

Introduction and Motivation

- Automated Formal Verification (model checking) involves exploiting Turing-decidable fragments of logic to ensure verification is decidable
- Linear Temporal Logic (LTL), among other logics, sees practical use with tools Spin, TLC [1]
- LTL model checking suffers from state-space explosion, the exponential blowup in search space with model size.
- With the goal of making model checking more efficient via heuristic search, we motivate and develop the notion of *language bounds*

Relevant Definitions

<u>A Büchi automata</u> is a tuple $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ are the accepting states.
- Σ is a finite input alphabet.
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is the transition relation.
- We consider some $\sigma \in L(\mathcal{B})$ iff it visits any $q \in Q$ infinitely often.

Linear Temporal Logic (LTL) is a modal logic for reasoning about program behaviors over time

- Semantically, adds 'next', 'eventually', and 'always' to boolean logic
- We denote a Büchi Automata M satisfying a LTL property ϕ in all program behaviors as $M \models \phi$.
- Any given LTL property can be reduced to a Büchi Automata in polynomial time.

Background on Model Checking

We know by the **Chomsky Hierarchy** we cannot reason about all program behaviors in a decidable way without scaling down complexity. It is only possible in the context of regular languages to decidably reason about all program behaviors.

Because the *language* (the behavior) of a program is just a set, reasoning about all program behaviors reduces to set operations:

$L(A) \subseteq L(B) \Leftrightarrow L(A) \cap L(\overline{B}) = \emptyset$

For regular languages, language inclusion and intersection emptiness are decidable in PSPACE**complete time** [2]. The decidability of LTL model checking stems from this via the relation:



To decide $L(\mathcal{M}_A) \subseteq L(\varphi_A)$, we:

- . Reduce $L(\mathcal{M}_A) \subseteq L(\varphi_A)$ to $L(\mathcal{M}_A) \cap L(\overline{\varphi_A}) = \emptyset$ in polynomial time
- 2. Construct the asynchronous composition and decide $L(\mathcal{M}_A \mid\mid \overline{\varphi_A}) = \emptyset$, which is just a reachability problem
- 3. Exhaustively search $\mathcal{M}_A \parallel \varphi_A$ with Depth-First Search

The exponential number of states produced via composition characterizes state-space explosion.

Regular Language Bounds: Extraction and Applications

Jake Ginesin¹

¹Northeastern University

²University of Oxford

Intersection Emptiness Heuristics

To motivate intersection emptiness, consider some $\mathcal{M}_A = \mathcal{B}_1 \parallel \mathcal{B}_2 \parallel \ldots \parallel \mathcal{B}_n$. Then, notice the LTL model checking problem becomes:

$$L(\mathcal{B}_1) \cap L(\mathcal{B}_2) \cap \ldots \cap L(\mathcal{B}_n)$$

We then search $\mathcal{B}_1 \parallel \ldots \parallel \mathcal{B}_1 \parallel \overline{\varphi_A}$ via **on-the-fly composition**. That is, from the initial state $(q_0^{(\mathcal{B}_1)}, \ldots, q_0^{(\mathcal{B}_n)}, q_0^{(\overline{\varphi_A})})$ we continually construct:

$$(\text{state}) \Rightarrow \left(\,\delta(\,\text{state}\,,\,\Sigma\,)\,\right) \\ (Q^{(\mathcal{B}_1)},\ldots,Q^{(\mathcal{B}_n)},Q^{(\overline{\varphi_A})}) \Rightarrow \left(\delta^{(\mathcal{B}_1)}(Q^{(\mathcal{B}_1)},\Sigma),\ldots,\delta^{(\mathcal{B}_n)}(Q^{(\mathcal{B}_n)},\Sigma),\delta^{(\overline{\varphi_A})}(Q^{(\overline{\varphi_A})},\Sigma)\right)$$

Then, we can *iteratively* search the state-space of $\mathcal{B}_1 \parallel \ldots \parallel \mathcal{B}_1 \parallel \overline{\varphi_A}$ via a typical graph search algorithm. For Büchi automata, we choose our *termination condition* to be finding a state whose set of successors includes both an acceptance state and itself.

Heuristic Definition: But, what if we use *heuristic graph search*? After much experimentation, we found the strongest heuristic (for some $q_i \in Q^{(\mathcal{B}_i)}$) to guide us to the termination condition to be:

heuristic(node
$$\Leftrightarrow$$
 $(q_1, q_2, \dots, q_n)) = \sum_{i=1}^n q_i$'s distance

Heuristic Implementation: A Büchi automata, like any graph structure, can be viewed as a network of flows. We turn solving for this heuristic into an optimization problem over some linear program for some \mathcal{B}_i via the constraint scheme:

- Initial state $q_i \Rightarrow 1 + \sum q_i$ incoming transitions $= \sum q_i$ outgoing transitions
- Accepting state(s) $q_i \Rightarrow 1 + \sum q_i$ incoming transitions $= 2 + \sum q_i$ outgoing transitions • Other states $q_i \Rightarrow \sum q_i$ incoming transitions $= \sum q_i$ outgoing transitions
- Minimize \sum all transitions

We used Z3 (SMT), Gurobi (LP), and some pre-processing tricks (read the paper) to efficiently implement the above scheme.

Heuristic Performance Analysis:



Figure 1. Heuristic performance over 10 randomly generated automata of different sizes. Averaged over 10 runs per automata size.

We find while bug (counterexample) finding is pretty fast with heuristics, exhaustive verification becomes much more expensive.

Christoph Haase²

 $L(\mathcal{B}_n) \cap L(\overline{\varphi_A}) = \emptyset$

e from acceptance condition in \mathcal{B}_i

Motivated by intersection emptiness heuristics results, we want to try to decide beforehand whether we need to do an exhaustive search at all. To do so, we define *language bounds*:

- given word in a language

We can use these definitions to reason about program behaviors:

- in the language bound intervals of B
- to be *disjoint* from the language bound intervals of B

Parikh Image: for a given word w, a Parikh image is a vector denoting the number of characters in said word. i.e. $abc \rightarrow [a:1,b:1,c:1]$

Finding the **minimum language bounds** for regular languages is a solved problem [3] - integer programming can extract the *minimal Parikh image* (and thus the lower language bound) from a language. We use the previously described constraint scheme to extract the parikh image, and we can set our objective to minimize over all transitions or a single character.

The maximum language bounds for regular languages is not as straight forward since regular languages can accept repeating words. We develop a procedure:

- all possible Parikh images
- emptiness via traditional numerical methods



Future Work & Opportunities

Automated verification is a rich problem space! Some ideas for future work:

- Real-world intersection emptiness (model checking) benchmarks

- Using linear programming and SMT to optimize other decidable logics (e.g., TLA+)
- 10.1109/32.588521.
- dence, RI, USA, 1977, pp. 254-266, doi: 10.1109/SFCS.1977.16.
- [3] Anthony Widjaja To. (2010). Parikh Images of Regular Languages: Complexity and Applications.

This project received funding from the University of Oxford and the European Research Council under the initiative "Advanced Reasoning in Arithmetic Theories." The views, findings, and conclusions presented herein are solely those of the authors and may not represent the opinions of the European Research Council.

Northeastern University

Language Bounds

• Word Length Language Bounds: The shortest and longest length words in a language • Character Number Language Bounds: The lowest and highest number of characters in any

• Language inclusion $(L(A) \subseteq L(B))$ requires all language bound intervals of A to be contained

• Similarly, intersection emptiness $(L(A) \cap L(B) = \emptyset)$ requires all language bound intervals of A

Built on top of depth-first search, iteratively assign each node a set of polynomials describing

• We consider the set of accepting polynomials and test polynomial inclusion and intersection

 $a(b)^*c \Rightarrow a \rightarrow c \rightarrow b \Rightarrow 1x_a + n_1x_b + 1x_c$

Figure 2. Polynomial Derivation Process (Regex), not Büchi

Polynomial-time reduction between language inclusion and intersection emptiness Heuristics for undecidable verification schemes (e.g., theorem proving in Lean)

References

[1] G. J. Holzmann, "The model checker SPIN," in IEEE Transactions on Software Engineering, vol. 23, no. 5, pp. 279-295, May 1997, doi:

[2] D. Kozen, "Lower bounds for natural proof systems," 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), Provi-