CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

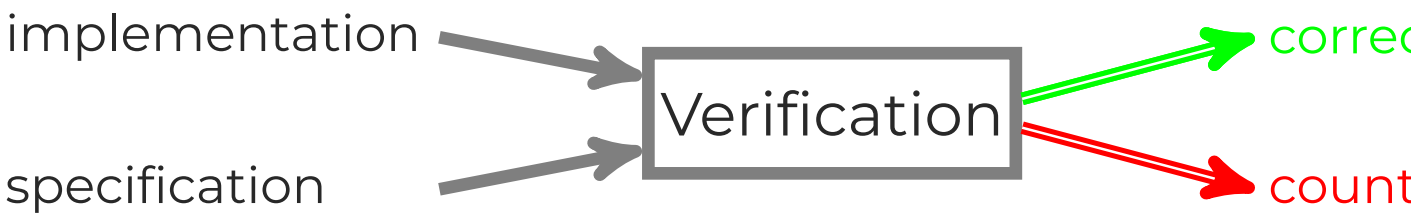
Synthesis of Infinite-State Reactive Systems (Part 1)

Rayna Dimitrova | VTSA 2025 | 03.09.2025

Verification versus Synthesis

When verification fails, need to fix bugs!

Verification versus Synthesis



When verification fails, need to fix bugs!

Verification versus Synthesis

When verification fails, need to fix bugs!

Main benefits of synthesis

- Automatic construction of correct system
- Specification debugging in early design stages

Synthesis from Specifications

Realizability

Does there exist an implementation satisfying the specification?

Synthesis

Construct an implementation satisfying the specification (if one exists).

Two Main Schools of Synthesis

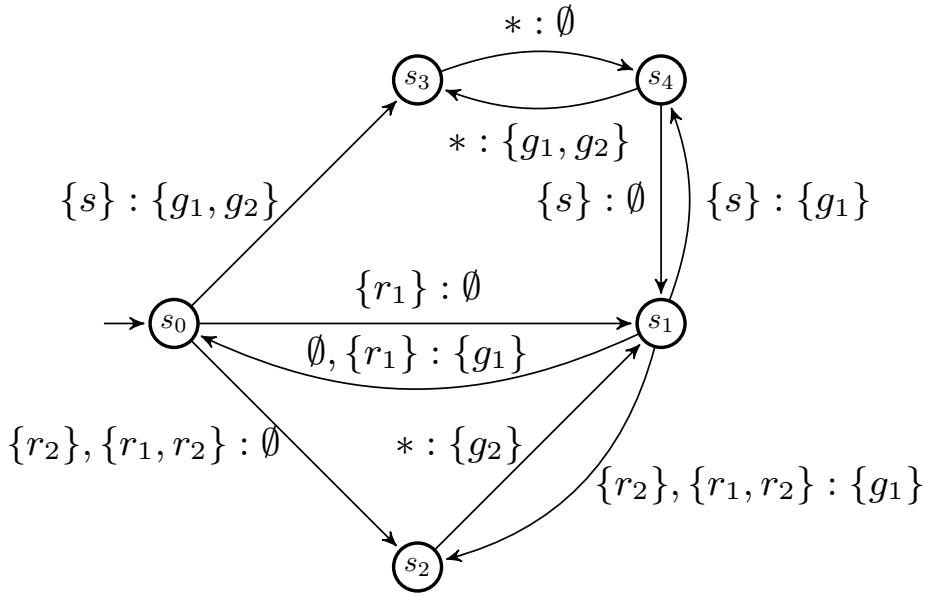
Program Synthesis

```
int max2sumprod(int a, int b, int c) {
    int op1;
    int op2;

    if (c + a > a + b) {
        op1 = b
        op2 = c;
    } else if(a + c > a + b) {
        op1 = a;
        op2 = c;
    } else {
        op1 = a;
        op2 = b;
    }
    return op1 * op2;
}
```

transformational/functional programs

Finite-State Reactive Synthesis



ongoing/reactive systems

Classical Program Synthesis

Classical Program Synthesis

Deductive synthesis in the **1960's** and **1970's**

Extract programs from proofs of satisfiability of specification

- **[Green, IJCAI 1969].**
  Application of Theorem Proving to Problem Solving
- **[Manna/Waldinger, IEEE Transactions on Software Engineering, 1979]**
  Synthesis: Dreams → Programs

Classical Program Synthesis

Deductive synthesis in the **1960's** and **1970's**

Extract programs from proofs of satisfiability of specification

- **[Green, IJCAI 1969].**
  Application of Theorem Proving to Problem Solving
- **[Manna/Waldinger, IEEE Transactions on Software Engineering, 1979]**
  Synthesis: Dreams → Programs

●●●

Classical Program Synthesis

Deductive synthesis in the **1960's** and **1970's**

Extract programs from proofs of satisfiability of specification

- **[Green, IJCAI 1969].**
  Application of Theorem Proving to Problem Solving
- **[Manna/Waldinger, IEEE Transactions on Software Engineering, 1979]**
  Synthesis: Dreams → Programs

•••

Modern program synthesis

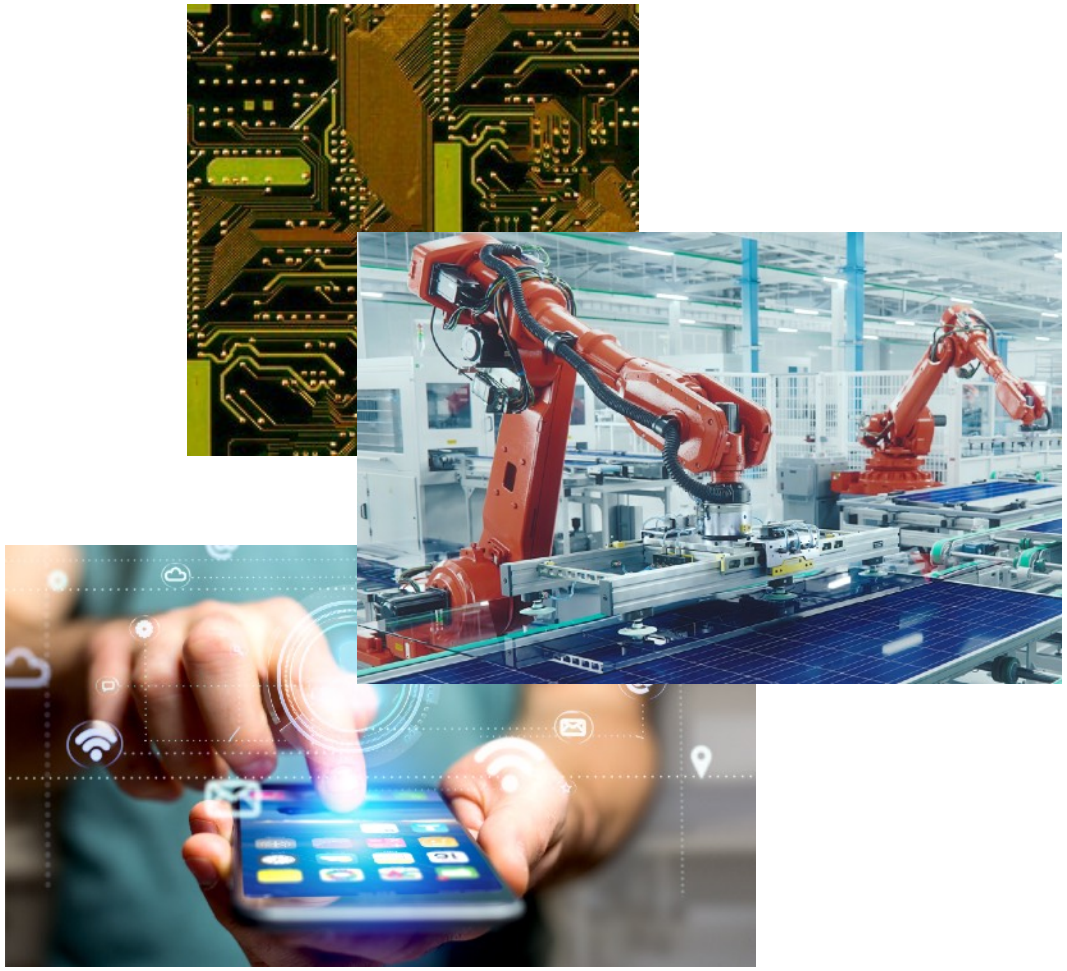
- **[Solar-Lezama/Tancau/Bodik/Seshia/Saraswat, ASPLOS 2006]**
  Combinatorial Sketching for Finite Programs
- **[Gulwani, POPL 2011]**
  Automatic String Processing in Spreadsheets using Input-Output Examples
- .... and many more

Reactive Systems

Systems that **react to environment inputs** in potentially **infinite executions**

Reactive Systems

Systems that **react to environment inputs** in potentially **infinite executions**



Typical examples:

- Hardware circuits
- Communication protocols
- Embedded controllers
- Operating systems

Synthesis of Reactive Systems

Church's problem (1957)

Given a **specification** φ over sequences of valuations of input and output Boolean variables,

construct a **finite-state implementation** that satisfies φ for all possible infinite sequences of input values.

[Church, Summaries of the Summer Institute of Symbolic Logic, 1957]
Application of Recursive Arithmetic to the Problem of Circuit Synthesis

[Church, International Congress of Mathematicians, 1962]
Logic, Arithmetic and Automata

Synthesis of Reactive Systems

Church's problem (1957)

Given a **specification** φ over sequences of valuations of input and output Boolean variables,

construct a **finite-state implementation** that satisfies φ for all possible infinite sequences of input values.

Applications today

- design of communication protocols
- robotic motion planning
- control of autonomous systems
- ...

Motivation: Synthesis of Infinite-State Reactive Systems

Program Synthesis

goal: classical program

spec: input/output relation
+ grammar

➡ Synthesize **data transformations**

Finite-State Reactive Synthesis

goal: finite-state controller
(e.g. Mealy machine)

spec: temporal logics (e.g. LTL)

➡ Synthesize **abstract control**

Motivation: Synthesis of Infinite-State Reactive Systems

Program Synthesis

goal: classical program

spec: input/output relation
+ grammar

➡ Synthesize **data transformations**

Finite-State Reactive Synthesis

goal: finite-state controller
(e.g. Mealy machine)

spec: temporal logics (e.g. LTL)

➡ Synthesize **abstract control**

Many problems need both control and data transformations

Motivation: Synthesis of Infinite-State Reactive Systems

Program Synthesis

goal: classical program

spec: input/output relation

\+ grammar

➡ Synthesize **data transformations**

Finite-State Reactive Synthesis

goal: finite-state controller

(e.g. Mealy machine)

spec: temporal logics (e.g. LTL)

➡ Synthesize **abstract control**

Many problems need both control and data transformations

The focus of this course

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations

- Symbolic methods enhanced with logical reasoning
  - for solving realizability/synthesis games
  - for translation of temporal logic formulas
  - combining symbolic methods and abstraction

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations

- Symbolic methods enhanced with logical reasoning
  - for solving realizability/synthesis games
  - for translation of temporal logic formulas
  - combining symbolic methods and abstraction

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations

- Symbolic methods enhanced with logical reasoning
  - for solving realizability/synthesis games
  - for translation of temporal logic formulas
  - combining symbolic methods and abstraction

Synthesis from Specifications

It is easier to say what a system should do than how it should be done!

Examples:

- Eventually the next waypoint should be reached.
- The vehicle should never collide with obstacles.

We need an expressive high-level specification language!

Linear Temporal Logic (LTL)

[Pnueli, Annual Symposium on Foundations of Computer Science, 1977]

- A logic to reason about execution traces of reactive systems
- States/transitions of the system are labeled by atomic propositions
- An (infinite) execution of the system induces an infinite sequence of truth values for the atomic propositions: execution trace
- LTL has modalities referring to the temporal flow of the truth values

LTL Syntax and Semantics

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi\mathbf{U}\varphi$$

where p ranges over a finite set P of Boolean atomic propositions

LTL Syntax and Semantics

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi\mathbf{U}\varphi$$

where p ranges over a finite set P of Boolean atomic propositions

traces $\pi \in (2^P)^\omega$ are infinite sequences of sets of atomic propositions

For $n \in \mathbb{N}$, $\pi, n \vDash \varphi \iff \varphi$ holds at position n of π

A trace $\pi \in (2^P)^\omega$ satisfies φ if and only if $\pi, 0 \vDash \varphi$

We denote with $L(\varphi)$ the set of traces that satisfy φ

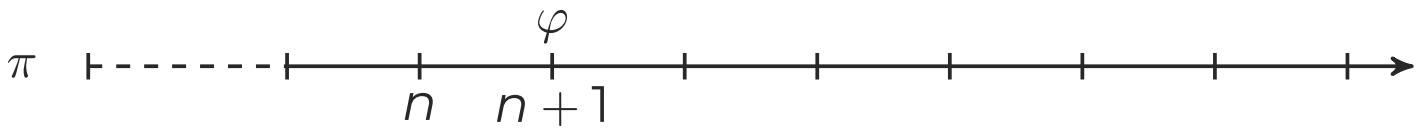
LTL Semantics: Propositions and Boolean Connectives

- $\pi, n \models p$ iff $p \in \pi_n$
- $\pi, n \models \neg\varphi$ iff it is not the case that $\pi, n \models \varphi$
- $\pi, n \models \varphi \wedge \psi$ iff $\pi, n \models \varphi$ and $\pi, n \models \psi$
- $\pi, n \models \varphi \vee \psi$ iff $\pi, n \models \varphi$ or $\pi, n \models \psi$

LTL Semantics: Temporal Operators

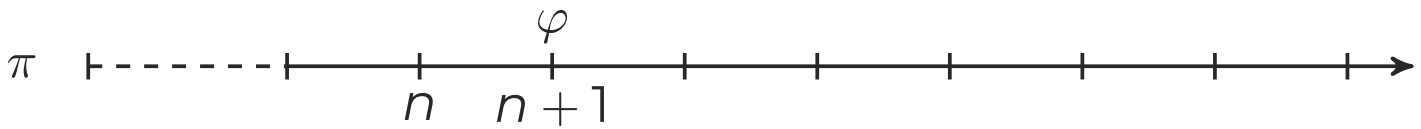
LTL Semantics: Temporal Operators

- $\pi, n \models \mathbf{X}\ \varphi$ iff $\pi, n+1 \models \varphi$ (at the next state φ is satisfied)

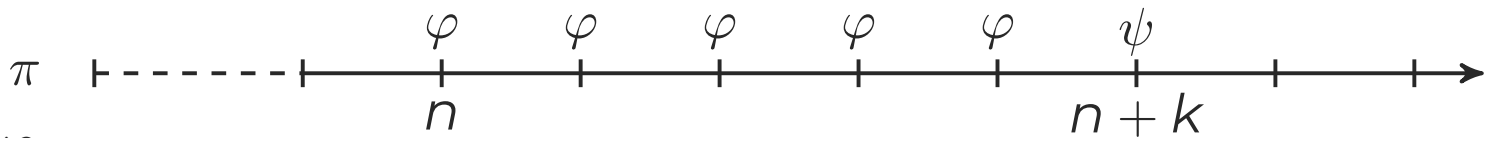


LTL Semantics: Temporal Operators

- $\pi, n \models \mathbf{X}\ \varphi$ iff $\pi, n+1 \models \varphi$ (at the next state φ is satisfied)



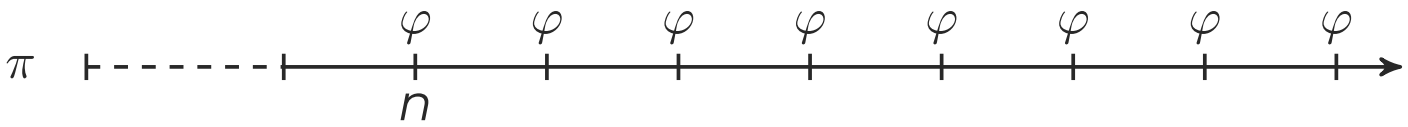
- $\pi, n \models \varphi\ \mathbf{U}\ \psi$ iff there is a $k \geq 0$ such that $\pi, n+k \models \psi$ and $\pi, n+j \models \varphi$ for all $0 \leq j < k$ (φ holds until a position where ψ is satisfied is reached)



LTL Semantics: Derived Temporal Operators

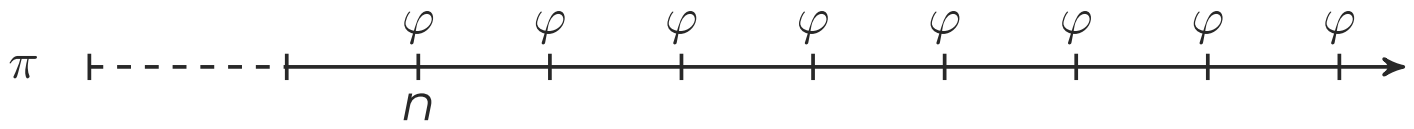
LTL Semantics: Derived Temporal Operators

- $\pi, n \models \mathbf{G}\ \varphi$ iff for all $k \geq 0$ it holds that $\pi, n + k \models \varphi$ (φ is satisfied **globally**)

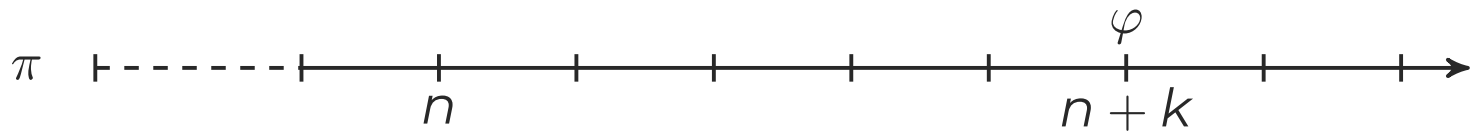


LTL Semantics: Derived Temporal Operators

- $\pi, n \models \mathbf{G}\ \varphi$ iff for all $k \geq 0$ it holds that $\pi, n + k \models \varphi$ (φ is satisfied **globally**)



- $\pi, n \models \mathbf{F}\ \varphi$ iff there is a $k \geq 0$ such that $\pi, n + k \models \varphi$ (φ is satisfied **finally**)



LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

Mutual exclusion: At no point in time there should be both g_1 and g_2 in the output.

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

Mutual exclusion: At no point in time there should be both g_1 and g_2 in the output.

$$\mathbf{G}\neg(g_1 \wedge g_2)$$

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

Response: Every request should eventually be followed by a grant for that client.

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

Response: Every request should eventually be followed by a grant for that client.

$$\mathbf{G}((r_1 \rightarrow \mathbf{F}\ g_1) \wedge (r_2 \rightarrow \mathbf{F}\ g_2))$$

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

No spurious grants: Only give grant to client c if there is an open request from client c

LTL: A classical example — Arbiter Circuit

The set of atomic propositions P is partitioned into set of inputs I and set of outputs O.

An arbiter circuit

- receives requests r_1, r_2 from two clients and
- produces grants g_1, g_2 for the two clients.

Example properties

No spurious grants: Only give grant to client c if there is an open request from client c

$$\bigwedge_{c \in \{1,2\}} \neg[\,(\neg r_c \,\mathbf{U}\, (\neg r_c \wedge g_c))\,] \;\wedge\; \neg[\,\mathbf{F}\,(g_c \wedge \mathbf{X}\,(\neg r_c \,\mathbf{U}\, (\neg r_c \wedge g_c)))\,]$$

Satisfiability vs Realizability

Satisfiability: Is there a **trace** that satisfies the specification?

That is, does $L(\varphi) \neq \emptyset$ hold?

Realizability: Is there a **system** that satisfies specification?

Meaning: the executions resulting from all input sequences satisfy the specification.

Satisfiability vs Realizability

Satisfiability: Is there a **trace** that satisfies the specification?
That is, does $L(\varphi) \neq \emptyset$ hold?

Realizability: Is there a **system** that satisfies specification?
Meaning: the executions resulting from all input sequences satisfy the specification.

Example specification: Mutual exclusion + every request is granted in the next step

Satisfiability vs Realizability

Satisfiability: Is there a **trace** that satisfies the specification?
That is, does $L(\varphi) \neq \emptyset$ hold?

Realizability: Is there a **system** that satisfies specification?
Meaning: the executions resulting from all input sequences satisfy the specification.

Example specification: Mutual exclusion + every request is granted in the next step

$$\mathbf{G}\neg(g_1 \wedge g_2) \wedge \bigwedge_{c \in 1,2} \mathbf{G}(r_c \rightarrow \mathbf{X} g_c)$$

Satisfiability vs Realizability

Satisfiability: Is there a **trace** that satisfies the specification?
That is, does $L(\varphi) \neq \emptyset$ hold?

Realizability: Is there a **system** that satisfies specification?
Meaning: the executions resulting from all input sequences satisfy the specification.

Example specification: Mutual exclusion + every request is granted in the next step

$$\mathbf{G}\neg(g_1 \wedge g_2) \wedge \bigwedge_{c \in 1,2} \mathbf{G}(r_c \rightarrow \mathbf{X} g_c)$$

satisfiable but not realizable!

Linear Temporal Logic + Data

In many cases we want to specify properties of systems over **unbounded domains**

- Input from sensors (physical environment)
- Counter variables (position, number of items, ….)
- ….

We need **temporal logics beyond propositional domains**.

Several proposals in the reactive synthesis community in the last couple of years.

Temporal Stream Logic (TSL)

[Finkbeiner/Klein/Piskac/Santolucito, CAV 2019]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

- τ_f is a function term
- τ_p is a predicate term
- $[\![s_o \leftarrowtail \tau_f]\!]$ is an update to an output or cell s_o

Temporal Stream Logic (TSL)

[Finkbeiner/Klein/Piskac/Santolucito, CAV 2019]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

- τ_f is a function term
- τ_p is a predicate term
- $[\![s_o \leftarrowtail \tau_f]\!]$ is an update to an output or cell s_o

Example:

$$\mathbf{G}(''100 <'' \; sensor \rightarrow [\![counter \leftarrowtail '' \; 1 +'' \; counter]\!])$$

Temporal Stream Logic (TSL)

[Finkbeiner/Klein/Piskac/Santolucito, CAV 2019]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

- τ_f is a function term
- τ_p is a predicate term
- $[\![s_o \leftarrowtail \tau_f]\!]$ is an update to an output or cell s_o

Example:

$$\mathbf{G}(''100 <'' \; sensor \rightarrow [\![counter \leftarrowtail '' \; 1 + '' \; counter]\!])$$

- The meaning of $''100 <''$ and $''1+''$ is unknown to the synthesis tool

Temporal Stream Logic (TSL)

[Finkbeiner/Klein/Piskac/Santolucito, CAV 2019]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

- τ_f is a function term
- τ_p is a predicate term
- $[\![s_o \leftarrowtail \tau_f]\!]$ is an update to an output or cell s_o

Example:

$$\mathbf{G}(''100 <'' \; sensor \rightarrow [\![counter \leftarrowtail '' \; 1 + '' \; counter]\!])$$

- The meaning of $''100 <''$ and $''1+''$ is unknown to the synthesis tool
- The synthesized strategy should work for any interpretation of the symbols

Temporal Stream Logic (TSL)

[Finkbeiner/Klein/Piskac/Santolucito, CAV 2019]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

- τ_f is a function term
- τ_p is a predicate term
- $[\![s_o \leftarrowtail \tau_f]\!]$ is an update to an output or cell s_o

Example:

$$\mathbf{G}(''100 <'' \; sensor \rightarrow [\![counter \leftarrowtail '' \; 1 +'' \; counter]\!])$$

- The meaning of $''100 <''$ and $''1+''$ is unknown to the synthesis tool
- The synthesized strategy should work for any interpretation of the symbols
- Necessary aspects of the semantics are added as assumptions to the specification

Temporal Stream Logic (TSL)

Key principle: separation of data and control

- use reactive synthesis for control, data handled with uninterpreted functions
- assumptions to add necessary information about uninterpreted functions

Data transformations are manually implemented or synthesized separately

Temporal Stream Logic (TSL)

Key principle: separation of data and control

- use reactive synthesis for control, data handled with uninterpreted functions
- assumptions to add necessary information about uninterpreted functions

Data transformations are manually implemented or synthesized separately

However, not all functions are uninterpreted

- logical theories such as arithmetic are commonly used
- encoding of the semantics as assumptions increases formula size

Temporal Stream Logic Modulo Theories (TSL-MT)

[Finkbeiner/Heim/Passing, FoSSaCS 2022]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

Extends TSL with first-order theories by restricting the possible interpretations of predicate and function symbols

Temporal Stream Logic Modulo Theories (TSL-MT)

[Finkbeiner/Heim/Passing, FoSSaCS 2022]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

Extends TSL with first-order theories by restricting the possible interpretations of predicate and function symbols

Example:

$$\mathbf{G}(\text{gt } \mathit{sensor}\ 100 \rightarrow [\![\mathit{counter} \leftarrowtail \text{add } \mathit{counter}\ 1]\!])$$

Temporal Stream Logic Modulo Theories (TSL-MT)

[Finkbeiner/Heim/Passing, FoSSaCS 2022]

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid [\![s_o \leftarrowtail \tau_f]\!] \mid \tau_p$$

Extends TSL with first-order theories by restricting the possible interpretations of predicate and function symbols

Example:

$$\mathbf{G}(\text{gt } \mathit{sensor}\ 100 \rightarrow [\![\mathit{counter} \leftarrowtail \text{add } \mathit{counter}\ 1]\!])$$

LTL$_{\mathcal{T}}$

[Rodríguez/Sánchez, CAV 2023]

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where a is a literal from a first-order theory $\mathcal{T}$ over input and output variables

$\mathbf{LTL}_{\mathcal{T}}$

[Rodríguez/Sánchez, CAV 2023]

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where a is a literal from a first-order theory $\mathcal{T}$ over input and output variables

Example:

$$\mathbf{G}(x > 10 \wedge y \leq 2 \rightarrow \mathbf{X}\ x \leq y)$$

$\mathbf{LTL}_{\mathcal{T}}$

[Rodríguez/Sánchez, CAV 2023]

$$\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where a is a literal from a first-order theory $\mathcal{T}$ over input and output variables

Example:

$$\mathbf{G}(x > 10 \wedge y \leq 2 \rightarrow \mathbf{X}\ x \leq y)$$

Unlike TSL-MT, does not have updates

➡ Limited expressivity for flow of data over time

Reactive Program LTL (RP-LTL)

[Heim/Dimitrova, POPL 2025]

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Reactive Program LTL (RP-LTL)

[Heim/Dimitrova, POPL 2025]

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Example:

$$(\mathbf{FG}\ i < 0) \rightarrow (x' \geq x + i \quad \mathbf{U} \quad x \leq 0)$$

- Integer **state** x
- Integer **input** i, environment-controlled
- **Next-state** x', system-controlled

Reactive Program LTL (RP-LTL)

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Reactive Program LTL (RP-LTL)

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Example:

$$(\mathbf{FG}\ i < 0) \rightarrow (x' \geq x + i \quad \mathbf{U} \quad x \leq 0)$$

- Allows for relating current and next-state variables
- Allows more general relations than assignments

Reactive Program LTL (RP-LTL)

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Traces are of the form $\mathbf{x}_0, \mathbf{i}_0, \mathbf{x}_1, \mathbf{i}_1, \ldots$

- $\mathbf{x}_k$ — k-th state
- $\mathbf{i}_k$ — k-th input

Reactive Program LTL (RP-LTL)

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

where α is a quantifier-free formula over **inputs, current and next-state variables**

Traces are of the form $\mathbf{x}_0, \mathbf{i}_0, \mathbf{x}_1, \mathbf{i}_1, \ldots$

- $\mathbf{x}_k$ — k-th state
- $\mathbf{i}_k$ — k-th input

Realizability: Does there exist a **function that maps trace prefixes** to next valuation of state variables such that all resulting traces satisfy the specification?

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

- **X G** $(i > 42 \rightarrow x = 0)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

- **X G** $(i > 42 \rightarrow x = 0)$

  yes; always set x to 0

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

- **X G** $(i > 42 \rightarrow x = 0)$

  yes; always set x to 0

- **X G** $(i > 42 \leftrightarrow x = 0)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

- **X G** $(i > 42 \rightarrow x = 0)$

  yes; always set x to 0

- **X G** $(i > 42 \leftrightarrow x = 0)$

  no; environment can choose i based on current value of x

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

- **X G** $(i > 42 \rightarrow x = 0)$

  yes; always set x to 0

- **X G** $(i > 42 \leftrightarrow x = 0)$

  no; environment can choose i based on current value of x

- **G** $(i > 42 \leftrightarrow x' = x + 1)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **G** $(i > 42 \rightarrow x = 0)$

  no; initial value of x is arbitrary

- **X G** $(i > 42 \rightarrow x = 0)$

  yes; always set x to 0

- **X G** $(i > 42 \leftrightarrow x = 0)$

  no; environment can choose i based on current value of x

- **G** $(i > 42 \leftrightarrow x' = x + 1)$

  yes; system can set x' based on current value of i

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- **F** $(i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

- $(\mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

- $(\mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  no; environment can set i to 0 only at the first step

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

- $(\mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  no; environment can set i to 0 only at the first step

- $(\mathbf{X}\ \mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

- $(\mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  no; environment can set i to 0 only at the first step

- $(\mathbf{X}\ \mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  yes; always set x to 42

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

- $(\mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  no; environment can set i to 0 only at the first step

- $(\mathbf{X}\ \mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  yes; always set x to 42

- $(\mathbf{G}\ \mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

RP-LTL Realizability: Examples

Input integer i, state variable integer x

Are the following RP-LTL formulas realizable?

If yes, what is a possible implementation?

If no, how should the environment behave to defeat any possible implementation?

- $\mathbf{F}\ (i = 0 \wedge x = 42)$

  no; the environment can always set i to non-zero value

- $(\mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  no; environment can set i to 0 only at the first step

- $(\mathbf{X}\ \mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  yes; always set x to 42

- $(\mathbf{G}\ \mathbf{F}\ (i = 0)) \rightarrow \mathbf{F}\ (i = 0 \wedge x = 42)$

  yes; always set x to 42

Realizability as a Game: System against Environment

Interaction between system and environment as a **two-player game:**

- Environment provides inputs
- System provides outputs

The system satisfies φ if the executions resulting from all input sequences satisfy φ.

φ is **realizable** $\iff$ the **system player has a winning strategy** in the game

Infinite-State Games

- Reactive Program Games (RPG)

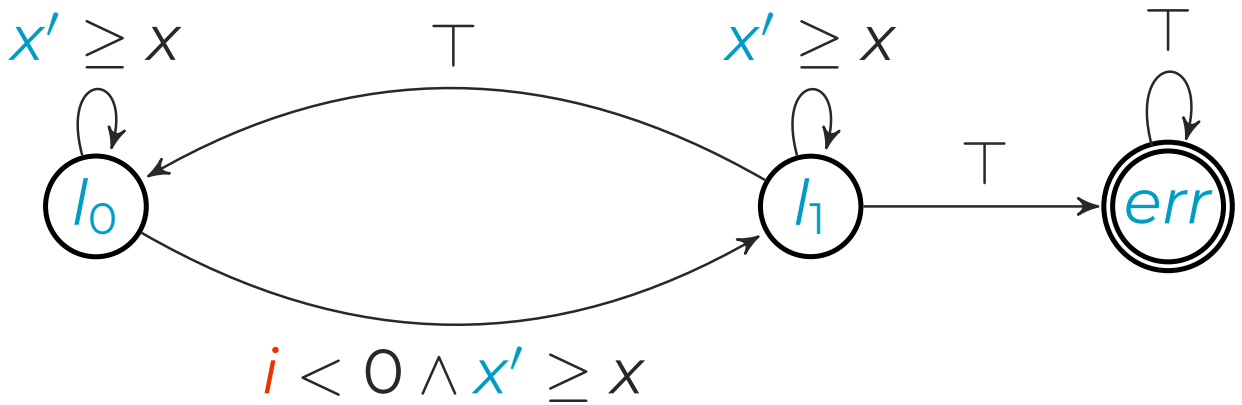
[Heim/Dimitrova, POPL 2024]

- Games with LTL objectives

[Azzopardi/Di Stefano/Piterman/Schneider, CAV 2025]

Symbolic Games

[Heim/Dimitrova, POPL 2025]

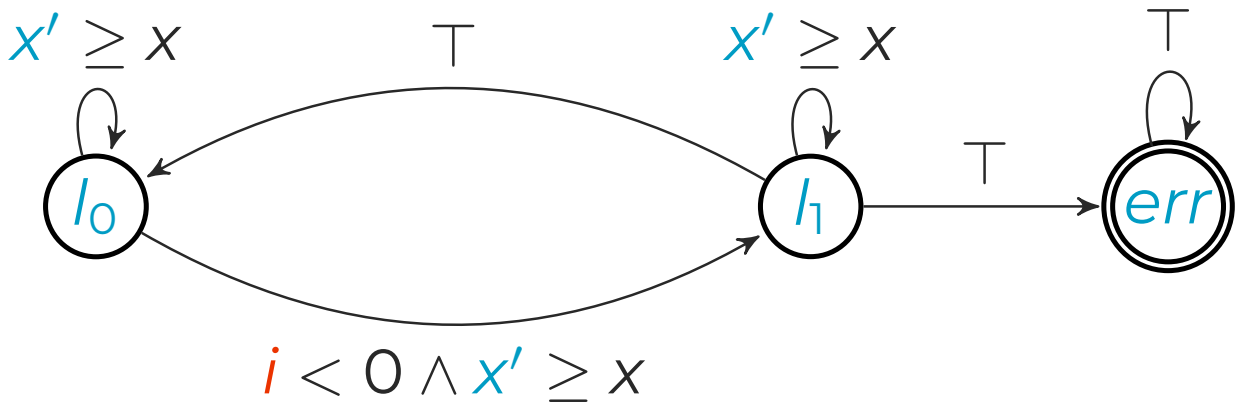


- **Locations** l_0, l_1, err
- Input variable i
- State variable x
- Transitions labelled with quantifier-free FO formulas over input, state and next-state variables

Winning condition: here avoid err

Symbolic Games

[Heim/Dimitrova, POPL 2025]



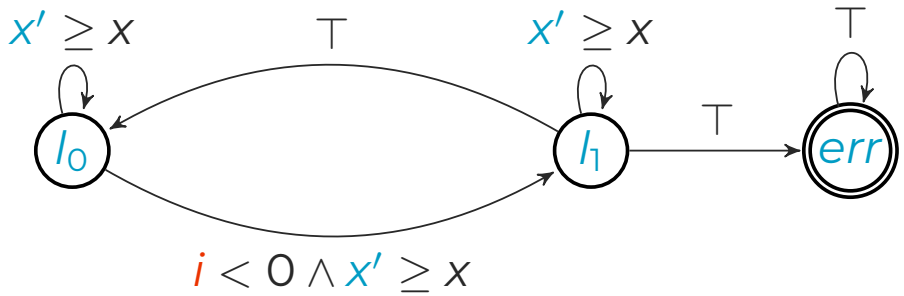
- **Locations** l_0, l_1, err
- Input variable i
- State variable x
- Transitions labelled with quantifier-free FO formulas over input, state and next-state variables

Winning condition: here avoid err

Winning condition in general: safety, reachability, Büchi, parity on locations

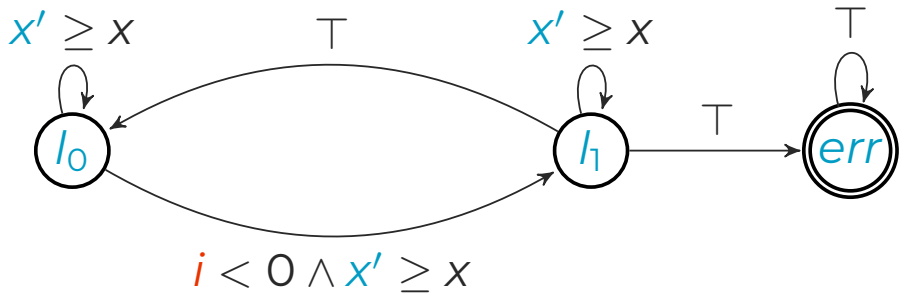
Both Specification Formalisms Have Advantages

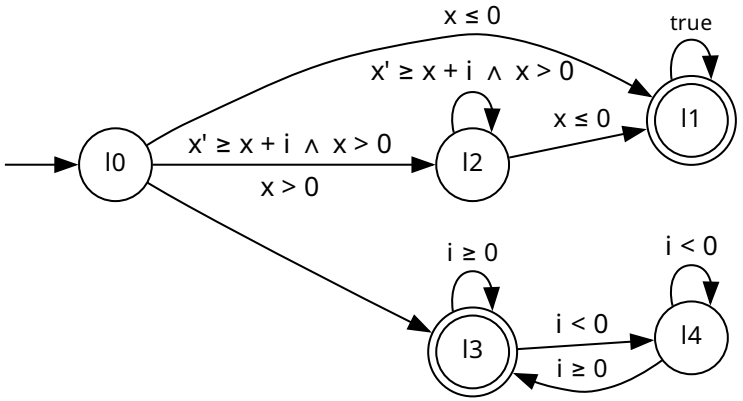
$$(\mathbf{FG}\ i < 0) \rightarrow (x' \geq x + i \quad \mathbf{U} \quad x \leq 0)$$



Both Specification Formalisms Have Advantages

$$(\mathbf{FG}\ i < 0) \rightarrow (x' \geq x + i \quad \mathbf{U} \quad x \leq 0)$$





instead of

$$l = 0 \rightarrow$$

$$\mathbf{G}\ ((l = 0 \rightarrow l' = 0 \wedge x' \geq x \vee l' = 1 \wedge i < 0 \wedge x' \geq x) \wedge (l = 1 \rightarrow l' = 1 \wedge x' \geq x \vee l' = 0 \vee l' = 2) \wedge \neg(l = 2))$$

Multi-Modal Specifications: The Issy Format

[Heim/Dimitrova, CAV 2025]

```
input int i
state int x

formula {
 assume F G [i < 0]
 assert
    [x' >= x + i] U [x <= 0]
}

game Safety from l0 {
 loc l0 1
 loc l1 1
 loc err 0
 from l0 to l0 with
    [x' >= x]
 from l0 to l1 with
    [x' >= x] && [i < 0]

 from l1 to l1 with
    [x' >= x]
 from l1 to l0 with true
 from l1 to err with true

 from err to err with true
}
```

Multi-Modal Specifications: The Issy Format

```
input int i
state int x

formula {
 assume F G [i < 0]
 assert
    [x' >= x + i] U [x <= 0]
}

game Safety from l0 {
 loc l0 1
 loc l1 1
 loc err 0
 from l0 to l0 with
    [x' >= x]
 from l0 to l1 with
    [x' >= x] && [i < 0]

 from l1 to l1 with
    [x' >= x]
 from l1 to l0 with true
 from l1 to err with true

 from err to err with true
}
```

Multi-Modal Specifications: The Issy Format

```
input int i
state int x

formula {
 assume F G [i < 0]
 assert
    [x' >= x + i] U [x <= 0]
}

game Safety from l0 {
 loc l0 1
 loc l1 1
 loc err 0
 from l0 to l0 with
    [x' >= x]
 from l0 to l1 with
    [x' >= x] && [i < 0]

 from l1 to l1 with
    [x' >= x]
 from l1 to l0 with true
 from l1 to err with true

 from err to err with true
}
```

Multi-Modal Specifications: The Issy Format

```
input int i
state int x

formula {
 assume F G [i < 0]
 assert
    [x' >= x + i] U [x <= 0]
}
```

```
game Safety from l0 {
 loc l0 1
 loc l1 1
 loc err 0

 from l0 to l0 with
    [x' >= x]
 from l0 to l1 with
    [x' >= x] && [i < 0]

 from l1 to l1 with
    [x' >= x]
 from l1 to l0 with true
 from l1 to err with true

 from err to err with true
}
```

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations
- Symbolic methods enhanced with logical reasoning
  - for solving realizability/synthesis games
  - for translation of temporal logic formulas
  - combining symbolic methods and abstraction

Omega Automata

ω-automata recognize sets of **infinite sequences**

A **nondeterministic ω-automaton** $A = (\Sigma, Q, q_0, \delta, acc)$ consists of

- an alphabet Σ
- a finite set of states Q
- an initial state $q_0 \in Q$
- a transition function $\delta : Q \times \Sigma \to 2^Q$
- acceptance condition acc (that determines which runs are accepting)

An infinite word $\sigma = \sigma_0\sigma_1\sigma_2\ldots$is accepted by A if and only if there exists an accepting run of A on σ.

Nondeterministic Büchi Automata (NBA)

Büchi acceptance condition

A Büchi condition is a set of accepting states $F \subseteq Q$.

An infinite sequence $q_0 q_1 q_2 \ldots$ of states is Büchi-accepting
if and only if for infinitely many indices n it holds that $q_n \in F$.

Nondeterministic Büchi Automata (NBA)

Büchi acceptance condition

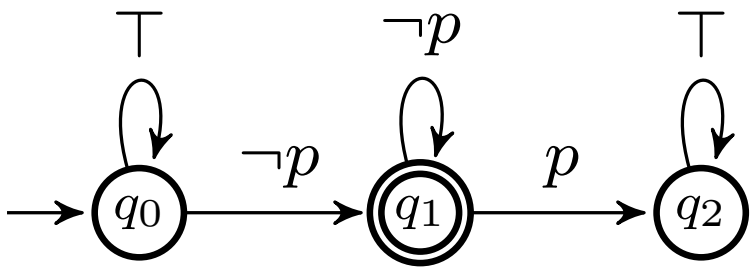
A Büchi condition is a set of accepting states $F \subseteq Q$.

An infinite sequence $q_0 q_1 q_2 \ldots$ of states is Büchi-accepting
if and only if for infinitely many indices n it holds that $q_n \in F$.

Example:

Let p be an atomic proposition.
Consider the language of words where "p holds only finitely often."

$\mathscr{L} = \{\pi_0 \pi_1 \pi_2 \ldots \in (2^{\{p\}})^\omega \mid p \in \pi_n \text{ for only finitely many } n \in \mathbb{N}\}$.



LTL to NBA

Theorem
[Wolper/Vardi/Sistla, Symposium on Foundations of Computer Science 1983]

For every LTL formula φ there exists an NBA A such that $L(A) = L(\varphi)$.

Furthermore, the number of states of A is in $2^{\mathcal{O}(|\varphi|)}$,

where $|\varphi|$ is the number of syntactically distinct subformulas of φ.

Büchi Automata are More Expressive than LTL

Example:

Let p be an atomic proposition.
Consider the language of words where "p holds in every even position."

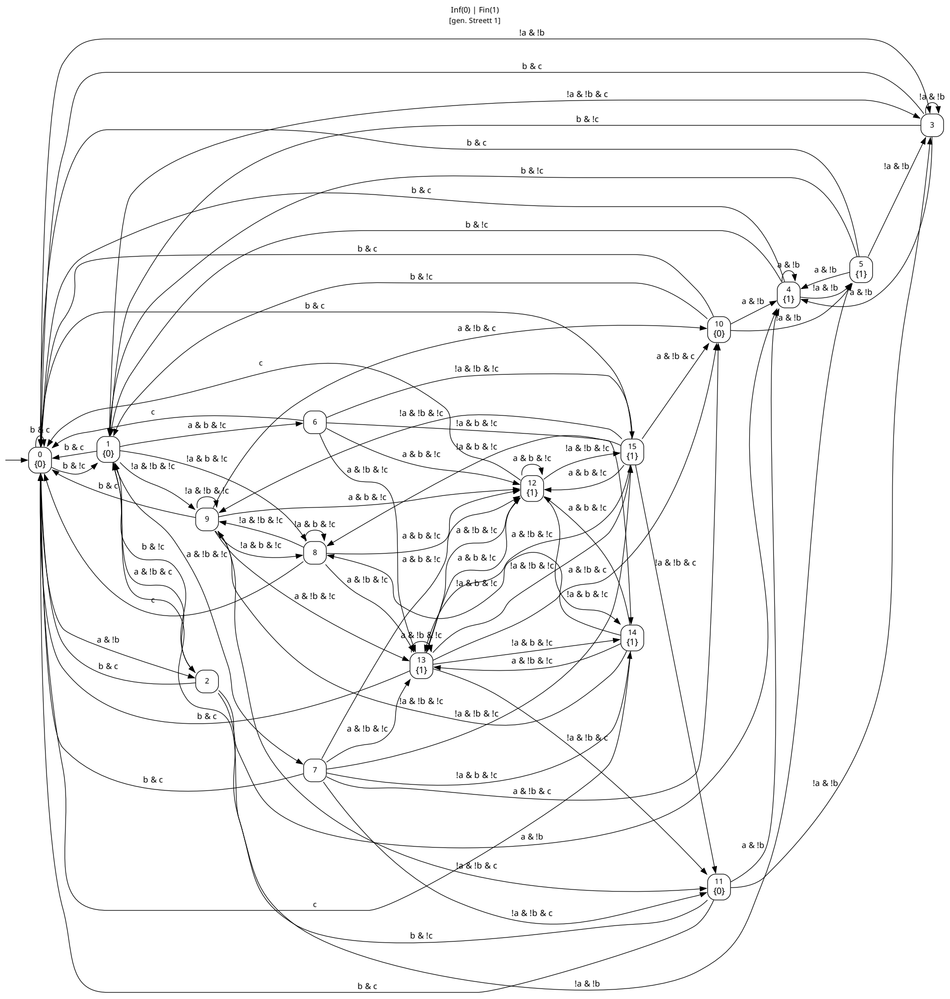
$$\mathscr{L} = \{\pi_0\pi_1\pi_2 \ldots \in (2^{\{p\}})^\omega \mid p \in \pi_{2k} \text{ for all k } \in \mathbb{N}\}.$$

There is no LTL formula φ with $L(\varphi) = \mathscr{L}$.
There exists an NBA A with $L(A) = \mathscr{L}$.

Why Logic?

$$\mathbf{GF}\ a \rightarrow \mathbf{GF}\ b \wedge \mathbf{GF}\ c$$

vs



Nondeterminism is Bad

To accept, nondeterministic automata can foresee the future (exists accepting run).

Strategies in synthesis games **cannot foresee the future!**

➡ The synthesis game cannot be played on NBA.

Example:

Let i be an input proposition and o be an output proposition.

$$\varphi = (\mathbf{FG}(i \wedge \mathbf{X}o)) \vee (\mathbf{GF}(\neg i \wedge \mathbf{X}\neg o))$$

There exists an implementation satisfying φ (copy input to output).

The system cannot choose between the two disjuncts.

Nondeterminism is Bad

To accept, nondeterministic automata can foresee the future (exists accepting run).

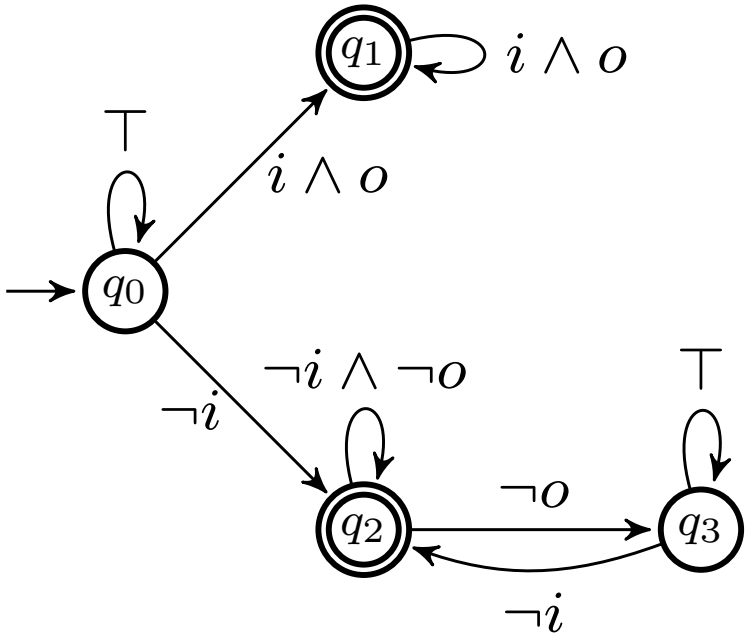
Strategies in synthesis games **cannot foresee the future!**

➡ The synthesis game cannot be played on NBA.

Example:

Let i be an input proposition and o be an output proposition.

$$\varphi = (\mathbf{FG}(i \wedge \mathbf{X}o)) \vee (\mathbf{GF}(\neg i \wedge \mathbf{X}\neg o))$$



Nondeterminism is Bad

To accept, nondeterministic automata can foresee the future (exists accepting run).

Strategies in synthesis games **cannot foresee the future!**

➡ The synthesis game cannot be played on NBA.

What about DBA (Deterministic Büchi Automaton)?

NBA are Strictly More Expressive Than DBA

Theorem [Landweber, Mathematical Systems Theory, 1969]

Nondeterministic Büchi automata are strictly more expressive than deterministic Büchi automata.

Example

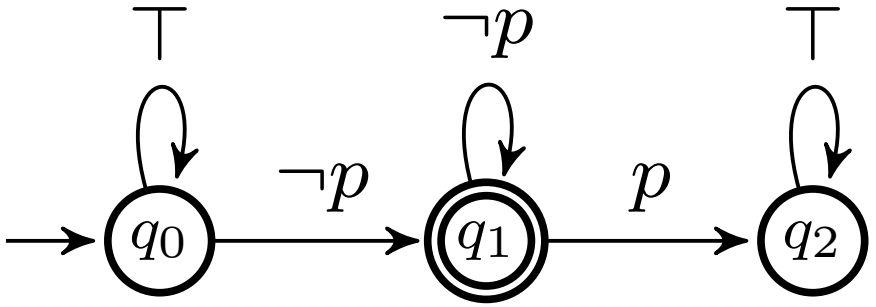
Let p be an atomic proposition.

Consider the language of words where "p holds only finitely often."

$$\mathscr{L} = \{\pi_0\pi_1\pi_2 \ldots \in (2^{\{p\}})^\omega \mid p \in \pi_n \text{ for only finitely many } n \in \mathbb{N}\}.$$

There is no DBA D with $L(D) = \mathscr{L}$.

There exists an NBA A with $L(A) = \mathscr{L}$.



More Powerful Accepting Conditions

Parity acceptance condition

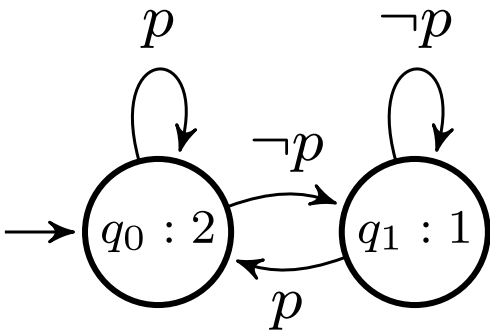
A parity condition is defined by a coloring function $\lambda : Q \to \mathbb{N}$.

An infinite sequence $q_0 q_1 q_2 \ldots$ of states is **max-odd-accepting** if and only if the highest color that appears infinitely often in the sequence $\lambda(q_0)\lambda(q_1)\lambda(q_2)\ldots$ is odd.

Example:

$\mathscr{L} = \{\pi_0\pi_1\pi_2 \ldots \in (2^{\{p\}})^\omega \mid p \in \pi_n \text{ for only finitely many } n \in \mathbb{N}\}$

DPA (Deterministic Parity Automaton)



Büchi Automata to Deterministic Parity Automata

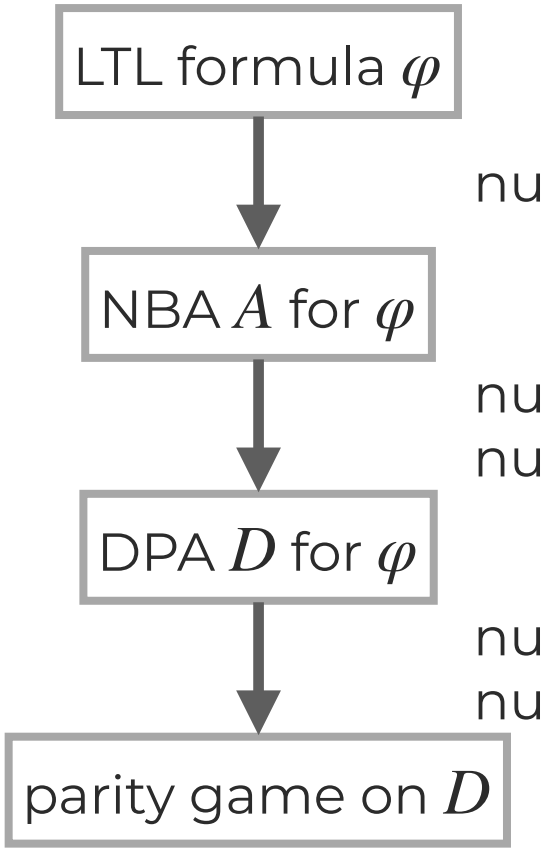
Theorem

For every NBA A there exists a DPA D such that $L(D) = L(A)$ with $2^{\mathcal{O}(|A|log|A|)}$ states and $\mathcal{O}(|A|)$ colors.

Corollary

For every LTL formula φ there is a DPA D with $L(D) = L(\varphi)$ and $2^{2^{\mathcal{O}(|\varphi|)}}$ states and $2^{\mathcal{O}(|\varphi|)}$ colors.

Classical Approach to Synthesis from LTL Specifications



number of states exponential

number of states **doubly exponential**,
number of colors exponential

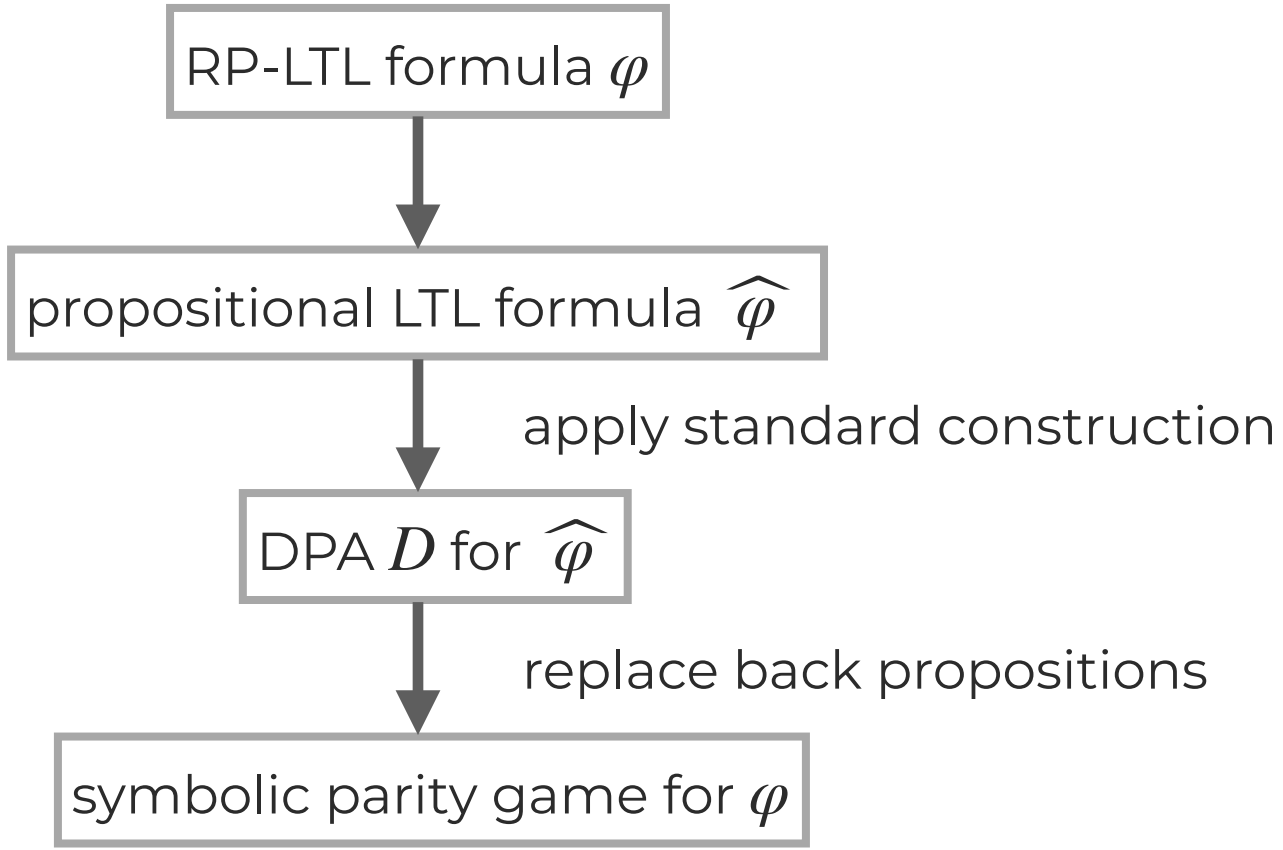
number of states **doubly exponential**,
number of colors exponential

game solving: polynomial in states
< exponential in number of colors

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- **Naive symbolic methods and their limitations**
- **Symbolic methods enhanced with logical reasoning**
  - **for solving realizability/synthesis games**
  - **for translation of temporal logic formulas**
  - **combining symbolic methods and abstraction**

Synthesis from RP-LTL Specifications



Direct Translation via LTL: Example

$$\mathbf{G}\,(x' = x + 1) \;\wedge\; \mathbf{GF}\,(x > 0)$$

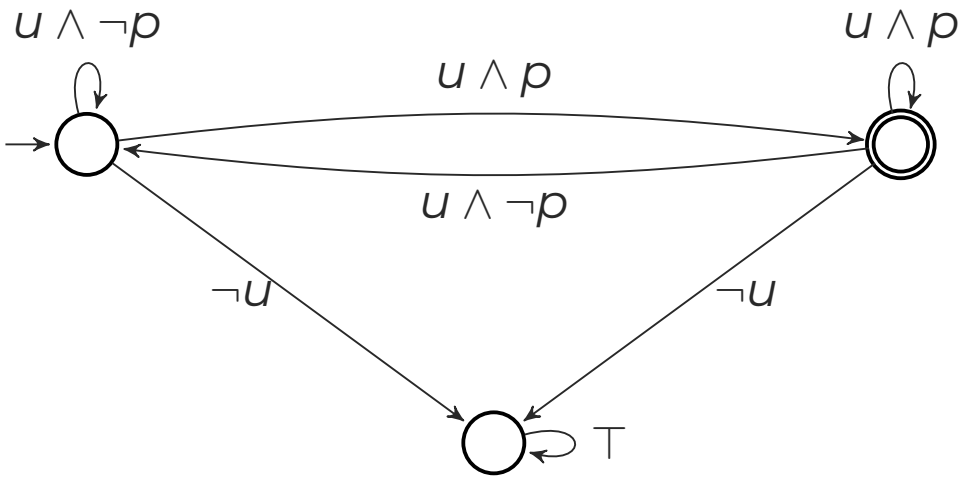
Step 1: Propositionalize to LTL formula

$$\mathbf{G}\,u \;\wedge\; \mathbf{GF}\,p$$

Direct Translation via LTL: Example

$$\mathbf{G}\, u \;\wedge\; \mathbf{GF}\, p$$

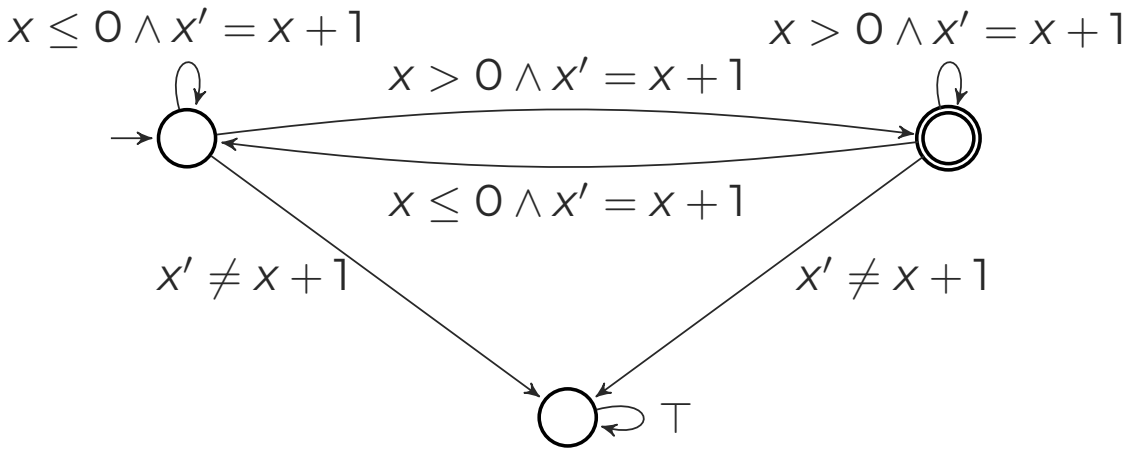
Step 2: Construct deterministic automaton for LTL formula



Direct Translation via LTL: Example

$$\mathbf{G}\,(x' = x + 1) \;\wedge\; \mathbf{GF}\,(x > 0)$$

Step 3: Transform back and construct game



(In this example, a one-player game)

Abstraction-Based Methods

Abstract and reduce to Boolean temporal formula or finite-state game

- TSL synthesis + Syntax-Guided Synthesis for data transformations

[Choi/Finkbeiner/Piskac/Santolucito, PLDI 2022]

- TSL synthesis + Predicate abstraction

[Maderbacher/Bloem, FMCAD 2022]

- Boolean-abstraction approach for realizability of LTL + logical theories

[Rodríguez/Sánchez, CAV 2023]

- CEGAR-based LTL synthesis on graphs defined by programs

[Azzopardi/Di Stefano/Piterman/Schneider, CAV 2025]

Abstraction-Based Methods

Abstract and reduce to Boolean temporal formula or finite-state game

Most common limitations

- refinement increases size of specification significantly
- iterative refinement can diverge in simple cases

Symbolic Methods

Lift classical game-solving algorithms to symbolic representation

- Symbolic fixpoint-based method for infinite-state games with LTL objectives

[Samuel/D'Souza/Komondoor, ASE 2023]

- CHC-based approach for infinite-state reachability games

[Faella/Parlato, AAAI 2023]

- Encode as nested fixpoint in μCLP (expressive first-order fixpoint logic)

[Unno/Satake/Terauchi/Koskinen, POPL 2023]

Symbolic Methods

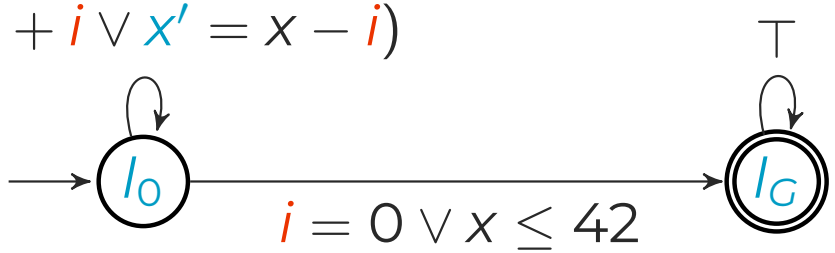
Lift classical game-solving algorithms to symbolic representation.

Most common limitations

- sometimes restricted specification types (safety and reachability)
- naive symbolic methods diverge in simple cases

Symbolic Games

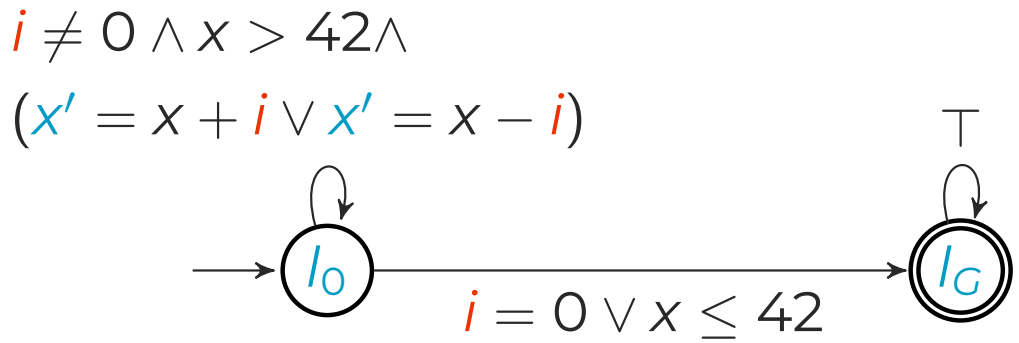
$i \neq 0 \wedge x > 42 \wedge$
$(x' = x + i \vee x' = x - i)$



- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

Symbolic Games



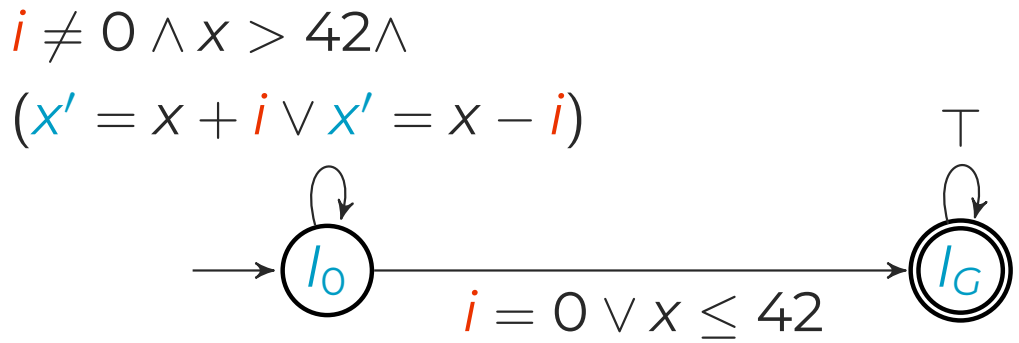
- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

Semantics

- **States:** (l, a) where l is a location and a maps the program variables to values
- **Transitions**: environment selects values for inputs, system selects next state

Symbolic Games



- Locations $\{l_0, l_G\}$
- Input variables $\{i\}$
- Program variables $\{x\}$

Spec: reach l_G from l_0 for any initial value in x

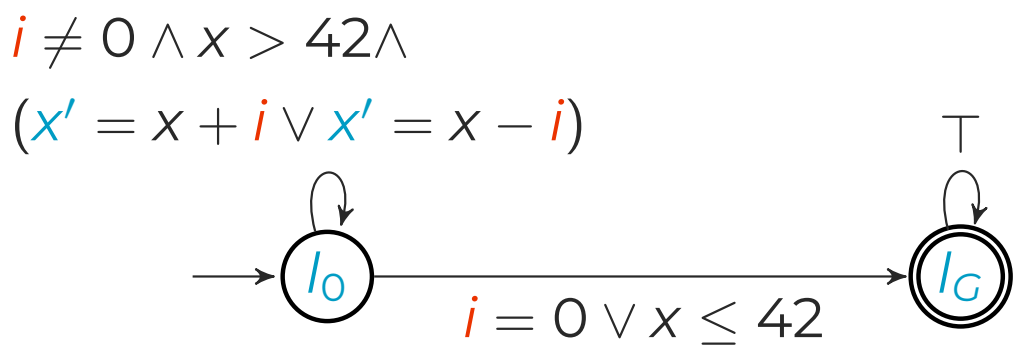
Semantics

- **States:** (l, a) where l is a location and a maps the program variables to values
- **Transitions**: environment selects values for inputs, system selects next state

Symbolic representation of sets of states:

functions from locations to first-order formulas over program variables

Solving Infinite-State Games



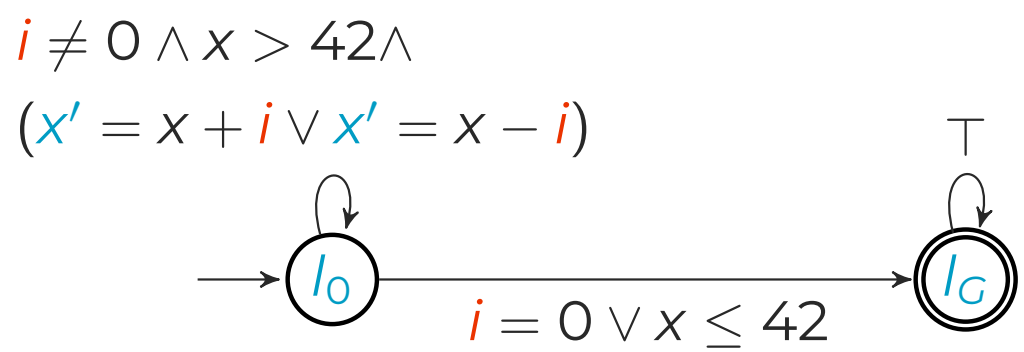
Spec: reach l_G from l_0 for any initial value in x

Semantics

- **States:** (l, a) where l is a location and a maps the program variables to values
- **Transitions**: environment selects values for inputs, system selects next state

Task: Compute the set of states W_p — the **winning region** for Player p

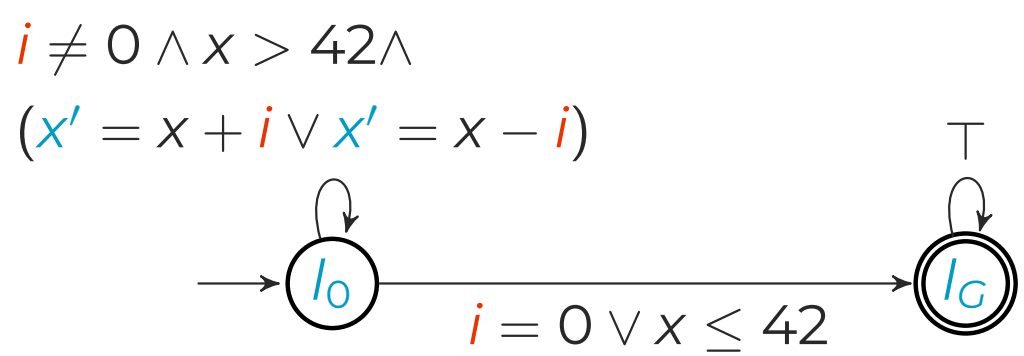
Solving Safety/Reachability Games



Computing W_{Sys} ...

Spec: reach l_G from l_0 for any initial value in x

Solving Safety/Reachability Games

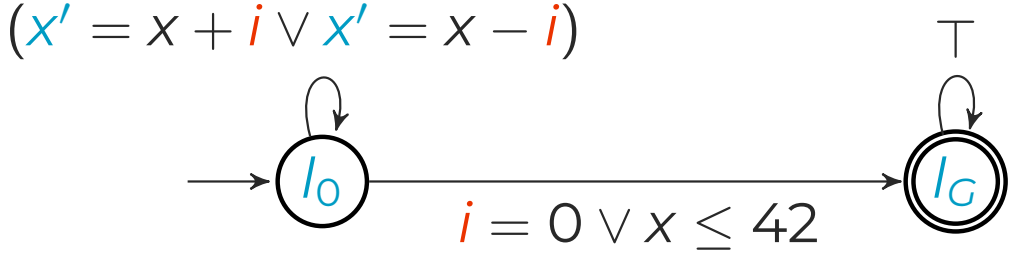


Computing W_{Sys} ...

$$Attr^0_{Sys} := \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$$

Spec: reach l_G from l_0 for any initial value in x

Solving Safety/Reachability Games



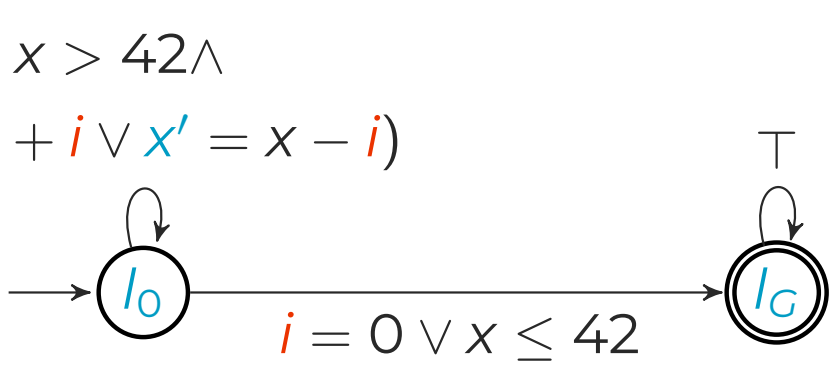
Computing W_{Sys} ...

$$Attr^0_{Sys} := \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$$

$$Attr^1_{Sys} := \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \top \}$$

Spec: reach l_G from l_0 for any initial value in x

Solving Safety/Reachability Games



Spec: reach l_G from l_0 for any initial value in x

Computing W_{Sys} ...

$$Attr^0_{Sys} := \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$$

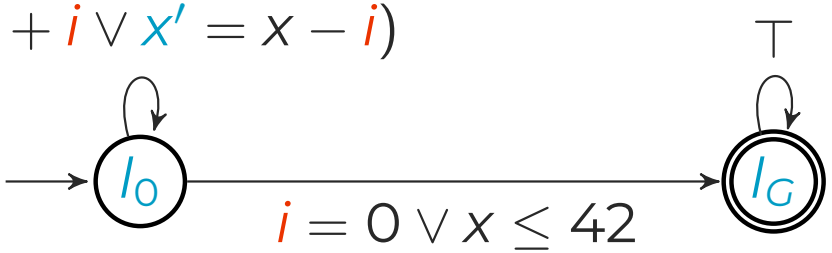
$$Attr^1_{Sys} := \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \top \}$$

$$Attr^2_{Sys} := \{ l_0 \mapsto x \leq 43, \, l_G \mapsto \top \}$$

Solving Safety/Reachability Games

$i \neq 0 \wedge x > 42 \wedge$

$(x' = x + i \vee x' = x - i)$



Spec: reach l_G from l_0 for any initial value in x

Computing W_{Sys} ...

$$Attr^0_{Sys} := \{ l_0 \mapsto \bot \, , \, l_G \mapsto \top \}$$

$$Attr^1_{Sys} := \{ l_0 \mapsto x \leq 42, \, l_G \mapsto \top \}$$

$$Attr^2_{Sys} := \{ l_0 \mapsto x \leq 43, \, l_G \mapsto \top \}$$

Does not terminate ...

Solving Safety/Reachability Games

Controllable predecessor operators:

map a (symbolically represented) sets of states to a (symbolically represented) sets of states

- Controllable predecessor for **system player**
  $(l, a) \in CPre_{\mathcal{G},Sys}(d)$ iff for every valid input i, there exists a state (l', a') such that
  i, a, a' satisfy the transition relation formula labelling edge (l, l')
  and $(l', a') \in d$

- Controllable predecessor for **environment player**
  $(l, a) \in CPre_{\mathcal{G},Env}(d)$ iff there exists a valid input i such that for all states (l', a') where
  i, a, a' satisfy the transition relation formula labelling edge (l, l')
  it holds that $(l', a') \in d$

Solving Safety/Reachability Games

Solving Safety/Reachability Games

Attractor: For a given player p and a given set of goal states d,
the attractor consists of the states from which player p
can enforce reaching a state in d in some number of steps

Solving Safety/Reachability Games

Attractor: For a given player p and a given set of goal states d,
the attractor consists of the states from which player p
can enforce reaching a state in d in some number of steps

Attractor Computation: can be done via iterative fixpoint computation,
which is not guaranteed to terminate in the case of an infinite state space

Solving Safety/Reachability Games

Attractor: For a given player p and a given set of goal states d,
the attractor consists of the states from which player p
can enforce reaching a state in d in some number of steps

Attractor Computation: can be done via iterative fixpoint computation,
which is not guaranteed to terminate in the case of an infinite state space

Solving Reachability Games: Compute attractor for reachability player (system)

Solving Safety/Reachability Games

Attractor: For a given player p and a given set of goal states d,
the attractor consists of the states from which player p
can enforce reaching a state in d in some number of steps

Attractor Computation: can be done via iterative fixpoint computation,
which is not guaranteed to terminate in the case of an infinite state space

Solving Reachability Games: Compute attractor for reachability player (system)

Solving Safety games: Compute attractor for environment player and error states,
then take the complement

Solving Safety/Reachability Games

Attractor: For a given player p and a given set of goal states d,
the attractor consists of the states from which player p
can enforce reaching a state in d in some number of steps

Attractor Computation: can be done via iterative fixpoint computation,
which is not guaranteed to terminate in the case of an infinite state space

Solving Reachability Games: Compute attractor for reachability player (system)

Solving Safety games: Compute attractor for environment player and error states,
then take the complement

Implementations can be extracted by book-keeping

Solving Safety/Reachability Games

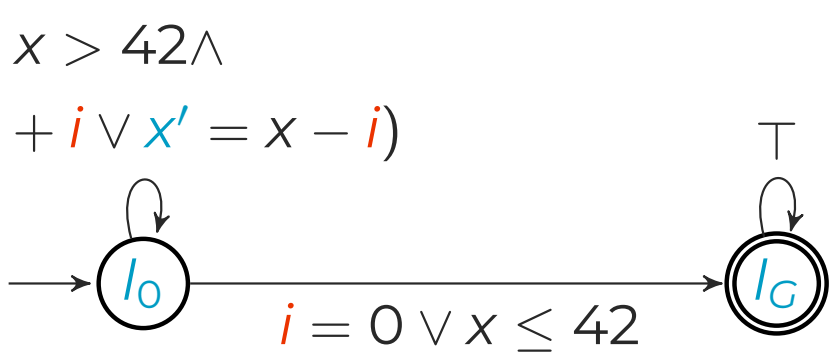
Symbolic Attractor Computation for Player p

- d is a symbolically represented set of states
- compute $d \vee CPre_{\mathcal{G},p}(d) \vee CPre_{\mathcal{G},p}(CPre_{\mathcal{G},p}(d) \vee d) \vee \ldots$ until reaching a fixpoint

function $\textsc{Attractor}(\mathcal{G}, p, d)$

1. $a^0 := \lambda l.\ \bot$; $a^1 := d$
2. **for** $n = 1, 2, \ldots$ **do**
3. **if** $a^n \equiv_T a^{n-1}$ **then return** a^n
4. $a^{n+1} := a^n \vee CPre_{\mathcal{G},p}(a^n)$

Issue: Non-Termination



Easy to see that

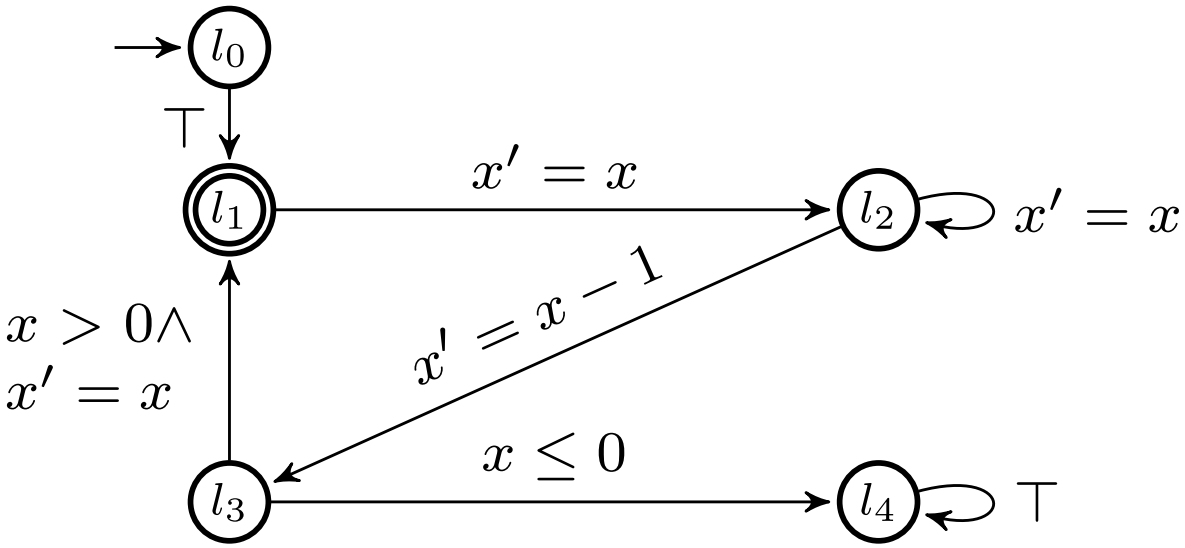
$W_{Sys} = \{ l_0 \mapsto \mathsf{T} \, , \, l_G \mapsto \mathsf{T} \, \}$

Spec: reach l_G from l_0 for any initial value in x

Challenges for automatic computation:

- number of iterations through l_0 is a priori **unbounded** (depends on initial value of x)
- in every step, the system has to perform **a strategic decision** which **depends on the input from the environment** at that step

Solving Büchi Games



Spec: visit location l_1 infinitely often

The environment wins the game for every possible state,

since the value of x chosen in l_0 bounds the number of visits to l_1.

Solving Büchi Games

Solving Büchi Games

Büchi games can be solved via **nested fixpoint computation:**

- the inner fixpoint iteration computes attractor sets for the Player p with Büchi objective
- the outer fixpoint iteration computes increasing underapproximations of the set of winning states for the Player $1 - p$ with co-Büchi objective

Implementations can be extracted by slightly more sophisticated book-keeping.

Solving Büchi Games

Symbolic Method for Solving Büchi Games

- f is the symbolically represented set of Büchi-accepting states for Player p
- a^n represents the states from which Player p can enforce a visit to f^n
- w^n_{1-p} represents the states from which Player $1-p$ can prevent Player p from revisiting f^n

function *SolveBüchiGame*$(\mathcal{G}, p, f)$

1 $f^0 := \lambda l.\ \top;\ f^1 := f;\ w^0_{1-p} := \lambda l.\bot$

2 **for** $n = 1, 2, \ldots$ **do**

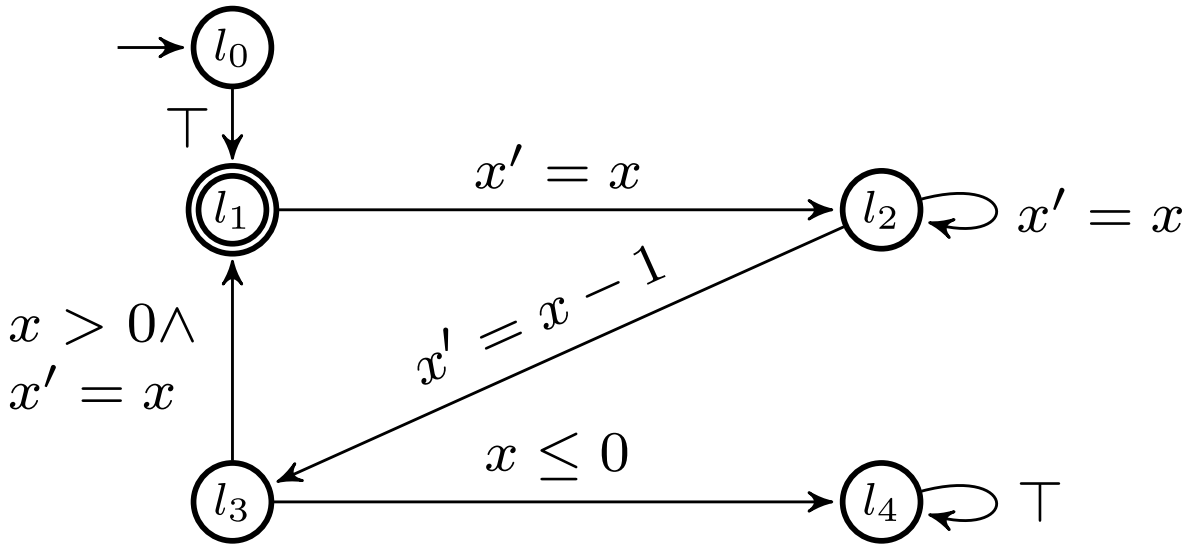
3 **if** $f^n \equiv_T f^{n-1}$ **then return** w^{n-1}_{1-p}

4 $a^n := \textsc{Attractor}(\mathcal{G}, p, f^n)$

5 $w^n_{1-p} := \textsc{Attractor}(\mathcal{G}, 1-p, \neg CPre_{\mathcal{G},p}(a^n))$

6 $f^{n+1} := f^n \wedge \neg w^n_{1-p}$

Solving Büchi Games

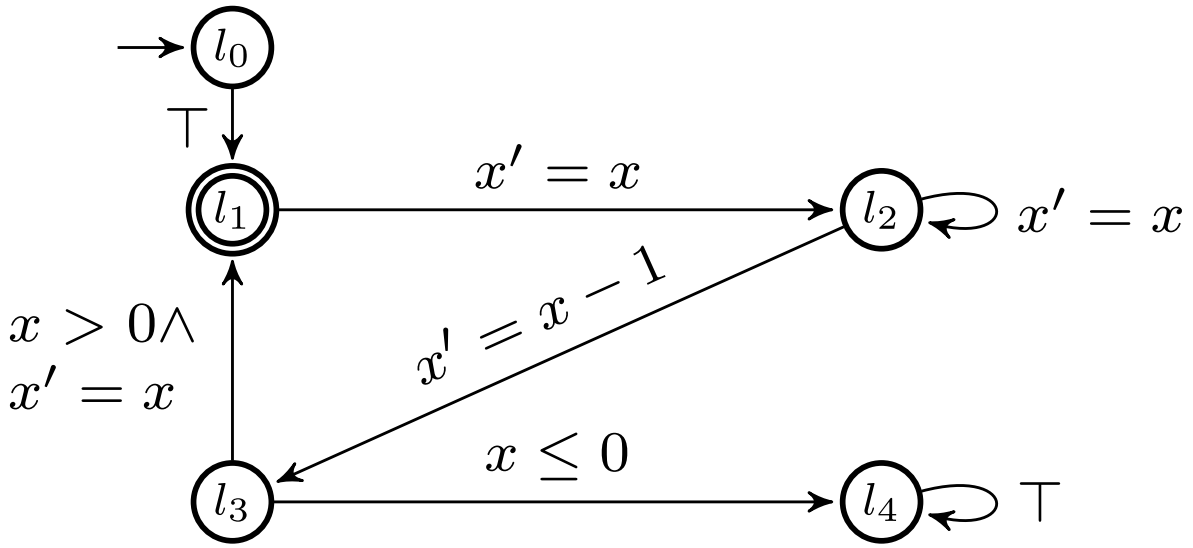


Spec: visit location l_1 infinitely often

The environment wins the game for every possible state,

since the value of x chosen in l_0 bounds the number of visits to l_1.

Solving Büchi Games



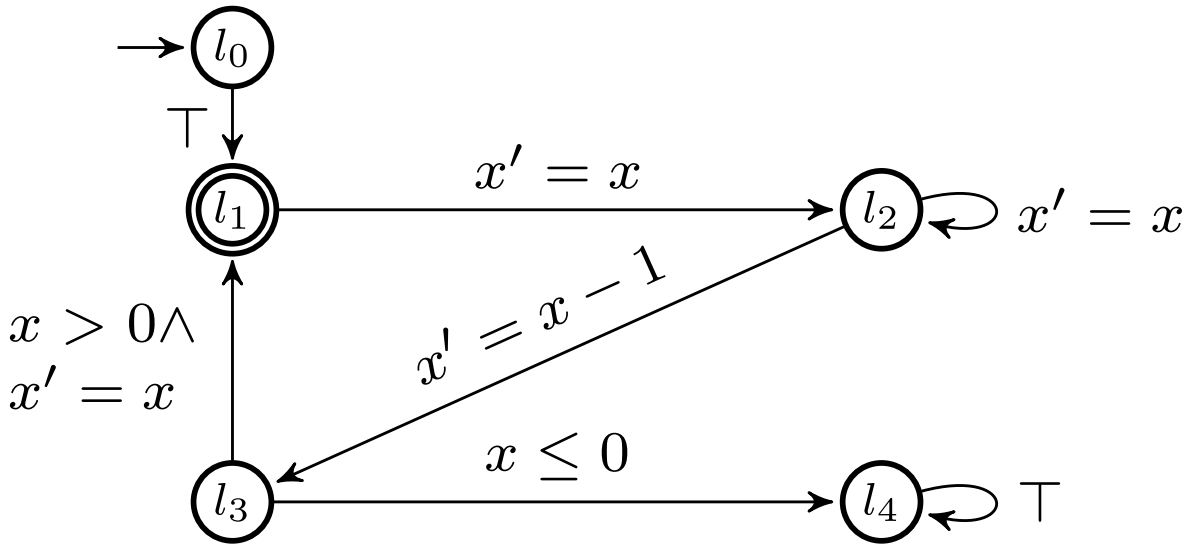
	f^0	a^0	w^1_{1-p}	f^1	a^1	w^2_{1-p}	...
l_1	$\top$	$\top$	$x \leq 1$	$x > 1$	$x > 1$	$x \leq 2$	...
l_2	$\bot$	$x > 1$	$x \leq 1$	$\bot$	$x > 2$	$x \leq 2$	...
l_3	$\bot$	$x > 0$	$x \leq 1$	$\bot$	$x > 1$	$x \leq 2$	...
l_4	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	...

Spec: visit location l_1 infinitely often

The environment wins the game for every possible state,

since the value of x chosen in l_0 bounds the number of visits to l_1.

Solving Büchi Games



	f^0	a^0	w^1_{1-p}	f^1	a^1	w^2_{1-p}	...
l_1	$\top$	$\top$	$x \leq 1$	$x > 1$	$x > 1$	$x \leq 2$	...
l_2	$\bot$	$x > 1$	$x \leq 1$	$\bot$	$x > 2$	$x \leq 2$	...
l_3	$\bot$	$x > 0$	$x \leq 1$	$\bot$	$x > 1$	$x \leq 2$	...
l_4	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	...

Spec: visit location l_1 infinitely often

Does not terminate ...

The environment wins the game for every possible state,

since the value of x chosen in l_0 bounds the number of visits to l_1.

Lectures Outline

- Specification formalisms for infinite-state reactive systems
- Main approaches to realizability checking and synthesis
- Naive symbolic methods and their limitations

- **Symbolic methods enhanced with logical reasoning**
  - **for solving realizability/synthesis games**
  - **for translation of temporal logic formulas**
  - **combining symbolic methods and abstraction**

References (Part 1)

[Green, IJCAI 1969]

C. Green. **Application of Theorem Proving to Problem Solving**. IJCAI 1969

[Manna/Waldinger, IEEE Transactions on Software Engineering, 1979]

Z. Manna and R. Waldinger. **Synthesis: Dreams →Programs**. IEEE Transactions on Software Engineering, 1979

[Solar-Lezama/Tancau/Bodik/Seshia/Saraswat, ASPLOS 2006]

A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat. **Combinatorial Sketching for Finite programs**. ASPLOS 2006

[Gulwani, POPL 2011]

S. Gulwani. **Automating String Processing in Spreadsheets Using Input-Output Examples**. POPL 2011

References (Part 1)

[Church, Summaries of the Summer Institute of Symbolic Logic, 1957]

A. Church. **Application of Recursive Arithmetic to the Problem of Circuit Synthesis**. Summaries of the Summer Institute of Symbolic Logic, 1957

[Church, International Congress of Mathematicians, 1962]

A. Church. **Logic, Arithmetic, and Automata**. International Congress of Mathematicians, 1962

[Pnueli, Annual Symposium on Foundations of Computer Science, 1977]

A. Pnueli. **The Temporal Logic of Programs**. Annual Symposium on Foundations of Computer Science 1977

References (Part 1)

[Finkbeiner/Klein/Piskac/Santolucito, CAV 2019]

B. Finkbeiner, F. Klein, R. Piskac, M. Santolucito. **Temporal Stream Logic: Synthesis Beyond the Bools**. CAV 2019

[Finkbeiner/Heim/Passing, FoSSaCS 2022]

B. Finkbeiner, P. Heim, and N. Passing. **Temporal Stream Logic modulo Theories**. FoSSaCS 2022

[Rodríguez/Sánchez, CAV 2023]

A. Rodríguez and C. Sánchez. **Boolean Abstractions for Realizability Modulo Theories**. CAV 2023

[Heim/Dimitrova, POPL 2025]

P. Heim and R. Dimitrova. **Translation of Temporal Logic for Efficient Infinite-State Reactive Synthesis**. POPL 2025

References (Part 1)

[Heim/Dimitrova, POPL 2024]

P. Heim and R. Dimitrova. **Solving Infinite-State Games via Acceleration**. POPL 2024

[Azzopardi/Di Stefano/Piterman/Schneider, CAV 2025]

S. Azzopardi, L. D. Stefano, N. Piterman, and G. Schneider. **Full LTL Synthesis over Infinite-state Arenas**. CAV 2025

[Heim/Dimitrova, CAV 2025]

P. Heim and R. Dimitrova. **Issy: A Comprehensive Tool for Specification and Synthesis of Infinite-State Reactive Systems**. CAV 2025

References (Part 1)

[Wolper/Vardi/Sistla, Symposium on Foundations of Computer Science 1983]

P. Wolper, M. Y. Vardi and A. P. Sistla. **Reasoning about infinite computation paths**. Annual Symposium on Foundations of Computer Science 1983

[Landweber, Mathematical Systems Theory, 1969]

L.H. Landweber. **Decision problems for omega-automata**. Mathematical systems theory, 1969

References (Part 1)

[Maderbacher/Bloem, FMCAD 2022]

B. Maderbacher and R. Bloem. **Reactive Synthesis Modulo Theories using Abstraction Refinement**. FMCAD 2022

[Samuel/D'Souza/Komondoor, ASE 2023]

S. Samuel, D. D'Souza, and R. Komondoor. **Symbolic Fixpoint Algorithms for Logical LTL Games**. ASE 2023

[Faella/Parlato, AAAI 2023]

M. Faella and G. Parlato. **Reachability Games Modulo Theories with a Bounded Safety Player**. AAAI 2023

[Unno/Satake/Terauchi/Koskinen, POPL 2023]

H. Unno, T. Terauchi, Y. Gu, and E. Koskinen. **Modular Primal-Dual Fixpoint Logic Solving for Temporal Verification**. POPL 2023

Acknowledgement

Some of the slides in this lecture are by **Philippe Heim** (CISPA Helmholtz Center for Information Security)