A Generic Approach to the Static Analysis of Concurrent Programs with Procedures

Ahmed Bouajjani* Javier Esparza† Tayssir Touili*

1 Introduction

Multithreading is widely used in modern software, but it also enables new kinds of programming errors. Developing static analysis techniques able to detect some of these errors is becoming increasingly important, and has been the subject of a number of recent papers (see [Rin01] for a recent survey). Developing these techniques is also a very challenging task, especially in the interprocedural case, i.e., when the programming language may contain not only synchronization primitives, but also procedures. In this case, a basic result [Ram00] states that even the basic problem of deciding if a given control point can ever be reached is undecidable. Notice that this is the case even if, as usual in a data-flow setting, all if-then-else instructions and loop guards are replaced by nondeterminism; the analysis must only be sensitive to procedure calls and synchronizations.

Due to this negative result, every analysis algorithm must necessarily work with an upper approximation of the set of possible program paths. Such algorithms have been presented in [DS91, BBS00]. These techniques approximate both the effect of procedures and synchronization. In other words, they remain approximate both in the intraprocedural case (no procedures but synchronization) and in the synchronization-free case (no synchronization but procedures). In this paper, we present an analysis technique based on our previous work on model-checking and interprocedural analysis of sequential programs [BEM97, EK99, EHRS00]. Our technique exhibits two main features with respect to previous work. First, it only approximates the effect of synchronizations; procedures are handled in an exact way. Second (and arguably more important), it presents a generic algorithm which allows to perform analysis of different precision and cost. The algorithm is based on the very elegant machinery of [HK99] for solving systems of equations in commutative Kleene algebras.

In order to explain the basic idea of the approach, let us briefly recall the result of [Ram00]. Consider a sequential program with possibly recursive procedures and one of its configurations c. Here, a configuration is a pair consisting of a control point and a stack of activation records containing the information about the procedures which have been called but whose execution is not yet finished. If we look at a possible execution path of the program as a sequence of statements, we can identify the set of all possible program paths reaching c from the initial configuration c_0 with a context-free language $L(c_0, c)$ (and every context-free language can be obtained this way). E.g. take a program that either terminates immediately or executes a statement a, calls itself recursively, executes a statement b, and terminates; let c be the configuration corresponding to termination; we have $L(c_0, c) = \{a^n b^n \mid n \geq 0\}$.

Consider now a concurrent system consisting of two sequential programs communicating with each other by rendezvous, and a configuration (c_1, c_2) of the system. The problem is if (c_1, c_2) is reachable from (c_{01}, c_{02}) when synchronizations are taken into account. A first necessary condition is that c_i be reachable from c_{0i} when synchronizations are not taken into account. We now interpret the execution paths reaching c_1 and c_2 as sequences of *synchronization* statements (i.e. we hide all statements that can be executed by one of the two programs independently of its partner), and obtain in this way two context-free languages $L(c_{01}, c_1), L(c_{02}, c_2)$. Then (c_1, c_2) is reachable if and only if $L(c_{01}, c_1) \cap L(c_{02}, c_2)$ is nonempty, i.e., if there exist sequences of both components offering the same sequence of communications to its partner. Unfortunately, since emptiness of the intersection of context-free languages is undecidable, forward reachability of (c_1, c_2) from (c_{01}, c_{02})

*LIAFA, University of Paris 7, E-mail: {Ahmed.Bouajjani,Tayssir.Touili}@liafa.jussieu.fr

†Lab. for Foundations of Computer Science, University of Edinburgh, E-mail: jav@dcs.ed.ac.uk

is also undecidable. In general, forward or backward reachability between configurations is undecidable.

Our attack on this problem is based on our previous work, which shows how to compute, given an arbitrary *regular* set C of configurations of a sequential program, the sets $pre^*(C)$ and $post^*(C)$ of predecessors and successors of C. We address the following problem: given two sets $C_1 \times C_2$, $C_1' \times C_2'$ of configurations of the concurrent system, is (some configuration of) $C_1 \times C_2$ reachable from (some configuration of) $C_1' \times C_2'$? This amounts to deciding if $(C_1' \times C_2') \cap pre^*(C_1 \times C_2) \neq \emptyset$ or $post^*(C_1' \times C_2') \cap (C_1 \times C_2) \neq \emptyset$. Let us consider the first possibility for this discussion. Our approach consists of computing an *abstraction* of the path language $L(C_i', C_i) = \bigcup_{c \in C_i, c' \in C_i} L(c, c')$, i.e., a set $A(C_i', C_i)$ satisfying $A(C_i', C_i) \supseteq L(C_i', C_i)$. Clearly, emptiness of $A(C_1', C_1) \cap A(C_2', C_2)$ implies emptiness of $L(C_1', C_1) \cap A(C_2', C_2)$. So we check if $A(C_1', C_1) \cap A(C_2', C_2) = \emptyset$, and if so we conclude that $C_1 \times C_2$ is unreachable from $C_1' \times C_2'$.

The problem is to find computable abstract path languages for which emptiness of intersection is decidable. In this paper, we present a generic algorithm to compute *commutative* abstract path languages, meaning that if a word belongs to $A(C', C)$ then all its permutations also belongs to $A(C', C)$. We give several examples of useful abstractions leading to analysis algorithms of different precision and cost.

The paper is organised as follows. Section 2 presents our program model and communicating pushdown systems, our formal model. Section 3 gives examples of abstractions and defines commutative abstractions, the class of abstractions used in the paper. Section 4 reduces the computation of commutative abstractions to the computation of certain sets of predecessors/successors. Section 5 presents the generic algorithm for computing predecessors and discusses its complexity. Finally, section 6 contains conclusions and discusses related work. Proofs and the algorithm for computing successors are presented in an Appendix.

2 The model

2.1 Program model: Flow Graph Systems

Our program model is very similar to the trace flow graph of [DC94] or [NA98]. We represent a sequential program by a *system of flow graphs*, one for each procedure. The nodes of a flow graph correspond to control points in the procedure, and its edges are annotated with statements. An unlabelled statement corresponds to a noop. We allow arbitrary recursion, even mutual procedure calls, between procedures; however, as usual, we require that infinite data types have been abstracted into finite types using standard techniques of abstract interpretation (in the classical dataflow framework, all datatypes are abstracted away). Abstraction usually leads to non-deterministic control flow, which is explicitely allowed. A concurrent program is represented by a tuple of flow graph systems, one for each sequential component.

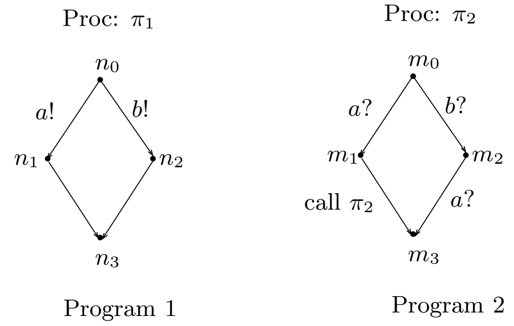

Figure 1: An example

Statements are assignments, calls to other procedures of the same sequential program, or communications with another sequential program. Communication takes place in rendezvous style by means of unidirectional point-to-point channels . A send statement $ch!x$ sends the value of x along channel ch, and a receive statement $ch?y$ waits for a value to be received along channel ch, and binds it to the variable y. We also allow communication of synchronisation signals by means of statements $ch!$ and $ch?$. Figure 1 shows two sequential programs, each one consisting of one procedure, which exchange signals through channels a and b.

2.2 Formal model: Pushdown systems

We mostly follow the notation of [EHRS00]. A *pushdown system* (PDS) is a five-tuple $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$ where P is a finite set of *control locations*, Act is a finite set of *actions*, Γ is a finite *stack alphabet*, and $\Delta \subseteq (P \times \Gamma) \times Act \times (P \times \Gamma^*)$ is a finite set of *transition rules*. If $((p, \gamma), a, (p', w)) \in \Delta$ then we write $\langle p, \gamma \rangle \overset{a}{\hookrightarrow} \langle p', w \rangle$. A *configuration* of

$\mathcal{P}$ is a pair $\langle p, w\rangle$ where $p \in P$ is a control location and $w \in \Gamma^*$ is a *stack content.* c_0 is called the *initial configuration* of $\mathcal{P}$. We assume w.l.o.g. that all the rules of Δ are of the form $\langle p, \gamma\rangle \xhookrightarrow{a} \langle p', w\rangle$ such that $|w| \leq 2$ (in fact, as we shall see below, the PDSs obtained from programs already satisfy this constraint). The set of all configurations is denoted by $\mathcal{C}$. A set C of configurations is *regular* if for each control location $p \in P$ the language $\{w \in Ga^* \mid \langle p, w\rangle \in C\}$ is regular.

For each action a, we define a relation $\xrightarrow{a} \subseteq \mathcal{C} \times \mathcal{C}$ as follows: if $\langle q, \gamma\rangle \xhookrightarrow{a} \langle q', w\rangle$, then $\langle q, \gamma v\rangle \xrightarrow{a} \langle q', wv\rangle$ for every $v \in \Gamma^*$; we say that $\langle q, \gamma v\rangle$ is an *immediate predecessor* of $\langle q', wv\rangle$, and $\langle q', wv\rangle$ an *immediate successor* of $\langle q, \gamma v\rangle$.

The predecessor function $pre^* : 2^{\mathcal{C}} \to 2^{\mathcal{C}}$ of $\mathcal{P}$ is defined as follows: c belongs to $pre^*(C)$ if some successor of c belongs to C. We define $post^*(C)$ similarly.

A *communicating pushdown system* (CPDS) is a tuple $CP = (\mathcal{P}_1, \ldots, P_n)$ of pushdown systems over the same set of actions Act. In order to model communication, we assume that Act contains a special action τ. We denote $Lab = Act \setminus \{\tau\}$, and call its elements *labels*.

A *global configuration* of CP is a tuple $g = (c_1, \ldots, c_n)$ of configurations of $P_1, \ldots P_n$. We extend the relations $\xrightarrow{a}$ to pairs of global configurations as follows. Let $g = (c_1, \ldots, c_n)$ and $g' = (c'_1, \ldots, c'_n)$ be global configurations:

- $g \xrightarrow{\tau} g'$ if there is $1 \leq i \leq n$ such that $c_i \xrightarrow{\tau} c'_i$ and $c'_j = c_j$ for each $j \neq i$;
- $g \xrightarrow{a} g'$ there are indices $i \neq j$ such that $c_i \xrightarrow{a} c'_i$, $c_j \xrightarrow{a} c'_j$, and $c'_k = c_k$ for every $i \neq k \neq j$.

Given a set G of global configurations, we define $pre^*(G)$ and $post^*(G)$ in the obvious way.

2.3 From the program model to the formal model

Given a tuple of flow-graph systems (our model of concurrent programs), we define a corresponding CPDS. Each flow-graph system is assigned a PDS; the CPDS is just the tuple of these PDSs.

For simplicity, we assume that all procedures of the flow-graph system have the same local variables. The corresponding PDS $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$ is defined as follows. P is the set of all possible valuations of the local variables, or a singleton if the program has no variables or all variables have been abstracted away. Act contains the action τ and an action $ch(n)$ for each channel ch and each possible value n that can be transmitted through it. If only signals are transmitted, then the action is ch. Γ is the set of all pairs (n, v), where n is a node of a flow graph, and v is a valuation of the local variables. We take $c_0 = \langle glob_0, (n_0, loc_0)\rangle$, where $glob_0$ and loc_0 are the initial values of the global and local variables, and n_0 is the initial node of the main procedure. Δ contains a set of rules for each program statement, defined next. Let s be the statement labelling an edge of the flow graph system from node n_1 to node n_2. If s is an *assignment*, then it is translated into a set of rules of the form

$$\langle glob, (n_1, loc)\rangle \xhookrightarrow{\tau} \langle glob', (n_2, loc')\rangle.$$

where $glob$ and $glob'$ (loc and loc') are the values of the global (local) variables before and after the assignment. If s is a procedure call, then it is translated into a set of rules of the form

$$\langle glob, (n_1, loc)\rangle \xhookrightarrow{\tau} \langle glob, (m_0, loc')\,(n_2, loc'')\rangle$$

where m_0 is the start node of the called procedure, loc' denotes the initial values of its local variables, and loc'' saves the local variables of the calling procedure. An input $ch?y$ is translated into a set of rules of the form

$$\langle glob, (n_1, loc)\rangle \xhookrightarrow{ch(v)} \langle glob', (n_2, loc')\rangle.$$

where $glob'$ and loc' is the result of assigning the value v to y in $glob$ or loc (depending on whether y is global or local). An output $ch!x$ is translated into a set of rules of the form

$$\langle glob, (n_1, loc)\rangle \xhookrightarrow{ch(v)} \langle glob, (n_2, loc)\rangle.$$

where v is the value of x in $glob$ or loc. Finally, a node n without output edges, corresponding to the return instruction of a procedure, is translated into rules of the form

$$\langle glob, (n, loc)\rangle \xhookrightarrow{\tau} \langle glob, \varepsilon\rangle$$

Procedures which return values can be simulated by introducing an additional global variable and assigning the return value to it.

Notice that, since channels are unidirectional and point to point, we can only have $g \xrightarrow{ch(v)} g'$ if one of the components executes an input and another one an output.

Figure 1 yields a CPDS with two elements. The PDS for the sequential program on the right-hand-side has one single control location p and rules $r_1 : \langle p, m_0\rangle \xhookrightarrow{a} \langle p, m_1\rangle$, $r_2 : \langle p, m_0\rangle \xhookrightarrow{b} \langle p, m_2\rangle$, $r_3 : \langle p, m_1\rangle \xhookrightarrow{\tau} \langle p, m_0 m_3\rangle$, $r_4 : \langle p, m_2\rangle \xhookrightarrow{a} \langle p, m_3\rangle$, and $r_5 : \langle p, m_3\rangle \xhookrightarrow{\tau} \langle p, \epsilon\rangle$.

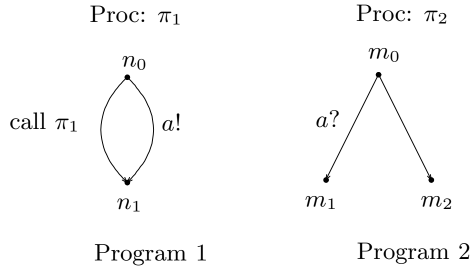

Figure 2: Another example

3 Abstract path languages

In section 3.1 we present three examples of abstractions. In section 3.2 we formally define the class of abstractions that we consider. In section 3.3 we show that the three examples of section 3.1 are indeed instances of the class.

3.1 Some examples

In the rest of the paper, and for the sake of simplicity, we fix a CPDS $CP = (\mathcal{P}_1, \mathcal{P}_2)$. The generalisation of our techniques to n components is straightforward.

As we mentioned in the introduction, given sets C_1, C_1' and C_2, C_2' of configurations of $\mathcal{P}_1, \mathcal{P}_2$, we wish to compute abstractions $A(C_i', C_i)$ of the path languages $L(C_i', C_i) \subseteq Lab^*$. Recall that $L(C_i', C_i)$ contains the sequences of labels (not of actions) leading from C_i' to C_i. Since τ actions are hidden in $L(C_i', C_i)$, we have that $C_1 \times C_2$ is reachable from $C_1' \times C_2'$ if and only if $L(C_1', C_1) \cap L(C_2', C_2)$ is empty. Once the abstractions have been computed, we check if $A(C_1', C_1) \cap A(C_2', C_2)$ is empty, and if the answer is affirmative we conclude that $C_1 \times C_2$ is not reachable from $C_1' \times C_2'$.

We consider three examples of abstract path languages. Notice that we must give a finite representation of $A(C', C)$. So, in fact, we compute an *abstract object* representing $A(C', C)$. For the moment we do this informally, and delay the formal presentation, which uses an standard abstract interpretation framework, to the next subsection.

Forbidden and required sets The abstract object representing $A(C', C)$ is a pair $[F, R]$, where $F, R \subseteq Lab$. F, the *forbidden* set, contains the labels a that do not occur in any sequence of $L(C', C)$. U, the *required* set, contains the labels a that appear in all sequences of $L(C', C)$. $[F, R]$ represents the language of all sequences containing no occurrence of letters in F and at least one occurrence of each letter in R. It is easy to see that the languages represented by $[F_1, R_1]$ and $[F_2, R_2]$ have empty intersection if and only if $F_1 \cap R_2 = \emptyset$ or $R_1 \cap F_2 = \emptyset$.

In Figure 2 we can use this information to conclude that no global configuration with n_1 and m_2 as control locations is reachable: The action a is required to reach n_1, but forbidden if we wish to reach m_2.

Label bitvectors The abstract object representing $A(C', C)$ is a set S of bitvectors $Lab \to \mathbb{B}$. A bitvector b belongs to S if there is a sequence in $L(C', C)$ such that $b(a) = 1$ if a occurs in the sequence and $b(a) = 0$ otherwise. Checking emptyness of the intersection of the languages represented by S_1 and S_2 is equivalent to checking emptiness of their intersection as sets.

In Figure 1 we can conclude that no global configuration with n_3 and m_3 as control locations is reachable: We have $S(n_3) = \{100, 001\}$, while $S(m_3) = \{101\}$ (the components of the bitvectors correspond to the labels a, b, c).

Parikh images The abstract object representing $A(C', C)$ is a (possibly infinite, but, as we shall see in the next section, finitely representable) set M of integer vectors $Lab \to \mathbb{N}$. A mapping m belongs to M if $L(C', C)$ contains a sequence in which each label a occurs exactly $m(a)$ times. We call m the *Parikh image* of the sequence. M represents the language of all sequences whose Parikh-images belong to M. Checking emptiness of the intersection of M_1 and M_2 seen as languages is equivalent to checking emptiness of their intersection as sets.[1]

In Figure 3 we can conclude that no global configuration with n_3 and m_2 as control location is reachable: In order to reach n_3, a must occur an even

[1]It is well-known that Parikh-images of context-free languages are semi-linear sets of integer vectors (i.e., unions of sets of the form $\{\vec{u} + k_1\vec{u}_1 + \cdots + k_n\vec{u}_n \mid k_1, \ldots, k_n \in \mathbb{N}\}$), or equivalently, Presburger arithmetics definable sets.

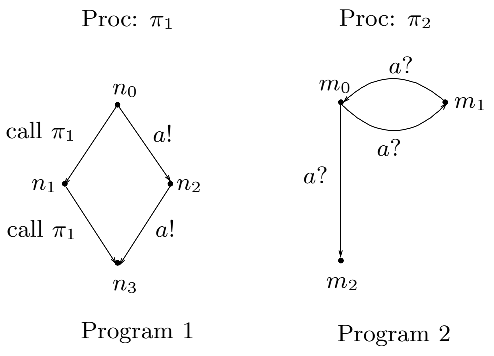

Figure 3: Yet another example

number of times; however, in order to reach m_3, it must occur ann odd number of times.

There are many other examples. For instance, it is possible to mix action sets and Parikh images. For some actions, we retain the information whether they occur in a sequence or not, while for others we count the number of occurrences. Also, in some cases, like the one of Figure 3, it is useful to count modulo a number.

3.2 A formal framework

Let $\mathcal{L}$ be the complete lattice of languages over *Lab*, i.e., $\mathcal{L} = (2^{Lab^*}, \subseteq, \cup, \cap, \emptyset, Lab^*)$.

Formally, an abstraction requires an *abstract lattice* $\mathcal{D} = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$, where D is some abstract domain, and a *Galois connection* (α, γ) between $\mathcal{L}$ and $\mathcal{D}$, i.e., a pair of mappings $\alpha : \mathcal{L} \to \mathcal{D}$ and $\gamma : \mathcal{D} \to \mathcal{L}$ such that

$$\forall x \in \mathcal{L},\ \forall y \in \mathcal{D}.\ \alpha(x) \sqsubseteq y \Longrightarrow x \subseteq \gamma(y)\ .$$

Our aim is to define *commutative abstractions*, a class of abstractions such that (i) the abstract path language $\alpha(L(C'_i, C_i))$ is computable when C_i and C'_i are regular, and (*ii*) emptiness of an intersection is decidable in the domain D.

The carrier D of the lattice of a commutative abstraction is a commutative and idempotent closed semiring. A *commutative idempotent semiring* is a structure $\mathcal{K} = (K, \oplus, \odot, \bar{0}, \bar{1})$, where $\oplus$ and $\odot$ are associative and commutative operations, $\oplus$ is idempotent ($a \oplus a = a$) $\bar{0}$ and $\bar{1}$ are neutral elements for $\oplus$ and $\odot$, respectively, $\bar{0}$ is an annihilator for $\odot$ ($a \odot \bar{0} = \bar{0} \odot a = \bar{0}$), and $\odot$ distributes over $\oplus$. A semiring is *closed* if $\oplus$ can be extended to an operator over countably infinite sets (i.e., countably infinite sums are allowed), and this operator has the same properties as $\oplus$ (it is associative, commutative, idempotent, and $\odot$ distributes over it). As usual, we define $a^0 = \bar{1}$, $a^{n+1} = a \odot a^n$, and $a^\star = \bigoplus_{n \geq 0} a^n$. Adding the $\star$-operation to $\mathcal{K}$ transforms it into a *commutative Kleene algebra*, which in particular allows us to apply the results of [HK99]. The semiring of a commutative abstraction is generated by $\bar{0}$, $\bar{1}$ and an element v_a for each $a \in Lab$.

Intuitively, the abstract operations $\oplus$ and $\odot$ of $\mathcal{K}$ correspond to union and concatenation of languages in the lattice $\mathcal{L}$, with the particularity that $\odot$ "forgets the order" of labels, since it is commutative. $\bar{0}$ and $\bar{1}$ are the abstract objects corresponding to the empty language and to $\{\epsilon\}$, respectively. The element v_a corresponds to the language $\{a\}$.

The order, bottom element and join operation of the abstract lattice are given by $x \leq y$ if $x \oplus y = y$, $\bot = \bar{0}$, and $\sqcup = \oplus$, respectively. Intuitively, the top element $\top \in K$ and the meet operation $\sqcap$ correspond in the lattice $\mathcal{L}$ to Lab^* and to language intersection, respectively.

The Galois connection of a commutative abstraction is given by $\alpha : 2^{Lab^*} \to K$ and $\gamma : K \to 2^{Lab^*}$ such that

$$\begin{aligned} \alpha(L) &= \bigoplus_{a_1 \cdots a_n \in L} v_{a_1} \odot \cdots \odot v_{a_n} \\ \gamma(x) &= \{a_1 \cdots a_n \in 2^{Lab^*} \mid v_{a_1} \odot \cdots \odot v_{a_n} \leq x\} \end{aligned}$$

It can easily be checked that (α, γ) is indeed a Galois connection between $\mathcal{L}$ and $\mathcal{D}$. We require this connection to be *faithful*, meaning that

$$\alpha(L) \sqcap \alpha(L') = \bot \Longrightarrow L \cap L' = \emptyset\ .$$

3.3 Instances of commutative abstractions

We show that the three examples of section 3.1 are commutative abstractions. We consider the forbidden-required case last, since in this case the semiring is a bit more complicated.

Label bitvectors: We consider the semiring $\mathcal{K}$ given by: (1) $K = 2^{[Lab \to \mathbb{B}]}$ is the set of sets of bitvectors indexed by *Lab*; this set is generated by $\{v_a \mid a \in Lab\}$, where v_a is the singleton $\{b_a\}$ given by $b_a(a) = \mathit{true}$, and $b_a(a') = \mathit{false}$ for every $a' \neq a$, (2) $\leq = \subseteq$, (3) $\oplus = \cup$, (4) $\odot = \vee$ (extended to vectors componentwise, and to sets of vectors), (5) $\overline{0} = \emptyset$, and (6) $\overline{1} = \{(\mathit{false}, \ldots, \mathit{false})\}$.

The abstract lattice is obtained by taking $\top = 2^{[Lab \to \mathbb{B}]}$, and $\sqcap = \cap$.

Parikh images: This case is similar to the previous one. Just replace bitvector by integer vector, *false* by 0, *true* by 1, and the or-operation $\vee$ by sum of integer vectors.

Forbidden and required sets: Abstract elements are no longer sets of vectors, but pairs of sets. The semiring $\mathcal{K}$ is defined as follows:

- $K = \{\overline{0}\} \cup \{[F, R] \in 2^{Lab} \times 2^{Lab} \mid F \cap R = \emptyset\}$, i.e., the set of all pairs of sets of actions generated by the elements v_a for each $a \in Act$, where $v_a(a) = [Act \setminus \{a\}, \{a\}]$, augmented with a special element $\overline{0}$,
- $[F_1, R_1] \leq [F_2, R_2]$ iff $F_1 \supseteq F_2$ and $R_1 \supseteq R_2$,
- $[F_1, R_1] \oplus [F_2, R_2] = [F_1 \cap F_2, R_1 \cap R_2]$,
- $[F_1, R_1] \odot [F_2, R_2] = [F_1 \cap F_2, R_1 \cup R_2]$,
- $\overline{0}$ is by definition neutral for $\oplus$ and annihilator for $\odot$,
- $\overline{1} = [Lab, \emptyset]$.

The abstract lattice is obtained by taking $\top = [\emptyset, \emptyset]$, and defining $\sqcap$ by:

- $[F_1, R_1] \sqcap [F_2, R_2] = [F_1 \cup F_2, R_1 \cup R_2]$ if $(F_1 \cap R_2) \cup (F_2 \cap R_1) = \emptyset$,
- $[F_1, R_1] \sqcap [F_2, R_2] = \overline{0}$ otherwise,
- $\overline{0} \sqcap x = x \sqcap \overline{0} = \overline{0}$.

It can be checked that for each of the three structures $\mathcal{K}$ defined above, the associated Galois connection (α, γ) is faithful. Notice also that in the three cases, the problem of emptiness of an intersection (whether $d \sqcap d' = \overline{0}$) is decidable.

4 Computing abstract path languages

We define structures for representing $\alpha(L(C', C))$ and show how their construction can be reduced to a predecessor/successor computation.

4.1 K-predecessors and K-successors

We introduce hereafter a notion of K-configuration and K-transition relation of pushdown systems.

Let us fix from now on a pushdown automaton $\mathcal{P} = (P, Act, \Gamma, c_0, \Delta)$ and a Kleene algebra $(K, \oplus, \odot, \star, \overline{0}, \overline{1})$ corresponding to some commutative abstraction. The set of path-expressions Π_K is the smallest subset of K containing $\overline{1}$ and all basic elements, and closed under $\odot$.

Then, a K-configuration of $\mathcal{P}$ is a pair (c, π), where c is a configuration of $\mathcal{P}$ and π is a path-expression over K.

We extend the pushdown transition relation $\xrightarrow{a}$ to K-configurations as follows: if $c \xrightarrow{a} c'$ for some action a, then $(c, \pi) \rightarrow_K (c', v_a \odot \pi)$ and $(c, v_a \odot \pi) \leftarrow_K (c', \pi)$ for every $\pi \in \Pi_K$; we say that $(c', v_a \odot \pi)$ is an immediate K-successor of (c, π) and $(c, v_a \odot \pi)$ an immediate K-predecessor of (c', v_a). The *forward (backward) reachability relation over* $K \Rightarrow_K$ ($\Leftarrow_K$) is the reflexive and transitive closure of $\rightarrow_K$ ($\leftarrow_K$).

Given a set of configurations C, we define $\mathit{pre}^*_K(C)$ ($\mathit{post}^*_K(C)$) as the set of K-configurations (c, π) such that $(c', \overline{1}) \Leftarrow_K (c, \pi)$ ($(c', \overline{1}) \Rightarrow_K (c, \pi)$) for some $c' \in C$. Intuitively, a K-successor (c', π) of a configuration c memorizes in π a sequence of actions that lead from c to c' (c' from c for K-predecessors). In other words, we have

$$\begin{aligned} \mathit{pre}^*_K(c) &= \{(c', \pi) \mid c' \in \mathit{pre}^*(c),\ \pi \in \alpha(L(c', c))\} \\ \mathit{post}^*_K(c) &= \{(c', \pi) \mid c' \in \mathit{post}^*(c),\ \pi \in \alpha(L(c, c'))\} \end{aligned}$$

4.2 K-multi-automata

The algorithmic approach we propose is based on constructing automata representations of pre^*_K and post^*_K-images of regular sets of configurations.

In [BEM97, EHRS00] *multi-automata* are used to represent possibly infinite sets of configurations. We now extend this notion to K-multi-automata in order to manipulate regular sets of K-configurations.

Definition 4.1 *A K-multi-automaton for the PDS $\mathcal{P}$ (K-MA for short) is a tuple $\mathcal{A} = (\Gamma, Q, \delta, P, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Gamma \times K \times Q$ is a set of transitions, $P \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of final states.*

We define the transition relation $\longrightarrow \subseteq Q \times \Gamma^* \times K \times Q$ *as the smallest relation satisfying:*

- *if $(q, \gamma, e, q') \in \delta$ then $q \xrightarrow{(\gamma, e)} q'$,*
- *$q \xrightarrow{(\epsilon, \overline{1})} q$ for every $q \in Q$, and*

- *if* $q \xrightarrow{(w_1,e_1)} q''$ *and* $q'' \xrightarrow{(w_2,e_2)} q'$ *then* $q \xrightarrow{(w_1w_2, e_1\odot e_2)} q'$.

A run of $\mathcal{A}$ *is a sequence* $p \xrightarrow{(\gamma_1,e_1)} q_1 \cdots \xrightarrow{(\gamma_n,e_n)} q_n$. *The label of this run is* $(\gamma_1\cdots\gamma_n, e_1 \odot \cdots \odot e_n)$. $\mathcal{A}$ *accepts a pair* $(\langle p,w\rangle,\pi)$ *if* $p \xrightarrow{(w,\pi\oplus e)} q$ *for some* $q \in F$ *and some* $e \in K$.

Multi-automata can be seen as a special case of K-multi-automata in which all transitions are labelled by $\bar{1}$. Equivalently, they can be obtained by dropping all the references to K in the definition of K-multi-automata. Given a MA $\mathcal{A}$, we denote by $Conf(\mathcal{A})$ the set of configurations recognized by $\mathcal{A}$. Throughout the paper, we will use the symbols p, p', p'', p_i, etc. to denote initial states of (K-)MAs. Arbitrary states will be denoted by q, q', q'', q_i, etc.

4.3 Reduction to computing pre_K^* and $post_K^*$ images

Let C and C' be two regular sets of configurations represented by means of multi-automata, and consider the problem of computing $\alpha(L(C',C))$. Assume that we are able to compute a K-multi-automaton $\mathcal{A}_{pre_K^*}$ which recognizes the set $pre_K^*(C)$. Then, we can construct a K-multi-automaton $\mathcal{A}_{pre_K^*}^{C'}$ over $\Gamma\times K$ which is the restriction of $\mathcal{A}_{pre_K^*}$ to configurations in C'. This automaton can be straightforwardly constructed by a product between $\mathcal{A}_{pre_K^*}$ and the multi-automaton recognizing C'. [2]

Let us assume w.l.o.g. that all configurations in C' have the same control location, say p (the generalization to several control locations is straightforward). Then, $\alpha(L(C',C))$ is the element of K defined by:

$$\bigoplus\{e_1 \odot \cdots \odot e_n \mid \exists\, p \xrightarrow{(w,e_1\odot\cdots\odot e_n)} q \text{ an accepting run in } \mathcal{A}_{pre_K^*}^{C'}\}$$

A symmetrical approach can be adopted by computing an automaton $\mathcal{A}_{post_K^*}$ recognizing $post_K^*(C')$ and then restricting it to the set of configurations C.

Therefore, we have reduced our problem of computing $\alpha(L(C',C))$ to (i) constructing K-multi-automata recognizing pre_K^* or $post_K^*$-images, and then (ii) constructing elements in K representing the K-languages recognized by finite K-automata.

[2]We consider the smallest set of transitions such that if $q_1 \xrightarrow{(a,e)} q_1'$ and $q_2 \xrightarrow{a} q_2'$ then $(q_1,q_2) \xrightarrow{(a,e)} (q_1',q_2')$.

The problem (ii) can be solved using standard techniques for solving linear equations in closed semirings (e.g., using Floyd-Warshall algorithm, or Gauss elimination). Indeed, this problem is similar to the problem of building regular expressions associated with finite-state automata.

Hence, the main problem to solve is how to compute representations for $post_K^*$ and pre_K^*-images. For this we need the fact that K is a *commutative* Kleene algebra, and we show that we can reduce this problem to solving polynomial (in)equations in commutative Kleene algebras.

5 Computing pre_K^* images

Given a regular set of configurations C recognized by a MA $\mathcal{A}$, we construct a K-MA $\mathcal{A}_{pre_K^*}$ that recognizes $pre_K^*(C)$. In the Appendix we describe another K-MA recognizing $post_K^*(C)$. We assume w.l.o.g. that $\mathcal{A}$ has no transition leading to an initial state.

5.1 A generic procedure

We proceed in two steps. First, we construct a MA $\mathcal{A}_{pre^*}$ recognizing the set $pre^*(C)$. Then, we tag the transitions of $\mathcal{A}_{pre^*}$ with adequate elements of K.

For the first step we use the procedure of [BEM97, EHRS00], which consists of adding new transitions to $\mathcal{A}$ according to the following *saturation rule:*

> If $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle$ and $p' \xrightarrow{w} q$ in the current automaton, add a transition (p,γ,q).

The second step consists of tagging the transitions t of $\mathcal{A}_{pre^*}$ with elements $l(t) \in K$, defined as the smallest elements of K (with respect to $\leq$) satisfying the following inequalities (where $v_\tau = \bar{1}$):

(α_1) if $t = (q,\gamma,q')$ was already a transition of $\mathcal{A}$, then $\bar{1} \leq l(t)$,

(α_2) for every rule $\langle p,\gamma\rangle \xhookrightarrow{a} \langle p',\gamma'\rangle$, and every $q \in Q$,
$$v_a \odot l\big((p',\gamma',q)\big) \leq l\big((p,\gamma,q)\big)$$

(α_3) for every rule $\langle p,\gamma\rangle \xhookrightarrow{a} \langle p',\epsilon\rangle$,
$$v_a \leq l\big((p,\gamma,p')\big)$$

(α_4) for every rule $\langle p,\gamma\rangle \hookrightarrow \langle a\rangle p',\gamma_1\gamma_2$, and every $q \in Q$,
$$\bigoplus_{q'\in Q}\Big(v_a \odot l\big((p',\gamma_1,q')\big) \odot l\big((q',\gamma_2,q)\big)\Big) \leq l\big((p,\gamma,q)\big)$$

Before showing how to compute these tags, let us briefly explain the intuition behind the rules.

- Rules (α_1) express that if c is a configuration from $Conf(\mathcal{A})$, then $(c, \bar{1})$ has to be accepted by $\mathcal{A}_{pre^*_K}$,
- Rules (α_2), (α_3), and (α_4) express that for every rule $r = (p, \gamma) \overset{a}{\hookrightarrow} (p', u)$, and every $q \in Q$, if w is such that $q \xrightarrow{w} q'$ for some final state q', and if $(\langle p', uw\rangle, \pi)$ is a K-predecessor of $Conf(\mathcal{A})$, then so is $(\langle p, \gamma w\rangle, v_a \odot \pi)$.

We have the following theorem, proved in the Appendix.

Theorem 5.1 *Given a PDS $\mathcal{P}$ and a regular set of configurations recognized by a MA $\mathcal{A}$, the K-MA $\mathcal{A}_{pre^*_K}$ described above recognizes precisely $pre^*_K\big(Conf(\mathcal{A})\big)$.*

It remains to show how to compute the tagging $l(t)$ for each transition t. For that let $t_1, t_2, \ldots, t_m$ be an arbitrary numbering of the transitions of $\mathcal{A}_{pre^*}$, and let $x_1, \ldots, x_m$ be associated variables. Then, by (α_1)–(α_4), $l(t_1), \ldots, l(t_m)$ is the smallest solution of a system of inequalities of the form

$$f_i(x_1, \ldots, x_m) \le x_i, \qquad 1 \le i \le m \tag{1}$$

where the f_i's are monomials in $K[x_1, \ldots, x_m]$. (It suffices to observe that two different inequalities of the form $e_1 \le l\big((p,\gamma,q)\big)$ and $e_2 \le l\big((p,\gamma,q)\big)$ can be replaced by the inequality $e_1 \oplus e_2 \le l\big((p,\gamma,q)\big)$.)

This system of equations can be solved using Gauss elimination, plus the elegant formula of [HK99] for solving an equation in one single variable. This is the most adequate procedure for an implementation. However, [HK99] also provides a fixpoint algorithm to directly solve (1). Since this is more adequate for a complexity estimate, we recall it briefly here. We need some preliminaries.

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a vector of indeterminates, let $\mathbf{a} = (a_1, \ldots, a_n)$ be a vector of elements of K, and let $\mathbf{f} = f_1, \ldots, f_m$ be a vector of polynomials with indeterminates $\mathbf{x}$ and coefficients in K. We write $\mathbf{f}(\mathbf{a})$ for the value of $\mathbf{f}$ evaluated at $\mathbf{a}$. The *jacobian* of $\mathbf{f}$, $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$, is the $m \times n$ matrix whose i, j^{th} element is $\frac{\partial f_i}{\partial x_j}(\mathbf{x})$, where $\frac{\partial}{\partial x_i}$ is the *differential operator* given by

- $\frac{\partial x_i}{\partial x_i} = \bar{1}$, $\frac{\partial x_j}{\partial x_i} = \bar{0}$ for $i \neq j$, and $\frac{\partial a}{\partial x_i} = \bar{0}$ for $a \in K$.

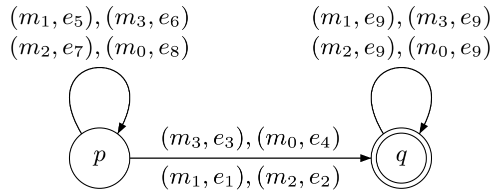

Figure 4: The K-MA representing $pre^*_K(\langle p, m_3\Gamma^*\rangle)$

- $\frac{\partial}{\partial x_i}(f \oplus g) = \frac{\partial f}{\partial x_i} \oplus \frac{\partial g}{\partial x_i}$
- $\frac{\partial}{\partial x_i}(f \odot g) = (f \odot \frac{\partial g}{\partial x_i}) \oplus (\frac{\partial f}{\partial x_i} \odot g)$
- $\frac{\partial}{\partial x_i}(f^\star) = f^\star \odot \frac{\partial f}{\partial x_i}$

Then, the least solution of (1) is the fixpoint of the chain $\mathbf{a_0} \le \mathbf{a_1} \le \mathbf{a_2} \ldots$[3] given by

$$\begin{aligned} \mathbf{a_0} &= \mathbf{f}(\bar{\mathbf{0}}) \\ \mathbf{a_{k+1}} &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{a_k})^\star \odot \mathbf{a_k}, \end{aligned}$$

That is, to compute $\mathbf{a_{k+1}}$ we evaluate the jacobian at the vector $\mathbf{a_k}$, obtaining a matrix over K. Then, we compute the transitive closure of this matrix using e.g. Floyd-Warshall's algorithm. Finally, we compute the product of the resulting matrix and the vector $\mathbf{a_k}$.

5.2 Example

Let us consider as example the program presented on the right of Figure 1. This program is represented by the pushdown system $\mathcal{P} = (\{p\}, \{a, b, \tau\}, \{m_0, m_1, m_2, m_3\}, \langle p, m_0\rangle, \Delta)$, where Δ is the set of rules given at the end of section 2.

Let $(K, \oplus, \odot, \star, \bar{0}, \bar{1})$ be a commutative Kleene algebra associated to $\mathcal{P}$. We compute $pre^*_K(\langle p, m_3\Gamma^*\rangle)$. We start with the MA $\mathcal{A} = (\{m_0, m_1, m_2, m_3\}, \{p, q\}, \{(p, m_3, q), (q, m_0, q), \ldots, (q, m_3, q)\}, \{p\}, \{q\})$ recognizing $\langle p, m_3\Gamma^*\rangle$.

Our algorithm yields the K-MA depicted in Figure 4, where the labels $e_1, \ldots, e_9$ are the least solution of the following inequalities:

[3]That this is in fact a chain follows easily from the axioms of a Kleene algebra.

$$\begin{aligned}
(x_4 \odot x_9) \oplus (x_8 \odot x_3) &\leq x_1 \\
v_a \odot x_3 &\leq x_2 \\
\bar{1} &\leq x_3 \\
(v_a \odot x_1) \oplus (v_b \odot x_2) &\leq x_4 \\
x_8 \odot x_6 &\leq x_5 \\
\bar{1} &\leq x_6 \\
v_a \odot x_6 &\leq x_7 \\
(v_b \odot x_7) \oplus (v_a \odot x_5) &\leq x_8 \\
\bar{1} &\leq x_9
\end{aligned}$$

Let us explain briefly these inequalities. The inequality $v_a \odot x_3 \leq x_2$ expresses that if π is a path-expression of K such that $(\langle p, m_3\rangle, \pi)$ is a K-predecessor of $(\langle p, m_3\Gamma^*\rangle, \bar{1})$, then so is $(\langle p, m_2\rangle, v_a \odot \pi)$ (rule r_4). Since the transitions $p \xrightarrow{m_3} q$ and $p \xrightarrow{m_2} q$ are respectively tagged with e_3 and e_2, it follows that

$$\pi \leq e_3 \Rightarrow v_a \odot \pi \leq e_2.$$

Therefore, $v_a \odot e_3 \leq e_2$.

Similarly, the inequality $(x_4 \odot x_9) \oplus (x_8 \odot x_3) \leq x_1$ expresses that if π is a path-expression of K such that $(\langle p, m_0 m_3\rangle, \pi)$ is a K-predecessor of $(\langle p, m_3\Gamma^*\rangle, \bar{1})$, then so is $(\langle p, m_1\rangle, \pi)$ (rule r_3). Since there are two paths leading from p to q by $m_0 m_3$ (which are respectively tagged with $e_4 \odot e_9$ and $e_8 \odot e_3$), it follows that if $\pi \leq e_8 \odot e_3$ or $\pi \leq e_4 \odot e_9$, then $\pi \leq e_1$, which means that $e_8 \odot e_3 \leq e_1$ and $e_4 \odot e_9 \leq e_1$, i.e., $(e_4 \odot e_9) \oplus (e_8 \odot e_3) \leq e_1$.

To solve these inequalities we apply the algorithm provided by [HK99], where, for example, $f_1(\mathbf{x})$ is the monomial $(x_4 \odot x_9) \oplus (x_8 \odot x_3)$, and $f_4(\mathbf{x})$ is the monomial $(v_a \odot x_1) \oplus (v_b \odot x_2)$. The jacobian $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is a 9×9 matrix whose elements are either 0, v_a, v_b, or x_i. For example, the $(1,3)^{th}$ element of this matrix is x_8 (since $\frac{\partial f_1}{\partial x_3} = x_8$), and the $(2,3)^{th}$ element is v_a (since $\frac{\partial f_2}{\partial x_3} = v_a$). The least solution, given by $e_1 = e_5 = e_8 = v_a^\star \odot v_a \odot v_b$, $e_3 = e_6 = e_9 = \bar{1}$, $e_2 = e_7 = v_a$, and

$$e_4 = (v_a \odot v_a^\star \odot v_a \odot v_b) \oplus (v_a \odot v_b)\ ,$$

is obtained after one iteration.

It follows that the configurations $\langle p, m_0\Gamma^*\rangle$ are tagged with e_4, meaning that $L(\langle p, m_0\Gamma^*\rangle, \langle p, m_3\Gamma^*\rangle)$ is approximated by $e_4 = (v_a \odot v_a^\star \odot v_a \odot v_b) \oplus (v_a \odot v_b)$.

For the three abstractions that we have considered, we get:

- In the "forbidden and required sets" framework, $v_a = [\{b\}, \{a\}]$ and $v_b = [\{a\}, \{b\}]$, which means that $e_4 = [\emptyset, \{a, b\}]$. This shows that starting from m_0, both a and b are required to reach the point m_3.
- In the "label bitvectors" framework, $v_a = \binom{1}{0}$, $v_b = \binom{0}{1}$, and $e_4 = \binom{1}{1}$. This means that we have to execute at least one a and one b to go from m_0 to m_3.
- In the "parikh images" framework, $v_a = \binom{1}{0}$, $v_b = \binom{0}{1}$, and $e_4 = \{\binom{k}{1} \mid k \geq 1\}$. This expresses that the paths leading from m_0 to m_3 contain an arbitrary number (≥ 1) of a's, and exactly one b.

5.3 Complexity of the generic procedure

The complexity is dominated by the fixpoint algorithm of [HK99]. Since the manipulated matrices have dimension $m \times m$, each iteration of this algorithm requires to compute $O(m^3)$ $\oplus/\odot/\star$-operations. Recall that m is the number of transitions of $\mathcal{A}_{pre^*}$. It is shown in [EHRS00] that $\mathcal{A}_{pre^*}$ has $O(|Q||\Delta| + |\delta|)$ transitions. So the running time is $O(t \cdot (|Q||\Delta| + |\delta|)^3 \cdot c)$, where t is the number of iterations, and c is the cost of an operation. In typical applications the automaton $\mathcal{A}$ has constant size (two or three states for pure dataflow analysis), and so we can simplify the expression to $O(t \cdot |\Delta|^3 \cdot c)$.

It is proved in [HK99] that for an *arbitrary* Kleene algebra the fixpoint of the chain is reached after at most $O(3^m)$ iterations. Since each iteration requires to compute a transitive closure, the size of the expressions can grow exponentially at each iteration. So the maximal size of an expression can be double exponential in m, and so the cost of an operation is also double exponential in m. This is also the dominating factor in the overall complexity.

However, for particular algebras the running time can be much lower. Let us compute it for our three abstractions.

5.4 Complexity of the instances

Forbidden and required sets: If the maximal $\leq$-chain in the abstract lattice has length k, then the number of iterations of the fixpoint algorithm is obviously bounded by k. In this instance we have $[F_1, R_1] \leq [F_2, R_2]$ if and only if $F_2 \subseteq F_1$ and

$R_2 \subseteq R_1$. Since moreover for every $[F, R] \in K$ we have $F, R \subseteq Lab$ and $F \cap R = \emptyset$, the longest possible chain in K has at most length $|Lab|$. So the fixpoint algorithm terminates after at most $|Lab|$ iterations. If sets of labels are implemented as bitvectors, the $\oplus$ and $\odot$ operations are bitwise OR and AND ($\star$ is the identity), whose cost is constant or $O(|Lab|)$, depending on the application. So the algorithm runs in $O(|Lab|^2 \cdot |\Delta|^3)$ or $O(|Lab| \cdot |\Delta|^3)$time.

Label bitvectors: In this case the order $\leq$ is simply set inclusion between sets of boolean vectors. So the longest chain in K has 2^{Lab} elements. The $\star$-operation dominates the cost part, since it takes $O(2^{|Lab|})$ time. So we get $2^{O(|Lab|)} \cdot |\Delta|^3$ time. (A finer amortised analysis could give a better bound, because there is a trade-off between the cost of the operations and the number of iterations).

Parikh images: In this case the complexity is the same as for arbitrary algebras.

As a result of this analysis, we conclude that the forbidden-required analysis has acceptable cost, the action-set analysis has acceptable cost if there are few synchronisation statements, and the Parikh-image analysis is likely to have too high cost in most cases. It only seems of practical interest when there are few synchronisation statements in loops, since those are responsible for the explosion in the size of the solution.

6 Conclusions and related work

We have presented a generic aproach to the static analysis of concurrent programs with procedures. The effect of the procedures is determined exactly, while the constraints on program paths induced by synchronizations is only approximated. Due to the undecidability result of [Ram00], approximation techniques must be used. The gist of our technique is to approximate context-free languages by commutative languages closed under intersection, and for which emptyness is decidable. Using algebraic results we have provided a generic algorithm for the computation of the approximations, and we have shown three possible instantiations leading to analysis of different precision and cost.

Work on static analysis of concurrent programs goes back to Taylor's seminal paper [Tay83], which showed that many problems are NP-complete even in the intraprocedural case (other problems are even PSPACE-complete) and proposed a general analysis algorithm for different problems. Since then, the intraprocedural case has received considerable attention, and [Rin01] is a good recent survey. Since we focus on the interprocedural case, we do not compare our work with these papers. However, [NA98] is especially interesting for us. It avoids an exponential state based analysis by means of what can be seen as the definition of a weak product of finite automata. This technique is orthogonal to the ones shown here, and can be adapted to our pushdown automata model.

To the best of our knowledge, the only papers dealing with concurrency analysis in the presence of procedures are [BBS00, DS91]. In [BBS00] procedures can be handled as tasks, but the analysis is not sensitive to possible multiple incarnations of the procedure; therefore, it is not really work on procedures. In [DS91] procedures are explicitely considered. The paper describes a before-after analysis for the detection of data races. The analysis has common points with the forbidden-required instantiation of our approach: if an action is required to occur in order to acces a program point, then that action must necessarily occur before the point is accessed. The algorithm of the paper runs in cubic time in the size of the program. The two advantages of our approach are that our algorithm is generic, and can be instantiated in many different ways, and that our analysis is exact in the absence of concurrency. However, in [DS91] dynamic task creation is allowed, which is not possible in our current setting. A possibility to add dynamic creation to our setting is to use the technique of [EP00, LS02], and we plan to investigate this in the future.

Acknowledgements

We thank Luca Aceto for helpful discussions on commutative semirings.

References

[BBS00] J. Blieberger, B. Burgstaller, and B. Scholz. Symbolic Data Flow Analysis for Detecting Deadlocks in Ada Tasking Programs. In *Proc. of the Ada-Europe International Conference on Reliable Software Technologies*, Potsdam, Germany, 2000.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. Proc of CONCUR'97, LNCS 1243, 1997.

[DC94] Matthew B. Dwyer and Lori A. Clarke. Data flow analysis for verifying properties of concurrent programs. Proc. of SIGSOFT '94, 1994.

[DS91] E. Duesterwald and M.L. Soffa. Concurrency analysis in the presence of of procedures using a data-flow framework. In *Proc. of the Symposium on Testing, Analysis, and Verification, Victoria, Canada*, ACM Press, 1991.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithm for model checking pushdown systems. Proc. of *CAV'00*, LNCS 1885, 2000.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. Proc. of FoSSaCS'99, LNCS 1578, 1999.

[EP00] J. Esparza and A.Podelski. Efficient Algorithms for *pre** and *post** in Parallel Interprocedural Flow Graphs Proc. of POPL'00, ACM Press, 2000.

[HK99] M.W. Hopkins and D.C. Kozen. Parikh's Theorem in Commutative Kleene Algebra. Proc. of LICS'99, IEEE, 1999 1999.

[LS02] D. Lugiez and P. Schnoebelen: : The Regular Viewpoint on PA-processes. TCS 274(1–2), 2002.

[NA98] G. Naumovich and G. Avrunin. A conservative data flow algorithm for detecting all pairs of statement that may happen in parallel. Proc. of SIGSOFT '98, ACM Press, 1998.

[Ram00] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. ACM TOPLAS 22, 2000.

[Rin01] M. Rinard. Analysis of multithreaded programs. Proc. of SAS 2001, LNCS 2126, 2001.

[Tay83] R.N. Taylor. A general-purpose algorithm for analyzing concurrent programs. CACM 26, 1983.

7 Appendix

7.1 Computing pre^*_K images

Theorem 5.1 *Given a PDS $\mathcal{P}$ and a regular set of configurations recognized by a MA $\mathcal{A}$, the procedure of section 5 constructs a K-MA $\mathcal{A}_{pre^*_K}$ recognizing precisely $pre^*_K\big(Conf(\mathcal{A})\big)$.*

We prove the more precise statement: if $\mathcal{A}_{pre^*} = (\Gamma, Q, \delta, P, F)$ is the MA which recognizes $pre^*\big(Conf(\mathcal{A})\big)$ constructed by the procedure of [BEM97, EHRS00], then $\mathcal{A}_{pre^*_K} = (\Gamma, Q, \delta', P, F)$ where

$$\delta' = \{(q, \gamma, e, q') \mid (q, \gamma, q') \in \delta, e = l\big((q, \gamma, q')\big)\}.$$

The proof of the facts above follows from the two lemmas we prove hereafter.

Lemma 7.1 *For every configuration $\langle p, v\rangle \in Conf(\mathcal{A})$, if $\big(\langle p, v\rangle, \bar{1}\big) \Leftarrow_K \big(\langle p', w\rangle, \pi\big)$ for some path-expression $\pi \in \Pi_K$, then there exists $e \in K$ such that $\pi \leq e$ and $p' \xrightarrow{(w,e)} q$ for some final state q of $\mathcal{A}_{pre^*_K}$.*

Proof: First, let us introduce some notations. Let $p' \xrightarrow{(w,e)} q$ be a path of $\mathcal{A}_{pre^*_K}$. It follows from the construction of this K-MA that there exists a corresponding path $\rho = p' \xrightarrow{w} q$ in $\mathcal{A}_{pre^*}$. Therefore, we will denote the expression e by $\lambda\big((p', w, q)_\rho\big)$. If $w = \gamma \in \Gamma$, i.e., if ρ corresponds to one single transition, then we simply write $e = \lambda\big((p', \gamma, q)\big)$.

To prove the statement it suffices to show that if $\big(\langle p, v\rangle, \bar{1}\big) \Leftarrow_K \big(\langle p', w\rangle, \pi\big)$ for some path-expression $\pi \in \Pi_K$, then there exists a final state q such that $\rho = p' \xrightarrow{w} q$ is a path of $\mathcal{A}_{pre^*}$ and $\pi \leq \lambda\big((p', w, q)_\rho\big)$.

Let $\big(\langle p, v\rangle, \bar{1}\big) \overset{k}{\Leftarrow}_K \big(\langle p', w\rangle, \pi\big)$ mean that $\big(\langle p', w\rangle, \pi\big)$ is a K-predecessor of $\big(\langle p, v\rangle, \bar{1}\big)$ after k steps. We show the statement by induction on k.

- $k = 0$. Then $p' = p$, $w = v$, and $\pi = \bar{1}$. Since $\langle p, v\rangle \in Conf(\mathcal{A})$, then there exists a final state q such that $\rho = p \xrightarrow{v} q$ is a path of $\mathcal{A}$ (and hence of $\mathcal{A}_{pre^*}$). Therefore, rules (α_1) imply that $\bar{1} \leq \lambda\big((p, v, q)_\rho\big)$.

- $k > 0$. Let then $p'' \in P, u \in \Gamma^*, \pi' \in \Pi_K$ such that

$$\big(\langle p, v\rangle, \bar{1}\big) \overset{k-1}{\Leftarrow}_K \big(\langle p'', u\rangle, \pi'\big) \overset{1}{\Leftarrow}_K \big(\langle p', w\rangle, \pi\big)$$

By induction hypothesis, there exists a final state q such that $\rho = p'' \xrightarrow{u} q$ is a run in $\mathcal{A}_{pre^*}$ and $\pi' \leq \lambda\big((p'', u, q)_\rho\big)$.

Since $\big(\langle p'', u\rangle, \pi'\big) \overset{1}{\Leftarrow}_K \big(\langle p', w\rangle, \pi\big)$, there are $\gamma \in \Gamma, w_1, u_1 \in \Gamma^*$ and a rule $r = (p', \gamma) \overset{a}{\hookrightarrow} (p'', u_1)$ of Δ such that $w = \gamma w_1$, $u = u_1 w_1$, and $\pi = v_a \odot \pi'$. Let then $q' \in Q$ be such that $\rho = p'' \xrightarrow{u_1} q' \xrightarrow{w_1} q$. Let $\rho_1 = p'' \xrightarrow{u_1} q'$ be the first part of ρ, and $\rho_2 = q' \xrightarrow{w_1} q$ be its second part. It follows that $\pi' \leq \lambda\big((p'', u_1, q')_{\rho_1}\big) \odot \lambda\big((q', w_1, q)_{\rho_2}\big)$. ¿From the saturation rules, it follows that there is a transition $p' \xrightarrow{\gamma} q'$ in $\mathcal{A}_{pre^*}$, which means that $\rho' = p' \xrightarrow{\gamma} q' \xrightarrow{w_1} q$ is a path in $\mathcal{A}_{pre^*}$. From rules $(\alpha_2), (\alpha_3)$, and (α_4), it follows that

$$v_a \odot \lambda\big((p'', u_1, q')_{\rho_1}\big) \leq \lambda\big((p', \gamma, q')\big)$$

Therefore, $\rho' = p' \xrightarrow{w} q$ holds in $\mathcal{A}_{pre^*}$ (since $\gamma w_1 = w$), where q is a final state, and

$$\begin{aligned} \pi &= v_a \odot \pi' \\ &\leq v_a \odot \lambda\big((p'', u_1, q')_{\rho_1}\big) \odot \lambda\big((q', w_1, q)_{\rho_2}\big) \\ &\leq \lambda\big((p', \gamma, q')\big) \odot \lambda\big((q', w_1, q)_{\rho_2}\big) \\ &= \lambda\big((p', w, q)_{\rho'}\big) \end{aligned}$$

This ends the proof of the lemma.

□

Let π be a path-expression over K. We denote by $|\pi|$, the length of π, the number of occurrences of basic elements v_a's in π.

Lemma 7.2 *Let $p \xrightarrow{(w,e)} q$ be a path of $\mathcal{A}_{pre^*_K}$, then for every path-expression $\pi \leq e$, there exists a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow{w'} q$ is a path in $\mathcal{A}$, and $\big(\langle p', w'\rangle, \bar{1}\big) \Leftarrow_K \big(\langle p, w\rangle, \pi\big)$.*

Proof: We will show by induction on $|\pi|$ that if $\pi \leq \lambda\big((p, w, q)_\rho\big)$ for some run $\rho = p \xrightarrow{w} q$ of $\mathcal{A}_{pre^*}$, then there exists a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow{w'} q$ is a path in $\mathcal{A}$ and $\big(\langle p', w'\rangle, \bar{1}\big) \Leftarrow_K \big(\langle p, w\rangle, \pi\big)$.

Let $\gamma \in \Gamma$, $w_1 \in \Gamma^*$, and $q' \in Q$ be such that $w = \gamma w_1$ and

$$\rho = p \xrightarrow{\gamma} q' \xrightarrow{w_1} q.$$

Let π be a path-expression such that

$$\pi \leq \lambda\big((p, w, q)_\rho\big).$$

- $|\pi| = 0$, i.e., $\pi = \bar{1}$. Then necessarily, according to the inequalities (α_1), $p \xrightarrow{w} q$ already holds in $\mathcal{A}$. Therefore, the statement holds with $p' = p$ and $w' = w$.

- $|\pi| > 0$. Then π satisfies $\pi \leq \lambda\big((p, \gamma, q')\big) \odot \lambda\big((q', w_1, q)_{\rho_1}\big)$, where $\rho_1 = q' \xrightarrow{w_1} q$ is the last part of ρ. According to the inequalities (α_2), (α_3), and (α_4), there exists a rule $r = (p, \gamma) \overset{a}{\hookrightarrow} (p_1, u)$ of Δ (where $u = \gamma'$ or $u = \gamma'\gamma''$, or $u = \epsilon$. Here we will only consider the case where $u = \gamma'$, the other cases are similar), and two path-expressions π_1 and π_2 such that $\pi = v_a \odot \pi_1 \odot \pi_2$, $v_a \odot \pi_1 \leq \lambda\big((p, \gamma, q')\big)$, $\pi_1 \leq \lambda\big((p_1, u, q')\big)$, and $\pi_2 \leq \lambda\big((q', w_1, q)_{\rho_1}\big)$.

  Let $\rho' = p_1 \xrightarrow{u} q' \xrightarrow{w_1} q$ be the corresponding run of $\mathcal{A}_{pre^*}$ (this run exists thanks to the saturation rules). It follows that

  $$\begin{aligned} \pi_1 \odot \pi_2 &\leq \lambda\big((p_1, u, q')\big) \odot \lambda\big((q', w_1, q)_{\rho_1}\big) \\ &= \lambda\big((p_1, uw_1, q)_{\rho'}\big) \end{aligned}$$

  Since $|\pi_1 \odot \pi_2| < |\pi|$, the induction hypothesis implies that there exists a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow{w'} q$ is a path in $\mathcal{A}$, and $\big(\langle p', w'\rangle, \bar{1}\big) \Leftarrow_K \big(\langle p_1, uw_1\rangle, \pi_1 \odot \pi_2\big)$. Moreover, by applying r we get that $\big(\langle p_1, uw_1\rangle, \pi_1 \odot \pi_2\big) \Leftarrow_K \big(\langle p, \gamma w_1\rangle, v_a \odot \pi_1 \odot \pi_2\big)$. Therefore, since $w = \gamma w_1$ and $\pi = v_a \odot \pi_1 \odot \pi_2$, we get

  $$\big(\langle p', w'\rangle, \bar{1}\big) \Leftarrow_K \big(\langle p, w\rangle, \pi\big)$$

  where $\langle p', w'\rangle$ is such that $p' \xrightarrow{w'} q$ is a path in $\mathcal{A}$. This ends the proof.

□

7.2 Computing $post^*_K$ images

Given a regular set of configurations C recognized by a MA $\mathcal{A} = (\Gamma, Q_0, \delta_0, P, F)$, we construct a K-MA $\mathcal{A}_{post^*_K}$ which recognizes $post^*_K(C)$. As in the previous section, we assume w.l.o.g. that $\mathcal{A}$ has no transition leading to an initial state.

Again, our algorithm is in two steps. First we construct a MA $\mathcal{A}_{post^*}$ recognizing $post^*(C)$, and then we tag the transitions of this automaton with elements of K in order to obtain $\mathcal{A}_{post^*_K}$.

For the first step we apply the procedure of [EHRS00]. This procedure constructs a MA with ϵ-transitions $\mathcal{A}_{post^*}$, by computing iteratively the finite sequence $\mathcal{A}'_i$ defined as follows:

- $\mathcal{A}'_0$ is the MA obtained from $\mathcal{A}$ by adding for each transition rule $r \in \Delta$ of the form $(p,\gamma) \overset{a}{\hookrightarrow} (p',\gamma'\gamma'')$ a new state r and a new transition (p',γ',r),
- for every rule $(p,\gamma) \overset{a}{\hookrightarrow} (p',\epsilon)$ and every state q such that $p \overset{\gamma}{\rightsquigarrow}_i q$, add the new transition $p' \xrightarrow{\epsilon}_{i+1} q$,
- for every rule $(p,\gamma) \overset{a}{\hookrightarrow} (p',\gamma')$ and every state q such that $p \overset{\gamma}{\rightsquigarrow}_i q$, add the new transition $p' \xrightarrow{\gamma'}_{i+1} q$,
- for every rule $(p,\gamma) \overset{a}{\hookrightarrow} (p',\gamma'\gamma'')$ and every state q such that $p \overset{\gamma}{\rightsquigarrow}_i q$, add the new transition $r \xrightarrow{\gamma''}_{i+1} q$,

where $\rightarrow_i$ is the transition relation of $\mathcal{A}'_i$, and $\overset{\gamma}{\rightsquigarrow}$ denotes the relation $\xrightarrow{\epsilon}^* \circ \xrightarrow{\gamma} \circ \xrightarrow{\epsilon}^*$. Observe that the initial MA $\mathcal{A}$ does not contain transitions leading to an initial state, and the procedure described above does not add such transitions. Moreover, all the ϵ-transitions added by this procedure start from an initial state. Therefore, $\overset{\gamma}{\rightsquigarrow} = \xrightarrow{\gamma} \cup \xrightarrow{\epsilon} \circ \xrightarrow{\gamma}$.

Then, the second step consists in tagging the transitions t of $\mathcal{A}_{post^*}$ with elements $h(t) \in K$ defined as the smallest elements satisfying the following inequalities:

(β_1) if $t = (q,\gamma,q')$ is already a transition of $\mathcal{A}'_0$, then $\bar{1} \leq h(t)$,

(β_2) for every rule $r = (p,\gamma) \overset{a}{\hookrightarrow} (p',\gamma')$ and every $q \in Q$,

$$v_a \odot h'\big((p,\gamma,q)\big) \leq h\big((p',\gamma',q)\big)$$

(β_3) for every rule $r = (p,\gamma) \overset{a}{\hookrightarrow} (p',\epsilon)$ and every $q \in Q$,

$$v_a \odot h'\big((p,\gamma,q)\big) \leq h\big((p',\epsilon,q)\big)$$

(β_4) for every rule $r = (p,\gamma) \overset{a}{\hookrightarrow} (p',\gamma'\gamma'')$ and every $q \in Q$,

$$v_a \odot h'\big((p,\gamma,q)\big) \leq h\big((r,\gamma'',q)\big)$$

where $h'\big((p,\gamma,q)\big)$ denotes the summary of the labels of all the runs of $\mathcal{A}_{post^*}$ of the form $p \overset{\gamma}{\rightsquigarrow} q$, i.e. (since $\overset{\gamma}{\rightsquigarrow} = \xrightarrow{\gamma} \cup \xrightarrow{\epsilon} \circ \xrightarrow{\gamma}$),

$$h'\big((p,\gamma,q)\big) = h\big((p,\gamma,q)\big) \oplus \bigoplus_{q' \in Q} \Big(h\big((p,\epsilon,q')\big) \odot h\big((q',\gamma,q)\big) \Big)$$

Theorem 7.1 *Given a PDS $\mathcal{P}$ and a regular set of configurations recognized by a MA $\mathcal{A}$, the procedure above constructs a K-MA $\mathcal{A}_{post^*_K}$ which recognizes precisely $post^*_K\big(Conf(\mathcal{A})\big)$.*

As in the previous case, we prove that: if $\mathcal{A}_{post^*} = (\Gamma, Q, \delta, P, F)$ is the MA recognizing $post^*\big(Conf(\mathcal{A})\big)$ constructed in [EHRS00], then $\mathcal{A}_{post^*_K} = (\Gamma, Q, \delta', P, F)$ where

$$\delta' = \{(q,\gamma,e,q') \mid (q,\gamma,q') \in \delta, e = h\big((q,\gamma,q')\big)\}.$$

Intuitively, the inequalities defining the mapping h express that, for every rule $r = (p,\gamma) \overset{a}{\hookrightarrow} (p',u)$, and every $q \in Q$, if w is such that $q \xrightarrow{w} q'$ for some final state q', and if $(\langle p, \gamma w\rangle, \pi)$ is a K-successor of $Conf(\mathcal{A})$, then so is $(\langle p', \gamma w\rangle, v_a \odot \pi)$.

Theorem 7.1 follows from the next two lemmas:

Lemma 7.3 *For every configuration $\langle p,v\rangle \in Conf(\mathcal{A})$, if $\big(\langle p,v\rangle, \bar{1}\big) \Rightarrow_K \big(\langle p',w\rangle, \pi\big)$ then there exists $e \in K$ such that $\pi \leq e$ and $p' \xrightarrow{(w,e)} q$ for some final state q of $\mathcal{A}_{post^*_K}$.*

Proof: We give some notations first. Let $p' \xrightarrow{(w,e)} q$ be a path of $\mathcal{A}_{post^*_K}$. It follows from the construction of this K-MA that there exists a corresponding path $\rho = p' \xrightarrow{w} q$ in $\mathcal{A}_{post^*}$. Therefore, we will denote the expression e by $\lambda\big((p',w,q)_\rho\big)$. If $w = \gamma \in \Gamma$, i.e., if ρ corresponds to one single transition, then we simply write $e = \lambda\big((p',\gamma,q)\big)$.

Let $\langle p,v\rangle \in Conf(\mathcal{A})$. We will show that if $\big(\langle p,v\rangle, \bar{1}\big) \Rightarrow_K \big(\langle p',w\rangle, \pi\big)$ then there exists a final state q such that $\rho = p' \xrightarrow{w} q$ is a path of $\mathcal{A}_{post^*}$ and $\pi \leq \lambda\big((p',w,q)_\rho\big)$.

Let $\big(\langle p,v\rangle, \bar{1}\big) \overset{k}{\Rightarrow}_K \big(\langle p',w\rangle, \pi\big)$ mean that $\big(\langle p',w\rangle, \pi\big)$ is a K-successor of $\big(\langle p,v\rangle, \bar{1}\big)$ after k steps. We show the statement by induction on k.

- $k = 0$. Then $p' = p$, $w = v$, and $\pi = \bar{1}$. Since $\langle p,v\rangle \in Conf(\mathcal{A})$, then there exists a final state q such that $\rho = p \xrightarrow{v} q$ is already a path of

$\mathcal{A}$ (and hence of $\mathcal{A}_{post^*}$). Therefore, rules (β_1) imply that $\bar{1} \le \lambda\big((p, v, q)_\rho\big)$.

- $k > 0$. Let then $p'' \in P, u \in \Gamma^*, \pi' \in \Pi_K$ such that

$$\big(\langle p, v\rangle, \bar{1}\big) \overset{k-1}{\Rightarrow}_K \big(\langle p'', u\rangle, \pi'\big) \overset{1}{\Rightarrow}_K \big(\langle p', w\rangle, \pi\big)$$

By induction hypothesis there exists a path $\rho' = p'' \xrightarrow{u} q$ of $\mathcal{A}_{post^*}$ such that q is a final state and $\pi' \le \lambda\big((p'', u, q)_{\rho'}\big)$.

Since $\big(\langle p'', u\rangle, \pi'\big) \overset{1}{\Rightarrow}_K \big(\langle p', w\rangle, \pi\big)$, there are $\gamma \in \Gamma, w_1, u_1 \in \Gamma^*$ and a rule $r = (p'', \gamma) \overset{a}{\hookrightarrow} (p', w_1)$ of Δ such that $u = \gamma u_1$, $w = w_1 u_1$, and $\pi = v_a \odot \pi'$. Let then $q' \in Q$ such that $\rho' = p'' \overset{\gamma}{\rightsquigarrow} q' \xrightarrow{u_1} q$. Let $\rho_1 = p'' \overset{\gamma}{\rightsquigarrow} q'$ and $\rho_2 = q' \xrightarrow{u_1} q$ be the two parts of ρ'. There are two cases depending on whether $|w_1| < 2$ or $|w_1| = 2$:

  - $|w_1| < 2$. From the saturation rules, it follows that there is a transition $p' \xrightarrow{w_1} q'$ in $\mathcal{A}_{post^*}$, which means that $\rho = p' \xrightarrow{w_1} q' \xrightarrow{u_1} q$ is a path in $\mathcal{A}_{post^*}$. From rules (β_2) and (β_3), it follows that

$$v_a \odot \lambda\big((p'', \gamma, q')_{\rho_1}\big) \le \lambda\big((p', w_1, q')\big)$$

  Therefore, since $w_1 u_1 = w$, $\rho = p' \xrightarrow{w} q$ is an accepting path of $\mathcal{A}_{post^*}$, and

$$\begin{aligned} \pi &= v_a \odot \pi' \\ &\le v_a \odot \lambda\big((p'', u, q)_{\rho'}\big) \\ &= v_a \odot \lambda\big((p'', \gamma, q')_{\rho_1}\big) \odot \lambda\big((q', u_1, q)_{\rho_2}\big) \\ &\le \lambda\big((p', w_1, q')\big) \odot \lambda\big((q', u_1, q)_{\rho_2}\big) \\ &= \lambda\big((p', w, q)_\rho\big) \end{aligned}$$

  - $|w_1| = 2$, let then $w_1 = \gamma'\gamma''$. It follows that there is a path

$$p' \xrightarrow{\gamma'} r \xrightarrow{\gamma''} q'$$

  in $\mathcal{A}_{post^*}$, which means that $\rho = p' \xrightarrow{\gamma'} r \xrightarrow{\gamma''} q' \xrightarrow{u_1} q$ is a path in $\mathcal{A}_{post^*}$. Rules (β_1) imply that $\bar{1} \le \lambda\big((p', \gamma', r)\big)$, and from rules ($\beta_4$) it follows that

$$v_a \odot \lambda\big((p'', \gamma, q')_{\rho_1}\big) \le \lambda\big((r, \gamma'', q')\big)$$

  Therefore, $\rho = p' \xrightarrow{w} q$ is an accepting path of $\mathcal{A}_{post^*}$ (since $\gamma'\gamma'' u_1 = w_1 u_1 = w$) and π satisfies:

$$\begin{aligned} \pi &= \bar{1} \odot v_a \odot \pi' \\ &\le \lambda\big((p', \gamma', r)\big) \odot v_a \odot \lambda\big((p'', u, q)_{\rho'}\big) \\ &= \lambda\big((p', \gamma', r)\big) \odot v_a \odot \lambda\big((p'', \gamma, q')_{\rho_1}\big) \\ &\quad \odot \lambda\big((q', u_1, q)_{\rho_2}\big) \\ &\le \lambda\big((p', \gamma', r)\big) \odot \lambda\big((r, \gamma'', q')\big) \odot \lambda\big((q', u_1, q)_{\rho_2}\big) \\ &= \lambda\big((p', w, q)_\rho\big) \end{aligned}$$

□

Lemma 7.4 *Let $p \xrightarrow{(w,e)} q$ be a path of $\mathcal{A}_{post^*_K}$ such that $q \in Q_0$, then for every path-expression $\pi \le e$, there exists a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow{w'} q$ is a path in $\mathcal{A}$, and $\big(\langle p', w'\rangle, \bar{1}\big) \Rightarrow_K \big(\langle p, w\rangle, \pi\big)$.*

Proof: We will show by induction on $|\pi|$ that if $\pi \le \lambda\big((p, w, q)_\rho\big)$ for some run $\rho = p \xrightarrow{w} q$ of $\mathcal{A}_{post^*}$ such that $q \in Q_0$, then there exists a configuration $\langle p', w'\rangle$ such that $p' \xrightarrow{w'} q$ is a path in $\mathcal{A}$ and $\big(\langle p', w'\rangle, \bar{1}\big) \Rightarrow_K \big(\langle p, w\rangle, \pi\big)$.

Let $\rho = p \xrightarrow{w} q$ be a path of $\mathcal{A}_{post^*}$ such that $q \in Q_0$, and let π be a path-expression such that $\pi \le \lambda\big((p, w, q)_\rho\big)$.

- $|\pi| = 0$, i.e., $\pi = \bar{1}$. Then necessarily, according to the inequalities (β_1), $\rho = p \xrightarrow{w} q$ is a run of $\mathcal{A}'_0$ and hence, it is already a run of $\mathcal{A}$ since $\mathcal{A}'_0$ contains the transitions of $\mathcal{A}$ together with transitions of the form $p \xrightarrow{\gamma} r$ where $r \notin Q_0$. It follows that in $\mathcal{A}'_0$ there is no run leading to a state in Q_0 that passes through a state r. The path ρ considers then only transitions of $\mathcal{A}$. Therefore, the statement holds with $p' = p$ and $w' = w$.

- $|\pi| > 0$. There are three cases:

  - There exist $\gamma \in \Gamma$, $w_1 \in \Gamma^*$, and $q' \in Q_0$ ($q' \ne r$), $w = \gamma w_1$ and

$$\rho = p \xrightarrow{\gamma} q' \xrightarrow{w_1} q.$$

  π satisfies $\pi \le \lambda\big((p, \gamma, q')\big) \odot \lambda\big((q', w_1, q)_{\rho_1}\big)$, where $\rho_1 = q' \xrightarrow{w_1} q$. According to the inequalities (β_2), there exists a rule $r = (p_1, \gamma_1) \overset{a}{\hookrightarrow} (p, \gamma)$ of

Δ, and two path-expressions π_1 and π_2 such that $\pi = v_a \odot \pi_1 \odot \pi_2$, $v_a \odot \pi_1 \leq \lambda\big((p,\gamma,q')\big)$, $\pi_1 \leq \lambda\big((p_1,\gamma_1,q')_{\rho_2}\big)$, and $\pi_2 \leq \lambda\big((q',w_1,q)_{\rho_1}\big)$, where ρ_2 is a path of the form $\rho_2 = p_1 \overset{\gamma_1}{\rightsquigarrow} q'$.

Let $\rho' = p_1 \overset{\gamma_1}{\rightsquigarrow} q' \xrightarrow{w_1} q$ be the corresponding run of $\mathcal{A}_{post^*}$. It follows that

$$\begin{aligned}\pi_1 \odot \pi_2 &\leq \lambda\big((p_1,\gamma_1,q')_{\rho_2}\big) \odot \lambda\big((q',w_1,q)_{\rho_1}\big)\\ &= \lambda\big((p_1,\gamma_1 w_1,q)_{\rho'}\big)\end{aligned}$$

Sine $|\pi_1 \odot \pi_2| < |\pi|$, the induction hypothesis implies that there exists a configuration $\langle p',w'\rangle$ such that $p' \xrightarrow{w'} q$ is a run of $\mathcal{A}$ and

$$\big(\langle p',w'\rangle,\bar{1}\big) \Rightarrow_K \big(\langle p_1,\gamma_1 w_1\rangle, \pi_1 \odot \pi_2\big).$$

Moreover, by applying r we get that

$$\big(\langle p_1,\gamma_1 w_1\rangle, \pi_1 \odot \pi_2\big) \Rightarrow_K \big(\langle p,\gamma w_1\rangle, v_a \odot \pi_1 \odot \pi_2\big).$$

Therefore, since $w = \gamma w_1$ and $\pi = v_a \odot \pi_1 \odot \pi_2$, we get $\big(\langle p',w'\rangle,\bar{1}\big) \Rightarrow_K \big(\langle p,w\rangle,\pi\big)$, where $\langle p',w'\rangle$ is such that $p' \xrightarrow{w'} q$ is a run of $\mathcal{A}$. i.e. $\big(\langle p',w'\rangle,\bar{1}\big) \Rightarrow_K \big(\langle p,w\rangle,\pi\big)$, since $w = \gamma w_1$ and $\pi = v_a \odot \pi_1 \odot \pi_2$.

- The case where $\rho = p \xrightarrow{\epsilon} q' \xrightarrow{w_1} q$ is similar. In this case, r is of the form $r = (p_1,\gamma_1) \overset{a}{\hookrightarrow} (p,\epsilon)$.
- There exist $\gamma_1,\gamma_2 \in \Gamma$, $w_1 \in \Gamma^*$, a rule r, and $q' \in Q$ such that $w = \gamma_1\gamma_2 w_1$ and

$$\rho = p \xrightarrow{\gamma_1} r \xrightarrow{\gamma_2} q' \xrightarrow{w_1} q.$$

π satisfies $\pi \leq \lambda\big((p,\gamma_1,r)\big) \odot \lambda\big((r,\gamma_2,q')\big) \odot \lambda\big((q',w_1,q)_{\rho_1}\big)$, where $\rho_1 = q' \xrightarrow{w_1} q$. According to the inequalities (β_1) and (β_4), r is of the form $r = (p_1,\gamma) \overset{a}{\hookrightarrow} (p,\gamma_1\gamma_2)$ of Δ, and two path-expressions π_1 and π_2 such that $\pi = \bar{1} \odot v_a \odot \pi_1 \odot \pi_2$, $v_a \odot \pi_1 \leq \lambda\big((r,\gamma_2,q')\big)$, $\pi_1 \leq \lambda\big((p_1,\gamma,q')_{\rho_2}\big)$, and $\pi_2 \leq \lambda\big((q',w_1,q)_{\rho_1}\big)$, where ρ_2 is a path of the form $\rho_2 = p_1 \overset{\gamma}{\rightsquigarrow} q'$. Indeed, the inequalities infer that $\lambda\big((p,\gamma_1,r)\big) = \bar{1}$.

Let $\rho' = p_1 \overset{\gamma}{\rightsquigarrow} q' \xrightarrow{w_1} q$ be the corresponding run of $\mathcal{A}_{post^*}$. It follows that

$$\begin{aligned}\pi_1 \odot \pi_2 &\leq \lambda\big((p_1,\gamma,q')_{\rho_2}\big) \odot \lambda\big((q',w_1,q)_{\rho_1}\big)\\ &= \lambda\big((p_1,\gamma w_1,q)_{\rho'}\big)\end{aligned}$$

Sine $|\pi_1 \odot \pi_2| < |\pi|$, the induction hypothesis implies that there exists a configuration $\langle p',w'\rangle$ such that $p' \xrightarrow{w'} q$ is a run of $\mathcal{A}$ and

$$\big(\langle p',w'\rangle,\bar{1}\big) \Rightarrow_K \big(\langle p_1,\gamma w_1\rangle, \pi_1 \odot \pi_2\big).$$

Moreover, by applying r we get that

$$\big(\langle p_1,\gamma w_1\rangle, \pi_1 \odot \pi_2\big) \Rightarrow_K \big(\langle p,\gamma_1\gamma_2 w_1\rangle, v_a \odot \pi_1 \odot \pi_2\big)$$

i.e. $\big(\langle p',w'\rangle,\bar{1}\big) \Rightarrow_K \big(\langle p,w\rangle,\pi\big)$, since $w = \gamma_1\gamma_2 w_1$ and $\pi = v_a \odot \pi_1 \odot \pi_2$.

$\square$