11 \wedge 0 \leq i \leq 10\rangle\}$
`Q();`
$(\mathsf{c^3_P})$ $\{\langle \mathsf{c^3_P}, 0 \leq g \leq 12 \wedge 0 \leq i \leq 10\rangle\}$
`done;`
$(\mathsf{e_P})$ $\{\langle \mathsf{e_P}, 0 \leq g \leq 12 \wedge 0 \leq i \leq 10\rangle\}$
`end`
	`proc Q():() is`
	`begin`
	$(\mathsf{s_Q})$ $\{\langle \mathsf{c^2_P} \cdot \mathsf{s_Q}, 0 \leq g \leq 11\rangle\}$
	`g := g+1;`
	$(\mathsf{e_Q})$ $\{\langle \mathsf{c^2_P} \cdot \mathsf{e_Q}, 0 \leq g \leq 12\rangle\}$
	`end`

Figure 12: Example Illustrating the Approximations induced by A_g.

variables passed as parameters at the entry point of the procedure with their values at the exit point, but we do not do this for global variables which are not passed as parameters.

Fig. 12 depicts an example where our analysis is very imprecise, giving the partial result of the analysis after 12 iterations. Here, as no relation is established between i and g, not only we do not obtain the information that at point $\mathsf{e_P}$ we have $i = 10 \wedge g = 11$, but we are also not able to show that g is bounded. Here, passing g as a parameter to the procedure Q would yield an exact analysis, because we would obtain at the exit point of Q the relation between the value of g at the entry of the procedure, available in the formal parameter of Q, with the value of g at the exit point.

We have then a full range of possibilities: as we add more and more global variables as parameters of procedures, the precision of A_g increases. If we decide to pass all global variables to every procedure, then the abstraction A_g is equivalent to the abstraction A_s applied to a program where all global variables have been made local and passed back and forth to procedures. In this sense, combining A_g with the program transformation consisting in adding new parameters to procedures allows to implement a flexible tradeoff between a rather imprecise but cheap analysis, or a precise or even exact analysis, but which is also more expensive, since more variables have to be taken into account.

6 Combining Stack and Data Abstraction

In the three previous sections, we have presented abstract interpretations for abstracting stacks, two addressing the case of functional programs, and the third dealing directly with programs with global variables. The elements of the considered abstract domains however may still not be finitely representable, either because of infinite data values, or because the size of call strings is not bounded.

Thus two problems are still to be solved:

1. We need to perform a further abstraction in order to obtain an implementable analysis. We consider here A_g as our reference abstract semantics. A_s is just a particular case of A_g, obtained when there is no global variables, and A_f is a further abstraction of A_s.

2. We also need to implement in the further abstraction of A_g the transfer functions for procedure calls and returns, which are more complicated than simple intraprocedural transfer functions as they involve a kind of unification of different (abstract) values, *cf.* (42c) and (43b).

Let D be the union of all data domains in which variables take their values, m and n are respectively the number of global and local variables. Then $GEnv \simeq D^m$ and $LEnv \simeq D^n$. With these notations, the picture is the following one:

$$\begin{array}{ccccc} \wp(\mathit{State}) & \underset{\gamma_g}{\overset{\alpha_g}{\rightleftarrows}} & A_g & \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} & A \\ = & & = & & \\ \wp\left(D^m \times (\mathit{Ctrl} \times D^n)^+\right) & & \left(\wp(\mathit{Ctrl}^+ \times D^{m+n})\right)^2 & & \end{array}$$

(Standard semantics) (Stack abstraction) (Call string & data abstraction)

However, another possibility would be for instance to abstract D with L before performing the stack abstraction

$$\begin{array}{ccccc} \wp(\mathit{State}) & \underset{\gamma_g}{\overset{\alpha_g}{\rightleftarrows}} & \wp(\mathit{State}^\alpha) & \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} & A_g \\ = & & = & & = \\ \wp\left(D^m \times (\mathit{Ctrl} \times D^n)^+\right) & & \wp\left(L^m \times (\mathit{Ctrl} \times L^n)^+\right) & & \left(\wp(\mathit{Ctrl}^+ \times L^{m+n})\right)^2 \\ \text{(Standard semantics)} & & \text{(Data abstraction)} & & \text{(Stack abstraction)} \end{array}$$

The latter approach, where environments are first abstracted, is the one chosen in [SP81, KS92]. Here we will rather suppose that the stack abstraction has been performed, and that the call string and data abstraction has now to be performed in A_g.

We first discuss how to abstract A_g, and then we present the adaptations to be made to an existing intraprocedural analysis in order to implement efficiently procedure returns in forward analysis and procedure calls in backward analysis, which both require to "unify" two abstract values.

6.1 A General Method for Approximating A_g

We propose here solutions for approximating A_g into a computable abstract domain. We assume that we already know how to perform *intraprocedural* analysis, that is, we have a lattice $A_D^{(k)}$ abstracting $\wp(D^k)$ for any k, such that elements of $A_D^{(k)}$ are finitely representable, and such that the operations needed for fixpoint analysis are computable in $A_D^{(k)}$.

We can rewrite A_g as $A_g \simeq \mathit{Ctrl} \to \left(\wp(\mathit{Ctrl}^* \times D^{m+n})\right)^2$, by partitioning A_g according to the control point of the top activation record, so as to associate a set of values to each control point as it is usually done in the analysis of imperative programs. As *Ctrl* is finite, we just need to design an abstract lattice for $\wp(\mathit{Ctrl}^* \times D^{m+n})$ in order to get an abstract lattice for A_g. A natural way to achieve this is to build an abstract lattice for $\wp(\mathit{Ctrl}^* \times D^{m+n})$ by composing abstract domains available for $\wp(\mathit{Ctrl}^*)$ and $\wp(D^{n+m})$. So we need to abstract the call strings lattice $\wp(\mathit{Ctrl}^*)$ by some lattice A_{Ctrl}, as well as a method for combining $A_D^{(m+n)}$ and A_{Ctrl}.

Abstracting Sets of Call Strings. Concerning $\wp(\mathit{Ctrl}^*)$, several abstract domains can be considered. Possible choices for A_{Ctrl} are:

- $\wp(\mathit{Ctrl}^p)$, for some fixed $p \geq 0$: the top p elements of stack tails are exactly represented, the others are completely forgotten;
- $\mathsf{Reg}(\mathit{Ctrl})$, the set of regular languages over the finite alphabet *Ctrl*, together with a suitable widening operator, as the one suggested in [Fer01].
- or some subsets of regular languages: simple regular languages, star-free regular languages, etc., with widening operators if necessary.

It should be noted that this approach for abstracting call strings is more general than the one suggested in the call string approach as described in [SP81], as there the only considered solution is partitioning $\wp(\mathit{Ctrl}^*)$ finitely. Here we do not restrict abstract lattices for $\wp(\mathit{Ctrl}^*)$ neither to finite lattices or even to finite-height lattice, thanks to the use of widening.

Combining Call String and Data Abstractions. Combining different datatypes in abstract interpretation is difficult. Here the datatypes are $Ctrl^*$ and D^k, and it seems difficult to design an *ad-hoc* lattice for their combination. How should we abstract for instance $\wp(Ctrl^* \times \mathbb{Z}^k)$? Instead, we suggest to combine the lattices abstracting each datatype. We could use the tensor product $A_{Ctrl} \otimes A_D^{(k)}$ [Nie85]. However, this product is not finitely representable as soon as either A_{Ctrl} or $A_D^{(k)}$ is not trivial, as discussed in [Jea02]. Instead, we suggest the simple solution of [JHR99, Jea03], which is to take the Cartesian product $A = A_{Ctrl} \times A_D^{(k)}$. In this lattice, relationships between call strings and data values cannot be directly represented: an abstract value is just the conjunction of an invariant on call strings and an invariant on data values. However, partitioning on A can be used to establish relationships between the two components. This technique has been used for combining invariants on Boolean and numerical variables, [JHR99, Jea03], and can be considered as a particular instance of the disjunctive or down-set completion method discussed in [CC92a], where the size of disjuncts is bounded by the size of the partition.

Call Strings and Procedure Inlining. Actually, partitioning $A = A_{Ctrl} \times A_D^{(k)}$ according to call strings allows to perform in the abstract lattice a common program transformation known as *procedure inlining*. Indeed, suppose we partition $\wp(Ctrl^*)$ according to the n top call sites. This equivalent to performing the inlining of procedures up to a depth of n procedure calls. Integrating such a program transformation into the abstract lattice itself allows to integrate it more elegantly and more smoothly with the analysis. One could it apply it during the analysis, possibly on the fly, or by performing an alternation of fixpoint computations and partition refinement as in [JHR99].

6.2 Effective Implementation of Abstract Instructions

Procedure Calls and Returns Implementing procedure calls in forward analysis, and dually procedure returns in backward analysis, is straighforward.

Implementing procedure returns in forward analysis (and dually, procedure calls in backward analysis) may seem difficult at first glance, especially when $\wp(D^{m+n})$ has been abstracted by some lattice $A_D^{(m+n)}$. This corresponds to the *combine* operation used in [SP81, JM82], but which is not explicitely defined.

Let us recall the expression of the postcondition operator for procedure returns in A_g:

$$\overline{Z} = \overline{post^{\mathsf{a}}_g}(e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c)(Y) = \left\{ \langle \omega\cdot c, \sigma', \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle \omega\cdot\mathsf{call}(c)\cdot e_j, \sigma, \epsilon_j\rangle \in \overline{Y} \\ \langle \omega\cdot\mathsf{call}(c), \widehat{\sigma}, \epsilon\rangle \in \widetilde{Y} \\ (\widehat{\sigma}\oplus\epsilon)(\mathbf{x}^{(k)}) = \epsilon_j(\mathbf{fp}_j^{(k)}) \\ \epsilon' = \epsilon[\mathbf{y}^{(k)} \mapsto \epsilon(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \notin \mathbf{g}] \\ \sigma' = \sigma[\mathbf{y}^{(k)} \mapsto \epsilon(\mathbf{fr}_j^{(k)}) \mid \mathbf{y}^{(k)} \in \mathbf{g}] \end{array} \right\} \tag{42c}$$

The problem is that generally one cannot any more manipulate single activation contexts in $A_D^{(m+n)}$. Abstract values rather represent *sets* of such environments.

The solution consists in rewriting the right hand side of (42c) by using a predicate formulation. Let us associate to any $Z \in \wp(Ctrl^+ \times D^{m+n})$ a predicate which is defined as follows:

$$Z(\omega)(\mathbf{g},\mathbf{l}) \;\equiv\; \text{true} \Leftrightarrow \langle \omega, \mathbf{g}, \mathbf{l}\rangle \in Z \tag{48}$$

Then we can rewrite the procedure return as:

$$\overline{Z}(\omega\cdot c)(\mathbf{g},\mathbf{l}) = \exists \widetilde{\mathbf{g}}, \widetilde{\mathbf{l}}, \overline{\mathbf{g}}, \overline{\mathbf{l}}_j : \left\{ \begin{array}{l} \widetilde{Y}\big(\omega\cdot\mathsf{call}(c)\big)(\widetilde{\mathbf{g}},\widetilde{\mathbf{l}}) \;\wedge\; \overline{Y}\big(\omega\cdot\mathsf{call}(c)\cdot e_j\big)(\overline{\mathbf{g}},\overline{\mathbf{l}}_j) \;\wedge\; \overline{\mathbf{fp}}_j = \widetilde{\mathbf{x}} \\ \mathbf{y} = \overline{\mathbf{fr}}_j \;\wedge\; \bigwedge_{z\in\mathbf{g}\setminus\mathbf{y}} z = \overline{z} \;\wedge\; \bigwedge_{z\in\mathbf{l}\setminus\mathbf{y}} z = \widetilde{z} \end{array} \right. \tag{49}$$

Such a formulation is much easier to abstract and to implement. For instance, if $D = \mathbb{Z}$ and $A_D^{(k)}$ is the lattice of octagons [Min01] or the lattice of convex polyhedra [CH78] of dimension k, then all the operations involved in (49) can be directly implemented. This is also the case if $D = \mathbb{B}$ and if one uses Boolean functions on k propositional variables to represent $\wp(\mathbb{B}^k)$. Such functions may typically be represented with BDDs [Bry86] or some of their variants.

Actually, a careful rewriting of (49) allows to perform this operation by considering at most $|\mathbf{g}| + |\mathbf{l}| + |\mathbf{fp}| + |\mathbf{fr}|$ dimensions at the same time, instead of $3|\mathbf{g}| + 2|\mathbf{l}| + |\mathbf{l}_j|$. This can be done by performing early quantifications of unused variables in $\widetilde{Y}\big(\omega{\cdot}\mathsf{call}(c)\big)$ and $\overline{Y}\big(\omega{\cdot}\mathsf{call}(c) \cdot e_j\big)$, as well as suitable variable renaming, *before* building the conjunction of $\widetilde{Y}\big(\omega \cdot \mathsf{call}(c)\big)$ and $\overline{Y}\big(\omega \cdot \mathsf{call}(c) \cdot e_j\big)$. Such implementation aspects are important, as the complexity of *relational* abstract lattices for data values is often high in the number of dimensions considered: this complexity is for instance exponential both for convex polyhedra and BDDs, and cubic for octagons.

Intraprocedural instructions. The other kind of transitions are intraprocedural transitions and do not raise particular problems, since they can be treated using standard techniques developed for intraprocedural analysis. Appendix B gives full definitions.

However efficient implementation is important. Although we *modeled* in an abstract way any intraprocedural instruction by a relating input and output environment, *cf.* rule (Intra) in Figure 3, effective implementation should be done differently. Indeed, using predicate formulation given in appendix B for intraprocedural instructions would be

- inefficient: it requires the introduction of new dimensions, intersection in a space of higher dimensions, and then elimination of the new dimension; the two later operations might be very costly;
- and also unprecise, with lattices which are only partially relational; for instance, octagons can represent accurate relations only between two variables.

The right way to do it is to use direct update of the abstract value Z when R corresponds to an assignement of a variable by an expression, which is possible in most cases, and intersection of Z with a condition converted to an abstract value when R corresponds to a conditional.

This observation is also relevant for efficiently implementing function composition in the functional approach of [SP81] and in [KS92]. Indeed, only composition of a general predicate transformer (i.e., a relation) with a predicate transformer corresponding to an elementary statement is used in their method, and this can be implemented by updating the general predicate transformer according to the semantics of the elementary statement.

7 Related Work

Let us detail more the related work, following the distinction made in the introduction between the functional approach and the operational approach to interprocedural analysis. We first focus on general methods, and then present their applications or methods tailored for more specific analysis like pointer aliasing or constant propagation analysis.

Functional Approach. Concerning the functional approach, an early paper is [CC77b]. The authors start with the standard (functional) semantics of the considered programs. The principle is to express the predicate transformer of each statement (single and compound) of the program in function of the predicate transformers of its substatements, using relation composition (which involves conjunction and existential quantification). This yields a set of fixpoint equations between predicate transformers. Abstract interpretation may then be used if necessary to abstract predicate transformers and to transpose the fixpoint equations in a simpler domain, where the iterative computation of the fixpoint is possible. The result of the analysis is then a (possibly abstract) predicate transformer for each instruction block, including full functions. The general framework of abstract interpretation provides directly correctness results. However, this technique does not directly answer the question: "what are the possible states in this control point ?" for any control point.

[SP81, KS92] answer this question, by proceeding in two steps. They differ from each other by the fact that [KS92] handles local variables.

- The first step follows similar aims as the technique of [CC77b] sketched above, but proceeds in a different way. Here the unknowns are predicate transformers associated to control points of the programs which relate the state of variables of the containing procedure at its entry point with the state of variables at the given control point, so that the predicate transformer associated to the exit point of a function corresponds then to the semantics function (of the procedure). An advantage of this technique compared to [CC77b] is that fixpoint equations involve only composition of general predicate transformers with predicate transformers associated to single statements of the program, which can often be implemented much more efficiently, without using conjunction and existential quantification, as mentioned in Section 6.

- The second step uses the result of the first step to compute possible values of variables at each control point, given the value of the variables at the entry point of the main function. This requires to solve a new set of fixpoint equations.

Our first, simple, stack abstraction presented in Section 3 shares several similarities with [SP81, KS92], as the hypothesis that formal parameters are not modified allows us to relate at any control point the current values of variables with the values of formal parameters at the entry point of the containing procedure. However, one of its advantages is to merge these two fixpoint computations into one and to obtain a better result whenever the lattice of properties is not distributive, as illustrated by Example 1 in the introduction. Another difference is that [SP81, KS92] need to introduce the notion of *valid interprocedural path* to prove correctness and optimality results, whereas we can prove such results by directly relating two semantics. We argue that this is more elegant and simpler simple, as *valid interprocedural paths* are not easy to manipulate and their relationship with the semantics of the program is not immediate, when the control depends on the values of variables through conditionals.

Operational Approach. The difficulty of explicitly representing full stack contents has lead to fewer attempts in such a direction. One of them is the "call-strings approach" presented in [SP81], but in the very restrictive case of programs without local variables, so that stacks are so-called *strings* of control points. The method can however be applied to programs with local variables, but that means that their value is completely forgotten when pushed on the stack.

[JM82] suggests a more general method. A stack is represented by a pair composed of a *token* abstracting all but the top activation record, and an environment for the top activation record. However, tokens are not given a lattice structure, tokens are just labelling top activation records and should form a finite set in practical implementations. Such tokens can be seen as a *partition* of the set of tails of stacks. In comparison, in our abstract lattice(s) the component $\widetilde{\alpha}(X) \subseteq \mathit{Ctrl}^+ \times D^n$ representing tails of stacks, further abstracted, can lead to an implementable lattice which may be not only infinite but even of infinite height. [CC92b] shows that using such lattices often leads to more precise results than those obtained when one is restricted to lattices of finite height.

Bourdoncle attacks much more complex programs in [Bou90], as he considers Pascal programs with reference parameter passing, which introduces aliasing on the stack (i.e., the fact that several variables may refer to the same location in the stack), nested procedure definitions, and last but not least, procedural parameters (i.e., variables representing procedures) in [Bou92]. The devised solution is however, and not surprisingly, rather complex, even without procedural parameters. It has been implemented in this latter case, using the lattice of intervals for integer variables [CC77a]. Activation records on stacks are collapsed more severely than in our model, and an accurate analysis of the MacCarthy'91 function requires partial unfolding of the original recursive procedure. As noticed in Section 6, in our case such an unfolding can be integrated directly in the analysis, and automated using partition refinement on the lattice abstracting $\wp(\mathit{Ctrl}^+)$.

A very different proposal was made in [EK99], using pushdown automata. It relies on the essential result that the set of reachable stacks of a pushdown automata is a regular language, that can be represented by finite-state automata [BEM97, FWW97]. The analysed program is converted to a pushdown automaton, with global variables encoded into its control part and local variables encoded both by the control part and the alphabet. The application of this approach to verification is however restricted either to programs manipulating finite-state variables, or to programs first abstracted to such programs. In other words, only properties belonging to a finite lattice can be analysed. This approach has also been applied to concurrent programs [BET03]. A recent extension relaxes this restriction by allowing some finite-height lattices, such as linear constant propagation [RSJ03]. It represents a very interesting mix of the two approaches: indeed, pushdown automata are here extended by associating weights to transition rules. These weights are composed with the multiplicative operation of a semiring when applying a rule. For data flow analysis applications, this allows to encode the control part of the program in the underlying pushdown automata, as well as properties belonging to a finite lattice, whereas properties belonging to a finite-height lattice can be handled by the weights viewed as predicate transformers.

Interprocedural analysis and verification. The general methods described above have been applied to several data flow analysis and verification problems. We cite some of them, but do not pretend to any overview.

[CC77b] has been applied by [Hal79] to verify recursive programs with integers and by [CI96] to analyse the access to arrays in (non-recursive) interprocedural Fortran programs, using the abstract domain of convex polyhedra [CH78]. [RHS95] can be viewed as an algorithmical implementation of [KS92] using graph reachability techniques, which can be used with finite lattices and distributive data flow functions. The idea is to represent such data flow functions with bipartite graphs connecting atomic elements of the lattice, and to build a so-called exploded supergraph by replacing edges of the CFG by these bipartite graphs. Function composition along paths can then be performed by graph reachability techniques on this supergraph. This approach can be applied among others to all *bit-vector* analyses. An extension [SRH96] allows to tackle some finite-height lattices, like (linear) constant propagation, by adopting a more sophisticated representation of data flow functions. [Mar99] gives an interesting comparison between the functional and the "call-strings" approach of [SP81] for two common data-flow analyses, copy and full constant propagation. The two methods are implemented in PAG [Mar98].

Many interprocedural analysis have been designed with more specific solutions to address the interprocedural aspects, relying on the properties of the specific analysis and practical heuristics. Handling interprocedural aspects is often performed by applying the principles of [JM82], even if the relationship between the standard semantics and the abstract one is not always explicit, nor separated from algorithmical aspects. Among them, pointer and alias analyses has been widely studied, being of primary interest for optimizing compilers. [Deu94] describes a powerful interprocedural alias analysis, which uses for the interprocedural aspects the tokens of [JM82] and also takes advantages of particular properties of the alias analysis.

8 Conclusion

In this paper, we have presented an approach to the verification of imperative programs with recursive procedures and variables over possibly infinite domains. Our method consists of two abstract interpretation steps, where we first abstract the call stack of the program, independently from a second step consisting in abstracting the data using classical intraprocedural abstractions. These two abstraction steps can be composed together to lead to an implementable abstraction which can maintain substantial information on the call-stack of the analysed program. We have proven that in the case of forward analysis, under some assumption on initial states, our first abstraction is optimal, and thus can be considered as a good starting point for the following data abstraction.

Our approach can be seen as combining the two different approaches described in [SP81], yielding an analysis which contains strictly more information. This is particularly useful for starting the analysis at initial states with a non-empty call-stack[15], since the functional approach cannot even represent these states, and our abstraction allows to specify constraints on the activation contexts in the stack-tail. Also, if the abstract domain is not distributive, as for instance convex polyhedrons, Example 1 shows that we can obtain a more precise analysis result. Furthermore, and in contrary to the methods based on call-strings or tokens, our abstraction of call stacks is independent of any abstraction of data, and in particular allows to represent precise information on the values of variables under the top activation record.

Compared to the functional approach [CC77b, SP81, KS92] and also to [JM82], our analysis is more flexible with respect to the treatment of global variables. While the only possible treatment of global variables in the functional approach consists in transforming the program in order to add these variables as parameters to the procedures, our analysis allows additionally to deal with global variables more directly, without requiring to store their values into activation records. While the direct treatment of global variables comes at the expense of a loss in precision, the analysis also becomes less expensive, since the relations to be established for each procedure need to consider less dimensions[16]. Thus it is possible to fine-tune the compromise between precision and cost using our approach by choosing those global variables to be considered as parameters.

An implementation of our analysis is under work.

As future work, we plan to investigate the necessary modifications, in order to obtain a dual exactness result for backward analysis, by introducing a similar parameter freezing mechanisms for the return parameters of a procedure. We also plan to use our abstraction of call stacks in order to check security properties, extending the work of [BJMT01] to take into account the data. The expressivity of the analyzed programs could be improved. Non-local jumps should be easily added, although their use would probably suppress our current optimality results. The extension of our abstraction to programs manipulating pointers to dynamically allocated objects is a more challenging task we want to undertake. Last, allowing reference parameter passing would be very useful to tackle C programs, but here it is much less clear to us whether our general approach can be used. However, concerning this last problem, apart Bourdoncle's work [Bou90], all the other general methods are either described in a too abstract way to lead directly to a practical solution for this, either assume like us that an intraprocedural statement modifies only the top activation record in the stack and so face the same difficulty as us to handle precisely the effect of aliasing on the stack.

References

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model checking. In *Proceedings of CONCUR'97*, volume 1243 of *Lecture Notes in Computer Science*, Amsterdam, 1997. Springer Verlag.

[BET03] Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. A generic approach to the static analysis of concurrent programs with procedures. In *Proceedings of the* 30^{th} *ACM SIGPLAN*

[15] In this case, our analysis is no longer exact.

[16] Roughly speaking, we gain one dimension for every global variable considered directly instead of being passed as a parameter

- *SIGACT Symposium on Principles of Programming Languages (POPL '03)*, New Orleans, 2003. ACM Press.

[BJMT01] Frédéric Besson, Thomas Jensen, Daniel Le Métayer, and Tommy Thorn. Model checking security properties of control flow graphs. *Journal of Computer Security*, 9:217 – 250, 2001.

[Bou90] François Bourdoncle. Interprocedural abstract interpretation of block structured languages with nested procedures, aliasing and recursivity. In Pierre Deransart and Jan Maluszynski, editors, *Proceedings of the 2^{nd} International Workshop on Programming Language Implementation and Logic Programming*, volume 456 of *Lecture Notes in Computer Science*, pages 307 – 323, Linköping, August 1990. Springer Verlag.

[Bou92] F. Bourdoncle. Sémantique des langages impératifs d'ordre supérieur et interprétation abstraite. Thesis Ecole Polytechnique, Paris, 1992.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–692, 1986.

[CC77a] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4^{th} ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL '77)*, pages 238 – 252, Los Angeles, January 1977. ACM Press.

[CC77b] Patrick Cousot and Radhia Cousot. Static determination of dynamic properties of recursive procedures. In E. J. Neuhold, editor, *Proceedings of the IFIP Conference on Formal Description of Programming Concepts*, pages 237 – 277, St. Andrews, 1977. North Holland Publishing Company.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. (The editor of Journal of Logic Programming has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.di.ens.fr/~cousot`).

[CC92b] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In M. Bruynooghe and M. Wirsing, editors, *PLILP'92*, Leuven (Belgium), January 1992. LNCS 631, Springer Verlag.

[CH78] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the 5^{th} Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84 – 97, Tuscon, 1978. ACM Press.

[CI96] Béatrice Creusillet and François Irigoin. Interprocedural array region analyses. *International Journal of Parallel Programming*, 24(6), 1996.

[Deu94] Alain Deutsch. Interprocedural may-alias analysis for pointers: Beyond k-limiting. In *Proceedings of the 23^{th} ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'94)*. ACM Press, June 1994.

[EK99] Javier Esparza and Jens Knoop. An automata-theoretic approach to interprocedural dataflow analysis. In Wolfgang Thomas, editor, *Proceedings of the 2^{nd} International Conference on Foundations of Software Science and Computation Structure (FoSSaCS '99)*, volume 1578 of *Lecture Notes in Computer Science*, pages 14 – 30, Amsterdam, March 1999. Springer Verlag.

[Fer01] Jérôme Feret. Abstract interpretation-based static analysis of mobile ambients. In 8^{th} *International Static Analysis Symposium (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science (LNCS)*, 2001.

[FWW97] Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems. *Electronic Notes in Theoretical Computer Science*, 9, 1997.

[Hal79] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de troisième cycle, Université de Grenoble, march 1979.

[Jea02] B. Jeannet. Representing and approximating transfer functions in abstract interpretation of hetereogeneous datatypes. In *Static Analysis Symposium, SAS'02*, volume 2477 of *Lecture Notes in Computer Science (LNCS)*, Madrid (Spain)), September 2002.

[Jea03] B. Jeannet. Dynamic partitioning in linear relation analysis. Application to the verification of reactive systems. *Formal Methods in System Design*, 23(1), July 2003.

[JHR99] B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic partitioning in analyses of numerical properties. In *Static Analysis Symposium, SAS'99*, volume 1694 of *Lecture Notes in Computer Science (LNCS)*, Venezia (Italy), September 1999.

[JM82] Neil D. Jones and Steven S. Muchnick. A flexible approach to interprocedural data flow analysis and programs with recursive data structures. In *Proceedings of the* 9^{th} *ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'82)*, 1982.

[KS92] J. Knoop and B. Steffen. The interprocedural coincidence theorem. In P. Pfahler U. Kastens, editor, *Proceedings of the 4th International Conference on Compiler Construction (CC'92), Paderborn (Germany)*, volume 641 of *Lecture Notes in Computer Science (LNCS)*, pages 125–140, Heidelberg, Germany, 1992. Springer-Verlag.

[Mar98] Florian Martin. PAG – an efficient program analyzer generator. *International Journal on Software Tools for Technology Transfer*, 2(1):46–67, 1998.

[Mar99] Florian Martin. Experimental comparison of call string and functional approaches to interprocedural analysis. In *8th International Conference on Compiler Construction*, volume 1575 of *Lecture Notes in Computer Science (LNCS)*, 1999.

[Min01] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE. IEEE CS Press, October 2001.

[Nie85] F. Nielson. Tensor products generalize the relational data flow analysis method. In *Fourth Hungarian Computer Science Conference*, 1985.

[RHS95] Tom Reps, Susan Horwitz, and Mooly Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Proceedings of the* 22^{th} *ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'95)*. ACM Press, January 1995.

[RSJ03] T. Reps, S. Schwoon, and S. Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *Static Analysis Symposium, SAS'03*, volume 2694 of *Lecture Notes in Computer Science (LNCS)*, June 2003.

[SP81] M. Sharir and A. Pnueli. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 7, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[SRH96] Mooly Sagiv, Tom Reps, and Susan Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. *Theoretical Computer Science*, 167, 1996.

A Proofs

This appendix contains the proofs omitted from the main part of the paper, namely the correctness proofs of the three abstractions A_f, A_s and A_g, which can be proven by inspection of the relevant definitions.

A.1 Proof of Proposition 7

A.1.1 Proof of (16)

We prove first (16), namely $post^{\mathsf{a}}_f \sqsupseteq \alpha_f \circ post \circ \gamma_f$. We consider each transition τ separately, and prove that $post^{\mathsf{a}}_f(\tau) \sqsupseteq \alpha_f \circ post(\tau) \circ \gamma_f$.

We refer to Tables 3 and 4. Since $GVar = \emptyset$, we need not consider global environments σ as well as conditions related to them. Also, we implicitly assume in the following that $n \geq 0$.

$\tau = c \xrightarrow{\langle R \rangle} c'$**:** According to (3a) and (11), we have:

$$
\begin{aligned}
&post(\tau) \circ \gamma_f(Y) \\
&\quad = \left\{ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c', \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle c_0, \epsilon_0 \rangle \dots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c, \epsilon \rangle \in \gamma_f(Y) \\ R(\epsilon, \epsilon') \end{array} \right\} \\
&\quad = \left\{ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c', \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle c, \epsilon \rangle \in \overline{Y} \\ \forall 0 \leq i < n : \langle c_i, \epsilon_i \rangle \in \widetilde{Y} \\ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c, \epsilon \rangle \text{ is consistent} \\ R(\epsilon, \epsilon') \end{array} \right\} \\
&\alpha_f \circ post(\tau) \circ \gamma_f(Y) \\
&\quad = \left\langle \begin{array}{c} \left\{ \langle c', \epsilon' \rangle \mid \langle c, \epsilon \rangle \in \overline{Y} \wedge R(\epsilon, \epsilon') \right\}, \\ \left\{ \langle c_i, \epsilon_i \rangle \;\middle|\; \begin{array}{l} \exists \langle c_j, \epsilon_j \rangle_{0 \leq j < n} \in \widetilde{Y}, \exists \langle c, \epsilon \rangle \in \overline{Y}, 0 \leq i < n : \\ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c, \epsilon \rangle \text{ is consistent} \end{array} \right\} \end{array} \right\rangle \\
&\quad \sqsubseteq post^{\mathsf{a}}_f(\tau)(Y) \qquad \big(\text{according to (13a) and (13e)}\big)
\end{aligned}
$$

$\tau = c \xrightarrow{\langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle} s_j$**:** According to (3b) and (11),

$$
\begin{aligned}
&post(\tau) \circ \gamma_f(Y) \\
&\quad = \left\{ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n+1}, \epsilon_{n+1} \rangle \;\middle|\; \begin{array}{l} \langle c_0, \epsilon_0 \rangle \dots \langle c_n, \epsilon_n \rangle \in \gamma_f(Y) \\ c_n = c \wedge c_{n+1} = s_j \wedge \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \end{array} \right\} \\
&\quad = \left\{ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n+1}, \epsilon_{n+1} \rangle \;\middle|\; \begin{array}{l} \langle c_n, \epsilon_n \rangle \in \overline{Y} \\ \forall 0 \leq i < n : \langle c_i, \epsilon_i \rangle \in \widetilde{Y} \\ \langle c_0, \epsilon_0 \rangle \dots \langle c_n, \epsilon_n \rangle \text{ is consistent} \\ c_n = c \wedge c_{n+1} = s_j \wedge \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \end{array} \right\} \\
&\alpha_f \circ post(\tau) \circ \gamma_f(Y) \\
&\quad = \left\langle \begin{array}{c} \left\{ \langle s_j, \epsilon_{n+1} \rangle \mid \langle c, \epsilon_n \rangle \in \overline{Y} \wedge \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \right\}, \\ \left\{ \langle c, \epsilon_n \rangle \in \overline{Y} \right\} \cup \left\{ \langle c_i, \epsilon_i \rangle \;\middle|\; \begin{array}{l} \exists \langle c_j, \epsilon_j \rangle_{0 \leq j < n} \in \widetilde{Y}, \exists \langle c, \epsilon \rangle \in \overline{Y}, 0 \leq i < n : \\ \langle c_0, \epsilon_0 \rangle \dots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c, \epsilon \rangle \text{ is consistent} \end{array} \right\} \end{array} \right\rangle \\
&\quad \sqsubseteq post^{\mathsf{a}}_f(\tau)(Y) \qquad \big(\text{according to (13b) and (13d)}\big)
\end{aligned}
$$

$\tau = e_j \xrightarrow{\langle \texttt{ret}\, \mathbf{y}:=P_j(\mathbf{x})\rangle} c$: According to (3c) and (11),

$$
\begin{aligned}
& post(\tau) \circ \gamma_f(Y) \\
&= \left\{ \langle c_0, \epsilon_0\rangle \ldots \langle c'_n, \epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle c_0, \epsilon_0\rangle \ldots \langle c_{n+1}, \epsilon_{n+1}\rangle \in \gamma_f(Y) \\ c'_n = c \wedge c_n = \mathsf{call}(c) \wedge c_{n+1} = e_j \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}_j^{(k)})] \end{array} \right\} \\
&= \left\{ \langle c_0, \epsilon_0\rangle \ldots \langle c'_n, \epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle c_{n+1}, \epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \langle c_i, \epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0, \epsilon_0\rangle \ldots \langle c_{n+1}, \epsilon_{n+1}\rangle \text{ is consistent} \\ c'_n = c \wedge c_n = \mathsf{call}(c) \wedge c_{n+1} = e_j \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}_j^{(k)})] \end{array} \right\} \\
& \alpha_f \circ post(\tau) \circ \gamma_f(Y) \\
&= \left\langle \begin{array}{c} \left\{ \langle c, \epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle \mathsf{call}(c), \epsilon_n\rangle \in \widetilde{Y} \wedge \langle e_j, \epsilon_{n+1}\rangle \in \overline{Y} \\ \langle \mathsf{call}(c), \epsilon_n\rangle \text{ is valid for } \langle e_j, \epsilon_{n+1}\rangle \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}_j^{(k)})] \end{array} \right\}, \\ \left\{ \langle c_i, \epsilon_i\rangle \;\middle|\; \begin{array}{l} \exists \langle c_j, \epsilon_j\rangle_{0\le j\le n} \in \widetilde{Y}, \exists \langle c, \epsilon\rangle \in \overline{Y}, 0 \le i < n : \\ c_n = \texttt{call}\,(c) \wedge \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle\langle c, \epsilon\rangle \text{ is consistent} \end{array} \right\} \end{array} \right\rangle \\
&\sqsubseteq post_f^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (13c) and (13e)}\big)
\end{aligned}
$$

So it follows that $\alpha_f \circ post \circ \gamma_f \sqsubseteq post_f^{\mathsf{a}}$.

A.1.2 Proof of (17)

Let us follow the same proof principle for showing equation (17), namely $pre_f^{\mathsf{a}} \sqsupseteq \alpha_f \circ pre \circ \gamma_f$. The cases where the transition τ does not modify add or remove elements from the stack are trivial, as it has been shown for $post_f^{\mathsf{a}}$ above. Let us detail only the procedure calls and returns.

$\tau = c \xrightarrow{\langle \texttt{call}\, \mathbf{y}:=P_j(\mathbf{x})\rangle} s_j$: According to (3b), (5) and (11),

$$
\begin{aligned}
& pre(\tau) \circ \gamma_f(Y) \\
&= \left\{ \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle \;\middle|\; \begin{array}{l} \langle c_0, \epsilon_0\rangle \ldots \langle c_{n+1}, \epsilon_{n+1}\rangle \in \gamma_f(Y) \\ c_n = c \wedge c_{n+1} = s_j \end{array} \right\} \\
&= \left\{ \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle \;\middle|\; \begin{array}{l} \langle c_{n+1}, \epsilon_{n+1}\rangle \in \overline{Y} \\ \forall i \le n : \langle c_i, \epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0, \epsilon_0\rangle \ldots \langle c_{n+1}, \epsilon_{n+1}\rangle \text{ is consistent} \\ c_n = c \wedge c_{n+1} = s_j \end{array} \right\} \\
& \alpha_f \circ pre(\tau) \circ \gamma_f(Y) \\
&= \left\langle \begin{array}{c} \big\{ \langle c, \epsilon_n\rangle \in \widetilde{Y} \mid \langle s_j, \epsilon_{n+1}\rangle \in \overline{Y} \wedge \langle c, \epsilon_n\rangle \text{ valid for } \langle s_j, \epsilon_{n+1}\rangle \big\}, \\ \left\{ \langle c_i, \epsilon_i\rangle \;\middle|\; \begin{array}{l} \exists \langle c_j, \epsilon_j\rangle_{0\le j\le n} \in \widetilde{Y}, \exists \langle s_j, \epsilon\rangle \in \overline{Y}, 0 \le i < n : \\ c_n = c \wedge \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle\langle s_j, \epsilon\rangle \text{ is consistent} \end{array} \right\} \end{array} \right\rangle \\
&\sqsubseteq pre_f^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (14b) and (14e)}\big)
\end{aligned}
$$

$\tau = e_j \xrightarrow{\langle \texttt{ret}\, \mathbf{y}:=P_j(\mathbf{x})\rangle} c$: According to (3c), (5) and (11),

$$pre(\tau) \circ \gamma_f(Y)$$

$$= \left\{ \langle c_0,\epsilon_0\rangle\dots\langle \mathsf{call}(c),\epsilon'_n\rangle\langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0,\epsilon_0\rangle\dots\langle c_n,\epsilon_n\rangle \in \gamma_f(Y) \\ c_n = c \wedge c_{n+1} = e_j \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \end{array} \right\}$$

$$= \left\{ \langle c_0,\epsilon_0\rangle\dots\langle \mathsf{call}(c),\epsilon'_n\rangle\langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0,\epsilon_0\rangle\dots\langle c_n,\epsilon_n\rangle \text{ is consistent} \\ c_n = c \wedge c_{n+1} = e_j \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \end{array} \right\}$$

$$\alpha_f \circ pre(\tau) \circ \gamma_f(Y)$$

$$= \left\langle \begin{array}{c} \left\{ \langle e_j,\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c,\epsilon_n\rangle \in \overline{Y} \wedge \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \end{array} \right\}, \\ \left(\begin{array}{l} \left\{ \langle \mathsf{call}(c),\epsilon'_n\rangle \middle| \langle c,\epsilon_n\rangle \in \overline{Y} \wedge \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \right\} \\ \cup \left\{ \langle c_i,\epsilon_i\rangle \;\middle|\; \begin{array}{l} \exists \langle c_j,\epsilon_j\rangle_{0\le j<n} \in \widetilde{Y}, \exists \langle c,\epsilon_n\rangle \in \overline{Y}, 0 \le i < n : \\ \langle c_0,\epsilon_0\rangle\dots\langle c_{n-1},\epsilon_{n-1}\rangle\langle c,\epsilon_n\rangle \text{ is consistent} \end{array} \right\} \end{array} \right) \end{array} \right\rangle$$

$$\sqsubseteq pre_f^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (14c) and (14d)}\big)$$

So it follows that $\alpha_f \circ pre \circ \gamma_f \sqsubseteq pre_f^{\mathsf{a}}$.

A.2 Proof of Proposition 10

A.2.1 Proof of (33)

We prove first the equation (33), namely $post_s^{\mathsf{a}} \sqsupseteq \alpha_s \circ post \circ \gamma_s$. We consider each transition separately, and prove that $post_s^{\mathsf{a}}(\tau) \sqsupseteq \alpha_s \circ post(\tau) \circ \gamma_s$.

$\tau = c \xrightarrow{\langle R\rangle} c'$**:** According to (3a) and (26) , we have:

$$post(\tau) \circ \gamma_s(Y)$$

$$= \left\{ \langle c_0,\epsilon_0\rangle\dots\langle c_{n-1},\epsilon_{n-1}\rangle\langle c',\epsilon'\rangle \;\middle|\; \begin{array}{l} \langle c_0,\epsilon_0\rangle\dots\langle c_{n-1},\epsilon_{n-1}\rangle\langle c,\epsilon\rangle \in \gamma_s(Y) \\ R(\epsilon,\epsilon') \end{array} \right\}$$

$$= \left\{ \langle c_0,\epsilon_0\rangle\dots\langle c_{n-1},\epsilon_{n-1}\rangle\langle c',\epsilon'\rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_{n-1}c,\epsilon\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_0\dots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0,\epsilon_0\rangle\dots\langle c_{n-1},\epsilon_{n-1}\rangle\langle c,\epsilon\rangle \text{ is consistent} \\ R(\epsilon,\epsilon') \end{array} \right\}$$

$$\sqsubseteq \left\{ \langle c_0,\epsilon_0\rangle\dots\langle c_{n-1},\epsilon_{n-1}\rangle\langle c',\epsilon'\rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_{n-1}c,\epsilon\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_0\dots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ R(\epsilon,\epsilon') \end{array} \right\}$$

Thus:

$$\alpha_s \circ post(\tau) \circ \gamma_s(Y)$$

$$\sqsubseteq \left\langle \begin{array}{c} \left\{ \langle c_0\dots c_{n-1}c',\epsilon'\rangle \mid \langle c_0\dots c_{n-1}c,\epsilon\rangle \in \overline{Y} \wedge R(\epsilon,\epsilon') \right\}, \\ \left\{ \langle c_0\dots c_i,\epsilon_i\rangle \in \widetilde{Y} \right\} \end{array} \right\rangle$$

$$= post^{\mathsf{a}}_s(\tau)(Y) \qquad \big(\text{according to (13a) and (13e)}\big)$$

$\tau = c \xrightarrow{\langle \mathtt{call}\, \mathbf{y}{:=}P_j(\mathbf{x})\rangle} s_j$: According to (3b) and (26),

$$
\begin{aligned}
&post(\tau) \circ \gamma_s(Y)\\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \in \gamma_s(Y) \\ c_n = c \wedge c_{n+1} = s_j \wedge \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = \epsilon_n(\mathbf{x}^{(k)}) \end{array} \right\}\\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \text{ is consistent} \\ c_n = c \wedge c_{n+1} = s_j \wedge \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = \epsilon_n(\mathbf{x}^{(k)}) \end{array} \right\}\\
&\sqsubseteq \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ c_n = c \wedge c_{n+1} = s_j \wedge \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = \epsilon_n(\mathbf{x}^{(k)}) \end{array} \right\}\\
&\alpha_s \circ post(\tau) \circ \gamma_s(Y)\\
&\sqsubseteq \left\langle \begin{array}{c} \big\{ \langle c_0 \ldots c_{n-1} c s_j,\epsilon_{n+1}\rangle \mid \langle c_0 \ldots c_{n-1} c,\epsilon_n\rangle \in \overline{Y} \wedge \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = \epsilon_n(\mathbf{x}^{(k)}) \big\}, \\ \big\{ \langle c_0 \ldots c_{n-1} c,\epsilon_n\rangle \in \overline{Y} \big\} \;\cup\; \big\{ \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \big\} \end{array} \right\rangle\\
&= post^{\mathsf{a}}_s(\tau)(Y) \qquad \big(\text{according to (13b) and (13d)}\big)
\end{aligned}
$$

$\tau = e_j \xrightarrow{\langle \mathtt{ret}\, \mathbf{y}{:=}P_j(\mathbf{x})\rangle} c$: According to (3c) and (26),

$$
\begin{aligned}
&post(\tau) \circ \gamma_s(Y)\\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c'_n,\epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \in \gamma_s(Y) \\ c'_n = c \wedge c_n = \mathsf{call}(c) \wedge c_{n+1} = e_j \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}^{(k)}_j)] \end{array} \right\}\\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c'_n,\epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1},\epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \text{ is consistent} \\ c'_n = c \wedge c_n = \mathsf{call}(c) \wedge c_{n+1} = e_j \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}^{(k)}_j)] \end{array} \right\}\\
&\sqsubseteq \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c'_n,\epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1},\epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_n,\epsilon_n\rangle \text{ is valid for } \langle c_{n+1},\epsilon_{n+1}\rangle \\ c'_n = c \wedge c_n = \mathsf{call}(c) \wedge c_{n+1} = e_j \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}^{(k)}_j)] \end{array} \right\}\\
&\alpha_s \circ post(\tau) \circ \gamma_s(Y)\\
&\sqsubseteq \left\langle \begin{array}{c} \left\{ \langle c_0 \ldots c'_n,\epsilon'_n\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n,\epsilon_n\rangle \in \widetilde{Y} \wedge \langle c_0 \ldots c_{n+1},\epsilon_{n+1}\rangle \in \overline{Y} \\ \langle c_n,\epsilon_n\rangle \text{ is valid for } \langle c_{n+1},\epsilon_{n+1}\rangle \\ c'_n = c \wedge c_n = \mathsf{call}(c) \wedge c_{n+1} = e_j \\ \epsilon'_n = \epsilon_n[\mathbf{y}^{(k)} \mapsto \epsilon_{n+1}(\mathbf{fr}^{(k)}_j)] \end{array} \right\}, \\ \big\{ \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \mid 0 \le i < n \wedge \langle c_0 \ldots c_n,\epsilon_n\rangle \in \overline{Y} \wedge c_n = \mathsf{call}(c) \big\} \end{array} \right\rangle\\
&= post^{\mathsf{a}}_s(\tau)(Y) \qquad \big(\text{according to (13c) and (13e)}\big)
\end{aligned}
$$

So it follows that $\alpha_s \circ post \circ \gamma_s \sqsubseteq post^{\mathsf{a}}_s$.

A.2.2 Proof of (34)

Let us follow the same proof principle as for showing (17) above. The cases where the transition τ does not modify add or remove elements from the stack are trivial, as it has been shown for $post^{\mathsf{a}}_s$ before. Let us detail only the procedure calls and returns.

$\tau = c \xrightarrow{\langle \texttt{call}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} s_j$**:** According to (3b) and (5),

$$
\begin{aligned}
& pre(\tau) \circ \gamma_s(Y) \\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \;\middle|\; \begin{array}{l} \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \in \gamma_s(Y) \\ c_n = c \wedge c_{n+1} = s_j \end{array} \right\} \\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1},\epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0,\epsilon_0\rangle \ldots \langle c_{n+1},\epsilon_{n+1}\rangle \text{ is consistent} \\ c_n = c \wedge c_{n+1} = s_j \end{array} \right\} \\
&\sqsubseteq \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1},\epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_n,\epsilon_n\rangle \text{ is valid for } \langle c_{n+1},\epsilon_{n+1}\rangle \\ c_n = c \wedge c_{n+1} = s_j \end{array} \right\} \\
& \alpha_s \circ pre(\tau) \circ \gamma_s(Y) \\
&\sqsubseteq \left\langle \begin{array}{l} \left\{ \langle c_0 \ldots c_n,\epsilon_n\rangle \in \widetilde{Y} \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1},\epsilon_{n+1}\rangle \in \overline{Y} \wedge \langle c_0 \ldots c_n,\epsilon_n\rangle \in \widetilde{Y} \\ \langle c_n,\epsilon_n\rangle \text{ valid for } \langle c_{n+1},\epsilon_{n+1}\rangle \wedge c_n = c \wedge c_{n+1} = s_j \end{array} \right\}, \\ \left\{ \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \;\middle|\; 0 \le i < n \wedge \langle c_0 \ldots c_{n+1},\epsilon_n\rangle \in \overline{Y} \wedge c_n = c \wedge c_{n+1} = s_j \right\} \end{array} \right\rangle \\
&= pre^{\mathsf{a}}_s(\tau)(Y) \qquad \big(\text{according to (14b) and (14e)}\big)
\end{aligned}
$$

$\tau = e_j \xrightarrow{\langle \texttt{ret}\,\mathbf{y}:=P_j(\mathbf{x})\rangle} c$**:** According to (3c) and (5),

$$
\begin{aligned}
& pre(\tau) \circ \gamma_s(Y) \\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle \mathsf{call}(c),\epsilon'_n\rangle\langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \in \gamma_s(Y) \\ c_n = c \wedge c_{n+1} = e_j \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \end{array} \right\} \\
&= \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle \mathsf{call}(c),\epsilon'_n\rangle\langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_0,\epsilon_0\rangle \ldots \langle c_n,\epsilon_n\rangle \text{ is consistent} \\ c_n = c \wedge c_{n+1} = e_j \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \end{array} \right\} \\
&\sqsubseteq \left\{ \langle c_0,\epsilon_0\rangle \ldots \langle \mathsf{call}(c_n),\epsilon'_n\rangle\langle c_{n+1},\epsilon_{n+1}\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0 \le i < n : \langle c_0 \ldots c_i,\epsilon_i\rangle \in \widetilde{Y} \\ c_n = c \wedge c_{n+1} = e_j \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \end{array} \right\}
\end{aligned}
$$

$$
\begin{aligned}
&\alpha_s \circ\, pre(\tau) \circ \gamma_s(Y) \\
&\sqsubseteq \left\langle \begin{array}{c}
\left\{ \langle c_0 \ldots \mathsf{call}(c) c_{n+1}, \epsilon_{n+1} \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n, \epsilon_n \rangle \in \overline{Y} \wedge c_n = c \wedge c_{n+1} = e_j \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \epsilon_{n+1}(\mathbf{fr}_j^{(k)}) = \epsilon_n(\mathbf{y}^{(k)}) \end{array} \right\}, \\
\left\{ \langle c_0 \ldots \mathsf{call}(c), \epsilon'_n \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n, \epsilon_n \rangle \in \overline{Y} \wedge c_n = c \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \end{array} \right\} \cup \left\{ \begin{array}{c} \langle c_0 \ldots c_i, \epsilon_i \rangle \\ \in \widetilde{Y} \end{array} \right\}
\end{array} \right\rangle \\
&= pre_s^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (14c) and (14d)}\big)
\end{aligned}
$$

So it follows that $\alpha_s \circ pre \circ \gamma_s \sqsubseteq pre_s^{\mathsf{a}}$.

A.3 Proof of Proposition 14

A.3.1 Proof of (44)

We prove first the equation (33), namely $post_g^{\mathsf{a}} \sqsupseteq \alpha_g \circ post \circ \gamma_g$. We consider each transition separately, and prove that $post_g^{\mathsf{a}}(\tau) \sqsupseteq \alpha_g \circ post(\tau) \circ \gamma_g$. In contrary to the proof in Appendixes, we now need to consider the global environments.

$\tau = c \xrightarrow{\langle R \rangle} c'$: According to (3a) and (41):

$$
\begin{aligned}
&post(\tau) \circ \gamma_g(Y) \\
&= \left\{ \big\langle \sigma', \langle c_0, \epsilon_0 \rangle \ldots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c', \epsilon' \rangle \big\rangle \;\middle|\; \begin{array}{l} \big\langle \sigma_n, \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \big\rangle \in \gamma_g(Y) \\ c_n = c \\ R\big(\langle \sigma_n, \epsilon_n \rangle, \langle \sigma', \epsilon' \rangle\big) \end{array} \right\} \\
&= \left\{ \big\langle \sigma', \langle c_0, \epsilon_0 \rangle \ldots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c', \epsilon' \rangle \big\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n, \sigma_n, \epsilon_n \rangle \in \overline{Y} \\ \forall 0 \le i < n : \exists \widehat{\sigma}_i : \\ \quad \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \\ \quad \langle c_i, \widehat{\sigma}_i, \epsilon_i \rangle \text{ valid for } \langle c_{i+1}, \epsilon_{i+1} \rangle \\ c_n = c \\ R\big(\langle \sigma_n, \epsilon_n \rangle, \langle \sigma', \epsilon' \rangle\big) \end{array} \right\} \\
&\sqsubseteq \left\{ \langle c_0, \epsilon_0 \rangle \ldots \langle c_{n-1}, \epsilon_{n-1} \rangle \langle c', \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n-1} c, \sigma, \epsilon \rangle \in \overline{Y} \\ \forall 0 \le i < n : \exists \widehat{\sigma}_i : \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \\ R\big(\langle \sigma, \epsilon \rangle, \langle \sigma', \epsilon' \rangle\big) \end{array} \right\} \\
&\alpha_g \circ\, post(\tau) \circ \gamma_g(Y) \\
&\sqsubseteq \left\langle \begin{array}{c} \big\{ \langle c_0 \ldots c_{n-1} c', \sigma', \epsilon' \rangle \mid \langle c_0 \ldots c_{n-1} c, \sigma, \epsilon \rangle \in \overline{Y} \wedge R\big(\langle \sigma, \epsilon \rangle, \langle \sigma', \epsilon' \rangle\big) \big\}, \\ \big\{ \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \big\} \end{array} \right\rangle \\
&= post_g^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (42a) and (42d)}\big)
\end{aligned}
$$

$\tau = c \xrightarrow{\langle \mathtt{call}\, \mathbf{y} := P_j(\mathbf{x}) \rangle} s_j$: According to (3b) and (41):

$$
\begin{aligned}
&post(\tau) \circ \gamma_g(Y) \\
&= \left\{ \big\langle \sigma_n, \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \langle s_j, \epsilon \rangle \big\rangle \;\middle|\; \begin{array}{l} \big\langle \sigma_n, \langle c_0, \epsilon_0 \rangle \ldots \langle c_n, \epsilon_n \rangle \big\rangle \in \gamma_g(Y) \\ c_n = c \\ \epsilon(\mathbf{fp}_j^{(k)}) = (\sigma_n \oplus \epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\}
\end{aligned}
$$

$$= \left\{ \langle \sigma_n,\, \langle c_0,\epsilon_0\rangle \dots \langle c_n,\epsilon_n\rangle\langle s_j,\epsilon\rangle \rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_n,\sigma_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0\le i<n : \exists\widehat{\sigma}_i : \\ \quad \langle c_0\dots c_i,\widehat{\sigma}_i,\epsilon_i\rangle \in \widetilde{Y} \\ \quad \langle c_i,\widehat{\sigma}_i,\epsilon_i\rangle \text{ valid for } \langle c_{i+1},\epsilon_{i+1}\rangle \\ c_n = c \\ \epsilon(\mathbf{fp}_j^{(k)}) = (\sigma_n\oplus\epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\}$$

$$\sqsubseteq \left\{ \langle \sigma_n,\, \langle c_0,\epsilon_0\rangle \dots \langle c_n,\epsilon_n\rangle\langle s_j,\epsilon\rangle \rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_n,\sigma_n,\epsilon_n\rangle \in \overline{Y} \\ \forall 0\le i<n : \exists\widehat{\sigma}_i : \langle c_0\dots c_i,\widehat{\sigma}_i,\epsilon_i\rangle \in \widetilde{Y} \\ c_n = c \\ \epsilon(\mathbf{fp}_j^{(k)}) = (\sigma_n\oplus\epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\}$$

$$\alpha_g \circ post(\tau)\circ\gamma_g(Y)$$

$$\sqsubseteq \left\langle \begin{array}{c} \left\{ \langle c_0\dots c_n s_j,\sigma_n,\epsilon\rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_n,\sigma_n,\epsilon_n\rangle \in \overline{Y}\wedge c_n=c \\ \epsilon(\mathbf{fp}_j^{(k)}) = (\sigma_n\oplus\epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\}, \\ \left\{\langle c_0\dots c_i,\widehat{\sigma}_i,\epsilon_i\rangle \in \widetilde{Y}\right\} \;\cup\; \left\{\langle c_0\dots c_{n-1}c,\sigma,\epsilon\rangle\in\overline{Y}\right\} \end{array} \right\rangle$$

$$= post_g^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (42b) and (42e)}\big)$$

$\tau = e_j \xrightarrow{\langle \texttt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c$**:** According to (3c) and (41):

$$post(\tau)\circ\gamma_g(Y)$$

$$= \left\{ \langle \sigma',\, \langle c_0,\epsilon_0\rangle \dots \langle c_{n-1},\epsilon_{n-1}\rangle\langle c,\epsilon'_n\rangle \rangle \;\middle|\; \begin{array}{l} \big\langle \sigma,\, \langle c_0,\epsilon_0\rangle\dots\langle c_{n+1},\epsilon_{n+1}\rangle\big\rangle \in \gamma_g(Y) \\ c_n = \mathsf{call}(c)\wedge c_{n+1}=e_j \\ \sigma' = \sigma\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\in GVar\big] \\ \epsilon'_n = \epsilon_n\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\notin GVar\big] \end{array} \right\}$$

$$= \left\{ \langle \sigma',\, \langle c_0,\epsilon_0\rangle \dots \langle c_{n-1},\epsilon_{n-1}\rangle\langle c,\epsilon'_n\rangle \rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_{n+1},\sigma,\epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0\le i\le n : \exists\widehat{\sigma}_i : \\ \quad \langle c_0\dots c_i,\widehat{\sigma}_i,\epsilon_i\rangle \in \widetilde{Y} \\ \quad \langle c_i,\widehat{\sigma}_i,\epsilon_i\rangle \text{ valid for } \langle c_{i+1},\epsilon_{i+1}\rangle \\ c_n = \mathsf{call}(c)\wedge c_{n+1}=e_j \\ \sigma' = \sigma\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\in GVar\big] \\ \epsilon'_n = \epsilon_n\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\notin GVar\big] \end{array} \right\}$$

$$\sqsubseteq \left\{ \langle \sigma',\, \langle c_0,\epsilon_0\rangle \dots \langle c_{n-1},\epsilon_{n-1}\rangle\langle c,\epsilon'_n\rangle \rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_{n+1},\sigma,\epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0\le i\le n : \exists\widehat{\sigma}_i : \langle c_0\dots c_i,\widehat{\sigma}_i,\epsilon_i\rangle \in \widetilde{Y} \\ \langle c_n,\widehat{\sigma}_n,\epsilon_n\rangle \text{ valid for } \langle c_{n+1},\epsilon_{n+1}\rangle \\ c_n = \mathsf{call}(c)\wedge c_{n+1}=e_j \\ \sigma' = \sigma\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\in GVar\big] \\ \epsilon'_n = \epsilon_n\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\notin GVar\big] \end{array} \right\}$$

$$\alpha_g \circ post(\tau)\circ\gamma_g(Y)$$

$$\sqsubseteq \left\langle \begin{array}{c} \left\{ \langle c_0\dots c_{n-1}c,\sigma',\epsilon'\rangle \;\middle|\; \begin{array}{l} \langle c_0\dots c_{n+1},\sigma,\epsilon_{n+1}\rangle \in \overline{Y} \\ \langle c_0\dots c_n,\widehat{\sigma},\epsilon_n\rangle \in \widetilde{Y} \\ (\widehat{\sigma}\oplus\epsilon_n)(\mathbf{x}^{(k)}) = \epsilon_{n+1}(\mathbf{fp}_j^{(k)}) \\ c_n = \mathsf{call}(c)\wedge c_{n+1}=e_j \\ \sigma' = \sigma\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\in GVar\big] \\ \epsilon' = \epsilon_n\big[\mathbf{y}^{(k)}\mapsto\epsilon_{n+1}(\mathbf{fr}_j^{(k)}) \;\big|\; \mathbf{y}^{(k)}\notin GVar\big] \end{array} \right\}, \\ \left\{\langle c_0\dots c_i,\widehat{\sigma}_i,\epsilon_i\rangle \in \widetilde{Y} \;\middle|\; \langle c_0\dots c_{n-1}\mathsf{call}(c),\widehat{\sigma},\epsilon\rangle\in\widetilde{Y}\wedge 0\le i<n\right\} \end{array} \right\rangle$$

$$= post^{\mathsf{a}}_g(\tau)(Y) \qquad \big(\text{according to (42c) and (42f)}\big)$$

A.3.2 Proof of (45)

We follow the same proof principle as for showing (44) above. Since the cases where the transition τ does not modify add or remove elements from the stack are trivial, as it has been shown for $post^{\mathsf{a}}_g$ before, we detail only the procedure calls and returns.

$\tau = c \xrightarrow{\langle \mathtt{call}\, \mathbf{y}{:=}P_j(\mathbf{x})\rangle} s_j$**:** According to (3b), (5) and (41):

$$
\begin{aligned}
& pre(\tau) \circ \gamma_g(Y) \\
&= \left\{ \big\langle \sigma,\, \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle \big\rangle \;\middle|\; \begin{array}{l} \big\langle \sigma,\, \langle c_0, \epsilon_0\rangle \ldots \langle c_{n+1}, \epsilon_{n+1}\rangle \big\rangle \in \gamma_g(Y) \\ c_n = c \wedge c_{n+1} = s_j \\ \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = (\sigma \oplus \epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\} \\
&= \left\{ \big\langle \sigma,\, \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle \big\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1}, \sigma, \epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \exists \widehat{\sigma}_i : \\ \quad \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i\rangle \in \widetilde{Y} \\ \quad \langle c_i, \widehat{\sigma}_i, \epsilon_i\rangle \text{ valid for } \langle c_{i+1}, \epsilon_{i+1}\rangle \\ c_n = c \wedge c_{n+1} = s_j \\ \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = (\sigma \oplus \epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\} \\
&\sqsubseteq \left\{ \big\langle \sigma,\, \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle \big\rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1}, \sigma, \epsilon_{n+1}\rangle \in \overline{Y} \\ \forall 0 \le i \le n : \exists \widehat{\sigma}_i : \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i\rangle \in \widetilde{Y} \\ \langle c_n, \widehat{\sigma}_n, \epsilon_n\rangle \text{ valid for } \langle c_{n+1}, \epsilon_{n+1}\rangle \\ c_n = c \wedge c_{n+1} = s_j \\ \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) = (\sigma \oplus \epsilon_n)(\mathbf{x}^{(k)}) \end{array} \right\} \\
& \alpha_g \circ pre(\tau) \circ \gamma_g(Y) \\
&\sqsubseteq \left\langle \begin{array}{c} \left\{ \langle c_0 \ldots c_{n-1} c, \sigma, \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n+1}, \sigma, \epsilon_{n+1}\rangle \in \overline{Y} \\ \langle c_0 \ldots c_n, \widehat{\sigma}, \epsilon'\rangle \in \widetilde{Y} \\ (\widehat{\sigma} \oplus \epsilon')(\mathbf{x}^{(k)}) = \epsilon_{n+1}(\mathbf{fp}^{(k)}_j) \\ c_n = c \wedge c_{n+1} = s_j \end{array} \right\}, \\ \big\{ \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i\rangle \in \widetilde{Y} \;\big|\; \langle c_0 \ldots c_{n-1} c, \widehat{\sigma}, \epsilon\rangle \in \widetilde{Y} \wedge 0 \le i < n \big\} \end{array} \right\rangle \\
&= pre^{\mathsf{a}}_g(\tau)(Y) \qquad \big(\text{according to (43b) and (43e)}\big)
\end{aligned}
$$

$\tau = e_j \xrightarrow{\langle \mathtt{ret}\, \mathbf{y}{:=}P_j(\mathbf{x})\rangle} c$**:** According to (3c), (5) and (41):

$$
\begin{aligned}
& pre(\tau) \circ \gamma_g(Y) \\
&= \left\{ \big\langle \sigma',\, \langle c_0, \epsilon_0\rangle \ldots \langle \mathsf{call}(c), \epsilon'_n\rangle \langle e_j, \epsilon'\rangle \big\rangle \;\middle|\; \begin{array}{l} \big\langle \sigma,\, \langle c_0, \epsilon_0\rangle \ldots \langle c_n, \epsilon_n\rangle \big\rangle \in \gamma_g(Y) \\ c_n = c \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall z \in \mathbf{g} \setminus \mathbf{y} : \sigma'(z) = \sigma(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon'(\mathbf{fp}^{(k)}_j) = (\sigma \oplus \epsilon_n)(\mathbf{x}^{(k)}) \\ \epsilon'(\mathbf{fr}^{(k)}_j) = (\sigma \oplus \epsilon_n)(\mathbf{y}^{(k)}) \end{array} \right\}
\end{aligned}
$$

$$= \left\{ \langle \sigma', \langle c_0, \epsilon_0 \rangle \ldots \langle \mathsf{call}(c), \epsilon'_n \rangle \langle e_j, \epsilon' \rangle \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n, \sigma, \epsilon_n \rangle \in \overline{Y} \\ \forall 0 \leq i < n : \exists \widehat{\sigma}_i : \\ \quad \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \\ \quad \langle c_i, \widehat{\sigma}_i, \epsilon_i \rangle \text{ valid for } \langle c_{i+1}, \epsilon_{i+1} \rangle \\ c_n = c \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall z \in \mathbf{g} \setminus \mathbf{y} : \sigma'(z) = \sigma(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon'(\mathbf{fp}_j^{(k)}) = (\sigma \oplus \epsilon_n)(\mathbf{x}^{(k)}) \\ \epsilon'(\mathbf{fr}_j^{(k)}) = (\sigma \oplus \epsilon_n)(\mathbf{y}^{(k)}) \end{array} \right\}$$

$$\sqsubseteq \left\{ \langle \sigma', \langle c_0, \epsilon_0 \rangle \ldots \langle \mathsf{call}(c), \epsilon'_n \rangle \langle e_j, \epsilon' \rangle \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n, \sigma, \epsilon_n \rangle \in \overline{Y} \\ \forall 0 \leq i < n : \exists \widehat{\sigma}_i : \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \\ c_n = c \\ \forall z \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'_n(z) = \epsilon_n(z) \\ \forall z \in \mathbf{g} \setminus \mathbf{y} : \sigma'(z) = \sigma(z) \\ \forall \mathbf{x}^{(k)} \notin \mathbf{y} : \epsilon'(\mathbf{fp}_j^{(k)}) = (\sigma \oplus \epsilon_n)(\mathbf{x}^{(k)}) \\ \epsilon'(\mathbf{fr}_j^{(k)}) = (\sigma \oplus \epsilon_n)(\mathbf{y}^{(k)}) \end{array} \right\}$$

$\alpha_g \circ \mathit{pre}(\tau) \circ \gamma_g(Y)$

$$\sqsubseteq \left\langle \begin{array}{c} \left\{ \langle c_0 \ldots c_{n-1} \mathsf{call}(c) e_j, \sigma', \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_n, \sigma, \epsilon_n \rangle \in \overline{Y} \wedge c_n = c \\ \epsilon'(\mathbf{fr}_j^{(k)}) = (\sigma \oplus \epsilon_n)(\mathbf{y}^{(k)}) \\ \forall \mathbf{x}^{(k)} \in \mathbf{l}_{\mathsf{proc}(c)} \setminus \mathbf{y} : \epsilon'(\mathbf{fp}^{(k)}) = \epsilon_n(\mathbf{x}^{(k)}) \\ \forall \mathbf{x}^{(k)} \in \mathbf{g} \setminus \mathbf{y} : \epsilon'(\mathbf{fp}^{(k)}) = \sigma(\mathbf{x}^{(k)}) \\ \forall z \in \mathbf{g} \setminus \mathbf{y} : \sigma'(z) = \sigma(z) \end{array} \right\}, \\ \left\{ \langle c_0 \ldots c_i, \widehat{\sigma}_i, \epsilon_i \rangle \in \widetilde{Y} \right\} \cup \\ \left\{ \langle c_0 \ldots c_{n-1} \mathsf{call}(c), \widehat{\sigma}, \epsilon' \rangle \;\middle|\; \begin{array}{l} \langle c_0 \ldots c_{n-1} c, \sigma, \epsilon \rangle \in \overline{Y} \\ \forall z \notin \mathbf{y} : \epsilon'(z) = \epsilon(z) \\ \forall z \in (\mathbf{x} \cap \mathbf{g}) \setminus \mathbf{y} : \widehat{\sigma}(z) = \sigma(z) \end{array} \right\} \end{array} \right\rangle$$

$$= \mathit{pre}_g^{\mathsf{a}}(\tau)(Y) \qquad \big(\text{according to (43c) and (43f)}\big)$$

B Predicate formulation of transfer functions

We give here the predicate formulations of $\mathit{post}_g^{\mathsf{a}}$ and $\mathit{pre}_g^{\mathsf{a}}$, the abstract postcondition and precondition operators in A_g. The formulas are depicted in Tables 10 and 11. We write $f[x/e]$ the formula f where all occurrences of x have been replaced by e.

It should be noted that intraprocedural instructions $c \xrightarrow{\langle R \rangle} c'$ can be implemented in a much more efficient way than here, in the common case where R models a variable assignment or a conditional. Indeed, in these situations, introducing and then eliminating dimensions is not needed and can be replaced by direct assignment, substitution of a variable by an expression, and intersection.

τ	Z	$=$	$\overline{post^{\mathsf{a}}}(\tau)\big(\langle\widetilde{Y},\overline{Y}\rangle\big)$
$c \xrightarrow{\langle R\rangle} c'$	$Z(\omega\cdot c')(\mathbf{g},\mathbf{l})$	$=$	$\exists \mathbf{g}',\mathbf{l}' : \overline{Y}(\omega\cdot c)(\mathbf{g},\mathbf{l}) \wedge R(\langle\mathbf{g},\mathbf{l}\rangle,\langle\mathbf{g}',\mathbf{l}'\rangle)$
$c \xrightarrow{\langle \mathtt{call}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} s_j$	$Z(\omega\cdot c\cdot s_j)(\mathbf{g},\mathbf{l}_j)$	$=$	$\exists \mathbf{l} \ : \ \overline{Y}(\mathbf{g},\mathbf{l}) \wedge (\mathbf{fp}_j = \mathbf{x})$
$e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c$	$Z(\omega\cdot c)(\mathbf{g},\mathbf{l})$	$=$	$\exists \widetilde{\mathbf{g}},\widetilde{\mathbf{l}},\overline{\mathbf{g}},\overline{\mathbf{l}}_j \ : \begin{cases} \widetilde{Y}(\omega\cdot\mathsf{call}(c))(\widetilde{\mathbf{g}},\widetilde{\mathbf{l}}) \\ \overline{Y}(\omega\cdot\mathsf{call}(c)\cdot e_j)(\overline{\mathbf{g}},\overline{\mathbf{l}}_j) \\ \overline{\mathbf{fp}}_j = \widetilde{\mathbf{x}} \\ \mathbf{y} = \overline{\mathbf{fr}}_j \\ \bigwedge_{z\in\mathbf{g}\setminus\mathbf{y}} z = \overline{z} \\ \bigwedge_{z\in\mathbf{l}\setminus\mathbf{y}} z = \widetilde{z} \end{cases}$
τ	Z	$=$	$\widetilde{post^{\mathsf{a}}}(\tau)\big(\langle\widetilde{Y},\overline{Y}\rangle\big)$
$c \xrightarrow{\langle R\rangle} c'$	$Z(\omega)(\mathbf{g},\mathbf{l})$	$=$	$\widetilde{Y}(\omega)(\mathbf{g},\mathbf{l})$
$c \xrightarrow{\langle \mathtt{call}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} s_j$	$\begin{cases} Z(\omega\cdot c)(\mathbf{g},\mathbf{l}) \\ Z(\omega)(\mathbf{g},\mathbf{l}) \end{cases}$	$=$ $=$	$\overline{Y}(\omega\cdot c)(\mathbf{g},\mathbf{l})$ $\widetilde{Y}(\omega)(\mathbf{g},\mathbf{l})$ (if $\nexists\omega' \ : \ \omega = \omega'\cdot c$)
$e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c$	$Z(\omega)(\mathbf{g},\mathbf{l})$	$=$	$\widetilde{Y}(\omega)(\mathbf{g},\mathbf{l}) \wedge \exists\omega' \in \mathit{Ctrl}^* : \widetilde{Y}(\omega\cdot\omega'\cdot\mathsf{call}(c)) \neq \text{false}$

We have $Z(\omega) = \text{false}$ for any $\omega \in \mathit{Ctrl}^+$ not matching the patterns as shown above.

Table 10: Predicates for $\overline{post^{\mathsf{a}}_g}$ and $\widetilde{post^{\mathsf{a}}_g}$

τ	Z	$=$	$\overline{pre^{\mathsf{a}}}(\tau)(Y)$
$c \xrightarrow{\langle R\rangle} c'$	$Z(\omega\cdot c)(\mathbf{g},\mathbf{l})$	$=$	$\overline{Y}(\omega\cdot c')(\mathbf{g}',\mathbf{l}') \wedge R(\langle\mathbf{g},\mathbf{l}\rangle,\langle\mathbf{g}',\mathbf{l}'\rangle)$
$c \xrightarrow{\langle \mathtt{call}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} s_j$	$Z(\omega\cdot c)(\mathbf{g},\mathbf{l})$	$=$	$\exists\overline{\mathbf{l}}_j \ : \begin{pmatrix} \widetilde{Y}(\omega\cdot c)(\mathbf{g},\mathbf{l})\wedge \\ \overline{Y}(\omega\cdot c\cdot s_j)(\mathbf{g},\overline{\mathbf{l}}_j)\wedge \\ \mathbf{x} = \overline{\mathbf{fp}}_j \end{pmatrix}$
$e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c$	$Z(\omega\cdot\mathsf{call}(c)\cdot e_j)(\mathbf{g},\mathbf{l}_j)$	$=$	$\exists\mathbf{y} : \begin{pmatrix} \exists\mathbf{l} : \overline{Y}(\omega\cdot c)(\mathbf{g},\mathbf{l}) \ \wedge \\ \mathbf{fr}_j = \mathbf{y} \ \wedge \\ \big(\bigwedge_{\mathbf{x}^{(k)}\notin\mathbf{y}} \mathbf{fp}_j^{(k)} = \mathbf{x}^{(k)}\big) \end{pmatrix}$
τ	Z	$=$	$\widetilde{pre^{\mathsf{a}}}(\tau)(Y)$
$c \xrightarrow{\langle R\rangle} c'$	$Z(\omega)(\mathbf{g},\mathbf{l})$	$=$	$\widetilde{Y}(\omega)(\mathbf{g},\mathbf{l})$
$c \xrightarrow{\langle \mathtt{call}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} s_j$	$Z(\omega)(\mathbf{g},\mathbf{l})$	$=$	$\widetilde{Y}(\omega)(\mathbf{g},\mathbf{l}) \wedge \exists\omega' \in \mathit{Ctrl}^* : \widetilde{Y}(\omega\cdot\omega'\cdot c) \neq \text{false}$
$e_j \xrightarrow{\langle \mathtt{ret}\ \mathbf{y}:=P_j(\mathbf{x})\rangle} c$	$\begin{cases} Z(\omega\cdot\mathsf{call}(c))(\mathbf{g},\mathbf{l}) \\ Z(\omega)(\mathbf{g},\mathbf{l}) \end{cases}$	$=$ $=$	$\exists\mathbf{y},\mathbf{g}\setminus\mathbf{x} : \overline{Y}(\omega\cdot c)(\mathbf{g},\mathbf{l}))$ $\widetilde{Y}(\omega)(\mathbf{g},\mathbf{l})$ (if $\nexists\omega' : \omega = \omega'\cdot\mathsf{call}(c)$)

We have $Z(\omega) = \text{false}$ for any $\omega \in \mathit{Ctrl}^+$ not matching the patterns as shown above.

Table 11: Predicates for $\overline{pre^{\mathsf{a}}_g}$ and $\widetilde{pre^{\mathsf{a}}_g}$

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399