,23Javier Esparza, Pierre Ganty, Rupak MajumdarJavier Esparza, Pierre Ganty, Rupak Majumdar,,23A Perfect Model for Bounded VerificationA Perfect Model for Bounded Verification,

A Perfect Model for Bounded Verification

Javier Esparza
Institut für Informatik,
Technische Universität München,
Garching, Germany

Pierre Ganty
IMDEA Software Institute,
Madrid, Spain

Rupak Majumdar
Max Planck Institute for Software Systems,
Kaiserslautern, Germany

***Abstract*—A class of languages *C* is *perfect* if it is closed under Boolean operations and the emptiness problem is decidable. Perfect language classes are the basis for the automata-theoretic approach to model checking: a system is correct if the language generated by the system is disjoint from the language of bad traces. Regular languages are perfect, but because the disjointness problem for context-free languages is undecidable, no class containing them can be perfect.**

In practice, verification problems for language classes that are not perfect are often under-approximated by checking if the property holds for all behaviors of the system belonging to a fixed subset. A general way to specify a subset of behaviors is by using bounded languages. A class of languages *C* is *perfect modulo bounded languages* if it is closed under Boolean operations relative to every bounded language, and if the emptiness problem is decidable relative to every bounded language.

We consider finding perfect classes of languages modulo bounded languages. We show that the class of languages accepted by multi-head pushdown automata are perfect modulo bounded languages, and characterize the complexities of decision problems. We also show that bounded languages form a maximal class for which perfection is obtained. We show that computations of several known models of systems, such as recursive multi-threaded programs, recursive counter machines, and communicating finite-state machines can be encoded as multi-head pushdown automata, giving uniform and optimal underapproximation algorithms modulo bounded languages.

***Index Terms*—verification, underapproximation, formal languages, complexity, algorithms.**

I. Introduction

The automata-theoretic approach to model checking linear-time properties formalizes the verification problem as a language-theoretic problem about two automata: the *system automaton*, which recognizes the set of executions of the system, and the *property automaton*, which recognizes either the sequences of actions satisfying the property (*positive* specification), or those violating it (*negative* specification). Given a system automaton S and a property automaton P, verification of positive and negative specifications reduces to checking $L(S) \subseteq L(P)$ (inclusion problem), or to checking $L(S) \cap L(P) = \emptyset$ (disjointness problem), respectively.

Language classes effectively closed under Boolean operations and with a decidable emptiness problem are particularly interesting for the automata-theoretic approach. Besides the inclusion and disjointness problems being decidable, such classes have many further advantages. For example, systems are closed under parallel composition by rendezvous, properties are closed under Boolean operations, and systems can be seen as properties, or vice versa, with many useful consequences for compositional and assume-guarantee verification techniques. For all these reasons, we call these classes *perfect*.

The regular languages are perfect but, since because the disjointness problem for the context-free languages (CFL) is undecidable (see [1]), no class containing CFL can be perfect. This "context-free barrier" restricts the search for perfect classes to those properly contained in CFL or incomparable with them, and both possibilities have been investigated. In a seminal paper [2], Alur and Madhusudan proved that the visibly pushdown languages—-a subclass of CFL—are perfect, a result that lead to a very successful theory and efficient algorithms (see e.g.[2]). Later La Torre, Madhusudan, and Parlato discovered a perfect class incomparable with CFL: the languages recognized by multi-stack visibly pushdown automata whose computations can be split into a fixed number of stages during which at most one stack is popped [3].

The "context-free barrier" continues to be a serious obstacle in many applications, in particular in the verification of concurrent systems. For this reason, many tools only check a subset of the executions of the system. Intuitively, they direct a *spotlight* to a subset of the possible executions, and check whether the executions under the spotlight satisfy the property. The spotlight is controlled by the user, who can freely move it around to check different subsets, and conventional verification corresponds to a spotlight that illuminates all the space of possible executions. In particular, the "spotlight principle" is applied by bounded model-checkers, which unroll program loops and recursion up to a fixed depth (often after taking the product of the program with an automaton for the property to be checked), leaving a system whose executions have a fixed bounded length (see e.g. [4], [5]). It is also used by *context-bounded* checkers for multi-threaded programs [6], [7], [8], which only examine executions containing at most a fixed number of context-switches (communication events between threads). Context-bounded checkers break the context-free barrier, but at the price of only exploring finite action sequences.[1] Recently, building on ideas by Kahlon [9] on bounded languages [10], context-bounded checking has been extended to *bounded verification* [11],[2] which checks

[1] More precisely, in automata-theoretic terms context-bounded checkers explore runs of S of arbitrary length, but containing only a fixed number of non-ε transitions.

[2] In [11] bounded verification was called pattern-based verification, but, since pattern is a rather generic term, we opt for bounded verification here.

whether executions of the system of the form $w_1^* \dots w_k^*$ for some finite words $w_1, \dots, w_k$ satisfy a property.

In automata-theoretic terms, the spotlight principle corresponds to *verification modulo a language*. The inclusion check $L(S) \subseteq L(P)$ and the disjointness check $L(S) \cap L(P) = \emptyset$ are replaced by checks $L_M(S) \subseteq L_M(P)$ and $L_M(S) \cap L_M(P) = \emptyset$, respectively, where L_M denotes $L \cap M$ for a language M specifying the subset of interest. Context-bounded checking corresponds to verification modulo the language of all words up to a fixed length, and bounded verification to verification modulo a bounded expression.

Verification modulo a language M allows to break the context-free barrier, which raises the question of identifying perfect classes *modulo language classes*. Given a Boolean operation $Op(L_1, \dots, L_n)$ on languages, let us define the same operation modulo a language M by $Op_M(L_1, \dots, L_n) = Op(L_1 \cap M, \dots, L_n \cap M) \cap M$, and, similarly, let us say that an automaton A is empty modulo M if $L(A) \cap M = \emptyset$. Let $\mathcal{L}$ and $\mathcal{C}$ be classes of languages. We call *$\mathcal{L}$ perfect modulo $\mathcal{C}$* if it is closed under Boolean operations modulo any $M \in \mathcal{C}$, and has a decidable emptiness problem modulo any $M \in \mathcal{C}$. It is easy to see that the recursive languages are perfect modulo the finite languages. But for bounded expressions the question becomes harder. The disjointness problem modulo a bounded expression is decidable for CFL [10], which hints at a perfect class modulo bounded expressions containing CFL. However, CFL itself is not perfect modulo bounded expressions, because it is not closed under intersection: there is no CFL L such that $\{a^n b^n c^* \mid n \geq 0\} \cap \{a^* b^n c^n \mid n \geq 0\} \cap a^* b^* c^* = L \cap a^* b^* c^*$.

In this paper we present a perfect class modulo bounded expressions: the languages recognized by *multihead pushdown automata* (MHPDA). This result is of interest, because the class has a simple and purely syntactic definition, and as we demonstrate, is expressive enough to capture many well-known models. We also characterize the complexity of the emptiness check and the Boolean operations modulo bounded expressions: we show that the emptiness check is coNEXPTIME-complete, union and intersection are polynomial, and complementation is at most triply exponential. Surprisingly, the emptiness problem is coNP-complete (and complementation doubly exponential) for the subclass of *letter-bounded* expressions, in which each string $w_1, \dots, w_k$ in the bounded language is a single letter. We also show that finite unions of bounded expressions are a maximal class of regular languages for which perfection can be attained for MHPDAs, any additional language leads to undecidability of emptiness.

In the second part of the paper, we show that several central automata models of software can be encoded into MHPDA. Encoding recursive multithreaded programs to MHPDA is obvious, since the intersection of CFLs is MHPDA-definable, and we subsume the results of Esparza and Ganty [11]. Additionally, we supply encodings for recursive counter machines (CM), the main automata-theoretic model of procedural programs with integer variables, and for finite-state machines communicating through unbounded perfect FIFO channels (CFSM), the most popular model for the verification of communication protocols. While the existence of some encoding is not surprising, since emptiness problems for CM, CFSM, and MHPDA are all co-r.e. complete, our encodings exhibit only a small polynomial blowup, and, perhaps more importantly, preserve bounded behaviours. More precisely, using our encodings we reduce *bounded control-state reachability* for CM and CFSM—deciding reachability of a given control state by means of a computation conforming to a bounded expression—to non emptiness of MHPDA modulo bounded expression. As a consequence, we prove that bounded control-state reachability for both CM and CFSM are NP-complete. The NP-completeness also extends to *unrestricted* control-state reachability for *flat* CM and *flat* CFSM, because by construction their computations conform to a bounded expression. (See, e.g., [12] and [13] for a study of these models). More generally, our language-based approach provides a uniform framework for the verification of models using auxiliary storage like counters, queues or a mix of both as defined in [14]. Incidentally, our framework allows to uniformly derive optimal complexity upper bounds for models manipulating counters, queues, or both, and shared memory multithreaded programs.

Related work. Multi-tape and multi-head finite-state and pushdown machines were extensively studied in the 1960's and 1970's, e.g. [15], [16], [17]. The decidability of emptiness for MHPDA modulo bounded languages was proved by Ibarra in [16], using previous results going back to his PhD thesis [15]. Our proof settles the complexity of the problem (coNEXPTIME-complete). Additionally, our constructions show the surprising coNP-completeness result for letter-bounded expressions. (A similar coNP-completeness result was recently obtained in [18], but for a different model.)

Reversal bounded counter machine accepting bounded languages (see e.g. [19]) and bounded Parikh automata [20] have the same expressive power as MHPDA modulo bounded expressions (they all recognize the languages of the form $\{w_1^{k_1} \dots w_n^{k_n} \mid (k_1, \dots, k_n) \in S\}$ for some semilinear set S). These three characterizations of the same class complement each other. While MHPDAs have the modeling advantage of directly encoding recursive procedures, queues and counters, reversal bounded counter machines (and by extension flat counter machines) have good algorithmic and tool support (see, e.g., [18][21]). Our results show that these algorithms and tools can be applied to a larger range of problems.

II. Preliminaries

Language theory. An alphabet Σ is a finite and non-empty set of letters. We write Σ^* for the set of finite words over Σ, and write ε for the empty word.

We assume the reader is familiar with the basics of language theory, such as regular languages, context-free languages (CFL), context-sensitive languages (CSL), and the formalisms to describe them: nondeterministic finite automata (NFA), context-free grammars (CFG), pushdown automata (PDA), etc. (see, e.g., [1]). For NFAs, CFGs, and PDAs, the size of their encoding (denoted using $|\cdot|$) is the number of bits required to represent them.

Parikh images. For $k \in \mathbb{N}$, we write $\mathbb{Z}^k$ and $\mathbb{N}^k$ for the sets of (k-dim) vectors of integers and naturals, $\mathbf{0}$ for $(0, \ldots, 0)$, and $\mathbf{e}_i$ for the vector $(z_1, \ldots, z_k) \in \mathbb{N}^k$ such that $z_j = 1$ if $j = i$ and $z_j = 0$ otherwise. *Addition* and *equality* on k-dim vectors are defined pointwise. Define $\boldsymbol{x} \times \boldsymbol{y} = (x_1, \ldots, x_n, y_1, \ldots, y_m)$ where $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$.

Given a fixed linear order $\Sigma = \{a_1, \ldots, a_n\}$, the *Parikh image* of $a_i \in \Sigma$, written $\mathsf{Parikh}^\Sigma(a_i)$, is the vector $\mathbf{e}_i$. The Parikh image is extended to words by defining $\mathsf{Parikh}^\Sigma(\varepsilon) = \mathbf{0}$ and $\mathsf{Parikh}^\Sigma(u \cdot v) = \mathsf{Parikh}^\Sigma(u) + \mathsf{Parikh}^\Sigma(v)$, and to languages by defining $\mathsf{Parikh}^\Sigma(L) = \{\mathsf{Parikh}^\Sigma(w) \mid w \in L\}$ for $L \subseteq \Sigma^*$. We omit Σ when it is clear from context.

Presburger Formulas. A *term* is a constant $c \in \mathbb{N}$, a variable x from a set X of variables, or an expression of the form $t_1 + t_2$ or $t_1 - t_2$, where t_1, t_2 are terms. A *Presburger formula* is an expression of the form $t \sim 0$, where t is a term and $\sim \in \{\leq, <, =, \neq, >, \geq\}$, or of the form $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\exists x.\, \phi$, $\forall x.\, \phi$, where ϕ, ϕ_1, ϕ_2 are Presburger formulas. Given a Presburger formula ϕ with free variables $x_1, \ldots, x_k$ (written $\phi(x_1, \ldots, x_k)$), we denote by $[\![\phi]\!]$ the set $\{(n_1, \ldots, n_k) \in \mathbb{N}^k \mid \phi(n_1, \ldots, n_k) \text{ is true}\}$, where $\phi(n_1, \ldots, n_k)$ denotes the formula without free variables obtained by substituting n_i for x_i. The size of a formula is defined as the number of symbols to write the formula when all constants are written in unary. Next, we recall a result.

Theorem 1: **[22]** *For each CFG G, there is a computable existential Presburger formula Φ of size $O(|G|)$ such that* $\mathsf{Parikh}(L(G)) = [\![\Phi]\!]$.

Bounded expressions. A *bounded expression* $\bar{\boldsymbol{w}}$ over Σ is a regular expression of the form $w_1^* \ldots w_n^*$ such that $n \geq 1$ and w_i is a non-empty word over Σ for each $i \in [1, n]$.[3] With abuse of notation, we write $\bar{\boldsymbol{w}}$ for $L(\bar{\boldsymbol{w}})$. The *size* of a bounded expression $\bar{\boldsymbol{w}}$ is defined as $|\bar{\boldsymbol{w}}| = \sum_{i=1}^n |w_i|$. A bounded expression is *letter-bounded* if $|w_1| = \ldots = |w_n| = 1$, where the w_is are not necessarily distinct.

Shuffle and indexed shuffle. The *shuffle* of two words $x, y \in \Sigma^*$ is the language

$$x \,\Delta\, y = \{x_1 y_1 \ldots x_n y_n \in \Sigma^* \mid \text{each } x_i, y_i \in \Sigma^* \\ \text{and } x = x_1 \cdots x_n \wedge y = y_1 \cdots y_n\}$$

and the shuffle of two languages $L_1, L_2 \subseteq \Sigma^*$ is the language $L_1 \,\Delta\, L_2 = \cup\{x \,\Delta\, y \mid x \in L_1, y \in L_2\}$. Shuffle is associative, hence we can write $L_1 \,\Delta \ldots \Delta\, L_k$ or $\Delta_{i=1}^k L_i$ for short.

Given $i > 0$, define $\Sigma \bowtie i = \{\langle \sigma, i\rangle \mid \sigma \in \Sigma\}$. We say that i is the *index* of $\langle \sigma, i\rangle$. We extend indexing to words and languages in the natural way: for $w = b_1 \ldots b_t \in \Sigma^*$, $L \subseteq \Sigma^*$, and $i > 0$, $(w \bowtie i) = \langle b_1, i\rangle \cdots \langle b_t, i\rangle$ and $L \bowtie i = \{w \bowtie i \mid w \in L\}$. The *indexed shuffle* of $L_1, \ldots, L_k$ is the language

$$\underline{\Delta}_{i=1}^k L_i = \Delta_{i=1}^k (L_i \bowtie i) \ .$$

For example, if we shorten $\langle a, 1\rangle$ to $a1$ etc., we have

$$\{ab\} \underline{\Delta} \{b\} = \{a1\, b1\} \,\Delta\, \{b2\} = \{a1\, b1\, b2, a1\, b2\, b1, b2\, a1\, b1\} \ .$$

It is well known that if L_i is recognized by a NFA of size n_i, then both $\Delta_{i=1}^k L_i$ and $\underline{\Delta}_{i=1}^k L_i$ are recognized by NFAs of size $O(\Pi_{i=1}^k n_i)$.

III. Models

A *tape content* (or simply tape) w over Σ is a word $w \in \Sigma^*$. For $d \geq 1$, a *d-tuple of tapes* is a d-tuple $(w_1, \ldots, w_d)$ where each w_i is a tape. For $w \in \Sigma^*$, define $[w]^d$ as the d-tuple $(w, \ldots, w)$. For a language $L \subseteq \Sigma^*$, define $[L]^d$ as the set of d-tuples of tapes $\{(w_1, \ldots, w_d) \mid w_i \in L\}$.

Definition 1: A d-tape pushdown automaton (d-TPDA, for short) is a 9-tuple $A = \langle S, \Sigma, \$, \Gamma, M, \nu, s_0, \bot, F\rangle$ where

1) S is a finite non-empty set of *states*,
2) Σ is the *tape alphabet*,
3) \$ is a symbol not in Σ (the *endmarker* for the tape),
4) Γ is the *stack alphabet*,
5) M, the *set of transitions*, is a mapping from $S \times (\Sigma \cup \{\$\} \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $S \times \Gamma^*$,
6) $\nu \colon S \to [1, d]$ is the *tape selector function*,
7) $s_0 \in S$ is the *start state*,
8) $\bot \in \Gamma$ is the *bottom stack symbol*,
9) $F \subseteq S$ is the set of *final states*.

Intuitively, a d-TPDA has a finite-state control (a state from S), d input tapes, and a stack. There is a separate input-reading head on each tape. Each state $s \in S$ in the finite state control reads from the tape given by $\nu(s)$ and pops the top of the stack. The transition relation then non-deterministically determines the new control state and the sequence of symbols pushed on to the stack. The read head on tape $\nu(s)$ moves one step to the right.

For readability, we write $(s, \gamma) \overset{[\sigma\rangle_i}{\hookrightarrow} (s', w)$ whenever $(s', w) \in M(s, \sigma, \gamma)$ and $\nu(s) = i$. We omit the tape number i if it is not important.

The size $|A|$ of a d-TPDA A is given by $|S| + |\Sigma| + |\Gamma| + |M| + |\nu|$, where in the encoding of the function ν, the numbers in $[1, d]$ are encoded in binary. Intuitively, $|A|$ is proportional to the number of bits required to represent a d-TPDA when numbers are represented in binary.

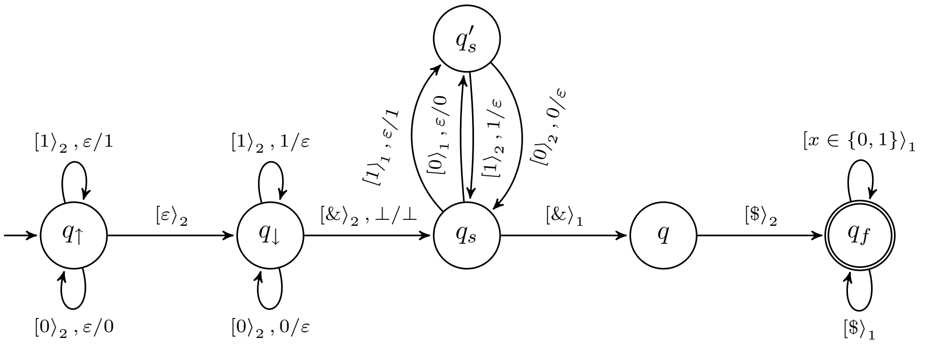
Let us fix a d-TPDA $A = \langle S, \Sigma, \$, \Gamma, M, \nu, s_0, \bot, F\rangle$.

Definition 2: Let # be a symbol distinct from symbols in $\Sigma \cup \{\$\}$. Define $\mathfrak{T} = \{w\#w' \mid w \cdot w' \in \Sigma^*\$\}$. An *instantaneous description* (ID) of A is a triple $(s, \bar{t} = \langle t_1, \ldots, t_d\rangle, w) \in S \times [\mathfrak{T}]^d \times \Gamma^*$. An ID $(s, \bar{t}, w)$ denotes that A is in state s, with pushdown store content $w \in \Gamma^*$, and where $\bar{t} = \langle t_1, \ldots, t_d\rangle$ is such that $t_i \in \mathfrak{T}$ gives the configuration of tape i where the position of the head indicated by #.

Definition 3: We define a binary relation $\vdash$, between IDs as follows. Let $c = (s, \bar{t}, w\gamma)$ and $c' = (s', \bar{t}', ww')$ be two IDs. We have $c \vdash c'$ iff each of following conditions is satisfied:

1) $(s, \gamma) \overset{\sigma_r}{\hookrightarrow} (s', w')$ for some $\sigma_r \in \Sigma \cup \{\varepsilon, \$\}$.
2) $t_{\nu(s)} = x\#\sigma_r y$ and $t'_i = \begin{cases} x\sigma_r\#y & \text{if } i = \nu(s) \\ t_i & \text{else} \end{cases}$

[3] For integers $x \leq x'$, we write $[x, x']$ for the set $\{i \in \mathbb{Z} \mid x \leq i \leq x'\}$.



$[1\rangle_2, \varepsilon/1$
$[0\rangle_2, \varepsilon/0$
$q_\uparrow$
$[\varepsilon\rangle_2$
$[1\rangle_2, 1/\varepsilon$
$[0\rangle_2, 0/\varepsilon$
$q_\downarrow$
$[\&\rangle_2, \bot/\bot$
q'_s
$[1\rangle_1, \varepsilon/1$
$[0\rangle_1, \varepsilon/0$
$[1\rangle_2, 1/\varepsilon$
$[0\rangle_2, 0/\varepsilon$
q_s
$[\&\rangle_1$
q
$[\$\rangle_2$
$[x \in \{0,1\}\rangle_1$
q_f
$[\$\rangle_1$

Fig. 1. 2-HPDA accepting $\{w\&w \mid w \in \{0,1\}^*$ and w is a palindrome$\}$.

Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$.

We now introduce helper functions Lft and Rgt which given an ID c and a tape $h \in [1,d]$ returns the tape content lying to the left and to the right, respectively.

Definition 4: Let $c = (s, \bar{t} = \langle t_1, \ldots, t_d\rangle, w)$ be an ID and $h \in [1,d]$. Assume $t_h = w_1 \# w_2$. Define $\mathsf{Rgt}(c,h) = w_2$ and $\mathsf{Lft}(c,h) = w_1$.

Given an ID c, we say that a head i is *off its tape* in c whenever $\mathsf{Rgt}(c,i) = \varepsilon$. Define next the languages of d-TPDA.

Definition 5: An ID $c = (s, \bar{t}, w)$ is *accepting* iff $s \in F$ and for every $i \in [1,d]$ head i is off its tape in c. A d-tuple of tapes $(x_1, \ldots, x_d) \in [\Sigma^*]^d$ is *accepted* by A if $(s_0, \langle \# x_1 \$, \ldots, \# x_d \$\rangle, \bot) \vdash^* (s, \bar{t}, w)$ for some accepting ID $(s, \bar{t}, w)$. The set of d-tuple of tapes accepted by A is denoted $T(A)$. A subset $L \subseteq [\Sigma^*]^d$ is d-TPDA *definable* if there exists some d-TPDA A such that $L = T(A)$.

Remark 1:

- Having each head off its tape is a necessary condition for acceptance. Therefore any accepting run (even for input $[\varepsilon]^d$) needs to do at least one read on each tape because of \$. This implies that for any non-trivial d-TPDA, $d \leq |S|$.
- Every d-TPDA definable language is recursive for each $d > 0$ [15]. The 1-TPDA definable languages are the CFLs. However, it distinguishes from the classical definition (see [1]) since, to accept, a 1-TPDA needs to be in a final state and the head to be off the tape.

We now introduce a generalization of pushdown automata with several *heads* working on a shared tape. This model is closely related to d-TPDA as described below.

Definition 6: Let $\Delta_d \subseteq [\Sigma^*]^d$ be given by $\{(w_1, \ldots, w_d) \in [\Sigma^*]^d \mid w_1 = \cdots = w_d\}$ and let $\pi_1 \colon [\Sigma^*]^d \to \Sigma^*$ be the function which maps $L \subseteq [\Sigma^*]^d$ onto the first tape: $\pi_1(L) = \{w_1 \in \Sigma^* \mid \exists w_2, \ldots, w_d . (w_1, \ldots, w_d) \in L\}$.

When the d-tuple of tapes is restricted to Δ_d, that is, when all the tapes have identical content, we can view A as a pushdown automaton with d-heads sharing a unique tape. In this case we define the language $L \subseteq \Sigma^*$ accepted by the d-head pushdown automaton (d-HPDA, for short) A to be $\pi_1(T(A) \cap \Delta_d)$ and we denote this language by $L(A)$. We write MHPDA for the class of models d-HPDA for $d \geq 1$.

In the following, we use an intuitive graphical notation for MHPDAs. Fig. 1 gives an example of a 2-HPDA which recognizes a language over symbols $\{0, 1, \&\}$ given by $\{w\&w \mid w \in \{0,1\}^*$ and w is a palindrome$\}$. Intuitively, in $q_\uparrow$ and $q_\downarrow$, the 2-HPDA uses head 2 to recognize the first palindrome using its stack. When head 2 reads & the MHPDA enters q_s in which it checks using both heads that the subwords before and after & are identical. If the check succeeds then the MHPDA enters q then q_f (head 2 has fallen off the tape) where it accepts after making head 1 fall off the tape. The transition from $q_\downarrow$ to q_s labelled $[\&\rangle_2, \bot/\bot$ reads as follows: if in state $q_\downarrow$, the bottom stack symbol $\bot$ is on the top of the stack then read & with head 2 and update the location to q_s. Also read the transition from q_f to itself and labelled $[x \in \{0,1\}\rangle_1$ as follows: in state q_f read any symbol of $\{0,1\}$, go to q_f, and do not modify the stack.

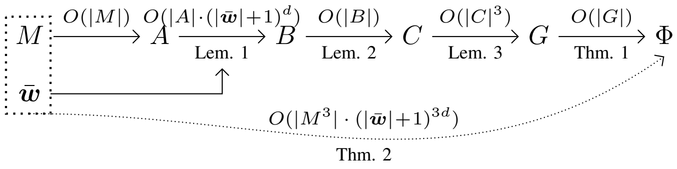


Fig. 2. Summary of the decision procedure steps.

A similar construction shows that a 2-HPDA (indeed, without using its stack) can recognize accepting runs of a Turing machine [23]. Thus, the emptiness problem for d-HPDA is undecidable for $d > 1$.

IV. Emptiness modulo Bounded Expressions

Given a d-HPDA M and a bounded expression $\bar{\boldsymbol{w}} = w_1^* \ldots w_n^*$, both over an alphabet Σ, we show how to check emptiness of $L(M) \cap \bar{\boldsymbol{w}}$. Notice that we can construct a d-TPDA A of size $O(|M|)$ such that $L(M) \cap \bar{\boldsymbol{w}} = \varnothing$ iff $T(A) \cap [\bar{\boldsymbol{w}}]^d \cap \Delta_d = \varnothing$, where Δ_d is the set of d-tuples of the form $(w, w, \ldots, w)$ (see Def. 6).

In Section IV-A we show that emptiness of $T(A) \cap [\bar{\boldsymbol{w}}]^d$ reduces to emptiness of a context-free grammar, and in Section IV-B that emptiness of $L(M) \cap \bar{\boldsymbol{w}}$ reduces to unsatisfiability of an existential Presburger formula. The steps of the reduction are summarized in Fig. 2.

A. *Emptiness of* $T(A) \cap [\bar{\boldsymbol{w}}]^d$

We construct a context-free grammar that recognizes an emptiness-preserving "encoding" of $T(A) \cap [\bar{\boldsymbol{w}}]^d$ in three steps.

In the first step, intuitively, we construct a d-TPDA recognizing the result of applying a transformation on $T(A) \cap [\bar{\boldsymbol{w}}]^d$ which "contracts" each word w_i of $\bar{\boldsymbol{w}}$ into a single letter.

Let $\overline{\Sigma} = \{a_1, \dots, a_n\}$ be a new alphabet and let $\bar{\boldsymbol{a}} = a_1^* \cdots a_n^*$ be a bounded expression over $\overline{\Sigma}$. Given a bounded expression $\bar{\boldsymbol{w}} = w_1^* \cdots w_n^*$ over Σ, we define the mapping $f_{\bar{\boldsymbol{w}}} \colon \mathbb{N}^n \to \Sigma^*$ by $f_{\bar{\boldsymbol{w}}} \colon (i_1, \dots, i_n) \mapsto w_1^{i_1} \cdots w_n^{i_n}$.

Lemma 1: There is a computable d-TPDA B over $\overline{\Sigma}$ of size $O(|A| \cdot (|\bar{\boldsymbol{w}}|+1)^d)$ such that for every $\boldsymbol{k_1}, \dots, \boldsymbol{k_d} \in \mathbb{N}^n$:

$$\big(f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_d})\big) \in T(A) \cap [\bar{\boldsymbol{w}}]^d$$
$$\text{iff}$$
$$\big(f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_d})\big) \in T(B) \ .$$

Proof: We construct a d-TPDA B_1 such that $T(B_1) = T(A) \cap [\bar{\boldsymbol{w}}]^d$. Let W be a NFA of size $O(|\bar{\boldsymbol{w}}| + 1)$ recognizing $\bar{\boldsymbol{w}} \cdot \$$, and let Q_W be its set of states and $F_W \subseteq Q_W$ the accepting ones. Adapting the shuffle construction for NFAs, we can construct a NFA W^d with states $Q_{W^d} = (Q_W)^d$ (the d-times Cartesian product of Q_W) recognizing $\underline{\Delta}_{i=1}^d L(\bar{\boldsymbol{w}} \cdot \$)$. We synchronize A with W^d as follows. The set of states of B_1 is $S \times (Q_W)^d$, where S is the set of states of A, and the set of final states is $F \times (F_W)^d$. The tape selection function of B_1 is determined by the one of A. If A has a transition $(s_a, \gamma) \overset{[\sigma\rangle_\ell}{\hookrightarrow} (s_b, w)$, where $\sigma \in \Sigma \cup \{\$\}$ is read from the tape $\ell = \nu(s_a)$, and W^d has a transition $\langle q_1, \dots, q_\ell, \dots, q_d\rangle \overset{\langle\sigma,\ell\rangle}{\rightarrow} \langle q_1, \dots, q'_\ell, \dots, q_d\rangle$, then B_1 has a transition $(\langle s_a, q_1, \dots, q_\ell, \dots, q_d\rangle, \gamma) \overset{[\sigma\rangle_\ell}{\hookrightarrow} (\langle s_b, q_1, \dots, q'_\ell, \dots, q_d\rangle, w)$. If A has a transition $(s_a, \gamma) \overset{[\varepsilon\rangle_\ell}{\hookrightarrow} (s_b, w)$ (resp. W^d has a transition $\langle q_1, \dots, q_j, \dots, q_d\rangle \overset{\langle\varepsilon,j\rangle}{\rightarrow} \langle q_1, \dots, q'_j, \dots, q_d\rangle$), then B_1 has transition $(\langle s_a, q_1, \dots, q_d\rangle, \gamma) \overset{[\varepsilon\rangle_\ell}{\hookrightarrow} (\langle s_b, q_1, \dots, q_d\rangle, w)$ (resp. $(\langle s_a, q_1, \dots, q_j, \dots, q_d\rangle, \gamma) \overset{[\varepsilon\rangle_\ell}{\hookrightarrow} (\langle s_a, q_1, \dots, q'_j, \dots, q_d\rangle, \gamma)$ for every $\gamma \in \Gamma$). B_1 has no further transitions.

Now we construct B. It is easy to construct W so that for every word w_i of $\bar{\boldsymbol{w}}$ it contains a state q_{w_i} that is entered every time (and only when) W reads the last letter of w_i. We proceed as follows. First, we transform all transitions of B_1, with the exception of those labeled with endmarkers, into ε-transitions. Then, we relabel again all transitions entering q_{w_i}, i.e., all transitions in which some copy of W takes a transition with target q_{w_i}: we replace ε by a_i. ■

In a second step we construct a PDA that recognizes interleavings of the different tapes in the order in which they are read by B. Let $\overline{\Sigma}_d = \bigcup_{i=1}^d (\overline{\Sigma} \bowtie i)$ with linear order $\langle a_1, 1\rangle\langle a_2, 1\rangle \dots \langle a_n, d\rangle$. Given a d-tuple of tapes $u = (u_1, \dots, u_d)$ define $\underline{\Delta}(u) = \underline{\Delta}_{i=1}^d \{u_i\}$.

Lemma 2: There is a computable PDA C over $\overline{\Sigma}_d$ of size $O(|B|)$ such that $u \in T(B)$ iff $\underline{\Delta}(u) \cap L(C) \neq \varnothing$ for every $u \in [\overline{\Sigma}^*]^d$. Also $L(C) \subseteq \underline{\Delta}_{i=1}^d \bar{\boldsymbol{a}}$.

Proof: B and C have the same states, initial and final states, and stack alphabets. Assume B is currently at state s, and the tape selector assigns to s tape number $\ell = \nu(s)$. The transitions of C are defined so that if in the next move B reads a letter σ, then C reads the letter $\langle\sigma, \ell\rangle$ (unless $\sigma \in \{\$, \varepsilon\}$, in which case C reads ε). Formally, for $\sigma \neq \$$ and $\sigma \neq \varepsilon$ the PDA C has a transition $(s, \gamma) \overset{\langle\sigma,\ell\rangle}{\hookrightarrow} (s', w)$ iff B has a transition $(s, \gamma) \overset{[\sigma\rangle_\ell}{\hookrightarrow} (s', w)$, and C has a transition $(s, \gamma) \overset{\varepsilon}{\hookrightarrow} (s', w)$ iff B has a transition $(s, \gamma) \overset{[\$\rangle_\ell}{\hookrightarrow} (s', w)$ or $(s, \gamma) \overset{[\varepsilon\rangle_\ell}{\hookrightarrow} (s', w)$. Now, C accepts the word of $\underline{\Delta}(u)$ that interleaves the different tapes in the order in which they are read by B. ■

The third step converts from PDAs to CFGs [1].

Lemma 3: [1] There is a computable CFG G over $\overline{\Sigma}_d$ of size $O(|C|^3)$ such that $L(G) = L(C)$.

Proposition 1 summarizes the three steps.

Proposition 1: There is a computable CFG G over $\overline{\Sigma}_d$ of size $O(|A|^3 \cdot (|\bar{\boldsymbol{w}}|+1)^{3d})$ such that $L(G) \subseteq \underline{\Delta}_{i=1}^d \bar{\boldsymbol{a}}$ and for every $\boldsymbol{k_1}, \dots, \boldsymbol{k_d} \in \mathbb{N}^n$:

$$\big(f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_d})\big) \in T(A) \cap [\bar{\boldsymbol{w}}]^d$$
$$\text{iff}$$
$$\underline{\Delta}\big(f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_d})\big) \cap L(G) \neq \varnothing \ .$$

B. *Emptiness of* $L(M) \cap \bar{\boldsymbol{w}}$

Recall that $L(M) \cap \bar{\boldsymbol{w}} = \varnothing$ iff $T(A) \cap [\bar{\boldsymbol{w}}]^d \cap \Delta_d = \varnothing$. To decide this problem, we rely on the notion of Parikh image.

Lemma 4: Let G be a CFG over $\overline{\Sigma}_d$ such that $L(G) \subseteq \underline{\Delta}_{i=1}^d \bar{\boldsymbol{a}}$. Then for every $\boldsymbol{k_1}, \dots, \boldsymbol{k_d} \in \mathbb{N}^n$ $\underline{\Delta}\big(f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_d})\big) \cap L(G) \neq \varnothing$ iff $\boldsymbol{k_1} \times \cdots \times \boldsymbol{k_d} \in \mathsf{Parikh}(L(G))$.

Proof: The left-to-right follows from $\underline{\Delta}$ which define a set of words, all with the same Parikh image. For the converse, observe that since $L(G) \subseteq \underline{\Delta}_{i=1}^d \bar{\boldsymbol{a}}$ any Parikh image of $L(G)$ is necessarily the result of applying the indexed shuffle on d words of $\bar{\boldsymbol{a}}$ which must be $f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_d})$. ■

Combining Prop. 1, Lem. 4, and Thm. 1, we get:

Proposition 2: There is a computable existential Presburger formula Φ with free variables $\{x_{ij} \mid i \in [1, n], j \in [1, d]\}$ of size $O(|A|^3 \cdot (|\bar{\boldsymbol{w}}|+1)^{3d})$ such that for every $\boldsymbol{k_1}, \dots, \boldsymbol{k_d} \in \mathbb{N}^n$

$$\big(f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_d})\big) \in T(A) \cap [\bar{\boldsymbol{w}}]^d$$
$$\text{iff}$$
$$\Phi(\boldsymbol{k_1}, \dots, \boldsymbol{k_d}) \text{ is true.}$$

Proof: Take G as in Prop. 1 and Φ as in Thm. 1. Then:

$$\begin{array}{ll}
\big(f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_d})\big) \in T(A) \cap [\bar{\boldsymbol{w}}]^d & \\
\text{iff } \underline{\Delta}\big(f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_d})\big) \cap L(G) \neq \varnothing & \text{Prop. 1} \\
\text{iff } \boldsymbol{k_1} \times \cdots \times \boldsymbol{k_d} \in \mathsf{Parikh}(L(G)) & \text{Lem. 4} \\
\text{iff } \Phi(\boldsymbol{k_1}, \dots, \boldsymbol{k_d}) \text{ is true} & \text{Thm. 1}
\end{array}$$
■

The advantage of Prop. 2 is that it can be easily extended to a procedure for checking not only emptiness of $T(A) \cap [\bar{\boldsymbol{w}}]^d$, but also emptiness of $T(A) \cap [\bar{\boldsymbol{w}}]^d \cap \Delta_d$. Recall that the tuples in $T(A) \cap [\bar{\boldsymbol{w}}]^d \cap \Delta_d$ are those d-tuples of $T(A)$ of the form $(w, \dots, w)$ where $w \in \bar{\boldsymbol{w}}$. Next, there exists $\boldsymbol{k_1}, \dots, \boldsymbol{k_d} \in \mathbb{N}^n$

$$\begin{aligned}
&\big(f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{w}}}(\boldsymbol{k_d})\big) \in T(A) \cap [\bar{\boldsymbol{w}}]^d \cap \Delta_d \\
&\text{iff (property of } \Delta_d, , \boldsymbol{k} = \boldsymbol{k_1} = \dots = \boldsymbol{k_d}) \\
&\quad \exists \boldsymbol{k} \colon [f_{\bar{\boldsymbol{w}}}(\boldsymbol{k})]^d \in T(A) \cap [\bar{\boldsymbol{w}}]^d \\
&\text{iff (Prop. 2)} \\
&\quad \exists \boldsymbol{k}.\, \Phi(\underbrace{\boldsymbol{k}, \dots, \boldsymbol{k}}_{d \text{ times}}) \text{ is true} \\
&\text{iff } \exists \{x_{ij}\}_{i \in [1,n]}^{j \in [1,d]}.\, \big(\Phi \wedge \textstyle\bigwedge_{i=1}^{n} x_{i1} = \dots = x_{id}\big) \text{ is true}
\end{aligned}$$

where Φ is the formula of Prop. 2. Let $\Psi(x_1, \dots, x_n)$ be

$$\exists \{x_{ij}\}_{i \in [1,n]}^{j \in [2,d]}.\, \big(\Phi \wedge \textstyle\bigwedge_{i=1}^{n} x_{i1} = \dots = x_{id}\big)$$

we get:

Theorem 2: There is a computable formula $\Psi(x_1, \dots, x_n)$ of existential Presburger arithmetic of size $O(|M|^3 \cdot (|\bar{\boldsymbol{w}}|{+}1)^{3d})$ such that $f_{\bar{\boldsymbol{w}}}(k_1, \dots, k_n) \in L(M) \cap \bar{\boldsymbol{w}}$ iff $\Psi(k_1, \dots, k_n)$ is true. In particular, $L(M) \cap \bar{\boldsymbol{w}} \neq \varnothing$ iff Ψ is satisfiable.

This theorem admits a simple but useful generalization.

Theorem 3: Let $\{M_i\}_{i \in [1,q]}$ be a family of MHPDA such that M_i is a c_i-HPDA for each $i \in [1, q]$. Let $c = \max\big(\{c_i\}_{i \in [1,q]}\big)$ and $m = \max\big(\{|M_i|\}_{i \in [1,q]}\big)$. There is a computable formula $\Psi(x_1, \dots, x_n)$ of existential Presburger arithmetic of size $O(q \cdot m^3 \cdot (|\bar{\boldsymbol{w}}| + 1)^{3c})$ such that $f_{\bar{\boldsymbol{w}}}(k_1, \dots, k_n) \in \bigcap_{i=1}^{q} L(M_i) \cap \bar{\boldsymbol{w}}$ iff $\Psi(k_1, \dots, k_n)$ is true.

Proof: Define $\Psi(x_1, \dots, x_n)$ to be $\bigwedge_{i=1}^{q} \Psi_i(x_1, \dots, x_n)$ such that each $\Psi_i(x_1, \dots, x_n)$ is the formula obtained by Thm. 2 on input M_i and $\bar{\boldsymbol{w}}$. Correctness is proved as follows:

$$\begin{aligned}
&f_{\bar{\boldsymbol{w}}}(k_1, \dots, k_n) \in \textstyle\bigcap_{i=1}^{q} L(M_i) \cap \bar{\boldsymbol{w}} \\
&\text{iff } \textstyle\bigwedge_{i=1}^{q} f_{\bar{\boldsymbol{w}}}(k_1, \dots, k_n) \in L(M_i) \cap \bar{\boldsymbol{w}} \\
&\text{iff } \textstyle\bigwedge_{i=1}^{q} \Psi_i(k_1, \dots, k_n) \text{ is true} && \text{Thm. 2} \\
&\text{iff } \Psi(k_1, \dots, k_n) \text{ is true} && \text{def. of } \Psi
\end{aligned}$$

We conclude from Thm. 2 that $|\Psi_i| = O(m \cdot (|\bar{\boldsymbol{w}}|{+}1)^{3c})$ for each $i \in [1, q]$, hence that $|\Psi| = O(q \cdot m \cdot (|\bar{\boldsymbol{w}}|{+}1)^{3c})$. ■

C. Complexity

We prove that emptiness modulo a bounded expression is coNEXPTIME-complete. From Thm. 2 and because satisfiability of existential Presburger formulas is in NP [24], we get an upper bound of coNEXPTIME for the emptiness problem of MHPDA modulo a bounded expression. In [25], we show the problem is coNEXPTIME-hard already for $\bar{\boldsymbol{w}} = (01)^*$ by reduction from the succinct Knapsack problem.

Theorem 4: The emptiness problem for MHPDAs modulo an arbitrary bounded expression is in coNEXPTIME. Moreover, the emptiness problem for MHPDAs and $\bar{\boldsymbol{w}} = (01)^*$ is coNEXPTIME-hard.

Remarkably, the emptiness problem is only coNP-complete for letter-bounded expressions. Fix a letter-bounded expression $\bar{\boldsymbol{b}} = b_1^* \dots b_n^*$ where b_i's are not necessarily distinct. The key to the result is that Lem. 1 (with $\bar{\boldsymbol{w}}$ now equal to $\bar{\boldsymbol{b}}$) can be replaced by the following one.

Lemma 5: There is a family $\{B_i\}_{i=1}^{\alpha}$ of d-TPDAs over $\overline{\Sigma}$, where $\alpha = d^{(|\bar{\boldsymbol{b}}|+1)d}$ and each B_i has size $O(|A| \cdot (|\bar{\boldsymbol{b}}|{+}1) \cdot d^2)$, such that for every $\boldsymbol{k_1}, \dots, \boldsymbol{k_d} \in \mathbb{N}^n$ we have

$$\big(f_{\bar{\boldsymbol{b}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{b}}}(\boldsymbol{k_d})\big) \in T(A) \cap [\bar{\boldsymbol{w}}]^d$$
$$\text{iff}$$
$$\big(f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_1}), \dots, f_{\bar{\boldsymbol{a}}}(\boldsymbol{k_d})\big) \in \textstyle\bigcup_{i=1}^{\alpha} T(B_i) \ .$$

Proof: We can easily construct a NFA W recognizing $L(\bar{\boldsymbol{b}} \cdot \$)$ with states $\{q_1, \dots, q_{n+1}\}$, initial state q_1, final state q_{n+1}, and transitions $\{q_i \xrightarrow{b_i} q_i \mid i \in [1, n]\} \cup \{q_j \xrightarrow{\varepsilon} q_{j+1} \mid j \in [1, n-1]\} \cup \{q_n \xrightarrow{\$} q_{n+1}\}$. Let W^d be the NFA defined in Lem. 1 recognizing $\underline{\Delta}_{i=1}^{d} L(\bar{\boldsymbol{b}} \cdot \$)$. While W^d has $(n+1)^d$ states, it is easy to see that every accepting run of W^d only visits $(n+1) \cdot d$ *distinct* states, because every transition $\langle q_{i_1}, \dots, q_{i_d} \rangle \xrightarrow{\langle \sigma, \ell \rangle} \langle q_{j_1}, \dots, q_{j_d} \rangle$ of W^d satisfies $i_1 \leq j_1, \dots, i_d \leq j_d$. Hence, we can associate to each accepting run ρ the subset $Q_{W^d}^{\rho}$ of the states of Q_{W^d} visited by ρ, and so the sub-NFA W_{ρ}^{d} of W^d with $Q_{W^d}^{\rho}$ as set of states, and whose transitions are the transitions of W^d between states of $Q_{W^d}^{\rho}$. Clearly, W_{ρ}^{d} has at most $(n{+}1) \cdot d$ states and at most $((n+1) \cdot d) \cdot (d + d) = O(n \cdot d^2)$ transitions. (Consider a state $\langle q_{i_1}, \dots, q_{i_d} \rangle$. The term $(d + d)$ corresponds to the transitions labeled by $\langle b_{i_j}, j \rangle$ or $\langle \varepsilon, j \rangle$ for each $j \in [1, d]$.) Moreover, even though there are infinitely many accepting runs, the number of different such sub-NFAs is at most $d^{(|\bar{\boldsymbol{b}}|+1)d}$, because each state of W^d has d successors different from itself, and every accepting run of W^d only visits $(n + 1) \cdot d$ distinct states. Let $W_1^d, \dots, W_{\alpha}^d$ be an enumeration of them.

In Lem. 1 we first constructed a d-TPDA B_1 by synchronizing A and W^d, and then transformed B_1 into another d-TPDA B. Now we first synchronize A and W_i^d, yielding a d-TPDA B_i for every $i \in [1, \alpha]$, and then we apply the same transformation as in Lem. 1 to obtain a d-TPDA B_i. Clearly, we have $T(B) = \bigcup_{i=1}^{\alpha} T(B_i)$, and so the result follows. ■

From the construction in Lem. 5, we find that there is a nondeterministic polynomial-time Turing machine that takes as input A and $\bar{\boldsymbol{b}}$ and writes an arbitrary B_i from the family on to the output tape. Moreover, the range of the machine is exactly $\{B_i\}_{i=1}^{\alpha}$. The Turing machine guesses a sub-NFA W_{ρ}^{d} of W^d with $(n + 1)d$ states and $O(nd^2)$ transitions, synchronizes A and W_{ρ}^{d}, and finally applies the transformation of Lem. 1.

Proceeding as in the previous section, we now obtain a grammar G_i for each d-TPDA B_i, and from it an existential Presburger formula Ψ_i. We get:

Corollary 1: There is a computable family $\{\Psi_i(x_1, \dots, x_n)\}_{i=1}^{\alpha}$ of existential Presburger formulas, each of them of size $O(|M|^3 \cdot (|\bar{\boldsymbol{b}}| + 1)^3 \cdot d^6)$, such that $f_{\bar{\boldsymbol{b}}}(k_1, \dots, k_n) \in L(M) \cap \bar{\boldsymbol{b}}$ iff $\bigvee_{i=1}^{\alpha} \Psi_i(k_1, \dots, k_n)$ is true. In particular, $L(M) \cap \bar{\boldsymbol{b}} \neq \varnothing$ iff at least one of the formulas in the family is satisfiable.

Lem. 5 and its corollary imply the coNP upper bound.

Theorem 5: The emptiness problem for MHPDAs modulo letter-bounded expressions is in coNP. Moreover, the emptiness problem for MHPDAs and $\bar{\boldsymbol{w}} = b^*$ is coNP-hard.

Proof: Let M be a d-HPDA and let $\bar{\boldsymbol{b}}$ be a letter-bounded expression. The nondeterministic polynomial algorithm for non-emptiness of $L(M) \cap \bar{\boldsymbol{b}}$ first guesses one of the automata $\{B_i\}_{i=1}^{\alpha}$ from Lem. 5 in polynomial time, then constructs the formula Ψ_i of Cor. 1, and finally, nondeterministically checks that it is satisfiable. Since Ψ_i has polynomial size in $|M| + |\bar{\boldsymbol{b}}| + d$, the whole procedure takes nondeterministic polynomial time.

The coNP-hardness result follows from [11, Theorem 1], which proves that given CFGs $G_1, \ldots, G_k$, deciding non emptiness of $L(G_1) \cap \ldots \cap L(G_k) \cap L(b^*)$ is coNP-hard. Since we can easily construct in linear time a k-HPDA recognizing $L(G_1) \cap \ldots \cap L(G_k)$, the result follows. ■

V. Closure under Boolean operations

It is straightforward to show that MHPDAs are effectively closed under union and intersection.

Proposition 3: Let A_1 be a k_1-HPDA and A_2 a k_2-HPDA. We can construct in linear time $(k_1 + k_2)$-HPDAs $A_\cup$ and $A_\cap$ such that $L(A_\cup) = L(A_1) \cup L(A_2)$ and $L(A_\cap) = L(A_1) \cap L(A_2)$.

Proof: $A_\cup$ nondeterministically decides to simulate A_1 or A_2; it requires $\max\{k_1, k_2\}$ heads. $A_\cap$ simulates A_1 with heads $[1, k_1]$ and if A_1 reaches an accepting state, then it simulates A_2 with heads $[k_1 + 1, k_1 + k_2]$. ■

Next we show that MHPDAs are closed under complement modulo any bounded expression.

Proposition 4: Given an d-HPDA A and a bounded expression $\bar{\boldsymbol{w}}$, there is a MHPDA B such that $L(B) = \bar{\boldsymbol{w}} \setminus L(A)$ and $|B|$ is at most triply exponential in $|A|$, $|\bar{\boldsymbol{w}}|$.

Proof: The complementation procedure works as follows:

- Compute the existential Presburger formula Ψ of Thm. 2 with constants written in unary. A simple inspection of the result of [22] shows that the size of Ψ is still $O(|A|^3 \cdot (|\bar{\boldsymbol{w}}|+1)^{3d})$ (the greatest constant in Ψ is 1).
- Compute a quantifier-free formula $\Phi \equiv \neg\Psi$ (with constants written in unary) of Presburger arithmetic with divisibility predicates, i.e., predicates of the form $a|t$ (a divides t), where a is a constant and t is a term. This is possible because Presburger arithmetic with these predicates has a quantifier elimination procedure. Moreover, since Ψ has one single block of existential quantifiers, we have $|\Phi| \in \mathit{2exp}(O(|\Psi|))$ [26][27], where $\mathit{2exp}(n) = 2^{2^n}$. We have $f_{\bar{\boldsymbol{w}}}(k_1, \cdots, k_n) \in (\bar{\boldsymbol{w}} \setminus L(A))$ iff $\Psi(k_1, \ldots, k_n)$ is false iff $\Phi(k_1, \ldots, k_n)$ is true.
- Construct the MHPDA B as follows. B has a head for each atomic formula of Φ. Control ensures that heads read the input one after the other (i.e., the $i+1$-st head starts reading the input after the i-th head has completely read it). The i-th head checks whether the i-th atomic formula is satisfied by the input. For instance, an atomic formula like $k_1 + k_1 + k_1 - k_2 - k_2 \leq 5$ is checked using the stack as follows: the stack is used as a counter over the integers (using two symbols, say P and N, and encoding i where $i \geq 0$ as $P^i\bot$ and $-i$ ($i > 0$) as $N^i\bot$); B reads $w_1^{k_1} w_2^{k_2}$, so that at the end the counter contains $3k_1 - 2k_2$; then B compares the content of the counter with 5. A formula like $4|k_1 + k_1 + k_1 - k_2 - k_2$ is checked similarly: at the end B pops the stack content and checks if it is a multiple of 4. Control takes care of evaluating the formula by combining the results of the evaluation of the atomic formulas. B accepts $w_1^{k_1} \ldots w_n^{k_n}$ if the evaluation of Φ is true. Since the constants of Φ are written in unary, we have $|B| \in O(|\Phi|)$.

This procedure yields a triple exponential bound for B in the size of A. More precisely, the procedure is only triply exponential in the number of heads of A, but not on its number of states or transitions. ■

For letter-bounded expressions, we get one exponential less by using Cor. 1 to compute a family of exponentially many Presburger formulas, each polynomial in the size of the automaton and the bounded expression, then following the previous construction and noting that the intersection of exponentially many MHPDAs, each doubly exponential, still gives a doubly exponential MHPDA.

Proposition 5: Given a d-HPDA A and a letter-bounded expression $\bar{\boldsymbol{b}}$, there is a MHPDA B such that $L(B) = \bar{\boldsymbol{b}} \setminus L(A)$ and $|B|$ is at most doubly exponential in $|A|$, $|\bar{\boldsymbol{b}}|$.

VI. Optimality questions

Let $\mathcal{P}$ denote the class of finite unions of bounded expressions, let $\mathcal{F}$ denote the class of finite languages, and let $\mathcal{U} = \mathcal{P} \cup \mathcal{F}$. We have shown that MHPDA is perfect modulo $\mathcal{U}$. This raises two questions: (1) is MHPDA perfect modulo some class of regular languages larger than $\mathcal{U}$?, and (2) is some class larger than MHPDA perfect modulo $\mathcal{U}$?.

Prop. 6.1 shows that the answer to (1) is negative. We do not settle (2), but show in Prop. 6.2 that the largest class of regular languages for which the context-sensitive languages (CSL) are perfect is $\mathcal{F}$. Actually, the proposition shows that no class with an undecidable emptiness problem (and satisfying some additional very weak properties) can be perfect modulo any class of regular languages larger than $\mathcal{F}$. So, in particular, no class containing the languages generated by Okhotin's conjunctive grammars [28] can be perfect modulo a class larger than $\mathcal{F}$.

Proposition 6:

1) $\mathcal{U}$ is the largest class of regular languages such that MHPDA is perfect modulo $\mathcal{U}$;
2) $\mathcal{F}$ is the largest class of regular languages such that CSL is perfect modulo $\mathcal{F}$.

Proof: Part 1. We show that the emptiness problem of MHPDA modulo $\mathcal{C}$ is undecidable for any class $\mathcal{C}$ of regular languages properly containing $\mathcal{U}$. This implies that MHPDA is not perfect modulo $\mathcal{C}$.

Since $\mathcal{C}$ properly contains $\mathcal{U}$, there is an infinite regular language $L \in \mathcal{C}$ that is not equal to a finite union of bounded expressions. We show that there are words u, v_0, v_1, x, such that $v_0 v_1 \neq v_1 v_0$ and $u(v_0 + v_1)^* x \subseteq L$.

We need some preliminaries. We call a NFA A with ε-transitions *simple* if every strongly connected component

(SCC) of A is either trivial or a cycle containing at least one non-ε transition, and every bottom SCC contains a final state. Clearly, if A is simple then there is a finite union $p_1, \ldots, p_n$ of bounded expressions such that $L(A) = p_1 + \cdots + p_n$ (informally, each p_i corresponds to a path in the acyclic graph obtained by contracting every SCC to a single node). Conversely, every finite union of bounded expressions is recognized by a simple NFA with ε-transitions.

Since L is regular, there is NFA with ε-transitions A_L such that $L(A_L) = L$. W.l.o.g. we can assume that every bottom SCC of A_L contains some final state. Since L is infinite, A_L contains at least one nontrivial SCC reachable from the initial state. Since L is not equal to a finite union of bounded expressions, A_L contains at least one SCC, say C, reachable from the initial state, that is not a cycle. Moreover, we can assume that from some state q of C there are two paths leading from q to q that read two words v_0, v_1 such that $v_0 v_1 \neq v_1 v_0$ (otherwise, C can be "replaced" by two cycles: one for v_0^* and one for v_1^*). Let u be any word leading to q, and x be any word leading from q to a final state. Clearly, $u(v_0 + v_1)^* x \subseteq L$.

We now prove that the emptiness problem of MHPDA modulo L (and so modulo $\mathcal{C}$) is undecidable by reduction from the emptiness problem for the intersection of CFGs over the alphabet $\{0, 1\}$. Let G_1, G_2 be two CFGs. Using closure of CFL w.r.t. concatenation and homomorphism, we can easily construct grammars G'_1, G'_2 such that G_i accepts $a_1 \ldots a_n \in \{0,1\}^*$ iff G'_i accepts the word $u(w_1 \ldots w_n)x$, where $w_j = v_0$ if $a_j = 0$, and $w_j = v_1$ if $a_j = 1$ for every $j \in [1, n]$. Now, since $L(G'_1), L(G'_2) \subseteq u(v_0 + v_1)^* x$, we have $L(G'_1) \cap L(G'_2) \cap L = L(G'_1) \cap L(G'_2) \cap u(v_0 + v_1)^* x$, and so $L(G_1) \cap L(G_2) = \varnothing$ iff $L(G'_1) \cap L(G'_2) \cap L = \varnothing$. So the emptiness problem of MHPDA modulo L is undecidable.

Part 2. Since CSL is closed under Boolean operations and has a decidable membership problem, CSL is perfect modulo $\mathcal{F}$. Consider any infinite regular language L. We prove that emptiness of CSL modulo L is undecidable by reduction from the emptiness problem for CSL. This implies that CSL is not perfect modulo $\mathcal{F} \cup \{L\}$.

Since L is infinite, there are words w_1, w_2, w_3 such that $w_1 w_2^* w_3 \in L$. Given a context-sensitive grammar G, it is easy to construct a grammar G' satisfying $L(G') \subseteq w_1 w_2^* w_3$ and such that $L(G)$ is empty iff $L(G')$ is empty. First, we replace every terminal symbol of G by a variable generating w_2, and then we add a new production $S' \to S_1 S S_3$, where S is the axiom of G, and S_1, S_3 are variables generating w_1, w_3. ■

VII. Applications to Verification

In this section, we show MHPDAs are expressive enough to capture several automata-theoretic models. More surprisingly, we show that reductions to MHPDA yield optimal complexity results as well. As an appetizer consider the non emptiness problem for the intersection of k context-free languages and a bounded expression $\bar{\boldsymbol{w}}$. In [11], the authors show that this problem is in NP, and use it to show that assertion checking of multithreaded programs communicating through shared memory modulo $\bar{\boldsymbol{w}}$ is in NP as well. To show that this result is subsumed by ours, proceed as follows. First, compute in polynomial time 1-HPDAs $\{M_i\}_{i \in [1,k]}$ recognizing the context-free languages. Then, use Thm. 3 to compute in $O(k \cdot \max_i(|M_i|) \cdot (|\bar{\boldsymbol{w}}|+1)^3)$ time a formula Ψ such that Ψ is satisfiable iff the intersection of k CFLs and $\bar{\boldsymbol{w}}$ is non empty. Conclude that the problem is in NP.

In the next two sections we prove that the control-state reachability problem modulo a bounded expression for recursive counter machines and communicating finite-state machines are both polynomial-time reducible to MHPDA emptiness modulo a bounded expression, hence they are in NP.

VIII. Recursive Counter Machines

Let $k \geq 1$. A *recursive counter machine* (CM) is a tuple $(S, \Gamma, \mathcal{C}, \mathcal{T}, s_0)$ where S is a non-empty finite set of *control states*; Γ is a *stack alphabet* with a distinguished *bottom stack symbol* $\perp$; $\mathcal{C} = \{c_1, \ldots, c_k\}$ is a finite set of k *counters*; $s_0 \in S$ is the *initial* control state; and $\mathcal{T}$ is a finite set of *transitions* of the form $(\alpha, \gamma) \overset{op}{\to} (\beta, w)$, where $\alpha, \beta \in S$, $\gamma \in \Gamma$, $w \in \Gamma^*$, and $op \in \{\mathsf{inc}_i, \mathsf{dec}_i, \mathsf{zerotest}_i\}_{i \in [1,k]}$ is one of the *counter operations* increment, decrement, or test for zero of $c_i \in \mathcal{C}$ respectively.

A *configuration* $(s, w, v_1, \ldots, v_k) \in S \times \Gamma^* \times \mathbb{N}^k$ consists of a control state s, a stack content w, and a valuation of the counters. The *initial configuration* is $\mathbf{c}_0 = (s_0, \perp, \mathbf{0})$. Let t be a transition $(\alpha, \gamma) \overset{op}{\to} (\beta, \delta)$. We say that a configuration $\mathbf{c}' = (s', w', v'_1, \ldots, v'_k)$ is a *flow t-successor* of $\mathbf{c} = (s, w, v_1, \ldots, v_k)$, denoted by $\mathbf{c}\, F_t\, \mathbf{c}'$, if $s = \alpha$, $s' = \beta$, $w = u\gamma$ and $w' = u\delta$ for some $u \in \Gamma^*$. We say that $\mathbf{c}'$ is a *t-successor* of $\mathbf{c}$, denoted by $\mathbf{c}\, R_t\, \mathbf{c}'$, if $\mathbf{c}\, F_t\, \mathbf{c}'$ and either $op = \mathsf{inc}_i$ and $(v'_1, \ldots, v'_k) = (v_1, \ldots, v_k) + \mathbf{e}_i$, or $op = \mathsf{dec}_i$ and $(v'_1, \ldots, v'_k) = (v_1, \ldots, v_k) - \mathbf{e}_i$, or $op = \mathsf{zerotest}_i$ and $v_i = 0$ and $(v'_1, \ldots, v'_k) = (v_1, \ldots, v_k)$. Given a sequence $\pi \in \mathcal{T}^*$, we define $F(\pi)$ recursively as follows: $F(\varepsilon)$ is the identity relation over configurations, and $F(\pi' \cdot t) = F(\pi') \circ F_t$, where $\circ$ denotes join of relations. Given $L \subseteq \mathcal{T}^*$, we define $F(L) = \bigcup_{\pi \in L} F(\pi)$. We define $R(\pi)$ and $R(L)$ analogously. The set of configurations *reachable through* L is $\mathrm{post}[L] = \{\mathbf{c} \mid \mathbf{c}_0\, R(L)\, \mathbf{c}\}$.

The *control reachability problem* for CM asks, given a control state s_f, whether $\mathrm{post}[\mathcal{T}^*]$ contains a configuration with control state s_f. The problem is undecidable even for non-recursive counter machines [29].

Given a bounded expression $\bar{\boldsymbol{w}}$ over the alphabet $\mathcal{T}$ of transitions, the *control reachability problem modulo $\bar{\boldsymbol{w}}$* is the question whether $\mathrm{post}[\bar{\boldsymbol{w}}]$ contains a configuration with control state s_f. We show that this problem is NP-complete by means of a reduction to the emptiness problem of a MHPDA modulo a bounded expression.

We encode CM as follows. Fix a CM $(S, \Gamma, \mathcal{C}, \mathcal{T}, s_0)$ with k counters and a bounded expression $\bar{\boldsymbol{w}}$ over $\mathcal{T}$. We construct $k+1$ 1-HPDAs such that $\pi \in \mathcal{T}^*$ is accepted by all the 1-HPDA iff $\mathrm{post}[\pi]$ contains a configuration with s_f as control state. (Following Prop. 3, we can then construct an equivalent $(k+1)$-HPDA if we wish.)

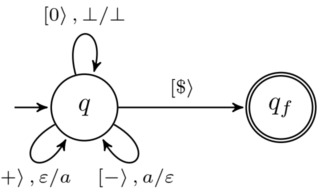


Fig. 3. The 1-HPDA $P_{\surd}$ over alphabet $\{+,-,0\}$.

The first PDA P_0 checks whether $\mathbf{c}_0\,F(\pi)\,\mathbf{c}$ holds for some configuration $\mathbf{c}$ having s_f as control state. Since for each transition t of the CM the relation F_t exactly corresponds to the relation induced by the productions of a pushdown automaton, the construction of P_0 is straightforward, and we omit the details.

A word π accepted by P_0 is consistent with the control flow of the CM, but might not be feasible (π may zero-test a counter whose value is not 0, or decrement a counter whose value is 0). Feasibility is checked by PDAs $P_1,\dots,P_k$. More precisely, P_i checks that the projection of π onto the operations of c_i is feasible. We first describe a generic PDA $P_{\surd}$ over the alphabet $\{+,-,0\}$, where "+" encodes increment, "−" decrement, and "0" a zero-test, as a template that can be instantiated to generate $P_1,\dots,P_k$.

$P_{\surd}$ is shown in Fig. 3. It uses its stack as a counter. The stack alphabet is $\{\bot,a\}$. When $P_{\surd}$ reads a $+$ (a $-$), it pushes an a into (pops an a from) the stack, and when it reads 0, it checks that the top element is the bottom stack symbol $\bot$ ($[0\rangle\,,\bot/\bot$). Now, P_i is a suitably modified version which, when reading a letter $t=(\alpha,\gamma)\xrightarrow{op}(\beta,w)$, acts according to the operation op: if $op=\mathsf{inc}_i$ (dec_i, $\mathsf{zerotest}_i$), then t is treated as $+$ ($-$, 0). If op does not operate on the i-the counter, then control ignores t. Applying Thm. 3, we get:

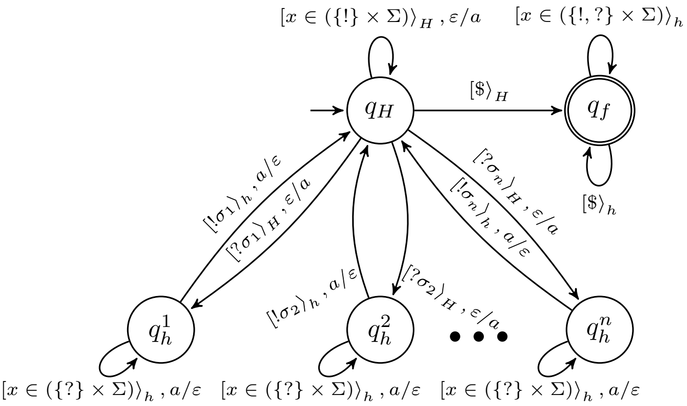
Theorem 6: Given a CM $A=(S,\Gamma,\mathcal{C},\mathcal{T},s_0)$ *with* k *counters, a control state* $s_f\in S$*, and a bounded expression* $\bar{\boldsymbol{w}}$ *over* $\mathcal{T}$*, there is a computable formula* Φ_{A,s_f} *of existential Presburger arithmetic of size* $O(k\cdot|A|^3\cdot(|\bar{\boldsymbol{w}}|{+}1)^3)$ *such that* $\mathrm{post}[\bar{\boldsymbol{w}}]$ *contains a configuration with state* s_f *iff* Φ_{A,s_f} *is satisfiable. Thus, the control reachability problem modulo a bounded expression for recursive counter machines is in NP.*

NP-hardness holds even for non-recursive counter machines (this result has been communicated to us by S. Demri and is shown in [25]), and therefore the bound of Thm. 6 is optimal.

A similar construction can be used to simulate recursive machines with k-auxiliary stacks.

IX. Communicating Finite State Machines

Let $k\geq 1$. A communicating finite state machines (CFSM) is a tuple $(S,K,\Sigma,\mathcal{T},s_0)$ where S is a non-empty finite set of *control states*; $K=\{\mathcal{C}_1,\dots,\mathcal{C}_k\}$ is a finite set of *unbounded FIFO channels*; Σ is a non-empty finite set of *messages*; $s_0\in S$ is the *initial* control state; and $\mathcal{T}$ is a finite set of *transitions*. Each transition $t\in\mathcal{T}$ is given by a triple (α,op,β) where $\alpha,\beta\in S$ and op is the *channel operation*: either $!\sigma\colon\mathcal{C}_i$, which writes message $\sigma\in\Sigma$ to channel $\mathcal{C}_i$ or $?\sigma\colon\mathcal{C}_j$, which reads message $\sigma\in\Sigma$ from channel $\mathcal{C}_j$. A *configuration* is a tuple $(s,x_1,\dots,x_k)\in S\times[\Sigma^*]^k$ containing a control state and the content of each channel $\mathcal{C}_i\in K$. The *initial configuration* is $\mathbf{c}_0=(s_0,\varepsilon,\dots,\varepsilon)$.



$[x\in(\{!\}\times\Sigma)\rangle_H\,,\varepsilon/a$
$[x\in(\{!,?\}\times\Sigma)\rangle_h$
q_H
$[\$\rangle_H$
q_f
$[\$\rangle_h$
$[!\sigma_1\rangle_h\,,a/\varepsilon$
$[?\sigma_1\rangle_H\,,\varepsilon/a$
$[!\sigma_2\rangle_h\,,a/\varepsilon$
$[?\sigma_2\rangle_H\,,\varepsilon/a$
$[?\sigma_n\rangle_H\,,\varepsilon/a$
$[!\sigma_n\rangle_h\,,a/\varepsilon$
q_h^1
q_h^2
q_h^n
$[x\in(\{?\}\times\Sigma)\rangle_h\,,a/\varepsilon$
$[x\in(\{?\}\times\Sigma)\rangle_h\,,a/\varepsilon$
$[x\in(\{?\}\times\Sigma)\rangle_h\,,a/\varepsilon$

Fig. 4. The 2-HPDA $P_{\surd}$ where $\Sigma=\{\sigma_1,\dots,\sigma_n\}$.

Given $t=(\alpha,op,\beta)\in\mathcal{T}$, we define the relations F_t and R_t over configurations as follows: $(s,x_1,\dots,x_k)$ F_t $(s',x_1',\dots,x_k')$ iff $\alpha=s$ and $\beta=s'$, and $(s,x_1,\dots,x_k)$ R_t $(s',x_1',\dots,x_k')$ iff $\alpha=s$, $\beta=s'$, and for all $i\in[1,k]$ either $op={?\sigma}\colon\mathcal{C}_i$ and $x_i=\sigma\cdot x_i'$; or $op={!\sigma}\colon\mathcal{C}_i$ and $x_i'=x_i\cdot\sigma$; or $x_i'=x_i$ for other choices of op.

The relations $F(L)$, $R(L)$, $\mathrm{post}[L]$, the control reachability problem, and the control reachability problem modulo a bounded expression for CFSMs are defined as for CM. The reachability problem for CFSM is undecidable [30].

To encode CFSM, we proceed as for recursive counter machines. Given a CFSM with k channels, we construct a finite automaton P_0 and k 2-HPDAs $P_1,\dots,P_k$ such that $\pi\in\mathcal{T}^*$ is accepted by all of $P_0,\dots,P_k$ iff $\mathrm{post}[\pi]$ contains a configuration with s_f as control state. Again, P_0 checks whether $\mathbf{c}_0\,F(\pi)\,\mathbf{c}$ holds for some configuration $\mathbf{c}$ having s_f as control state, and P_1 to P_k check feasibility of π. In the case of CFSM, feasibility means that the contents of the channels after taking a transition t are the ones given by $R(t)$.

P_0 is even simpler than for CM, since there is no recursion.[4]

P_i checks feasibility of π with respect to the i-th channel. As in the case of CM, we define a generic 2-HPDA $P_{\surd}$, depicted in Fig. 4, that checks consistency for a channel $\mathcal{C}$.

For convenience the heads of $P_{\surd}$ are named h and H. The stack alphabet is $\{\bot,a\}$. $P_{\surd}$ works as follows. In state q_H, head H reads symbols $\{!\sigma_i\mid i\in[1,n]\}$ to channel $\mathcal{C}$ until a symbol $?\sigma_i$ for some $i\in[1,n]$ or \$ is read. When $?\sigma_i$ is read, control jumps to q_h^i. In q_h^i, head h looks for the first symbol $\{!\sigma_i\mid i\in[1,n]\}$. If it is $!\sigma_i$ (which corresponds to $?\sigma_i$) then control returns to q_H. Intuitively, if a symbol is read from channel $\mathcal{C}$ it must have been written previously. Observe that the stack ensures ensure that h does not move beyond H. In fact, in every reachable configuration not in state q_f, $P_{\surd}$

[4] Our results also hold for recursive CFSM, but since this model is rather artificial, we refrain from describing it.

maintains the invariant that the number of symbols between H and h coincides with the number of a's on the stack. For instance for tapes $\langle t_h, t_H \rangle$ where

$$\begin{aligned} t_h &= !\sigma_1 \# !\sigma_2 ?\sigma_1 ?\sigma_2 \\ t_H &= !\sigma_2 \quad !\sigma_2 ?\sigma_1 ?\sigma_2 \# \end{aligned}$$

the stack content is given by $\bot a^3$. Because of the invariant, head H will be the first to read \$ in which case the control is updated to q_f. Hence transitions read anything until with head h until it falls down the tape.

We can now apply Thm. 3 again. In this case, the members of our family of MHPDAs have at most 2 heads, i.e., $c = 2$.

Theorem 7: Given a CFSM $A = (S, K, \Sigma, \mathcal{T}, s_0)$ *with* k *channels, a control state* $s_f \in S$*, and a bounded expression* $\bar{\boldsymbol{w}}$ *over* $\mathcal{T}$*, there is a computable formula* Φ_{A,s_f} *of existential Presburger arithmetic of size* $O(k \cdot |A|^3 \cdot (|\bar{\boldsymbol{w}}|+1)^6)$ *such that* $\mathrm{post}[\bar{\boldsymbol{w}}]$ *contains a configuration with state* s_f *iff* Φ_{A,s_f} *is satisfiable. Thus, the control reachability problem modulo a bounded expression for* CFSM *is in NP.*

In [25], we prove that NP-hardness holds for CFSM, and therefore our bound is optimal. Finally, we observe that the above reductions can be combined to handle machines where transitions are either counter operations or channel operations. The construction of P_0 is as for CM. Then, for each auxiliary storage (channel or counter), it suffices to use the appropriate MHPDA to check that a sequence of operations is feasible. Again, we can show an NP upper bound for the control reachability problem modulo a bounded expression.

X. Conclusion

We have introduced verification modulo a class of languages, which formalizes the common practice, for efficiency reasons, of checking only a subset of the behaviours of a system. This leads to the notion of a perfect computational model $\mathcal{M}$ modulo a class of behaviors $\mathcal{C}$. We have presented a perfect model for the class of bounded expressions: multi-head pushdown automata (MHPDA). We have determined the complexity of the emptiness problem, shown that many popular modelling formalisms can be easily compiled into MHPDA, and proved that the compilation leads to verification algorithms of optimal complexity.

Acknowledgment

P. G. was sponsored by Comunidad de Madrid's Program PROMETIDOS-CM (S2009TIC-1465), PEOPLE-COFUND's program AMAROUT (PCOFUND-2008-229599), and by the Spanish Ministry of Economy and Finance (TIN2010-20639).

References

[1] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 1st ed. Addison-Wesley, April 1979.

[2] R. Alur and P. Madhusudan, "Adding nesting structure to words," *J. ACM*, vol. 56, no. 3, 2009.

[3] S. L. Torre, P. Madhusudan, and G. Parlato, "A robust class of context-sensitive languages," in *LICS '07: Logic in Computer Science*. IEEE Computer Society, 2007, pp. 161–170.

[4] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.

[5] E. M. Clarke, D. Kroening, and K. Yorav, "Behavioral consistency of C and Verilog programs using bounded model checking," in *DAC '03: Design Automation Conference*. ACM, 2003, pp. 368–371.

[6] S. Qadeer and J. Rehof, "Context-bounded model checking of concurrent software," in *TACAS '05: Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 3440. Springer, 2005, pp. 93–107.

[7] S. Qadeer, "The case for context-bounded verification of concurrent programs," in *SPIN'08: Model Checking Software*, ser. LNCS, vol. 5156. Springer, 2008, pp. 3–6.

[8] M. F. Atig, A. Bouajjani, and S. Qadeer, "Context-bounded analysis for concurrent programs with dynamic creation of threads," *Logical Methods in Computer Science*, vol. 7, no. 4, 2011.

[9] V. Kahlon, "Tractable dataflow analysis for concurrent programs via bounded languages," July 2009, patent WO/2009/094439.

[10] S. Ginsburg, *The Mathematical Theory of Context-Free Languages*. New York, NY, USA: McGraw-Hill, Inc., 1966.

[11] J. Esparza and P. Ganty, "Complexity of pattern-based verification for multithreaded programs," in *POPL '11: Principles of Programming Languages*. ACM, 2011, pp. 499–510.

[12] J. Leroux and G. Sutre, "Flat counter automata almost everywhere," in *ATVA '05: Automated Technology for Verification and Analysis*, ser. LNCS, vol. 3707. Springer, 2005, pp. 489–503.

[13] A. Bouajjani and P. Habermehl, "Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations," *Theor. Comput. Sci.*, vol. 221, no. 1-2, pp. 211–250, 1999.

[14] S. Bardin and A. Finkel, "Composition of accelerations to verify infinite heterogeneous systems," in *ATVA '04: Automated Technology for Verification and Analysis*, ser. LNCS, vol. 3299. Springer, 2004, pp. 248–262.

[15] O. H. Ibarra, "Generalizations of pushdown automata," Ph.D. dissertation, University of California, Berkeley, 1967.

[16] ——, "A note on semilinear sets and bounded-reversal multihead pushdown automata," *Inf. Proc. Letters*, vol. 3, no. 1, pp. 25–28, 1974.

[17] I. H. Sudborough, "Bounded-reversal multihead finite automaton languages," *Information and Control*, vol. 25, pp. 317–328, 1974.

[18] M. Hague and A. W. Lin, "Model checking recursive programs with numeric data types," in *CAV '11: Computer Aided Verification*, ser. LNCS. Springer, 2011.

[19] O. H. Ibarra, "Reversal-bounded multicounter machines and their decision problems," *Journal of the ACM*, vol. 25, no. 1, pp. 116–133, 1978.

[20] M. Cadilhac, A. Finkel, and P. McKenzie, "Bounded Parikh automata," in *WORDS'11: Proc. 8th Int. Conf. WORDS*, ser. Electronic Proceedings in Theoretical Computer Science, vol. 63, 2011, pp. 93–102.

[21] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci, "FAST: acceleration from theory to practice," *STTT*, vol. 10, no. 5, pp. 401–424, 2008.

[22] K. N. Verma, H. Seidl, and T. Schwentick, "On the complexity of equational Horn clauses," in *CADE '05: 20th Int. Conf. on Automated Deduction*, ser. LNCS, vol. 1831. Springer, 2005, pp. 337–352.

[23] A. L. Rosenberg, "On multi-head finite automata," *IBM J. Res. Dev.*, vol. 10, no. 5, pp. 388–394, 1966.

[24] J. von zur Gathen and M. Sieveking, "A bound on solutions of linear integer equalities and inequalities," *Proceedings of the American Mathematical Society*, vol. 72, no. 1, pp. 155–158, 1978.

[25] J. Esparza, P. Ganty, and R. Majumdar, "A perfect model for bounded verification," *CoRR*, vol. abs/1201.3194, 2012.

[26] C. R. Reddy and D. W. Loveland, "Presburger arithmetic with bounded quantifier alternation," in *STOC '78: Proc. of the 10th Annual ACM Symposium on Theory of Computing*. ACM, 1978, pp. 320–325.

[27] M. Fürer, "The complexity of Presburger arithmetic with bounded quantifier alternation depth," *Theor. Comput. Sci.*, vol. 18, pp. 105–111, 1982.

[28] A. Okhotin, "Conjunctive grammars," *J. Automata, Languages and Combinatorics*, vol. 6:4, pp. 519–535, 2001.

[29] M. Minsky, *Finite and Infinite Machines*. Englewood Cliffs, N.J., Prentice-Hall, 1967.

[30] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *J. ACM*, vol. 30, no. 2, pp. 323–342, 1983.

Appendix
Missing Proofs

A. *Proposition 4*

The emptiness problem for MHPDAs modulo an arbitrary bounded expression is in coNEXPTIME. Moreover, the emptiness problem for MHPDAs and $\bar{w} = (01)^*$ is coNEXPTIME-hard.

Proof of Prop. 4: Membership in coNEXPTIME follows immediately from Thm. 2 and the fact that satisfiability of existential Presburger formulas is in NP [24].

For the hardness part, we reduce from 0-1 Succinct Knapsack.

> **Input**: Boolean circuit θ with $k+n$ variables ($k, n > 0$ given in unary). The circuit represents 2^k numbers $a_0, \ldots, a_{2^k-1}$, each with 2^n bits in binary, as follows. The ith bit of the binary representation of a_j is $x \in \{0,1\}$ if the circuit θ on input $bin^k(j)$, $bin^n(i)$ evaluates to x, for $i \in [0, 2^n - 1]$, $j \in [0, 2^k - 1]$, where $bin^\alpha(\beta)$ is the binary representation of β using α bits.
>
> **Output**: "Yes" if there exist $z_1, \ldots, z_{2^k-1} \in \{0,1\}$ such that $a_0 = \sum_{i=1}^{2^k-1} a_i z_i$; "No" otherwise.

Given an instance of the 0-1 Succinct Knapsack problem, we construct in polynomial time a MHPDA that accepts a string of the form $(01)^*$ iff the 0-1 Succinct Knapsack problem answers "Yes."

The idea of the proof is to use d heads of a MHPDA and the bounded expression $(01)^*$ to encode 2^d states, and to use the stack to compute up to 2^{2^d}. The MHPDA use two heads, one to track a_0 and one to track the sum on the r.h.s. If these heads point to the same location at the end, we accept. Note that we cannot directly check if two heads are pointing to the same location. However, we can alternately move the heads to the right (by reading) and check that they hit the endmarker at the same time.

We start with some preliminary constructions. We use d heads $h_1, \ldots, h_d$ to encode a d-bit configuration $b \in \{0,1\}^d$: to encode b, we make sure that head h_i is pointing to b_i on the tape where b_i is the ith bit of b. With this representation, we denote by $\boldsymbol{h}$ the d-bit binary number given by the symbols under the heads $h_1, \ldots, h_d$. Also, we denote by $[\![\boldsymbol{h}]\!]$ the number that is represented.

Given a constant $c \in \{0,1\}^d$, we can check that the current store encodes c without destroying the current encoding as follows. First, observe that the bounded expression $(01)^*$ ensures that by reading twice from any head, the head points to the same bit as it was pointing to before the two reads (for a long enough string). For $i = 1, \ldots, d$, read twice with h_i and remember the first value, say x, that is read. Then check that $x = c_i$. If not, we go to a state signifying that the current configuration is not storing c, otherwise we continue the next iteration of the loop. At the end of the loop, we go to a state that signifies that the current encoding is equal to c. The stack is not touched.

Given heads $h_1, \ldots, h_d$, we can "reset" the encoding to a specific $c \in \{0,1\}^d$ (noted $\boldsymbol{h} \leftarrow c$) as follows. For $i = 1, \ldots, d$, read with h_i and let x be the value read. If $x = c_i$, then again read with h_i; else do nothing (because after the read with h_i, it points to c_i). The stack is not touched.

Given heads $h_1, \ldots, h_d$ and $h'_1, \ldots, h'_d$, we can "copy" the encoding of the h_is to h'_is ($\boldsymbol{h'} \leftarrow \boldsymbol{h}$) as follows. For $i = 1, \ldots, d$, we execute the following. Read twice with h_i and remember the first value, say x, that is read. Now read with h'_i, if the value read equals x then read again; else do nothing. At the end of updating the d heads we have that $\boldsymbol{h'}$ equals $\boldsymbol{h}$. The stack is not touched.

Given the binary number $\boldsymbol{h}$, $\boldsymbol{h} \neq 1^d$, we show how to add one to the number such that the resulting $\boldsymbol{h}$ encodes $[\![\boldsymbol{h}]\!] + 1$. Read with h_1, if the symbol read is 0 then we are done (h_1 points to 1); else (h_1 points to 0) do the following: read with h_2, if the symbol read is 0 (h_2 points to 1) then we are done. In general, if h_i points to zero (and all $h_1, \ldots, h_{i-1}$ point to 1's), read with each head $h_1, \ldots, h_i$. We can similarly subtract one from the number $\boldsymbol{h}$, $\boldsymbol{h} \neq 0^d$, by replacing zero with one in the above construction. In both constructions, the stack is not touched.

Finally, suppose we have heads $h_0, \ldots, h_d$, a head H, and 2^d bits $c_1 c_2 \ldots c_{2^d}$ on the stack. Let C be the number with binary representation $c_{2^d} \ldots c_1$. We show how the head H can be moved C times to the right, using the heads $h_0, \ldots, h_d$. Note that C can be as large as $2^{2^d} - 1$, so we cannot directly store C using $poly(d)$ heads. Instead, we use the d heads to count the position in the stack, and perform binary arithmetic on the number in the stack. We execute the following program.

```
   $[\![h]\!] \leftarrow 2^d$
1: while $[\![h]\!] \neq 0$ and top of stack is 0 {
      pop;
      $[\![h]\!] \leftarrow [\![h]\!] - 1$;
   }
   if $[\![h]\!] = 0$ {
      exit; /* H has now moved C times */
   } else { /* top of stack is necessarily 1 */
      pop 1; push 0; read with $H$;
      while $[\![h]\!] \neq 2^d$ {
         push 1;
         $[\![h]\!] \leftarrow [\![h]\!] + 1$;
      }
   }
   goto 1;
```

Using the constructions above, the program can be implemented by a MHPDA of size polynomial in d. We call this procedure $MoveRight(H)$.

We now show how to evaluate the circuit θ. W.l.o.g., we assume that θ is given as $p_1(n+k)$ layers, each layer has $p_2(n+k)$ binary gates, for polynomials p_1 and p_2. We use $k + n + p_1(k+n)p_2(k+n)$ heads. The inputs are copied into

$k+n$ heads. Then, we evaluate the value of each gate, starting at the lowest layer, and store it into the head representing that gate. To evaluate the gate, we look at the values encoded by the heads representing its inputs, and evaluate the Boolean function for the gate. The stack is untouched in the evaluation. Thus, circuit evaluation can be performed by a MHPDA (indeed, a multi-head finite automaton) using polynomially many (in $k+n$) heads.

Now we come to the main construction. The MHPDA has the following heads:

- a head A_0 to track a_0, a head SUM to track the r.h.s.
- k heads $K_1, \ldots, K_k$ to track the indices of the numbers $a_0, \ldots, a_{2^k-1}$;
- m heads $M_1, \ldots, M_m$ to track the 2^m bits of each number;
- $m+1$ heads $H_0, \ldots, H_m$ to implement procedure *MoveRight* above;
- additional heads (polynomial in $k+n$) to evaluate circuit θ.

Initially, each head points to a 0, in particular, $\boldsymbol{K} = 0^k$. The MHPDA works in the following phases.

In the first phase, we initialize $\boldsymbol{M}$ to 1^m and then run the following iteratively. We evaluate θ on the input $\boldsymbol{K};\boldsymbol{M}$ (by copying $K_1, \ldots, K_k, M_1, \ldots, M_m$ on to the circuit inputs and then evaluating the circuit), and push the evaluated value on to the stack. If $\boldsymbol{M} = 0^m$ we move to the next phase of the construction. Otherwise, we subtract 1 from $[\![\boldsymbol{M}]\!]$ and repeat the evaluation.

At the end of the above loop, we have 2^m bits, representing the number a_0 stored on the stack (least significant bit on top). We now invoke $\mathit{MoveRight}(A_0)$, which will move head A_0 of a_0 times to the right.

Then comes the phase of guessing and summing a subset of $\{a_1, \ldots, a_{2^k-1}\}$ to compare the resulting value against a_0. First we set $[\![\boldsymbol{K}]\!]$ to 1. For $[\![\boldsymbol{K}]\!] = 1$ to 2^k-1, do the following loop. We guess if $z_{[\![\boldsymbol{K}]\!]}$ is zero or one, using the finite state of the automaton. If $z_{[\![\boldsymbol{K}]\!]}$ is guessed to be zero, we continue with the next iteration of the loop. Otherwise, we initialize $\boldsymbol{M}$ to 1^m, and iteratively evaluate θ on $\boldsymbol{K};\boldsymbol{M}$ for each $\boldsymbol{M}$ from 1^m to 0^m, and push each evaluated bit on the stack. At the end of the process, we have the 2^m bits of $a_{[\![\boldsymbol{K}]\!]}$ on the stack, least significant bit first. We now invoke $\mathit{MoveRight}(SUM)$ to move the head SUM $a_{[\![\boldsymbol{K}]\!]}$ times to the right.

At the end of the loop, we have that the head SUM has moved $\sum_{i=1}^{2^k-1} z_i a_i$ times to the right, where the z_i's are the guesses made by the MHPDA. We now check if A_0 and SUM are pointing to the same tape cell by moving them alternately and checking that they read the endmarker \$ immediately one after the other. If so, we read with all heads until they fall off the tape and accept. Otherwise, we reject. Note that the computations can be performed by a MHPDA that is polynomial in the size of the input.

If the answer to the 0-1 Succinct Knapsack instance is "Yes," then there is a sequence of guesses, and a string in $(01)^*$ that is sufficiently long to perform all the computations, such that the MHPDA accepts. However, if the answer is "No" then the language of the automaton is empty.

Thus, given a MHPDA M, and the fixed bounded expression $(01)^*$, checking if $L(M) \cap (01)^*$ is empty is coNEXPTIME-hard. ■

B. Proposition 5

Given a d-HPDA A and a letter-bounded expression $\bar{\boldsymbol{b}} = b_1^* \ldots b_n^*$, there is a MHPDA B such that $L(B) = \bar{\boldsymbol{b}} \setminus L(A)$ and $|B|$ is at most doubly exponential in $|A|$, $|\bar{\boldsymbol{b}}|$.

Proof of Prop. 5: The complementation procedure follows these steps:

- Compute the family $\{\Psi_i(x_1, \ldots, x_n)\}_{i=1}^{\alpha}$ of existential Presburger formulas of Cor. 1, each of them of size $p(|A| \cdot (|\bar{\boldsymbol{b}}|+1) \cdot d)$ for a suitable polynomial p. Recall that $\alpha = d^{(n+1)d}$.
- Compute quantifier-free formulas $\Phi_i \equiv \neg\Psi_i$ with constants in unary of size $|\Phi_i| \in \mathit{2exp}(O(|\Psi_i|))$. By Cor. 1 we have $b_1^{k_1} \ldots b_n^{k_n} \in (\bar{\boldsymbol{b}} \setminus L(A))$ iff $\bigvee_{i=1}^{\alpha} \Psi_i(k_1, \ldots, k_n)$ is false iff $\bigwedge_{i=1}^{\alpha} \Phi_i(k_1, \ldots, k_n)$ is true.
- Construct for every formula Φ_i a MHPDA B_i of size $O(|\Phi_i|)$ as in Prop. 4.
- Let B be a MHPDA accepting $\bigcap_{i=1}^{\alpha} L(B_i)$, which exists by Prop. 3. We have

$$\begin{aligned}|B| &\in O(\textstyle\sum_{i=1}^{\alpha}|B_i|)\\ &\in O(\textstyle\sum_{i=1}^{\alpha}|\Phi_i|)\\ &\in \textstyle\sum_{i=1}^{\alpha}\mathit{2exp}(O(|\Psi_i|))\\ &\in d^{(n+1)d} \cdot \mathit{2exp}(p'(|A| \cdot n + 1 \cdot d))\\ &= \mathit{2exp}(p''(|A| \cdot n + 1 \cdot d))\end{aligned}$$

for suitable polynomials p', p''.

■

C. NP-hardness of control state reachability modulo bounded expressions for Counter Machines

Proof: We reduce from 3SAT. Given a 3SAT formula $c_1 \wedge \ldots \wedge c_m$ over variables $x_1, \ldots, x_n$, we construct a CM with counters $\{t_{x_i}, f_{x_i} \mid i \in [1,n]\} \cup \{c_i \mid i \in [1,m]\}$. We use a gadget to assign values to variables and a gadget to check that a clause is satisfied by the current assignment to variables. Fig. 5 shows the gadgets.

For each variable x in the formula, we keep two counters t_x and f_x. The variable gadget (top of Fig. 5) ensures that when control reaches q_2, then either $t_x = 1$ and $f_x = 0$ (encoding that x is true) or $t_x = 0$ and $f_x = 1$ (encoding that x is false), depending on whether the loop is executed one or zero times, respectively. Note that the loop can be executed at most once: the second iteration gets stuck decrementing f_x.

The clause gadget (bottom of Fig. 5) shows how we check that a clause $c \equiv x_1 \vee \neg x_2 \vee x_3$ is satisfied. The gadget keeps a "control" counter c. The first loop checks that $f_{x_1} = 0$ (i.e., $t_{x_1} = 1$, and x_1 is set to true) and increments c. The second loop checks that $t_{x_2} = 0$ (i.e., $f_{x_1} = 1$, and x_2 is set to

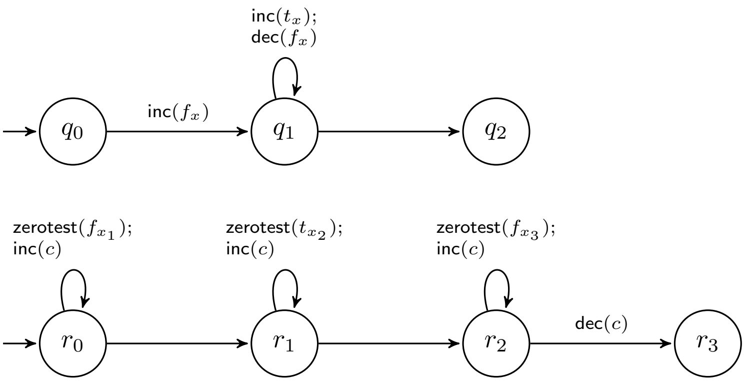


Fig. 5. Reduction for CMs. The top gadget shows variable assignment. The bottom gadget shows the checks for a clause $c \equiv x_1 \vee \neg x_2 \vee x_3$.

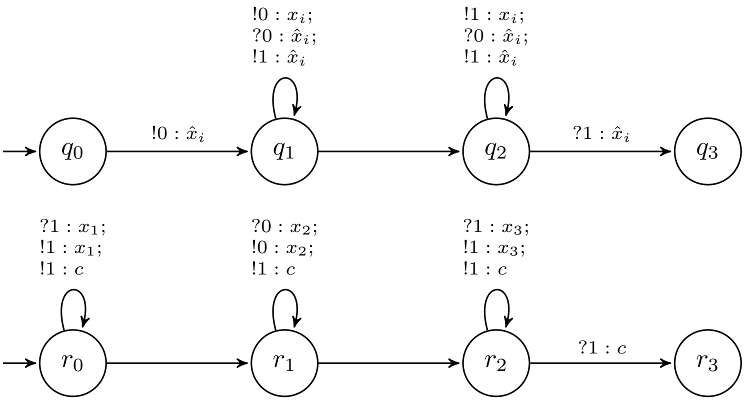


Fig. 6. Reduction for CFSMs. The top gadget shows variable assignment. The bottom gadget shows the checks for a clause $c \equiv x_1 \vee \neg x_2 \vee x_3$.

false) and increments c. The third loop checks that $f_{x_3} = 0$ (i.e., $t_{x_3} = 1$, and x_3 is set to true) and increments c. Each loop can be executed any number of times. At the end, the decrement succeeds only when at least one iteration of a loop has executed, which indicates that c is satisfied. Note that if c is not satisfied, control cannot reach the last location r_3: either one of the tests in the loops get stuck, or the decrement at the end gets stuck.

For the reduction, we sequentially compose gadgets for all the variables and then all the clauses and ask if the control state at the end of the last clause can be reached. Clearly, paths of the automaton conform to a bounded expression. ■

D. NP-hardness of control state reachability modulo bounded expressions for CFSM

Proof: We reduce from 3SAT. Given a 3SAT formula $c_1 \wedge \ldots \wedge c_m$ over variables $x_1, \ldots, x_n$, we construct a CFSM with channels $\{x_i, \hat{x}_i \mid i \in [1, n]\} \cup \{c_i \mid i \in [1, m]\}$. There are two messages: 0 and 1. The channel x_i is used to keep a guess for the variable x_i. The channel $\hat{x}_i$ is a "control channel" used to ensure only one guess is made. The control flow graph of the CFSM consists of gadgets selecting a value for each variable and gadgets checking that each clause is satisfied.

The gadget for variables is shown on the top of Fig. 6. It first puts a single message 0 into the control channel $\hat{x}_i$. It then defines two loops. The first puts 0 in the channel x_i (thereby guessing x_i is false) and flips the control channel by dequeueing the 0 and enqueueing a 1. The second puts 1 in the channel x_i (thereby guessing that x_i is true) and flips the control channel as before. Finally, the edge from q_2 to q_3 dequeues a 1 from the control channel.

By the use of the control channel, we note that any execution that reaches q_3 must execute exactly one loop, exactly one time. When control reaches q_3, the control channel $\hat{x}_i$ is empty, and the channel x_i is either 0 or 1.

The gadget for clauses is shown in the bottom of Fig. 6, for the particular clause $c \equiv (x_1 \vee \neg x_2 \vee x_3)$ (the general case is immediate). The gadget for the clause has three loops, one for each literal in the clause. Each loop checks if the value guessed for the variable matches the literal (i.e., the clause is satisfied). If so, a message is added to the channel c. At the end of the three loops (edge r_2 to r_3), we check that the control channel c has at least one message. By construction, control can reach r_3 only when the current guess for the variables satisfies the clause. Moreover, the channels x_i are unchanged.

The CFSM sequentially composes the variable gadgets and the clause gadgets, and checks if control can reach the last node of the last clause gadget. Clearly, paths of the automaton conform to a bounded expression. ■