Context-Free Ambiguity Detection Using Multi-stack Pushdown Automata

H.J.S. Basten[1,2(✉)]

[1] Basten Science & Software LLP, Zevenhuizen, The Netherlands
basten@bsns.nl
[2] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Abstract. We propose a method for detecting ambiguity in context-free grammars using multi-stack pushdown automata. Since the ambiguity problem is undecidable in general, we use restricted MPDAs that have a limited configuration space. The analysis might thus not be complete, but it is able to detect both ambiguity and unambiguity. Our method is general in the type of automata used. We discuss the suitability of existing MPDAs in our setting and present a new class called bounded-balance MPDAs. These MPDAs allow for infinitely deep nesting/nesting intersection, as long as the nesting depth differences within each scope stay within the balance bound. We compare our contributions to various related MPDAs and ambiguity detection methods.

1 Introduction

Context-free ambiguity detection and related problems like intersection emptiness and inclusion are important in various fields like programming language development [5], program verification [13], model checking and bioinformatics [7]. For instance, context-free grammars are very suitable for specifying formal languages because they allow the definition of regular as well as nested language constructs. However, they have the often undesirable property that they can be ambiguous. Their combinatorial complexity makes ambiguities very hard to spot, which makes automated ambiguity detection essential.

Unfortunately, deciding the ambiguity of a grammar is undecidable in the general case. Still, various ambiguity detection methods exist that aim at being either sound or complete. They limit the possibly infinite search space to either a finite subset [3,6,9,12,13,22,25] or an infinite overapproximation that is checkable in finite time [4,7,21]. For practical purposes however, it is desirable for a method to be able to answer both 'ambiguous' and 'unambiguous'. In this paper we describe a novel way to search an infinite subset in finite time, without approximation. This allows us to detect both ambiguity and unambiguity.

We propose a framework for ambiguity detection using restricted multi-stack pushdown automata. These types of automata are often used in model checking [1,8,10,16,17,20] because they can represent concurrent recursive processes. In general they are Turing complete, but with certain restrictions their configuration space can be limited and searched in finite time. Our framework is general

S. Brlek and C. Reutenauer (Eds.): DLT 2016, LNCS 9840, pp. 1–12, 2016.
DOI: 10.1007/978-3-662-53132-7_1

in the type of MPDA used, allowing the reuse of results from the model checking literature.

In addition, we propose a new class of multi-pushdown automata called bounded-balance multi-stack pushdown automata. The balance of a scope—the part of a run between the matching push and pop of a symbol—is the number of symbols pushed but not popped on other stacks during this scope. In the context of language intersection, limiting the balances in a run has several advantages over limiting the number of contexts or phases. First, it enables the possibility to detect the unambiguity of a grammar. Second, it allows for full intersection of the regular structures within a grammar with other regular or nesting structures. Third, nesting-only intersection can reach unbounded depth, as long as the nesting depths within each scope do not differ more than the balance bound.

Outline. This paper is structured as follows. The next section starts by introducing some basic definitions and notational conventions. In Sect. 3 we propose our ambiguity detection method and discuss the use of different automata types. Section 4 presents bounded-balance multi-stack pushdown automata. In Sect. 5 we compare our ambiguity detection method and MPDA type to other approaches and MPDAs. Section 6 concludes.

2 Preliminaries and Notational Conventions

Throughout this paper we use the following definitions and notations.

2.1 Context-Free Grammars

A context-free grammar G is a 4-tuple (N, T, P, S) consisting of N, a finite set of *nonterminals*, T, a finite set of *terminals* (the alphabet), P, a finite subset of $N \times (N \cup T)^*$, called the *production rules*, and $S \in N$, the *start symbol*. The character ε represents the empty string. We use V to denote the set $N \cup T$ and T^ε for $T \cup \{\varepsilon\}$. The following characters are used to represent different symbols and strings: $a, b, \ldots$ are terminals, $A, B, \ldots$ are nonterminals, $\alpha, \beta, \ldots$ are strings in V^*, $u, v, \ldots$ are strings in T^*. A production (A, α) in P is written as $A \rightarrow \alpha$. We use the function $\mathsf{pid} : P \rightarrow \mathbb{N}$ to relate each production to a unique integer. Given the string $\alpha B \gamma$ and a production rule $B \rightarrow \beta$ from P, we can write $\alpha B \gamma \Longrightarrow \alpha\beta\gamma$—read $\alpha B \gamma$ directly derives $\alpha\beta\gamma$. The *language* of a grammar G is $\mathcal{L}(G) = \{u \mid S \Longrightarrow^+ u\}$. A nonterminal A is said to be self-embedding or *nesting* iff $A \Longrightarrow^+ uAv$, otherwise its language is regular.

The *parse tree* of a sentential form describes how it is derived from S, but disregards the order of the derivation steps. To represent parse trees we use bracketed strings, which are described by bracketed grammars [11]. From a grammar $G = (N, T, P, S)$ a *bracketed grammar* G_b can be constructed by adding unique terminals to the beginning and end of every production rule. The bracketed grammar G_b is defined as the 4-tuple (N, T_b, P_b, S), where $T_b = T \cup T_\langle \cup T_\rangle$,

$T_\langle = \{\langle_i \mid \exists p \in P : i = \mathsf{pid}(p)\}$, $T_\rangle = \{\rangle_i \mid \exists p \in P : i = \mathsf{pid}(p)\}$, and $P_b = \{A \to \langle_i \alpha \rangle_i \mid A \to \alpha \in P, i = \mathsf{pid}(A \to \alpha)\}$. V_b is defined as $T_b \cup N$. The homomorphism yield from V_b^* to V^* maps each string in T_b^* to T^*. It is defined by $\mathsf{yield}(\langle_i) = \varepsilon$, $\mathsf{yield}(\rangle_i) = \varepsilon$, and $\mathsf{yield}(a) = a$. $\mathcal{L}(G_b)$ describes exactly all parse trees of all strings in $\mathcal{L}(G)$. The set of ambiguous strings of G is $\mathcal{A}(G) = \{\mathsf{yield}(u) \mid u, v \in \mathcal{L}(G_b),\, u \neq v,\, \mathsf{yield}(u) = \mathsf{yield}(v)\}$. A grammar G is ambiguous iff $\mathcal{A}(G)$ is non-empty.

2.2 Pushdown Automata

A *pushdown automaton*, or PDA, M is a 6-tuple $(Q, T^\varepsilon, \Gamma, \Delta, q_0, F)$ consisting of: Q, a finite set of *states*, T^ε, a finite set of *input symbols*, Γ, a finite set of *stack symbols* containing a bottom-of-stack symbol $\bot$, $\Delta = \Delta_\to \cup \Delta_\downarrow \cup \Delta_\uparrow$ is the *transition relation*, $\Delta_\to$ over $Q \times T^\varepsilon \times Q$ are *shift transitions*, $\Delta_\downarrow$ over $Q \times \{\downarrow\} \times \Gamma \times Q$ are *push transitions*, $\Delta_\uparrow$ over $Q \times \{\uparrow\} \times \Gamma \times Q$ are *pop transitions*, $q_0 \in Q$, is the *start state*, $F \subseteq Q$, a finite set of *accepting states.* To distinguish between pushes and pops of stack symbols we define $\Gamma' = \{\downarrow, \uparrow\} \times \Gamma$. We use p to represent stack symbols in Γ, π for strings in Γ^* and φ for symbols in $T^\varepsilon \cup \Gamma'$. An element in $Q \times \Gamma^*$ is called a *configuration*, representing a state and stack contents. We assume every PDA to start with the initial configuration $c_0 = (q_0, \bot)$. The relation Δ defines state transitions. We write $q \xrightarrow{\alpha} q'$ for tuples in $\Delta_\to$, $q \xrightarrow{\downarrow p} q'$ for tuples in $\Delta_\downarrow$ and $q \xrightarrow{\uparrow p} q'$ for tuples in $\Delta_\uparrow$. Configuration transition is denoted with $\vdash$. We write $(q, \pi) \vdash^\alpha (q', \pi)$ if $q \xrightarrow{\alpha} q'$, $(q, \pi) \vdash^{\downarrow p} (q', \pi p)$ if $q \xrightarrow{\downarrow p} q'$ and $(q, \pi p) \vdash^{\uparrow p} (q', \pi)$ if $q \xrightarrow{\uparrow p} q'$. A *run* ρ is a sequence of $\vdash$ steps. We write $c_0 \vdash^\rho c_n$ if $\rho = p_1 \ldots p_n \in (T^\varepsilon \cup \Gamma')^+$ and for every $i \in [n]$ there exists c_i s.t. $c_{i-1} \vdash^{p_i} c_i$, where $[n]$ denotes the set $\{1 \ldots n\}$. The set of possible configurations of M is $\mathcal{C}(M) = \{c \mid c_0 \vdash^* c\}$. The set of accepting runs of M is $\mathcal{R}(M) = \{\rho \mid \exists q_f \in F,\, \pi \in \Gamma^* : c_0 \vdash^\rho (q_f, \pi)\}$.

A *multi-stack pushdown automaton*, or MPDA, M_n with n stacks is a tuple $(Q, T, \widetilde{\Gamma}_n, \Delta, q_0, F)$ where Q, T, Δ, q_0 and F are defined the same as for a PDA and $\widetilde{\Gamma}_n = \bigcup_{i=1}^n \Gamma_i$ are the n stack alphabets, each containing $\bot_i$. W.l.o.g. we assume all subsets $\Gamma_i \subset \widetilde{\Gamma}_n$ to be disjoint. We use $\{\pi_i\}_{i \in [n]}$ to denote a set of stacks. A configuration is a tuple over $Q \times \Gamma_1^* \times \cdots \times \Gamma_n^*$ and $c_0 = \{q_0, \{\bot_i\}_{i \in [n]}\}$. We write $(q, \{\pi_i\}_{i \in [n]}) \vdash^\alpha (q', \{\pi_i\}_{i \in [n]})$ if $q \xrightarrow{\alpha} q'$; $(q, \{\pi_i\}_{i \in [n]}) \vdash^{\downarrow p} (q', \{\pi_i'\}_{i \in [n]})$ if $q \xrightarrow{\downarrow p} q'$, $p \in \Gamma_j$, $\pi_j' = \pi_j p$ and $\pi_i' = \pi_i$ for $i \neq j$; and $(q, \{\pi_i\}_{i \in [n]}) \vdash^{\uparrow p} (q', \{\pi_i'\}_{i \in [n]})$ if $q \xrightarrow{\uparrow p} q'$, $p \in \Gamma_j$, $\pi_j = \pi_j' p$ and $\pi_i' = \pi_i$ for $i \neq j$.

3 Ambiguity Detection with MPDAs

We present a framework for ambiguity detection of context-free grammars using multi-stack automata.

3.1 Checking Ambiguity

Given a PDA M that defines the derivations of a context-free grammar G, we can express the ambiguity problem for the grammar using an MPDA. This can be done on the condition that there is a bijective relation between the runs of the PDA and parse trees of G, let us call it $\mathsf{tree} : \mathcal{R}(M) \to \mathcal{L}(G_b)$. Two different runs of the same input string then prove the ambiguity of G.

We build a 2-stack MPDA that simulates two runs of the PDA for the same input string. The states of this MPDA consist of pairs of states of the PDA. Both stacks can be modified independently of each other, but non-empty shifts are synchronized to ensure both runs parse the same input string. Different runs for the same input string both start from q_0 but eventually split up. There are two ways in which the runs can deviate from a common state: the runs can each take different transitions, or only one of the two continues independently until the next common shift. W.l.o.g. we distinguish three possible phases in this process:

1. the runs are not split up yet and alternately follow the same transitions;
2. the first run continues with independent transitions while the second run waits for the next shift;
3. both runs are in different states.

We add two additional fields to the state pairs to register these phases: an integer field denoting the current phase and a symbol from $T^\varepsilon \cup \widetilde{\Gamma}'_2$ to record the last action taken by the first run. The second field is used to synchronize shifts and internal transitions during phase 1, and to recognize phase transitions.

Definition 1. *Given a PDA* $M = (Q, T^\varepsilon, \Gamma, \Delta, q_0, F)$ *the ambiguity MPDA of* M *is a 2-stack MPDA* $M^a = (Q^a, T^\varepsilon, \widetilde{\Gamma}_2, \Delta^a, q_0^a, F^a)$*, where* $Q^a \subseteq Q \times Q \times (T^\varepsilon \cup \widetilde{\Gamma}'_2) \times [3]$, $q_0^a = (q_0, q_0, \bot, 1)$, $F^a = F \times F \times (T^\varepsilon \cup \widetilde{\Gamma}'_2) \times \{2,3\}$, $\Delta_{\downarrow\uparrow} = \Delta_\downarrow \cup \Delta_\uparrow$,

$$
\begin{aligned}
\Delta^a = &\{\, (q,q,\bot,1) \xrightarrow{\varphi} (q',q,\varphi,1) &&\mid q \xrightarrow{\varphi} q' \in \Delta_{\downarrow\uparrow}\} \cup \\
&\{\, (q',q,\varphi,1) \xrightarrow{\varphi} (q',q',\bot,1) &&\mid \} \cup \\
&\{\, (q,q,\bot,1) \xrightarrow{\alpha} (q',q,\alpha,1) &&\mid q \xrightarrow{\alpha} q' \in \Delta_{\rightarrow}\} \cup \\
&\{\, (q',q,\alpha,1) \xrightarrow{\varepsilon} (q',q',\bot,1) &&\mid \} \cup \\
&\{\, (q,q,\bot,1) \xrightarrow{\varphi} (q',q,\bot,2) &&\mid q \xrightarrow{\varphi} q' \in \Delta_{\downarrow\uparrow}\} \cup \\
&\{\, (q',q,\varphi,1) \xrightarrow{\varphi'} (q',q'',\bot,3) &&\mid q \xrightarrow{\varphi'} q'' \in \Delta_{\downarrow\uparrow},\ \varphi' \neq \varphi \vee q'' \neq q'\} \cup \\
&\{\, (q',q,\alpha,1) \xrightarrow{\varepsilon} (q',q'',\bot,3) &&\mid q \xrightarrow{\alpha} q'' \in \Delta_{\rightarrow},\ q'' \neq q'\} \cup \\
&\{\, (q,q',\bot,y) \xrightarrow{\varphi} (q'',q',\bot,y) &&\mid q \xrightarrow{\varphi} q'' \in \Delta_{\downarrow\uparrow},\ y \in \{2,3\}\} \cup \\
&\{\, (q,q',\bot,y) \xrightarrow{\varepsilon} (q'',q',\bot,y) &&\mid q \xrightarrow{\varepsilon} q'' \in \Delta_{\rightarrow},\ y \in \{2,3\}\} \cup \\
&\{\, (q,q',\bot,y) \xrightarrow{b} (q'',q',b,y) &&\mid q \xrightarrow{b} q'' \in \Delta_{\rightarrow},\ y \in \{2,3\}\} \cup \\
&\{\, (q,q',b,y) \xrightarrow{\varepsilon} (q,q'',\bot,3) &&\mid q' \xrightarrow{b} q'' \in \Delta_{\rightarrow},\ y \in \{2,3\}\} \cup \\
&\{\, (q,q',x,3) \xrightarrow{\varphi} (q,q'',x,3) &&\mid q' \xrightarrow{\varphi} q'' \in \Delta_{\downarrow\uparrow}\} \cup \\
&\{\, (q,q',x,3) \xrightarrow{\varepsilon} (q,q'',x,3) &&\mid q' \xrightarrow{\varepsilon} q'' \in \Delta_{\rightarrow}\}.
\end{aligned}
$$

The input strings of runs leading to accepting states are the ambiguous strings of G. To test for ambiguity we choose a restricted MPDA class and compute the image of $\{q_0^a, \{\bot_i\}_{i\in[n]}\}$ under $\vdash^*$. If we can reach a state in F^a the MPDA's language is non-empty and G is ambiguous. On the other hand, if the chosen MPDA class allows the complete exploration of the configuration space of M^a and no accepting state can be reached then G is unambiguous. Otherwise, the problem remains unanswered. This is formalized by the following statements.

Lemma 2. *Given a grammar G, a PDA M and a bijective relation* $\mathsf{tree} : \mathcal{R}(M) \to \mathcal{L}(G_b)$, *the language $\mathcal{L}(M^a)$ equals $\mathcal{A}(G)$.*

Definition 3. *Given an MPDA class C^m, a PDA M is MSA(C^m)-ambiguous iff M^a has at least one run that complies with the restrictions of C^m. The PDA is MSA(C^m)-unambiguous iff M^a is a member of C^m and $\mathcal{L}(M^a)$ is empty.*

Definition 4. *Given a PDA class C^p and an MPDA class C^m, a grammar G is MSA(C^p, C^m)-ambiguous iff a C^p-PDA of G is MSA(C^m)-ambiguous. Similarly, G is MSA(C^p, C^m)-unambiguous iff a C^p-PDA of G is MSA(C^m)-unambiguous.*

Definition 5. *Given a PDA class C^p and an MPDA class C^m, a grammar G is in MSA(C^p, C^m) iff it is MSA(C^p, C^m)-ambiguous or MSA(C^p, C^m)-unambiguous.*

Theorem 6. *Given a PDA class C^p, an MPDA class C^m and a grammar G, if G is MSA(C^p, C^m)-ambiguous then G is ambiguous.*

Theorem 7. *Given a PDA class C^p, an MPDA class C^m and a grammar G, if G is MSA(C^p, C^m)-unambiguous then G is unambiguous.*

3.2 Choice of Pushdown Automaton

Since our method is parametric in the type of PDA, it can apply different strategies for exploring parse trees. Furthermore, this allows for easy integration with existing parser implementations. The parse tree exploration depends on the way a PDA uses its stack. For instance, recursive descent parsers—like LL [15]—push on every entry of a production and pop on a reduce. This makes the stack depth correspond to parse tree height. The number of pushes in a run corresponds to the number of non-leaf parse tree nodes. Shift-reduce parsers—like LR [14]—push on every shift and pop when a production is reduced, followed by another push of the reduced nonterminal. In this case the number of pushes in a run corresponds to the total number of parse tree nodes.

In general, the less stack activity a PDA requires for a given language, the larger the set of parse trees that can be covered by the configuration space of the restricted MPDA. Reduce incorporated parsers [24] are aimed at reducing the stack activity of a parser. They use the PDA as a DFA for regular structures and only use the stack to record the derivation of nesting nonterminals. Parse trees of the regular structures are built using special ε-transitions that mark reductions. However, when a nesting nonterminal is also right or left recursive the stack is used to track these kinds of derivations as well. To reduce the stack activity even further—and use it purely for nesting—we can apply a similar strategy as Nederhof ([19] Sect. 4.2), which completely separates the regular structures in a grammar from the context-free ones. This way we can completely intersect the regular structures in a grammar with each other and with the nesting ones, and fully use the stack space for nesting/nesting intersection. We do not define such a PDA here, but only mention their possibility. We will call them Nesting Stack PDAs or NSPDAs.

3.3 Choice of Multi-stack Pushdown Automaton

Below we discuss various existing MPDA types and explore their suitability for detecting ambiguity and unambiguity. To detect the ambiguity of a grammar with a certain type of MPDA, it suffices to explore a single path in M^a to an accepting state. The more paths an MPDA can cover, the higher the chance of finding an ambiguous one. In advance we can state that this is possible with all MDPA types described below, to varying extents. However, to detect unambiguity we need to ensure no state in F^a can be reached at all. This requires the configuration space of the MPDA to cover all possible paths of M^a. In order to do so, an MPDA type should pose no restrictions on nesting depth, since every nesting nonterminal will create paths in (at least) phase 1 that push to infinite stack depths and pop out of these as well. We will see that no discussed MPDA is able to cover such paths.

Another criteria we will look at is to what extent an MPDA type is able to intersect the regular structures in a grammar with other regular structures as well as with nesting structures. Since the emptiness of both regular/regular intersection and regular/nesting intersection is decidable, making use of these results enlarges the class of grammars the MPDA can decide the ambiguity of. With NSPDAs all MPDAs allow full regular/regular intersection, since this requires no stack activity. For full regular/nesting intersection an MPDA should allow one stack to reach and return from infinite depths, while the other remains untouched.

Bounded nesting depth MPDAs [10] pose an intuitive restriction, which allows complete regular/regular intersection, but only limited regular/nesting and nesting/nesting intersection. They are thus suitable for ambiguity detection, but not for unambiguity detection.

Bounded-context switching MPDAs [20] use the concept of contexts—a part of a run in which only one stack can push and pop—and restrict runs to a limited number of contexts. This allows for complete regular/regular and regular/nesting intersection. However, the depth of nesting/nesting intersection is bounded since every alternate nesting requires a context-switch. Bounded-context MPDAs can be useful for finding ambiguity, but not for finding unambiguity.

Bounded-phase MPDAs [16] use the concept of phases, in which only one stack can pop, but others are free to push. These MPDAs cover a strictly larger search space than bounded-context MPDAs [17] and are thus better suitable for finding ambiguity. Stacks can nest simultaneously to unlimited depth, but only pop out together for a limited number of steps. Hence, they can still not completely explore all configurations of phase 1.

MPDAs with scope-bounded matching relations [17,18] require that every push is popped within a bounded number of rounds, or never at all. During a round all stacks are allowed one context each, in a predefined order. These MPDAs have a larger coverage than bounded-context MPDAs, but are incomparable with bounded-phase [17]. The fact that pushes do not have to be popped can be helpful for finding ambiguity, but not for detecting unambiguity if we require all pushes to be popped. In this case, the first push of any stack has to immediately start a scope and the MPDA reverts to a bounded-context exploration.

Budget bounded MPDAs [1] allow unlimited context switches for stacks whose depth is below a certain bound, and a limited number of contexts for as long as they are above this depth bound. In other words, once the depth limit is reached, a new scope is started which has to be closed within a bounded number of contexts. Budget bounded MPDAs are thus closely related to scope-bounded MPDAs, but because they also allow pops before the start of a scope they have a larger coverage. Nevertheless, there remains a bound on the nesting depth.

Ordered MPDAs [8], the earliest type of restricted MPDA, assume an ordering of the stacks and allow only the first non-empty stack to pop. All stacks can push freely at any time. At first sight this concept might not seem suitable for nesting/nesting intersection, because it does not allow simultaneous pops. However, ordered MDPAs can simulate bounded-phase MPDAs [2] and thus allow bounded nesting/nesting intersection. In fact, they are even more expressive than bounded-phase, which makes them at least equally suitable for detecting ambiguity and unambiguity.

Concluding, we can say that all discussed MPDA types are suitable to find ambiguities with our scheme, resulting in different exploration strategies of strings and prefixes. All MPDAs can either simultaneously push into bounded or unbounded nesting depths, and some can simultaneously pop a bounded number of steps as well. However, none of the MPDAs are able to let both stacks pop out of infinitely deep nestings together, making them unsuitable for detecting the unambiguity of context-free grammars in general. In the next section we describe a new type of restricted MPDA that does have this property.

4 Bounded-Balance Multi-Stack Pushdown Automata

We propose a new type of restricted MPDA called bounded-balance multi-stack pushdown automata, or BBMPDAs. They are MPDAs with an upper bound on the number of symbols that are pushed but not popped within each scope of matching push and pop transitions.

4.1 Definition

First we introduce the concepts of scope and balance. A *scope* is the part of a run between the push of a symbol and the pop of that symbol. We use $\mu \subseteq \mathbb{N} \times \mathbb{N}$ to hold matching transition indices that open and close a scope (as in [18]). Given a run $\rho = c_0 \vdash^{\varphi_1} c_1 \vdash^{\varphi_2} \cdots \vdash^{\varphi_m} c_m$, a pair $(s,t) \in \mu_\rho$ iff $s < t$ and exists $p \in \Gamma_i$ for some $i \in [n]$ s.t. $\varphi_s = \downarrow p$, $\varphi_t = \uparrow p$ and

– for all $s < s' < t$, if $\varphi_{s'} = \downarrow p'$, $p' \in \Gamma_i$ then there exists $s' < t' < t$ such that $(s',t') \in \mu_\rho$, and
– for all $s < t' < t$, if $\varphi_{t'} = \uparrow p'$, $p' \in \Gamma_i$ then there exists $s < s' < t'$ such that $(s',t') \in \mu_\rho$.

The *balance* of a scope is the number of stack symbols that were pushed but not popped within the scope.

Definition 8. *The balance of a scope* $(s,t) \in \mu_\rho$ *is* $\mathsf{balance}(s,t) = |\{\, s' \mid (s',t') \in \mu_\rho,\ s < s' < t < t' \}|$.

Viewed differently, balance corresponds to the stack depth differences built up during a scope. By limiting the balances during runs, we acquire a new class of restricted MPDAs, which we call bounded-balance MPDAs.

Definition 9. *An n-MPDA M_n is a BB(k)MPDA iff for every run $\rho \in \mathcal{R}(M_n)$ and scope $(s,t) \in \mu_\rho$ it holds that* $\mathsf{balance}(s,t) \leq k$.

A finite balance bound allows for a finite representation of the possibly infinite configuration space of BBMPDAs. This makes testing for the BB(k) property decidable. In the following section we show how a BBMPDA can be simulated by a standard single stack PDA, which enables using existing techniques for configuration space exploration [23].

4.2 Configuration Exploration

BB(k)MPDAs can be simulated by a standard PDA that can pop from the topmost $k+1$ symbols of its stack, by temporarily remembering up to k stack symbols in its states. It has a single stack over $\widetilde{\Gamma}_n$, storing pushes of all stacks sequentially. When a certain stack needs to be popped, but its top symbol is not at the top of the simulating stack, intermediary symbols are popped and temporarily stored in the PDA states, until the required symbol is reached. This symbol is then popped as per usual, and temporarily stored symbols are pushed back to the stack again. The number of the stack to be popped is also stored in the states, so a series of borrows is always targeted at a single stack. To make sure that the order in which the symbols of the individual stacks are pushed and popped remains unchanged, a stack cannot be borrowed from once it has been targeted, i.e. only the top of the targeted stack can be popped. The number 0 is used to indicate no stack is currently targeted.

Definition 10. *Given an n-stack MPDA $M = (Q, T^\varepsilon, \widetilde{\Gamma}_n, \Delta, q_0, F)$ the k-borrowing PDA of M is $M_k^b = (Q^b, T^\varepsilon, \widetilde{\Gamma}_n, \Delta^b, (q_0, 0, \varepsilon), F \times \{0\} \times \{\varepsilon\})$, where $Q^b = Q \times \{0 \ldots n\} \times (\widetilde{\Gamma}_n \cup \{\varepsilon\})^k$, $\Delta^b =$*

$$\begin{array}{lll}
\{ (q,0,\varepsilon) \xrightarrow{X} (q',0,\varepsilon) \mid q \xrightarrow{X} q' \in \Delta\} \cup & & \textit{(copy of } \Delta\textit{)} \\
\{ (q,i,\pi) \xrightarrow{\uparrow p} (q,j,p\pi) \mid q \xrightarrow{\uparrow p'} q' \in \Delta_\uparrow, p' \in \Gamma_j, p \notin \Gamma_j, i \in \{0,j\}\} \cup & & \textit{(borrows)} \\
\{ (q,i,\pi) \xrightarrow{\uparrow p} (q',0,\pi) \mid q \xrightarrow{\uparrow p} q' \in \Delta_\uparrow, p \in \Gamma_i\} \cup & & \textit{(pops)} \\
\{ (q,0,p\pi) \xrightarrow{\downarrow p} (q,0,\pi) \mid \}. & & \textit{(returns)}
\end{array}$$

The initial configuration of M_k^b is $((q_0, 0, \varepsilon), \bot_1 \ldots \bot_n)$.

Theorem 11. *The k-borrowing PDA M_k^b of a BB(k)MPDA M simulates exactly all runs of M.*

Testing whether an MPDA is BB(k) for a fixed k comes down to constructing M_{k+1}^b and testing whether no states with borrowed stacks of size $k+1$ can be reached. Note that this scheme allows for incremental search with increasing k. The computational complexity depends on the chosen model checking algorithm, of which most are polynomial in the size of the PDA. The number of states and transitions of M_k^b is exponential in k, which puts our approach in EXPTIME.

4.3 Application to Ambiguity Detection

When applied in the ambiguity detection scheme of Sect. 3, BBMPDAs yield several desirable properties. First, they allow for full regular/regular intersection and regular/nesting intersection of the paths of M in M^a. In combination with NSPDAs, regular/regular intersection requires no stack activity and will not build up any balance. During regular/nesting intersection, which starts with the opening of a scope on one stack and ends when this scope is closed or the other stack becomes active, pushes on the active stack do add to the balance of the current scope of the other stack. However, this balance is only compared to the bound at the moment the regular/nesting intersection ends. During the intersection the nesting stack is allowed to grow and shrink indefinitely.

Second, full nesting/nesting intersection is also possible in case M^a meets the BB(k) condition. Both stacks are allowed to reach unbounded depths together and pop out of them as well, as long as the scope balances stay within the bound.

Third, BBMPDAs have the possibility of detecting the unambiguity of grammars with nesting structures, which is a consequence from the previous property. As the next theorem states, the scope balances in the paths of M^a in phase 1 will never be more than 1. Therefore, the configuration space of BB(k)MPDAs with $k \geq 1$ will at least cover all these paths. If in the continuations of these paths in phases 2 and 3, the scope balances stay within the balance bound as well and no end state is reached, the tested grammar is unambiguous.

Theorem 12. *Given a PDA M, for all partial runs $c_0 \ldots \vdash^{\uparrow p} ((q, q', x, 1), \{\pi_i\}_{i \in [n]})$ in phase 1 of M^a, the balance of the closed scope of p is at most 1.*

5 Comparisons and Related Work

In this section we compare our contributions to related MPDAs and ambiguity detection methods.

5.1 Multi-Stack Pushdown Automata

We show that BBMPDAs include bounded depth MPDAs but that they are incomparable with bounded-context switching MPDAs. This implies they are also incomparable with the larger MPDA classes mentioned in Sect. 3.3—bounded-phase, scope-bounded, budget-bounded and ordered MPDAs—since none of these can, in general, cover all paths in phase 1 of M^a. For detecting ambiguity however, these MPDAs and BBMPDAs are complementary.

Theorem 13. *If M is a n-stack MPDA with depth bound k, its scope balances are bounded by $k * (n - 1)$.*

Theorem 14. *The class of BBMPDAs is incomparable with bounded-context switching MPDAs.*

Regarding simulation, any 1-stack MPDA with enough freedom to be a plain PDA can simulate BBMPDAs. With the exception of bounded depth MPDAs this is the case for all MPDA types mentioned above.

5.2 Ambiguity Detection Methods

In this section we will discuss related work in ambiguity detection and compare it with our approach if possible.

Bounded-Search Methods. There are several methods that enumerate strings in $\mathcal{L}(G)$ of bounded length and test them for ambiguity [3,6,9,12,13,22,25]. In general these approaches are only able to detect ambiguities, because they can never entirely cover $\mathcal{L}(G)$. In essence our approach also applies a bounded search, but with the difference that the search space can cover an infinitely large language. Depending on G and the types of PDA and MPDA we can cover $\mathcal{L}(G)$ entirely and detect unambiguity. However, with certain types of PDA and MPDA, our method can also be set up for bounded string exploration. For example, using a—nondeterministic—LR PDA with a bounded-context MPDA will result in the exploration of strings and prefixes of bounded length. An LR PDA pushes with every shift, requiring a context switch to allow both stacks to push. Every reduction requires a number of pops and a push, but each stack can perform all its reductions within one context, either after the last push or before the next push, requiring no additional context switches.

Conservative Approximation Methods. Contrary to bounded search, other methods apply conservative approximation to reduce the infinite $\mathcal{L}(G)$ to a limited space. This yields the possibility of detecting unambiguity, but prohibits detecting ambiguity in most cases. The ACLA-test [7] applies regular approximation to the languages of individual production rules and searches for the absence of horizontal and vertical ambiguities using intersection and overlap operations. The NU-test [21] approximates the set of parse trees of a grammar and searches for the absence of different trees for the same ambiguous string. An extension to the NU-test allows the detection of harmless productions, which are rules that do not contribute to any ambiguity [4]. The rules can be filtered from the grammar to incrementally improve the approximation.

A significant difference between both approximative methods and our approach is that they are less able to recognize the unambiguity of nesting structures. Due to the regular approximation they lose the ability to match the left and right contexts of nestings, i.e. count the nesting depth. The ACLA-test applies production unfolding to counter this disadvantage, but this is only possible up to a certain depth. As an example, both tests are not able the detect the unambiguity of the following grammar, which is MSA(LL(0), BB(3))-unambiguous.

$$S \to A \mid B,\ A \to aAb \mid ab,\ B \to aBb \mid a \tag{1}$$

6 Conclusion

We present a novel method for detecting ambiguity in context-free grammars using restricted multi-stack pushdown automata. It is able to find both ambiguity and unambiguity. We discuss the use of existing MPDA classes within our framework, as well as propose a new class called bounded-balance MPDAs. These MPDAs are particularly useful for language intersection since they allow for unbounded nesting/nesting intersection, as long as the nesting depth differences stay within the balance bound.

References

1. Abdulla, P., Atig, M., Rezine, O., Stenman, J.: Budget-bounded model-checking pushdown systems. Form. Methods Syst. Des. **45**(2), 273–301 (2014)
2. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
3. Axelsson, R., Heljanko, K., Lange, M.: Analyzing context-free grammars using an incremental SAT solver. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 410–422. Springer, Heidelberg (2008)
4. Basten, H.J.S.: Tracking down the origins of ambiguity in context-free grammars. In: Cavalcanti, A., Deharbe, D., Gaudel, M.-C., Woodcock, J. (eds.) ICTAC 2010. LNCS, vol. 6255, pp. 76–90. Springer, Heidelberg (2010)
5. Basten, H.J.S.: Ambiguity detection for programming language grammars. Ph.D. thesis, Universiteit van Amsterdam (2011)
6. Basten, H.J.S., Vinju, J.J.: Faster ambiguity detection by grammar filtering. In: Proceedings of the Tenth Workshop on Language Descriptions, Tools and Applications (LDTA 2010), pp. 5:1–5:9. ACM (2010)
7. Brabrand, C., Giegerich, R., Møller, A.: Analyzing ambiguity of context-free grammars. Sci. Comput. Program. **75**(3), 176–191 (2010)

8. Breveglieri, L., Cherubini, A., Citrini, C., Reghizzi, S.C.: Multi-push-down languages and grammars. Int. J. Found. Comput. Sci. **07**(03), 253–291 (1996)
9. Cheung, B.S.N., Uzgalis, R.C.: Ambiguity in context-free grammars. In: Proceedings of the 1995 ACM Symposium on Applied Computing (SAC 1995), pp. 272–276. ACM, New York (1995)
10. Clarke, E., Kroning, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004)
11. Ginsburg, S., Harrison, M.A.: Bracketed context-free languages. J. Comput. Syst. Sci. **1**(1), 1–23 (1967)
12. Gorn, S.: Detection of generative ambiguities in context-free mechanical languages. J. ACM **10**(2), 196–208 (1963)
13. Kieżun, A., Ganesh, V., Guo, P.J., Hooimeijer, P., Ernst, M.D.: HAMPI: a solver for string constraints. In: Proceedings of the 2009 International Symposium on Software Testing and Analysis (ISSTA 2009), pp. 105–116. ACM (2009)
14. Knuth, D.E.: On the translation of languages from left to right. Inf. Control **8**(6), 607–639 (1965)
15. Knuth, D.E.: Top-down syntax analysis. Acta Informatica **1**, 79–110 (1971)
16. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007), pp. 161–170. IEEE (2007)
17. La Torre, S., Napoli, M.: Reachability of multistack pushdown systems with scope-bounded matching relations. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 203–218. Springer, Heidelberg (2011)
18. La Torre, S., Parlato, G.: Scope-bounded multistack pushdown systems: fixed-point, sequentialization, and tree-width. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012), pp. 173–184 (2012)
19. Nederhof, M.: Practical experiments with regular approximation of context-free languages. Comput. Linguist. **26**(1), 17–44 (2000)
20. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
21. Schmitz, S.: Conservative ambiguity detection in context-free grammars. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 692–703. Springer, Heidelberg (2007)
22. Schröer, F.W.: AMBER, an ambiguity checker for context-free grammars. Technical report (2001). compilertools.net, http://accent.compilertools.net/Amber.html
23. Schwoon, S.: Model-checking pushdown systems. Ph.D. thesis, Technische Universität München, June 2002
24. Scott, E., Johnstone, A.: Generalized bottom up parsers with reduced stack activity. Comput. J. **48**(5), 565–587 (2005)
25. Vasudevan, N., Tratt, L.: Detecting ambiguity in programming language grammars. In: Erwig, M., Paige, R.F., Van Wyk, E. (eds.) SLE 2013. LNCS, vol. 8225, pp. 157–176. Springer, Heidelberg (2013)