Complementable Normal Form of Parametrized Automata

Franziska Alber$^{1[0009-0001-9940-8799]}$ and Philipp Rümmer$^{1,2[0000-0002-2733-7098]}$

^{1} University of Regensburg, Germany
^{2} Uppsala University, Sweden

Abstract. Parametrized automata (PA) are an extension of two existing kinds of automata, symbolic automata and variable automata. In PA, transitions are labeled with formulas that may contain variables. PA are a powerful tool for modeling systems over infinite alphabets, but complementation of PA has been proven to be challenging: not every PA has a complement, and complementation in general may be non-computable. This paper presents a new notion of normal form for PA, called complementable normal form (CFPA). CFPA is sufficiently expressive to completely characterize the class of PA that can be complemented. We show that all Boolean operations (including complementation) can be computed efficiently on CFPA, and key problems such as the universality and non-emptiness problem are decidable for CFPA. Based on CFPA, we propose a new method for complementing PA.

Keywords: Parametrized Automata · Infinite Alphabets · Complementation

1 Introduction

Parametrized Automata (PA, [10]) are an extension of finite-state automata to infinite alphabets: the transitions are labeled with logical formulas that may contain (non-reassignable) parameters. As such, PA subsume both symbolic automata [4] and variable automata [8] and are more expressive than either. As an example, we may choose the real numbers as an alphabet and allow relations and operations on the real numbers to occur in transition formulas. Then, PA can compare two different input letters, or measure the distance between letters. PA over the real numbers can identify unsorted words, words where the last letter is largest, or words where all letters fall within a fixed range. PA find application in verification tasks, for example verifying invariant properties of Dijkstra's self-stabilizing protocol or Quicksort [5].

PA are not closed under complementation: there are languages that can be represented by PA while their respective complement languages cannot [1]. In the context of software verification, this is a serious limitation. Proving that a system satisfies a property often requires showing that no execution exists that violates the property, therefore involving complement operations. If the complement language of a PA exists and can be represented by another PA, we call that PA complementable. Because the universality problem for PA is undecidable, even the simple task of deciding whether a given PA is complementable might only be possible for limited subclasses of PA.

Contributions. As our first contribution, we present Complementable Normal Form, a new kind of normal form for PA. The subclass of PA in complementable normal form (CFPA) is both easy to complement and sufficiently expressive to represent all complementable PA. In a CFPA, the set of states is partitioned into three subsets: accepting states, complement-accepting states, and weak states. If the run of a word terminates in an accepting state, it is part of the PA's language. If the run terminates in a complement-accepting state, the word is part of the complement language. If the run terminates in a weak state, we do not gain any information. In section 3, we formally define CFPA and show that every complementable PA is equivalent to a CFPA.

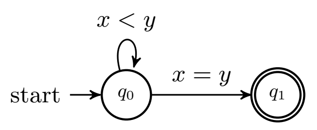
As the second contribution, we survey the properties of CFPA, in particular closure under Boolean operations, efficient computation thereof and decidability of key decision problems such as universality. We also compare CFPA to the related concept of strong determinism in subsection 3.2. We propose CFPA as the foundation for a new method for complementing PA: every complementable PA is equivalent to some CFPA, and complementation of CFPA is then straightforward. The legwork, therefore, lies in finding an equivalent CFPA.

Our third and most important contribution is found in section 4, where we describe a method for transforming arbitrary complementable PA into CFPA. The key idea behind the method is to exploit the relationship between words and accepting parameter assignments that is implicitly defined by a PA. We show that the method is applicable for all complementable PA.

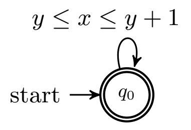
Related Work. More information on PA can be found in [10] and [1]. There are two classes of automata that have a particularly close relationship to PA: symbolic automata [4], where transitions are labeled using logical formulas, and variable automata [8], which can compare input letters to non-reassignable variables for equality and inequality. PA are strictly more expressive than either of these classes, because symbolic automata cannot compare different input letters, while variable automata are blind to the logical "structure" of the input alphabet. Variable automata and PA are incomparable to register automata [11], as the latter can overwrite stored values while variable automata and PA use fixed parameter assignments.

PA can be considered a generalization of parametric semilinear data automata, which are described in [6]. The latter use a specific extension of linear temporal logic while PA allow a wider range of first-order theories. PA should also be distinguished from symbolic register automata [3], which combine symbolic automata [4] with register automata [11] instead of variable finite automata [8].

We further point out differences to the known notion of ambiguity in automata [7]. In an unambiguous finite-state automaton, every word has at most



(a) A_2, identifying words whose last letter is largest.



(b) A_3, identifying words whose letters fall within an interval of length 1.

Fig. 1: Examples of parametrized automata using the theory of real numbers. In both examples, y denotes the single parameter.

one accepting run (although multiple non-accepting runs are permitted). In CFPA, by contrast, there is no limit on the number of distinct acceptable runs. Instead, CFPA have limitations on which states are "reachable" by words that lie in the CFPA's language.

2 Parametrized Automata

2.1 Definition and Notation

A finite-state automaton (FSA, [9]) is a tuple $A = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite alphabet, Q a set of states, $q_0 \in Q$ an initial state, $\delta \subseteq Q \times \Sigma \times Q$ a transition relation, and $F \subseteq Q$ a set of accepting states. A word w is accepted by A (i.e., is part of the regular language corresponding to A) if there is a sequence of transitions $(q_0, w_1, q_1), \ldots, (q_{n-1}, w_n, q_n) \subseteq Q$, called a run of w, such that $w_1 w_2 \ldots w_n = w$ and $q_n \in F$.

Parametrized automata (PA) extend FSA in the following manner: instead of letters, the transitions in PA are labeled with logical formulas which also contain a finite number of variables. PA are therefore equipped to handle infinite alphabets. A PA is always defined in relation to a fixed underlying first-order theory T (represented by a structure M), and we require T-satisfiability to be decidable. We refer to [2] for a brief revision of the terminology.

Examples in this paper use the theory of real numbers as defined in [2], i.e., input letters are real numbers and transition formulas may contain the operations $+$, $-$ and $\cdot$ and the predicates $=$ and $\leq$.

Definition 1 (parametrized automata). *Let D be an infinite alphabet and $M = (D, I)$ be a structure. Let $Y = \{y_1, y_2, \ldots\}$ be an infinite set of variables, or parameters, and x be a distinguished variable representing the current input letter. Let Φ be the set of first-order formulas over M using variables from $\{x\} \cup Y$. A* parametrized automaton *is a tuple $A = (M, Q, q_0, \delta, F)$, where*

- *Q is a finite set of states,*
- *$q_0 \in Q$ is the initial state,*
- *$F \subseteq Q$ is the set of accepting states, and*
- *$\delta \subseteq_{fin} Q \times \Phi \times Q$ denotes the finite transition relation.*

Since δ is finite, each PA uses only a finite subset of parameters $Y_A \subset Y$. For a PA A, we call a function $\mu : Y_A \to D$ a parameter assignment. The set of all possible parameter assignments for A is denoted Θ_A.

A word $w = (w_1, \ldots, w_k) \in D^*$ is accepted by A if there exists a parameter assignment $\mu \in \Theta_A$ and a sequence of transitions $(q_0, \varphi_1, q_1), \ldots, (q_{k-1}, \varphi_k, q_k)$, called a complete run, such that $q_k \in F$ and for each i, φ_i evaluates to true when x is assigned w_i and parameters are assigned according to μ. The assignment μ is fixed throughout the run, and different words may be accepted using different parameter assignments.

We write $L(A)$ for the language accepted by A. Two PA A and A' are equivalent if $L(A) = L(A')$.

A symbolic automaton is therefore a PA in which no variables occur and predicates have to be drawn from an effective Boolean algebra. It can be shown (see [1]) that every variable automaton is equivalent to a PA which, aside from variables and constant symbols, only uses the predicates $=$, $\wedge$, $\vee$ and $\neg$ (and vice versa: every such PA is equivalent to a variable automaton). PA over finite alphabets identify exactly the regular languages, because all possible parameter assignments can be enumerated.

For a fixed parameter assignment μ, A_μ denotes the symbolic automaton obtained by replacing each parameter y in A's transition formulas with its value $\mu(y)$. The language $L(A)$ can thus be alternatively described as the union $\bigcup_{\mu \in \Theta} L(A_\mu)$.

We also point out that there is no straightforward way of defining determinism in PA, and introduce two different notions of determinism: we say that A is deterministic per assignment if each A_μ is a deterministic symbolic automaton. Because the corresponding algorithms (see [12]) can be lifted straight from symbolic automata, every PA can effectively be transformed to a PA that is deterministic per assignment. Determinism per assignment is a useful property that we will need later.

Observe that, in a PA that is deterministic per assignment, two different parameter assignments may still cause a word to complete two different runs. A different way to define determinism in PA would therefore be to demand that every word completes exactly one run over all possible parameter assignments. We call this property "strong determinism," and will explore the notion in more detail in subsection 3.2.

2.2 The Complementation Problem

Throughout this paper, we use $\cdot^c$ for the complement operation. We say that a PA A is *complementable* if there exists another PA A^c such that $L(A)^c = L(A^c)$. This leads to a first theorem.

Theorem 1. *Not every PA is complementable.*

Proof. Consider example A_1 seen in Figure 2, a PA using the theory of real numbers which randomly assigns a letter w_i of a word $w = (w_1, \ldots, w_k)$ to its

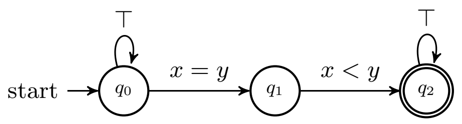


Fig. 2: A_1, a PA that cannot be complemented.

parameter y and accepts the word if the succeeding letter is smaller than $w_i = y$. As such, A_1 identifies unsorted words: $L(A_1) = \{w = (w_1, \ldots, w_k) \in D^* \mid \exists i.1 \leq i < k \wedge w_i > w_{i+1}\}$. Note that, by this definition, a word is sorted if its letters occur in *non-decreasing* order. An example of a sorted word is $(1, 2, 3)$, while $(1, 3, 2)$ is not a sorted word. The complement of $L(A_1)$ is the set of all sorted words $w = (w_1, \ldots, w_k)$ with the property $i < j \Rightarrow w_i \leq w_j$.

We will prove that a complement automaton identifying $L(A_1)^c$ cannot exist using a proof by contradiction that has a similar flavor as the pumping lemma (see [9, section 4.1]). Assume for contradiction that such an automaton A_1^c exists and has n states. Let $w = (1, 2, 3, \ldots, 2n + 1)$. Since $w \in L(A_1)^c$, there is a parameter assignment μ such that w completes an accepting run in $(A_1)_\mu$. Then by a counting argument, we can argue that the run of w in A_1^c traverses some state q thrice, creating a loop of length greater than 1. This loop is traversed by a subword $(i, i + 1, \ldots, i + k)$ of w where $k \geq 1$.

Therefore, the word $w' = (1, 2, \ldots, i+k-1, i+k, i, i+1, \ldots, 2n+1)$ obtained by repeating the found loop completes an accepting run in A_1^c. Now we see why it was necessary for w to traverse some state thrice: w' is not a sorted word because $i + k > i$. Therefore, A_1^c does not correctly identify the complement language of A_1. A PA identifying all sorted words cannot exist. $\square$

Other operations and decision problems. It can be shown that PA are closed under the Boolean operations of union and intersection, using a product construction similar to symbolic automata, see [12]. Here, for both automata to operate independently we have to ensure that their sets of parameters do not overlap; this can be achieved by renaming the parameters in one PA. We call this type of product construction the *direct product*.

We are interested in two particular decision problems, checking whether a PA accepts at least one word (the non-emptiness problem) and checking whether a PA accepts all words (the universality problem). The non-emptiness problem is decidable for PA because the finite set of paths without loops terminating in accepting states can be checked for T-satisfiability. The universality problem is not decidable, a trait that is inherited from variable automata (see [8]).

This, in turn, makes the complementation problem more challenging because of the following conclusion:

Theorem 2. *At least one of the following problems is non-computable:*

- *Deciding whether an arbitrary PA is complementable.*
- *For an arbitrary complementable PA, find a complement automaton.*

Proof. Assume that both problems are computable. Then we can solve the universality problem for PA: the universal language and its complement can both be represented by PA. If an arbitrary PA is not complementable, it is not universal. If an arbitrary PA is complementable and its complement can be computed, then the PA is universal if and only if its complement is empty. □

3 Complementable Normal Form

3.1 Definition and Examples

In PA, not all non-accepting states are created equal: some may be reached both by words in the language and words in the complement language, depending on the parameter assignment. Others can only be reached by words in the complement language, and this distinction is the idea behind complementable normal form.

Definition 2 (complementable normal form). *A PA $A = (M, Q, q_0, \delta, F)$ is in complementable normal form (called a CFPA) if there is a subset $F_c \subseteq Q \setminus F$ such that the automaton $C = (M, Q, q_0, \delta, F_c)$ identifies the complement of A, i.e., $L(C) = L(A)^c$.*

In a CFPA, the set of states Q is therefore partitioned into three pairwise disjoint subsets:

- F is the set of accepting states,
- F_c is the set of complement-accepting states,
- Any state not in F or in F_c is called a weak state. If the run of a word terminates in a weak state, we cannot deduce whether the word is part of $L(A)$ or $L(A)^c$.

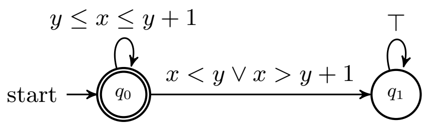
We illustrate CFPA using an example, and afterwards show that every complementable PA is equivalent to a CFPA.

Example 1. Consider the PA A_3, which can be seen in Figure 1b. The automaton corresponds to the language L_3 of all words whose letters fall within an interval of length 1. The automaton A_3 has an accepting state but no weak or complement-accepting states.

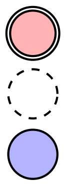
A_3 can be made deterministic per assignment by adding a sink state q_1 and redirecting a run to q_1 as soon as the condition $y \leq x \leq y + 1$ is broken by a letter (Figure 3a). The new state q_1 is a weak state.

We can obtain an equivalent CFPA by reasoning about the relationship between accepted words and the corresponding parameter assignment. For instance, every word in $L(A_3)$ is accepted if y is assigned the minimum letter of the word. If on the other hand, y is assigned the minimum letter of a word and then the condition $y \leq x \leq y + 1$ is broken, we know that that word could not have been part of $L(A_3)$.

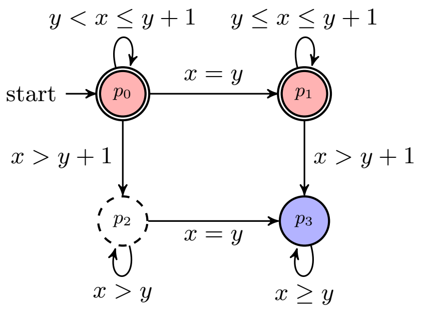
Based on this observation, we construct the equivalent CFPA C_3 seen in Figure 3c. There are no transitions permitting $x < y$: this forces the value



(a) The PA A'_3, which is deterministic per assignment.



(b) Symbols used in CFPA.



(c) A PA C_3 for L_3 in complementable normal form. State p_3 identifies the complement language.

Fig. 3: The language L_3 of words with letters within an interval of length 1.

assigned to y to be less than or equal to all letters of the word. If a letter satisfying $x < y$ is encountered, there is no viable exiting transition and the run is aborted before proper termination. The state p_3 can only be entered if y corresponds to the minimum letter and another letter satisfying $x > y + 1$ has been encountered, thus p_3 identifies the complement language correctly. If a letter satisfying $x > y + 1$ has been encountered before confirmation that y corresponds to the minimum letter, the weak state p_2 is entered and can be exited as soon as there is a confirmation that y corresponds to the minimum letter. Otherwise, as long as the condition $y \leq x \leq y + 1$ is not violated, we remain in the accepting states p_0 or p_1 depending on whether the minimum letter has already been encountered.

Theorem 3. *For every complementable PA, there is an equivalent CFPA.*

Proof. Let $A = (M, Q, q_0, \delta, F)$ be a PA and $A^c = (M, P, p_0, \delta_c, F_c)$ be a complement automaton of A. Assume without loss of generality that $Q \cap P = \varnothing$. We introduce a new initial state $r_0 \notin Q \cup P$ and copy each of the transitions exiting either q_0 or p_0 so they also exit r_0: let $\delta' = \{(r_0, \varphi, r) \mid (q_0, \varphi, r) \in \delta \vee (p_0, \varphi, r) \in \delta_c\}$. Exactly one of the automata A and A^c accepts the empty word ε, so either $q_0 \in F$ or $p_0 \in F_c$ holds. We need to assign r_0 to either the language-accepting or to the complement-accepting set accordingly. Without loss of generality, let $q_0 \in F$. Then the PA $A' = (M, Q \cup P \cup \{r_0\}, r_0, \delta \cup \delta_c \cup \delta', F \cup \{r_0\})$ is equivalent to A and the PA $(M, Q \cup P \cup \{r_0\}, r_0, \delta \cup \delta_c \cup \delta', F_c)$ is equivalent to A^c. Therefore, A' is a CFPA. □

Unfortunately, this expressivity comes at a cost that has already been discussed in Theorem 2: there is no algorithm that can both identify and then complement all complementable PA. The same restriction has to apply to CFPA.

Corollary 1. *There is no algorithm that can either transform a PA into an equivalent CFPA, or return that no such CFPA exists.*

CFPA are closed under union, intersection and complementation, all of which can be computed efficiently because the direct product of two CFPA is again a CFPA. In order to show that the universality problem is decidable for CFPA, we need to mind a little detail: the definition of CFPA requires the existence of a set of states F_c which accepts exactly the complement language, but knowing exactly which states belong to F_c is not required.

We do not need to communicate information about F_c along with a CFPA $A = (M, Q, q_0, \delta, F)$ because this information can be recovered algorithmically: we assign a state q to F_c if the intersection of A and $(M, Q, q_0, \delta, \{q\})$ is empty. Note that F_c may not be unique, and this algorithm identifies the largest possible set of complement-accepting states.

At this point, the undecidability of the universality problem throws another wrench into our gears. Every universal PA is naturally a CFPA whose set of complement-accepting states is empty. If there was an algorithm for identifying CFPA, we could take any universal PA, verify that it is a CFPA, identify the set F_c of complement-accepting states and then confirm that no words have runs terminating in F_c. This is a recipe for identifying universal PA, which would contradict the undecidability of the universality problem.

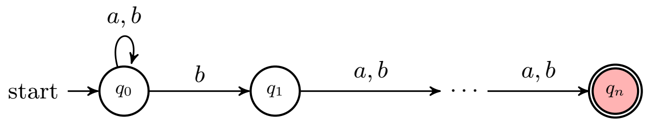
Corollary 2. *There is no algorithm that can decide whether a given PA is a CFPA.*

Example 2 (Finite-State Automata). Complementable normal form can also be applied to finite-state automata (FSA), and can lead to automata that are much smaller than their deterministic equivalent while still being easy to complement. For a fixed $n \in \mathbb{N}$, consider the automaton A for the regular language $(a + b)^*b(a + b)^n$.

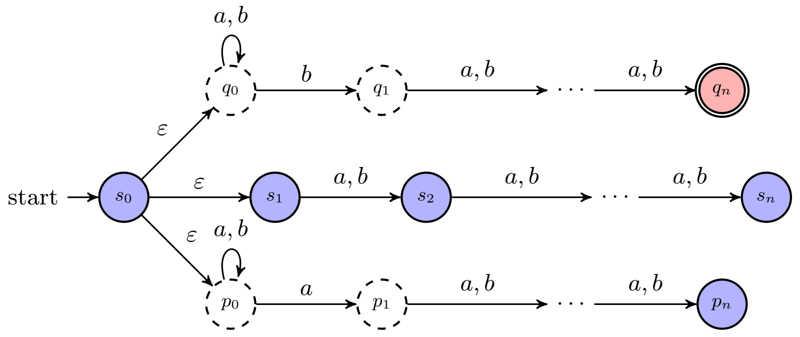
The automaton will be familiar to the reader, because it is a standard example to illustrate the exponential blowup when transforming a nondeterministic finite-state automaton into a deterministic one. Both A and its smallest nondeterministic complement automaton have $O(n)$ states, but a deterministic FSA that is equivalent to A will have $O(2^n)$ states. In the FSA in Figure 4b, we have added two additional "tracks" for all words that cannot be accepted by A: one for words where the nth letter from the last is an a, and one for all words that have fewer than n letters. The resulting FSA is in complementable form. Just like deterministic FSA, it can accept either $L(A)$ or $L(A)^c$ depending on how the set of accepting states is chosen. The size of this automaton is still $O(n)$, a significant improvement compared to deterministic FSA.

3.2 Strong Determinism

Complementation of FSA is easy if the FSA in question is deterministic. In a deterministic FSA, every word completes exactly one run, so the word is part



(a) A finite-state automaton A accepting all words in which the nth letter from the end is a b.



(b) This FSA is equivalent to A and in complementable normal form. The states that are not weak states accept the complement language of A.

Fig. 4: Complementable Normal Form in finite-state automata.

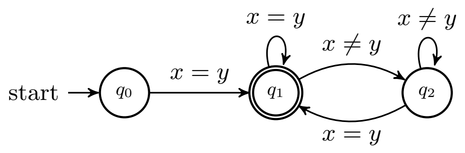


Fig. 5: D, an SDPA identifying all words whose first and last letter coincide.

of the complement if and only if its unique run terminates in a non-accepting state. We will briefly explore what happens if the same notion of determinism is applied to PA, and why this approach might be inferior to CFPA.

Definition 3. *A PA is called* strongly deterministic *(or an SDPA) if every word completes exactly one run.*

In contrast to determinism per assignment, in an SDPA every word has to complete a unique run considering *all possible parameter assignments*. The two notions are strictly orthogonal: SDPA are in general not deterministic per assignment, and vice versa. An example of an SDPA can be seen in Figure 5.

SDPA are a subset of CFPA in which no weak states occur. As a consequence, SDPA can be complemented in the same manner as deterministic FSA.

Obviously, not every PA is equivalent to some SDPA, because SDPA are always complementable. In fact, this limitation goes even further, because there are even some complementable PA that are not equivalent to any SDPA.

Theorem 4. *Strongly deterministic PA form a strict subset of complementable PA. There are complementable PA that do not have a strongly deterministic equivalent.*

Proof. We claim that the language $K = \{w = (w_1, \ldots, w_k) \in D^* \mid \exists 1 \leq i < j \leq k : |w_i - w_j| > 1\}$, which is the complement of L_3 and therefore recognized by a PA, cannot be recognized by any SDPA.

Consider the sequence of words $((1, \frac{1}{2}, \ldots, \frac{1}{n}, 1 + \frac{1}{n-1}))_{1<n\in\mathbb{N}}$, whose first few elements are $(1, \frac{1}{2}, 2)$, $(1, \frac{1}{2}, \frac{1}{3}, 1 + \frac{1}{2})$, $(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, 1 + \frac{1}{3}), \ldots$.

Assume that there is an SDPA A identifying K, and let A consist of k states. Each word of the sequence has to be accepted by A upon reaching the last letter, and all runs of prefixes have to terminate in non-accepting states. We make use of a peculiar property of SDPA: If two words w and v complete runs in A using the parameter assignments μ_w and μ_v, respectively, and if w is a prefix of v, then the runs of w in A_{μ_w} and A_{μ_v} have to be identical. It can be shown by induction (which we skip here; the full version of the proof is available in [1]) that all runs of proper prefixes of the sequence, which have the form $(1, \frac{1}{2}, \ldots, \frac{1}{n})$ for some n, have to traverse n distinct states in A.

The final state is accepting and therefore cannot coincide with any state that has been previously traversed. Therefore, the run of the word $(1, \frac{1}{2}, \ldots, \frac{1}{n}, 1 + \frac{1}{n-1})$ traverses $n + 1$ distinct states.

A run of the word $(1, \frac{1}{2}, \ldots, \frac{1}{k}, 1 + \frac{1}{k-1})$ needs to traverse $k+1$ distinct states, contradicting the assumption of A only having k states. □

In conclusion, SDPA only represent a fraction of all complementable PA. Therefore, CFPA are the superior choice, as they are both strictly more expressive and share the same pleasant properties regarding closure under Boolean operations and decidability of important decision problems.

4 Construction of CFPA

Idea. We will now present a method for transforming arbitrary complementable PA into CFPA. We generalize the approach of Example 1, where the relationship between words and accepting parameter assignments was exploited.

For the algorithm, we introduce two new concepts:

Definition 4 (Skolem automaton). *Let A be a PA. Let B be a PA such that:*

- *B is universal,*
- *the parameters of A are a subset of the parameters of B: $Y_A \subseteq Y_B$, where Y_A and Y_B are the finite sets of parameters used by A resp. B, and*
- *for all $w \in L(A)$, if $w \in L(B_\mu)$ for some $\mu \in \Theta$ then $w \in L(A_\mu)$.*

Then B is called a Skolem automaton of A.

We can use Skolem automata for complementation thanks to the contrapositive of the third statement: if a word is accepted by B using a parameter assignment μ, but is not accepted by A_μ, this implies that the word is part of the complement language.

Definition 5 (synchronized product). *Let $A = (M, Q, q_0, \delta_A, F_A)$ and $B = (M, P, p_0, \delta_B, F_B)$ be two PA whose parameter sets may intersect, i.e., $Y_A \cap Y_B \neq \varnothing$. Let $F \subseteq F_A \times F_B$ be arbitrary, and let $\delta = \{((q,p), \varphi_1 \wedge \varphi_2, (q',p')) \mid (q, \varphi_1, q') \in \delta_A, (p, \varphi_2, p') \in \delta_B\}$. Then the PA $(A \otimes B, F) := (M, Q \times P, (q_0, p_0), \delta, F)$ is called a synchronized product of A and B. The parameter set of $(A \otimes B, F)$ is $Y_A \cup Y_B$.*

The synchronized product allows a PA and its Skolem automaton to exchange information. Constraints placed on the parameter assignment by B also have to hold in A.

Theorem 5. *Let $A = (M, Q, q_0, \delta, F_A)$ be a PA that is deterministic per assignment. Let $B = (M, P, p_0, \delta', F_B)$ be a Skolem automaton of A. Then the synchronized product $(A \otimes B, F_A \times F_B)$ is equivalent to A and is in complementable normal form. The complement of $L(A)$ is accepted by the set of states $(Q \setminus F_A) \times F_B$.*

Proof. Let $w \in L(A)$. Since B is universal, there is a parameter assignment $\mu \in \Theta$ such that $w \in L(B_\mu)$. By the definition of B, this means $w \in L(A_\mu)$, and therefore, $L(A) \subseteq L((A \otimes B, F_A \times F_B))$.

Let now $w \in L((A \otimes B, F_A \times F_B))$. Then there exists a parameter assignment μ such that $w \in L(B_\mu)$ and $w \in L(A_\mu)$. Therefore, $L((A \otimes B, F_A \times F_B)) \subseteq L(A)$. This concludes the proof that $L(A) = L((A \otimes B, F_A \times F_B))$.

In order to prove that $(A \otimes B, F_A \times F_B)$ is in complementable normal form, it is sufficient to prove that $(A \otimes B, (Q \setminus F_A) \times F_B)$ identifies the complement of $L(A)$. Let $w \in L(A)^c$. As B is universal, there is a parameter assignment $\mu \in \Theta$ such that $w \in L(B_\mu)$. Since A is deterministic per assignment, w completes a run in A_μ which terminates in a non-accepting state since $w \notin L(A)$. Therefore, w completes a run in $(A \otimes B)_\mu$ which terminates in $(Q \setminus F_A) \times F_B$.

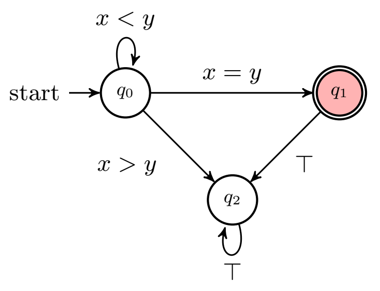
Vice versa, let a word w terminate in $(p, b) \in (Q \setminus F_A) \times F_B$ for some parameter assignment μ. Thus, $w \in L(B_\mu)$. If w were in $L(A)$, then the the third condition would force the run of w in A_μ to terminate in an accepting state. However, because w terminates its run in A_μ in a non-accepting state, w cannot lie in $L(A)$. □

Example 3. Figure 6 illustrates the method with an example based on A_2 seen in Figure 1, the automaton which accepts all words in which the last letter is largest.

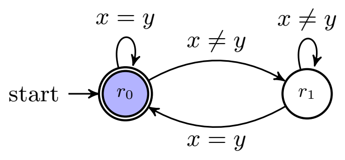
Applicability. According to Theorem 3, for every complementable PA, there is an equivalent CFPA. A similarly structured proof can show the existence of a Skolem automaton for every complementable PA:

Proposition 1. *For every complementable PA A, a suitable Skolem automaton B exists.*

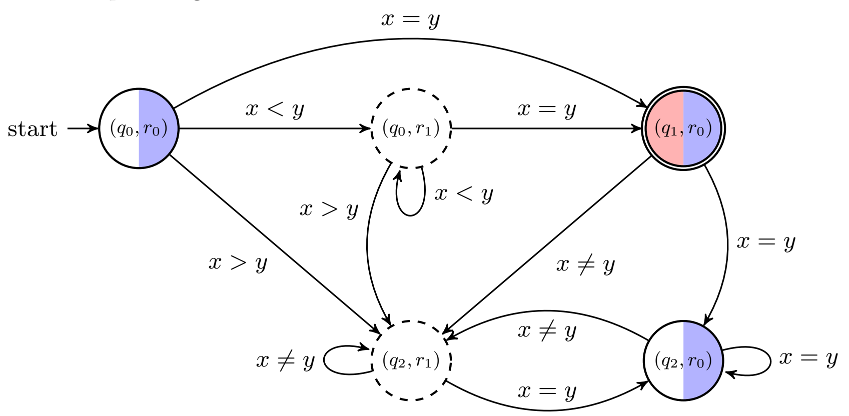
Proof. The product automaton in the proof of Theorem 3 is a Skolem automaton when picking the set of accepting states $F \cup F_c \cup \{r_0\}$. □



(a) A_2', which is equivalent to A_2 and deterministic per assignment.



(b) B, a suitable Skolem automaton for A_2'.



(c) The synchronized product of A_2' and B, a CFPA.

Fig. 6: An example of our method. Weak states are indicated by dashed lines, complement-accepting states are half blue and accepting states are half red and half blue and marked as accepting.

Skolem automata are not unique: for example, a CFPA equivalent to A_2 can also be obtained by demanding that y is the maximum letter of the word.

A Skolem automaton has to fulfill three properties, which each can be confirmed with widely differing degrees of difficulty: the universality problem is undecidable, while $Y_A \subseteq Y_B$ is easily checked. The third property can be verified as well, hinging on the complexity of the non-emptiness problem.

Proposition 2. *Given a PA A that is deterministic per assignment and a universal PA B, it can be decided whether B is a Skolem automaton of A.*

Proof. We need to prove that for every $w \in L(A)$ and every parameter assignment μ, $w \in L(B_\mu)$ implies $w \in L(A_\mu)$. The condition is breached if there is a $w \in D^*$ and a μ such that $w \in L(A) \cap L(B_\mu) \cap L(A_\mu)^c$. Such a w exists if and only if the intersection of $L(A)$ and $\bigcup_{\mu \in \Theta}(L(B_\mu) \cap L(A_\mu)^c)$ is non-empty, and $\bigcup_{\mu \in \Theta}(L(B_\mu) \cap L(A_\mu)^c)$ corresponds to the synchronized product of B and A. □

5 Conclusion

In our search for new approaches to the complementation problem for PA, we have identified CFPA as a formalism that sits in a particularly sweet spot. CFPA are powerful enough to completely characterize the class of complementable PA, yet at the same time complementation of CFPA is easy. Their computational properties are pleasant and comparable in complexity to those of SDPA, a model that is far more restrictive.

Because of the undecidability of the universality problem for PA in general, per Theorem 2, there can be no algorithm that either complements a given PA or returns that a complement PA does not exist. Because of this restriction, we have closely monitored the effects of Theorem 2. This has allowed us to identify the parts of the problem that are decidable to a reasonable degree.

We have shown that every complementable PA has a Skolem automaton, and studied how to confirm whether a given PA is a Skolem automaton (barring the universality property, which is of course undecidable). Given a Skolem automaton, the synchronized product with the PA as well as the complement of the PA can be constructed. The big, remaining, unmendable hole in the method is therefore the construction of suitable Skolem automata.

This is interesting, because the relationship between complementable PA and Skolem automata goes both ways: a PA is complementable iff it has a Skolem automaton as in Definition 4. The complementation problem can thus be rephrased "compute a Skolem automaton, or return that no such Skolem automaton exists." We point out that one part of the problem might be computable, as long as the other part is not. Future research might identify the computable part, or establish that both parts of the problem are non-computable.

The gap can also be closed with an application-oriented approach. We are delighted to announce that a library for PA is currently being implemented. The library will support all operations and algorithms described in this paper, make PA more accessible for applications and open up new avenues of research.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Acknowledgement. This work was supported by the Swedish Research Council through grant 2021-06327, and by the Knut and Alice Wallenberg Foundation through project UPDATE. We are grateful for the useful feedback by the anonymous referees!

References

1. Franziska Alber. Parametrized automata over infinite alphabets: Properties and complementation. *Master Thesis*, 2024.

2. Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007.
3. Loris D'Antoni, Tiago Ferreira, Matteo Sammartino, and Alexandra Silva. Symbolic register automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2019.
4. Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 47–67. Springer, 2017.
5. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
6. Diego Figueira and Anthony Widjaja Lin. Reasoning on data words over numeric domains. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 37:1–37:13. ACM, 2022.
7. Jonathan Goldstine, Martin Kappes, Chandra M. R. Kintala, Hing Leung, Andreas Malcher, and Detlef Wotschke. Descriptional complexity of machines with limited resources. *J. Univers. Comput. Sci.*, 8(2):193–234, 2002.
8. Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Variable automata over infinite alphabets. In Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, volume 6031 of *Lecture Notes in Computer Science*, pages 561–572. Springer, 2010.
9. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition.* Pearson international edition. Addison-Wesley, 2007.
10. Artur Jez, Anthony W. Lin, Oliver Markgraf, and Philipp Rümmer. Decision procedures for sequence theories. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*, volume 13965 of *Lecture Notes in Computer Science*, pages 18–40. Springer, 2023.
11. Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
12. Margus Veanes, Peli de Halleux, and Nikolai Tillmann. Rex: Symbolic regular expression explorer. In *Third International Conference on Software Testing, Verification and Validation, ICST 2010, Paris, France, April 7-9, 2010*, pages 498–507. IEEE Computer Society, 2010.