Branching Pushdown Tree Automata *

Rajeev Alur and Swarat Chaudhuri

University of Pennsylvania

Abstract. We observe that pushdown tree automata (PTAs) known in the literature cannot express combinations of branching and pushdown properties. This is because a PTA processes the children of a tree node in possibly different control states but with identical stacks. We propose *branching pushdown tree automata* (BPTAs) as a solution. In a BPTA, a push-move views its matching pops as an unbounded, unordered set of successor moves and can assert existential and universal requirements on them, just the way finite automata on unranked, unordered trees pass requirements to the children of a tree node. We show that BPTAs can express some natural properties and are more expressive than PTAs. Using a small-model theorem, we prove their emptiness problem to be decidable. The problem becomes undecidable, however, if push-moves are allowed to specify the *ordering* of matching pops.

1 Introduction

Regular languages of trees [1] have been studied extensively in the literature [10] and found a number of applications. Automata accepting such languages can reason about paths in a tree existentially ("a symbol a is seen along *some* path from the current node") and universally ("a is seen on *all* paths"). Concretely, while reading a node in a binary tree, a nondeterministic, top-down tree automaton can nondeterministically pick pairs of different states to be sent to the children of the current node. Such "branching" of the finite control permits tree automata to specify properties of trees such as: "every node labeled a has a descendant labeled b and another descendant labeled c."

Above the class of regular tree languages in the hierarchy of expressiveness lies the class of *context-free* tree languages [7, 1]. Such languages are accepted by *nondeterministic pushdown tree automata* (PTAs) [3, 9, 8, 4, 6], which augment tree automata with pushdown stores. PTAs are expressively equivalent to context-free tree grammars [1, 7], and their emptiness problem is in EXPTIME [11, 6]. Their usual operational definition runs as follows: while reading a tree node, a PTA $\mathcal{A}$ assumes a *configuration* of the form (q, w), where q is a state and w is a stack. At any point, $\mathcal{A}$ may push or pop the stack, or it may fork copies to be sent to the children of the current node. The essence of the expressiveness of PTAs, however, lies in the fact that they allow information stored on the stack

* This research was partially supported by ARO URI award DAAD19-01-1-0473 and NSF award CCR-0306382.

at a push-transition to be retrieved at "matching" pop-transitions arbitrarily far away. Another way to view this is: a push-move in a PTA $\mathcal{A}$ can *constrain* its matching pops—for instance, it may require q' to be the only state reached via the latter. This is analogous to the way a transition in a tree automaton can constrain the automaton's state at the children of a tree node.

We note, however, that in existing definitions of PTAs, copies of the automaton forked by a branch-transition have *identical stacks*, even though they may differ in control state (in some definitions, the stacks may differ, but only in a bounded way). If a push stores γ on the stack, then at *every matching pop*, it is the same γ that must be popped. Thus, while γ may be used to require that every matching pop leads to state q', a push-move cannot assert properties such as: "there exists a unique matching pop leading to state q_1, and every other matching pop leads to state q_2." Intuitively, a PTA can only express universal (as opposed to universal *and* existential) matching requirements. On the other hand, tree automata can reason universally and existentially about the children of a tree node—for unranked trees, MSO-complete tree automata [5] have transitions asserting requirements such as: "there exists a unique child to which state q_1 is passed, and every remaining child gets state q_2." Thus, PTAs do not really combine the way tree automata specify branching properties with the way pushdown automata express matching requirements.

To see how this prevents PTAs from capturing the interplay of matching and tree branching, consider a basic pushdown language: that of words over brackets $[$, $]_1$ and $]_2$ where every bracket $[$ has a matching instance of $]_1$ or $]_2$. Now consider the language L of trees labeled by the above brackets where: (1) each node labeled $[$ has a single descendant labeled $]_1$ such that the path from the former to the latter is "matched," and (2) every other "matched descendant" is labeled $]_2$. A push-transition taken by a PTA at a node labeled $[$ (or within a bounded distance from it) can check that *all* matching brackets reachable from the point of push are of a certain type. However, no PTA can accept L.

In this paper, we introduce *branching pushdown tree automata* (BPTAs), a class of pushdown automata which run on trees but do not suffer from this shortcoming in expressiveness. A push-transition in a BPTA views the tree nodes reached via its matching pops as an unbounded, unordered set of successor nodes, and can assert existential and universal requirements on them. More precisely, a push-transition is of the form $q \to (q', \mathit{push}(\chi))$, where q is the source state, q' is the destination state, and χ, a *constraint* on the states reached by the matching pops, can demand a requirement such as: "state q_1 is reached through one matching pop, and the rest lead to q_2." Note how this is analogous to the way MSO-complete finite automata on unranked, unordered trees can assert requirements on the children of a tree node. Thus, the ability of tree automata to reason about tree branches is combined seamlessly with the power of pushdown automata to match brackets, letting BPTAs accept a "truly pushdown" class of tree languages. Note also that the language L may now be accepted easily. At nodes labeled $]_1$ and $]_2$, the BPTA $\mathcal{A}$ for L pops and moves respectively to states

q_1 and q_2, then continues down the tree. At a node labeled [, $\mathcal{A}$ pushes, asserting the constraint χ on the matching pops, before it branches.

BPTAs enjoy closure properties similar to PTAs but are provably more expressive. The main technical result of this paper is an algorithm for their emptiness problem. The analogous problem for PTAs reduces to pushdown games [11]; however, such a reduction seems impossible in this case. Instead, we define a proof system that, for states q and constraints χ, derives facts such as "starting at state q with empty stack from the root of some tree, the automaton has a way to reach the leaves of that tree with empty stack, at states that together satisfy χ." Using a small-model theorem that states that a short proof exists for every proof in this system, we obtain a 3-EXPTIME algorithm for the emptiness problem. Intriguingly, checking emptiness becomes undecidable if we allow BPTAs to reason about the *order* among the matching pops of a push by allowing the constraints asserted at push-moves to be regular expressions.

The organization of this paper is as follows. In Sec. 2, we present some definitions we use in the rest of the paper. In Sec. 3, we formally define BPTAs, and in Sec. 4, we present our main decision procedure. We study the expressiveness of BPTAs in Sec. 5, and conclude with some discussion in Sec. 6.

2 Basics

Binary trees Our models in this paper are *binary trees*. Let Σ be an input alphabet. A finite binary tree over Σ is a term given by the grammar $T := \bot \mid a(T, T)$, for $a \in \Sigma$. The tree $\bot$ is the *empty tree*, and the *root* of a tree $a(T_1, T_2)$ is the letter a. The i-th *leaf* of T is the i-th instance of $\bot$ in it (reading left-to-right). The *i-th composition* $(T \circ_i T')$ of T and T' is the term obtained by replacing the i-th leaf of T by T'.

Count constraints Consider a finite set Q and a word $\alpha \in Q^+$. We denote the length of α by $|\alpha|$ and the i-th symbol in α by $\alpha(i)$. The *count* of $q \in Q$ in α is the number of times q occurs in α.

We will be interested in *count constraints* over Q. Such a constraint χ follows the grammar $\chi := (\mathit{count}(q) \geq k) \mid (\mathit{count}(q) = k) \mid \chi \wedge \chi$, for $k \in \mathbb{N}$. A word α satisfies χ (written as $\alpha \models \chi$) iff it satisfies each conjunct of form $(\mathit{count}(q) \geq k)$ or $(\mathit{count}(q) = k)$; the former holds iff the count N of q in α satisfies $N \geq k$, and the latter iff $N = k$. We assume our constraints to be in the simplest possible form, i.e. no two conjuncts refer to $\mathit{count}(q)$ for the same q.

Let us now construct an alphabet of *starred elements* $Q_* = \{q^* | q \in Q\}$. We will represent a count constraint χ over Q as a *multiset* $(\mathbf{U} \cup U)$, where $\mathbf{U}$ is a multiset over Q, and $U \subseteq Q_*$. For each $q \in Q$, let m_q be the number of occurrences of q in $\mathbf{U}$; if $q^* \in U$, then set $\tau_q = (\mathit{count}(q) \geq m_q)$, else set $\tau_q = (\mathit{count}(q) = m_q)$. Then we must have $\chi = \bigwedge_q \tau_q$. Intuitively, m_q copies of q in χ guarantees any word satisfying χ to have at least m_q occurrences of q; absence of q^* (q^* is read as "an unspecified number of q-s") guarantees that the

constraint is an equality. For instance, $\chi = \{q_1, q_1^*, q_2\}$ represents the constraint $(\mathit{count}(q_1) \geq 1) \wedge (\mathit{count}(q_2) = 1) \wedge \bigwedge_{i \neq 1,2} (\mathit{count}(q_i) = 0)$.

In the sequel, we denote by $\mathit{count}(\chi, q)$ the number of times q appears in χ, and we define the *size* of χ to be $\mathit{Size}(\chi) = \sum_q \mathit{count}(\chi, q)$. Also, binary relations over constraints χ and χ' are to be interpreted as relations over the corresponding multisets. Now we define an "implied-by" relation $\preccurlyeq$ for count constraints. For constraints χ and χ', we define $\chi \preccurlyeq \chi'$ iff (1) for each $q \in Q$ such that $q^* \in \chi$, we have $\mathit{count}(\chi, q) \leq \mathit{count}(\chi', q)$, and (2) for each $q \in Q$ such that $q^* \notin \chi$, we have $\mathit{count}(\chi, q) = \mathit{count}(\chi', q)$. Clearly, if $\chi \preccurlyeq \chi'$, then for every word $\alpha \in Q^+$, we have $\alpha \models \chi' \Rightarrow \alpha \models \chi$.

For count constraints $\chi_1 = \mathbf{U}_1 \cup U_1$ and $\chi_2 = \mathbf{U}_2 \cup U_2$, where $\mathbf{U}_1, \mathbf{U}_2$ are multisets over Q and $U_1, U_2 \subseteq Q_*$, we define the *sum* $(\chi_1 + \chi_2)$ as $(\mathbf{U}_1 \cup \mathbf{U}_2) \cup (U_1 \cup U_2)$. Note that the union of $\mathbf{U}_1$ and $\mathbf{U}_2$ is a multiset union that duplicates states, whereas $U_1 \cup U_2$ is a simple set union. Likewise, for $q \in \mathbf{U}_1$, we define $(\chi_1 - \{q\})$ to be $(\mathbf{U}_1 \setminus \{q\}) \cup U_1$.

3 Branching pushdown tree automata

Syntax and semantics A *(top-down) branching pushdown tree automaton* (BPTA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where Q is a finite set of states, Σ is an input alphabet, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final states. The transition relation δ consists of four kinds of transitions: (1) push-transitions $q \to (q', \mathit{push}(\chi))$, where $q, q' \in Q$ and χ is a count constraint over Q; (2) pop-transitions $q \to (q', \mathit{pop})$, where $q, q' \in Q$; (3) swap-transitions $q \to q'$, where $q, q' \in Q$, and (4) branch-transitions $q \xrightarrow{a} (q_1, q_2)$, where $a \in \Sigma$ and $q_1, q_2 \in Q$.

Intuitively, while processing a binary tree, a BPTA is able to change its configuration using a push, pop or swap transition and process the *same tree* while in the new configuration. It may also read the root of the tree, fork two copies using a branch transition, and use them to inductively process the left and right subtrees of the present tree. We note that the assumption that the current input symbol is ignored during pushes, pops and swaps is only for simpler exposition, and does not limit expressiveness. Also, observe that the transitions of a BPTA do not manipulate a stack explicitly—indeed, we avoid the use of a stack altogether while defining runs of BPTAs. However, we will see that our definition can encode the usual stack-based semantics for pushdown automata. We will also see that pushdown tree automata (PTAs) can be encoded by BPTAs where for every constraint χ appearing in a push-transition, $\mathit{Size}(\chi) = 0$.

The semantics of a BPTA $\mathcal{A}$ is defined inductively via predicates $\mathit{Run}(q, \alpha, T)$, where $q \in Q$, α is a word over Q, and T is a binary tree. Intuitively, the predicate $\mathit{Run}(q, \alpha, T)$ is true iff the automaton has a run on the tree T, starting at state q with empty (implicit) stack and ending at the leaves of T with empty stack, such that α is obtained by reading from left to right the states of $\mathcal{A}$ at the leaves of T. Formally:

- $\mathit{Run}(q, q, \bot)$ is true;

– if $\mathcal{A}$ has a transition $q \to q'$, then $Run(q, q', \bot)$;
– if $T = a(\bot, \bot)$ and $\mathcal{A}$ has a transition $q \xrightarrow{a} (q_1, q_2)$, then $Run(q, q_1 q_2, T)$;
– assume that $Run(q', \alpha', T)$ and $\mathcal{A}$ has a transition $q \to (q', push(\chi))$. Then $Run(q, \alpha, T)$ holds if for some $\alpha \in Q^*$, we have: (1) $\alpha \models \chi$, and (2) there is a bijection $\mu : \{1, 2, \ldots, |\alpha|\} \to \{1, 2, \ldots, |\alpha|\}$ such that $\mathcal{A}$ has a transition $\alpha(i) \to (\alpha'(\mu(i)), pop)$ for all $1 \leq i \leq |\alpha|$;
– if $Run(q, \alpha, T)$ and $Run(q', \alpha', T')$, and $\alpha(i) = q'$, then $Run(q, \alpha'', T \circ_i T')$, where α'' is obtained by substituting $\alpha(i)$ by α'.

The BPTA $\mathcal{A}$ *accepts* a tree T if $Run(q_0, \alpha, T)$ for some word α over F. Informally, the acceptance condition requires the automaton to reach each leaf of T in a final state with an empty (implicit) stack. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all trees it accepts.

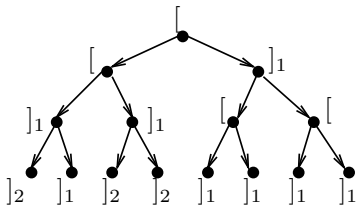

Fig. 1. A BPTA example

Among the above, the fourth and the fifth clauses are the most interesting. The fourth clause captures matching—if $\mathcal{A}$ pushes to go from q to q', and there is an empty-stack-to-empty-stack run from q' to q'', then a pop from q'' to q''' matches the original push. The distinguishing feature of BPTAs is that the word obtained by reading the q''''s from left to right must now satisfy a count constraint χ. The fifth clause captures the way a run from q and ending, among others, at q', can be *composed* with a run from q'.

To see a language recognized by a BPTA, consider binary trees over the input alphabet $\Sigma = \{[,]_1,]_2\}$. Let nodes and paths in such trees have the natural definitions, and let brackets [be *matched* by brackets $]_1$ and $]_2$. Call a node x' in a tree a *matching node* of a node x if x is labeled [, x' is labeled $]_1$ or $]_2$, and there is a well-matched path (defined in the natural way) from x to x'. Now consider the language L of such trees where (1) every path from the root to a "leaf" $\bot$ (not including the leaf itself) is matched, and (2) every node labeled [has exactly two matching nodes labeled $]_1$, and every remaining matching node is labeled $]_2$. The tree in Fig. 1, for instance, belongs to L (the leaves have been omitted to keep the figure clean).

A BPTA $\mathcal{A}$ for L has states q, $q_{]_1}$, and $q_{]_2}$, the initial state being q. On reading a node labeled $]_1$ (similarly $]_2$), $\mathcal{A}$ pops and changes state to $q_{]_1}$ (or $q_{]_2}$). On reading a node labeled [, $\mathcal{A}$ pushes and sends the state q to the children of the current node, the count constraint in the push being: "state $q_{]_1}$ appears exactly twice, and $q_{]_2}$ occurs 0 or more times."[1] It is easy to see that $\mathcal{A}$ accepts L.

Pushdown tree automata A *(top-down) pushdown tree automaton* (PTA) $\mathcal{P}$ has a finite state set H, an initial state h_0, a finite stack alphabet Γ, a set of

[1] While BPTAs, as defined, cannot push and branch in a "compound" transition, a move like this can be implemented using extra "book-keeping" states.

final states, and transitions of the types $h \to (h_1, \mathit{push}(\gamma))$, $h \to (h_1, \mathit{pop}(\gamma))$, $h \to h_1$, and $h \xrightarrow{a} (h_1, h_2)$, where $h, h_1, h_2 \in H$, $\gamma \in \Gamma$, and a is an input symbol. A *configuration* of $\mathcal{P}$ is of the form (h, w), where $h \in H$, and $w \in \Gamma^*$ is a *stack*. We define the semantics of $\mathcal{A}$ on an input tree T via predicates of the type $\mathit{Accept}((h, w), T)$, which intuitively means "$\mathcal{P}$ accepts T from configuration (h, w)," and is true if one of the following conditions holds:

- $w = \epsilon$, h is a final state, and $T = \perp$;
- there is a transition $h \to h'$ such that $\mathit{Accept}((h', w), T)$;
- $T = a(T_1, T_2)$, and for some transition $h \xrightarrow{a} (h_1, h_2)$, $\mathit{Accept}((h_1, w), T_1)$ and $\mathit{Accept}((h_2, w), T_2)$;
- there is a transition $h \to (h', \mathit{push}(\gamma))$ such that $\mathit{Accept}((h', \gamma.w), T)$;
- $w = \gamma.w'$, and for some transition $h \to (h', \mathit{pop}(\gamma))$, $\mathit{Accept}((h', w'), T)$.

The automaton $\mathcal{P}$ accepts a tree T if $\mathit{Accept}((h_0, \epsilon), T)$ holds; $\mathcal{L}(\mathcal{P})$ is the language of $\mathcal{P}$. Now, to see that BPTAs can encode PTAs, note that the only way a push-transition is different from a swap transition is that it *constrains* the "matching" pop transitions—if γ is pushed, then it is γ that must be popped at every matching pop. More precisely, construct from $\mathcal{P}$ a BPTA $\mathcal{A}$ with state set $H \cup (H \times \Gamma)$—intuitively, a pop in $\mathcal{A}$ to state (h, γ) simulates a move in $\mathcal{P}$ that pops γ and changes state to h. Every branch and swap transition in $\mathcal{P}$ is also a transition in $\mathcal{A}$; $\mathcal{A}$ also has extra swap-transitions $(h, \gamma) \to h$ for all $h \in H, \gamma \in \Gamma$. For every pop-transition $h \to (h', \mathit{pop}(\gamma))$ in $\mathcal{P}$, $\mathcal{A}$ has a transition $h \to ((h', \gamma), \mathit{pop})$, and for every push $h \to (h', \mathit{push}(\gamma))$ in $\mathcal{P}$, $\mathcal{A}$ has a transition $h \to (h', \mathit{push}(\chi))$, where the count constraint $\chi = \bigwedge_h (\mathit{count}((h, \gamma) \geq 0) \wedge \bigwedge_{\forall h.q \neq (h,\gamma)} (\mathit{count}(q) = 0)$. It is not hard to see that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{P})$. Now, let a 0-BPTA be a BPTA where for every constraint χ appearing in a push-transition, we have $\mathit{Size}(\chi) = 0$. We note that $\mathcal{A}$ is a 0-BPTA. We can also show that for every 0-BPTA, there is an equivalent PTA. Then:

Theorem 1. *The class of languages accepted by PTAs equals the class of languages accepted by BPTAs where for every constraint χ appearing in a push-transition, we have $\mathit{Size}(\chi) = 0$.*

4 Emptiness

Now we present our main technical result: a decision procedure for the problem of checking, given a BPTA $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A})$ is empty.

Consider a BPTA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where $Q = \{q_1, q_2, \ldots, q_n\}$, and recall the predicates $\mathit{Run}(q, \alpha, T)$ defined in Sec. 3. We would like to prove such predicates inductively—however, since they are unboundedly many, we must quotient them in a finite way.

We do so using predicates of the form $\mathcal{F}(q, \chi)$, where $q \in Q$ and χ is a *count constraint* over Q. The predicate $\mathcal{F}(q, \chi)$ holds iff $\mathit{Run}(q, \alpha, T)$ holds for some tree T and some word $\alpha \in Q^+$ such that $\alpha \models \chi$. Then, by definition:

Lemma 1. *If $\chi \preccurlyeq \chi'$, then for all q, $\mathcal{F}(q, \chi') \Rightarrow \mathcal{F}(q, \chi)$.*

Lemma 2. *$\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{F}(q_0, \chi_F)$, where $\chi_F = \{q^* : q \in F\}$.*

A natural question is whether it suffices to consider only predicates $\mathcal{F}(q, \chi)$ where χ appears in a push-transition of $\mathcal{A}$. It turns out that it does not. Consider a BPTA $\mathcal{A}$ that has, among others, a push-transition $q \to (q', push(\psi))$, where $\psi = \{p_1^*, p_2, p_2^*\}$, and pop-transitions $q'' \to (p_1, pop)$ and $q'' \to (p_2, pop)$. Now suppose the constraint $\chi = \{p_1, p_1^*, p_2^*\}$ appears in a different push-transition, and that we want to prove $\mathcal{F}(q, \chi)$. We note that to use the push-transition involving ψ in such a proof, we need to prove the stronger predicate $\mathcal{F}(q, \chi'')$, where $\chi'' = \chi + \psi = \{p_1, p_1^*, p_2, p_2^*\}$. If we are to use this push along with the pop-transitions from q'', we also need to prove $\mathcal{F}(q', \chi')$, where $\chi' = \{q'', q''^*\}$. However, there is no reason why χ' must appear in a transition of $\mathcal{A}$.

Hence we define a proof system $\mathbb{F}$ for facts $\mathcal{F}(q, \chi)$. The system derives predicates $\mathsf{F}(q, \chi)$ (designed to be the syntactic analog of $\mathcal{F}(q, \chi)$), and uses the rules:

1. $\dfrac{}{\mathsf{F}(q, \chi)}$, for $\chi \preccurlyeq \{q\}$ (Base)
2. $\dfrac{\mathsf{F}(q, \chi) \qquad \mathsf{F}(q', \chi')}{\mathsf{F}(q, \chi'')}$ (Compose),
   if q' is in χ and $\chi'' \preccurlyeq \chi + \chi' - \{q'\}$.
3. $\dfrac{}{\mathsf{F}(q, \chi)}$ (Swap),
   if $\mathcal{A}$ has a transition $q \to q'$ and $\chi \preccurlyeq \{q'\}$.
4. $\dfrac{}{\mathsf{F}(q, \chi)}$ (Branch),
   if $\mathcal{A}$ has a transition $q \xrightarrow{a} (q_1, q_2)$ for some a such that $\chi \preccurlyeq \{q_1, q_2\}$.
5. $\dfrac{\mathsf{F}(q', \chi')}{\mathsf{F}(q, \chi)}$ (Summarize),
   if there are count constraints χ'' and ψ such that: (1) $\chi \preccurlyeq \chi''$, (2) $\psi \preccurlyeq \chi''$, (3) $\mathcal{A}$ has a push-transition $q \to (q', push(\psi))$, and (4) there is a relation $\nu \subseteq \chi' \times \chi''$ such that:
   (a) for every $v \in \chi''$, there is some $u \in \chi'$ such that $\nu(u, v)$
   (b) for each $u \in \chi'$ that is a state of $\mathcal{A}$, we have a unique $v \in \chi''$ such that $\nu(u, v)$, v is a state of $\mathcal{A}$, and $\mathcal{A}$ has a transition $u \to (v, pop)$;
   (c) for each u of form q''^*, for $q'' \in Q$, in χ', every $v \in \chi''$ such that $\nu(u, v)$ must satisfy: (i) v is of form q'''^* for some $q''' \in Q$, and (ii) $\mathcal{A}$ has a transition $q'' \to (q''', pop)$.

Here, the rule Base may be used to establish that $\mathcal{F}(q, \{q\})$ is true. In addition, this rule can prove a "weaker" fact such as $\mathcal{F}(q, \{q, q^*\})$, implied by $\mathcal{F}(q, \{q\})$ according to Lemma 2. Generally, if any of our rules can derive a fact, then it can also derive every weaker fact.

We will explain why the rule Compose is sound to demonstrate its purpose. Suppose we have $\mathsf{F}(q, \chi)$ and $\mathsf{F}(q', \chi')$, for $q' \in \chi$. Inductively, we have $Run(q, \alpha, T)$ for some α, T such that $\alpha \models \chi$; likewise, we have $Run(q', \alpha', T')$

for some α', T' such that $\alpha' \models \chi'$. Since $q' = \alpha(i)$ for some i, we have, by the semantics of $\mathcal{A}$, $Run(q, \alpha'', T \circ_i T')$, where α'' is obtained by replacing the i-th letter of α by α'. As $\alpha'' \models \chi + \chi' - \{q'\}$, we can soundly derive any goal weaker than $\mathsf{F}(q, \chi + \chi' - \{q'\})$.

Rules Swap and Branch capture the semantics of the swap and branch transitions of $\mathcal{A}$. We will explain the rule Summarize via an example (Fig. 2). Suppose we want to derive $\mathsf{F}(q, \chi)$, where $\chi = \{p_1, p_1^*, p_2\}$. Let $\mathcal{A}$ have a push-transition $q \rightarrow (q', push(\psi))$ that, matched by some pop-transitions and combined with the true predicate $\mathcal{F}(q', \chi')$, proves $\mathcal{F}(q, \chi'')$ for some χ'' satisfying $\chi \preccurlyeq \chi''$. We must have $\psi \preccurlyeq \chi''$; also, there must exist appropriate pop-transitions from χ' to χ''. To be concrete, let $\chi'' = \{p_1, p_1, p_1, p_1^*, p_2\}$, and $\psi = \{p_1, p_1, p_1^*, p_2\}$, and suppose $\mathcal{A}$ has pop transitions $q \rightarrow (p_1, pop)$, $q' \rightarrow (p_1, pop)$, and $q' \rightarrow (p_2, pop)$. Now, every instance of p_1 (or p_2) in χ'' guarantees one copy of p_1 (p_2) reached in a run of $\mathcal{A}$, and must be derived via a pop from a copy of q or q' in χ'. The element p_1^* stands for "an unspecified number (zero or more) of p_1's," and must be derived from "an unspecified number of states that, via a pop, may lead to p_1." Thus, we may set $\chi' = \{q, q, q', q^*, q', q'^*\}$ (as in the figure) or $\chi' = \{q, q, q, q'^*, q'\}$, but not, say, $\chi' = \{q, q, q, q^*, q\}$. Now, the relation $\nu \subseteq \chi' \times \chi''$ collects the pairs (u, v) such that v is derived from u.

Let us write $\vdash_{\mathbb{F}} \mathsf{F}(q, \chi)$ if $\mathsf{F}(q, \chi)$ is derivable in $\mathbb{F}$. We can prove that:

Lemma 3. *$\mathcal{F}(q, \chi)$ iff $\vdash_{\mathbb{F}} \mathsf{F}(q, \chi)$.*

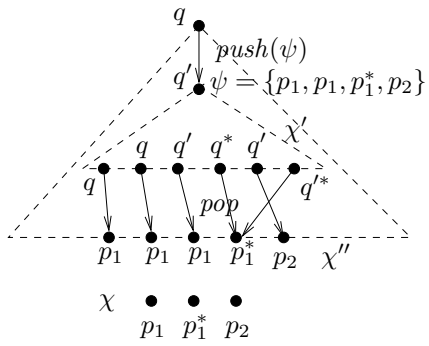

Fig. 2. The rule Summarize

Now take a proof tree for $\mathsf{F}(q_0, \chi_F)$, where χ_F is as in Lemma 2. We show that for every such tree, there is a proof for the same predicate involving a small number of predicates. Consider a path in this tree from the root (the target predicate $\mathsf{F}(q_0, \chi_F)$) to a leaf (a predicate derived without a premise). For predicates P and P' that lie on such a path, let us write $\mathsf{P} \lhd \mathsf{P}'$ if P is derived using P' in one step. We write $\mathsf{P} \lhd^+ \mathsf{P}'$ if P is obtained via a positive number of derivations from P'. Call a proof tree $\mathcal{S}$ *minimal* if it cannot be further reduced by any of the following two operations: (1) replace the proof for a predicate P in $\mathcal{S}$ by a proof tree with fewer vertices, and (2) if $\mathsf{F}(q, \chi) \lhd \mathsf{F}(q', \chi')$ in $\mathcal{S}$, then replace $\mathsf{F}(q', \chi')$ by a predicate $\mathsf{F}(q', \chi'')$, where $\chi'' \preccurlyeq \chi'$ and $\chi'' \neq \chi'$, such that $\mathsf{F}(q, \chi)$ can be derived from $\mathsf{F}(q', \chi'')$ (and the other predicates used to derive $\mathsf{F}(q, \chi)$ in $\mathcal{S}$). Note that in the above, a proof for $\mathsf{F}(q', \chi'')$ follows directly from the proof for $\mathsf{F}(q', \chi')$).

Let $\mathcal{S}$ be a minimal proof tree. We note that if $\chi \preccurlyeq \chi'$ for some χ and χ', and $\mathsf{F}(q, \chi) \lhd^+ \mathsf{F}(q, \chi')$ for some q in Q, then by Lemma 1, we can compact $\mathcal{S}$ by replacing the proof of $\mathsf{F}(q, \chi)$ by the (stronger) proof for $\mathsf{F}(q, \chi')$. Since $\mathcal{S}$ is minimal, this is a contradiction, so that:

Lemma 4. *In a minimal proof tree, we cannot have* $\mathsf{F}(q,\chi) \lhd^+ \mathsf{F}(q,\chi')$ *if* $\chi \preccurlyeq \chi'$.

Now note that if $\mathsf{F}(q,\chi) \lhd \mathsf{F}(q',\chi')$ in a minimal proof tree, then we can have $\mathit{Size}(\chi) < \mathit{Size}(\chi')$ only if the rule SUMMARIZE is used for this derivation. Now consider a state p such that $p^* \in \chi$, and let $c_{\max} = \max_\psi \max_q (\mathit{count}(\psi, q))$ be the maximum count of a state in a count constraint that appears in a push-transition of $\mathcal{A}$. Because we must have $\psi \preccurlyeq \chi''$, it may not suffice to have $\mathit{count}(\chi'', p) = \mathit{count}(\chi, p)$, but the number of extra copies of p that we may need to add is at most $c_{\max}$ (we may also have to add elements of the form q^*, but recall that they do not figure in the size of a constraint). At the same time, for states p' for which $p'^* \notin \chi$, we cannot have $\mathit{count}(\chi, p') > c_{\max}$—this is because $\mathit{count}(\chi, p') = \mathit{count}(\chi'', p') = \mathit{count}(\psi, p')$, which contradicts the definition of $c_{\max}$. Setting $k = \theta(nc_{\max})$ for the rest of this section, we have $\mathit{Size}(\chi') \leq \mathit{Size}(\chi) + k$. Then:

Lemma 5. *If* $\mathsf{F}(q,\chi) \lhd \mathsf{F}(q',\chi')$ *in a minimal proof, then* $\mathit{Size}(\chi') \leq \mathit{Size}(\chi) + k$.

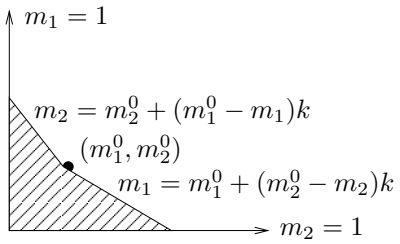

Fig. 3. Bound on constraint size

Now we abstract the problem further. From now on, only consider count constraints χ where $q \in \chi \Rightarrow q^* \in \chi$. Call a sequence of such count constraints $\varphi = \chi_1\chi_2\ldots\chi_l$ a *proof witness* if it satisfies the conditions: (1) for all i, $\mathit{Size}(\chi_{i+1}) \leq \mathit{Size}(\chi_i) + k$, (2) if $j > i$, then we cannot have $\chi_i \preccurlyeq \chi_j$, and (3) $\chi_1 = \chi_F$, where χ_F is as in Lemma 2. Let us denote by $\lambda(\varphi)$ the maximum size of a constraint in a proof witness φ. We ask the question: what is the maximum value of $\lambda(\varphi)$ over all proof witnesses φ? The answer is an upper bound on the maximum size of a count constraint χ appearing in a predicate $\mathsf{F}(q,\chi)$ in a minimal proof tree.

Define the *basis* of a count constraint χ as the set of states q such that $q^* \in \chi$. The total number of bases is bounded by 2^n. Using the facts that the ordering $\preccurlyeq$ only relates count constraints over the same basis and that we assume nothing about the specific counts of states in a constraint, we observe:

Lemma 6. *For any proof witness* $\varphi = \chi_1\chi_2\ldots\chi_l$, *there are* 2^n *sequences of count constraints* $\varphi_1, \varphi_2, \ldots, \varphi_{2^n}$, *such that (1) each constraint appearing in a particular* φ_i *has the same basis, (2) constraints in different* φ_i*'s have different bases, and (3) the concentenation* φ' *of the* φ_i*'s is a proof witness satisfying* $\lambda(\varphi) \leq \lambda(\varphi')$.

Call a proof witness *contiguous* if it may be split into sub-witnesses over particular bases in the above way. To find $\lambda(\varphi')$ for a contiguous proof witness $\varphi' = \varphi_1\varphi_2\ldots\varphi_{2^n}$, we consider a sub-witness $\varphi_i = \chi_1\chi_2\ldots\chi_l$, such that $\mathit{count}(\chi_1, q) = m$ for all q in the common basis of constraints in φ_i (in general,

the state q_i can have a count m_i, but we could set $m = \max_i m_i$ without decreasing $\lambda(\varphi)$). We will find a bound $\pi(m, n, k)$ on $\lambda(\varphi_i)$ under this assumption. First, note that for $i_1 > i_2$, we cannot have $count(\chi_{i_1}, q) > count(\chi_{i_2}, q)$ for *all* q in the basis of φ_i. Then, starting from χ_1, if we decrease the count of one of the states to 1, then the other counts can grow at most to $(m + (m-1)k) = O(mk)$ (the situation is illustrated in Fig. 3-(b) for basis size 2; here, allowed pairs (m_1, m_2) of counts are depicted as points in a 2-dimensional space, and can only lie in its shaded part—(m_1^0, m_2^0) is the "initial" point). In this way we can show that $\pi(m, n, k) \leq \pi(O(mk), n-1, k)$, and using this inequality, that $\lambda(\varphi_i) = O(m^n k^n)$. Now note that in φ_{i+1}, the first constraint has the form $(m', m', \ldots, m')$, where $m' = O(m^n k^n)$, so that $\lambda(\varphi_{i+1}) = O(m^{n^2} k^{n^2+n})$. Also, in the first constraint in φ_1, the count of each state is 0. From all this and using induction, we obtain that $\lambda(\varphi) = O(k^{n^{2^n}})$.

Now note that the total number of multisets over a basis of size n where each element can have at most r copies is r^n. Therefore, the total number of predicates $\mathsf{F}(q, \chi)$ is $O(k^{n^{2^n}})$. Since every derivation step in $\mathbb{F}$ derives at least one new predicate, we have:

Theorem 2. *The emptiness problem of BPTAs is in 3-EXPTIME.*

5 Expressiveness

Basic properties In this section, we study the expressiveness of BPTAs further. First, note that on word models, a push in a run of a BPTA has a single matching pop, so that the count constraints in push-transitions applicable in this setting can be simplified to: "one of the states in $Q' \subseteq Q$ appears once, and the other states do not occur." This can be encoded by nondeterministic pushdown word automata, proving (along with Theorem 1) that BPTAs on words accept precisely the class of context-free languages.

As for closure properties of BPTAs, closure under union is trivial. Some "hardness" results follow from Theorem 1 and results for pushdown automata:

Theorem 3. *BPTAs are closed under union, but not under intersection or complementation. The problems of checking the emptiness of (1) the complement of a BPTA and (2) the intersection of two BPTAs are undecidable.*

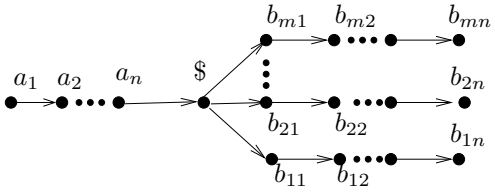

Fig. 4. Expressiveness of BPTAs

We show that BPTAs are more expressive than PTAs by considering trees as in Fig. 4 (the leaves have been omitted). Here, the input alphabet is $\Sigma = \{0, 1, \$\}$, and the symbols a_i, b_{ij} are in Σ for all i, j (while these trees are not binary, we can always encode them by such). Now let L be the language of trees of the above form where for all $i \leq n$, there is exactly one

$k \leq m$ such that $a_{n-i+1} = b_{ki}$. This language is recognized by a BPTA that has states q_0 and q_1 (along with a couple of other states needed for "book-keeping") corresponding to the input symbols 0 and 1. While reading each a_i, it executes a push-transition that enforces the following count constraint χ on its matching pops: "state q_{a_i} appears exactly once, and state q_j, where $j \neq a_i \in \{0, 1\}$ can appear an unspecified number of times." On reading a symbol b_{ij}, the BPTA executes a pop-transition to the state $q_{b_{ij}}$.

To see why L cannot be recognized by a PTA $\mathcal{M}$ with N states, take a tree as above where $n = m > N$. In any run, $\mathcal{M}$ must enter two branches of the tree in the same configuration. Then we can replace one of these branches with the other to get an accepting run on a tree not in L. This leads to:

Theorem 4. *There is a BPTA $\mathcal{A}$ such that no PTA recognizes $\mathcal{L}(\mathcal{A})$.*

Alternation One may wonder if BPTAs can be simulated by *alternating* pushdown tree automata (APTAs), which can fork copies during a run and require that all forked copies accept the input tree. Such automata have undecidable membership and emptiness problems and can accept languages not recognizable by BPTAs. For instance, the non-context free word language $L = \{a^i b^i c^i : i \geq 1\}$, clearly not accepted by a BPTA, can be accepted by an APTA [6].

However, alternation does not appear to be the source of expressiveness of BPTAs. Consider the language L of trees as in Fig. 4 where there is a $j \leq n$ such that for all $i \leq j$, there is a branch k such that $a_{n-j+1} = b_{kj}$ and $a_{n-i+1} = b_{ki}$. An APTA $\mathcal{M}$ running on such trees cannot track the universal quantifier over i just by forking copies. Such copies would run independently and not agree on the value of j. We conjecture that L cannot be accepted by an APTA. On the contrary, consider a BPTA $\mathcal{A}$ that has a pair of states $q_\sigma, q_\sigma^\#$ for each $\sigma \in \Sigma$, and pushes on the a's and pops on the b's. At every a_i preceding some nondeterministically guessed a_j, $\mathcal{A}$ pushes and asserts that, among the states reached via the matching pops, "q_{a_i} appears at least once." At a_j, $\mathcal{A}$ demands that "$q_{a_j}^\#$ occurs once or more" among the states reached by the matching pops. While popping along the k-th branch of b's, $\mathcal{A}$ has, in the beginning, the option to move to a state $q_\sigma^\#$ at any point. If it does so on a symbol b_{kl}, then it checks that $\sigma = b_{kl}$. Now it waits to move to a state $q_{\sigma'}$. If it does so on a symbol b_{kp}, then it checks that $\sigma' = b_{kp}$. We can show that $\mathcal{A}$ accepts a tree if and only if it belongs to L.

Regular expressions instead of count constraints? While a count constraint χ in a push-transition in a BPTA $\mathcal{A}$ can reason about *state counts* in the multiset of states reached via the pops matching the push, it cannot *order* them by the position of the leaves they reach. A way to let BPTAs reason about the order of matching pops would be to let χ be a *regular expression*. The semantics for push-transitions is the obvious one; pop, swap and branch transitions stay the same.

Such automata can trivially encode BPTAs; unfortunately, their emptiness problem is undecidable (we omit the proof). Evidently, the expressiveness of BPTAs is quite close to the maximum permitted by decidability contraints.

6 Conclusions

In this paper, we introduced BPTAs as a new automaton model for pushdown tree languages. Unlike pushdown tree automata studied in the literature, BPTAs allow path quantifiers to be combined with pushdown properties satisfied along a path. We established that BPTAs are strictly more expressive than classical PTAs and presented a decision procedure for their emptiness problem.

There is an intriguing connection between our decidability result and known results [2] for transition systems equipped with well-founded quasi-orders (wqo). Using Lemma 6, we can establish that the relation $\preccurlyeq$ defines a wqo on a transition system whose states are predicates of the form $\mathsf{F}(q, \chi)$. We can then pose the emptiness question for BPTAs as an *alternating coverability problem* on this transition system, which can then be proved decidable by extending existing decidability proofs for coverability in such systems.

Several questions are left open. First, we are not convinced that the upper bound for our decision procedure is tight, and it is possible that an entirely new approach would yield a better upper bound. Secondly, an extension of context-free tree grammars that is equivalent to BPTAs would be interesting to study. Finally, this paper exclusively deals with automata on finite trees, and a generalization to infinite trees and infinitary acceptance conditions would be of interest.

Acknowledgement: We thank P. Madhusudan for valuable discussions.

References

1. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Draft, 2002.
2. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
3. I. Guessarian. Pushdown tree automata. *Math. Systems Theory*, 16(4):237–263, 1983.
4. D. Harel and D. Raz. Deciding emptiness for stack automata on infinite trees. *Information and Computation*, 113(2):278–299, 1994.
5. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR 1996*, LNCS 1119, pages 263–277. Springer-Verlag, 1996.
6. O. Kupferman, N. Piterman, and M.Y. Vardi. Pushdown specifications. In *LPAR 2002*, LNCS 2514, pages 262–277. Springer, 2002.
7. W. C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.
8. A. Saoudi. Pushdown automata on infinite trees and nondeterministic context-free programs. *International Journal of Foundations of Comp. Sci.*, 3(1):21–39, 1992.
9. K. M. Schimpf and J. H. Gallier. Tree pushdown automata. *Journal of Computer and System Sciences*, 30(1):25–40, 1985.
10. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.
11. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.