Pushdown Systems for Monotone Frameworks*

Michał Terepeta, Hanne Riis Nielson, and Flemming Nielson

Technical University of Denmark
{mtte,riis,nielson}@imm.dtu.dk

Abstract. Monotone frameworks is one of the most successful frameworks for intraprocedural data flow analysis extending the traditional class of bitvector frameworks (like live variables and available expressions). Weighted pushdown systems is similarly one of the most general frameworks for interprocedural analysis of programs. However, it makes use of idempotent semirings to represent the sets of properties and unfortunately they do not admit analyses whose transfer functions are not strict (e.g., classical bitvector frameworks).

This motivates the development of algorithms for backward and forward reachability of pushdown systems using sets of properties forming so-called flow algebras that weaken some of the assumptions of idempotent semirings. In particular they do admit the bitvector frameworks, monotone frameworks, as well as idempotent semirings. We show that the algorithms are sound under mild assumptions on the flow algebras, mainly that the set of properties constitutes a join semi-lattice, and complete provided that the transfer functions are suitably distributive (but not necessarily strict).

1 Introduction

Monotone frameworks [1] is a unifying approach to static analysis of programs. It creates a generic foundation for specifying various analyses and by imposing very modest requirements can accommodate a wide range of analyses, including the bitvector frameworks as well as more complex ones such as constant propagation. However, the original formulation was focused on the intraprocedural setting and did not discuss the interprocedural one.

Interprocedural analysis has always been an interesting challenge for static analysis. Two of the main reasons for that are the unbounded stack and recursive (or mutually recursive) procedures. Moreover, only some paths in the interprocedural flow graph are valid — the call and returns should match. All of this opens up many possibilities for various trade-offs, such as taking into account or ignoring the calling context. In their seminal work Sharir and Pnueli [2] presented two approaches allowing for precise interprocedural analysis. One of them, known as the call-strings approach, is based on “tagging” the analysis information with the current call stack. Obviously the length of call-strings should be limited to

* The research presented in this paper has been supported by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology.
some threshold in order to ensure the termination of the analysis. However, in this paper we will be more interested in the other presented approach. It is called the functional approach and is based on the idea of computing the summarizations of procedures, i.e., establishing the relationships between the inputs and the outputs of the blocks of the program and procedures (composing the results for the blocks). A similar idea, from the abstract interpretation perspective, was explored in [3], which considered predicate transformers as the basis for the analysis and also involved constructing systems of functional equations.

Pushdown systems [4,5,6] are one of the more recently proposed approaches to interprocedural analysis. One of the underlying ideas behind them is to use a construction similar to pushdown automata to model the use of the stack by a program. An interesting advantage of the approach is the ability to compute the (possibly) infinite sets of predecessor and successor configurations for a given program and some initial configurations. Since the pushdown systems can only handle programs with finite abstractions, they have been extended with semiring weights/annotations in weighted pushdown systems [7,8,9] and communicating pushdown systems [10,11]. The extensions proposed in both of these approaches are actually very close, although the former focuses on dataflow analysis and generalizing the functional approach to interprocedural analysis, while the latter on the abstractions of language generated by synchronization actions in a concurrent setting. Pushdown systems have been used for verification purposes in many different projects and contexts. The examples include the Moped [6] and jMoped [12] model checkers that extensively use pushdown systems or Codesurfer [13] that takes advantage of weighted pushdown systems.

However, both the WPDS and CPDS use semiring structure for analysis purposes and therefore exclude many classical approaches, such as bitvector frameworks where the transfer functions are not strict. In this paper we are bringing the pushdown systems based analysis closer to the monotone frameworks. To achieve that we use the concept of flow algebra [14] that is a structure similar to semiring, but more permissive. In particular we do not impose the annihilation requirement, nor the distributivity. This allows us to present examples of classical analyses that thanks to our extensions are admitted by the framework, and did not directly fit into the previous semiring-based approaches. Since the existing algorithms are based on the assumption of working with semiring structure, we develop our slightly different algorithms that allow us to relax the requirements. Then we go on to establish the soundness result, i.e., the analysis result safely over-approximates the join over all valid paths of the pushdown system. Furthermore, we also prove the completeness of the analysis, that is, provided that the flow algebra satisfies certain additional properties the result of the analysis will coincide with the join over all valid paths.

The structure of the paper is as follows. In Sec. 2 we recall and introduce the necessary concepts, e.g., monotone frameworks, pushdown systems including both the weighted and communicating variants. Then in Sec. 3 we present basic

\footnote{Although it is possible to sidestep this problem by introducing an “artificial” annihilator to the semiring.}
definitions, while in Sec. 4 we describe our algorithms and provide some intuition behind them. Then Sec. 5 presents the soundness result for both the forward and backward reachability. Similarly Sec. 6 describes the completeness result for both of them. Finally, we discuss the results and provide some examples in Sec. 7 and conclude in Sec. 8.

2 Monotone Frameworks, Semirings and Flow Algebras

In this section we will present the basic definitions that will be used throughout the rest of the paper. We will start with recalling the classical approach to static analysis known as monotone frameworks [1, 15]. Here we present a slightly more convenient (in the context of this paper) definition of monotone framework.

**Definition 1.** A complete monotone framework is a tuple

\[ (L, \bigvee, F, \circ, \text{id}, (f_l)_{l \in L}) \]

where \( L \) is a complete lattice, \( \bigvee \) is its least upper bound operator. We use \( F \) to denote a monotone function space on \( L \), i.e., a set of monotone functions that contains the identity function and is closed under function composition. Finally, \( \circ \) is function composition, \( \text{id} \) is the identity function and \( f_l = \lambda l'. l \) for every \( l \in L \).

We will also discuss bitvector frameworks, which are a special case of monotone frameworks. The lattice used is \( L = \mathcal{P}(D) \) for some finite set \( D \), the ordering is either \( \subseteq \) or \( \supseteq \) and the least upper bound is either \( \cup \) or \( \cap \) and the monotone and distributive function space is defined as

\[ \{ f : \mathcal{P}(D) \to \mathcal{P}(D) \mid \exists Y_1, Y_2 \subseteq D : \forall Y \subseteq D : f(Y) = (Y \cap Y_1) \cup Y_2 \} \]

One of the main reasons for distinguishing them is the fact that they can be implemented very efficiently using bitvectors and include common analyses such as live variables, available expressions, reaching definitions, etc.

Since both weighted and communicating pushdown systems are using semirings, we will introduce some of the basic definitions associated with them [16], starting with the definition of a monoid.

**Definition 2.** A monoid is a tuple \( (M, \otimes, \bar{1}) \) such that \( M \) is non-empty, \( \otimes \) is an associative operator on \( M \) and \( \bar{1} \) is a neutral element for \( \otimes \), i.e.,

\[ \forall a \in M : a \otimes \bar{1} = \bar{1} \otimes a = a \]

A monoid is idempotent if \( \otimes \) operator is idempotent, that is

\[ \forall a \in M : a \otimes a = a \]

Similarly it is commutative if the operator is commutative, in which case we usually use the symbol \( \oplus \) to denote it (and also use \( \bar{0} \) for the neutral element).

\[ \forall a, b \in M : a \oplus b = b \oplus a \]
A commutative monoid \((M, \oplus, \bar{0})\) is naturally ordered if the relation defined as
\[\forall a, b \in M : a \sqsubseteq b \iff \exists c \in M : a \oplus c = b\]
is a partial order. Moreover, if the monoid is idempotent then it is naturally ordered and we have that
\[\forall a, b \in M : a \sqsubseteq b \iff a \oplus b = b\]
and \(\oplus\) is the least upper bound operator. Note that this corresponds to a join semi-lattice.

Now we are ready do define the semiring structure.

**Definition 3.** A semiring is a tuple \((S, \oplus, \otimes, \bar{0}, \bar{1})\) such that

- \((S, \oplus, \bar{0})\) is a commutative monoid (hence \(\bar{0}\) is a neutral element for \(\oplus\))
- \((S, \otimes, \bar{1})\) is a monoid (hence \(\bar{1}\) is a neutral element for \(\otimes\))
- \(\otimes\) distributes over \(\oplus\), that is
  \[a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)\]
  \[(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)\]
- \(\bar{0}\) is an annihilator for \(\otimes\), that is \(a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}\)

Similarly to the above, we call a semiring idempotent if \(\oplus\) is idempotent, and commutative if \(\otimes\) is commutative. The ordering for idempotent semiring is defined in the same way as for idempotent and commutative monoids, with the additional requirement that \(\otimes\) preserves the order (i.e., is monotonic).

As already mentioned we will use the notion of a flow algebra \([14]\), which is similar to idempotent semirings, but less restrictive. The main difference is that flow algebras do not require the distributivity and annihilation properties. Instead we replace the first one with a monotonicity requirement and dispense with the second one. It is formally defined as follows.

**Definition 4.** A flow algebra is a structure of the form \((F, \oplus, \otimes, \bar{0}, \bar{1})\) such that:

- \((F, \oplus, \bar{0})\) is an idempotent and commutative monoid
- \((F, \otimes, \bar{1})\) is a monoid
- \(\otimes\) is monotonic in both arguments, that is:
  \[f_1 \sqsubseteq f_2 \Rightarrow f_1 \otimes f \sqsubseteq f_2 \otimes f\]
  \[f_1 \sqsubseteq f_2 \Rightarrow f \otimes f_1 \sqsubseteq f \otimes f_2\]

where \(f_1 \sqsubseteq f_2\) if and only if \(f_1 \oplus f_2 = f_2\).

\(^2\) The name comes from the idea of performing dataflow analyses using an algebraic structure.
Clearly in a flow algebra all finite subsets \( \{ f_1, \ldots, f_n \} \) have a least upper bound, which is given by \( \bar{0} \oplus f_1 \oplus \cdots \oplus f_n \).

Since the assumptions on a flow algebra are less demanding than in the case of semirings, we additionally introduce the notions of distributive and strict flow algebras.

**Definition 5.** A distributive flow algebra is a flow algebra \((F, \oplus, \otimes, \bar{0}, \bar{1})\), where \( \otimes \) distributes over \( \oplus \) on both sides, i.e.,

\[
\begin{align*}
    f_1 \otimes (f_2 \oplus f_3) &= (f_1 \otimes f_2) \oplus (f_1 \otimes f_3) \\
    (f_1 \oplus f_2) \otimes f_3 &= (f_1 \otimes f_3) \oplus (f_2 \otimes f_3)
\end{align*}
\]

We also say that a flow algebra is strict if

\[
\bar{0} \otimes f = \bar{0} = f \otimes \bar{0}
\]

**Fact 1** Every idempotent semiring is a strict and distributive flow algebra.

One of the motivations of flow algebras is that the classical bit-vector frameworks \cite{15} are not strict; hence they are not directly expressible using idempotent semirings. Therefore, from this perspective the flow algebras are closer to Monotone Frameworks, and other classical static analyses. Restricting our attention to semirings rather than flow algebras would mean restricting attention to strict and distributive frameworks.

**Definition 6.** A complete flow algebra is a flow algebra \((F, \oplus, \otimes, \bar{0}, \bar{1})\), where \( F \) is a complete lattice; we write \( \bigoplus \) for the least upper bound. It is affine \cite{15} if for all non-empty subsets \( F' \neq \emptyset \) of \( F \)

\[
\begin{align*}
    f \otimes \bigoplus F' &= \bigoplus \{ f \otimes f' \mid f' \in F' \} \\
    \bigoplus F' \otimes f &= \bigoplus \{ f' \otimes f \mid f' \in F' \}
\end{align*}
\]

Furthermore, it is completely distributive if it is affine and strict.

If the complete flow algebra satisfies the ascending chain condition \cite{15} then it is affine if and only if it is distributive.

Let us emphasize the connection between the flow algebras and the monotone frameworks. As defined above a complete monotone framework is

\[
(L, \bigcup, F, \circ, id, (f_i)_{i \in L})
\]

Note that this immediately gives us a flow algebra by taking

\[
(F, \bigcup, \bar{\circ}, f_\perp, id)
\]

where \( \bigcup Y = \lambda l. \bigcup_{f \in Y} f(l) \) and \( f \bar{\circ} g = g \circ f \).
3 Pushdown Systems

In order to present our results, it is necessary first to introduce some basic definitions related to pushdown systems as well as weighted/communicating pushdown systems.

3.1 Introduction to Pushdown Systems

We will start with recalling some of the basic definitions of pushdown systems and their extensions with semiring weights, namely weighted pushdown systems (WPDS) [7,8] and communicating pushdown systems (CPDS) [10,11]. We will mostly follow the notation used for WPDS (note that CPDS use slightly different notation, but the intent is basically the same in both approaches — one equips every pushdown rule with a semiring weight).

**Definition 7.** A pushdown system is a tuple $\mathcal{P} = (P, \Gamma, \Delta)$ where $P$ is a finite set of control locations, $\Gamma$ is a finite set of stack symbols and $\Delta$ is a finite set of pushdown rules of the form $\langle p, \gamma \rangle \rightarrow \langle p', w \rangle$, where $w \in \Gamma^*$ and $|w| \leq 2$.

Note that the requirement $|w| \leq 2$ is not a serious restriction and any pushdown system can be transformed to satisfy it. This can be achieved by adding some fresh control locations and pushing $|w|$ in a few steps. The above is already quite enough for checking the reachability of finite abstractions of programs. The valuation of global variables can be encoded using control locations $P$ (and the local variables, if needed, in the stack alphabet $\Gamma$).

Clearly a pushdown system gives rise to a (possibly infinite) transition systems, where we can move between configurations using the pushdown rules. The transition relation for this system is defined more formally below. For every pushdown rule $r = \langle p_1, \gamma \rangle \rightarrow \langle p_2, u \rangle$ we have

$$\langle p_1, \gamma w \rangle \xrightarrow{r} \langle p_2, uw \rangle$$

for all $w \in \Gamma^*$. Sometimes we will omit the annotation of the specific pushdown rule — this means that we assume there exists a rule that allows moving between the given configurations. The reflexive, transitive closure of $\xrightarrow{}$ will be denoted as $\xrightarrow{*}$ (and annotated with sequences of pushdown rules). Having a precise definition of the transition relation (and its reflexive transitive closure) allows us to define the concepts of successor and predecessor configurations. We call a configuration $c_2$ an immediate successor (predecessor) of $c_1$ if $c_1 \xrightarrow{} c_2$ ($c_2 \xleftarrow{} c_1$). Similar to immediate successors (predecessors) one can also define the general successors (predecessors) using the $\xrightarrow{*}$, namely a configuration $c_2$ a successor (predecessor) of $c_1$ if $c_1 \xrightarrow{*} c_2$ ($c_2 \xleftarrow{*} c_1$).

In many verification problems it is desirable to talk about the sets of successors or predecessors of a given configuration or set of configurations. They are often denoted as $\text{Pre}^*(C)$ and $\text{Post}^*(C)$ respectively, where $C$ is some set of
configurations. More formally:

\[ \text{Pre}^*(C) = \{ c_2 \mid c_2 \Rightarrow^* c_1, c_1 \in C \} \]
\[ \text{Post}^*(C) = \{ c_2 \mid c_1 \Rightarrow^* c_2, c_1 \in C \} \]

Note that those sets can be in general infinite (even if \( C \) is finite). In order to compute the sets of successors and predecessor we need some symbolic representation. Therefore, we define the following.

**Definition 8.** Given a pushdown system \( P = (P, \Gamma, \Delta) \) a \( P \)-automaton is a tuple \((Q, \Gamma, \to, P, F)\), where:

- \( Q \) is a finite set of states such that \( P \subseteq Q \)
- \( \to \subseteq Q \times \Gamma \times Q \) is a finite set of transitions
- \( P \subseteq Q \) is a finite set of initial states
- \( F \subseteq Q \) is a finite set of final states

We denote the transitive closure of \( \to \) as \( \to^* \). Then we say that a \( P \)-automaton accepts a configuration \((p, s)\) if and only if \( p \xrightarrow{\to^*} q \) where \( q \in F \). Moreover, a set of configurations is regular if it is accepted by some \( P \)-automaton.

One of the crucial results in the pushdown systems says that the sets of successors or predecessors of a regular set of configurations are regular themselves [4,5,6]. This is essential since it guarantees that we can always represent those sets as \( P \)-automata. Therefore, the algorithms for \( \text{Pre}^* \) and \( \text{Post}^* \) take as input a pushdown system and an initial automaton \( A \) that represents the set of configurations whose predecessors or successors we want to compute. Both algorithms are basically saturation procedures, i.e., they keep adding new transitions to the \( A \) according to some rule until no further transitions (or constraints) can be added. Since the number of possible transitions is finite (in \( \text{Pre}^* \) the algorithm does not add any new states, and in \( \text{Post}^* \) always a bounded number of them), the algorithms must terminate and return the \( A_{\text{pre}}^* \) or \( A_{\text{post}}^* \), which represent the possibly infinite number of reachable configurations.

### 3.2 Weighted and Communicating Pushdown Systems

This approach requires that the sets \( P \) and \( \Gamma \) are finite, which makes it impossible to use infinite abstractions. To make it possible to use such abstractions, the papers [7,8,10,11] equipped every pushdown rule with a semiring value. As already mentioned we will mostly follow the notation from WPDS, and thus we present its slightly modified definition below.

**Definition 9.** A weighted pushdown system a tuple \( W = (P, S, f) \), where \( P \) is a pushdown system, \( S = (S, \oplus, \otimes, 0, 1) \) is an idempotent flow algebra and \( f : \Delta \to S \) maps pushdown rules to the elements of \( S \).

The main difference when compared to the original definition is that we require a flow algebra instead of bounded and idempotent semiring.\(^3\) Now we can use

---

3 Bounded is used to mean that it contains no infinite ascending chains [7,8].
the fact that every pushdown rule has a flow algebra weight to define the weight of a sequence of pushdown rules. Let $\sigma = [r_1, \ldots, r_n] \in \Delta^*$ be such a sequence, then we define $v(\sigma) = f(r_1) \otimes \cdots \otimes f(r_n)$. Moreover, the papers extended the algorithms for $\text{Pre}^*$ and $\text{Post}^*$ (in slightly different ways in case of WPDS and CPDS) to handle the addition of weights. The result is that both the $\mathcal{A}_{\text{pre}^*}$ and $\mathcal{A}_{\text{post}^*}$ return weighted NFAs, i.e., where each transition is annotated with a weight. Apart from making it possible to answer reachability queries, they also provide additional dataflow information for the given configuration. In other words, we can not only ask whether a configuration is a successor or predecessor but also what is the flow algebra value of getting from that configuration ($\text{Pre}^*$) or to that configuration ($\text{Post}^*$). More formally, we additionally compute the following information:

- in case of predecessors of some regular set of configurations $C$ (i.e., if $c_1$ is a predecessor of some configuration in $C$)

$$\delta(c_1) = \bigoplus \{v(\sigma) \mid c_1 \xrightarrow{\sigma} c_2, c_2 \in C\}$$

is the flow algebra value of all the paths going from configuration $c_1 = \langle p, s \rangle$ ($s \in \Gamma^*$) to any configuration in $C$. It can be obtained by simulating $\mathcal{A}_{\text{pre}^*}$ from state $p$ with input $s$ multiplying the weights of the transitions in the same order as they are taken.

- in case of successors of some regular set of configurations $C$ (i.e., if $c_1$ is a successor of some configuration in $C$)

$$\delta(c_1) = \bigoplus \{v(\sigma) \mid c_2 \xrightarrow{\sigma} c_1, c_2 \in C\}$$

is the flow algebra value of all the paths going from any configuration in $C$ to $c_1 = \langle p, s \rangle$ ($s \in \Gamma^*$). It can be obtained by simulating $\mathcal{A}_{\text{post}^*}$ from state $p$ with input $s$ multiplying the weights of the transitions in the reverse order as they are taken.

Note that in both cases we only want to calculate the value for a predecessor or successor, thus the sets of paths are never empty. In case of $\text{Post}^*$, the intuition behind reading the weights of a path in the automaton in the reverse order is that when a configuration $\langle p, \gamma_k \ldots \gamma_1 \rangle$ is accepted, this means that there are transitions in the automaton such that the first one is labeled with $\gamma_k$, the second with $\gamma_{k-1}$ and so on. However, when one thinks how the program would actually execute, it would build the stack from the other end, i.e before it can push $\gamma_2$ on the stack, it must push $\gamma_1$. Therefore, the weights should be multiplied in the reverse order.

4 Algorithms

As already mentioned, WPDS and CPDS are assuming that the abstract domain forms a semiring structure. This immediately excludes standard analyses based
on monotone framework or bitvector framework. Fortunately we will show that it is possible to formulate algorithms for $Pre^*$ and $Post^*$ that do not need this assumption. We achieve that by generating the constraints during the saturation procedures that create the $A_{pre^*}$ and $A_{post^*}$ (in WPDS no constraints are generated and the weights are calculated directly, in CPDS constraints are generated independently of the $A_{pre^*}$ and $A_{post^*}$ construction). We will use $A_{pre^*}^C$ and $A_{post^*}^C$ to denote the automata with the associated set of constraints $C$. The rest of the section will introduce the algorithms and in the subsequent sections we will discuss their soundness and completeness. In this way we believe that we can present the minimum requirements that are necessary for interprocedural analysis based on pushdown systems.

4.1 Algorithm for $Pre^*$

The procedure introduced in this section is quite similar to the one from [10,11] as it generates explicit constraints. However, it does it during the automaton computation not separately. In this respect it is somewhat similar to the procedure from [7,8] that computes both the weights and the automaton at the same time. Also, note that there is no difference with respect to how the new transitions are added to the automaton. Therefore, we are able to reuse the standard results with respect to the automaton itself (i.e., excluding the weights).

The algorithm is as follows. First, for every transition $q \overset{\gamma}{\rightarrow} q'$ in $\mathcal{A}$ we add a constraint

$$\overline{1} \subseteq l(q \overset{\gamma}{\rightarrow} q')$$

(we use $l(-)$ in the constraints to denote the weight of the given transition). Then we perform the saturation procedure on $\mathcal{A}$ along with the generation of constraints that are added to $C$. For every pushdown rule $r$ in $\Delta$:

- if $r = (p, \gamma) \leftrightarrow (p', \epsilon)$ we add a transition

  $$p \overset{\gamma}{\rightarrow} p'$$

  along with the following constraint

  $$f(r) \subseteq l(p \overset{\gamma}{\rightarrow} p')$$

- if $r = (p, \gamma) \leftrightarrow (p', \gamma')$ and there is a transition $p' \overset{\gamma'}{\rightarrow} q$ in the current automaton, we add a transition

  $$p \overset{\gamma}{\rightarrow} q$$

  along with the following constraint

  $$f(r) \otimes l(p' \overset{\gamma'}{\rightarrow} q) \subseteq l(p \overset{\gamma}{\rightarrow} q)$$
– if $r = \langle p, \gamma \rangle \xrightarrow{\gamma'} q' \xrightarrow{\gamma''} q$ (for some $q'$) in the current automaton, we add a transition

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

along with the following constraint

$$f(r) \otimes l(q' \xrightarrow{\gamma'} q') \otimes l(q' \xrightarrow{\gamma''} q) \subseteq l(p \xrightarrow{\gamma} q)$$

We stop once we cannot add any new constraints or transitions. And since the number of possible transitions and constraints is finite, the procedure will always terminate.

### 4.2 Algorithm for Post*

As in the case of Pre* algorithm, we only change the way the constraints are generated, and not how new transitions are added to the automaton. Recall that we require the initial automaton $A$ to have no transitions going into the initial states nor any $\epsilon$-transitions. As already mentioned, the weights of a $A_{post^*}$ should be multiplied in the reverse order compared to the take transitions. Therefore, we will use the reverse arrow notation for the transitions of the automata, i.e., we will write $q \xleftarrow{\gamma} p$ for the transition earlier denoted by $p \xrightarrow{\gamma} q$. Furthermore, as already noted in [10,11] the $\epsilon$-transitions added by the algorithm always originate in an initial state and go only to some non-initial state. Therefore, we can conclude that we can take at most one $\epsilon$-transition (when going from initial state to some non-initial one) and then we can only take non $\epsilon$-transitions. Therefore, let us use $\epsilon\gamma -$ to denote $(\gamma - \circ \epsilon) \cup \epsilon -$ and define $h^\epsilon$ as

$$h^\epsilon(p) = \begin{cases} h(q \xrightarrow{\gamma} p) & \text{if } \rho = q \xrightarrow{\gamma} p \\ h(q \xleftarrow{\gamma} q') \otimes h(q' \xleftarrow{\epsilon} p) & \text{if } \rho = q \xleftarrow{\gamma} q' \xleftarrow{\epsilon} p \end{cases}$$

The algorithm is as follows. First, for every transition $q' \xleftarrow{\gamma} q$ in $A$ we add a constraint

$$1 \subseteq h(q' \xleftarrow{\gamma} q)$$

Then for all pushdown rules of the form $\langle p, \gamma \rangle \xrightarrow{\gamma'} q' \xrightarrow{\gamma''} q$ we add a new state $q_{p',\gamma'}$ to the automaton. Finally, for every pushdown rule $r$ in $\Delta$:

– if $r = \langle p, \gamma \rangle \xrightarrow{\gamma' \epsilon} q'$ and there is a path $\rho = q \xleftarrow{\gamma -} p$ then add a transition

$$q \xleftarrow{\gamma \epsilon} p'$$

along with the following constraint

$$h^\epsilon(q \xleftarrow{\gamma \epsilon \rho} p) \otimes f(r) \subseteq h(q \xleftarrow{\epsilon} p')$$

Note that this transition (and its weight) takes care of the return from a procedure.
- if \( r = \langle p, \gamma \rangle \mapsto \langle p', \gamma' \rangle \) and there is a path \( \rho = q \leadsto p \) then add a transition
  \[ q \xrightarrow{\gamma'} p' \]
  along with the following constraint
  \[ h^\epsilon(q \xrightarrow{\gamma'} p) \otimes f(r) \sqsubseteq h(q \xrightarrow{\gamma'} p') \]

- if \( r = \langle p, \gamma \rangle \mapsto \langle p', \gamma' \gamma'' \rangle \) and there is a path \( \rho = q \leadsto p \) then add transitions
  \[ q \xrightarrow{\gamma''} q_{p', \gamma'}, \quad q_{p', \gamma'} \xrightarrow{\gamma'} p' \]
  along with the following constraints
  \[ \mathbf{1} \sqsubseteq h(q_{p', \gamma'} \xrightarrow{\gamma'} p') \]
  \[ h^\epsilon(q \xrightarrow{\gamma'} p) \otimes f(r) \sqsubseteq h(q \xrightarrow{\gamma''} q_{p', \gamma'}) \]

Note that this transition \( q \xrightarrow{\gamma''} q_{p', \gamma'} \) (and its weight) takes care of the procedure call.

Again, as in the case of \( Pre^* \) we stop once we cannot add any new constraints or transitions. And since the number of possible transitions and constraints is finite, the procedure will always terminate (note that we add some new states only at the beginning of the procedure and not in the saturation phase).

### 5 Soundness

In this section we will discuss and present the main results regarding the soundness of our algorithms. Since one of the goals of our formulation of the algorithms is to make the requirements imposed on the abstract domain explicit and precise, we take a particular approach to the soundness proofs. We do not discuss how the generated constraints can be solved (and if they can be solved at all). Instead we assume that some solution to those constraints is available and show that it is a safe over-approximation of the join over all valid paths.

Apart from that, separating the requirements necessary to solve the constraints from the soundness result gives us the flexibility to easily accommodate different techniques of solving the constraints. One can use the usual Kleene iteration, but also more recent approaches using Newton’s method generalized to \( \omega \)-continuous semirings \cite{17,18}. Furthermore, it also makes it clear that techniques such as widening can be used for domains that contain infinite ascending chains but do not satisfy the requirements of Newton’s method.
5.1 $Pre^*$

We will start with some intuition about how the pushdown system $P$ and the automaton $A$ fit together. Observe that if a configuration is backward reachable from $C$, there exists a sequence of pushdown rules in $\Delta$ such that the resulting configuration is accepted by $A$. Therefore, we can intuitively think about this system as a one big pushdown system $PA = (P, \Gamma, \Delta_{pre})$, where

$$\Delta_{pre} = \Delta \cup \{(q, \gamma) \rightarrow (q', \epsilon) \mid q \xrightarrow{\gamma} q' \in \rightarrow\}$$

With each added pushdown rule we associate the weight $\bar{1}$. This system works by first acting like $P$ and then, at some point, switching to simulating $A$ (with the added pushdown rules). Note that once $PA$ starts using the added pushdown rules, it cannot use the ones of $P$. This is because rules in $P$ correspond to the initial states of $A$ and since it does not have any transitions going to initial states, then the first used rule from $\Delta_{pre} \setminus \Delta$ will go to some non-initial state. Thus no pushdown rule of $P$ will be applicable.

This is useful because it allows us to look at the problem of predecessors of $C$ from a slightly different angle. Let us consider the automaton $A_{pre}^*$, we say that a configuration $c_p$ is a predecessor of some configuration $c \in C$ if there is a sequence $\sigma \in \Delta^*$ of pushdown rules such that $c_p \xrightarrow{\sigma} c$. But since $c$ is recognized by $A$ then there is a sequence $\sigma' \in \Delta_{pre}^*$ such that $c \xrightarrow{\sigma'} (q_f, \epsilon)$ for some final state $q_f$. Therefore, an alternative way to define a predecessor is to say that a configuration $c_p$ is a predecessor of some configuration $c \in C$ if there is a sequence $\sigma_p \in \Delta_{pre}^*$ of pushdown rules such that $c_p \xrightarrow{\sigma} (q_f, \epsilon)$ for some state $q_f \in F$. Moreover, since we have that each of the added rules has weight $\bar{1}$ then $v(\sigma) = v(\sigma_p)$.

In the following sections the solution to the constraints will be denoted as $\lambda$ (i.e., maps each transition to its weight). Its generalization to paths $\lambda^*$ is inductively defined as follows:

$$\lambda^*(\rho) = \begin{cases} 
\lambda(q \xrightarrow{\gamma} q') & \text{if } \rho = q \xrightarrow{\gamma} q' \\
\lambda(q \xrightarrow{\gamma} q'') \otimes \lambda^*(\rho') & \text{if } \rho = q \xrightarrow{\gamma} q' \xrightarrow{\gamma'} q'' \xrightarrow{\gamma''} q' 
\end{cases}$$

Now we are ready to prove that a solution to the constraints generated by our saturation procedure is sound.

**Theorem 1.** Consider an automaton $A$ and its corresponding $A_{pre}^C$ generated by the saturation procedure. Let us assume that we have a solution $\lambda$ to the set of constraints $C$. Then for each pair $(p, s)$ such that $\langle p, s \rangle \xrightarrow{\sigma}^* \langle q_f, \epsilon \rangle$ (where $\sigma \in \Delta_{pre}^*$ and $q_f \in F$), we have $v(\sigma) \subseteq \lambda^*(\rho)$ where $\rho = p \xrightarrow{\sigma}^* q_f$ is in $A_{pre}^*$.

**Proof.** The proof is available in App. A.1
5.2 Post

As previously we can think about this system as a one big pushdown system. However, this time such a system would first simulate the reverse of $A$, i.e., instead of accepting some configuration, it generates one; and only then continue by running the pushdown system itself. Let us denote such a system as $A^R\mathcal{P} = (P, \Gamma, \Delta_{post})$, where $\Delta_{post}$ is defined as follows.

- For every $q' \xleftarrow{\gamma} q$ in $A$ we have a rule $r = \langle q', \epsilon \rangle \rightarrow \langle q, \gamma \rangle$ in $\Delta_{post}$ such that $f(r) = 1$.
- All other rules of $\Delta$ are included in $\Delta_{post}$.

Let us consider the automaton $A_{post}^\ast$. We say that a configuration $c'$ is a successor of some configuration $c$ in $C$ if there is a sequence $\sigma \in \Delta_{post}^\ast$ of pushdown rules such that $c \xrightarrow{\sigma} \sigma c$. But since $c$ is recognized by $A$ then there is a sequence $\sigma_p \in \Delta_{post}^\ast$ of pushdown rules such that $\langle q_f, \epsilon \rangle \xrightarrow{\sigma_p} c$ for some final state $q_f$. Therefore, an alternative way to define a successor is to say that a configuration $c'$ is a successor of some configuration $c \in C$ if there is a sequence $\sigma_p \in \Delta_{post}^\ast$ of pushdown rules such that $\langle q_f, \epsilon \rangle \xrightarrow{\sigma_p} c$ for some state $q_f \in F$. Moreover, since we have that each of the added rules has weight $\bar{1}$ then $v(\sigma) = v(\sigma_p)$.

Similarly as in the case of Pre$^\ast$, we define $\lambda_R^\ast$ in the following way:

$$
\lambda_R^\ast(p) = \begin{cases} 
\lambda(q \xleftarrow{\gamma} q') & \text{if } \rho = q' \xleftarrow{\gamma} q \\
\lambda_R^\ast(p') \otimes \lambda(q \xleftarrow{\gamma} q'') & \text{if } \rho = q' \mathbin{\*} q'' \xleftarrow{\gamma} q 
\end{cases}
$$

As already mentioned we multiply the weight in the reverse order compared to the order of transitions in the given path.

**Theorem 2.** Consider an automaton $A$ and its corresponding $A_{post}^C\mathcal{P}$ generated by the saturation procedure. Let us assume that we have a solution $\lambda$ to the set of constraints $C$. Then for each pair $(p, s)$ such that $\langle q_f, \epsilon \rangle \xrightarrow{\sigma} \sigma \langle p, s \rangle$ (where $\sigma \in \Delta_{post}^\ast$ and $q_f \in F$), we have $v(\sigma) \subseteq \lambda_R^\ast(p)$ where $\rho = q_f \mathbin{\*} p$ is in $A_{post}^C\mathcal{P}$.

**Proof.** The proof is available in App. A.2.

### 6 Completeness

In this section we will prove the completeness of our procedure, i.e., we will show that provided the abstract domain satisfies certain conditions, the solution to the generated constraints will coincide with the join over all valid paths. The presentation of the results (and their proofs) is quite a bit different than in the case of soundness. This is mainly due to the additional complexity of the proofs as well as some additional restrictions that must to be imposed. Throughout the whole section we assume that the flow algebra is both complete and affine. In
other words we have least upper bounds of arbitrary sets and \( \otimes \) distributes over sums of all non-empty sets.

Before we present the main results for each of the two algorithms, let us first establish that the solution to the generated constraints can be obtained by Kleene iteration. To achieve that we will define a function that represents the constraints and show that it is continuous. Let us recall that all generated constraints are of similar form: the right-hand side is a variable and the left-hand side is a finite expression mentioning at most two variables. The finite expressions are constructed using \( \oplus \) and \( \otimes \) which are themselves affine and hence continuous.

For clarity let \( \mathcal{C}_t \subseteq \mathcal{C} \) denote the finite set of the constraints that have the variable \( t \) on the right-hand side. Recall that each variable corresponds to a transition in an automaton. Similarly we will use \( \text{lhs}_m(c) (c \in \mathcal{C}) \) to denote the interpretation of the left-hand side of the constraint \( c \) under the assignment \( m \).

What we want to compute is a mapping \( m \) that is a fixed point of:

\[
F : (\delta \to D) \to (\delta \to D)
\]

\[
F(m)t = \bigoplus_{c \in \mathcal{C}_t} \text{lhs}_m(c)
\]

where \( \delta \) is the set of all transitions.

**Lemma 1.** \( F \) is continuous, i.e., for any non-empty chain \( Y \):

\[
F\bigcup Y = \bigcup_{m \in Y} F(m)
\]

**Proof.** The proof is available in App. B.

It follows that \( \bigcup \{ F^n(\bot) | n \in \mathbb{N} \} \) is the least solution to our constraint system.

### 6.1 \( \text{Pre}^* \)

We will first establish a lemma showing that every transition in the \( \mathcal{A}_{\text{pre}} \) automaton has at least one corresponding path in the \( \mathcal{P} \mathcal{A} \). This will be useful in subsequent proofs where we need the fact that certain sets of \( \mathcal{P} \mathcal{A} \) paths are not empty.

**Lemma 2.** For every transition \( q \xrightarrow{\gamma} q' \) in \( \mathcal{A}_{\text{pre}} \) there exists a sequence \( \sigma \in \Delta_{\text{pre}} \) such that \( \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \).

**Proof.** The proof is available in App. C.1

First we will establish the essential result for a single transition of the created automaton.

**Lemma 3.** Consider a weighted pushdown system \( \mathcal{W} = (\mathcal{P}, \mathcal{F}, f) \) where \( \mathcal{F} \) is affine and an automaton \( \mathcal{A}_{\text{pre}}^\mathcal{C} \) created by the saturation procedure. Moreover, let \( \lambda \) be the least solution to the set of constraints \( \mathcal{C} \). For every transition \( q \xrightarrow{\gamma} q' \) in this automaton we have that

\[
\lambda(q \xrightarrow{\gamma} q') \subseteq \bigoplus \{ v(\sigma) | \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle, \sigma \in \Delta_{\text{pre}}^* \}
\]
Proof. The proof is available in App. C.2.

This is also the place that we have used the fact that the solution is equal to the least upper bound of the ascending Kleene sequence.

And now we can generalize the above to the case of a path in the automaton.

**Lemma 4.** Consider a weighted pushdown system \( \mathcal{W} = (P, F, f) \) where \( F \) is affine and a \( \mathcal{A}_{pre}^* \) automaton created by the saturation procedure. Moreover, let \( \lambda \) be the least solution to the set of constraints \( C \). For every path \( \rho = q \xrightarrow{*} q' \) in this automaton we have that

\[
\lambda^*(q \xrightarrow{\rho} \cdot \xrightarrow{*} q') \subseteq \bigoplus \{v(\sigma) \mid (q, s) \xrightarrow{\sigma} (q', \epsilon), \sigma \in \Delta^*_{pre}\}
\]

**Proof.** The proof is available in App. C.3.

And finally, using both the Thm. 1 and the above Lemma, we can formulate the main result.

**Theorem 3.** Consider an automaton \( \mathcal{A}_{pre}^* \) constructed by the saturation procedure and let \( \lambda \) be the least solution to the set of its constraints \( C \). If the flow algebra is affine then for every path \( \rho = q \xrightarrow{\cdot} q_f \) where \( q_f \in F \) we have that

\[
\lambda^*(p \xrightarrow{\rho} \cdot \xrightarrow{\cdot} q_f) = \bigoplus \{v(\sigma) \mid (p, s) \xrightarrow{\sigma} (q_f, \epsilon), \sigma \in \Delta^*_{pre}\}
\]

**Proof.** The proof is available in App. C.4.

### 6.2 Post *

Consider a pushdown system \( P \) with pushdown rules \( \Delta \) and a regular set of configurations \( C \) with an automaton \( A \) that accepts \( C \). First let us define a small modification of the pushdown rules \( \Delta \). Each rule \( r \) of the form \( \langle p, \gamma \rangle \xrightarrow{\cdot} \langle q' \rangle \) can be “split” into two rules \( r_1 \) and \( r_2 \):

\[
\begin{align*}
r_1 & = \langle p, \gamma \rangle \xrightarrow{\cdot} \langle q'_p, \gamma_1, \gamma_2 \rangle \\
r_2 & = \langle q'_p, \gamma_1, \epsilon \rangle \xrightarrow{\cdot} \langle p', \gamma_1 \rangle
\end{align*}
\]

with weights \( f(r_1) = f(r) \) and \( f(r_2) = 1 \). Note that the second rule is not really a pushdown rule as defined earlier. Fortunately, all we need to do, is to redefine \( \Rightarrow \) in the following way:

- if \( r = \langle q, \gamma \rangle \xrightarrow{\cdot} \langle q', w \rangle \) then \( \forall w' \in \Gamma^* : \langle q, \gamma s \rangle \Rightarrow \langle q', ws \rangle \)
- if \( r = \langle q, \epsilon \rangle \xrightarrow{\cdot} \langle q', \gamma \rangle \) then \( \forall w' \in \Gamma^* : \langle q, s \rangle \Rightarrow \langle q', \gamma s \rangle \)

This does not change the pushdown system in any way. Since we add a fresh state, there is no danger of changing any paths except for the ones we intend to. Moreover, the weight remains the same (1 is neutral element for \( \otimes \), so \( f(r_1) \otimes f(r_2) = f(r) \)).

Therefore, in place of \( \Delta_{post} \) we will use \( \Delta_{post-2} \), which is defined as follows:
For every $q' \xrightarrow{\gamma_c} q$ in $\mathcal{A}$ we have a rule $r = \langle q', \epsilon \rangle \rightarrow \langle q, \gamma \rangle$ in $\Delta_{\text{post-2}}$ such that $f(r) = 1$.

For every $r \in \Delta$ of the form $r = \langle p, \gamma \rangle \rightarrow \langle p', \gamma_1 \gamma_2 \rangle$ there are $r_1$ and $r_2$ in $\Delta_{\text{post-2}}$ as described above.

All other rules of $\Delta$ are included in $\Delta_{\text{post-2}}$ without any modification.

So compared to $\Delta_{\text{post}}$ the only difference is that we split the push-rules into two separate rules. At the same time we do not change the behavior of the system in any way.

This allows us to prove the following lemma, which is used in subsequent proofs.

**Lemma 5.** For every transition $q' \xrightarrow{\gamma_c} q$ ($\gamma_c \in \Gamma \cup \{\epsilon\}$) in $\mathcal{A}_{\text{post}}$ there exists a sequence $\sigma$ of pushdown rules in $\Delta_{\text{post-2}}$ such that $\langle q', \epsilon \rangle \sigma \Rightarrow^* \langle q, \gamma_c \epsilon \rangle$.

**Proof.** The proof is available in App. C.5.

Again, as in the case of $\text{Pre}^*$ we first establish the result for a single transition in the automaton.

**Lemma 6.** Consider a weighted pushdown system $W = (\mathcal{P}, \mathcal{F}, f)$ where $\mathcal{F}$ is affine and an automaton $\mathcal{A}_{\text{post}}^*$ created by the saturation procedure. Moreover, let $\lambda$ be the least solution to the set of constraints $\mathcal{C}$. For every transition $q' \xrightarrow{\gamma_c} q$ ($\gamma_c \in \Gamma \cup \{\epsilon\}$) in this automaton we have that

$$
\lambda(q' \xrightarrow{\gamma_c} q) \sqsubseteq \bigoplus \left\{ v(\sigma) \mid \langle q', \epsilon \rangle \sigma \Rightarrow^* \langle q, \gamma_c \epsilon \rangle, \sigma \in \Delta_{\text{post-2}}^* \right\}
$$

**Proof.** The proof is available in App. C.6.

And again, as in the case of $\text{Pre}^*$, this is the place that we have used the fact that the solution is equal to the least upper bound of the ascending Kleene sequence.

Now we can generalize the obtained result for the paths in the automaton.

**Lemma 7.** Consider a weighted pushdown system $W = (\mathcal{P}, \mathcal{F}, f)$ where $\mathcal{F}$ is affine and a $\mathcal{A}_{\text{post}}^*$ automaton created by the saturation procedure. Moreover, let $\lambda$ be the least solution to the set of constraints $\mathcal{C}$. For every path $\rho = q' \xrightarrow{\gamma} q$ ($s \in \Gamma^*$) in this automaton we have that

$$
\lambda_R(q' \xrightarrow{\gamma} q) \sqsubseteq \bigoplus \left\{ v(\sigma) \mid \langle q', \epsilon \rangle \sigma \Rightarrow^* \langle q, \gamma \epsilon \rangle, \sigma \in \Delta_{\text{post-2}}^* \right\}
$$

**Proof.** The proof is available in App. C.7.

And finally using both the soundness Thm. 2 and the above, we can establish the main result.

**Theorem 4.** Consider an automaton $\mathcal{A}_{\text{post}}^*$ constructed by the saturation procedure and let $\lambda$ be the least solution to the set of its constraints $\mathcal{C}$. If the flow algebra is affine then for every path $\rho = qf \xrightarrow{\gamma} p$ where $qf \in \mathcal{F}$ we have that

$$
\lambda_R(qf \xrightarrow{\gamma} p) \sqsubseteq \bigoplus \left\{ v(\sigma) \mid \langle qf, \epsilon \rangle \sigma \Rightarrow^* \langle p, s \rangle, \sigma \in \Delta_{\text{post-2}}^* \right\}
$$

**Proof.** The proof is available in App. C.8.
7 Discussion and examples

In this section we will discuss the relation of our development to the area of interprocedural analysis, as well as the challenges and advantages of the approach. Furthermore, we will present an example of analyses that thanks to our algorithms are directly expressible in our framework, which was not possible before.

7.1 Monotone frameworks and pushdown systems

To put our approach into perspective, it is useful to emphasize that it is a generalization of the functional approach to interprocedural analysis by Sharir and Pnueli [2]. In both of these approaches the underlying idea is to compute the summarizations of actions and by composing them obtain the summarizations of procedures. The generality of weighted pushdown systems stems from the fact that they make it possible to obtain the analysis information for specific calling contexts or even families of calling contexts. In other words one can perform queries of weighted $A_{\text{pre}}$ and $A_{\text{post}}$ automata, to get the summarization of all the paths between the initial set of configurations and a given stack or even a regular set of stacks. Applying the summarization to some initial analysis information, we can obtain the desired result. This is possible due to the way the algorithms for pushdown systems construct the $A_{\text{pre}}$ and $A_{\text{post}}$ automata and generate the constraints whose solution provides us with the weights of all the transition in those automata.

One of the most significant advantages of using summarizations is the fact that each procedure can be analyzed only once and the result can be used at all the call sites. In other words the summarization of a procedure is independent of the calling context, which is the key to reusing the information. However, there is also a downside to this approach, namely the fact that the analysis has to work on the dataflow transformers and not directly on some dataflow facts (i.e., we compute what and how the dataflow facts can change). This often makes it more difficult to formulate analyses whose results we can actually compute. The main challenge is that if some domain $D$ satisfies, e.g., the ascending chain condition, when lifted to transformers $D \to D$ it might not satisfy this condition anymore. Fortunately we can still express many analyses. Even for cases like constant propagation where $D$ is usually a mapping from variables to integers/reals, it is possible to define computable variants, i.e., copy- and linear-constant propagation [7,8]. Obviously whenever $D$ is finite then $D \to D$ will be finite as well. This might seem a bit restrictive, but there are many analyses that satisfy the requirement. In fact the interprocedural analysis based on graph reachability [19] works on distributive functions $\mathcal{P}(D) \to \mathcal{P}(D)$ where $D$ is required to be some finite set.

7.2 Example

As an example let us consider the family of forward, may analyses that are instances of bitvector framework. They are generally defined in the following way:
– The lattice $L$ is equal to $\mathcal{P}(D)$ for some finite $D$.
– The least upper bound operator is $\bigcup$.
– The transfer functions are monotone functions of the shape

$$f_i(l) = (l \setminus k_i) \cup g_i$$

where $k_i, g_i \in \mathcal{P}(D)$ correspond to the elements of $D$ that are “killed” and “generated” at some program point $i$. This is also the source of a popular name for similar analyses — “kill/gen” analyses.
– The least element $\bot = \emptyset$.

In order to use such an analyses with weighted pushdown systems we will construct a flow algebra $(\mathcal{F}, \oplus, \otimes, \bar{0}, \bar{1})$ that expresses the transformers $\mathcal{P}(D) \to \mathcal{P}(D)$. Since we are dealing with “kill/gen” analysis, this is actually quite easy — we express a function $f_i(l) = (l \setminus k_i) \cup g_i$ by a pair $(k_i, g_i)$. Therefore, we have:

– $\mathcal{F} = \mathcal{P}(D) \times \mathcal{P}(D)$
– The $\oplus$ operator is defined as $f_1 \oplus f_2 = (k_1, g_1) \oplus (k_2, g_2) = (k_1 \cap k_2, g_1 \cup g_2)$
– The $\otimes$ operator is defined as $f_1 \otimes f_2 = (k_1, g_1) \otimes (k_2, g_2) = (k_1 \cup k_2, (g_1 \setminus k_2) \cup g_2)$
– $\bar{0} = (D, \emptyset)$
– $\bar{1} = (\emptyset, \emptyset)$

It should be easy to see that $\oplus$ is idempotent and commutative. Therefore, the semiring is naturally ordered with $f_1 \sqsubseteq f_2 \iff f_1 \oplus f_2 = f_2$. Furthermore, $\bar{0}$ is a neutral element for $\oplus$ and $\bar{1}$ is neutral for $\otimes$.

However, the interesting part is that $\bar{0}$ is not an annihilator for $\otimes$. Consider the following:

$$(D, \emptyset) \otimes (k, g) = (D \cup k, (\emptyset \setminus k) \cup g) = (D, g)$$

which clearly is not equal to $\bar{0}$ (unless $g = \emptyset$). Interestingly the annihilation works from the right:

$$(k, g) \otimes (D, \emptyset) = (k \cup D, (g \setminus D) \cup \emptyset) = (D, \emptyset) = \bar{0}$$

This makes perfect sense if we consider for a moment the classical transfer functions of such analyses. If we extend the ordering of $\mathcal{P}(D)$ pointwise to the monotone functions $\mathcal{P}(D) \to \mathcal{P}(D)$, the least element will be a function that always returns $\emptyset$, i.e., $f_\bot = \lambda l. \emptyset$. Clearly we have that

$$\forall f : f_\bot \circ f = f_\bot$$
but in the second case

$$\neg(\forall f : f \circ f_\perp = f_\perp)$$

Therefore, such analyses do not directly fit in the original framework of WPDS or CPDS. Yet they do in our modified one that relaxes the requirement of annihilation.

8 Conclusions

Weighted/communicating pushdown systems have been used in many contexts and are a popular approach to interprocedural analysis. However, their requirements with respect to the abstract domain were quite restrictive and did not admit some of the classical analyses directly. In this paper we have shown that some of the restrictions are not necessary. We have achieved that by reformulating the algorithms for backward and forward reachability. Furthermore, we have proved that they are sound — they always provide a safe over-approximation of the join over all valid paths solution. Provided some additional properties of the abstract domain, we have also shown that those solutions coincide, i.e., the algorithms are complete.

We believe that our results strengthen the connection between the monotone frameworks and the pushdown systems by making it possible to directly express more analyses based on monotone frameworks in the setting of pushdown systems. Moreover, the development does provide some additional flexibility when both designing and implementing analyses using pushdown systems. For instance, the annihilation property might be useful for certain analyses, but now this is the choice of the designer of the analysis and not a hard requirement from the framework. Last, but not least, we believe that the paper improves the understanding of using weighted pushdown systems for interprocedural program analysis.

References

1. Kam, J.B., Ullman, J.D.: Monotone data flow analysis frameworks. Acta Inf. 7 (1977) 305–317
2. Sharir, M., Pnueli, A.: Two approaches to interprocedural data flow analysis. In Muchnick, S.S., Jones, N.D., eds.: Program Flow Analysis: Theory and Applications. Prentice Hall Professional Technical Reference, Englewood Cliffs, NJ (1981) 189–234
3. Cousot, P., Cousot, R.: Static determination of dynamic properties of recursive procedures. In Neuhold, E., ed.: IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA, North-Holland (1977) 237–277
4. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In Mazurkiewicz, A.W., Winkowski, J., eds.: CONCUR. Volume 1243 of Lecture Notes in Computer Science., Springer (1997) 135–150
5. Esparza, J., Hanusel, D., Rossmanith, P., Schwon, S.: Efficient algorithms for model checking pushdown systems. In Emerson, E.A., Sistla, A.P., eds.: CAV. Volume 1855 of Lecture Notes in Computer Science., Springer (2000) 232–247
6. Schwoon, S.: Model-Checking Pushdown Systems. PhD thesis, Technical University Munich (2002)
7. Reps, T.W., Schwoon, S., Jha, S.: Weighted pushdown systems and their application to interprocedural dataflow analysis. In Cousot, R., ed.: SAS. Volume 2694 of Lecture Notes in Computer Science., Springer (2003) 189–213
8. Reps, T.W., Schwoon, S., Jha, S., Melski, D.: Weighted pushdown systems and their application to interprocedural dataflow analysis. Sci. Comput. Program. 58(1-2) (2005) 206–263
9. Reps, T.W., Lal, A., Kidd, N.: Program analysis using weighted pushdown systems. In Arvind, V., Prasad, S., eds.: FSTTCS. Volume 4855 of Lecture Notes in Computer Science., Springer (2007) 23–51
10. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. In Aiken, A., Morrisett, G., eds.: POPL, ACM (2003) 62–73
11. Bouajjani, A., Esparza, J., Touili, T.: A generic approach to the static analysis of concurrent programs with procedures. Int. J. Found. Comput. Sci. 14(4) (2003) 551–582
12. Suwimonteerabuth, D., Schwoon, S., Esparza, J.: jmoped: A java bytecode checker based on moped. In Halbwachs, N., Zuck, L.D., eds.: TACAS. Volume 3440 of Lecture Notes in Computer Science., Springer (2005) 541–545
13. Balakrishnan, G., Reps, T.W., Kidd, N., Lal, A., Lim, J., Melski, D., Gruian, R., Yong, S.H., Chen, C.H., Teitelbaum, T.: Model checking x86 executables with codesurfer/x86 and wpds++. In Etessami, K., Rajamani, S.K., eds.: CAV. Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 158–163
14. Filipiuk, P., Terepeta, M., Nielson, H.R., Nielson, F.: Galois connections for flow algebras. In Bruni, R., Dingel, J., eds.: FMOODS/FORTE. Volume 6722 of Lecture Notes in Computer Science., Springer (2011) 138–152
15. Nielson, F., Nielson, H.R., Hankin, C.: Principles of program analysis (2. corr. print). Springer (2005)
16. Droste, M., Kuich, W.: Semirings and formal power series. In Droste, M., Kuich, W., Vogler, H., eds.: Handbook of Weighted Automata. Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg (2009) 3–28
17. Esparza, J., Kiefer, S., Luttenberger, M.: Newtonian program analysis. J. ACM 57(6) (2010) 33
18. Esparza, J., Luttenberger, M.: Solving fixed-point equations by derivation tree analysis. In Corradini, A., Klin, B., Cirstea, C., eds.: CALCO. Volume 6859 of Lecture Notes in Computer Science., Springer (2011) 19–35
19. Reps, T.W., Horwitz, S., Sagiv, S.: Precise interprocedural dataflow analysis via graph reachability. In Cytron, R.K., Lee, P., eds.: POPL, ACM Press (1995) 49–61
A Soundness proofs

A.1 Proof of Thm. 1

Consider an automaton $A$ and its corresponding $A_{pre}^c$ generated by the saturation procedure. Let us assume that we have the least solution $\lambda$ to the set of constraints $C$. Then for each pair $(p, s)$ such that $\langle p, s \rangle \xrightarrow{\sigma}^* (q_f, \epsilon)$ (where $\sigma \in \Delta_{pre}^*$ and $q_f \in F$), we have $v(\sigma) \sqsubseteq \lambda^*(\rho)$ where $\rho = p \xrightarrow{\sigma}^* q_f$ is in $A_{pre}^c$.

**Proof.** Note that we do not need to prove the existence of the paths in the $A_{pre}^c$ — it is a previously known result [6,7]. We can use it because our algorithm differs only in the constraint generation, and not in the way new transitions are added. Moreover, as explained above, the additional rules in $\Delta_{pre}$ do not change that result.

The proof will proceed by induction on $|\sigma|$ (note that since $P$ and $F$ are disjoint, it is not possible to have $|\sigma| = 0$).

$|\sigma| = 1$ We know that the path in the pushdown system is $\langle p, \gamma \rangle \xrightarrow{r} \langle q_f, \epsilon \rangle$. But this means that $r \in \Delta_{pre} \setminus \Delta$. Existence of $p \xrightarrow{r} q_f$ follows directly from the definition of $\Delta_{pre}$. We also have that $f(r) = \overline{1}$. Finally, according to the saturation procedure there exists a constraint: $\overline{1} \sqsubseteq l(p \xrightarrow{r} q_f)$. Therefore, clearly $v([r]) \sqsubseteq \lambda(p \xrightarrow{r} q_f)$.

$|\sigma| > 1$ In this case we know that the path in the pushdown system is

$$\langle p, \gamma s_0 \rangle \xrightarrow{r\,+\,s} \langle q' \,+\, w s_0 \rangle \xrightarrow{\sigma'}^* (q_f, \epsilon)$$

for some $q', \gamma$, and $w$. Moreover, $r = \langle p, \gamma \rangle \xrightarrow{\sigma} \langle q', w \rangle$ where $s = \gamma s_0$.

If $q' \notin P$ then $r \in \Delta_{pre} \setminus \Delta$ and $f(r) = \overline{1}$ ($r$ is one of the added rules to $\Delta_{pre}$). Furthermore, all the rules of $\sigma'$ must also be in $\Delta_{pre} \setminus \Delta$ and thus there must be a path $\rho = p \xrightarrow{\sigma'}^* q_f$ in $A_{pre}^c$ (since it must also be in $A$). Therefore, $v(\sigma') = \overline{1}$ and for each transition $t$ on the path $\rho$ we have a constraint of the form $\overline{1} \sqsubseteq l(\sigma')$, thus by monotonicity we have $v(\sigma') \sqsubseteq \lambda^*(\rho)$. Otherwise $q' \in P$ and $r \in \Delta$, so we can use the induction hypothesis to get that

$$v(\sigma') \sqsubseteq \lambda^*(q' \xrightarrow{w s_0} q_f)$$

where

$$\rho' = \rho'\xrightarrow{q' w s_0} q'' \xrightarrow{s_0} q_f$$

Now the saturation procedure must have added the transition $p \xrightarrow{r} q''$. So we have a path $\rho = p \xrightarrow{r} q'' \xrightarrow{s_0} q_f$ along with a constraint:
1. if \( w = \epsilon \) (so \( q' = q'' \)) the added constraint is
\[
f(r) \sqsubseteq l(p \xrightarrow{\gamma} q')
\]

2. if \( w = \gamma' \) the added constraint is
\[
f(r) \otimes l(q' \xrightarrow{\gamma'} q'') \sqsubseteq l(p \xrightarrow{\gamma''} q'')
\]

3. if \( w = \gamma_1' \gamma_2' \) the added constraint is
\[
f(r) \otimes l(q' \xrightarrow{\gamma_1'} q_x) \otimes l(q_x \xrightarrow{\gamma_2'} q'') \sqsubseteq l(p \xrightarrow{\gamma} q'')
\]

For case 1 we have:
\[
v(\sigma) = f(r) \otimes v(\sigma') \sqsubseteq f(r) \otimes \lambda^*(q'' \xrightarrow{\sigma_0} q_f) \sqsubseteq \lambda(p \xrightarrow{\gamma} q') \otimes \lambda^*(q'' \xrightarrow{\sigma_0} q_f) = \lambda^*(p \xrightarrow{\rho} q_f)
\]

And for both 2 and 3
\[
v(\sigma) = f(r) \otimes v(\sigma') \sqsubseteq f(r) \otimes \lambda^*(q' \xrightarrow{w} q'') \otimes \lambda^*(q'' \xrightarrow{\rho} q_f) \sqsubseteq \lambda^*(p \xrightarrow{\gamma} q') \otimes \lambda^*(q'' \xrightarrow{\rho} q_f) = \lambda^*(p \xrightarrow{\rho} q_f)
\]

Thus in all possible cases we have that:
\[
v(\sigma) \sqsubseteq \lambda^*(p \xrightarrow{\rho} q_f)
\]

A.2 Proof of Thm. 2

Consider an automaton \( A \) and its corresponding \( A_{post}^C \) generated by the saturation procedure. Let us assume that we have the least solution \( \lambda \) to the set of constraints \( \mathcal{C} \). Then for each pair \( (p, s) \) such that \( (q_f, c) \xrightarrow{\sigma}^*(p, s) \) (where \( \sigma \in A_{post}^C \) and \( q_f \in F \)), we have \( v(\sigma) \sqsubseteq \lambda^*_t(\rho) \) where \( \rho = q_f \xrightarrow{\sigma} p \) is in \( A_{post}^C \).
Proof. Note that, as in the case of \( \text{Pre}^* \), we do not need to prove the existence of the paths in the \( A_{\text{pre}}^* \) — it is a previously known result [67]. Again this is due to the fact that our algorithm differs only in the constraint generation, and not in the way new transitions are added. Moreover, as explained above, the additional rules in \( \Delta_{\text{post}} \) do not change that result.

The proof will proceed by induction on \(|\sigma|\) (note that since \( P \) and \( F \) are disjoint, it is not possible to have \(|\sigma| = 0\)).

\(|\sigma| = 1\) So \( s = \gamma \) and we have \( \langle q_f, \epsilon \rangle \xrightarrow{r} \langle p, s \rangle \). We know that \( r \in \Delta_{\text{post}} \setminus \Delta \), and so from the definition of \( \Delta_{\text{post}} \) we have that there is transition \( q_f \xleftarrow{\epsilon} p \) and \( v([r]) = 1 \). Moreover, from the saturation procedure we have a constraint \( 1 \subseteq h(q_f \xleftarrow{\epsilon} p) \). Therefore, \( v([r]) \subseteq \lambda(q_f \xleftarrow{\epsilon} p) \).

\(|\sigma| > 1\) So we have

\[ \langle q_f, \epsilon \rangle \xrightarrow{\sigma'} \langle q', s' \rangle \xrightarrow{r} \langle p, s \rangle \]

where \( \sigma = \sigma' r \).

If \( q' \notin P \) then \( r \in \Delta_{\text{post}} \setminus \Delta \) and it must be of the form \( r = \langle q', \epsilon \rangle \xrightarrow{} \langle p, \gamma \rangle \) where \( s = \gamma s' \) (\( r \) is one of the additional rules to the \( \Delta_{\text{post}} \)). But that means that all the remaining rules in \( \sigma' \) must also be one of those additional rules (\( \Delta_{\text{post}} \setminus \Delta \)). Thus the weight of every transition \( t \) on the path \( q_f \xleftarrow{\epsilon} p \) is \( \lambda(t) = 1 \) (its existence follows directly from the definition of \( \Delta_{\text{post}} \)). Moreover, all of them must have a corresponding constraint of the form \( 1 \subseteq h(q_f \xleftarrow{\epsilon} p) \). Therefore, by monotonicity we have \( 1 \subseteq \lambda_R(q_f \xleftarrow{\epsilon} p) \) and so \( v(\sigma) \subseteq \lambda_R(q_f \xleftarrow{\epsilon} p) \).

Otherwise \( q' \in P \) and \( r \in \Delta \), \( r = \langle q', \gamma' \rangle \xrightarrow{} \langle p, w \rangle \) and \( s = ws_0, s' = \gamma's_0 \).

Since \(|\sigma'| < |\sigma|\) we can use the induction hypothesis to get that

\[ v(\sigma') \subseteq \lambda_R(q_f \xleftarrow{\epsilon} q') \]

where

\[ \rho' = q_f \xleftarrow{\epsilon'} q' = q_f \xleftarrow{\epsilon' \rho_1} q'' \xleftarrow{\rho_2} q' \]

for some \( q'' \). And so we have three possibilities, depending on \( w \):

1. if \( w = \epsilon \), the transition \( q'' \xleftarrow{\epsilon} p \) along with the following constraint

\[ h(\epsilon)(q'' \xleftarrow{\epsilon'} q') \otimes f(r) \subseteq h(q'' \xleftarrow{\epsilon} p) \]

Therefore, the solution will have to satisfy:

\[ \lambda_R(q'' \xleftarrow{\epsilon} q') \otimes f(r) \subseteq \lambda(q'' \xleftarrow{\epsilon} p) \]
and so
\[ v(\sigma) = v(\sigma') \otimes f(r) \]
\[ \sqsubseteq \lambda_R(q_f * \overset{\gamma'}{\rho'} q') \otimes f(r) \]
\[ = \lambda_R(q_f * \overset{\epsilon_{\rho_2}}{\rho_1} q'') \otimes \lambda_R(q' * \overset{\gamma'}{\rho_1} q') \otimes f(r) \]
\[ \sqsubseteq \lambda_R(q_f * \overset{\epsilon_{\rho_2}}{\rho_1} q'') \otimes \lambda(q' * \overset{\gamma_0}{\rho_1} p) \]
\[ = \lambda_R(q_f * \overset{\epsilon_{\rho_2}}{\rho_1} p) \]

2. if \( w = \gamma_1 \), the transition \( q'' \overset{\rho_1}{\gamma_1} p \) along with the following constraint
\[ h'(q'' \overset{\gamma'}{\rho_1} q') \otimes f(r) \sqsubseteq h(q'' \overset{\gamma_1}{\rho_1} p) \]

Therefore, the solution will have to satisfy:
\[ \lambda_R(q'' * \overset{\gamma'}{\rho_1} q') \otimes f(r) \sqsubseteq \lambda(q'' \overset{\gamma_1}{\rho_1} p) \]

and so
\[ v(\sigma) = v(\sigma') \otimes f(r) \]
\[ \sqsubseteq \lambda_R(q_f * \overset{\gamma'}{\rho'} q') \otimes f(r) \]
\[ = \lambda_R(q_f * \overset{\epsilon_{\rho_2}}{\rho_1} q'') \otimes \lambda_R(q' * \overset{\gamma'}{\rho_1} q') \otimes f(r) \]
\[ \sqsubseteq \lambda_R(q_f * \overset{\epsilon_{\rho_2}}{\rho_1} q'') \otimes \lambda(q' * \overset{\gamma_0}{\rho_1} p) \]
\[ = \lambda_R(q_f * \overset{\epsilon_{\rho_2}}{\rho_1} p) \]

3. if \( w = \gamma_1 \gamma_2 \), the transitions \( q_p, \gamma_1 \overset{\gamma_1}{\rho_1} q' \) and \( q'' \overset{\gamma_2}{\rho_1} q_p, \gamma_1 \) along with the following constraints
\[ \bar{1} \sqsubseteq h(q_p, \gamma_1 \overset{\gamma_1}{\rho_1} q) \]

and
\[ h'(q'' \overset{\gamma'}{\rho_1} q') \otimes f(r) \sqsubseteq h(q'' \overset{\gamma_2}{\rho_1} q_p, \gamma_1) \]

Therefore, the solution will have to satisfy:
\[ \bar{1} \sqsubseteq \lambda(q_p, \gamma_1 \overset{\gamma_1}{\rho_1} q') \]
\[ \lambda_R(q'' * \overset{\gamma'}{\rho_1} q') \otimes f(r) \sqsubseteq \lambda(q'' \overset{\gamma_2}{\rho_1} q_p, \gamma_1) \]
and so

\[ v(\sigma) = v(\sigma') \otimes f(r) \]

\[ \sqsubseteq \lambda^*_R(q_f \ast \frac{\tau}{\rho'} q') \otimes f(r) \]

\[ = \lambda^*_R(q_f \ast \frac{\tau}{\rho''_2} q''_1 \otimes \lambda^*_R(q''_1 \ast \frac{\tau}{\rho_1} q') \otimes f(r) \]

\[ \sqsubseteq \lambda^*_R(q_f \ast \frac{\tau}{\rho''_2} q''_1 \otimes \lambda(q''_1 \ast \frac{\tau}{\rho_1} p) \]

\[ = \lambda^*_R(q_f \ast \frac{\tau}{\rho''_2} q''_1 \otimes \lambda(q''_1 \ast \frac{\tau}{\rho_1} q_p, q_1) \otimes \lambda(q_p, q_1 \ast \frac{\tau}{\rho} q_1) \]

\[ = \lambda^*_R(q_f \ast \frac{\tau}{\rho} p) \]

\[ \square \]

## B Continuity proof (Lem. 1)

The function \( F \), defined as:

\[ F : (\delta \rightarrow D) \rightarrow (\delta \rightarrow D) \]

\[ F(m)t = \bigoplus_{c \in C_t} rh_{s_m}(c) \]

is continuous, i.e., for any non-empty chain \( Y \):

\[ F(\bigsqcup Y) = \bigsqcup_{m \in Y} F(m) \]

**Proof.** Since we are assuming that \( D \) is a complete lattice and \( m \) is a total function, then \( \delta \rightarrow D \) defines a complete lattice as well. Furthermore, we have that for any \( Y \subseteq \delta \rightarrow D \)

\[ (\bigsqcup Y)t = \bigoplus_{m \in Y} m(t) \quad (1) \]

Therefore, we have:
\[ F(\bigsqcup Y)t \]
\[ = \text{[ definition of } F \text{]} \]
\[ \bigoplus \{ \text{lhs} \bigsqcup_Y (c) \mid c \in C_t \} \]
\[ = \text{[ equation (1) ]} \]
\[ \bigoplus \{ \text{lhs}_{\lambda_t'} \bigsqcup \bigoplus_{m \in Y} m(t')(c) \mid c \in C_t \} \]
\[ = \text{[ D is affine, } Y \text{ is not empty and the constraints are finite ]} \]
\[ \bigoplus \{ \bigoplus_{m \in Y} \text{lhs}_m(c) \mid c \in C_t \} \]
\[ = \text{[ definition of } F \text{]} \]
\[ \bigoplus_{m \in Y} F(m)t \]
\[ = \text{[ equation (1) ]} \]
\[ (\bigsqcup_{m \in Y} F(m))t \]

C  Completeness proofs

C.1  Proof of Lem. [2]

For every transition \( q \xrightarrow{\gamma} q' \) in \( A_{\text{pre}} \), there exists a sequence \( \sigma \in \Delta_{\text{pre}} \) such that \( \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \).

Proof. Proof will proceed by induction on \( A_i \), where \( A_i \) corresponds to the initial automaton after \( i \) steps of the saturation procedure.

- \( i = 0 \) Follows from the definition of \( \Delta_{\text{pre}} \).
- \( i > 0 \) We assume the property holds for \( A_i \) and prove it for \( A_{i+1} \). Consider that the saturation procedure adds a transition \( p_s \xrightarrow{\gamma'} q_d \) (note that the saturation procedure works on \( \Delta \)) because of:
  - a pushdown rule \( r = \langle p_s, \gamma \rangle \rightarrow \langle q_d, \epsilon \rangle \). The result is immediate from the rule.
  - a pushdown rule \( r = \langle p_s, \gamma \rangle \rightarrow \langle p', \gamma' \rangle \) and a transition \( p' \xrightarrow{\gamma'} q_d \) in \( A_i \). We use the induction hypothesis on \( p' \xrightarrow{\gamma'} q_d \) and get that there exists \( \sigma \) such that \( \langle p', \gamma' \rangle \xrightarrow{\sigma}^* \langle q_d, \epsilon \rangle \). But then we also have that

\[ \langle p_s, \gamma \rangle \xrightarrow{r} \langle p', \gamma' \rangle \xrightarrow{\sigma}^* \langle q_d, \epsilon \rangle \]
– a pushdown rule \( r = (p_s, \gamma) \rightarrow (p', \gamma' \gamma'') \) and a path \( p' \xrightarrow{\gamma'} q'' \xrightarrow{\gamma''} q_d \) in \( A_i \). We use the induction hypothesis on \( p' \xrightarrow{\gamma'} q'' \) and \( q'' \xrightarrow{\gamma''} q_d \) to get that there exists \( \sigma' \) and \( \sigma'' \) such that \( (p', \gamma') \xrightarrow{\sigma'} (q'', \epsilon) \) and \( (q'', \gamma') \xrightarrow{\sigma''}^* (q_d, \epsilon) \). And again we have that:

\[
\langle p_s, \gamma \rangle \xrightarrow{r} \langle p', \gamma' \gamma'' \rangle \xrightarrow{\sigma' \sigma''}^* (q_d, \epsilon)
\]

\( \square \)

C.2 Proof of Lem. 3

Consider a weighted pushdown system \( \mathcal{W} = (\mathcal{P}, \mathcal{F}, f) \) where \( \mathcal{F} \) is affine and an automaton \( \mathcal{A}_{pre}^C \) created by the saturation procedure. For every transition \( q \xrightarrow{\gamma} q' \) in this automaton we have that

\[
\lambda(q \xrightarrow{\gamma} q') \subseteq \bigoplus \{ v(\sigma) | \langle q, \gamma \rangle \xrightarrow{\sigma} (q', \epsilon), \sigma \in \Delta_{pre}^* \}
\]

**Proof.** Let us also denote by \( \mathcal{A}_i^C \) the automaton \( \mathcal{A} \) after \( i \) steps of the saturation procedure. Also let us denote the least solution for \( \mathcal{A}_i^C \) by \( \lambda_i \). We will prove by induction on \( i \) that for every transition \( q \xrightarrow{\gamma} q' \) in \( \mathcal{A}_i^C \) we have that

\[
\lambda_i(q \xrightarrow{\gamma} q') \subseteq \bigoplus \{ v(\sigma) | \langle q, \gamma \rangle \xrightarrow{\sigma} (q', \epsilon), \sigma \in \Delta_{pre}^* \}
\]

\( i = 0 \) \( \mathcal{A}_0^C \) is just the initial automaton \( \mathcal{A} \) with the set \( C \) containing one constraint for every transition of \( \mathcal{A} \). The property clearly holds.

\( i > 0 \) We assume the property holds for \( \mathcal{A}_i^C \) and prove it for \( \mathcal{A}_{i+1}^C \), i.e., prove that adding a constraint (and maybe a transition as well) preserves the property of interest.

Let \( t \) be the transition that the added constraint refers to. Observe that if \( t \) was already in the automaton \( \mathcal{A}_i^C \), then it is possible that \( \lambda(t) \) might be on the left-hand side of some other constraint. Therefore, the least solution for the new set of constraints might be different for other transitions as well; in other words the value/information from the new constraint might have to be propagated throughout other constraints to get \( \lambda_{i+1} \). Now let \( \lambda_i^j \) denote the solution after \( j \) steps of fixed point computation with the new constraint, starting with

\[
\lambda_i^0(t) = \begin{cases} 0 & \text{if } t \text{ was added} \\ \lambda_i(t) & \text{otherwise (} t \text{ was in } \mathcal{A}_i^C \) \end{cases}
\]

Using induction on \( j \) we will prove that the property of interest is maintained by the computation.

Note that we can use here Kleene iteration due to Lemma 1.

\( j = 0 \) Immediate from outer induction hypothesis.
Consider each form of the possible constraints:

- \( f(r) \subseteq \lambda(q \xrightarrow{\gamma} q') \) where \( r = \langle q, \gamma \rangle \mapsto \langle q', \epsilon \rangle \). We know that

\[
\lambda_i^{j+1}(q \xrightarrow{\gamma} q') = \lambda_i^j(q \xrightarrow{\gamma} q') \oplus f(r)
\]

Moreover, from the rule \( r \) it immediately follows that

\[
f(r) \subseteq \bigoplus \{ v(\sigma) \mid \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

Using this and the induction hypothesis on \( \lambda_i^j(q \xrightarrow{\gamma} q') \)

\[
\lambda_i^{j+1}(q \xrightarrow{\gamma} q') \subseteq \bigoplus \{ v(\sigma) \mid \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

- \( f(r) \otimes \lambda(q'' \xrightarrow{\gamma''} q') \subseteq \lambda(q \xrightarrow{\gamma} q') \) where \( r = \langle q, \gamma \rangle \mapsto \langle q'', \gamma'' \rangle \) and \( q'' \xrightarrow{\gamma''} q' \). We have that

\[
\lambda_i^{j+1}(q \xrightarrow{\gamma} q') = \lambda_i^j(q \xrightarrow{\gamma} q') \oplus (f(r) \otimes \lambda_i^j(q'' \xrightarrow{\gamma''} q'))
\]

Now let us use the induction hypothesis:

\[
\lambda_i^j(q'' \xrightarrow{\gamma''} q') \subseteq \bigoplus \{ v(\sigma) \mid \langle q'', \gamma'' \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

Multiplying both sides by \( f(r) \) and using that \( \otimes \) is affine:

\[
f(r) \otimes \lambda_i^j(q'' \xrightarrow{\gamma''} q') \subseteq \bigoplus \{ f(r) \otimes v(\sigma) \mid \langle q'', \gamma'' \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

\[
\subseteq \bigoplus \{ v(\sigma) \mid \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

Therefore:

\[
\lambda_i^{j+1}(q \xrightarrow{\gamma} q') \subseteq \bigoplus \{ v(\sigma) \mid \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

- \( f(r) \otimes \lambda(q'' \xrightarrow{\gamma''} q'') \otimes \lambda(q' \xrightarrow{\gamma'} q') \subseteq \lambda(q \xrightarrow{\gamma} q') \) where \( r = \langle q, \gamma \rangle \mapsto \langle q'', \gamma'', \gamma' \rangle \) and \( q'' \xrightarrow{\gamma''} q' \). We have that

\[
\lambda_i^{j+1}(q \xrightarrow{\gamma} q') = \lambda_i^j(q \xrightarrow{\gamma} q')
\]

\[
\oplus (f(r) \otimes \lambda_i^j(q'' \xrightarrow{\gamma''} q'') \otimes \lambda_i^j(q' \xrightarrow{\gamma'} q'))
\]

We use the induction hypothesis twice to get

\[
\lambda_i^j(q'' \xrightarrow{\gamma''} q'') \subseteq \bigoplus \{ v(\sigma) \mid \langle q'', \gamma'' \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]

\[
\lambda_i^j(q' \xrightarrow{\gamma'} q') \subseteq \bigoplus \{ v(\sigma) \mid \langle q', \gamma' \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle \}
\]
From monotonicity and the fact that $\otimes$ is affine we get that:

$$f(r) \otimes \lambda^i_j(q'' \xrightarrow{\gamma''} q') \otimes \lambda^i_j(q' \xrightarrow{\gamma} q')$$

$$\sqsubseteq \bigoplus \{f(r) \otimes v(\sigma_1) \otimes v(\sigma_2) \mid \langle q'', \gamma'' \rangle \xrightarrow{\sigma_1}^* \langle q_1', \epsilon \rangle, \langle q_1', \gamma_1'' \rangle = \sigma_2 \Rightarrow \langle q', \epsilon \rangle\}$$

$$\sqsubseteq \bigoplus \{f(r) \otimes v(\sigma) \mid \langle q'', \gamma'' \gamma'' \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle\}$$

$$\sqsubseteq \bigoplus \{v(\sigma) \mid \langle q, \gamma \rangle \xrightarrow{\sigma}^* \langle q', \epsilon \rangle\}$$

Therefore

$$\lambda^{i+1}_i(q \xrightarrow{\gamma} q') \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q, \gamma \rangle = \sigma \Rightarrow^* \langle q', \epsilon \rangle, \sigma \in \Delta^{pre}\}$$

C.3 Proof of Lem. 4

Consider a weighted pushdown system $W = (P, F, f)$ where $F$ is affine and an $A_{pre}^C$ automaton created by the saturation procedure. For every path $\rho = q \xrightarrow{\gamma} q'$ in this automaton we have that

$$\lambda^*(q \xrightarrow{\gamma} q') \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q, \gamma \rangle = \sigma \Rightarrow^* \langle q', \epsilon \rangle, \sigma \in \Delta^{pre}\}$$

Proof. The proof will proceed with the induction on the number of transitions $|\rho|$ (we will use the inductive definition of $\lambda^*$).

$1 < |\rho|$. Again using the definition of $\lambda^*$ we have

$$\lambda^*(q \xrightarrow{\gamma} q') = \lambda(q \xrightarrow{\gamma} q')$$

The result follows from Lemma 3.

$|\rho| = 1$. So $\rho$ is just a single transition, therefore according to the definition of $\lambda^*$ we have

$$\lambda^*(q \xrightarrow{\gamma} q') = \lambda^*(q \xrightarrow{\gamma} q')$$

Now we can use Lemma 3 again and the induction hypothesis (since $|\rho'| < |\rho|$) to get:

$$\lambda(q \xrightarrow{\gamma} q'') \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q, \gamma \rangle \Rightarrow^* \langle q'', \epsilon \rangle, \sigma \in \Delta^{pre}\}$$

$$\lambda^*(q'' \xrightarrow{\gamma'} q') \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q'', \gamma' \rangle \Rightarrow^* \langle q', \epsilon \rangle, \sigma \in \Delta^{pre}\}$$

29
Finally, we use the fact that the flow algebra is affine:

$$\mathcal{A}(q \xrightarrow{\rho} q')$$

$$\sqcup \{v(\sigma) \otimes v(\sigma') \mid \langle q, \gamma \rangle \xrightarrow{\sigma} \langle q'', \epsilon \rangle, \langle q'', s' \rangle \xrightarrow{\sigma'} \langle q', \epsilon \rangle, \sigma, \sigma' \in \Delta_{\text{pre}}\}$$

$$\sqcup \{v(\sigma) \mid \langle q, s \rangle \xrightarrow{\sigma} \langle q', \epsilon \rangle, \sigma \in \Delta_{\text{pre}}\}$$

\[\square\]

C.4 Proof of Thm. 3

Consider an automaton \(A_{\text{pre}}^C\) constructed by the saturation procedure and the least solution \(\lambda\) to the set of its constraints \(C\). If the flow algebra is affine then for every path \(\rho = p \xrightarrow{*} q_f\) where \(q_f \in F\) we have that

$$\mathcal{A}(p \xrightarrow{\rho} q_f) = \bigoplus \{v(\sigma) \mid \langle p, s \rangle \xrightarrow{\sigma} \langle q_f, \epsilon \rangle, \sigma \in \Delta_{\text{pre}}^*\}$$

**Proof.** The result follows directly from Theorem 1 and Lemma 4.

\[\square\]

C.5 Proof of Lem. 5

For every transition \(q' \xleftarrow{\gamma_f} q\) (\(\gamma_f \in \Gamma \cup \{\epsilon\}\)) in \(A_{\text{post}}^*\) there exists a sequence \(\sigma\) of pushdown rules in \(\Delta_{\text{post-2}}\) such that \(\langle q', \epsilon \rangle \xrightarrow{\sigma} \langle q, \gamma_f \rangle\).

**Proof.** Let us denote by \(\mathcal{A}_i\) the automaton \(\mathcal{A}\) after \(i\) steps of the saturation procedure. Proof will proceed by induction on \(i\).

\(i = 0\) Follows from the definition of \(\Delta_{\text{post-2}}\).

\(i > 0\) We assume the property holds for \(\mathcal{A}_i\) and prove it for \(\mathcal{A}_{i+1}\). Consider that the saturation procedure\(^4\):

- adds a transition \(q_d \xleftarrow{\epsilon} p_s\) because of a pushdown rule \(r = \langle p', \gamma' \rangle \rightarrow \langle p_s, \epsilon \rangle\) and a path \(q_d \xleftarrow{\gamma'} p'\). We can use the induction hypothesis to get that there exists \(\sigma\) such that \(\langle q_d, \epsilon \rangle \xrightarrow{\sigma} \langle p', \gamma' \rangle\). But then clearly \(\langle q_d, \epsilon \rangle \xrightarrow{\sigma} \langle p', \gamma' \rangle \xrightarrow{\epsilon} \langle p, \epsilon \rangle\).

- adds a transition \(q_d \xleftarrow{\gamma} p_s\) because of a pushdown rule \(r = \langle p', \gamma' \rangle \rightarrow \langle p_s, \epsilon \rangle\) and a path \(q_d \xleftarrow{\gamma'} p'\). We can use the induction hypothesis to get that there exists \(\sigma\) such that \(\langle q_d, \epsilon \rangle \xrightarrow{\sigma} \langle p', \gamma' \rangle\). Again it is clear that \(\langle q_d, \epsilon \rangle \xrightarrow{\sigma} \langle p', \gamma' \rangle \xrightarrow{\epsilon} \langle p, \epsilon \rangle\).

- adds \(q_{p_s, \gamma_1} \xleftarrow{\gamma_1} p_s\) and \(q_d \xleftarrow{\gamma_2} q_{p_s, \gamma_1}\) because of a pushdown rule \(r = \langle p', \gamma' \rangle \rightarrow \langle p_s, \gamma_1 \gamma_2 \rangle\) and a path \(q_d \xleftarrow{\gamma'} p'\). According to the definition of \(\Delta_{\text{post-2}}\) we know that there are \(r_1 = \langle p', \gamma' \rangle \rightarrow \langle q_{p_s, \gamma_1}, \gamma_2 \rangle\) and \(r_2 = \langle q_{p_s, \gamma_1}, \gamma_2 \rangle \rightarrow \langle p_s, \gamma_1 \gamma_2 \rangle\).

\(^4\) Note that the saturation procedure works on \(\Delta\).
\[ \langle q_{ps}, \epsilon \rangle \xrightarrow{\gamma} \langle p, \gamma_1 \rangle. \] So we immediately have the path for the first transition:

\[ \langle q_{ps}, \epsilon \rangle \xrightarrow{\gamma_1} \langle p, \gamma_1 \rangle. \]

Moreover, we can use the induction hypothesis to get that there exists \( \sigma \) such that \( \langle q_d, \epsilon \rangle \xrightarrow{\sigma} \ast \langle p', \gamma' \rangle \) and so we also have that

\[ \langle q_d, \epsilon \rangle \xrightarrow{\sigma} \ast \langle p', \gamma' \rangle \xrightarrow{\gamma_2} \langle q_{ps}, \gamma_1 \rangle. \]

\[ C.6 \] Proof of Lem. \[ C \]

Consider a weighted pushdown system \( \mathcal{W} = (P, F, f) \) where \( F \) is affine and an automaton \( \mathcal{A}_{post}^C \) created by the saturation procedure. For every transition \( q' \xrightarrow{\gamma, \epsilon} q \ (\gamma \in \Gamma \cup \{\epsilon\}) \) in this automaton we have that

\[ \lambda(q' \xrightarrow{\gamma, \epsilon} q) \subseteq \bigoplus \{v(\sigma) \mid \langle q', \epsilon \rangle \xrightarrow{\sigma} \ast \langle q, \gamma \rangle, \sigma \in \Delta_{post-2}^* \} \]

\[ \text{Proof.} \] Let us denote by \( \mathcal{A}_{i}^C \) the automaton \( \mathcal{A}_{i}^C \) after \( i \) steps of saturation procedure and similarly the least solution for it by \( \lambda_i \). We will prove by induction on \( i \) that for every transition \( q' \xrightarrow{\gamma, \epsilon} q \) we have that

\[ \lambda_i(q' \xrightarrow{\gamma, \epsilon} q) \subseteq \bigoplus \{v(\sigma) \mid \langle q', \epsilon \rangle \xrightarrow{\sigma} \ast \langle q, \gamma \rangle, \sigma \in \Delta_{post-2}^* \} \]

\( i = 0 \) The only constraints are of the form \( I \subseteq l(t) \) where \( t \) is a transition in \( A \).

Therefore, the least solution for each \( t \) is \( \lambda_i(t) = I \). We also know that for every \( r \in \Delta_{post-2} \setminus \Delta \), \( f(r) = I \). So the right hand side is at least \( I \). Thus our property holds.

\( i > 0 \) We assume the property holds for \( \mathcal{A}_{i}^C \) and prove it for \( \mathcal{A}_{i+1}^C \), i.e., prove that adding a constraint (and maybe a transition as well) preserves the property of interest.

Let \( t \) be the transition that the added constraint refers to. Observe that if \( t \) was already in the automaton \( \mathcal{A}_{i}^C \), then it is possible that \( h(t) \) might be on the left-hand side of some other constraint. Therefore, the least solution for the new set of constraints might be different for other transitions as well; in other words the value/information from the new constraint might have to be propagated throughout other constraints to get \( \lambda_{i+1} \). Now let \( \lambda^0_i \) denote the solution after \( j \) steps of fixed point computation with the new constraint, starting with

\[ \lambda^0_i(t) = \begin{cases} \emptyset & \text{if } t \text{ was added} \\ \lambda_i(t) & \text{otherwise (} t \text{ was in } \mathcal{A}_{i}^C \) \end{cases} \]

Using induction on \( j \) we will prove that the property is maintained by the computation.

Note that we can use here Kleene iteration due to Lemma \( \square \)
\( j = 0 \) Immediate from outer induction hypothesis.

\( j > 0 \) We assume the property hold for \( \lambda^j \) and prove that it also holds for \( \lambda^{j+1} \). In the following we use the fact that the flow algebra is affine, this is enough since from Lemma \( \text{[5]} \) it follows that the sets (of pushdown paths) on the right hand sides are not empty. Let us consider three possibilities of constraints:

- if the constraint is

  \[
  \begin{align*}
  h(q \xleftarrow{\epsilon} p') \otimes f(r) & \sqsubseteq h(q \xleftarrow{\epsilon} p) \\
  \end{align*}
  \]

  or

  \[
  \begin{align*}
  h(q \xleftarrow{\epsilon} q'') \otimes h(q'' \xleftarrow{\epsilon} p') \otimes f(r) & \sqsubseteq h(q \xleftarrow{\epsilon} p) \\
  \end{align*}
  \]

  where \( r = (q', \gamma') \xrightarrow{\epsilon} (p, \epsilon) \in \Delta \). Let us only consider the more complex case with additional \( \epsilon \) transition (the one without is similar).

  We need to calculate the value of \( \lambda^{j+1} \) — it should be its old value combined with the new one

  \[
  \lambda^{j+1}(q \xleftarrow{\epsilon} p) = \lambda^j(q \xleftarrow{\epsilon} p) \oplus \left( \lambda^j(q \xleftarrow{\epsilon} q'') \otimes \lambda^j(q'' \xleftarrow{\epsilon} p') \otimes f(r) \right)
  \]

  Let us use the induction hypothesis (inner induction) three times to get:

  \[
  \begin{align*}
  \lambda^j(q \xleftarrow{\epsilon} p) & \subseteq \bigoplus \{ v(\sigma) \mid (q, \epsilon) \xrightarrow{\sigma}^* (p, \epsilon), \sigma \in \Delta^*_{\text{post-2}} \} \\
  \lambda^j(q \xleftarrow{\epsilon} q'') & \subseteq \bigoplus \{ v(\sigma) \mid (q, \epsilon) \xrightarrow{\sigma}^* (q'', \gamma'), \sigma \in \Delta^*_{\text{post-2}} \} \\
  \lambda^j(q'' \xleftarrow{\epsilon} p') & \subseteq \bigoplus \{ v(\sigma) \mid (q'', \epsilon) \xrightarrow{\sigma}^* (p', \epsilon), \sigma \in \Delta^*_{\text{post-2}} \}
  \end{align*}
  \]

  Using the above and the fact that our flow algebra is affine, we get:

  \[
  \begin{align*}
  \lambda^j(q \xleftarrow{\epsilon} q'') \otimes \lambda^j(q'' \xleftarrow{\epsilon} p') \otimes f(r) & \subseteq \bigoplus \{ v(\sigma_1) \otimes v(\sigma_2) \otimes f(r) \mid (q, \epsilon) \xrightarrow{\sigma_1}^* (q'', \gamma'), (q'', \epsilon) \xrightarrow{\sigma_2}^* (p', \epsilon), (p', \gamma') \xrightarrow{\epsilon} (p, \epsilon), \sigma \in \Delta^*_{\text{post-2}} \} \\
  \end{align*}
  \]

  Now since \( \bigoplus \) gives the least upper bound, we have that

  \[
  \lambda^{j+1}(q \xleftarrow{\epsilon} p) \subseteq \bigoplus \{ v(\sigma) \mid (q, \epsilon) \xrightarrow{\sigma}^* (p, \epsilon), \sigma \in \Delta^*_{\text{post-2}} \}
  \]
– if the constraint is
\[ h(q \xleftarrow{r} p') \otimes f(r) \subseteq h(q \xleftarrow{r} p) \]
or
\[ h(q \xleftarrow{r} q'') \otimes h(q'' \xleftarrow{r} p') \otimes f(r) \subseteq h(q \xleftarrow{r} p) \]
where \( r \) must be \( r = (p', \gamma') \mapsto (p, \gamma) \in \Delta \). The case is analogous to the previous one (we just have \( \gamma \) instead of \( \epsilon \)).
– if the constraint is one of
\[ I \subseteq h(q_{p, \gamma_1} \xleftarrow{r} p) \]
or
\[ h(q \xleftarrow{r} q'') \otimes h(q'' \xleftarrow{r} p') \otimes f(r) \subseteq h(q \xleftarrow{r} q_{p, \gamma_1}) \]
(alternatively without the \( \epsilon \)-transition:
\[ h(q \xleftarrow{r} p') \otimes f(r) \subseteq h(q \xleftarrow{r} q_{p, \gamma_1}) \]
but we will only consider the former, since it is a bit more complex and the proof for the latter is almost the same).
We know that \( r = (p', \gamma') \mapsto (p, \gamma_1 \gamma_2) \in \Delta \) and so that we have \( r_1, r_2 \in \Delta_{\text{post-2}} \) such that \( r_1 = (p', \gamma') \mapsto (q_{p, \gamma_1}, \gamma_2) \) and \( r_2 = (q_{p, \gamma_1}, \epsilon) \mapsto (p, \gamma_1) \) with \( f(r_1) = f(r) \) and \( f(r_2) = I \).
For the first trivial inequality the property is clearly preserved. Let us focus on the second one. We know that
\[ \lambda_i^{i+1}(q \xleftarrow{r} q_{p, \gamma_2}) = \lambda_i^1(q \xleftarrow{r} q_{p, \gamma_2}) \]
\[ \oplus \left( \lambda_i^1(q \xleftarrow{r} q'') \otimes \lambda_i^1(q \xleftarrow{r} p') \otimes f(r) \right) \] (2)
for some \( q' \in Q \). Using induction hypothesis we have that:
\[ \lambda_i^1(q \xleftarrow{r} q_{p, \gamma_1}) \subseteq \bigoplus \{ v(\sigma) \mid \langle q, \epsilon \rangle \xrightarrow{\sigma}^* (q_{p, \gamma_1}, \gamma_2) \} \]
(3)
\[ \lambda_i^1(q \xleftarrow{r} q'') \subseteq \bigoplus \{ v(\sigma) \mid \langle q, \epsilon \rangle \xrightarrow{\sigma}^* (q', \gamma') \} \]
\[ \lambda_i^1(q \xleftarrow{r} p') \subseteq \bigoplus \{ v(\sigma) \mid \langle q, \epsilon \rangle \xrightarrow{\sigma}^* (p', \epsilon) \} \]
Using the last two and the fact that the flow algebra is affine, we get the following
\[ \lambda_i^1(q \xleftarrow{r} q'') \otimes \lambda_i^1(q \xleftarrow{r} p') \otimes f(r) \]
\[ \subseteq \{ v(\sigma_1) \otimes v(\sigma_2) \otimes f(r) \mid \langle q, \epsilon \rangle \xrightarrow{\sigma_2}^* (q', \gamma'), \langle q', \epsilon \rangle \xrightarrow{\sigma_3}^* (p', \epsilon) \} \]
\[ \subseteq \{ v(\sigma) \otimes f(r_1) \otimes f(r_2) \mid \langle q, \epsilon \rangle \xrightarrow{\sigma_1}^* (q_{p, \gamma_1}, \gamma') \xrightarrow{r_1} (q_{p, \gamma_1}, \gamma_2) \xrightarrow{r_2} (p, \gamma_1 \gamma_2) \} \]
\[ \subseteq \{ v(\sigma) \mid \langle q, \epsilon \rangle \xrightarrow{\sigma}^* (p, \gamma_1 \gamma_2) \} \]
So from this and [2] and [3] we have the desired result.
C.7 Proof of Lem. 7

Consider a weighted pushdown system \( W = (P, F, f) \) where \( F \) is affine and a \( A^*_{\text{post}} \) automaton created by the saturation procedure. For every path \( \rho = q' \xrightarrow{\gamma} q \ (s \in \Gamma^*) \) in this automaton we have that

\[
\lambda^*_R(q' \xrightarrow{\gamma} q) \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q', \epsilon \rangle \xrightarrow{\sigma} \langle q, s \rangle, \sigma \in \Delta^*_{\text{post-2}}\}
\]

**Proof.** The proof will proceed with the induction on the number of transitions in \( \rho \) (we will use the inductive definition of \( \lambda \)).

| \( |\rho| = 1 \) |
|---|
| According to the definition of \( \lambda \) we have |
| \[
\lambda^*_R(q' \xrightarrow{\gamma} q) = \lambda(q' \xrightarrow{\gamma} q)
\]
| The result follows from Lemma 6. |

| \( |\rho| > 1 \) |
|---|
| Again using the definition of \( \lambda^*_R \) we have |
| \[
\lambda^*_R(q' \xrightarrow{\gamma} q) = \lambda^*_R(q' \xrightarrow{\gamma'} q') \otimes \lambda(q' \xrightarrow{\gamma} q)
\]
| where \( s = \gamma \epsilon \gamma', q'' \in Q \), and |
| \[
\rho = q' \xrightarrow{\gamma'} q'' \xrightarrow{\gamma} q
\]
| Now we can use the Lemma 6 along with the induction hypothesis (since |\( |\rho| > |\rho'| | \) to get: |
| \[
\lambda(q'' \xrightarrow{\gamma} q) \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q'', \epsilon \rangle \xrightarrow{\sigma} \langle q', s \rangle, \sigma \in \Delta^*_{\text{post-2}}\}
\]
| \[
\lambda^*_R(q' \xrightarrow{\gamma} q) \sqsubseteq \bigoplus \{v(\sigma) \mid \langle q', \epsilon \rangle \xrightarrow{\sigma} \langle q'' \xrightarrow{\gamma} q, s \rangle, \sigma \in \Delta^*_{\text{post-2}}\}
\]
| Finally, we use the fact that the flow algebra is affine: |
| \[
\lambda^*_R(q' \xrightarrow{\gamma} q) \sqsubseteq \bigoplus \{v(\sigma) \otimes v(\sigma') \mid \langle q' \xrightarrow{\gamma} q, \epsilon \rangle \xrightarrow{\sigma} \langle q'' \xrightarrow{\gamma} q, s \rangle, \sigma, \sigma' \in \Delta^*_{\text{post-2}}\}
\]
| \[
\sqsubseteq \bigoplus \{v(\sigma) \otimes v(\sigma') \mid \langle q' \xrightarrow{\gamma} q, \epsilon \rangle \xrightarrow{\sigma} \langle q, s \rangle, \sigma \in \Delta^*_{\text{post-2}}\}
\]

C.8 Proof of Thm. 4

Consider an automaton \( A^*_{\text{post}} \) constructed by the saturation procedure and the least solution \( \lambda \) to the set of its constraints \( \mathcal{C} \). If the flow algebra is affine then for every path \( \rho = qf \xrightarrow{\gamma} p \) where \( qf \in F \) we have that

\[
\lambda^*_R(qf \xrightarrow{\gamma} p) = \bigoplus \{v(\sigma) \mid \langle qf, \epsilon \rangle \xrightarrow{\sigma} \langle p, s \rangle, \sigma \in \Delta^*_{\text{post-2}}\}
\]

**Proof.** Follows directly from Theorem 2 and Lemma 7. \( \square \)