The Beauty of Predicate Automata

Azadeh Farzan[1] and Zachary Kincaid[2]

[1] University of Toronto
[2] Princeton University

Abstract. In [6], we introduced a class *data* automata, called *Predicate Automata*, which recognize languages over an alphabet $\Sigma \times \mathbb{N}$ where Σ is a finite alphabet. These automata were designed to capture the behaviour of parametrized programs, i.e. programs with an unbounded number of threads, and their correctness proofs. Since the focus of [6] was on the verification problem, the paper does not investigate the properties of these automata beyond the minimum which is necessary for the verification problem. In this paper, we study some of their key properties and relate them to other classes of data automata.

1 Introduction

For verification of concurrent and distributed programs, which consist of components executing simultaneously, the axiomatic approach to program reasoning was extended by Owicki/Gries [15] and Jones [10] to one that aims to construct a proof of correctness for the whole program out of proofs for individual components (a thread or a process) and some small glue that connects them together. This reasoning is based on the principle of non-interference: other threads/processes do not interfere with the functionality of the current one, and therefore do not have much impact on its proof of correctness.

Not all concurrent programs are designed with this principle in mind. For cooperative concurrency, in which processes work together to achieve some common goal, the notion of non-interference is not natural. Axiomatic proof systems have clever workarounds (e.g., ghost state) for these cases, but cleverness is an obstacle to automation: human verifiers are clever, but automated verifiers are not. In contrast, informal *operational arguments*, accommodate arbitrary global facts into proofs more naturally. Operational reasoning is very common among programmers as an informal way of reasoning about correctness of programs. One can think of it as a way of grouping similar program behaviours into a handful of *scenarios*, and arguing that the program behaves correctly in each *scenario*. One can think of doing operational reasoning using the following general proof rule:

$$\frac{P \subseteq \Pi, \;\; \Pi \text{ is correct by construction}}{P \text{ is correct}} \qquad \text{(OR)}$$

where P stands for the set of behaviours of the program, and Π is a set of behaviours that are known to be correct by construction. The *subsumption* test

$P \subseteq \Pi$ certifies P as correct.

In [6, 7], we proposed algorithmic tools to implement the above proof rule as a sound and (in some sense) complete verification algorithm for parameterized programs. The thesis of our work is that operational arguments can be formalized using axiomatic proofs of a programs runs, which are sequences of instructions that are executed in order (without conditional branching, looping, etc). The basic object of interest in such a proof system is a valid Hoare triple, $\{P\}\ \rho\ \{Q\}$, which consists of two assertions P and Q (called the pre-condition and post-condition of the triple, respectively), and a run ρ. Validity of the triple means that if ρ begins in a state that satisfies the pre-condition, it must end in a state that satisfies the post-condition. Hoare triples form the foundation of most axiomatic proof systems; the distinguishing feature of our notion of operational reasoning is that the program must be straight-line. This divergence reflects a difference of perspective: in, say, Hoare's proof system, we think of a program as a syntax tree, which is constructed from other syntax trees by certain formation rules (sequential composition, while loops, if-then-else statements, etc). In an operational argument, we think of a program as a set of behaviours, corresponding to the possible paths of execution the program may take.

Any algorithmic implementation of Rule OR requires (1) an effective representation for Π and P, and (2) a way of constructing Π, and (3) a decidable subsumption test for $P \subseteq \Pi$. First, we briefly recall how we addressed (1) and (3) in [6] using a new model of infinite state automata, called *Predicate Automata* (PA). These two points inspired the design of this class of automata, whereas (2) is orthogonal to the investigation of this paper.

1.1 Proof Spaces

We proposed to construct a formal operational proof, like Π in Rule OR, around a finite set of valid Hoare triples H which can be associated with an infinite set of Hoare triples $S(H)$, comprised of Hoare triples that be obtained from H using the following three rules of inference:

- *Sequencing:* if $\{P_1\}\ \rho_1\ \{P_2\}$ and $\{P_2\}\ \rho_2\ \{P_3\}$ are valid triples, then so is $\{P_1\}\ \rho_1; \rho_2\ \{P_3\}$[3]
- *Conjunction:* if $\{P_1\}\ \rho\ \{Q_1\}$ and $\{P_2\}\ \rho\ \{Q_2\}$ are valid, then so is $\{P_1 \wedge P_2\}\ \rho\ \{Q_1 \wedge Q_2\}$
- *Symmetry:* if $\{P\}\ \rho\ \{Q\}$ is a valid, then any triple $\{P'\}\ \rho'\ \{Q'\}$ that can be obtained by consistently renaming thread identifiers in all three components of the Hoare triple is also valid

We call the set of Hoare triples $S(H)$ a **proof space**, and H a basis for $S(H)$. We say that a proof space proves that a set of runs $\mathcal{R}$ is correct with respect to a specification $\mathsf{Pre}/\mathsf{Post}$ if for each run ρ in $\mathcal{R}$, the Hoare triple $\{\mathsf{Pre}\}\ \rho\ \{\mathsf{Post}\}$ belongs to the proof space $S(H)$ generated by H; that is, there is a derivation

[3] The sequencing rule in [6] also incorporates a weak form of the rule of consequence, which we omit here for brevity.

of $\{\mathsf{Pre}\}\ \rho\ \{\mathsf{Post}\}$ using the sequencing, conjunction, and symmetry rules, using the Hoare triples in H as axioms.

1.2 A Suitable Class of Automata

Let us now turn our attention to the subsumption test in the premise of Rule OR. A proof space is a proof for the program if and only if for every possible run of the program, there exists a valid Hoare triple proving its correctness in the proof space. If the proof space is represented by a finite basis, then this means that the Hoare triple for the run can be derived by the repeated application of the three rules (sequencing, conjunction, and symmetry).

We reduce the problem of deciding whether such a derivation exists for a given run to checking acceptance of the run (represented as a word) by a *Predicate Automaton* (PA). It is easy to see that the sequencing rule corresponds to the sequencing of transitions of an automaton. In this view, program statements are associated with letters of the alphabet of the automaton, and proof assertions with the states of the automaton. Predicate automata use the notion of *alternation* from Alternating Finite Automata (AFA) [3] to additionally accommodate the conjunction rule; this allows us to account for concurrent programs with a fixed number of threads. Finally, to accommodate also the symmetry rule (and account for concurrent programs with an unbounded number of threads), predicate automata make a shift from finite alphabets to alphabets indexed on natural numbers, in the spirit of *data* words [1].

1.3 Properties of Predicate Automata

In [6], we introduced predicate automata and briefly (i.e. within the confined limits of conference paper page limits) stated its core properties as relevant to the task of verification of parametrized systems. In this paper, we would like to take the opportunity to expand on our exposition of this simple yet powerful data automata model and its properties in connection to both verification and other automata models.

Specifically, in this paper:

- We establish that predicate automata are equivalent in expressive power to a variant of Alternating Register Automata (ARA) [1].
- We investigate properties of languages recognized by predicate automata, and in particular, we prove that they are not closed under reversal.
- We investigate the properties of the images of these languages under a homomorphism that forgets the *data* part of the alphabet, and in particular show that these languages are not necessarily context free.
- We give an alternative proof of decidability of verification of parameterized boolean programs, by giving a construction for a *most general proof* Π_{bool}, as a *monadic* predicate automaton, that subsumes *all* correct parametrized boolean programs over a fixed set of commands. This, combined by the results already presented in [6] that parametrized programs can also be

modelled using *monadic* predicate automata and the subsumption test is decidable for monadic predicate automata, amounts to the new argument.

2 Predicate Automata

Predicate automata [6] are a class of infinite-state automata which recognize languages over an infinite alphabet of the form $\Sigma \times \mathbb{N}$.[4] We use the notation $a : \mathtt{i}$ or $\langle a : \mathtt{i} \rangle$ (with $a \in \Sigma$ and $\mathtt{i} \in \mathbb{N}$) to denote an element of this alphabet. A predicate automaton (PA) is equipped with a *vocabulary* $\langle Q, ar \rangle$ (in the usual sense of first-order logic) consisting of a finite set of predicate symbols Q and a function $ar : Q \to \mathbb{N}$ which maps each predicate symbol to its arity.

A state of a PA is a proposition $q(\mathtt{i}_1, ..., \mathtt{i}_{ar(q)})$ with $q \in Q$ and $\mathtt{i}_1, ..., \mathtt{i}_n \in \mathbb{N}$. The transition function maps such states to formulas in the vocabulary of the PA; disjunction corresponds to nondeterministic (existential) choice, and conjunction corresponds to universal choice. It is important to note that the symbols $q \in Q$ are "uninterpreted": they have no special semantics, and any subset of $\mathbb{N}^{ar(q)}$ is a valid interpretation of q.

For readers familiar with alternating finite automata (AFA) [3, 4], a helpful analogy might be that predicate automata are to first-order logic what AFA are to propositional logic.

Given a vocabulary $\langle Q, ar \rangle$, let the set of *positive formulas* $\mathcal{F}(Q, ar)$ be the negation-free formulas over $\langle Q, ar \rangle$ where each atom is either (1) a proposition of the form $q(i_1, ..., i_n)$ (where $i_1, ..., i_n$ are natural number-typed variables), or (2) an equation $i = j$ (where i, j are variable symbols), or (3) a disequation $i \neq j$ (where i, j are variable symbols). A *ground* formula is defined similarly, except that natural numbers take the place of natural number-typed variables; the set of ground formulae over $\langle Q, ar \rangle$ is denoted $\underline{\mathcal{F}}(Q, ar)$. We adopt the convention that i, j, k range over natural number-typed variables, while $\mathtt{i}, \mathtt{j}, \mathtt{k}$ range over natural numbers. Predicate automata are defined as follows:

Definition 1 (Predicate automata [6]). *A* predicate automaton *(PA) is a 6-tuple* $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$ *where*

- $\langle Q, ar \rangle$ *is a vocabulary*
- Σ *is a finite alphabet*
- $\varphi_{\mathsf{start}} \in \mathcal{F}(Q, ar)$ *is an initial formula with no free variables.*
- $F \subseteq Q$ *is a set of accepting predicate symbols*
- $\delta : Q \times \Sigma \to \mathcal{F}(Q, ar)$ *is a transition function which satisfies the property that for any* $q \in Q$ *and* $\sigma \in \Sigma$*, the free variables of* $\delta(q, \sigma)$ *are members of the set* $\{i_0, ..., i_{ar(q)}\}$. □

We can think of $\delta(q, \sigma)$ as a rewrite rule which instantiates the (implicit) formal parameters $i_1, ..., i_{ar(q)}$ of q to the actual parameters $\mathtt{i}_1, ..., \mathtt{i}_n$ and instantiates i_0 to $\mathtt{k}$ (the index of the letter being read). In light of this interpretation,

[4] Such languages are commonly called *data languages* [14].

we will often write δ in a form that makes the formal parameters explicit: for example, instead of

$$\delta(q,\sigma) = (i_0 \neq i_1 \wedge (q(i_0,i_1) \vee q(i_1,i_2))) \vee (i_0 = i_1 \wedge q(i_1,i_2) \wedge q(i_2,i_1))$$

we typically write

$$\delta(q(i,j),\langle\sigma : k\rangle) = (k \neq i \wedge (q(k,i) \vee q(i,j))) \vee (k = i \wedge q(i,j) \wedge q(j,i))$$

or more succinctly:

$$q(i,j) \xrightarrow{\sigma:k} (k \neq i \wedge (q(k,i) \vee q(i,j))) \vee (k = i \wedge q(i,j) \wedge q(j,i)) \ .$$

The arity of a PA is the a maximum arity among all of its predicate symbols. We call a PA *monadic* if this maximum arity is at most one.

In the following, we define the language accepted by a predicate automaton in two different (but equivalent) ways. The first is via an *extended transition relation*, which treats a PA as an deterministic automaton whose states are formulas, where each state is a ground positive formula over the vocabulary of the PA. This semantics is convenient (for instance) for relating predicate automata with Boolean programs (Section 5). The second definition treats a PA as an nondeterministic automaton whose states are conjunctive formulas. This is the semantics presented in [6], and is the basis of the decision procedure for checking of a monadic PA.

Deterministic semantics Let $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F\rangle$ be a predicate automaton. We may extend the transition relation δ to a function $\delta^* : \underline{\mathcal{F}}(Q, ar) \times (\Sigma \times \mathbb{N})^* \to \underline{\mathcal{F}}(Q, ar)$ as follows:

$$\begin{aligned}
\delta^*(F \wedge G, w) &\triangleq \delta^*(F, w) \wedge \delta^*(G, w)\\
\delta^*(F \vee G, w) &\triangleq \delta^*(F, w) \vee \delta^*(G, w)\\
\delta^*(i = j, w) &\triangleq i = j\\
\delta^*(i \neq j, w) &\triangleq i \neq j\\
\delta^*(F, \epsilon) &\triangleq F\\
\delta^*(q(\mathtt{j}_1, \ldots, \mathtt{j}_n), w\langle a : \mathtt{j}\rangle) &\triangleq \delta^*(\delta(q,a)[i_0 \mapsto \mathtt{j}, i_1 \mapsto \mathtt{j}_1, \ldots, i_n \mapsto \mathtt{j}_n], w)
\end{aligned}$$

Note that the automaton reads the input word backwards (right-to-left).

The *accepting formulas* of A are defined as follows:

- $q(\mathtt{i}_1, \ldots, \mathtt{i}_n)$ is accepting iff $q \in F$
- $\mathtt{i} = \mathtt{i}$ is accepting
- $\mathtt{i} \neq \mathtt{j}$ (with $\mathtt{i}$ and $\mathtt{j}$ distinct) is accepting
- If F is accepting, then $F \vee G$ and $G \vee F$ are accepting for any G
- If F and G are accepting, so is $F \wedge G$

Finally, we say that a word w is accepted by a PA A if the formula $\delta^*(\varphi_{\mathsf{start}}, w)$ is accepting, and define $\mathcal{L}(A)$ to be the set of all words accepted by A.

While checking emptiness of $\mathcal{L}(A)$ for a PA A is undecidable in general, it is decidable for *monadic* predicate automata [6]. The decision procedure is based on a *nondeterministic* semantics, which we describe in the following.

Nondeterministic semantics The high-level idea is that, rather than viewing a "state" of the automaton as a ground formula, we view it as a *conjunctive* formula which we call a *configuration*. For any configuration C and any indexed letter $\langle a : \mathtt{i} \rangle$, C may nondeterministically transition to any of $C_1, \ldots, C_n$, where $C_1 \vee \cdots \vee C_n$ is the disjunctive normal form of $\delta^*(C, a : \mathtt{i})$. Formally,

Definition 2 (Configuration). *Let $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$ be a PA. A* configuration $\mathcal{C}$ *of A is finite set of ground propositions of the form $q(\mathtt{i}_1, ..., \mathtt{i}_{ar(q)})$, where $q \in Q$ and $\mathtt{i}_1, ..., \mathtt{i}_{ar(q)} \in \mathbb{N}$. We may a configuration $\mathcal{C}$ with the conjunctive formula $\bigwedge_{q(\mathtt{i}_1,...,\mathtt{i}_{ar(q)}) \in \mathcal{C}} q(\mathtt{i}_1, ..., \mathtt{i}_{ar(q)})$.*

A PA $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$ induces a transition relation on configurations as follows: $\mathcal{C} \xrightarrow{\sigma:\mathtt{k}} \mathcal{C}'$ if $\mathcal{C}'$ is a clause in the DNF of the ground formula

$$\bigwedge_{q(\mathtt{i}_1,...,\mathtt{i}_{ar(q)}) \in \mathcal{C}} \delta(q, \sigma)[i_0 \mapsto \mathtt{k}, i_1 \mapsto \mathtt{i}_1, ..., i_{ar(q)} \mapsto \mathtt{i}_{ar(q)}]$$

Note also that the formula above may contain equalities and disequalities, but since they are ground (have no free variables), they are equivalent to either *true* or *false*, and thus can be simplified.

A configuration is *initial* if it is a cube of the DNF of φ_{start}. A configuration is *accepting* if for all $q(\mathtt{i}_1, ..., \mathtt{i}_{ar(q)}) \in \mathcal{C}$, we have $q \in F$; otherwise, it is *rejecting*. Finally, a word $w = \langle \sigma_1 : \mathtt{i}_1 \rangle \cdots \langle \sigma_n : \mathtt{i}_n \rangle$ is accepted by A if there is a sequence of configurations $\mathcal{C}_{n+1}, ..., \mathcal{C}_0$ such that:

1. $\mathcal{C}_{n+1}$ is initial
2. for each $r < n$, $\mathcal{C}_{r+1} \xrightarrow{\sigma_r:\mathtt{i}_r} \mathcal{C}_r$
3. $\mathcal{C}_0$ is accepting

It is straightforward to see that the deterministic and nondeterministic semantics of a predicate automaton define the same language.

The key observation of regarding the nondeterministic semantics is that we can define a covering relation $\preceq$ on configurations such that the transition relation is downwards-compatible with $\preceq$: if a configuration $\mathcal{C}'$ has a w-labeled path to an accepting configuration and $\mathcal{C} \preceq \mathcal{C}'$, then $\mathcal{C}$ must have a w'-labeled path to an accepting configuration for some relabeling w' of w. Formally,

Definition 3 (Covering). *Given a PA $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathit{init}}, F \rangle$, we define the covering pre-order on the configurations of A as follows: if $\mathcal{C}$ and $\mathcal{C}'$ are configurations of A, then $\mathcal{C} \preceq \mathcal{C}'$ ("$\mathcal{C}$ covers $\mathcal{C}'$") if there is a permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that for all $q \in Q$ and all $q(\mathtt{i}_1, \ldots, \mathtt{i}_{ar(q)}) \in \mathcal{C}$, we have $q(\pi(\mathtt{i}_1), \ldots, \pi(\mathtt{i}_{ar(q)}))$.*

Lemma 1 (Downwards compatibility [6]). *Let $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$ be a PA, and let $\mathcal{C}, \mathcal{C}'$ be configurations of A such that $\mathcal{C} \preceq \mathcal{C}'$. Then we have the following:*

1. *If $\mathcal{C}'$ is accepting, then $\mathcal{C}$ is accepting.*

2. *For any* $\langle \sigma : \mathtt{j} \rangle \in \Sigma \times \mathbb{N}$*, if we have*

$$\mathcal{C}' \xrightarrow{\sigma:\mathtt{j}} \overline{\mathcal{C}}'$$

then there exists a configuration $\overline{\mathcal{C}}$ *and an index* $\mathtt{k} \in \mathbb{N}$ *such that*

$$\mathcal{C} \xrightarrow{\sigma:\mathtt{k}} \overline{\mathcal{C}}$$

and $\overline{\mathcal{C}} \preceq \overline{\mathcal{C}}'$*.* □

If we imagine an emptiness checking algorithm as searching the reachable states of a PA for an accepting configuration, then the significance of downwards compatibility is that we may prune from the search space any configuration that is covered by another. Thus pruning is sufficient to make the search space finite for *monadic* predicate automata, for which $\preceq$ is a well-quasi order (i.e., for any infinite sequence of configurations $\mathcal{C}_1, \mathcal{C}_2, \ldots$, there is some $i < j$ such that $\mathcal{C}_i \preceq \mathcal{C}_j$) [6].[5]

Theorem 1 ([6]). *Checking emptiness of monadic PAs is decidable.*

2.1 Predicate Automata with ϵ-transitions

In many classes of automata, for example nondeterministic finite automata (NFA) or nondeterministic pushdown automata, the addition of epsilon transitions does not change the expressive power of the model. This is also true for predicate automata.

Definition 4 (ϵ-Predicate Automata). *An ϵ-Predicate Automaton (ϵ-PA) is a 7-tuple $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F, \epsilon \rangle$ where $\langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$ is a predicate automaton and $\epsilon : Q \to \mathcal{F}(Q, ar)$ is a ϵ-transition function which satisfies the property that for any $q \in Q$, the free variables of $\delta(q, \sigma)$ are members of the set $\{i_1, ..., i_{ar(q)}\}$.*

The nondeterministic semantics of PA can be extended to ϵ-PA as follows. Define a labeled transition relation $\to^*$ on configurations to be the least relation such that

- If $\mathcal{C}'$ is cube of the DNF of $\epsilon(q)[i_1 \mapsto \mathtt{i}_1, \ldots, i_{ar(q)} \mapsto \mathtt{i}_{ar(q)}]$, then

$$\mathcal{C} \wedge q(\mathtt{i}_1, \ldots, \mathtt{i}_{ar(q)}) \xrightarrow{\epsilon}{}^* \mathcal{C} \wedge \mathcal{C}'$$

- If $\mathcal{C}'$ is cube of the DNF of $\delta^*(\mathcal{C}, \langle a : i \rangle)$, then $\mathcal{C} \xrightarrow{\langle a:i \rangle}{}^* \mathcal{C}'$
- If $\mathcal{C} \xrightarrow{u} \mathcal{C}'$ and $\mathcal{C}' \xrightarrow{v} \mathcal{C}''$, then $\mathcal{C} \xrightarrow{uv}{}^* \mathcal{C}''$

Finally, define $\mathcal{L}(A)$ to be the set of words w such that there exists an initial configuration $\mathcal{C}$ and an accepting configuration $\mathcal{C}'$ such that $\mathcal{C} \xrightarrow{w^R}{}^* \mathcal{C}'$.

[5] That is, monadic predicate automata are well-structured transition systems [9].

Lemma 2 (ϵ-elimination). *Let A be an ϵ-PA. Then we can construct a predicate automaton B from A such that $\mathcal{L}(A) = \mathcal{L}(B)$.*

Proof. Let $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F, \epsilon \rangle$ be an ϵ-PA. We show that we can construct a function $\overline{\epsilon}$ such that $B = \langle Q, ar, \Sigma, \overline{\epsilon} \circ \delta, \overline{\epsilon}(\varphi_{\mathsf{start}}), F \rangle$ accepts the same language as A. Define a sequence of functions $\epsilon_0, \epsilon_1, \ldots$ mapping $\mathcal{F}(Q, ar)$ to $\mathcal{F}(Q, ar)$ as follows.

$$\begin{aligned} &\epsilon_0 \text{ is the identity function} \\ &\epsilon_{i+1}(F) = \epsilon_i(F) \vee \hat{\epsilon}(\epsilon_i(F)) \end{aligned}$$

where $\hat{\epsilon}$ is defined as

$$\begin{aligned} \hat{\epsilon}(q(\mathtt{i}_1, \ldots, \mathtt{i}_{ar(q)})) &\triangleq \epsilon(q)[i_1 \mapsto \mathtt{i}_1, \ldots, i_{ar(q)} \mapsto \mathtt{i}_{ar(q)}] \\ \hat{\epsilon}(F \wedge G) &\triangleq (\hat{\epsilon}(F) \wedge G) \vee (F \wedge \hat{\epsilon}(G)) \\ \hat{\epsilon}(F \vee G) &\triangleq (\hat{\epsilon}(F) \vee \hat{\epsilon}(G)) \end{aligned}$$

Due to the free variable restriction of ϵ, all free variables of $\epsilon_i(F)$ are also free in F. Since there are only finitely many formulas in $\mathcal{F}(Q, ar)$ with a fixed set of free variables (up to logical equivalence), the ascending chain $\epsilon_0(F) \models \epsilon_1(F) \models \epsilon_2(F) \ldots$ must stabilize: there is some k such that $\epsilon_k(F) \equiv \epsilon_{k+1}(F) \equiv \ldots$. Define $\overline{\epsilon}(F)$ to this stabilization point $\epsilon_k(F)$.

Observe that for any configuration $\mathcal{C}$, we have $\mathcal{C} \xrightarrow{\epsilon}{}^* \mathcal{C}'$ if and only if $\mathcal{C}'$ is a cube of the DNF of $\overline{\epsilon}(\mathcal{C})$. One may thus show (by induction on length) that for any word w, have a w-labelled path from $\mathcal{C}$ to $\mathcal{C}'$ in A iff there is a w-labelled path from $\mathcal{C}$ to $\mathcal{C}'$ in B, and thus the language accepted by B is exactly that of A. □

3 Languages of Predicate Automata

This section studies the class of languages that can be accepted by a PA.

Example 1 (from [6]). Let us illustrate how the language of runs in a parameterized program P can be defined as a predicate automaton. A parameterized program is one in which any number of threads may execute the same *thread template* in parallel. A thread template $T = \langle \mathsf{Loc}, E, \ell_{\mathsf{init}}, \ell_{\mathsf{err}}, \mathsf{src}, \mathsf{tgt} \rangle$ is a 6-tuple in which Loc is a finite set of control locations, E is a finite set of edges, $\ell_{\mathsf{init}} \in \mathsf{Loc}$ is an initial location $\ell_{\mathsf{err}} \in \mathsf{Loc}$ is an error location, and $\mathsf{src}, \mathsf{tgt} : E \to \mathsf{Loc}$ map each edges to their source and target. The idea is that a thread template is equipped with a distinguished error location ℓ_{err} (for example, marking an assertion violation) and we want to capture the language of all (syntactic) runs of the program that reach it. To use this model in verification, it suffices to show that all these runs are infeasible and as such establish that the program admits no feasible erroneous execution. In fact, the automaton will recognize the reversal of all these runs, namely, the runs that start at an error location and follow the control flow of the program back to a start state.

We define $A_P = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$, where

- $Q = \{\mathsf{loc}, \mathsf{err}\} \cup \mathsf{Loc}$, where err is a distinguished symbol which intuitively represents "some thread is at the error location", and loc is a distinguished symbol which intuitively represents "every thread is at some location."
- $ar(\mathsf{loc}) = ar(\mathsf{err}) = 0$ and for all $\ell \in \mathsf{Loc}$, $ar(\ell) = 1$
- For $\sigma \in \Sigma$, and $\ell \in \mathsf{Loc}$, we define

$$\delta(\ell(i), \langle \sigma : j \rangle) \triangleq \begin{cases} (i = j \wedge \ell'(i)) \vee (i \neq j \wedge \ell(i)) & \text{if } \mathsf{tgt}(\sigma) = \ell \\ i \neq j \wedge \ell(i) & \text{otherwise} \end{cases}$$

  where ℓ' denotes $\mathsf{src}(\sigma)$. The transition rule for loc is given by

$$\delta(\mathsf{loc}, \langle \sigma : i \rangle) = \mathsf{loc} \wedge \ell'(i)$$

  where $\sigma \in \Sigma$ and ℓ' denotes $\mathsf{src}(\sigma)$. The transition rule for err is given by

$$\delta(\mathsf{err}, \langle \sigma : i \rangle) = \begin{cases} \ell'(i) & \text{if } \mathsf{tgt}(\sigma) = \ell_{\mathsf{err}} \\ \mathsf{err} & \text{otherwise} \end{cases}$$

  where ℓ' denotes $\mathsf{src}(\sigma)$.
- $\varphi_{\mathsf{start}} = \mathsf{loc} \wedge \mathsf{err}$
- $F = \{\ell_{\mathsf{init}}, \mathsf{loc}\}$ □

The structure of A_P fairly closely mirrors the (reversed) control structure of T. The predicate loc deserves further discussion: loc ensures that for every reachable configuration $\mathcal{C}$, every $\sigma \in \Sigma$ and every $\mathtt{i} \in \mathbb{N}$, we have that if $\mathcal{C} \xrightarrow{\sigma:\mathtt{i}} \mathcal{C}'$, then $\ell'(\mathtt{i}) \in \mathcal{C}'$, where $\ell' = \mathsf{src}(\sigma)$.

Note that it is essential to recognize this language in the reverse order: we begin in state where *some* thread is at the error location, and work backwards to a state where *all* threads are in their initial location. The acceptance condition of predicate automata (*all* predicates in a configuration are accepting) can be used to encode the initial state of the program. To capture the initial condition of the program in the *initial* condition of the PA (rather than its accepting condition) would require universal quantification, which PA do not have (though note that such an extension was introduced in [7]).

In [6], we proved the following closure properties of the class of languages recognized by predicate automata, which are essential to their use in algorithmic verification:

Proposition 1 ([6]). *Predicate automata languages, denoted by* PAL*, are closed under intersection and complement.*

Remark 1. All constructions that prove the closure properties in Proposition 1 preserve the arity of the PA. Hence, languages recognized by monadic PAs are closed under all boolean operations (union, intersection, and complementation).

In this section, we explore more properties of this class of languages (PAL) an also the class of languages which are images of these languages under a homomorphism that forgets the natural number part of the alphabet symbols, and hence are just languages over Σ^* (PAFL, for predicate automata finite languages).

3.1 Data Languages of Predicate Automata

For any $\pi : \mathbb{N} \to \mathbb{N}$ and any $w \in (\Sigma \times \mathbb{N})^*$ define:

$$\pi(w) \triangleq \begin{cases} \epsilon & \text{if } w = \epsilon \\ \langle a : \pi(k) \rangle \pi(u) & \text{if } w = \langle a : k \rangle u \end{cases}$$

Proposition 2 (Symmetry). *Every language recognized by a PA A is* symmetric*, in the sense that for all $w \in \mathcal{L}(A)$, and any bijection $\pi : \mathbb{N} \to \mathbb{N}$, we have $\pi(w) \in \mathcal{L}(A)$.*

Proposition 2 provides a light tool to formally argue why certain languages, i.e. non-symmetric ones, are not recognized by any PA. In the spirit of a pumping lemma, the following proposition, provides another such tool.

Lemma 3 (Characterization). *Let $L \subseteq (\Sigma \times \mathbb{N})^*$ be a language recognized by a monadic PA. Let $u_1, u_2, \ldots$ and $v_1, v_2, \ldots$ be infinite sequences of words in $(\Sigma \times \mathbb{N})^*$ such that for each i, $u_i v_i \in L$. Then there exists $i < j$ and some permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that $\pi(u_j) v_i \in L$.*

Proof. Let $\langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F \rangle$ be a PA that recognizes L. By assumption, A recognizes $u_i v_i$ for each i. It follows that there are configurations $\mathcal{C}_{i,0}$, $\mathcal{C}_{i,1}$, and $\mathcal{C}_{i,2}$ such that

1. $\mathcal{C}_{i,0} \models \varphi_{\mathsf{init}}$
2. $\mathcal{C}_{i,0} \xrightarrow{v_i^R} \mathcal{C}_{i,1} \xrightarrow{u_i^R} \mathcal{C}_{i,2}$
3. $\mathcal{C}_{i,2}$ is accepting.

Accordingly, consider the infinite sequence of (blue) configurations induced by the original sequences of the words from the assumption:

$$\begin{array}{ccccccc} \mathcal{C}_{1,0} & \mathcal{C}_{2,0} & \cdots & \mathcal{C}_{i,0} & \cdots & \mathcal{C}_{j,0} & \cdots \\ \downarrow v_1^R & \downarrow v_2^R & & \downarrow v_i^R & & \downarrow v_j^R & \\ \mathcal{C}_{1,1} & \mathcal{C}_{2,1} & \cdots & \mathcal{C}_{i,1} & \cdots & \mathcal{C}_{j,1} & \cdots \\ \downarrow u_1^R & \downarrow u_2^R & & \downarrow u_i^R & & \downarrow u_j^R & \\ \mathcal{C}_{1,2} & \mathcal{C}_{2,2} & \cdots & \mathcal{C}_{i,2} & \cdots & \mathcal{C}_{j,2} & \cdots \end{array}$$

Since $\preceq$ is a well-quasi order on configurations, there exists $i < j$ such that $\mathcal{C}_{i,1} \preceq \mathcal{C}_{j,1}$. By downwards compatibility (Lemma 1), there exists a configuration $\mathcal{C}'_{i,2}$ and a permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that $\mathcal{C}_{i,1} \xrightarrow{\pi(u_j^R)} \mathcal{C}'_{i,2}$ and $\mathcal{C}'_{i,2} \preceq \mathcal{C}_{j,2}$, and furthermore (since $\mathcal{C}_{j,2}$ is accepting by assumption), $\mathcal{C}'_{i,2}$ is accepting. It follows that A has an accepting path $\mathcal{C}_{i,0} \xrightarrow{v_i^R} \mathcal{C}_{i,1} \xrightarrow{\pi(u_j^R)} \mathcal{C}'_{i,2}$ and thus accepts $\pi(u_j) v_i$.

Example 2. Let L be the language consisting of all data words w that can be obtained by shuffling n words $w_1, \ldots, w_n$ where each w_i is either $\langle a : j_i \rangle \langle b : j_i \rangle$ or $\langle b : j_i \rangle$ for some j, and each j_i is distinct. Using Lemma 3, we can observe that L cannot be recognized by a PA. Consider the sequences $u_1, u_2, \ldots$ and $v_1, v_2, \ldots$ where

$$u_i \triangleq \langle a : 1 \rangle \langle a : 2 \rangle \ldots \langle a : i \rangle$$
$$v_i \triangleq \langle b : 1 \rangle \langle b : 2 \rangle \ldots \langle b : i \rangle$$

Then for each i we have $u_i v_i \in L$. For a contradiction, suppose that L is recognized by PA. By Lemma 3, there is some $i < j$ and some permutation $\pi : \mathbb{N} \to \mathbb{N}$ such that $\pi(u_j)v_i \in L$. However, $\pi(u_j)v_i$ contains strictly more occurrences of a than b, which is not possible for a word in L.

Lemma 3 leads to the observation that PAL is not closed under reversal (justifying the necessity of the reversed order in Example 1):

Proposition 3. *Languages recognized by monadic predicate automata are not closed under reversal.*

Proof. Let L be the language from Example 2, which is not recognizable by a PA. We show that its reversal is accepted by a PA.

Define a PA $A = \langle \{q_0, q_a, q_b\}, ar, \Sigma, \delta, q_0, \{q_0, q_b\} \rangle$ as follows. The vocabulary consists of one nullary predicate q_0 along with two unary predicates q_a and q_b. The intuition is that $q_a(i)$ indicates that thread i has read a (and is obligated to read b), while $q_b(i)$ indicates that thread i may not read any more letters. The predicates q_0 and q_b are accepting, while q_a is not. The transition relation is

$$q_0 \xrightarrow{a:i} q_a(i) \qquad q_0 \xrightarrow{b:i} q_b(i)$$
$$q_a(i) \xrightarrow{a:j} i \neq j \wedge q_a(i) \qquad q_a(i) \xrightarrow{b:j} (i = j \wedge q_b(i)) \vee (i \neq j \wedge q_a(i))$$
$$q_b(i) \xrightarrow{a:j} i \neq j \wedge q_b(i) \qquad q_b(i) \xrightarrow{b:j} i \neq j \wedge q_b(i)$$

3.2 Finite Alphabets Languages of Predicate Automata

This section compares predicate automata with classical automata that operate on finite alphabets.

One mechanism for understanding PA languages over a finite alphabet is to restrict the alphabet $\Sigma \times \mathbb{N}$ to a finite number of threads $[n] = \{1, \ldots, n\}$. Any language over a finite alphabet like this recognized by a PA is *regular*. This can be argued easily once one observes that the PA in this case is syntactically nearly identical to an alternating finite automaton. One simply renames every ground instance of a predicate $q(\mathtt{i}_1, \ldots, \mathtt{i}_k)$ from the set of states of A as a fresh proposition $q_{(\mathtt{i}_1, \ldots, \mathtt{i}_k)}$ and the rest follows by definition.

Remark 2. For any language $L \subseteq (\Sigma \times \mathbb{N})^*$ recognized by a predicate automaton, $L \cap (\Sigma \times [n])^*$ is regular.

Besides restriction, another way to finitize the alphabet of a predicate automaton is by considering the image of PA language under a homomorphism mapping $(\Sigma \times \mathbb{N})^*$ to Σ^*. We define two such homomorphisms (by their action on the generators $(\Sigma \times \mathbb{N})$, extending to words and languages in the natural way):

- π_Σ forgets the natural number component of each letter: $\pi_\Sigma(a : \mathtt{i}) \triangleq a$
- $\pi_{\mathtt{i}}$ forgets all letters except those of $\mathtt{i}$: $\pi_{\mathtt{i}}(a : \mathtt{j}) \triangleq \begin{cases} a & \text{if } \mathtt{i} = \mathtt{j} \\ \epsilon & \text{otherwise} \end{cases}$

An immediate question jumping to mind is where PAFL fits in the Chomsky-Schützenberger hierarchy.

Proposition 4 (Inverse Images of Regular Languages). *Let $L \subseteq \Sigma^*$ be a regular language. Then the language $\{w \in (\Sigma \times \mathbb{N})^* : \pi_\Sigma(w) \in L\}$ is recognizable by a nullary predicate automaton.*

Proof. Since L is regular, its reversal is recognizable by an NFA $A = \langle Q, \Sigma, \Delta, s, F\rangle$. Construct the PA as follows. The set of predicates is exactly Q, the accepting predicates are F, and the arity of each is zero. For each $q \in Q$ and $a \in \Sigma$, we define $\delta(q, a : i) \triangleq \bigvee_{\langle q,a,q'\rangle \in \Delta} q'()$. Finally, the initial formula is $s()$.

Proposition 4 implies that PAFL includes all regular languages. The proposition below then indicates that PAFL can include strictly context-sensitive languages.

Proposition 5. *There is language L, recognized by a monadic PA, such that $\pi_\Sigma(L) = \{a^i b^j c^k : i \geq j \geq k\}$*

Proof. Define a PA $A = \langle Q, ar, \Sigma, \delta, s_c, F\rangle$ as follows. The vocabulary consists of three nullary predicates s_c, s_b, s_a along with three unary predicates q_b, q_a, and q_ϵ; the alphabet Σ is $\langle a, b, c\rangle$; the final predicates are s_a, s_c, and q_ϵ. Intuitively, s_c, s_b, and s_a track whether the automaton is currently reading a sequence of c's, b's, or a's; $q_b(i)$ indicates an obligation to read $b : i$, $q_a(i)$ indicates an obligation to read $a : i$, and $q_\epsilon(i)$ indicates an obligation to read no more letters with index i. The transition relation is defined as follows:

$$
\begin{array}{lll}
s_c \xrightarrow{c:i} (s_c \vee s_b) \wedge q_b(i) & s_b \xrightarrow{c:i} \mathit{false} & s_a \xrightarrow{c:i} \mathit{false} \\
s_c \xrightarrow{b:i} \mathit{false} & s_b \xrightarrow{b:i} (s_b \vee s_a) \wedge q_a(i) & s_a \xrightarrow{b:i} \mathit{false} \\
s_c \xrightarrow{a:i} \mathit{false} & s_b \xrightarrow{a:i} \mathit{false} & s_a \xrightarrow{a:i} s_a \wedge q_\epsilon(i)
\end{array}
$$

$$
\begin{array}{lll}
q_b(i) \xrightarrow{c:j} \mathit{false} & q_a(i) \xrightarrow{c:j} \mathit{false} & q_\epsilon \xrightarrow{c:j} \mathit{false} \\
q_b(i) \xrightarrow{b:j} i = j \vee (i \neq j \wedge q_b(i)) & q_a(i) \xrightarrow{b:j} \mathit{false} & q_\epsilon \xrightarrow{b:j} \mathit{false} \\
q_b(i) \xrightarrow{a:j} \mathit{false} & q_a(i) \xrightarrow{a:j} i = j \vee (i \neq j \wedge q_a(i) & q_\epsilon \xrightarrow{c:j} i \neq j
\end{array}
$$

Observe that

$$\mathcal{L}(A) = \{w \in (\Sigma \times \mathbb{N})^* : \pi_\Sigma(w) \in a^*b^*c^* \wedge \forall i \in \mathbb{N}.\pi_i(w) = \{\epsilon, a, ab, abc\}\}$$

and so $\pi_\Sigma(\mathcal{L}(A)) = \{a^i b^j c^k : i \geq j \geq k\}$.

Finally, let us turn our attention to the images of PA languages under $\pi_{\mathtt{i}}$ homomorphisms. It is tempting to believe that restricted to a single index, the language of a PA would turn into a regular language. The following proposition refutes that notion.

Proposition 6. *There exists a language L recognized by a monadic PA such that $\pi_1(L)$ is not context-free.*

Proof. For a contradiction, suppose that for any language L recognized by a monadic PA, $\pi_1(L)$ is regular. Clearly, the language $L_1 \triangleq \bigcup_{i\in\mathbb{N}}(d:i)^*(e:i)^*(f:i)^*$ is recognizable by a monadic PA. By Proposition 5, there is a PA-recognizable language L_2 such that $\pi_\Sigma(L_2) = \{a^i b^j c^k : i \geq j \geq k\}$. Then we have that $L_1 \times L_2$ is PA-recognizable by Lemma 4 (below). By Proposition 4, the set of all words w such that $\pi_\Sigma(w) \in (da)^*(eb)^*(fc)^*$ is recognizable by a monadic PA, and so $L_3 \triangleq \{w \in L_1 \times L_2 : \pi_\Sigma(w) \in (da)^*(eb)^*(fc)^*\}$ is recognizable by a monadic PA. By assumption, $\pi_1(L_3)$ is context-free. Let $h : \{a,b,c,d,e,f\}^* \to \{d,e,f\}^*$ be the homomorphism mapping a, b, and c to ϵ and which is the identity on d, e, and f. Since context-free languages are closed under homomorphism,

$$h(\pi_1(L_3)) = \{d^i e^j c^k : i \geq j \geq k\}$$

is context-free, a contradiction.

The proof relies on the following lemma:

Lemma 4. *Let L and L' be PA-recognizable languages. The language*

$$\begin{aligned} L_1 \times L_2 \triangleq \{&(a_1:i_1)(a'_1:i'_1)\dots(a_n:i_n)(a'_n:i'_n) : \\ &(a_1:i_1)\dots(a_n:i_n) \in L, (a'_1:i'_1)\dots(a'_n:i'_n) \in L'\} \end{aligned}$$

is PA-recognizable.

Proof. Let $A = \langle Q, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, F\rangle$ and $A' = \langle Q', ar', \Sigma, \delta', \varphi'_{\mathsf{start}}, F'\rangle$ be PA that recognize L and L' respectively. Without loss of generality, suppose Q and Q' are disjoint. Construct a PA as follows:

- For each $q \in Q \cup Q'$ there are two predicates q_0 and q_1, with the same arity as q. For any formula $F \in \mathcal{F}(Q, ar)$, we use $F|_0$ to denote the result of replacing each $q \in Q$ with q_0, and $F|_1$ to denote the result of replacing each $q \in Q$ with q_1; similarly for formulas in $\mathcal{F}(Q', ar)$. Intuitively, we construct an automaton that simulates A on the odd positions and A' on the even positions of an input word.
- The initial formula is $\varphi_{\mathsf{init}}|_0 \wedge \varphi'_{\mathsf{init}}|_0$.
- For each $q \in Q$, the transition function $\hat{\delta}$ is defined

$$\begin{aligned} \hat{\delta}(q_0(i_1,\dots,i_n), a:j) &\triangleq \delta(q(i_1,\dots,i_n), a:j)|_1 \\ \hat{\delta}(q_1(i_1,\dots,i_n), a:j) &\triangleq q_0(i_1,\dots,i_n) \end{aligned}$$

– For each $q' \in Q'$, the transition function $\hat{\delta}$ is defined

$$\hat{\delta}(q'_0(i_1, \ldots, i_n), a : j) \triangleq q'_1(i_1, \ldots, i_n)$$
$$\hat{\delta}(q'_1(i_1, \ldots, i_n), a : j) \triangleq \delta'(q'(i_1, \ldots, i_n), a : j)|_0$$

– Finally, the final predicates are $\{q_1 : q \in F\} \cup \{q'_1 : q' \in F'\}$.

4 Comparison with Register automata

Register automata are a class of automata that, like predicate automata, recognize languages of data words. A register automaton is equipped with finitely many registers, each of which may store data values (which for our purposes are natural numbers). The transition relation of a register automaton may test equality between values stored in registers and the data component of the input letter and may assign new data to registers. There is an apparent gap in the operational descriptions of predicate automata and register automata; predicate automata are declarative whereas register automata are imperative. Nonetheless, predicate automata have the same expressive power as a variant of register automata with ϵ transitions and alternation. Intuitively, one can think of a ground predicate $p(\mathtt{i}_1, \ldots, \mathtt{i}_n)$ of a predicate automaton to be analogous to a configuration of a register automaton with n registers holding data values $\mathtt{i}_1, \ldots, \mathtt{i}_n$. The fact that the only operations on registers are equality tests and assignment corresponds to the fact that PA treat thread identifiers symmetrically. For instance, a transition $q \xrightarrow{a:\mathtt{k}} q'$ of a two-register automaton that stores the value $\mathtt{k}$ to the second register on the condition that both registers store the same value corresponds to the PA transition $q(i, j) \xrightarrow{a:k} i = j \wedge q'(i, k)$.

There are several different definitions of (alternating) register automata in the literature [1, 5, 8, 12]. The model of [12] is strictly more expressive than predicate automata, because it permits starting from an initial sate that does not have to be symmetric. The one in [8] is a variation that restricts this choice symmetric states and therefore is similar to predicate automata, but is only defined for a single register.

We choose the model of [1] to illustrate the equality of the two models and extend it with ϵ-transitions (which appear in some other models in the form of "non-moving" transitions [5, 8]). This model is closer to predicate automata in that it eschews imperative register operations in favor of an abstract transition relation satisfying a *semantic equivariance* condition that enforces data symmetry (in the same way threads are treated symmetrically by predicate automata).

Definition 5 (Alternating register automaton). *An **alternating register automaton** (ARA) is an 8-tuple $A = \langle Q_\forall, Q_\exists, R, \Sigma, \Delta, \Delta_\epsilon, q_0, F\rangle$ where Σ is a finite alphabet and*

– *$Q \triangleq Q_\forall \cup Q_\exists$ is set of states, partitioned into a set of universal ($Q_\forall$) and existential ($Q_\exists$) states; $q_0 \in Q$ is an initial state and $F \subseteq Q$ is a set of final states.*

- R *is a finite set of register names.*
- $\Delta \subseteq Q \times (\mathbb{N} \cup \{\bot\})^R \times \Sigma \times \mathbb{N} \times Q \times (\mathbb{N} \cup \{\bot\})^R$ *is a transition relation.* Δ *is required to be* ***equivariant*** *in the sense that for any permutation* $\pi : \mathbb{N} \to \mathbb{N}$ *and any transition* $\langle q, v, a, i, q', v' \rangle \in \Delta$*, we have* $(q, \pi_\bot \circ v, a, \pi(i), q', \pi_\bot \circ v') \in \Delta$*, where* $\pi_\bot : \mathbb{N} \cup \{\bot\} \to \mathbb{N} \cup \{\bot\}$ *is the extension of* π *that maps* $\bot \mapsto \bot$*.*
- $\Delta_\epsilon \subseteq Q \times (\mathbb{N} \cup \{\bot\})^R \times Q \times (\mathbb{N} \cup \{\bot\})^R$ *is an* ϵ*-transition relation.* Δ_ϵ *is similarly required to be equivariant.*

A is ***non-guessing*** *if (1) for each transition* $(q, v, a, i, q', v') \in \Delta$*, for each register* r*,* $v'(r)$ *is either* i *or in the range of* v*, and (2) for each transition* $(q, v, q', v') \in \Delta_\epsilon$*, for each register* r*,* $v'(r)$ *is in the range of* v*.*

A **configuration** of an alternating register automaton is a set of states, where each state belongs to $Q \times (\mathbb{N} \cup \{\bot\})^R$. We say that a configuration C is **accepting** if for all $\langle q, v \rangle \in C$, we have $q \in F$. The **initial** configuration is $\{\langle q_0, \bot^R \rangle\}$. For configurations C, C', a letter $a \in \Sigma$, and a thread identifier $i \in \mathbb{N}$, write $C \xrightarrow{a:i} C'$ iff

- For each $\langle q, v \rangle \in C$ such that $q \in Q_\exists$, there is some $(q', v') \in C'$ such that $(q, v, a_j, i_j, q', v') \in \Delta$
- For each $\langle q, v \rangle \in C$ such that $q \in Q_\forall$, for all $q' \in Q$ and $v' \in (\mathbb{N} \cup \{\bot\})^R$ such that there is some $\langle q, v, a_j, i_j, q', v' \rangle \in \Delta$, we have $\langle q', v' \rangle \in C'$.

For configurations C and C', we write $C \xrightarrow{\epsilon} C'$ if there is some $\langle q, v \rangle \in C$ such that

- $q \in Q_\exists$ and there is some $(q, v, q', v') \in \Delta_\epsilon$ such that $C' = (C \setminus \{(q, v)\}) \cup \{(q', v')\}$
- $q \in Q_\forall$ and $C' \setminus (C \setminus \{(q, v)\}) = \{(q', v') : (q, v, q', v') \in \Delta_\epsilon\}$.

We say that an alternating register automaton accepts a word w if there is a sequence $C_0 \xrightarrow{w_1} C_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} C_n$ such that $w = w_1 \ldots w_n$, C_0 is initial, and C_n is accepting.

Proposition 7. *For any language* L *recognized by a* k*-register* ϵ*-ARA,* L^R*, the language of the words of* L *in reverse, is recognized by a PA with arity* k*.*

Proof. We prove the case with one register. Let $A = \langle Q_\forall, Q_\exists, R, \Sigma, \Delta, \Delta_\epsilon, q_0, F \rangle$ be an non-guessing 1-register ϵ-ARA. Define an ϵ-PA $\widehat{A} = \langle \widehat{Q}, ar, \Sigma, \delta, \varphi_{\mathsf{start}}, \widehat{F}, \epsilon \rangle$ that recognizes the reverse of the language recognized by A as follows.

The set of predicates $\widehat{Q}$ contains one unary predicate q and one nullary predicate $q_\bot$ for each $q \in Q$; intuitively, the predicate $q_\bot()$ corresponds the ARA state $\langle q, \bot \rangle$, and the predicate $q(i)$ corresponds to the ARA state $\langle q, i \rangle$. Define the initial formula to be $q_{0,\bot}$, and the accepting predicates $\widehat{F}$ to be $F \cup \{q_\bot : q \in F\}$. Finally, we define the transition relation. Observe that since Δ is equivariant, it is a finite union of *transition orbits* [1], of the form

$$\{(q, \pi_\bot \circ v, a, \pi(i), q', \pi_\bot \circ v') : \pi \text{ is a permutation } \mathbb{N} \to \mathbb{N}\}$$

for some $\langle q, v, a, i, q', v' \rangle$. Since A is non-guessing, each orbit can be classified as one of the following seven types $\mathcal{O}_1 - \mathcal{O}_7$, each of which can be associated with an $\mathcal{F}(\widehat{Q}, ar)$-formula $F_1 - F_7$:

Transition orbit	Corresponding formula
$\mathcal{O}_1(q,a,q') \triangleq \{(q,i,a,j,q',j) : i \neq j \in \mathbb{N}\}$	$F_1(q') \triangleq i_0 \neq i_1 \wedge q'(i_1)$
$\mathcal{O}_2(q,a,q') \triangleq \{(q,i,a,j,q',i) : i \neq j \in \mathbb{N}\}$	$F_2(q') \triangleq i_0 \neq i_1 \wedge q'(i_0)$
$\mathcal{O}_3(q,a,q') \triangleq \{(q,i,a,i,q',i) : i \in \mathbb{N}\}$	$F_3(q') \triangleq i_0 = i_1 \wedge q'(i_1)$
$\mathcal{O}_4(q,a,q') \triangleq \{(q,i,a,j,q',\bot) : i \neq j \in \mathbb{N}\}$	$F_4(q') \triangleq i_0 \neq i_1 \wedge q'_\bot()$
$\mathcal{O}_5(q,a,q') \triangleq \{(q,i,a,i,q',\bot) : i \in \mathbb{N}\}$	$F_5(q') \triangleq i_0 = i_1 \wedge q'_\bot()$
$\mathcal{O}_6(q,a,q') \triangleq \{(q,\bot,a,i,q',\bot) : i \in \mathbb{N}\}$	$F_6(q') \triangleq q'_\bot()$
$\mathcal{O}_7(q,a,q') \triangleq \{(q,\bot,a,i,q',i) : i \in \mathbb{N}\}$	$F_7(q') \triangleq q'(i)$

For any $q \in Q$ and $a \in \Sigma$, define the transition relation as follows:

$$\delta(q,a) \triangleq \begin{cases} \bigvee \{F_k(q') : q' \in Q, 1 \leq k \leq 5, \mathcal{O}_k(q,a,q') \subseteq \Delta\} & q \in Q_\exists \\ \bigwedge \{F_k(q') : q' \in Q, 1 \leq k \leq 5, \mathcal{O}_k(q,a,q') \subseteq \Delta\} & q \in Q_\forall \end{cases}$$

$$\delta(q_\bot,a) \triangleq \begin{cases} \bigvee \{F_k(q') : q' \in Q, k \in \{6,7\}, \mathcal{O}_k(q,a,q') \subseteq \Delta\} & q \in Q_\exists \\ \bigwedge \{F_k(q') : q' \in Q, k \in \{6,7\}, \mathcal{O}_k(q,a,q') \subseteq \Delta\} & q \in Q_\forall \end{cases}$$

The construction of the ϵ transition function from Δ_ϵ is similar. Finally, we may find a PA equivalent to the ϵ-PA A via the ϵ-elimination lemma (Lemma 2).

The construction for k registers is similar.

Proposition 8. *For any language L recognized by a PA with arity k, L^R is recognized by a $(k+1)$-register non-guessing ϵ-ARA.*

Proof. The essential obstacle in translating PA to ARA is that PA transitions involve arbitrary combinations of conjunctions and disjunctions, whereas each state in an ARA may have *only* conjunctive (universal) or disjunctive (existential) transitions.

Without loss of generality, suppose that for each state predicate $q \in Q$ and each letter $a \in \Sigma$, $\delta(q,a) = \bigvee_{i=1}^{N_{q,a}} C_{q,a,i}$, where $C_{q,a,i}$ is a conjunctive formula. Construct an alternating register automaton with one existential state p_q for each $q \in Q$, one universal state $p_{q,a,i}$ for each $q \in Q$, $a \in \Sigma$, and $1 \leq i \leq N_{q,a}$, one rejecting existential state *sink*, and registers $r_0, \ldots, r_k$ (where k is the maximum arity of any predicate in Q). The transition relation stores the read thread index i of the input into the register r_0 and transitions to *some* state $p_{q,a,j}$ (corresponding a cube of the formula $\delta(q,a)$):

$$\Delta \triangleq \left\{ \langle p_q, v, a, i, p_{q,a,j}, v[r_0 \leftarrow i] \rangle : q \in Q, a \in \Sigma, v \in (\mathbb{N} \cup \{\bot\})^R \right\}$$

The ϵ-transitions are responsible for transitioning from states of the form $p_{q,a,j}$ to *all* states described by $C_{q,a,i}$.

$$\Delta_\epsilon \triangleq \{(p_{q,a,i}, v, p', v') : (p', v') \in At(C_{q,a,i}, v)\}$$

where

$$At(i_j = i_k, v) = \begin{cases} \emptyset & \text{if } v(r_j) = v(r_k) \\ (\mathit{sink}, v) & \text{otherwise} \end{cases}$$

$$At(i_j \neq i_k, v) = \begin{cases} \emptyset & \text{if } v(r_j) \neq v(r_k) \\ (\mathit{sink}, v) & \text{otherwise} \end{cases}$$

$$At(q'(i_{j_1}, \ldots, i_{j_n}), v) = \{(p_{q'}, (\bot, v(i_{j_1}), \ldots, v(i_{j_n}), \bot, \ldots, \bot))\}$$

$$At(F \wedge G, v) = At(F, v) \cup At(G, v)$$

5 Verification of Parameterized Boolean Programs

Here, we discuss how one can use predicate automata as a decision procedure for verification of parameterized boolean programs. We give a construction for a *universal* language of all infeasible runs, made of commands from a finite predetermined set Σ (that of some program). Then this serves as a proof Π_B for any Boolean program P whose commands belong to Σ. Since we limit the set of commands, without loss of generality, to assume statements and assign statements, the set Σ is determined by the set of local and global variables of the program.

A boolean program is defined by a set of variables $X = X_L \uplus X_G$, partitioned into a set of local boolean variables X_L and global boolean variables X_G, a thread template $T = \langle \mathsf{Loc}, E, \ell_{\mathsf{init}}, \ell_{\mathsf{err}}, \mathsf{src}, \mathsf{tgt} \rangle$, and a function $C : E \to \mathit{Cmd}$ that maps each edge to a *command* in the language defined below

$$\begin{aligned} x, y, z &\in X \\ F, G \in \mathit{Formula} &::= x \mid \neg x \mid F \wedge G \mid F \vee G \\ c \in \mathit{Cmd} &::= \texttt{assume}(F) \\ &\mid x := F \end{aligned}$$

v_0 is a designated entry location, and v_{err} is designated error location. Note that formulas are in negation normal form. For a formula F, we use $\neg F$ to denote the negation-normal formula obtained from the negation of F by application of De Morgan's laws.

A formula in the above syntax can be thought of as a *thread state predicate*, in the sense that it can be interpreted from the perspective of any one thread. To express predicates over many threads, we extend formulas to **indexed formulas** as follows

$$\begin{aligned} F, G \in \mathit{IndexedFormula} &::= x(\texttt{i}) \mid \neg x(\texttt{i}) && x \in X_L, \texttt{i} \in \mathbb{N} \\ &\mid y \mid \neg y && y \in X_G \\ &\mid F \wedge G \mid F \vee G \end{aligned}$$

For instance $(x(0) \wedge \neg x(1)) \vee (\neg x(0) \wedge x(1))$ expresses that exactly one of threads 0 and 1 have their local variable x set to *true*. For any formula F and any natural

number $\mathtt{i}$, we use $F(\mathtt{i})$ to denote the result of replacing each local variable x appearing in F with $x(\mathtt{i})$. We can define a weakest precondition operator on indexed formulas and indexed commands as usual:

$$\begin{aligned} wp(x := F : \mathtt{j}, G) &\triangleq G[x(\mathtt{j}) \mapsto F(\mathtt{j})] \\ wp(\mathtt{assume}(F) : \mathtt{j}, G) &\triangleq G \vee \neg F(\mathtt{j}) \\ wp(wc, G) &\triangleq wp(w, wp(c, G)) \end{aligned}$$

Let Σ denote a finite subset of *Cmd*. We construct an automaton Π_B that recognizes exactly the set of *infeasible* paths over Σ (assuming each variable is initialized to *false*) as follows. The vocabulary Q consists of

- One nullary predicate *contra*, indicating a contradiction.
- Two nullary predicates q_z and $q_{\overline{z}}$ for each $z \in X_G$.
- Two unary predicates q_x and $q_{\overline{x}}$ for each $x \in X_L$.

For any variable symbol i, define a function $h_i : \mathit{Formula} \to \mathcal{F}(Q, ar)$ by

$$\begin{aligned} h_i(F \wedge G) &= h_i(F) \wedge h_i(G) \\ h_i(F \vee G) &= h_i(F) \vee h_i(G) \\ h_i(x) &= \begin{cases} q_x(i) & \text{if } x \in X_L \\ q_x() & \text{otherwise} \end{cases} \\ h_i(\neg x) &= \begin{cases} q_{\overline{x}}(i) & \text{if } x \in X_L \\ q_{\overline{x}}() & \text{otherwise} \end{cases} \end{aligned}$$

The transition relation of the automaton mimics the weakest precondition operator:

$$\begin{aligned} \delta(\mathit{contra}(), \mathtt{assume(F)} : j) &= h_j(\neg F) \vee \mathit{contra}() \\ \delta(\mathit{contra}(), x := F : j) &= \mathit{contra}() \\ \delta(q_x(i), x := F : j) &= (i \neq j \wedge q_x(i)) \vee (i = j \wedge h_i(F)) \\ \delta(q_{\overline{x}}(i), x := F : j) &= (i \neq j \wedge q_{\overline{x}}(i)) \vee (i = j \wedge h(\neg F)) \\ \delta(q_z(), z := F : j) &= h_j(F) \\ \delta(q_{\overline{z}}(), z := F : j) &= h_j(\neg F) \\ \delta(q_x(i), y := F : j) &= q_x(i) && x \neq y \\ \delta(q_{\overline{x}}(i), y := F : j) &= q_{\overline{x}}(i) && x \neq y \\ \delta(q_z(), y := F : j) &= q_z() && y \neq z \\ \delta(q_{\overline{z}}(), y := F : j) &= q_{\overline{z}}() && y \neq z \\ \delta(q_x(i), \mathtt{assume(F)} : j) &= q_x(i) \\ \delta(q_{\overline{x}}(i), \mathtt{assume(F)} : j) &= q_{\overline{x}}(i) \end{aligned}$$

The accepting predicates for Π_B are $\{q_{\overline{x}} : x \in X\}$, and the initial formula is $\mathit{contra}()$.

Theorem 2. *Let Σ be a finite alphabet of commands, and let Π_B be the predicate automaton as constructed above. For any sequence $w \in (\Sigma \times \mathbb{N})^*$, we have $w \in \mathcal{L}(\Pi_B)$ iff w is infeasible starting from a state where all variables are false.*

Proof. For any formula $F \in \underline{\mathcal{F}}(Q, ar)$, let $\underline{F}$ denote the indexed formula obtained by replacing each $q_x(\mathtt{i})$ with $x(\mathtt{i})$, each $q_{\overline{x}}(\mathtt{i})$ with $\neg x(\mathtt{i})$, each $q_y()$ with y, each $q_{\overline{y}}()$ with $\neg y$, and *contra*() with *false*.

One may show (by induction on length) that for any word w we have that $wp(w, \mathit{false}) \equiv \delta^*(\mathit{contra}(), w)$, by induction on w. Finally, observe that a formula $F \in \underline{\mathcal{F}}(Q, ar)$ is accepting exactly when $\underline{F}$ is satisfied by the state where all variables are *false*.

The construction of Π_B, as a monadic predicate automaton, yields a decision procedure for reachability of parameterized boolean programs. Example 1 gives a construction, as a monadic predicate automaton, of the language of error runs of the program. Theorem 1 and Proposition 1 imply that checking the subsumption of the set of program error runs by $\mathcal{L}(\Pi_B)$ is decidable.

6 Related work

Automata on infinite alphabets The automata theory community has developed generalizations of automata to infinite alphabets; the most relevant to our work is *(alternating) register automata (ARA)*.

Register automata were first introduced in [12]. Universality for register automata was shown to be undecidable in [14], which implies alternating register automata emptiness is undecidable in the general case. However, the emptiness problem for alternating register automata with 1 register (cf. monadic predicate automata) was proved to be decidable in [5] by reduction to reachability for lossy counter machines; and a direct proof based on well-structured transition systems was later presented in [8].

It is noteworthy that the models of register automata used in the aforementioned papers are all different, and they are different from the one from [1] that we use in Section 4. In particular, the models from [5, 1] recognize symmetric languages, like predicate automata, however, the one from [12] recognizes non-symmetric languages. The model from [5] is closed under union, intersection, and complementation, while the one from [12] is not closed under complementation.

The model from [8] is an extension of the one from [5], which is also not closed under complementation. In both models, most transitions are *non-moving*; that is, the transition does not proceed forward when it reads an input symbol, similar to a classic ϵ-transition. There is a specific type of transition that moves the head to the next position to the right.

Predicate automata have alternation built-in. To relate them to alternating register automata, we opted for the definition in [1], except we added ϵ-transitions to the model. The similarity between the notion of *equivariance* in the definition of alternating register automata in [1] and the notion of symmetry in predicate automata is extremely helpful in producing elegant reductions between the

two models. We added ϵ-transitions to the model in [1] to make the model as expressive as the one in [5] and proved equivalence of the model to predicate automata.

In [1] relations between weaker models of register automata (namely, nondeterministic and deterministic ones) and *Data Automata* and *Nominal Automata* is discussed. At the high level, it is understood that data automata are a slight generalization of nondeterministic register automata, and alternation strictly adds expressive power.

Parameterized Boolean Programs There has been a great deal of work in the area of automated verification and analysis of concurrent programs where the number of threads is unbounded but the threads are finite-state [2, 16, 13, 11]. This paper provides an alternative proof for the known result that verification of parameterized boolean programs is decidable.

References

1. Bojanczyk, M.: Slightly infinite sets, https://www.mimuw.edu.pl/ bojan/upload/main-10.pdf
2. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: CAV. pp. 403–418 (2000)
3. Brzozowski, J., Leiss, E.: On equations for regular languages, finite automata, and sequential networks. Theoretical Computer Science **10**(1), 19 – 35 (1980)
4. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**(1), 114–133 (Jan 1981)
5. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Logic **10**(3), 16:1–16:30 (Apr 2009)
6. Farzan, A., Kincaid, Z., Podelski, A.: Proof spaces for unbounded parallelism. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015. pp. 407–420. ACM (2015). https://doi.org/10.1145/2676726.2677012, https://doi.org/10.1145/2676726.2677012
7. Farzan, A., Kincaid, Z., Podelski, A.: Proving liveness of parameterized programs. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 185–196. ACM (2016). https://doi.org/10.1145/2933575.2935310, https://doi.org/10.1145/2933575.2935310
8. Figueira, D.: Alternating register automata on finite words and trees. Logical Methods in Computer Science (1) (2012)
9. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science **256**(1), 63–92 (2001)
10. Jones, C.B.: Tentative steps toward a development method for interfering programs. ACM Trans. Program. Lang. Syst. **5**(4), 596–619 (1983). https://doi.org/10.1145/69575.69577, https://doi.org/10.1145/69575.69577
11. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: CAV. pp. 645–659 (2010)

12. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (Nov 1994)
13. Namjoshi, K.S.: Symmetry and completeness in the analysis of parameterized systems. pp. 299–313. VMCAI (2007)
14. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic **5**(3), 403–435 (Jul 2004)
15. Owicki, S.S., Gries, D.: An axiomatic proof technique for parallel programs I. Acta Informatica **6**, 319–340 (1976). https://doi.org/10.1007/BF00268134, https://doi.org/10.1007/BF00268134
16. Pnueli, A., Ruah, S., Zuck, L.D.: Automatic deductive verification with invisible invariants. In: TACAS. pp. 82–97 (2001)