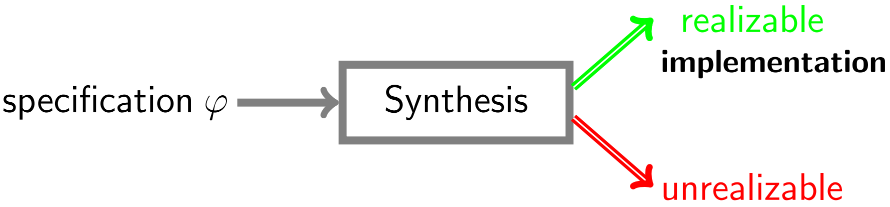
Synthesizing Approximate Implementations for Unrealizable Specifications

Rayna Dimitrova   Bernd Finkbeiner   **Hazem Torfah**

March 8, 2021
Workshop on Synthesis of Models and Systems
Simons Institute Berkeley

Synthesis of Reactive Systems



specification φ

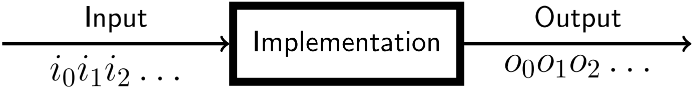
Synthesis

realizable

implementation

unrealizable

For φ a linear-time property, the implementation satisfies φ **for every possible infinite sequence of input values.**



Input

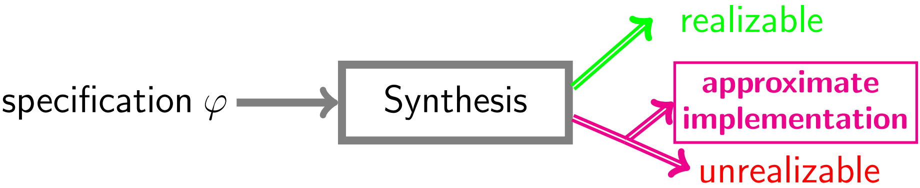
$i_0 i_1 i_2 \ldots$

Implementation

Output

$o_0 o_1 o_2 \ldots$

Synthesis of Approximate Implementations



specification φ

Synthesis

realizable

approximate implementation

unrealizable

Approximate implementation: guaranteed to satisfy the specification for

1. sequences of input values that belong to a given class
2. at least a specified portion of the sequences in the given class

Example

Synthesise an arbiter serving two client processes

Input: r_1, r_2 (requests), Output: g_1, g_2 (grants)

Specification:

The two grants are never given simultaneously: $\Box\neg(g_1 \wedge g_2)$

Every request is eventually followed by a grant: $\Box\bigwedge_{i=1}^{2}(r_i \rightarrow \Diamond g_i)$

A grant should not be revoked while the client keeps requesting it, provided that the client will eventually release the grant:

$$\Box\bigwedge_{i=1}^{2}(g_i \wedge r_i \wedge (\Diamond\neg r_i) \rightarrow \bigcirc g_i)$$

Example

Synthesise an arbiter serving two client processes

Input: r_1, r_2 (requests), Output: g_1, g_2 (grants)

Specification:

The two grants are never given simultaneously: $\Box \neg (g_1 \wedge g_2)$

Every request is eventually followed by a grant: $\Box \bigwedge_{i=1}^{2} (r_i \rightarrow \Diamond g_i)$

A grant should not be revoked while the client keeps requesting it, provided that the client will eventually release the grant:

$$\Box \bigwedge_{i=1}^{2} (g_i \wedge r_i \wedge (\Diamond \neg r_i) \rightarrow \bigcirc g_i)$$

Unrealizable!

After gaining a grant, the client might never lower the request.

Unrealizable Specifications: Environment Assumptions

In reality, environments are often subject to restrictions.

Arbiter example: Each client either releases the grant within some fixed bounded number of steps, or keeps it forever.

$\Longrightarrow$ The example specification becomes realizable.

Unrealizable Specifications: Environment Assumptions

How are environment assumptions specified?

- Logical formulas or automata describing allowed sequences of inputs [Chatterjee et al.'08, Li et al.'11, Alur et al.'13]
- Structural assumptions, such as bounded size of environment process [Kupferman et al.'11]

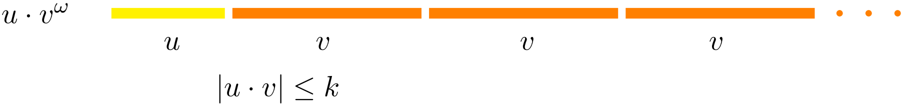
In this work:

bound associated with the input sequences produced by the environment

Environments with Bounded-Lassos

Inputs generated by the environment:

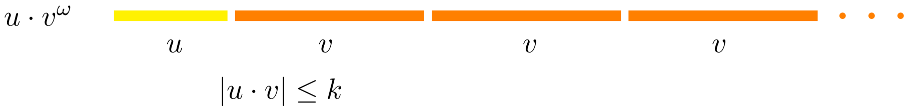
ultimately periodic, defined by lassos of length at most k



Environments with Bounded-Lassos

Inputs generated by the environment:

ultimately periodic, defined by lassos of length at most k



$u \cdot v^\omega$

u v v v

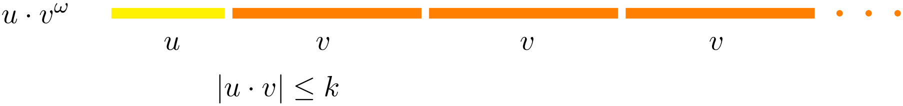
$|u \cdot v| \leq k$

k-lasso-precise implementation
satisfies φ for input sequences representable as lassos of length at most k

Environments with Bounded-Lassos

Inputs generated by the environment:

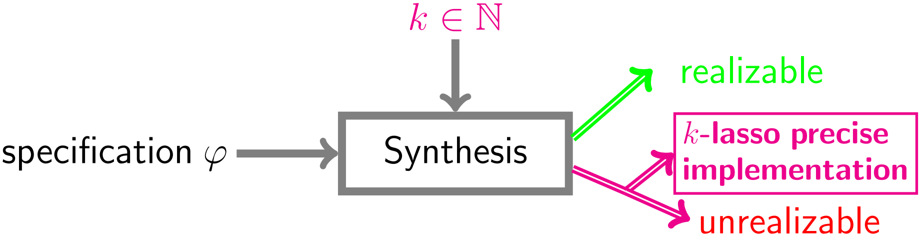
ultimately periodic, defined by lassos of length at most k



k-lasso-precise implementation
satisfies φ for input sequences representable as lassos of length at most k

ϵ-k-approximate implementation
satisfies φ on at least a $(1-\epsilon)$ fraction of the input lassos of length k

Synthesis of k-lasso precise implementations



k-lasso-precise implementation

satisfies φ for input sequences representable as lassos of length at most k

Our Contributions

1. Automata-based synthesis of k-lasso precise implementations
2. Symbolic bounded synthesis of k-lasso precise implementations
3. Synthesis algorithms for ϵ-k-approximate implementations

Outline

1. Automata-based synthesis of k-lasso precise implementations
2. Symbolic bounded synthesis of k-lasso precise implementations
3. Synthesis algorithms for ϵ-k-approximate implementations

Lasso-precise Implementations

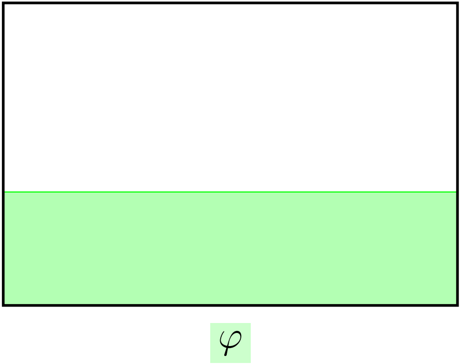
A k-lasso precise implementation

- satisfies φ on inputs representable as lassos of length at most k,
- can behave arbitrarily (violate φ) on all other input sequences.

Lasso-precise Implementations

A k-lasso precise implementation

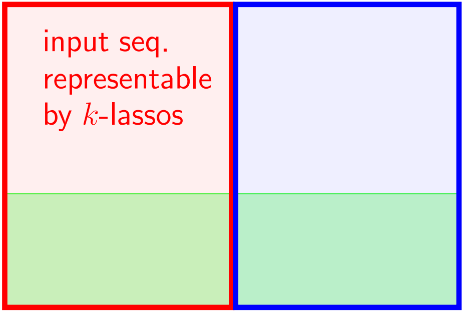
- satisfies φ on inputs representable as lassos of length at most k,
- can behave arbitrarily (violate φ) on all other input sequences.



Lasso-precise Implementations

A k-lasso precise implementation

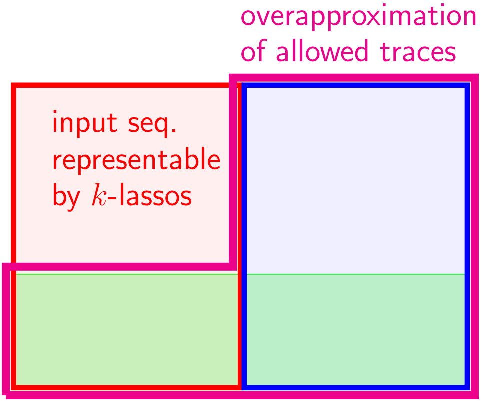
- satisfies φ on inputs representable as lassos of length at most k,
- can behave arbitrarily (violate φ) on all other input sequences.



Lasso-precise Implementations

A k-lasso precise implementation

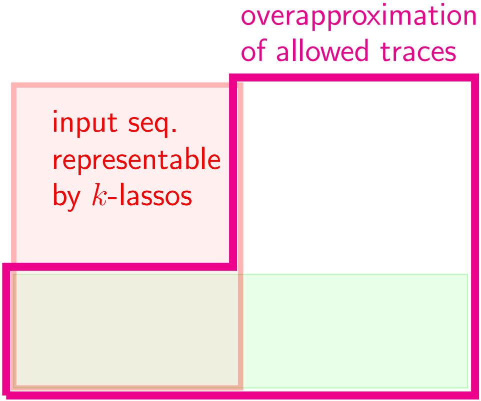
- satisfies φ on inputs representable as lassos of length at most k,
- can behave arbitrarily (violate φ) on all other input sequences.



Lasso-precise Implementations

A k-lasso precise implementation

- satisfies φ on inputs representable as lassos of length at most k,
- can behave arbitrarily (violate φ) on all other input sequences.

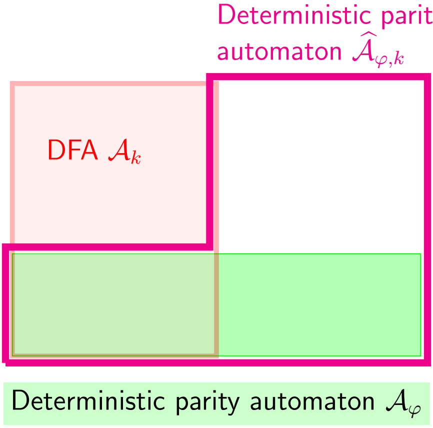


Automata-based Synthesis of Lasso-precise Implementations

Given: deterministic parity automaton $\mathcal{A}_\varphi$ accepting the words satisfying φ

1. Construct a DFA $\mathcal{A}_k$ that accepts the prefixes of words representable by k-lassos

2. Construct a deterministic parity automaton $\widehat{\mathcal{A}}_{\varphi,k}$ accepting the words that
   - are accepted by $\mathcal{A}_\varphi$ or
   - have a prefix rejected by $\mathcal{A}_k$

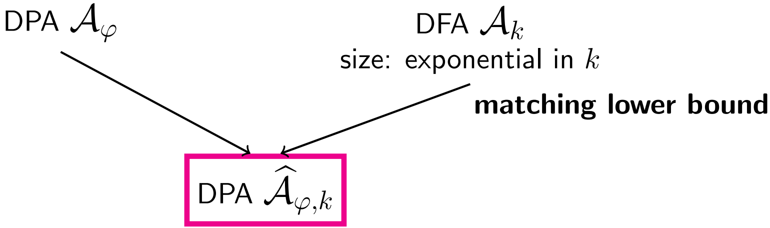


Deterministic parity automaton $\widehat{\mathcal{A}}_{\varphi,k}$

DFA $\mathcal{A}_k$

Deterministic parity automaton $\mathcal{A}_\varphi$

Automata-based Synthesis of Lasso-precise Implementations



DPA $\mathcal{A}_\varphi$

DFA $\mathcal{A}_k$

size: exponential in k

matching lower bound

DPA $\widehat{\mathcal{A}}_{\varphi,k}$

size: polynomial in $|\mathcal{A}_\varphi|$ and exponential in k

$\mathcal{L}(tree(\widehat{\mathcal{A}}_{\varphi,k})) \neq \emptyset \iff \exists$ k-lasso precise implementation for φ

Theorem

For a specification given as a deterministic parity automaton $\mathcal{A}_\varphi$, the synthesis problem for k-lasso precise implementations can be solved in time polynomial in the size of the automaton $\mathcal{A}_\varphi$ and exponential in k.

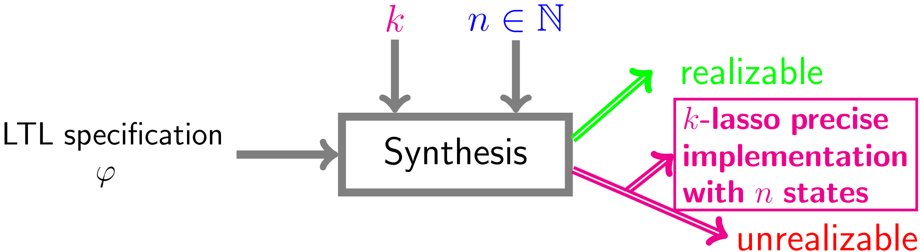
Outline

1. Automata-based synthesis of k-lasso precise implementations
2. Symbolic bounded synthesis of k-lasso precise implementations
3. Synthesis algorithms for ϵ-k-approximate implementations

Outline

1. Automata-based synthesis of k-lasso precise implementations
2. Symbolic bounded synthesis of k-lasso precise implementations
3. Synthesis algorithms for ϵ-k-approximate implementations

Bounded Synthesis of Lasso-precise Implementations



SAT encoding: using $\mathcal{A}_k$, encode as in [Finkbeiner, and Schewe'13]
size of constraint system: exponential in $|\varphi|$ and in k

QBF encoding: we propose a direct encoding as $\exists\forall$ QBF problem
size of constraint system: polynomial in $|\varphi|$ and in k

Symbolic bounded synthesis of k-lasso precise implementations

Idea of the QBF-encoding:

$\exists$ implementation of size n, encoded as

- transitions
- output labelling

$\forall$ input lasso of length k, encoded as

- k input values
- loop start position

the corresponding sequence of inputs and outputs satisfies φ

Theorem

For a specification given as an LTL formula φ, there exists a QBF formula of size $O(|\varphi| + n^2 + k^2)$ that is satisfiable if and only if there exists a k-lasso precise implementation of size n for the property φ.

Outline

1. Automata-based synthesis of k-lasso precise implementations
2. Symbolic bounded synthesis of k-lasso precise implementations
3. Synthesis algorithms for ϵ-k-approximate implementations

Outline

1. Automata-based synthesis of k-lasso precise implementations
2. Symbolic bounded synthesis of k-lasso precise implementations
3. Synthesis algorithms for ϵ-k-approximate implementations

ϵ, k-approximate Implementations: Example

Simplified arbiter with

Input: r (request), p (permission); Output: g (grant)

Specification:

Every request is eventually followed by a grant: $\square(r \to \diamond g)$

Grants are not to be given when there is no permission:

$$\square(\neg p \to \bigcirc \neg g)$$

ϵ, k-approximate Implementations: Example

Simplified arbiter with

Input: r (request), p (permission); Output: g (grant)

Specification:

Every request is eventually followed by a grant: $\Box(r \rightarrow \Diamond g)$

Grants are not to be given when there is no permission:

$$\Box(\neg p \rightarrow \bigcirc \neg g)$$

Unrealizable!

ϵ, k-approximate Implementations: Example

Simplified arbiter with

Input: r (request), p (permission); Output: g (grant)

Specification:

Every request is eventually followed by a grant: $\Box(r \rightarrow \Diamond g)$

Grants are not to be given when there is no permission:

$$\Box(\neg p \rightarrow \bigcirc \neg g)$$

Unrealizable for bound on the input lassos equal to 1!

ϵ, k-approximate Implementations: Example

Simplified arbiter with

Input: r (request), p (permission); Output: g (grant)

Specification:

Every request is eventually followed by a grant: $\square(r \rightarrow \diamondsuit g)$

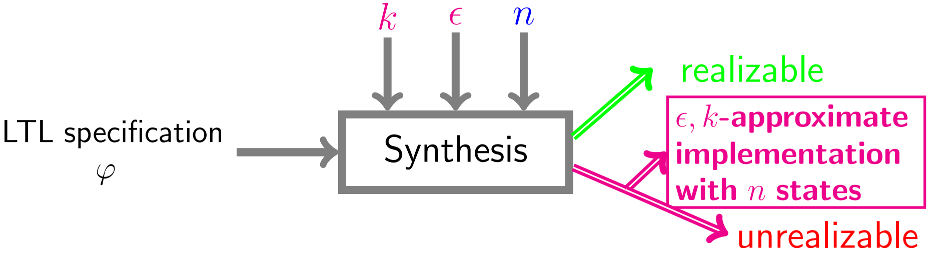
Grants are not to be given when there is no permission:

$$\square(\neg p \rightarrow \bigcirc \neg g)$$

Unrealizable for bound on the input lassos equal to 1!

Ratio of lassos of given length for which the spec. can be enforced?

Bounded Synthesis of ϵ, k-approximate Implementations

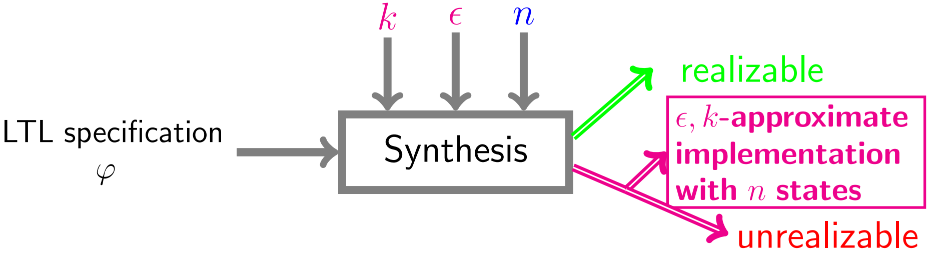


Goal: synthesize an implementation with n states such that

$$\frac{\#k\text{-lasso-representable input sequences which satisfy } \varphi}{\#k\text{-lasso-representable input sequences}} \geq 1 - \epsilon$$

Approach: compute an implementation that maximizes this ratio

Bounded Synthesis of ϵ, k-approximate Implementations



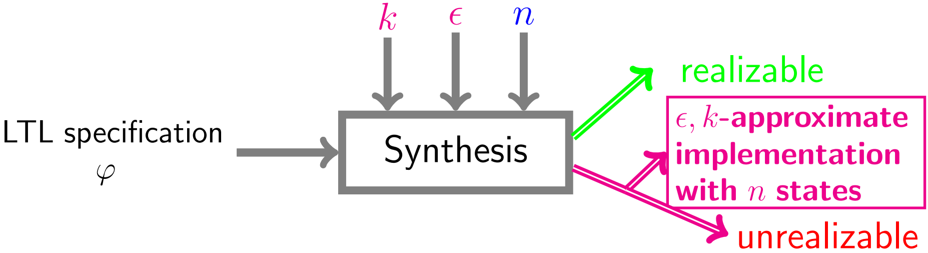
Encoding the problem as a **maximum model counting** instance

maximize over the implementations of size n

maximizing the number of input lassos of length k

for which the corresponding sequence of inputs and outputs satisfies φ

Bounded Synthesis of ϵ, k-approximate Implementations


k
ϵ
n
LTL specification φ
Synthesis
realizable
ϵ, k-approximate implementation with n states
unrealizable

Theorem

For a specification given as an LTL formula φ, there exists a maximum model counting instance of size $O(|\varphi| + n^2 + k^2)$ that encodes the synthesis of an implementation with n states that satisfies φ for a maximum number of input sequences representable as lassos of size k.

Experimental Evaluation

Proof-of-concept implementation of the bounded synthesis methods:

- using the CAQE QBF solver [1] for k-lasso precise implementations
- using the MaxCount tool [2] for ϵ-optimal implementations

For small bounds on lasso and system size (up to 4) we can synthesize

- k-lasso precise implementations for unrealizable specifications
- ϵ-optimal implementations for input lassos of length k

Instances quickly become large

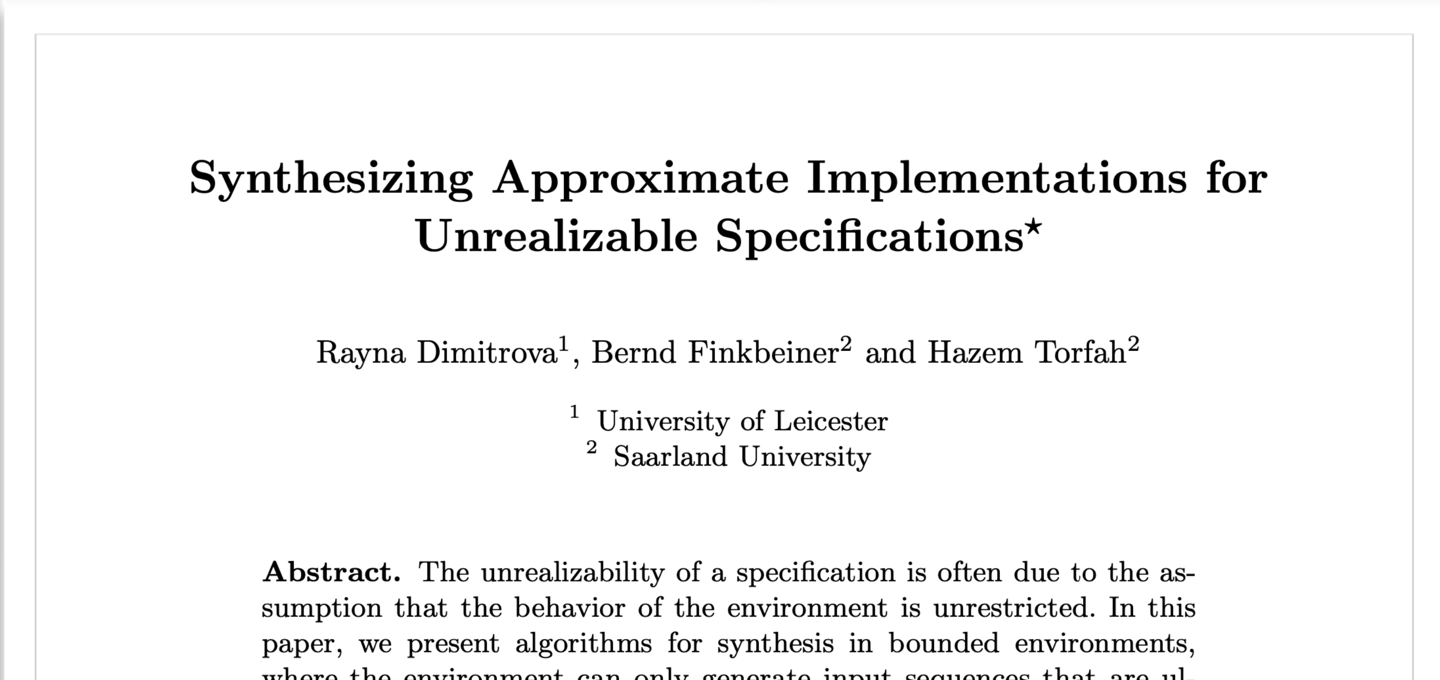
a lot of room for optimization and experimentation

[1] Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. (FMCAD'15)

[2] Daniel J. Fremont et al. Maximum model counting (AAAI'17)

Summary

- Introduced notion of k-lasso precise implementations
- Provided two methods for synthesizing lasso-precise approximations
  - explicit automata-based approach
  - symbolic QBF-based approach
- Introduced notion of ϵ, k-approximate implementations
- Provided a maximum model counting-based method for synthesis of approximate implementations that are ϵ-optimal for k-lassos

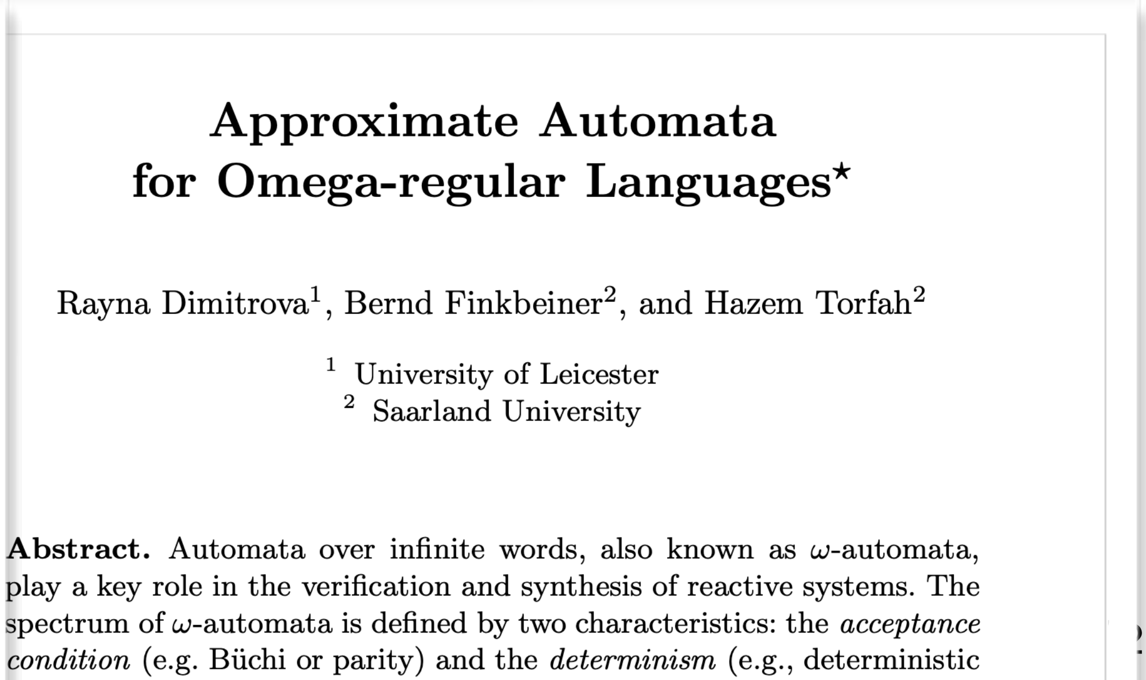


Synthesizing Approximate Implementations for Unrealizable Specifications*

Rayna Dimitrova[1], Bernd Finkbeiner[2] and Hazem Torfah[2]

[1] University of Leicester
[2] Saarland University

Abstract. The unrealizability of a specification is often due to the assumption that the behavior of the environment is unrestricted. In this paper, we present algorithms for synthesis in bounded environments, where the environment can only generate input sequences that are ul-



Approximate Automata for Omega-regular Languages*

Rayna Dimitrova[1], Bernd Finkbeiner[2], and Hazem Torfah[2]

[1] University of Leicester
[2] Saarland University

Abstract. Automata over infinite words, also known as ω-automata, play a key role in the verification and synthesis of reactive systems. The spectrum of ω-automata is defined by two characteristics: the *acceptance condition* (e.g. Büchi or parity) and the *determinism* (e.g., deterministic