Towards Efficient Synthesis of LTL Specifications

Emmanuel Filiot
joint with Naiyong Jin and Jean-François Raskin

Université Libre de Bruxelles

FNRS contact day
Reactive Systems

- continuous interaction with their environment
- non-terminating
- have to respect real-time properties (e.g., safety properties)
- have to cope with the uncontrollable behavior of their environment
Reactive Systems

- continuous interaction with their environment
- non-terminating
- have to respect real-time properties (e.g. safety properties)
- have to cope with the uncontrollable behavior of their environment

Hard to design, needs *synthesis* from specification!
Example

Environment

Process 1

req1

Process 2

ackl

Resource Access Controller

req2

ack2

Reactive System
Example

Environment

Process 1

Process 2

Reactive System

Resource Access Controller

-executions:- infinite sequences of sets of signals

\{req1, req2\} \{ack1\} \{req1\} \{ack2\} \{req1\} \{ack1\} \ldots
Properties we would like to ensure

Liveness property: \( G (req_i \rightarrow F ack_i) \quad i=1,2 \)

Safety property: \( G (\neg ack_1 \lor \neg ack_2) \)
LTL Synthesis

Liveness property: \( G ( \text{reqi} \rightarrow F \text{acki} ) \) \( i=1,2 \)

Safety property: \( G (\neg \text{ack1} \lor \neg \text{ack2}) \)

generate a RS that realizes the spec
LTL Synthesis

Liveness property: \( G \left( req_i \rightarrow F ack_i \right) \)  \( i=1,2 \)

Safety property: \( G \left( \neg ack_1 \lor \neg ack_2 \right) \)

Realizability
Given an LTL spec, does there exists a RS such that all its executions (whatever the environment does) satisfy the spec?
Unrealizable Spec

G (ack₁ -> F req₁)

“Each time the system acknowledge, the environment eventually sends a request”
Synthesis as Game

Diagram showing a tree structure with nodes labeled "req1", "req2", "ack1", and "ack2".
All the infinite paths have to satisfy the spec
Existing Procedures

- if a spec is realizable, it is realizable by a finite-state strategy
- 2ExpTime-Complete [Rosner, 92]
- “classical” procedure [Pnueli, Rosner, 89]

\[ \text{LTL} \quad \rightarrow \quad \text{Rabin Game} \]
Existing Procedures

• if a spec is realizable, it is realizable by a finite-state strategy
• 2ExpTime-Complete [Rosner, 92]
• “classical” procedure [Pnueli, Rosner, 89]

Needs Safra’s Determinization!
Existing Procedures

- if a spec is realizable, it is realizable by a finite-state strategy
- 2ExpTime-Complete [Rosner, 92]
- “classical” procedure [Pnueli, Rosner, 89]

Needs Safra’s Determinization!

- Safranless procedure [Kupferman, Vardi, 05]

LTL $\rightarrow$ Büchi Game
Existing Procedures

- if a spec is realizable, it is realizable by a finite-state strategy
- 2ExpTime-Complete \[\text{[Rosner, 92]}\]
- "classical" procedure \[\text{[Pnueli, Rosner, 89]}\]

Needs Safra's Determinization!

- Safraless procedure \[\text{[Kupferman, Vardi, 05]}\]

Implemented in Lily \[\text{[Jobstmann, Bloem]}\]
Existing Procedures

- if a spec is realizable, it is realizable by a finite-state strategy
- 2ExpTime-Complete [Rosner, 92]
- “classical” procedure [Pnueli, Rosner, 89]

Needs Safra’s Determinization!

- Safraleless procedure [Kupferman, Vardi, 05]

Implemented in Lily [Jobstmann, Bloem]

- in this talk: LTL → Safety Game
An (infinite) word is accepted iff all its runs visits at most $K$ accepting states

In this example: words with at most $K$ symbols $b$
Automata as a spec

• K-co-Büchi automata specify infinite words
• they can be used as RS specifications

**Theorem**

For any LTL specification $\Phi$, one can construct a $K$-co-Büchi automaton $A$ such that:

$\Phi$ is realizable
iff
$A$ is realizable

Construction: exptime and $K = O(2^{|\Phi|})$
Determinization

- $K$-Co-Büchi automata are easily determinizable
- extend subset construction with counters (up to $K+1$)
- states: Functions $F: Q \rightarrow \{0, 1, \ldots, K+1\}$
Playing on automata

System

Environment

Safety Winning Condition:
Avoid functions with some counter $K+1$
Controllable Predecessors

- $P \subseteq F$: subset of system positions
- Safe controllable predecessors of $P$
  
  $Pre(P) = \{ F \mid \exists o \subseteq O, \forall F', ((F,o),F') \in T \Rightarrow F' \in P\}$

- Greatest fixpoint $Pre^* = \text{winning region for System}$
Controllable Predecessors

1. partial order on counting functions:

\[ F \leq_d F' \text{ if } \forall q: F(q) \leq F'(q) \]

2. if System wins from \( F' \), she also wins from \( F \).

3. \( \text{Pre(.)} \) preserves downward-closed sets

4. represent each (downward) set of the fixpoint computation by its maximal elements
Symbolic Fixpoint Computation

$F$ (counting functions)
Symbolic Fixpoint Computation

\( F \) (counting functions)

Pre(\( F \))
Symbolic Fixpoint Computation

F (counting functions)
Pre(F)
Pre(Pre(F))
Symbolic Fixpoint Computation

\[ F \]
\[ \text{Pre}(F) \]
\[ \text{Pre}(\text{Pre}(F)) \]
\[ \text{Pre}^* \]

\( F \) (counting functions)
Symbolic Fixpoint Computation

\[ F \]
\[ \text{Pre}(F) \]
\[ \text{Pre}(\text{Pre}(F)) \]
\[ \text{Pre}^* \]

(counting functions)
Incremental Algorithm

- the bound $K$ is very big (doubly exponential)
- if the spec is realizable with a “small” bound, it is realizable with a “big” bound
- iterate over $k=0, 1, \ldots, K$
the bound $K$ is very big (doubly exponential)

- if the spec is realizable with a "small" bound, it is realizable with a "big" bound

- iterate over $k=0,1,...,K$

**Not reasonable for unrealizable specifications**
Incremental Algorithm

- the bound $K$ is very big (doubly exponential)
- if the spec is realizable with a "small" bound, it is realizable with a "big" bound
- iterate over $k=0,1,...,K$

But by Martin’s determination theorem:

$\phi$ is unrealizable for the System iff $\neg \phi$ is realizable for the Environment.
Experiments

- implementation in Perl (as Lily)
- if the spec is realizable, output a Moore machine that realizes it
- formula to automata construction borrowed from Lily (based on Wring [Somenzi, Bloem])
- **significantly faster** on all realizable Lily’s examples
- **bottleneck**: formula to automaton construction
Example

assume Sto_REQ=0;
assume G(Sto_REQ=1 * BtoS_ACK=0) -> X(Sto_REQ=1);
assume G((Sto_REQ=1 * BtoS_ACK=1) -> X(Sto_REQ=1));
assume G(BtoS_ACK=0) -> X(Sto_REQ=0);
assume G(BtoS_ACK=1) -> X(Sto_REQ=1);

(BtoS_ACK=0)
\( G( (Sto_REQ=0 * X(Sto_REQ=1)) \rightarrow X(BtoS_ACK=0) * X(F(BtoS_ACK=1))) \) *
\( G( (BtoS_ACK=0 * X(Sto_REQ=0)) \rightarrow X(BtoS_ACK=0) ) \) *
\( G(BtoS_ACK=0 * BtoS_ACK=1) \);

(BtoS_ACK=1)
\( G( (Sto_REQ=1 * X(Sto_REQ=1)) \rightarrow X(BtoS_ACK=1) * X(F(BtoS_ACK=1))) \) *
\( G( (BtoS_ACK=1 * X(Sto_REQ=0)) \rightarrow X(BtoS_ACK=1) ) \) *
\( G(BtoS_ACK=0 * BtoS_ACK=1) \);

assume RtoB_ACK=0;
assume RtoB_ACK=1;
assume G(RtoB_REQ=0) -> X(RtoB_ACK=0);
assume G(RtoB_REQ=1) -> X(RtoB_ACK=1);
assume G(RtoB_ACK=0 * RtoB_ACK=1) -> X(RtoB_ACK=0);
assume G(RtoB_ACK=1 * RtoB_ACK=0) -> X(RtoB_ACK=1);
assume G(RtoB_REQ=0 -> X(RtoB_ACK=0));
assume G(RtoB_REQ=1 -> X(RtoB_ACK=1));

(BtoR_REQ=0)
\( G(RtoB_ACK=0 \rightarrow X(BtoR_REQ=0)) \) *
\( G((BtoR_REQ=1 * RtoB_ACK=0) \rightarrow X(BtoR_REQ=1)) \) *
\( G(F(BtoR_REQ=0)) \) *
\( G((BtoR_REQ=0) + (BtoR_REQ=1)) \);

(BtoR_REQ=1)
\( G(RtoB_ACK=1 \rightarrow X(BtoR_REQ=1)) \) *
\( G((BtoR_REQ=1 * RtoB_ACK=0) \rightarrow X(BtoR_REQ=1)) \) *
\( G(F(BtoR_REQ=0)) \) *
\( G((BtoR_REQ=0) + (BtoR_REQ=1)) \);
Future Work ...

- compositionnality
- avoid automata construction to handle larger formulas
Future Work ...

• compositionnality

• avoid automata construction to handle larger formulas

... Thank You