Weighted Pushdown Systems and Weighted Transducers

Akash Lal[1], Tayssir Touili[2], Nicholas Kidd[1], and Thomas Reps[1]

[1] University of Wisconsin, Madison, Wisconsin, USA. {akash, kidd, reps}@cs.wisc.edu
[2] LIAFA, CNRS & University of Paris 7, Paris, France. touili@liafa.jussieu.fr

Abstract. Pushdown Systems (PDSs) are an important formalism for modeling programs. Reachability analysis on PDSs has been used extensively for program verification. A key result, which made PDSs popular in the model-checking community was that the set of reachable stack configurations starting from a regular set of configurations is also regular. A more general result was given by Caucal [7] who showed that a PDS's reachability relation, which maps stack configurations to their reachable set of stack configurations, can be encoded using a finite-state transducer. In this paper, we generalize the result to *weighted* pushdown systems, which have proven to be very useful for model checking as well as dataflow analysis. The same algorithm provides an efficient construction of transducers for ordinary (unweighted) PDSs. We also give a direct saturation algorithm for constructing transducers for single-state PDSs.

1 Introduction

Pushdown Systems (PDSs) are an important formalism for modeling programs [26, 25, 19, 16, 11, 1]. They are a generalization of finite-graph control models of programs because of their ability to simulate a program's control stack. The stack allows a PDS to describe faithfully paths with matched procedure calls and returns. Because a program stack can be of unbounded size, the PDS represents an infinite graph in which the nodes correspond to stack configurations of the PDS and the edges are valid transitions between configurations, as defined by the rules of the PDS.

In program verification, it is common to reduce safety-property verification to a problem of reachability in a graph (i.e., a program model); the program violates a particular property if a node t (target) is reachable from node s (source) in the graph. If there are multiple such source and target pairs, it may be better to precompute and summarize reachability information in the graph, rather than performing a separate reachability query for each pair. For finite-graph models, such a summarization can be performed by taking a transitive closure of the graph. Source-target reachability can then be answered in constant time. It is natural, therefore, to ask if such a summarization can also be done for the set of infinite graphs definable using PDSs. In 1992, Caucal [7] answered this question by showing that the reachability relation of the infinite graph defined by a PDS can be encoded using a finite-state transducer, such that a node (stack configuration) s can reach a node t if and only if the pair (s, t) is in the input-output relation of the transducer.

In this paper, we extend the above result to *weighted* pushdown systems (WPDSs). While PDSs are able to encode an infinite control abstraction (using the stack), they are not able to model an infinite data abstraction. (Unweighted

PDSs can only encode finite abstractions of data, such as predicate abstraction and Boolean programs [26].) WPDSs address this issue by associating a weight with each PDS rule [25, 5]. These weights can encode abstract transformers on an infinite lattice (with some additional properties), and, consequently, WPDSs can model a program more effectively than a PDS. Because of weights, our goal is no longer to compute reachability; instead, we need to compute meet-over-all-paths values over the PDS's transition relation.[3] Path problems on WPDSs can be solved by first creating a weighted transducer, and then applying a simple transducer-automaton composition operation (§3.1).

There has been considerable work on forward and backward reachability algorithms for PDSs and WPDSs [4, 26, 25, 17].[4] These algorithms are based on saturation procedures that add more transitions (possibly with weights) to an automaton (that represents a set of configurations) until a fixpoint is reached, at which point the resulting automaton represents the desired answer. When we started working on constructing transducers, our first attempt was to generalize these algorithms to work with transducers instead of automata. However, we only succeeded in extending the algorithm to single-state PDSs. That algorithm, given in §2.2, does not work with general PDSs or with WPDSs. Even though it is not a general algorithm, it highlights the difficulties in extending existing algorithms to work with transducers.

To obtain a general transducer construction for PDSs and WPDSs, we use a different principle that uses multiple calls (actually only 2) to existing saturation procedures to construct parts of the transducer, which are then combined to obtain the desired transducer. This algorithm is presented for WPDSs in §3.1. The techniques that we use in the algorithm resemble the ones used by Caucal [7] (see §4); however, we make use of recent developments in WPDS technology and also provide bounds on the complexity of our construction. The algorithm, when specialized to PDSs, provides an efficient construction of transducers for PDSs. We also show that single-source, single-target path problems on WPDSs (st-path problems for short) can be solved more efficiently using these transducers (after they are constructed) rather than using existing algorithms based on saturation.

The contributions of this paper can be summarized as follows:

- We present an algorithm for constructing a transducer that represents the reachability relation of a given PDS or WPDS. While an algorithm for constructing transducers for (unweighted) PDSs has been given before [7], no such transducer construction has been attempted before for WPDSs.
- We show how transducers can be used to solve st-path problems on WPDSs more efficiently than existing algorithms.
- Similar to current saturation algorithms for forward and backward reachability on PDSs, which deal with finite-state automata, we give a (direct) saturation procedure for constructing transducers for single-state PDSs. While this construction does not work for general PDSs or WPDSs, it illustrates the difficulty in extending existing algorithms to produce a transducer.

[3] Similarly, in finite weighted graphs, transitive closure generalizes to such problems as the all-pairs shortest-path problem and the meet-over-all-paths problem [15].

[4] See [3] for a summary of the history of reachability algorithms for PDSs.

The rest of the paper is organized as follows. In §2, we discuss PDSs. We give some background on how they can be used for modeling programs, and show how transducers can be used for solving reachability problems on PDSs. We also give a direct saturation-based algorithm for constructing a transducer for single-state PDSs. In §3, we discuss WPDSs. We give the weighted-transducer construction, and show how they can be used for efficiently solving *st*-path problems. §4 discusses related work. Short proofs of non-trivial lemmas and theorems are given in Appendix A.

2 Pushdown Systems

Definition 1. *A* **pushdown system** *is a triple $\mathcal{P} = (P, \Gamma, \Delta)$, where P is the set of states or control locations, Γ is the set of stack symbols, and $\Delta \subseteq P \times \Gamma \times P \times \Gamma^*$ is the set of pushdown rules. A* **configuration** *of $\mathcal{P}$ is a pair $\langle p, u \rangle$ where $p \in P$ and $u \in \Gamma^*$. A rule $r \in \Delta$ is written as $\langle p, \gamma \rangle \hookrightarrow \langle p', u \rangle$, where $p, p' \in P$, $\gamma \in \Gamma$ and $u \in \Gamma^*$. These rules define a transition relation $\Rightarrow$ on configurations of $\mathcal{P}$ as follows: If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', u \rangle$ then $\langle p, \gamma u' \rangle \Rightarrow \langle p', u u' \rangle$ for all $u' \in \Gamma^*$. The reflexive transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. For a set of configurations C, we define $pre^*(C) = \{c' \mid \exists c \in C : c' \Rightarrow^* c\}$ and $post^*(C) = \{c' \mid \exists c \in C : c \Rightarrow^* c'\}$, which are just backward and forward reachability under the transition relation $\Rightarrow$.*

We restrict the pushdown rules to have at most two stack symbols on the right-hand side. This restriction does not decrease the power of pushdown systems because by increasing the number of stack symbols by a constant factor, an arbitrary pushdown system can be converted into one that satisfies this restriction [12, 27, 26].

The standard approach for modeling program control flow with a pushdown system is as follows: P contains a single state $\{p\}$, Γ corresponds to program locations, and Δ corresponds to transitions in the interprocedural control-flow graph (ICFG) (see Fig. 1). The state space P can be expanded to encode a finite abstraction of the global state, and the stack space can be expanded to encode local variables [26].

Rule	Control flow modeled
$\langle p, u \rangle \hookrightarrow \langle p, v \rangle$	Intraprocedural CFG edge $u \rightarrow v$
$\langle p, c \rangle \hookrightarrow \langle p, g_{\text{enter}}\ r \rangle$	Call to g at c that returns to r
$\langle p, g_{\text{exit}} \rangle \hookrightarrow \langle p, \varepsilon \rangle$	Return from procedure g

Fig. 1. The encoding of ICFG edges as PDS rules.

Because the number of configurations of a pushdown system is unbounded, it is useful to use finite automata to describe regular sets of configurations.

Definition 2. *If $\mathcal{P} = (P, \Gamma, \Delta)$ is a pushdown system then a* **$\mathcal{P}$-automaton** *is a finite automaton $(Q, \Gamma, \rightarrow, P, F)$, where $Q \supseteq P$ is a finite set of states, $\rightarrow \subseteq Q \times \Gamma \times Q$ is the transition relation, P is the set of initial states, and F is the set of final states. We say that a configuration $\langle p, u \rangle$ is accepted by a $\mathcal{P}$-automaton if the automaton can accept u when it is started in the state p (written as $p \xrightarrow{u}{}^* q$, where $q \in F$). A set of configurations is called* **regular** *if*

some $\mathcal{P}$-automaton accepts it. Without loss of generality, we restrict $\mathcal{P}$-automata to not have any transitions leading to an initial state.

An important result is that for a regular set of configurations C, both $post^*(C)$ and $pre^*(C)$ are also regular sets of configurations [3, 26, 4, 9, 10]. The algorithms for computing $post^*$ and pre^*, called *poststar* and *prestar*, respectively, take a $\mathcal{P}$-automaton $\mathcal{A}$ as input, and if C is the set of configurations accepted by $\mathcal{A}$, they produce $\mathcal{P}$-automata $\mathcal{A}_{post^*}$ and $\mathcal{A}_{pre^*}$ that accept the set of configurations $post^*(C)$ and $pre^*(C)$. The algorithms *prestar* and *poststar* are often called *saturation procedures* because they involve adding additional transitions to the automaton for C as long as certain criteria are met.

2.1 Transducers for Expressing Reachability in Pushdown Systems

In this section, we define the *st*-reachability problem for PDSs and show how it can be solved using transducers. For this section, fix $\mathcal{P} = (P, \Gamma, \Delta)$ as a PDS.

Definition 3. *Given two configurations $s, t \in P \times \Gamma^*$, the st-**reachability** problem is to determine if $s \Rightarrow^* t$.*

This problem can be solved using *prestar* and *poststar* by checking if $t \in post^*(\{s\})$ or $s \in pre^*(\{t\})$. We show next that this problem can also be solved using transducers.

Definition 4. *A **finite-state transducer** τ is a tuple $(Q, \Sigma_i, \Sigma_o, \lambda, I, F)$, where Q is a finite set of states, Σ_i and Σ_o are input and output alphabets, $\lambda \subseteq Q \times (\Sigma_i \cup \{\varepsilon\}) \times (\Sigma_o \cup \{\varepsilon\}) \times Q$ is the transition relation, $I \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of final states. If $(q_1, a, b, q_2) \in \lambda$, written as $q_1 \xrightarrow{a/b} q_2$, we say that the transducer can go from state q_1 to q_2 on input a, and outputs the symbol b. Given a state $q \in I$, we say that the transducer can accept a string $c_i \in \Sigma_i^*$ with output $c_o \in \Sigma_o^*$ if there is a path from state q to a final state that takes input c_i and outputs c_o. The **language** of the transducer $\mathcal{L}(\tau)$ is defined as the following subset of $\Sigma_i^* \times \Sigma_o^*$: $\{(c_i, c_o) \mid$ the transducer can output string c_o when the input is $c_i\}$.*

Given a PDS $\mathcal{P}$, we want to construct a transducer $\tau_{\mathcal{P}} = (Q, \Gamma \cup P, \Gamma \cup P, \lambda, I, F)$, with a single initial state $q_i \in I$, such that $\tau_{\mathcal{P}}$ accepts input $(p_1\ u_1)$ with output $(p_2\ u_2)$ if and only if $\langle p_1, u_1\rangle \Rightarrow^* \langle p_2, u_2\rangle$. The transducer $\tau_{\mathcal{P}}$, in effect, captures $\Rightarrow^*$, the transitive closure of the PDS transition relation. For now, we assume that there exists a method for constructing $\tau_{\mathcal{P}}$ from $\mathcal{P}$. The actual algorithm is given later in §3.1 for WPDSs. Because WPDSs are a generalization of PDSs, that algorithm applies to PDSs as well. We can derive the following theorem from Thm. 3 given in §3.1.

Theorem 1. *Given a PDS $\mathcal{P}$, a transducer $\tau_{\mathcal{P}}$ can be constructed such that it accepts input $(p_1\ u_1)$ and outputs $(p_2\ u_2)$ if and only if $\langle p_1, u_1\rangle \Rightarrow^* \langle p_2, u_2\rangle$. Moreover, this transducer can be constructed in time $O(|P||\Delta|(|P||\Gamma| + |\Delta|))$ and has at most $|P|^2|\Gamma| + |P||\Delta|$ states.*

The *st*-reachability problem reduces to a membership query in the language of the transducer $\tau_{\mathcal{P}}$. The general reachability problems solved by *prestar* and *poststar* can also be solved using this transducer. First, we need to define the application of a transducer to an automaton. To simplify the discussion, and without loss of generality, we assume here that the input and output alphabets of the transducer are the same.

Definition 5. *Given a transducer τ that defines the language $\mathcal{L}(\tau) \subseteq \Sigma^* \times \Sigma^*$, and an automaton $\mathcal{A}$ that accepts the language $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$, the transducer-automaton application $\tau(\mathcal{A})$ is defined as an automaton that accepts the image of $\mathcal{L}(\mathcal{A})$ under $\mathcal{L}(\tau)$, i.e., the language $\{u \mid \exists u' \in \mathcal{L}(\mathcal{A}), (u', u) \in \mathcal{L}(\tau)\}$.*

This application can be computed in a manner similar to automaton intersection: For each transition $p \xrightarrow{a/b} q$ in τ and transition $p' \xrightarrow{a} q'$ in $\mathcal{A}$, add the transition $(p, p') \xrightarrow{b} (q, q')$ to $\tau(\mathcal{A})$. Thus, the application can be computed with at most a quadratic explosion in the state space. Let τ^{-1} be the same transducer as τ but with input and output labels on transitions swapped. The following corollary follows from the above definition and Thm. 1.

Corollary 1. *For a PDS $\mathcal{P}$, let $\tau_{\mathcal{P}}$ be the transducer as in Thm. 1. Then, given a configuration set C represented by an automaton $\mathcal{A}$, $post^*(C) = \mathcal{L}(\tau(\mathcal{A}))$ and $pre^*(C) = \mathcal{L}(\tau^{-1}(\mathcal{A}))$.*

Corollary 1 gives us a way of computing forward and backward reachable sets using just transducer-automaton applications.[5] This may not lead to faster algorithms than the previous saturation-based algorithms, but it provides a different approach to solving the same problem, and it may be possible to combine it with recent advances in automata-theoretic algorithms such as regular model checking [13, 28, 6, 2].

In the next section, we give a construction of a transducer for a single-state PDS. The construction is a direct saturation algorithm that applies to a transducer, unlike the construction given later for WPDSs, which uses two saturation steps to compute a transducer. The construction does not generalize to multiple-state PDSs (which are strictly more powerful than single-state PDSs [8]) or to WPDSs.

2.2 A transducer construction for single-state PDSs

Let $\mathcal{P}$ be a PDS with a single state p, i.e., $P = \{p\}$. Because we are restricting ourselves to single-state PDSs, the transducers we consider in this section do not include P in their input or output alphabet, i.e., they just describe relations on stacks.

Let $\tau = (Q, \Gamma, \Gamma, \lambda, I, F)$ be a transducer such that $Q = \{p\} \cup Q_1$, $F \subseteq Q_1$, $I = \{p\}$, and all transitions in λ have the form $(q_1, \gamma/\gamma, q_2), q_1, q_2 \in Q, \gamma \in \Gamma$.

[5] $\mathcal{A}$ needs to be a slightly modified $\mathcal{P}$-automaton. Its alphabet is extended to be states as well as stack symbols of the PDS, and it accepts a PDS state as its first symbol to initialize its starting state (as required by a $\mathcal{P}$-automaton). This is a very minor change in representation and we do not mention it explicitly in the rest of the paper.

(Such a transducer represents a subset of the identity relation.) Starting with τ, we create a transducer τ' such that $(c_1, c) \in \mathcal{L}(\tau')$ iff $(c_1, c_2) \in \mathcal{L}(\tau)$ and $c_2 \Rightarrow^* c$ in the transition relation of $\mathcal{P}$. If τ represents the identity relation on configurations, i.e., $\mathcal{L}(\tau) = \{(c, c) \mid c \in \Gamma^*\}$, then τ' will capture the reachability relation of $\mathcal{P}$ (same as $\tau_{\mathcal{P}}$ in the preceding section).

The idea behind the algorithm below is to consider τ' as having two parts: a first part where only transitions of the form γ/ε are allowed, and a second part where such transitions are not allowed. The transducer is constructed so that it can move from the first part to the second part, but not in the other direction (hence, γ/ε-transitions appear only in the beginning of an accepting run of τ').

Intuitively, the first part can be obtained by making a "kind of a copy" of the second part that rewrites letters into "ε" (when possible and when allowed by the rules of the PDS). To do so, for every state q in Q_1, we introduce a new state q' that is intended to satisfy the following condition: $(p, \gamma/\varepsilon, q')$ is a transition in τ' iff $(p, \gamma/\gamma, q)$ is a transition in τ and $\langle p, \gamma\rangle \Rightarrow^* \langle p, \varepsilon\rangle$.

We create the transducer $\tau' = (\mathcal{Q}, \Gamma, \Gamma, \lambda', I', F')$ from τ as follows:

- $F' = F$ and $I' = I$.
- Let $Q_2 = \{s_{(q,\gamma,\gamma')} \mid q \in Q_1 \cup \{p\}, \gamma, \gamma' \in \Gamma,\}$. Then $\mathcal{Q} = Q \cup Q_2 \cup \{q' \mid q \in Q_1 \cup Q_2\}$. The states in Q_2 correspond to the extra states added by the *poststar* saturation algorithm.
- λ' is defined as follows, where $q'_j \overset{\gamma/\gamma'}{\rightsquigarrow} q_i$ denotes $q'_j \xrightarrow{\varepsilon/\varepsilon} q_j \xrightarrow{\gamma/\gamma'} q_i$ or $q'_j \xrightarrow{\gamma/\gamma'} q_i$.
  - (α_1) $\lambda \subseteq \lambda'$;
  - (α_2) for every $q \in Q_1 \cup Q_2$, $(q', \varepsilon/\varepsilon, q) \in \lambda'$;
  - (α_3) for every rule $\langle p, \gamma\rangle \hookrightarrow \langle p, \varepsilon\rangle$, if λ' contains a transition of the form $(p, \gamma_1/\gamma, q)$, then add $(p, \gamma_1/\varepsilon, q')$ to λ'. Moreover, if λ' has a path of the following form:
    $$p \xrightarrow{\gamma_1/\varepsilon} q'_1 \xrightarrow{\gamma_2/\varepsilon} q'_2 \xrightarrow{\gamma_3/\varepsilon} \cdots \xrightarrow{\gamma_i/\varepsilon} q'_i \overset{\gamma_{i+1}/\gamma}{\rightsquigarrow} q_{i+1}$$
    then add the transition $(q'_i, \gamma_{i+1}/\varepsilon, q'_{i+1})$ to λ'.
  - (α_4) for every rule $\langle p, \gamma\rangle \hookrightarrow \langle p, \gamma'\rangle$, if λ' contains a transition of the form $(p, \gamma_1/\gamma, q)$, then add $(p, \gamma_1/\gamma', q)$ to λ'. Moreover, if λ' has a path of the following form:
    $$p \xrightarrow{\gamma_1/\varepsilon} q'_1 \xrightarrow{\gamma_2/\varepsilon} q'_2 \xrightarrow{\gamma_3/\varepsilon} \cdots \xrightarrow{\gamma_i/\varepsilon} q'_i \overset{\gamma_{i+1}/\gamma}{\rightsquigarrow} q_{i+1}$$
    then add the transition $(q'_i, \gamma_{i+1}/\gamma', q_{i+1})$ to λ'.
  - (α_5) for every rule $\langle p, \gamma\rangle \hookrightarrow \langle p, \gamma'\gamma''\rangle$, if λ' contains a transition of the form $(p, \gamma_1/\gamma, q)$, then add $(p, \gamma_1/\gamma', s_{(p,\gamma_1,\gamma')})$ and $(s_{(p,\gamma_1,\gamma')}, \varepsilon/\gamma'', q)$ to λ'. Moreover, if λ' has a path of the following form:
    $$p \xrightarrow{\gamma_1/\varepsilon} q'_1 \xrightarrow{\gamma_2/\varepsilon} q'_2 \xrightarrow{\gamma_3/\varepsilon} \cdots \xrightarrow{\gamma_i/\varepsilon} q'_i \overset{\gamma_{i+1}/\gamma}{\rightsquigarrow} q_{i+1}$$
    then add $(q'_i, \gamma_{i+1}/\gamma', s_{(q_i,\gamma_{i+1},\gamma')})$ and $(s_{(q_i,\gamma_{i+1},\gamma')}, \varepsilon/\gamma'', q_{i+1})$ to λ'.

The rules above are extensions for the standard saturation procedures for PDSs to transducers. For example, if $p \xrightarrow{\gamma_1 \cdots \gamma_{i+1}/\gamma} q$ in λ', then we should also have the path $p \xrightarrow{\gamma_1 \cdots \gamma_{i+1}/\epsilon} q$ in λ' when there is a PDS rule $\langle p, \gamma\rangle \hookrightarrow \langle p, \varepsilon\rangle$. This is ensured by rules (α_2) and (α_3).

The definition of λ' is inductive and can be computed as the limit of a finite sequence of increasing sets of transitions $\lambda'_1 \subset \lambda'_2 \subset \ldots \subset \lambda'_n$, where each λ'_{i+1} contains at most two transitions more than λ'_i. Termination is guaranteed because there is a bounded number of states in τ', hence λ' can only have a bounded number of transitions. Proofs of the following lemmas are given in Appendix A.

Lemma 1. *Let $\langle p, v\rangle$ be a configuration such that $p \xrightarrow{v/v}_{\lambda} q$ for some final state $q \in F$. If $\langle p, v\rangle \Rightarrow^* \langle p, w\rangle$, then $p \xrightarrow{v/w}_{\lambda'} q$.*

Lemma 2. *Let $p \xrightarrow{v/w}_{\lambda'} q$, where $q \in Q_1$, then $p \xrightarrow{v/v}_{\lambda} q$, and $\langle p, v\rangle \Rightarrow^* \langle p, w\rangle$.*

Theorem 2. *Given a transducer τ whose language is the identity relation on Γ^*, and a single-state PDS $\mathcal{P}$, the transducer $\tau_{\mathcal{P}}$ can be constructed using the steps given above, such that $\mathcal{L}(\tau_{\mathcal{P}}) = \{(c_1, c_2) \mid c_1 \Rightarrow^* c_2\}$. Moreover, $\tau_{\mathcal{P}}$ has at most $|Q||\Gamma|^2$ states and $|Q||\Gamma|^6$ transitions, where Q is the set of states of τ. The running time of this construction is $O(|Q||\Gamma|^6)$.*

The construction of transducers for WPDSs, given in the next section, when specialized to (multiple-state) PDSs gives a more efficient algorithm than the one presented above. However, the above algorithm shows that it is non-trivial to extend existing saturation algorithms to transducers. The key difficulty is that each of the saturation rules ($\alpha_3 - \alpha_5$) need to look at *paths* in the transducer, unlike saturation algorithms on automata that just look at one or two transitions leaving from an initial state. This arises due to the fact that ε-closure on automata can be performed easily, but ε-closure on transducers (i.e., removing transitions of the form (γ/ε) or (ε/γ)) may not be possible at all [20].

The next section uses a different principle to construct transducers for WPDSs. It calls two saturation-based procedures to construct different parts of the transducer.

3 Weighted Pushdown Systems

A weighted pushdown system is obtained by augmenting a pushdown system with a weight domain that is a bounded idempotent semiring [24, 5]. Such semirings are powerful enough to encode finite-state data abstractions such as the one required for Boolean program verification, as well as infinite-state data abstractions, such as copy-constant propagation and affine-relation analysis [23, 19]. A small survey of safety-property and Boolean program verification using PDSs and their respective encoding as a WPDS is given in [18]. The basic idea is to use weights to encode the effect that each rule has on the data state of the program.

Definition 6. *A* **bounded idempotent semiring** *is a quintuple $(D, \oplus, \otimes, \overline{0}, \overline{1})$, where D is a set whose elements are called* **weights**, *$\overline{0}$ and $\overline{1}$ are elements of D, and $\oplus$ (the combine operation) and $\otimes$ (the extend operation) are binary operators on D such that*

1. $(D, \oplus)$ *is a commutative monoid with* $\overline{0}$ *as its neutral element, and where* $\oplus$ *is idempotent.* $(D, \otimes)$ *is a monoid with the neutral element* $\overline{1}$.
2. $\otimes$ *distributes over* $\oplus$*, i.e., for all* $a, b, c \in D$ *we have*
$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \text{ and } (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)\,.$$
3. $\overline{0}$ *is an annihilator with respect to* $\otimes$*, i.e., for all* $a \in D$*,* $a \otimes \overline{0} = \overline{0} = \overline{0} \otimes a$.
4. *In the partial order* $\sqsubseteq$ *defined by* $\forall a, b \in D,\ a \sqsubseteq b$ *iff* $a \oplus b = a$*, there are no infinite descending chains.*

The *height* of a weight domain is defined to be the length of the longest descending chain in the domain.

Definition 7. *A* **weighted pushdown system** *is a triple* $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$ *where* $\mathcal{P} = (P, \Gamma, \Delta)$ *is a pushdown system,* $\mathcal{S} = (D, \oplus, \otimes, \overline{0}, \overline{1})$ *is a bounded idempotent semiring and* $f : \Delta \to D$ *is a map that assigns a weight to each rule of* $\mathcal{P}$.

Let $\sigma \in \Delta^*$ be a sequence of rules. Using f, we can associate a value to σ, i.e., if $\sigma = [r_1, \ldots, r_k]$, then we define $v(\sigma) \stackrel{\text{def}}{=} f(r_1) \otimes \ldots \otimes f(r_k)$. Moreover, for any two configurations c and c' of $\mathcal{P}$, we use $path(c, c')$ to denote the set of all rule sequences that transform c into c'. If $\sigma \in path(c, c')$, then we say $c \Rightarrow^{\sigma} c'$. Reachability problems on PDSs are generalized to WPDSs as follows:

Definition 8. *Let* $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$ *be a weighted pushdown system, where* $\mathcal{P} = (P, \Gamma, \Delta)$*, and let* $S, T \subseteq P \times \Gamma^*$ *be regular sets of configurations. Then the* ***meet-over-all-paths*** *value* $MOP(S, T)$ *is defined as* $\bigoplus\{v(\sigma) \mid s \Rightarrow^{\sigma} t, s \in S, t \in T\}$.

A PDS is simply a WPDS with the Boolean weight domain $(\{\overline{0}, \overline{1}\}, \oplus, \otimes, \overline{0}, \overline{1})$ and weight assignment $f(r) = \overline{1}$ for all rules $r \in \Delta$. In this case, $\text{MOP}(S, T) = \overline{1}$ iff there is a path from a configuration in S to a configuration in T.

An important weight domain for WPDSs is the set of all binary relations on a finite set. If G is a finite set of atoms, then $(2^{G \times G}, \cup, \circ, \emptyset, id)$ is a valid weight domain: weights are binary relations on G, combine is union (of sets of pairs), extend is relational composition, $\overline{0}$ is the empty relation, and $\overline{1}$ is the identity relation on G. This weight domain is useful for encoding Boolean programs (programs with only Boolean variables) as WPDSs. The set G can be instantiated to be the set of all valuations of Boolean variables, and the weight associated with a PDS rule is the effect of executing the corresponding ICFG edge on the program variables. Such a weight domain is used in the tool MOPED [26, 14] (it includes local variables as well). Assertion checking in Boolean programs can then be performed by checking if a configuration set T (of all assertions) is reachable with non-$\overline{0}$ weight, i.e., $\text{MOP}(S, T) \neq \overline{0}$, where S is the starting set of configurations for the program. More details on the uses of PDSs for model checking, and their encoding as WPDSs can be found in [25, 18].

A WPDS with a weight domain that has a finite number of weights, such as the one described above, can be encoded as a PDS. However, WPDSs can also deal with infinite weight domains, such as the one for affine-relation analysis [23, 19]. Moreover, even for finite weight domains, it is often useful to use weights because they can be symbolically encoded. The tool MOPED is able scale to a

large number of variables because it uses BDDs to encode the binary relations represented by weights.

There are two algorithms for solving for MOP values, called *prestar* and *poststar* (by analogy with the reachability algorithms for PDSs). They take as input an automaton that accepts the set of initial configurations. As output, they produce a *weighted automaton* defined as follows.

Definition 9. *Given a weighted pushdown system $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, a* ***$\mathcal{W}$-automaton*** *$\mathcal{A}$ is a $\mathcal{P}$-automaton, where each transition in the automaton is labeled with a weight. The weight of a path in the automaton is obtained by taking an extend of the weights on the transitions in the path in either a forward or backward direction, depending on the context in which the automaton is used. The automaton is said to accept a configuration $c = \langle p, u \rangle$ with weight $w = \mathcal{A}(c)$ if w is the combine of weights of all accepting paths for u starting from state p in $\mathcal{A}$. We call the automaton a* ***backward $\mathcal{W}$-automaton*** *if the weight of a path is read backwards, and a* ***forward $\mathcal{W}$-automaton*** *otherwise.*

Let $\mathcal{A}$ be an unweighted automaton and $\mathcal{L}(\mathcal{A})$ be the set of configurations accepted by it. Then, *prestar*$(\mathcal{A})$ produces a forward weighted automaton $\mathcal{A}_{pre^*}$ as output such that $\mathcal{A}_{pre^*}(c) = \text{MOP}(\{c\}, \mathcal{L}(\mathcal{A}))$, whereas *poststar*$(\mathcal{A})$ produces a backward weighted automaton $\mathcal{A}_{post^*}$ as output such that $\mathcal{A}_{post^*}(c) = \text{MOP}(\mathcal{L}(\mathcal{A}), \{c\})$ [25]. Using standard automata-theoretic techniques, we can also compute $\mathcal{A}_w(C)$ for (forward or backward) weighted automaton $\mathcal{A}_w$ and a regular set of configurations C, where $\mathcal{A}_w(C) = \bigoplus\{\mathcal{A}_w(c) \mid c \in C\}$. This allows one to solve for the meet-over-all-paths value $\text{MOP}(S, T)$ for configuration sets S and T, using either *poststar* or *prestar*.

The following lemma states the complexity for solving *poststar* by the algorithm of Reps et al. [25], which we use later for comparison against algorithms presented in this paper. We use the notation $O_s(g)$ to denote the time bound $O(g\mathcal{S}_t)$, where $\mathcal{S}_t$ is an upper bound on the time taken by a semiring operation.

Lemma 3. *[25] Given a WPDS with PDS $\mathcal{P} = (P, \Gamma, \Delta)$, if $\mathcal{A} = (Q, \Gamma, \rightarrow, P, F)$ is the $\mathcal{P}$-automaton accepting an input set of configuration, poststar produces a backward weighted automaton with at most $|Q| + |\Delta|$ states in time $O_s(|P||\Delta|(|Q_0| + |\Delta|)H + |P||\lambda_0|H)$, where $Q_0 = Q \backslash P$, $\lambda_0 \subseteq \rightarrow$ is the set of all transitions leading from states in Q_0, and H is the height of the weight domain.*

For the rest of this section, we fix $\mathcal{P} = (P, \Gamma, \Delta)$ to be a PDS, $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$ to be a WPDS, and H to be the height of $\mathcal{S}$.

3.1 Transducers for Solving Path Problems in WPDSs

For WPDSs, we want to create a weighted transducer that will accept input $(p_1\ u_1)$ and output $(p_2\ u_2)$ with weight w if and only if w is the combine of values of all paths in the WPDS that take $s = \langle p_1, u_1 \rangle$ to $t = \langle p_2, u_2 \rangle$, i.e., $w = \text{MOP}(\{s\}, \{t\})$. We defer the definition of a weighted transducer to a little later in this section (Defn. 10).

To set about solving *st*-path problems, we first make an observation about paths in a PDS's transition relation. Suppose that $\langle p, \gamma_1\gamma_2 \cdots \gamma_n \rangle$ is a configuration of a PDS $\mathcal{P}$. Then any path in the transition relation described by $\mathcal{P}$,

starting from this configuration, can be written as shown in Fig. 2. The figure shows that the path starts initially by *popping off* some stack symbols (k symbols in the figure) in possibly multiple steps, after which it does not touch the rest of the stack ($\gamma_{k+2} \cdots \gamma_n$), except for the top symbol (γ_{k+1}). Note that it is also possible for the path to merely pop off all stack symbols ($k = n$) and stop because no PDS rule can fire on an empty stack.

$$\begin{aligned}\langle p, \gamma_1\gamma_2 \cdots \gamma_n\rangle &\Rightarrow^* \langle p_1, \gamma_2 \cdots \gamma_n\rangle \\ &\Rightarrow^* \langle p_2, \gamma_3 \cdots \gamma_n\rangle \\ &\Rightarrow^* \cdots \\ &\Rightarrow^* \langle p_k, \gamma_{k+1} \cdots \gamma_n\rangle \\ &\Rightarrow^* \langle p_{k+1}, u\gamma_{k+2} \cdots \gamma_n\rangle\end{aligned}$$

Fig. 2. A path in the PDS's transition relation. Here, $k + 2 \leq n$ and $u \in \Gamma^*$.

To make this observation more formal, we decompose a path into phases as follows:

1. **Pop-phase.** The path pops off the top stack symbol without looking at the rest of the stack, i.e., it follows a sequence of rules that takes the configuration $\langle p, \gamma u\rangle$ to $\langle p', u\rangle$, for any $u \in \Gamma^*$.

2. **Growth-phase.** The path only looks at the top of the stack, and possibly rewrites it, but does not pop it off, i.e., it follows a sequence of rules that takes a configuration $\langle p, \gamma u\rangle$ to $\langle p', u'u\rangle$ with $u' \in \Gamma^+$, for any $u \in \Gamma^*$.

Each path in the PDS's transition relation has zero or more pop-phases followed by a single growth-phase. We construct the transducer for *st*-reachability by essentially *pre-computing* each of these phases. Let $\mathcal{W}$ be a WPDS as defined in Defn. 8. First, we define two procedures:

1. $pop : P \times \Gamma \times P \to D$ is defined as follows:
$$pop(p, \gamma, p') = \bigoplus\{v(\sigma) \mid \langle p, \gamma\rangle \Rightarrow^\sigma \langle p', \varepsilon\rangle\}$$
2. $grow : P \times \Gamma \to ((P \times \Gamma^+) \to D)$ is defined as follows:
$$grow(p, \gamma)(p', u) = \bigoplus\{v(\sigma) \mid \langle p, \gamma\rangle \Rightarrow^\sigma \langle p', u\rangle\}$$
A simple but inefficient way of computing this function is to solve $poststar(\langle p, \gamma\rangle)$ to produce a weighted automaton that represents the function on the right-hand side.

The following Lemmas give efficient algorithms for computing the above procedures. Proofs are given in Appendix A.

Lemma 4. *Let $\mathcal{A} = (P, \Gamma, \emptyset, P, P)$ be a $\mathcal{P}$-automaton that represents the set of configurations $C = \{\langle p, \varepsilon\rangle \mid p \in P\}$. Let $\mathcal{A}_{pop}$ be the forward weighted-automaton obtained by running prestar on $\mathcal{A}$. Then $pop(p, \gamma, p')$ is the weight on the transition (p, γ, p') in $\mathcal{A}_{pop}$. We can generate $\mathcal{A}_{pop}$ in time $O_s(|P|^2|\Delta|H)$, and it has at most $|P|$ states.*

Lemma 5. *Let $\mathcal{A}_F = (Q, \Gamma, \to, P, F)$ be a $\mathcal{P}$-automaton, where $Q = P \cup \{q_{p,\gamma} \mid p \in P, \gamma \in \Gamma\}$ and $p \xrightarrow{\gamma} q_{p,\gamma}$ for each $p \in P, \gamma \in \Gamma$. Then $\mathcal{A}_{\{q_{p,\gamma}\}}$ represents the configuration $\langle p, \gamma\rangle$. Let $\mathcal{A}$ be this automaton where we leave the set of final states undefined. Let $\mathcal{A}_{grow}$ be the backward weighted-automaton obtained from*

running poststar on $\mathcal{A}$. If we restrict the final states in $\mathcal{A}_{grow}$ to be just $q_{p,\gamma}$ (and remove all states that do not have an accepting path to the final state), we obtain a backward weighted-automaton that represents grow(p,γ), i.e., when this automaton is started in state p' and accepts input u with weight w, then $w = grow(p,\gamma)(p',u)$. We call this automaton $\mathcal{A}_{p,\gamma}$. We can compute $\mathcal{A}_{grow}$ in time $O_s(|P||\Delta|(|P||\Gamma| + |\Delta|)H)$, and it has at most $|P||\Gamma| + |\Delta|$ states.

The advantage of the construction presented in Lemma 5 is that it just requires a single *poststar* query to compute all the $\mathcal{A}_{p,\gamma}$, instead of one query for each $p \in P$ and $\gamma \in \Gamma$. Because the standard *poststar* automaton construction builds an automaton that is larger than the input automaton (Lemma 3), $\mathcal{A}_{grow}$ has many fewer states than those in all the $\mathcal{A}_{p,\gamma}$ put together.

Fig. 3 shows these automata for a simple WPDS constructed over the *minpath* semiring. This semiring is defined as $(\mathbb{N}\cup\{\infty\}, \min, +, \infty, 0)$. If all rules are given the weight 1 (different from the semiring weight $\overline{1}$, which is the numeric value 0), then the MOP weight between two configurations is the length of the shortest path between them.

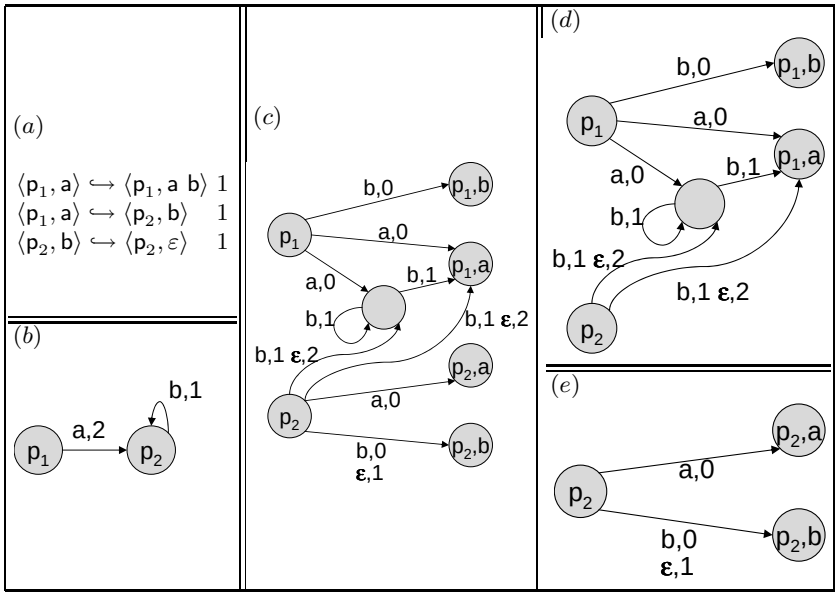

Fig. 3. (*a*) A simple WPDS with the minpath semiring. (*b*) The $\mathcal{A}_{pop}$ automaton. Edges are labeled with their stack symbol and weight. (*c*) The $\mathcal{A}_{grow}$ automaton. (*d*) The $\mathcal{A}_{p_1}$ automaton obtained from $\mathcal{A}_{grow}$. (*e*) The $\mathcal{A}_{p_2}$ automaton obtained from $\mathcal{A}_{grow}$. The unnamed state in (*c*) and (*d*) is an extra state added by the *post** algorithm used in Lemma 5.

The idea behind our approach is to use $\mathcal{A}_{pop}$ to simulate the first phase where the PDS pops off stack symbols. When the transducer (non-deterministically) decides to switch over to the growth phase, and is in state p in $\mathcal{A}_{pop}$ with γ being the next symbol in the input, it passes control to $\mathcal{A}_{p,\gamma}$ to start generating the output. Then it moves into an accept phase where it copies the untouched part of the input stack to the output.

We use $\mathcal{A}_{grow}$ to avoid introducing a separate copy of $\mathcal{A}_{p,\gamma}$ for each γ. Let $\mathcal{A}_p$ be the same as $\mathcal{A}_{grow}$, but with final states restricted to $\{q_{p,\gamma} \mid \gamma \in \Gamma\}$, and unreachable states appropriately pruned. In essence, the transducer will non-deterministically guess the stack symbol γ, pass control to $\mathcal{A}_p$, and then verify that the guess was correct when it reaches the final state $q_{p,\gamma}$ in $\mathcal{A}_p$. As a result, we just need $|P|$ copies of $\mathcal{A}_{grow}$. Note that $\mathcal{A}_{pop}$ is a forward weighted-automaton, whereas $\mathcal{A}_{grow}$ is a backward weighted-automaton. Therefore, when we mix them together in the same transducer, we allow the transducer to switch directions for computing the weight of a path. This is necessary, because going back to Fig. 2, a PDS rule sequence consumes the input configuration from left to right (in the pop phase), but produces the output stack configuration u from right to left (as it pushes symbols on the stack). Because we want the transducer to produce output from left to right, we need to switch directions for computing the weight of a path. For this, we define partitioned transducers.

Definition 10. *A **partitioned weighted finite-state transducer** τ is a tuple $(Q, \{Q_i\}_{i=1}^n, \mathcal{S}, \Sigma_i, \Sigma_o, \lambda, I, F)$ where Q is a finite set of states, $\{Q_i\}_{i=1}^n$ is a partition of Q, $\mathcal{S} = (D, \oplus, \otimes, \overline{0}, \overline{1})$ is a bounded idempotent semiring, Σ_i and Σ_o are input and output alphabets, $\lambda \subseteq Q \times D \times (\Sigma_i \cup \{\varepsilon\}) \times (\Sigma_o \cup \{\varepsilon\}) \times Q$ is the transition relation, $I \subseteq Q_1$ is the set of initial states, and $F \subseteq Q_n$ is the set of final states. We impose the following restriction on the transition relation: if $(q_k, w, a, b, q_l) \in \lambda$ and $q_k \in Q_k$ and $q_l \in Q_l$ then either $l = k$ or $l = k + 1$ and $w = \overline{1}$. Given a state $q \in Q_1$, we say that the transducer can accept a string $\sigma_i \in \Sigma_i^*$ with output $\sigma_o \in \Sigma_o^*$ if there is a path from state q to a final state that takes input σ_i and outputs σ_o.*

Computing the weight of a path requires more care. For a path η that goes through states $q_1, \cdots, q_m$, such that the weight of the i^{th} transition is w_i, and all states q_i are in Q_j for some j, then the weight of this path $v(\eta)$ is $w_1 \otimes w_2 \otimes \cdots \otimes w_m$ if j is odd and $w_m \otimes w_{m-1} \otimes \cdots \otimes w_1$ if j is even, i.e., the state-space partition determines the direction in which we perform extend. For a path η that crosses multiple partitions, i.e., $\eta = \eta_i \eta_{i+1} \cdots \eta_m$ such that each η_j is a path entirely inside Q_j, then $v(\eta) = v(\eta_i) \otimes v(\eta_{i+1}) \otimes \cdots \otimes v(\eta_m)$. We restrict a weighted finite-state transducer to have final states only in the last partition set Q_n and initial states only in partition set Q_1.

In this paper, we refer to partitioned weighted transducers as weighted transducers, or simply transducers when there is no possibility of confusion. Note that when the extend operator is commutative, as in the case of the Boolean semiring used for encoding PDSs as WPDSs, the partitioning is unnecessary.

Let $St(\mathcal{A})$ denote the set of states of an automaton $\mathcal{A}$. Because each of $\mathcal{A}_{pop}$ and $\mathcal{A}_p$ have P as a subset of their set of states, we distinguish them by referring to a state $q \in St(\mathcal{A}_{pop})$ by q_{pop} and $q \in St(\mathcal{A}_p)$ by q_p.

Given a WPDS $\mathcal{W}$, we construct a weighted transducer $\tau_{\mathcal{W}}$ using the steps given below. $\tau_{\mathcal{W}}$ has states $\{q_i, q_f\} \cup St(\mathcal{A}_{pop}) \cup (\bigcup_{p \in P} St(\mathcal{A}_p))$, input alphabet $P \cup \Gamma$, output alphabet $P \cup \Gamma$, weight domain D, initial state q_i, and final state q_f. Given a pair of configurations (c_1, c_2), $\tau_{\mathcal{W}}$ operates on them as shown in Fig. 4 (for simplicity, only the stack of the configurations is shown): it first consumes

a prefix x of c_1 (using $\mathcal{A}_{pop}$; Step 2 below), then produces a prefix y of c_2 (using $\mathcal{A}_{grow}$; Steps 3 to 6 below), and then copies the suffix z of c_1 over to the output (Step 7 below).

1. For each state $p \in P$, add the transition $(q_i, p/\varepsilon, p_{pop})$ with weight $\overline{1}$ to $\tau_{\mathcal{W}}$.
2. For each transition $(p^1_{pop}, \gamma, p^2_{pop})$ with weight w in $\mathcal{A}_{pop}$ add the transition $(p^1_{pop}, (\gamma/\varepsilon), p^2_{pop})$ with the same weight to $\tau_{\mathcal{W}}$, i.e., copy over $\mathcal{A}_{pop}$.
3. For each transition (q_p, γ', q'_p) in each automaton $\mathcal{A}_p$ add the transition $(q_p, (\varepsilon/\gamma'), q'_p)$ with the same weight to $\tau_{\mathcal{W}}$, i.e., copy over each of the $\mathcal{A}_p$.
4. For each $p, p' \in P$, add the transition $(p_{pop}, (\varepsilon/p'), p'_p)$ with weight $\overline{1}$ to $\tau_{\mathcal{W}}$. This transition permits a switch from the pop phase to the growth phase. At this point, we just know that the growth phase begins in state p and ends in state p'. We guess the stack symbol from which the growth phase starts. The next step verifies that our guess was correct.
5. For each final state $q_{p,\gamma} \in St(\mathcal{A}_p)$, add the transition $(q_{p,\gamma}, (\gamma/\varepsilon), q_f)$ with weight $\overline{1}$ to $\tau_{\mathcal{W}}$. This transition verifies that γ was on the input tape, and we just computed the growth phase starting from γ.
6. For each $p, q \in P$, add the transition $(q_p, (\varepsilon/\varepsilon), q_f)$ with weight $\overline{1}$ to $\tau_{\mathcal{W}}$. This transition allows us to skip the growth phase ($\mathcal{A}_p$).
7. For each $\gamma \in \Gamma$, add the transition $(q_f, (\gamma/\gamma), q_f)$ with weight $\overline{1}$ to $\tau_{\mathcal{W}}$. This part of the transducer copies over the untouched part of the input tape to the output tape.

The state-partition of $\tau_{\mathcal{W}}$ is $Q_1 = \{q_i\} \cup St(\mathcal{A}_{pop})$ and $Q_2 = \{q_f\} \cup (\bigcup_{p \in P} St(\mathcal{A}_p))$. The transducer for the WPDS given in Fig. 3(a) is shown in Fig. 5.

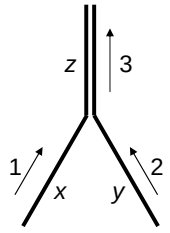

Fig. 4. The three phases of operation of $\tau_{\mathcal{W}}$ when given input-output pair (xz, yz).

Theorem 3. *When the transducer $\tau_{\mathcal{W}}$, as constructed above, is given input $(p\ u)$, $p \in P, u \in \Gamma^*$, then the combine over the values of all paths in $\tau_{\mathcal{W}}$ that output the string $(p'\ u')$ is precisely MOP$(\{\langle p, u\rangle\}, \{\langle p', u'\rangle\})$. Moreover, this transducer can be constructed in time $O_s(|P||\Delta|(|P||\Gamma| + |\Delta|)H)$, has at most $|P|^2|\Gamma| + |P||\Delta|$ states and at most $|P|^2|\Delta|^2$ transitions.*

Usually the WPDSs derived from programs have $|P| = 1$ and $|\Gamma| < |\Delta|$. In that case, constructing a transducer has similar complexity and size as running a single *poststar* query. A short proof of Thm. 3 is given in Appendix A.

After constructing the transducer, forward and backward reachability can be solved using transducer-automaton application as was done in the unweighted case. The difference is that weight labels need to be preserved while performing the application. This gives us weighted automata that are partitioned, like the transducer. However, these can still be used for computing MOP values as required.

Given an (unweighted) automaton $\mathcal{A}$ that represents a set of configurations, we can compute *poststar*$(\mathcal{A})$ by computing $\tau_{\mathcal{W}}(\mathcal{A})$, which requires time $O(|P|^2|\Delta|^2|\mathcal{A}|)$, where $|\mathcal{A}|$ is the number of transitions in $\mathcal{A}$. Note that this does

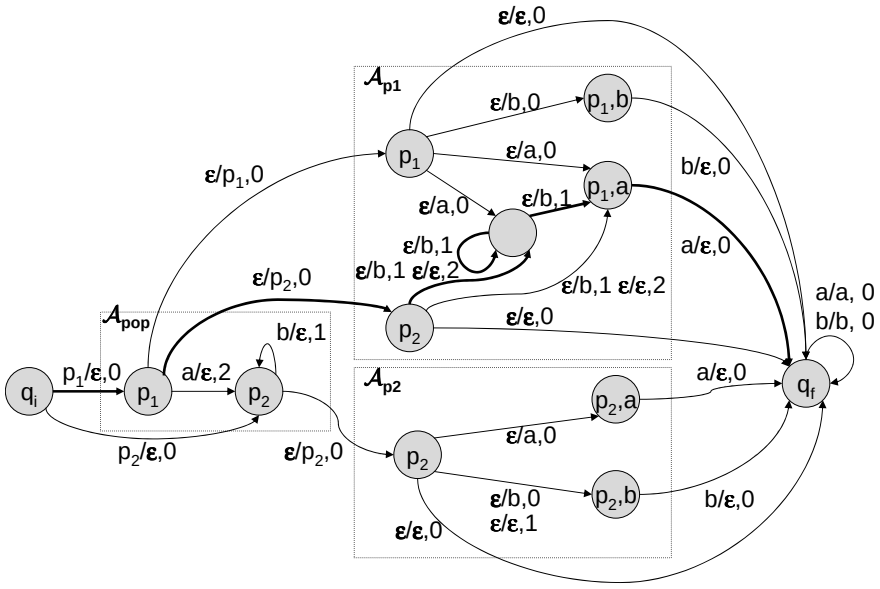

Fig. 5. The weighted transducer for the WPDS shown in Fig. 3(a). The boxes represent "copies" of $\mathcal{A}_{pop}$, $\mathcal{A}_{p_1}$ and $\mathcal{A}_{p_2}$ as required by steps 2 and 3. The transducer paths that accept input (p_1 a) and output (p_2 b^n), for $n \geq 2$, with weight n are highlighted in bold.

not require any semiring operations, and is independent of H. This, along with the fact that a transducer-automaton application is a very simple algorithm, might imply that *poststar* can be solved faster using transducers. However, the disadvantage of this approach is that the weighted automaton obtained as a result of the transducer-automaton application can be larger than the one obtained from previous saturation algorithms. Consequently, using the transducer in applications may not be advantageous. We leave this study as future work. The next section shows that the transducer can be used to efficiently solve *st*-path problems, which are restricted to have a single source and a single target configuration.

3.2 Solving *st*-path problems

Let s and t be two configurations of $\mathcal{W}$. In this section, we show that MOP($\{s\}, \{t\}$) can be solved efficiently using the transducer $\tau_{\mathcal{W}}$ constructed in the previous section. Using Thm. 3, we know that all we need to do is to compute the combine over the values of all paths in $\tau_{\mathcal{W}}$ that take s as input and produce t as output (call this weight $\tau_{\mathcal{W}}(s,t)$). This is carried out in much the same way as weights are read from weighted automaton (which is based on a standard NFA simulation algorithm).

Let $s = \langle p_s, u_s \rangle$ and $t = \langle p_t, u_t \rangle$. First, we look back at the intuition given in Fig. 2. Let u be a common suffix of u_s and u_t, such that $u_s = u_1\ \gamma\ u$ and $u_t = u_2\ u$. Then all paths from s to t will either be of length 0 (if $s = t$) or be of the form $\langle p_s, u_1\ \gamma\ u \rangle \Rightarrow^* \langle p, \gamma\ u \rangle \Rightarrow^* \langle p_t, u_2\ u \rangle$, where the first part is

composed of rules σ_1 such that $\langle p_s, u_1\rangle \Rightarrow^{\sigma_1} \langle p, \varepsilon\rangle$ (pop-phase) and the second part is composed of rules σ_2 such that $\langle p, \gamma\rangle \Rightarrow^{\sigma_2} \langle p_t, u_2\rangle$ (growth-phase). For each u as above, define the weights:
$x_u^p = \bigoplus\{v(\sigma_1) \mid \langle p_s, u_1\rangle \Rightarrow^{\sigma_1} \langle p, \varepsilon\rangle\}$ and $y_u^p = \bigoplus\{v(\sigma_2) \mid \langle p, \gamma\rangle \Rightarrow^{\sigma_2} \langle p_t, u_2\rangle\}$ and $w_u = \bigoplus_{p\in P}(x_u^p \otimes y_u^p)$. Let $w_{s=t}$ be $\overline{1}$ if $s = t$ and $\overline{0}$ otherwise. Then $\tau_{\mathcal{W}}(s,t) = \bigoplus\{w_u \mid u \text{ is a common suffix of } u_s \text{ and } u_t\} \oplus w_{s=t}$.

For example, if $s = \langle p_1, abbbbb\rangle$ and $t = \langle p_2, bbb\rangle$ with the WPDS given in Fig. 3(a), and we choose suffix $u = b$, then $\gamma = b$ and $x_b^{p_1} = \infty$, $x_b^{p_2} = 5$, $y_b^{p_1} = \infty$, and $y_b^{p_2} = \infty$. Thus, $w_b = \infty$. When we choose suffix $u = bbb$, we get $w_{bbb} = 4$, which is the length of the shortest path from s to t.

Again, let $u_s = u_1\ \gamma\ u$ and $u_t = u_2\ u$. Let $\mathcal{A}_1 = \mathcal{A}_{pop}$ and $\mathcal{A}_2 = \mathcal{A}_{grow}$ with final states restricted to $\{q_{p,\gamma} \mid p \in P\}$. Then x_u^p is the meet-over-all-paths weight on state p when $\mathcal{A}_1$ is started in state p_s with input u_1. This requires time $O_s(|P|^2|s|)$ to compute for all u_1, where $|s|$ is the length of u_s. The weight y_u^p is the meet-over-all-paths weight on state $q_{p,\gamma}$ when $\mathcal{A}_2$ is started in state p_t and given input u_2. This requires time $O_s((|P| + |\Delta|)^2|t|)$ to compute for all u_2, where $|t|$ is the length of u_t. Putting these results together, we can calculate the combine of weights of all paths from s to t in time $O_s(|P|^2|s| + (|P| + |\Delta|)^2|t|)$. This is much faster than using existing *prestar* or *poststar* algorithms because it is independent of the height of the weight domain. This can be a big improvement because the height of weight domains used for Boolean programs (described at the beginning of §3) or predicate abstraction can be exponential in the number of Boolean variables or predicates, respectively.

Simulating $\tau_{\mathcal{W}}$ on $(p_s\ u_1\ \gamma, p_t\ u_2)$ will also give the weight w_u with the same time complexity as above, because $\tau_{\mathcal{W}}$, without the last part that accepts the untouched part of the stack (i.e., u) is simply a copy of $\mathcal{A}_{pop}$ followed by some copies of $\mathcal{A}_{grow}$.

4 Related Work

As mentioned in the introduction, a transducer construction for solving reachability in PDSs was given earlier by Caucal [7]. However, the construction was given for prefix-rewriting systems in general and is not accompanied by a complexity result, except for the fact that it runs in polynomial time. Our construction for PDSs, obtained as a special case of the construction given in §3, is quite simple and efficient. The technique, however, seems to be related. Caucal constructed the transducer by exploiting the fact that the language of the transducer is a union of the relations $(pre^*(\langle p, \gamma\rangle), post^*(\langle p, \gamma\rangle))$ for all $p \in P$ and $\gamma \in \Gamma$, with an identity relation appended onto them to accept the untouched part of the stack. This is similar to our decomposition of PDS paths (see Fig. 2). The extension of the result to WPDSs also allows one to solve *st*-path problems more efficiently than existing algorithms.

There is a large body of work on weighted automata and weighted transducers in the speech-recognition community [21, 22]. However, the weights in their applications usually satisfy many more properties than those of a semiring, including the existence of an inverse and commutativity of extend. We refrain

from making such assumptions because the weight domains used for dataflow analysis do not have these properties.

References

1. G. Balakrishnan, T. Reps, N. Kidd, A. Lal, J. Lim, D. Melski, R. Gruian, S. Yong, C.-H. Chen, and T. Teitelbaum. Model checking x86 executables with CodeSurfer/x86 and WPDS++. In *CAV*, 2005.
2. A. Bouajjani. Languages, rewriting systems, and verification of infinte-state systems. In *ICALP*, 2001.
3. A. Bouajjani, J. Esparza, A. Finkel, O. Maler, P. Rossmanith, B. Willems, and P. Wolper. An efficient automata approach to some problems on context-free grammars. *Inf. Proc. Let.*, 74(5-6):221–227, 2000.
4. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *CONCUR*, 1997.
5. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL*, 2003.
6. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, 2000.
7. D. Caucal. On the regular structure of prefix rewriting. *TCS*, 106(1):61–86, 1992.
8. D. Caucal and R. Monfort. On the transition graphs of automata and grammars. In *Graph-Theoretic Concepts in Computer Science*, pages 311–337, 1990.
9. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV*, 2000.
10. A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems. *Electronic Notes in Theoretical Computer Science*, 9, 1997.
11. GrammaTech, Inc. CodeSurfer Path Inspector, 2005. http://www.grammatech.com/products/codesurfer/overview_pi.html.
12. S. Jha and T. Reps. Analysis of SPKI/SDSI certificates using model checking. In *CSFW*, 2002.
13. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256:93–112, 2001.
14. S. Kiefer, S. Schwoon, and D. Suwimonteerabuth. Moped, 2005. http://www.informatik.uni-stuttgart.de/fmi/szs/tools/moped/nmoped/.
15. G. A. Kildall. A unified approach to global program optimization. In *POPL*, 1973.
16. A. Lal, J. Lim, M. Polishchuk, and B. Liblit. Path optimization in programs and its application to debugging. In *ESOP*, 2006.
17. A. Lal and T. Reps. Improving pushdown system model checking. In *CAV*, pages 343–357, 2006.
18. A. Lal and T. Reps. Improving pushdown system model checking. Technical Report 1552, University of Wisconsin-Madison, Jan. 2006.
19. A. Lal, T. Reps, and G. Balakrishnan. Extended weighted pushdown systems. In *CAV*, 2005.
20. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2), 1997.
21. M. Mohri, F. C. N. Pereira, and M. Riley. Weighted automata in text and speech processing. In *ECAI*, 1996.
22. M. Mohri, F. C. N. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. In *TCS*, 2000.

23. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *POPL*, 2004.
24. T. Reps, S. Schwoon, and S. Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *SAS*, 2003.
25. T. Reps, S. Schwoon, S. Jha, and D. Melski. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *SCP*, 2005.
26. S. Schwoon. *Model-Checking Pushdown Systems.* PhD thesis, Technical Univ. of Munich, Munich, Germany, July 2002.
27. S. Schwoon, S. Jha, T. Reps, and S. Stubblebine. On generalized authorization problems. In *CSFW*, 2003.
28. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV*, 1998.

A Proofs

Lemma 1. Let λ be the transition system of τ, and $p \overset{v_1/v_2}{\rightarrow}_\lambda q$ mean that there is a sequence of transitions in λ that go from state p to state q on input v_1 and output v_2. Let $\langle p, v\rangle \overset{k}{\Rightarrow}^* \langle p, w\rangle$ denote that $\langle p, w\rangle$ is derived from $\langle p, v\rangle$ in k steps. We proceed by induction on k:

- $k = 0$. Then $v = w$, and the property holds since $p \overset{v/v}{\rightarrow}_\lambda q$.
- $k > 0$. Let $\langle p, u\rangle$ be a configuration such that

$$\langle p, v\rangle \overset{k-1}{\Rightarrow}^* \langle p, u\rangle \overset{1}{\Rightarrow} \langle p, w\rangle$$

By induction, it follows that $p \overset{v/u}{\rightarrow}_{\lambda'} q$. Because $\langle p, u\rangle \overset{1}{\Rightarrow}^* \langle p, w\rangle$, there are γ, u_1, w_1 such that

$$u = \gamma u_1, w = w_1 u_1, \text{ and } \langle p, \gamma\rangle \hookrightarrow \langle p, w_1\rangle$$

There are three cases, depending on the length of w_1. Let us consider the case where $w_1 = \gamma'\gamma''$, the other ones being similar. Let $v = v_1 v_2 \cdots v_n$. There are 2 cases:

1. The derivation $p \overset{v/u}{\rightarrow}_{\lambda'} q$ is of the form:

$$p \overset{v_1/\gamma}{\rightarrow} q_1 \overset{v'/u_1}{\rightarrow} q$$

where $v' = v_2 \cdots v_n$. In this case, rule (α_5) implies that

$$p \overset{v_1/\gamma'}{\rightarrow} s_{(p,v_1,\gamma')} \overset{\varepsilon/\gamma''}{\rightarrow} q_1 \overset{v'/u_1}{\rightarrow} q$$

2. The derivation $p \overset{v/u}{\rightarrow}_{\lambda'} q$ is of the form:

$$p \overset{v_1/\varepsilon}{\rightarrow} q'_1 \overset{v_2/\varepsilon}{\rightarrow} q'_2 \cdots \overset{v_i/\varepsilon}{\rightarrow} q'_i \overset{v_{i+1}/\gamma}{\rightsquigarrow} q_{i+1} \overset{v''/u_1}{\rightarrow} q$$

where $v'' = v_{i+2} \cdots v_n$. In this case also, rule (α_5) implies that

$$p \overset{v_1/\varepsilon}{\rightarrow} q'_1 \overset{v_2/\varepsilon}{\rightarrow} q'_2 \cdots \overset{v_i/\varepsilon}{\rightarrow} q'_i \overset{v_{i+1}/\gamma'}{\rightarrow} s_{(q_i,v_{i+1},\gamma')} \overset{\varepsilon/\gamma''}{\rightarrow} q_{i+1} \overset{v''/u_1}{\rightarrow} q$$

Therefore, in both cases, we have that $p \overset{v/w}{\rightarrow} q$.

Lemma 2. Let $q \in Q_1$. We proceed by induction on the number j of occurrences of states of the form q', for $q \in Q_1 \cup Q_2$, that appear in the derivation $p \overset{v/w}{\rightarrow}_{\lambda'} q$.

- $j = 0$. The proof of this case follows the lines of the proof of Schwoon that shows that the saturation procedure *poststar* computes an automaton that recognizes $post^*$ [26].
- $j > 0$. The derivation $p \overset{v/w}{\rightarrow}_{\lambda'} q$ is then of the following form,:

$$p \xrightarrow{\gamma_1/\varepsilon} q'_1 \xrightarrow{\gamma_2/\varepsilon} q'_2 \xrightarrow{\gamma_3/\varepsilon} \cdots \xrightarrow{\gamma_{m-1}/\varepsilon} q'_{m-1} \xrightarrow{\gamma_m/\varepsilon} q'_m \overset{\gamma_{m+1}/\gamma}{\rightsquigarrow} q_{m+1} \xrightarrow{v_2/w_2} q$$

where $v = v_1 v_2$ such that $v_1 = \gamma_1 \cdots \gamma_{m+1}$, and $w = \gamma w_2$. Then, let $\langle p, \gamma' \rangle \hookrightarrow \langle p, \varepsilon \rangle$ be a rule such that the transition $(q'_{m-1}, \gamma_m/\varepsilon, q'_m)$ is added to λ' because $q'_{m-1} \overset{\gamma_m/\gamma'}{\rightsquigarrow} q_m$.

There are two cases, depending on the form of the derivation $q'_m \overset{\gamma_{m+1}/\gamma}{\rightsquigarrow} q_{m+1}$:

- This derivation is of the form $q'_m \xrightarrow{\varepsilon/\varepsilon} q_m \xrightarrow{\gamma_{m+1}/\gamma} q_{m+1}$. Then, the following derivation holds:

$$p \xrightarrow{\gamma_1/\varepsilon} q'_1 \xrightarrow{\gamma_2/\varepsilon} q'_2 \xrightarrow{\gamma_3/\varepsilon} \cdots \xrightarrow{\gamma_{m-1}/\varepsilon} q'_{m-1} \quad \text{and}$$

$$q'_{m-1} \overset{\gamma_m/\gamma'}{\rightsquigarrow} q_m \xrightarrow{\gamma_{m+1}/\gamma} q_{m+1} \xrightarrow{v_2/w_2} q$$

  This derivation contains $j-1$ occurrences of states of the form q', for $q \in Q_1 \cup Q_2$. By induction, it follows that $p \overset{v/v}{\rightarrow}_{\lambda} q$ and $\langle p, v \rangle \Rightarrow^* \langle p, \gamma' \gamma w_2 \rangle$. Because $\langle p, \gamma' \rangle \hookrightarrow \langle p, \varepsilon \rangle$ is a rule of the PDS, it follows that $\langle p, v \rangle \Rightarrow^* \langle p, \gamma w_2 \rangle = \langle p, w \rangle$. Therefore, the property holds in this case.
- This derivation is of the form $q'_m \xrightarrow{\gamma_{m+1}/\gamma} q_{m+1}$. This transition is added to λ' because there is a sequence of PDS rules $\langle p, n_1 \rangle \hookrightarrow \langle p, \gamma \rangle$, $\langle p, n_2 \rangle \hookrightarrow \langle p, n_1 \rangle$, $\cdots$, and $\langle p, n_l \rangle \hookrightarrow \langle p, n_{l-1} \rangle$ such that λ' initially contains the derivation $q'_m \xrightarrow{\varepsilon/\varepsilon} q_m \xrightarrow{\gamma_{m+1}/n_l} q_{m+1}$, and then the following transitions were added to λ':
  - the transition $(q'_m, \gamma_{m+1}/n_{l-1}, q_{m+1})$, because $\langle p, n_l \rangle \hookrightarrow \langle p, n_{l-1} \rangle$ is a rule of the PDS.
  - then, the transition $(q'_m, \gamma_{m+1}/n_{l-2}, q_{m+1})$, because $\langle p, n_{l-1} \rangle \hookrightarrow \langle p, n_{l-2} \rangle$ is a rule of the PDS.
  - etc., until the transition $(q'_m, \gamma_{m+1}/\gamma, q_{m+1})$, because $\langle p, n_1 \rangle \hookrightarrow \langle p, \gamma \rangle$ is a rule of the PDS.

  Then, the following derivation holds in λ':

$$p \xrightarrow{\gamma_1/\varepsilon} q'_1 \xrightarrow{\gamma_2/\varepsilon} q'_2 \xrightarrow{\gamma_3/\varepsilon} \cdots \xrightarrow{\gamma_{m-1}/\varepsilon} q'_{m-1} \quad \text{and}$$

$$q'_{m-1} \xrightarrow{\gamma_m/\varepsilon} q'_m \xrightarrow{\varepsilon/\varepsilon} q_m \xrightarrow{\gamma_{m+1}/n_l} q_{m+1} \xrightarrow{v_2/w_2} q$$

We can show as before that $p \xrightarrow{v/v}_{\lambda} q$ and $\langle p, v\rangle \Rightarrow^* \langle p, n_l w_2\rangle$. Then, because $\langle p, n_l\rangle \Rightarrow^* \langle p, \gamma\rangle$, it follows that $\langle p, v\rangle \Rightarrow^* \langle p, \gamma w_2\rangle = \langle p, w\rangle$. Therefore, the property holds in this case.

Lemma 4. A formal proof for this lemma would follow from a characterization of the rule sequences that each automaton transition represents, based on the *abstract grammar* formulation of *prestar* [25]. We give a slightly informal, but intuitive, proof here. We use the fact that the saturation-based implementation of *prestar* is correct [25].

The lemma runs *prestar* on the empty automaton (which represents the configuration set $C = \{\langle p, \varepsilon\rangle \mid p \in P\}$). Let β be a stack symbol not in Γ, and $\mathcal{A}^p_\beta$ be an automaton with two states $\{p, q\}$, $q \notin P$ and a single transition (p, β, q). Let q be the final state of this automaton. Because $\beta \notin \Gamma$, running *prestar* on $\mathcal{A}^p_\beta$ will return the same automaton as the one returned by running *prestar* on the empty automaton, except for the extra transition (p, β, q) (because no rule can match β). $\mathcal{A}^p_\beta$ represents the configuration set $\{\langle p, \beta\rangle\}$, and therefore, $\mathcal{A}^p_\beta(\langle p', \gamma\ \beta\rangle) = \mathit{pop}(p', \gamma, p)$ according to the definition of *pop*. However, $\mathcal{A}^p_\beta(\langle p', \gamma\ \beta\rangle)$ is exactly the weight on the transition (p', γ, p) because the only path in $\mathcal{A}^p_\beta$ that accepts $(\gamma\ \beta)$ starting in state p' is the one that follows transitions (p', γ, p) and (p, β, q). The results follows by repeating the argument for all $p \in P$.

Lemma 5. The proof is similar to the one given for Lemma 4. Let $\beta \notin \Gamma$ be a new stack symbol. Let $\mathcal{A}^{p,\gamma}_\beta$ be the automaton $\mathcal{A}$ with an extra state q_f and an extra transition $(q_{p,\gamma}, \beta, q_f)$. Let q_f be the final state of this automaton. $\mathcal{A}^{p,\gamma}_\beta$ represents the configuration set $\{\langle p, \gamma\ \beta\rangle\}$. The automaton returned by $\mathit{poststar}(\mathcal{A}^{p,\gamma}_\beta)$ would then represent the configuration set $\mathit{grow}(p, \gamma)$ with β appended at the end of the stack. The proof follows from the fact that running *poststar* on $\mathcal{A}^{p,\gamma}_\beta$ is the same as running it on $\mathcal{A}$ (for all p and γ) with the exception of the extra β-transition.

Theorem 3. The proof is based on the observation made in Fig. 2. Suppose we have a path in the PDS transition relation from $\langle p, \gamma_1\gamma_2\cdots\gamma_n\rangle$ to $\langle p_{k+1}, u\gamma_{k+2}\cdots\gamma_n\rangle$ that can be broken down as shown in Fig. 6. Then

$$\begin{aligned}
\langle p, \gamma_1\gamma_2\cdots\gamma_n\rangle &\Rightarrow^* \langle p_1, \gamma_2\cdots\gamma_n\rangle && w_1\\
&\Rightarrow^* \langle p_2, \gamma_3\cdots\gamma_n\rangle && w_2\\
&\Rightarrow^* \cdots\\
&\Rightarrow^* \langle p_k, \gamma_{k+1}\cdots\gamma_n\rangle && w_k\\
&\Rightarrow^* \langle p_{k+1}, u\gamma_{k+2}\cdots\gamma_n\rangle && w_{k+1}
\end{aligned}$$

Fig. 6. A path in the PDS's transition relation with corresponding weights of each step.

in the transducer, we can take the path starting at q_i that first takes the transition $(q_i, (p/\varepsilon), p_{pop})$ (Step 1 of the construction) and moves into state p of $\mathcal{A}_{pop}$. Then it successively takes the transitions $(p_1, (\gamma_2/\varepsilon), p_2)$, $(p_2, (\gamma_3/\varepsilon), p_3), \cdots, (p_{k-1}, (\gamma_k/\varepsilon), p_k)$ (Step 2), all the time staying inside $\mathcal{A}_{pop}$.

If the weight of the i^{th} such transition is w^i, then $w^i \sqsubseteq w_i$ (where $a \sqsubseteq b$ iff $a \oplus b = a$). This follows from Lemma 4. Next, the transducer can take transition $(p_k, (\varepsilon/p_{k+1}), p_{k+1})$ (Step 4) and move into $\mathcal{A}_{p_k}$. Then it can take a path that outputs u and move into state $q_{p_k,\gamma_{k+1}}$. There is one such path because $\mathcal{A}_{p_k}$ can accept u starting in state p_{k+1} (representing the configuration $\langle p_{k+1}, u\rangle$) when the final state is $q_{p_k,\gamma_{k+1}}$ (Lemma 5). Moreover, the combine of weights of all such paths in the transducer is $\sqsubseteq w_{k+1}$. After this, the transducer can take transition $(q_{p_{k+1},\gamma_{k+1}}, (\gamma_{k+1}/\varepsilon), q_f)$ (Step 5) and copy the stack $(\gamma_{k+2} \cdots \gamma_n)$ on to the output tape in the final state q_f (Step 7). The path we just described took input $(p\ \gamma_1\gamma_2 \cdots \gamma_n)$ and output $(p_{k+1}\ u\gamma_{k+2} \cdots \gamma_n)$ as required, and the combine of weights of all such paths is $\sqsubseteq$ the weight of the path shown in Fig. 6 $(w_1 \otimes w_2 \otimes \cdots \otimes w_{k+1})$. Note that there is a corresponding path in the transducer (that uses transitions inserted in Step 6) when the path shown in Fig. 6 has no growth phase.

To argue the other direction, the reasoning is similar. A path in the transducer must start in state q_i, then move into $\mathcal{A}_{pop}$, then into $\mathcal{A}_p$ (for some $p \in P$) and then move to state q_f. Keeping track of the input and output required for this path, we can build the WPDS path as in Fig. 6. Using Lemmas 4 and 5, the weight of such a path in the transducer would be $\sqsupseteq$ the combine of weights of all paths between the configurations in the PDS's transition relation.