Weighted Dynamic Pushdown Networks

Alexander Wenner

Institut für Informatik, Fachbereich Mathematik und Informatik
Westfälische Wilhelms-Universität Münster
alexander.wenner@uni-muenster.de

Abstract. We develop a generic framework for the analysis of programs with recursive procedures and dynamic process creation. To this end we combine the approach of weighted pushdown systems (WPDS) with the model of dynamic pushdown networks (DPN). Weighted dynamic pushdown networks (WDPN) describe processes running in parallel. Each process may perform pushdown actions and spawn new processes. Transitions are labelled by weights to carry additional information. We derive a method to determine meet-over-all-paths values for the paths from a starting configuration to a regular set of configurations of a WDPN.

1 Introduction

The interest in writing parallel programs has increased in recent years. However parallel programming is notoriously difficult and error-prone. Thus static analysis of parallel programs has become more and more important. The goal of this paper is to present a generic framework for the analysis of parallel programs, especially in the presence of recursive procedures and dynamic process creation. We base our framework on DPN [1] and WPDS [2]. DPN precisely model procedures and process creation and have been studied for reachability analyses. Since the analysis of recursive procedures and synchronisation is undecidable [3], DPNs do not model synchronisation between processes. However, through the addition of weights we will be able to analyse some interaction between processes. WPDS extend pushdown systems (PDS) by labelling transitions with weights and solving the generalised pushdown predecessor (GPP) problem, which is the meet-over-all-paths solution for paths from a starting configuration into a regular set of target configurations. The weights can be used to formulate a wide range of analysis problems. The GPP problem formulation allows for a specific query depending on the shape of the entire call-stack, in contrast to standard dataflow techniques, where typically all information at the topmost program point is merged. Analogous to WPDS we extend DPN to WDPN by annotating weights to transitions and study the GPP problem. Even though a WPDS is then simply a WDPN with one process, adapting the approach to solve the GPP problem from WPDS to WDPN is problematic. In general a path of a DPN is an interleaving of the transitions of arbitrary many parallel processes. Results from [1] show, that such a set of paths can not be described using a constraint system. We avoid these problems by introducing a branching semantics for DPN

Mathematical and Engineering Methods in Computer Science (MEMICS), Znojmo, Czech Republic, 2009.
Petr Hliněný, Vashek Matyáš, Tomáš Vojnar (Eds.)
Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany
Digital Object Identifier: 10.4230/DROPS.MEMICS.2009.2344

similar to the tree semantics in [4]. Transitions of newly spawned processes are no longer mixed with the transitions of the creating process, but contained in their own branch. This results in executions which are tree shaped for single processes and form hedges, which contain a tree for each process, for configurations with multiple processes. We introduce an extended weight domain to abstract these trees, and study the analogous branching GPP (BGPP) problem, which is the meet-over-all-hedges solution, for these branching WDPN (BWDPN). We show, that if the weight domain of a WDPN and the extended weight domain of a BWDPN, based on the same DPN, are related, the solution for the GPP problem of the WDPN can be derived from the solution of the corresponding BGPP problem of the BWDPN. The BGPP problem can be solved using an approach adapted from WPDS.

Up to this point our framework of WDPN and BWDPN can solve the bitvector problems for DPNs formulated in [1], the more general KILL/GEN analyses described in [5] and the shortest path analysis from [2]. In [6] a different approach to generalize WPDS to parallel programs is presented, by introducing a context bound. This leads to an underapproximation, whereas our approach handles unbounded context switches precisely.

The remainder of the paper is organised as follows: Section 2 presents the intuitive extension of WPDS to DPN called WDPN and defines the GPP. Section 3 introduces BWDPN. We formulate the BGPP problem and present the relation to the GPP problem. Section 4 presents two applications and Section 5 presents the approach to solve the BGPP problem for BWDPN.

2 Weighted Dynamic Pushdown Networks

A DPN [1] is a model for parallel programs with multiple processes and dynamic process creation. Each process is modeled as a PDS, where the rules are extended to allow creation of new processes. Formally a DPN is a tuple $\mathcal{M} = (P, \Gamma, \Delta)$, where P is a finite set of control states, Γ is a finite set of stack symbols, with $P \cap \Gamma = \emptyset$, and Δ is a finite set of transition rules of the form $p\gamma \hookrightarrow c$ with $p \in P$, $\gamma \in \Gamma$ and $c \in (P\Gamma^*)^*P\Gamma^*$. The right side of a rule consists of the new control state and stacktop of the original process in the rightmost position and the control states and stacks of all processes spawned by this rule to the left. Configurations of the DPN are words from $\mathsf{Conf} = (P\Gamma^*)^*$. The empty configuration is written as ε. For the rest of the paper we fix a DPN $\mathcal{M} = (P, \Gamma, \Delta)$, a configuration c and a regular set $C \subseteq \mathsf{Conf}$.

An execution of a DPN is represented by a path. A path is defined as a sequence of rules $\rho = r_1 \dots r_n$ with $r_i \in \Delta$. The empty path is denoted by ε_ρ and Paths is the set of all paths. The execution of a path is modeled by the labelled transition relation $\longrightarrow \subseteq \mathsf{Conf} \times \mathsf{Paths} \times \mathsf{Conf}$, where for $c, c' \in \mathsf{Conf}$, $p \in P$, $\gamma \in \Gamma$, $u \in (P\Gamma^*)^*$ and $v \in \Gamma^*(P\Gamma^*)^*$:

$$[\mathsf{empty}]\ c \xrightarrow{\varepsilon_\rho} c \qquad [\mathsf{rule}]\ up\gamma v \xrightarrow{r\rho} c \ \text{ if } r = p\gamma \hookrightarrow c' \text{ and } uc'v \xrightarrow{\rho} c$$

Application of a rule replaces the control state and top symbol of one stack by the new control state and stacktop specified by the rule and inserts the newly created processes with their initial stacks to the left. We call this the interleaving semantics of the DPN, since the rules of all processes are mixed together. We are interested in the set $\mathsf{Paths}(c, C) = \{\rho \in \mathsf{Paths} \mid \exists c' \in C \text{ with } c \xrightarrow{\rho} c'\}$ of connecting paths from c to C.

In order to abstract from the set of connecting paths to the aspects which are relevant to the desired analysis, we assign a weight to each transition of the DPN. The structure of the weight domain is captured by a complete idempotent semiring, which supports the necessary operators $\odot$ for concatenation of weights along a path and $\oplus$ for combination of weights of different paths. A complete idempotent semiring is a tuple $\mathcal{S} = (D, \oplus, \odot, 0, 1)$, where D is a set of elements with $0, 1 \in D$ and $\oplus, \odot$ are binary operators on D with:

- $(D, \oplus)$ is a commutative monoid with neutral element 0 and $\oplus$ is idempotent
- $(D, \odot)$ is a monoid with neutral element 1 and 0 annihilates $\odot$
- $(D, \sqsubseteq)$ is a complete lattice, where $\sqsubseteq$, with $d_1 \sqsubseteq d_2 :\Leftrightarrow d_1 \oplus d_2 = d_1$ for $d_1, d_2 \in D$, is the partial order induced by $\oplus$
- $\odot$ distributes over arbitrary $\oplus$, i.e. $\bigoplus D_1 \odot \bigoplus D_2 = \bigoplus\{d_1 \odot d_2 \mid d_i \in D_i\}$ for $D_1, D_2 \subseteq D$

Furthermore we assume, that a weight function $f : \Delta \to D$ is given. The weight function assigns a weight to each transition of our DPN and depends on the analysis, since it describes how the transitions of the DPN are connected to the analysed information represented by the semiring. We fix the tuple $\mathcal{W} = (\mathcal{M}, \mathcal{S}, f)$, with $\mathcal{S} = (D, \oplus, \odot, 0, 1)$, called a WDPN, for the rest of the paper and define an abstraction function $\alpha : \mathsf{Paths} \to D$ for paths:

$$[\mathsf{empty}]\ \alpha(\varepsilon_\rho) = 1 \quad [\mathsf{rule}]\ \alpha(r\rho) = f(r) \odot \alpha(\rho)$$

Overloading it for sets of paths with $\alpha(M) = \bigoplus\{\alpha(\rho) \mid \rho \in M\}$, we can formulate the GPP problem for WDPN as computing $\delta(c, C) = \alpha(\mathsf{Paths}(c, C))$.

3 Branching Weighted Dynamic Pushdown Networks

It follows from results in [1] that the set $\mathsf{Paths}(c, C)$ can not be characterised as least solution of a constraint system. Therefore we can not compute the solution for the GPP problem directly by an abstract interpretation [7] of such a constraint system. To avoid these problems we consider an alternative interpretation of an execution of a DPN in form of a tree or hedge, first introduced in [4]. The set of connecting hedges can then be described using a constraint system.

We recursively define the sets Trees and $\mathsf{Hedges} = \mathsf{Trees}^*$ of execution trees and hedges. The empty hedge is written as ε_σ. The empty tree ε_τ consisting of a single leaf node, representing a finished execution, is a tree. $r(\sigma\tau)$ is a tree with a root node labelled with a rule $r \in \Delta$, describing the first step of the execution, and an ordered list of subtrees $\sigma\tau \in \mathsf{Hedges}$, representing the executions σ of

spawned processes and the rest of the execution τ of the spawning process. We define the **;** operator to concatenate a tree to the last tree of a hedge:

$$[\mathsf{hedge}]\ (\sigma\tau)\,;\tau' = \sigma(\tau\,;\tau') \quad [\mathsf{empty}]\ \varepsilon_\tau\,;\tau' = \tau' \quad [\mathsf{rule}]\ r(\sigma)\,;\tau' = r(\sigma\,;\tau')$$

Appending a tree replaces the rightmost leaf of the hedge with that tree. Thus concatenation of trees is concatenation of the rightmost branches. Since the rightmost branch represents the execution of the initial process, this will later be used to assemble execution trees from partial executions of an initial process. The execution of a hedge is modeled by the labelled transition relation $\Longrightarrow \subseteq \mathsf{Conf} \times \mathsf{Hedges} \times \mathsf{Conf}$, where for $c, c', \tilde{c} \in \mathsf{Conf}$, $p \in P$, $\gamma \in \Gamma$ and $w \in \Gamma^*$:

$$[\mathsf{none}]\ \ \varepsilon \overset{\varepsilon_\sigma}{\Longrightarrow} \varepsilon \qquad [\mathsf{tree}]\ cpw \overset{\sigma\tau}{\Longrightarrow} c'\tilde{c}\ \text{ if } c \overset{\sigma}{\Longrightarrow} c' \text{ and } pw \overset{\tau}{\Longrightarrow} \tilde{c}$$
$$[\mathsf{empty}]\ pw \overset{\varepsilon_\tau}{\Longrightarrow} pw \quad [\mathsf{rule}]\ p\gamma w \overset{r(\sigma)}{\Longrightarrow} c \quad \text{if } r = p\gamma \hookrightarrow c' \text{ and } c'w \overset{\sigma}{\Longrightarrow} c$$

We call this the branching semantics of the DPN, since each process has its own branch in the execution. We are interested in the set $\mathsf{Hedges}(c, C) = \{\sigma \in \mathsf{Hedges} \mid \exists c' \in C \text{ with } c \overset{\sigma}{\Longrightarrow} c'\}$ of connecting hedges.

To abstract hedges we define an extended complete idempotent semiring, which contains the additional $\bar{\otimes}$ operator for parallel combination of weights. An extended complete idempotent semiring $\mathcal{E} = (E, \bar{\oplus}, \bar{\odot}, \bar{\otimes}, \bar{0}, \bar{1})$ is a tuple, where E is a set of values and $\bar{\oplus}, \bar{\odot}, \bar{\otimes}$ are binary operators on E with:

- $(E, \bar{\oplus}, \bar{\odot}, \bar{0}, \bar{1})$ is a complete idempotent semiring
- $(E, \bar{\otimes})$ is a semigroup, $\bar{1} \,\bar{\otimes}\, e = e$ for $e \in E$ and $\bar{0}$ annihilates $\bar{\otimes}$
- $\bar{\otimes}$ distributes over arbitrary $\bar{\oplus}$, i.e. $\bar{\bigoplus} E_1 \,\bar{\otimes}\, \bar{\bigoplus} E_2 = \bar{\bigoplus}\{e_1 \,\bar{\otimes}\, e_2 \mid e_i \in E_i\}$ for $E_1, E_2 \subseteq E$
- $(e_1 \,\bar{\otimes}\, e_2) \,\bar{\odot}\, e_3 = e_1 \,\bar{\otimes} (e_2 \,\bar{\odot}\, e_3)$, for $e_1, e_2, e_3 \in E$

The fourth property ensures, that **;** is abstracted by $\bar{\odot}$, by always appending weights to the rightmost weight of a parallel combination. In this regard the $\bar{\otimes}$ operator differs from the interleaving operator $\otimes$ introduced in [8], since weights that are concatenated after an interleaving need to be considered as well.

Furthermore we assume, as with WDPN, that a weight function $\bar{f} : \Delta \to E$ is given. We fix the tuple $\mathcal{B} = (\mathcal{M}, \mathcal{E}, \bar{f})$, with $\mathcal{E} = (E, \bar{\oplus}, \bar{\odot}, \bar{\otimes}, \bar{0}, \bar{1})$, called BWDPN, for the rest of the paper and define an abstraction function $\beta : \mathsf{Hedges} \to E$ for hedges:

$$[\mathsf{none}]\ \ \beta(\varepsilon_\sigma) = \bar{1} \quad [\mathsf{tree}]\ \beta(\sigma\tau) \ \ = \beta(\sigma) \,\bar{\otimes}\, \beta(\tau)$$
$$[\mathsf{empty}]\ \beta(\varepsilon_\tau) = \bar{1} \quad [\mathsf{rule}]\ \beta(r(\sigma)) = \bar{f}(r) \,\bar{\odot}\, \beta(\sigma)$$

Overloading it for sets of hedges with $\beta(M) = \bar{\bigoplus}\{\beta(\sigma) \mid \sigma \in M\}$, we define the BGPP problem for BWDPN as computing $\theta(c, C) = \beta(\mathsf{Hedges}(c, C))$.

There is a strong connection between the interleaving and branching semantics of a DPN. A hedge represents of a set of paths, which can be constructed by interleaving the branches and trees of the hedge. In [4] it was shown, that if we take a function $\psi : \mathsf{Hedges} \to 2^{\mathsf{Paths}}$ that computes the set of interleavings of a hedge, and overload it for sets of hedges, we have:

Theorem 1. $\mathsf{Paths}(c, C) = \psi(\mathsf{Hedges}(c, C))$

A similar result can be shown for the solutions of the GPP and BGPP problems, if the semiring of the WDPN is related to the extended semiring of the BWDPN. We describe the necessary relation by an extension. An extension is a tuple $(\mathcal{S}, \mathcal{E}, \iota, \eta)$, containing embedding and projection functions $\iota : D \to E$ and $\eta : E \to D$, where for $d, d_i \in D, e, e_i \in E$ the following conditions must hold:

- E is the smallest set with $\iota(D) \subseteq E$, closed under $\bar{\odot}, \bar{\otimes}$ and arbitrary $\bar{\oplus}$
- $\iota(0) = \bar{0}$, $\iota(1) = \bar{1}$ and $\eta(\iota(d)) = d$
- η distributes over arbitrary $\bar{\oplus}$, i.e. $\eta(\bar{\bigoplus} M) = \bigoplus\{\eta(e) \mid e \in M\}$ for $M \subseteq E$
- $\eta(\iota(d) \,\bar{\odot}\, e) = d \odot \eta(e)$
- $\eta(e_1 \,\bar{\otimes} \ldots \bar{\otimes}\, e_n) = \eta(e_{i_1} \,\bar{\otimes} \ldots \bar{\otimes}\, e_{i_m})$ with $e_i = \bar{1}$ for $i \notin \{i_1, \ldots, i_m\}$
- $\eta(e_1 \,\bar{\otimes} \ldots \bar{\otimes}\, e_n) = \bigoplus_{i=1}^{n} d_i \odot \eta(e_1 \,\bar{\otimes} \ldots \bar{\otimes}\, e'_i \,\bar{\otimes} \ldots \bar{\otimes}\, e_n)$ with $e_i = \iota(d_i) \,\bar{\odot}\, e'_i$

The first three points ensure, that every weight of the original semiring has a corresponding weight in the extended semiring. The fourth point guarantees, that a simple concatenation of extended weights is mapped to the corresponding concatenation of weights. The last two points ensure, that the parallel combination of extended weights is mapped to the meet over all interleavings of the weights they are constructed from. For the rest of the paper, we assume that the semiring and extended semiring are connected by the extension $(\mathcal{S}, \mathcal{E}, \iota, \eta)$.

If $\bar{f}(r) = \iota(f(r))$, for all $r \in \Delta$, i.e. the analysis of the WDPN is embedded in the BWDPN, we can proof $\alpha(\psi(\sigma)) = \eta(\beta(\sigma))$ for all $\sigma \in \mathsf{Hedges}$ by induction on σ. Consequently with Theorem 1 we have:

Theorem 2. $\delta(c, C) = \eta(\theta(c, C))$.

4 Applications

Since the existence of an extended semiring and a matching extension for a given semiring is not self-evident, we first give two examples of semirings, for which an extended semiring and a corresponding extension can be constructed, before describing the approach to solve the BGPP problem in Section 5.

The shortest path analysis assigns a positive integer weight to all transitions. The weight of a path is the sum of the weights of the transitions occurring on the path. The goal is to find the weight of the path with the smallest weight. We use the semiring $\mathcal{S} = (\mathbb{N} \cup \{0, \infty\}, min, +, \infty, 0)$ introduced in [2]. Since $+$ is commutative and associative, the order in which transitions occur and are combined on a path is irrelevant. Thus $+$ can be used as the interleaving operator $\bar{\otimes}$ in an extended semiring. The semiring in combination with the interleaving operator fulfills all necessary conditions for an extended semiring $\mathcal{E} = (\mathbb{N} \cup \{0, \infty\}, min, +, +, \infty, 0)$ and the matching extension is simply $(\mathcal{S}, \mathcal{E}, id, id)$.

Bitvector Analyses analyse a property represented by a single bit. For lack of space, we consider only forward may bitvector analysis. Backward or must analyses can be handled similarly. The transitions of the DPN are annotated with transformers, that change the current state of the bit. We use the semiring

$\mathcal{S} = (D, \oplus, \odot, \mathsf{zero}, \mathsf{id})$, where $D = \{\mathsf{gen}, \mathsf{id}, \mathsf{kill}, \mathsf{zero}\}$. Here gen represents the transformer setting the bit to 1, id is the identity and kill sets the bit to 0. The artificial weight zero is introduced to represent the zero element of the ring. $\odot$ is reversed functional concatenation extended to include zero. $\oplus$ is a meet operator inducing the ordering $\mathsf{gen} \sqsubseteq \mathsf{id} \sqsubseteq \mathsf{kill} \sqsubseteq \mathsf{zero}$. In [8] it was shown, that the operator $\otimes$, defined as $f \otimes g = (f \odot g) \oplus (g \odot f)$, is an interleaving operator on the path level. However the semiring in combination with the interleaving operator can not be used as extended semiring, since it does not fulfill the property $(f \otimes g) \odot h = f \otimes (g \odot h)$ for all $f, g, h \in D$. Especially for $f = \mathsf{gen}, g = \mathsf{id}$ and $h = \mathsf{kill}$, the terms $(f \otimes g) \odot h = \mathsf{kill}$ and $f \otimes (g \odot h) = \mathsf{gen}$ evaluate differently. This is caused by the fact, that a gen occurring in a parallel process can always be executed last in an interleaving and reset the bit. However the operator $\otimes$ does only consider the gen to be parallel to g and not the later appended h. We solve this problem by introducing a new weight $\overline{\mathsf{gen}}$, that stores the information, that a gen weight was encountered in parallel. This leads to the extended semiring $(\{\overline{\mathsf{gen}}, \mathsf{gen}, \mathsf{id}, \mathsf{kill}, \mathsf{zero}\}, \bar{\oplus}, \bar{\odot}, \bar{\otimes}, \mathsf{zero}, \mathsf{id})$ and extension $(\mathcal{S}, \mathcal{E}, \iota, \eta)$, where $\bar{\oplus}$ induces the ordering $\overline{\mathsf{gen}} \sqsubseteq \mathsf{gen} \sqsubseteq \mathsf{id} \sqsubseteq \mathsf{kill} \sqsubseteq \mathsf{zero}$ and:

$$f \mathbin{\bar{\odot}} g = \begin{cases} f \odot g & \text{if } f, g \neq \overline{\mathsf{gen}} \\ \overline{\mathsf{gen}} & \text{if } f = \overline{\mathsf{gen}} \text{ or } g = \overline{\mathsf{gen}} \end{cases} \qquad f \mathbin{\bar{\otimes}} g = \begin{cases} f \mathbin{\bar{\odot}} g & \text{if } f \notin \{\overline{\mathsf{gen}}, \mathsf{gen}\} \\ \overline{\mathsf{gen}} & \text{if } f \in \{\overline{\mathsf{gen}}, \mathsf{gen}\} \end{cases}$$

$$\eta(f) = \begin{cases} f & \text{if } f \in \{\mathsf{gen}, \mathsf{id}, \mathsf{kill}, \mathsf{zero}\} \\ \mathsf{gen} & \text{if } f = \overline{\mathsf{gen}} \end{cases} \qquad \iota(f) = f$$

5 Solving the BGPP Problem for BWDPN.

We use $\mathcal{M}$- and $\mathcal{M}^*$-automata, adapted from [1], as a compact representation for the target set. A $\mathcal{M}^*$-automaton is a finite automaton $\mathcal{A}^* = (S, P \cup \Gamma, \delta, \dot{s}, F)$ that satisfies the following additional conditions:

- $S_c, S_p \subseteq S$, where for all $s \in S_c, p \in P$ exists a unique and distinguished state $s_p \in S_p$
- $\delta = \delta_P \cup \delta_\Gamma$ where $\delta_P = \{(s, p, s_p) \mid s \in S_c, p \in P\}$ and $\delta_\Gamma \subseteq S \times (\Gamma \cup \{\varepsilon\}) \times S$
- $\mathcal{L}(\mathcal{A}^*) \subseteq \mathsf{Conf}$

A $\mathcal{M}$-automaton $\mathcal{A}$ is a $\mathcal{M}^*$-automaton, where $\dot{s} \in S \setminus S_p$ and the transition relation δ satisfies the stronger condition $\delta_\Gamma \subseteq S \times (\Gamma \cup \{\varepsilon\}) \times (S \setminus S_p)$. We write $s \xrightarrow{\lambda}_\delta s'$ for $(s, \lambda, s') \in \delta$ and $s \xrightarrow{c}{}^*_\delta s'$ for the transitive closure. For the rest of the paper we fix an $\mathcal{M}$-automaton $\mathcal{A} = (S, P \cup \Gamma, \delta, \dot{s}, F)$ describing the set C.

Now consider an execution hedge in $\mathsf{Hedges}(c, C)$. Each tree of the hedge transforms a stack in c into a configuration containing the transformed original stack and stacks of spawned processes. Analogous to the approach in [2], we can split each tree into several phases along the rightmost branch, each transforming a stacksymbol of the corresponding initial stack. During these transformation new processes may be spawned and transformed themselves. The idea is to compute for each symbol of the starting configuration the set of trees, that transform the stack symbol into a configuration, that is part of a configuration in C.

To this end we take a closer look at the saturation procedure used in [1] to construct the set $PRE^*(C) = \{c \mid \exists c' \in \mathsf{Conf}, \sigma \in \mathsf{Hedges} \text{ with } c \overset{\sigma}{\Longrightarrow} c'\}$ of all predecessor configurations. The saturation procedure works by adding new transitions to the automaton $\mathcal{A}$, thus allowing more configurations to be accepted. The result is a $\mathcal{M}^*$-automaton $\mathcal{A}^* = (S, P \cup \Gamma, \delta', \dot{s}, F)$, with $\delta' = \delta_P \cup \delta'_\Gamma$, where δ'_Γ is the smallest set fulfilling the conditions:

$$\begin{array}{ll} [\mathsf{init}] & t \in \delta'_\Gamma \text{ if } t \in \delta_\Gamma \\ [\mathsf{step}] & (s_p, \gamma, s') \in \delta'_\Gamma \text{ if } r = p\gamma \hookrightarrow c \in \Delta, s \in S_c \text{ and } s \xrightarrow{c}{}^*_{\delta'} s' \end{array}$$

A transition is added, if there is a rule transforming the symbol into a configuration which can be read by previously existing transitions. If these transitions were also added by the saturation, they themselves have a rules, which transform their symbols. If we follow this recursion and assemble the rules into a tree, we have a tree that transform the symbol of the newly added transition into a configuration that can be read using only transition of $\mathcal{A}$ and therefore is part of a configuration in C. We extend the saturation procedure to keep track of these trees by constructing a constraint system L over $(2^{\mathsf{Trees}}, \cup)$. The variables of the constraint system $\mathsf{L}[t]$ with $t \in \delta'_\Gamma$ can be seen as annotations to the transitions of the saturated automaton. Additionally we define a function $\pi_\mathsf{L} : S \times \mathsf{Conf} \times S \to 2^{\mathsf{Hedges}}$ that constructs a set of hedges for a configuration by reading the annotations from the automaton:

$$\begin{array}{lll} [\mathsf{empty}] & \pi_\mathsf{L}(s, \varepsilon, s') & = \begin{cases} \{\varepsilon_\sigma\} & \text{if } s \xrightarrow{\varepsilon}{}^*_{\delta'} s' \\ \emptyset & \text{else} \end{cases} \\ [\mathsf{control}] & \pi_\mathsf{L}(s, cp, s') & = \bigcup\{\pi_\mathsf{L}(s, c, \tilde{s})\varepsilon_\tau \mid s \xrightarrow{c}{}^*_{\delta'} \tilde{s} \xrightarrow{p}_{\delta'_P} \hat{s} \xrightarrow{\varepsilon}{}^*_{\delta'} s'\} \\ [\mathsf{stack}] & \pi_\mathsf{L}(s, c\gamma, s') & = \bigcup\{\pi_\mathsf{L}(s, c, \tilde{s}) \,;\, \mathsf{L}[(\tilde{s}, \gamma, \hat{s})] \mid s \xrightarrow{c}{}^*_{\delta'} \tilde{s} \xrightarrow{\gamma}_{\delta'_\Gamma} \hat{s} \xrightarrow{\varepsilon}{}^*_{\delta'} s'\} \end{array}$$

ε transitions do not contribute any information and are simply skipped. If a control state is encountered a new empty tree is added to the current hedges for the following new process stack. In case of a stack symbol, the trees which transform the stack symbol are appended to the current hedges. By appending the trees of the individual stack symbols, we get a single tree, that transforms the whole stack.

We construct a set of constraints in a similar way the saturation procedure adds transitions to the automaton. Here $r(\cdot) : 2^{\mathsf{Hedges}} \to 2^{\mathsf{Trees}}$ are operators generating new trees out of a root node labelled with $r \in \Delta$ and lists of subtrees from a given set:

$$\begin{array}{lll} [\mathsf{init}] & \mathsf{L}[t] \supseteq \{\varepsilon_\tau\} & \text{if } t \in \delta_\Gamma \\ [\mathsf{step}] & \mathsf{L}[(s_p, \gamma, s')] \supseteq r(\pi_\mathsf{L}(s, c, s')) & \text{if } r = p\gamma \hookrightarrow c \in \Delta, s \in S_c \text{ and } s \xrightarrow{c}{}^*_{\delta'} s' \end{array}$$

If we annotate the transitions of $\mathcal{A}^*$ with the least solution $\mathsf{lfp}(\mathsf{L})$ of L we can proof, by induction on the length of c, that the solution of the constraint system can be used to describe the set of all connecting hedges:

Theorem 3. $\mathsf{Hedges}(c, C) = \bigcup\{\pi_{\mathsf{lfp}(\mathsf{L})}(\dot{s}, c, s) \mid s \in F\}$.

To compute the weight of the hedges, we construct a constraint system $\mathsf{L}^\#$ and a function $\pi^\#_{\mathsf{L}^\#}$ over the weight domain by replacing the operators and constants in the constraint system L and the function π_L, with the corresponding operators and constants according to the abstraction function β. By standard results from abstract interpretation [7], we get $\mathsf{lfp}(\mathsf{L}^\#) = \beta(\mathsf{lfp}(\mathsf{L}))$ and $\pi^\#_{\mathsf{lfp}(\mathsf{L}^\#)} = \beta(\pi_{\mathsf{lfp}(\mathsf{L})})$ and with Theorem 3 we have:

Theorem 4. $\theta(c, C) = \bar{\bigoplus}\{\pi^\#_{\mathsf{lfp}(\mathsf{L}^\#)}(\dot{s}, c, s) \mid s \in F\}$.

Thus we can solve the BGPP problem by solving for $\mathsf{lfp}(\mathsf{L}^\#)$ using standard techniques and evaluating $\pi^\#_{\mathsf{lfp}(\mathsf{L}^\#)}$. Theorem 2 states, that we get the solution to the GPP problem by applying η.

6 Conclusion

We presented the GPP problem for a WDPN, which is a model for parallel programs with dynamic process creation and recursive procedures. The GPP problem is a general problem formulation, which can, for example, be used to capture basic dataflow analysis problems. Since the GPP problem can not be solved directly, our approach is based on an alternative branching semantics for DPN. The resulting tree shaped executions can be characterised using a constraint system, which can then be solved over an abstract domain to get a solution for the BGPP problem for BWDPN. If the weight domains for the BWDPN and WDPN are connected through an extension, the solution for the GPP problem can be derived from the corresponding BGPP problem. We have shown how the results can be used to solve basic dataflow analysis problems like bitvector analyses or shortest path problems.

References

1. Bouajjani, A., Müller-Olm, M., Touili, T.: Regular symbolic analysis of dynamic networks of pushdown systems. In: CONCUR. LNCS 3653, Springer (2005)
2. Reps, T., Schwoon, S., Jha, S., Melski, D.: Weighted pushdown systems and their application to interprocedural dataflow analysis. Sci. Comp. Prog. **58**(1-2) (2005)
3. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst. **22**(2) (2000)
4. Lammich, P., Müller-Olm, M., Wenner, A.: Predecessor sets of dynamic pushdown networks with tree-regular constraints. In: CAV. LNCS 5643, Springer (2009)
5. Lammich, P., Müller-Olm, M.: Precise fixpoint-based analysis of programs with thread-creation and procedures. In: CONCUR. LNCS 4703 (2007)
6. Lal, A., Touili, T., Kidd, N., Reps, T.W.: Interprocedural analysis of concurrent programs under a context bound. In: TACAS. LNCS 4963, Springer (2008)
7. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, ACM Press (1977)
8. Seidl, H., Steffen, B.: Constraint-based inter-procedural analysis of parallel programs. Nordic J. of Computing **7**(4) (2000)