Reactive Synthesis

Lecture 11

Swen Jacobs and Martin Zimmermann
(Saarland University)

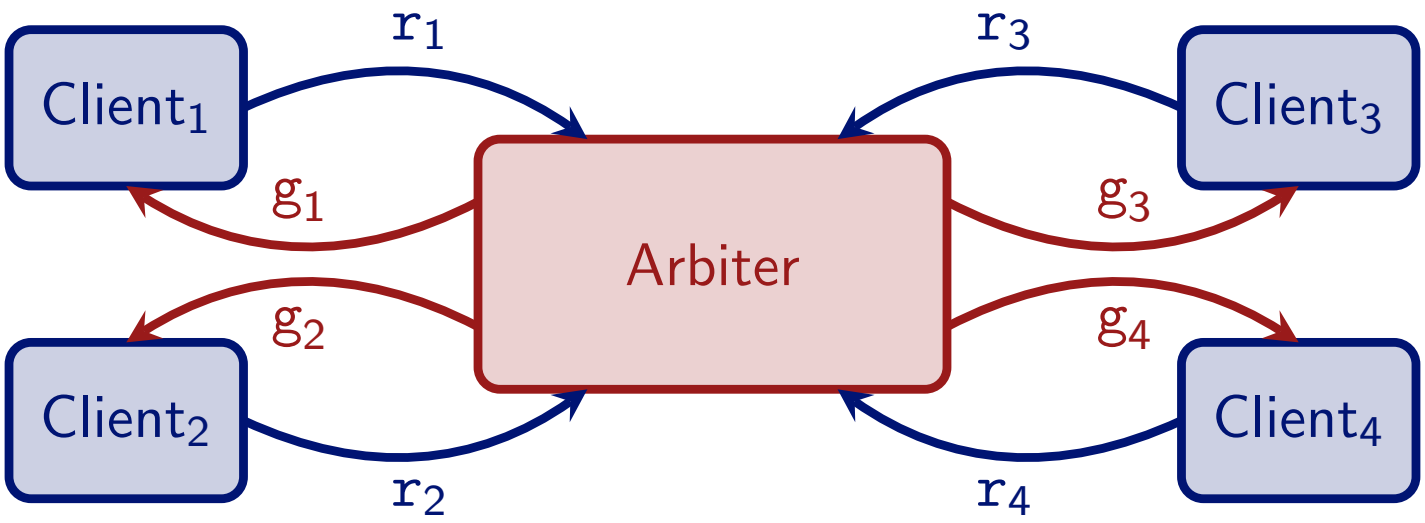
Plan for Today

- Basic Games
- Algorithms & Data Structures
- Advanced Games
- Temporal Logic Synthesis

Plan for Today

- Basic Games
- Algorithms & Data Structures
- Advanced Games
- Temporal Logic Synthesis
  - Solving the LTL Synthesis Problem:
    - Bounded Synthesis
    - Reducing Bounded Synthesis to SMT/QBF/SAT Solving

Recap: Motivating Example



Typical requirements on an arbiter:

- Every request of a client is eventually granted by the arbiter.
- Never two grants at the same time.
- No spurious grants, i.e., arbiter only gives grant to client i if it has an open request.
- A client may only pose a request if it has no open request.

Recap: Arbiter Specification in LTL

Example specifications

1. Answer every request: $\bigwedge_i \mathbf{G}(\mathtt{r}_i \rightarrow \mathbf{F}\, \mathtt{g}_i)$
2. At most one grant at a time: $\mathbf{G} \bigwedge_{i \neq j} \neg(\mathtt{g}_i \wedge \mathtt{g}_j)$
3. No spurious grants:

$$\bigwedge_i \neg[\,(\neg \mathtt{r}_i \mathbf{U} (\neg \mathtt{r}_i \wedge \mathtt{g}_i))\,] \wedge \neg[\,\mathbf{F}(\mathtt{g}_i \wedge \mathbf{X}(\neg \mathtt{r}_i \mathbf{U} (\neg \mathtt{r}_i \wedge \mathtt{g}_i)))\,]$$

4. No new requests while request is open:
   $\bigwedge_i \mathbf{G}(\mathtt{r}_i \rightarrow \mathbf{X}(\neg \mathtt{r}_i \,\mathbf{U}\, \mathtt{g}_i))$

Recap: LTL Games

Notation

Given a labeling $\ell\colon V \to 2^P$ of V by atomic propositions and a play $\rho = \rho_0\rho_1\rho_2\cdots$ define $\ell(\rho) = \ell(\rho_0)\ell(\rho_1)\ell(\rho_2)\cdots$.

Definition

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $\ell\colon V \to 2^P$ be a labeling of $\mathcal{A}$'s vertices by atomic propositions. Then, the LTL condition $\mathrm{LTL}(\varphi, \ell)$ is defined as

$$\mathrm{LTL}(\varphi, \ell) := \{\rho \in V^\omega \mid (\ell(\rho), 0) \models \varphi\}.$$

We call a game $\mathcal{G} = (\mathcal{A}, \mathrm{LTL}(\varphi, \ell))$ an *LTL game*.

Recap: Main Result

Theorem

LTL games are determined with finite-state strategies of doubly-exponential size (in the size of the formula) and can be solved in doubly-exponential time.

Remark

Both lower bounds are tight.

Recap: Main Result

Theorem

LTL games are determined with finite-state strategies of doubly-exponential size (in the size of the formula) and can be solved in doubly-exponential time.

Remark

Both lower bounds are tight.

Problem: size of game doubly exponential in specification, even if solution is very simple. (and simple solutions will be hard to find)

Bounded Synthesis

Synthesis:
Is there an implementation that satisfies the specification?

Bounded Synthesis

Synthesis:
Is there an implementation that satisfies the specification?

Bounded Synthesis:
Is there an implementation *with at most N states* that satisfies the specification?

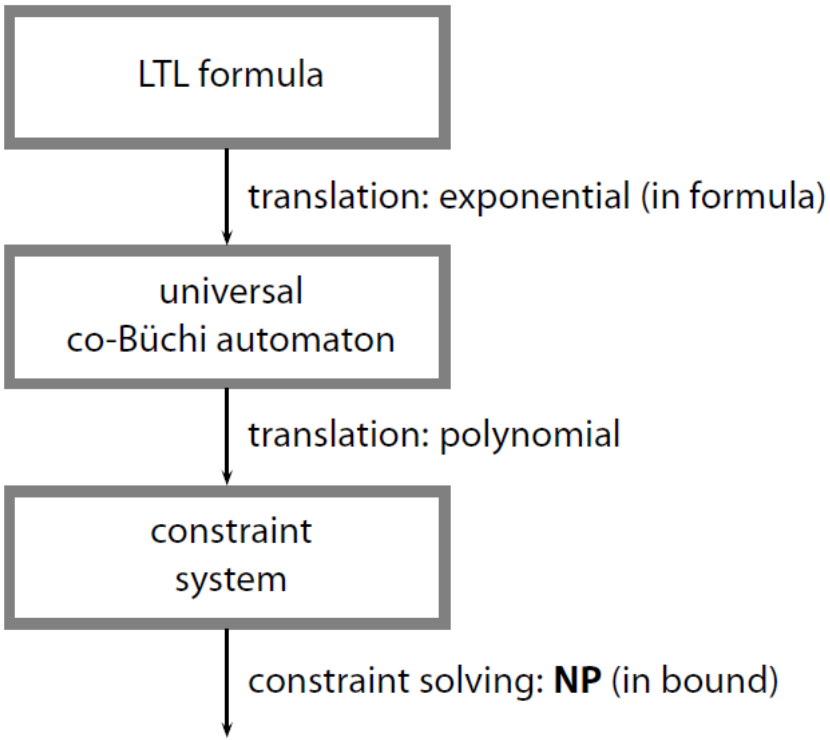
Bounded Synthesis

Synthesis:
Is there an implementation that satisfies the specification?

Bounded Synthesis:
Is there an implementation *with at most N states* that satisfies the specification?

Idea: Search for **simple** implementations first, find them **fast**.

Bounded LTL Synthesis



Input Complexity:
Exponential in the size of the specification.

Output Complexity:
NP-complete in the size of the implementation.

Labeled Transition Systems

Let I be a finite set of boolean input variables and O a finite set of boolean output variables.

Definition

A **labeled transition system** is a tuple $\mathcal{T} = (T, t_0, \tau, o)$, where

- T is a set of states,
- $t_0 \in T$ is the initial state,
- $\tau : T \times \mathbb{B}^I \to T$ is the transition function, and
- $o : T \to \mathbb{B}^O$ is the labeling function.

Labeled Transition Systems

Let I be a finite set of boolean input variables and O a finite set of boolean output variables.

Definition

A **labeled transition system** is a tuple $\mathcal{T} = (T, t_0, \tau, o)$, where

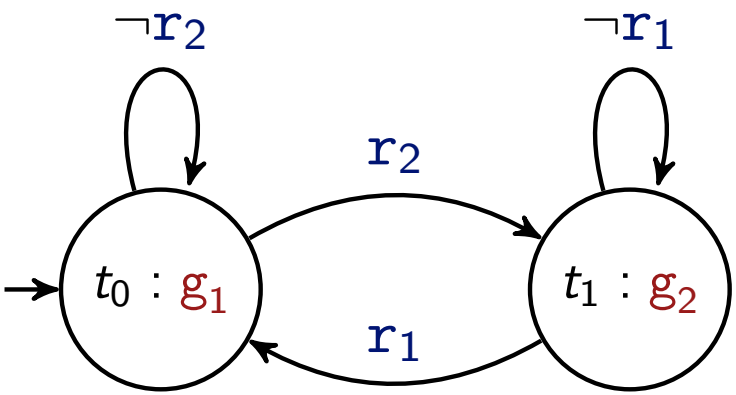
- T is a set of states,
- $t_0 \in T$ is the initial state,
- $\tau : T \times \mathbb{B}^I \to T$ is the transition function, and
- $o : T \to \mathbb{B}^O$ is the labeling function.

Connection to games:

Labeled transition systems correspond to finite-state strategies: choice of next state in game corresponds to choice of output values of transition system.
(like for safety games, but with an underlying memory structure)

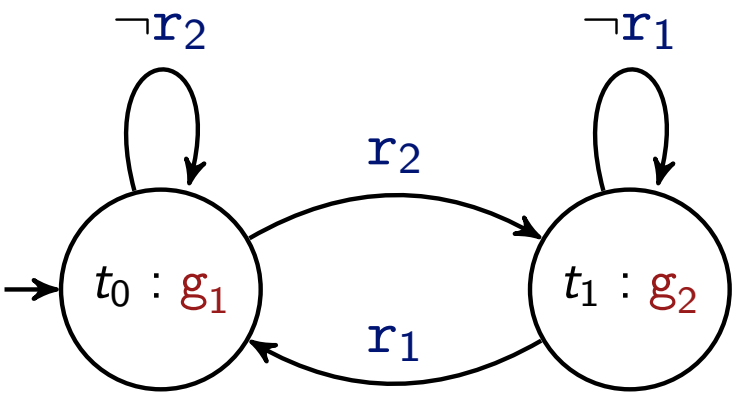
Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$



Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$



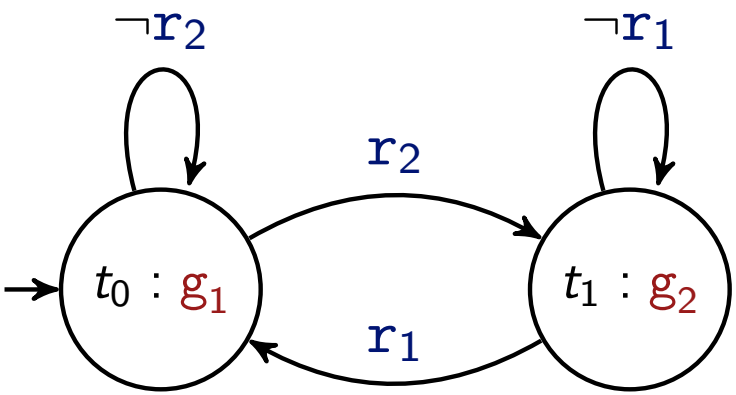
Example run

Env:

Sys:

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

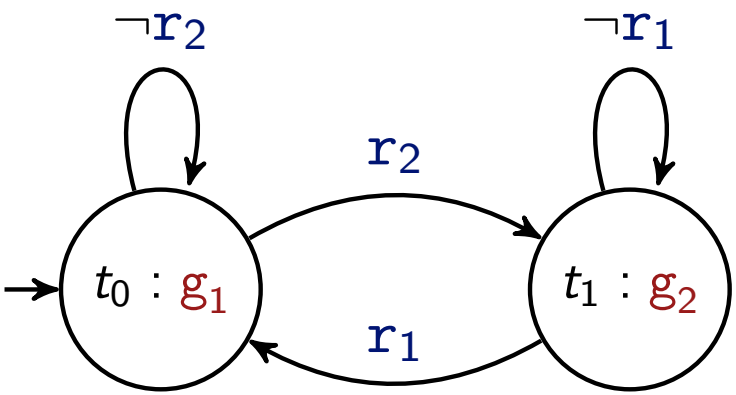


Example run

Env:
Sys: $\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

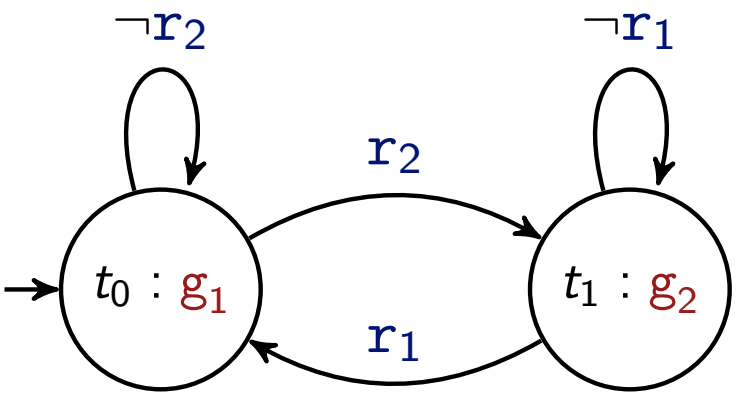


Example run

Env: $\mathtt{r}_1, \mathtt{r}_2$
Sys: $\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

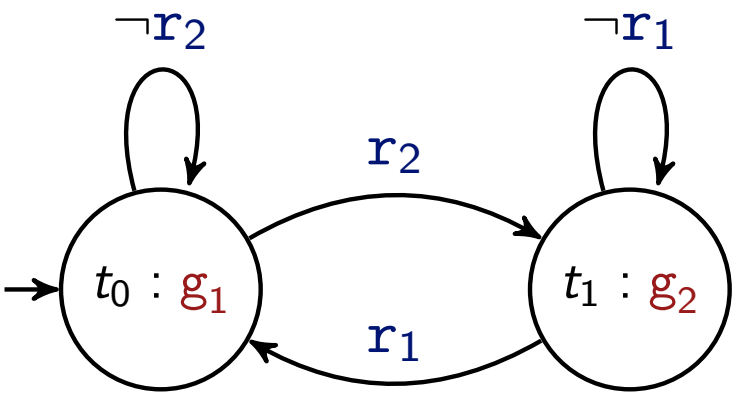


Example run

Env: $\mathtt{r}_1, \mathtt{r}_2$
Sys: $\mathtt{g}_1$ $\mathtt{g}_2$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

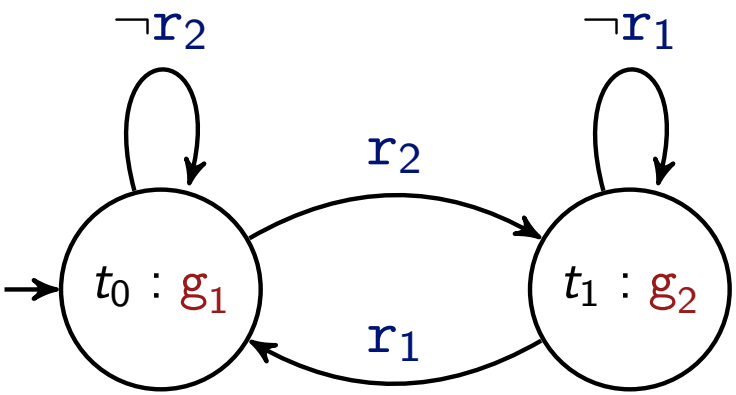


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

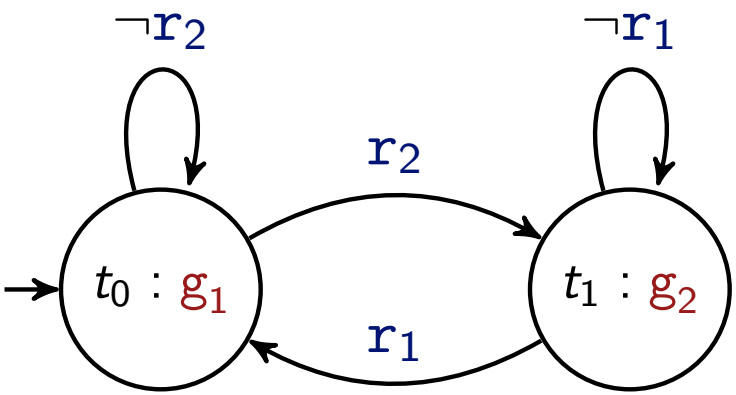


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

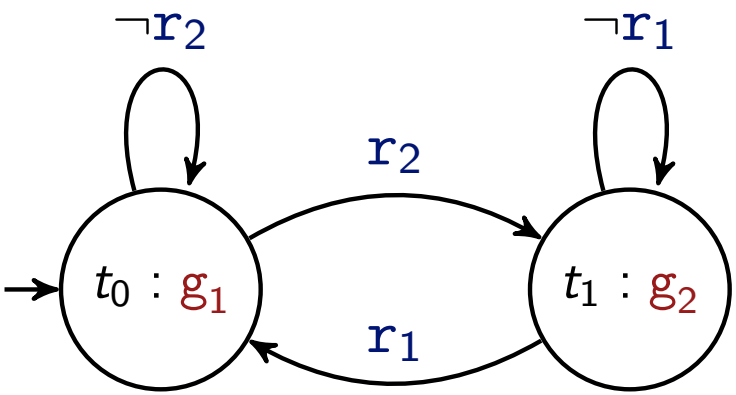


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

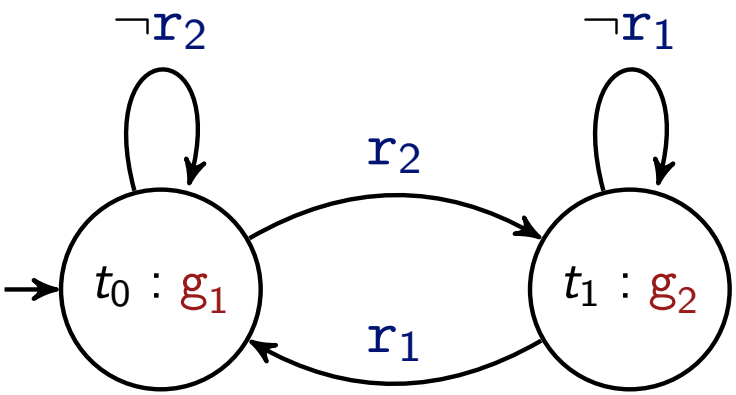


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

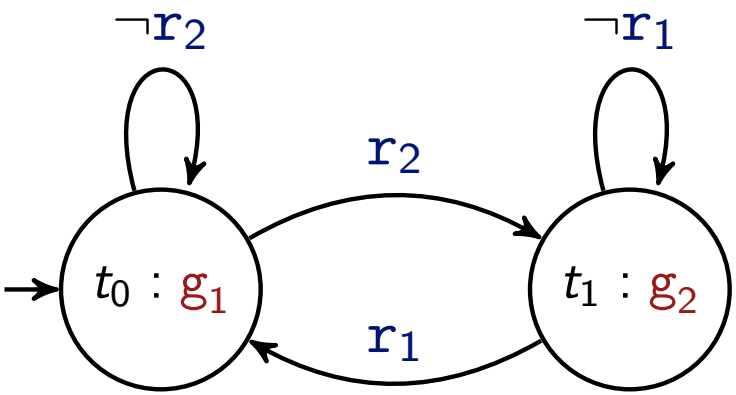


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	–
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

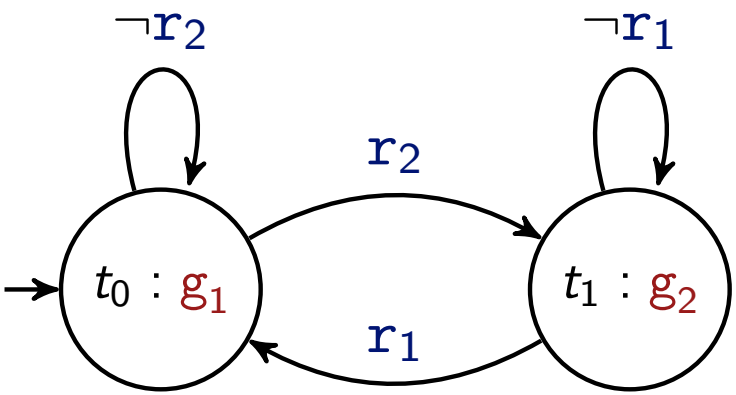


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	$-$	
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

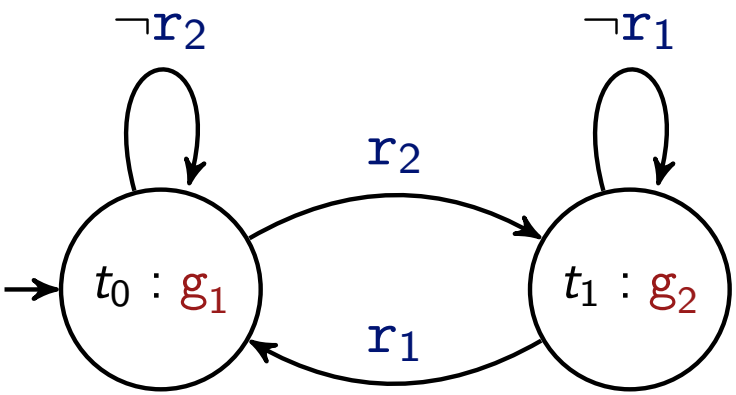


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	–	–
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

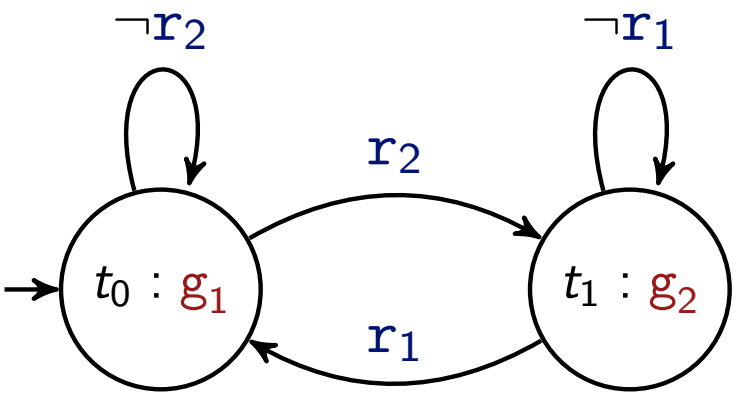


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$		$\mathtt{r}_1$	$\mathtt{r}_1$	–	–
Sys:		$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

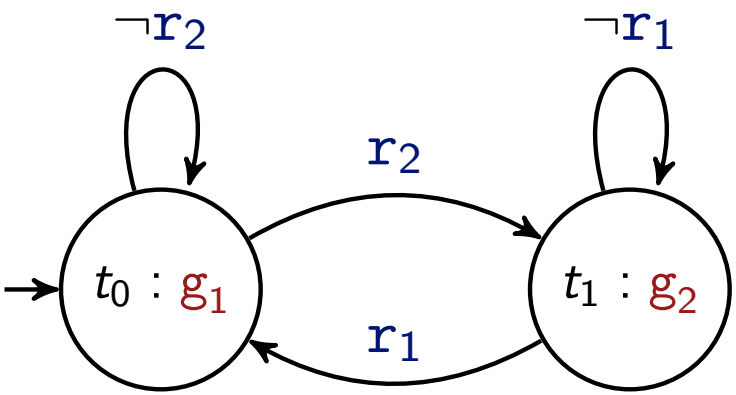


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	–	–	$\mathtt{r}_2$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

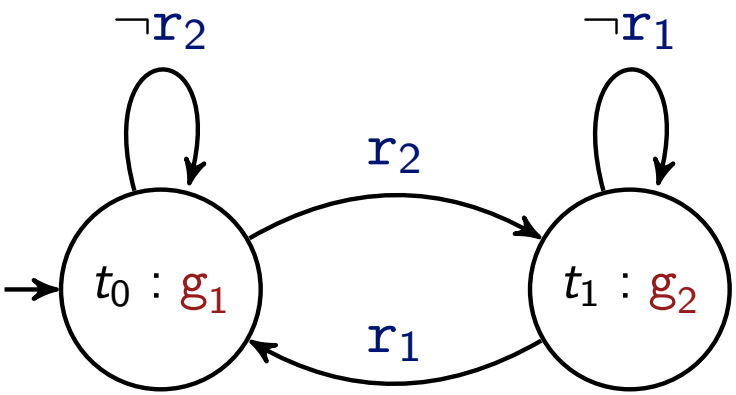


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$		$\mathtt{r}_1$	$\mathtt{r}_1$	–	–	$\mathtt{r}_2$	
Sys:		$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_2$

Example: Labeled Transition System

Input variables: $I = \{r_1, r_2\}$
Output variables: $O = \{g_1, g_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{g_1, \neg g_2\}, o(t_1) = \{\neg g_1, g_2\}$

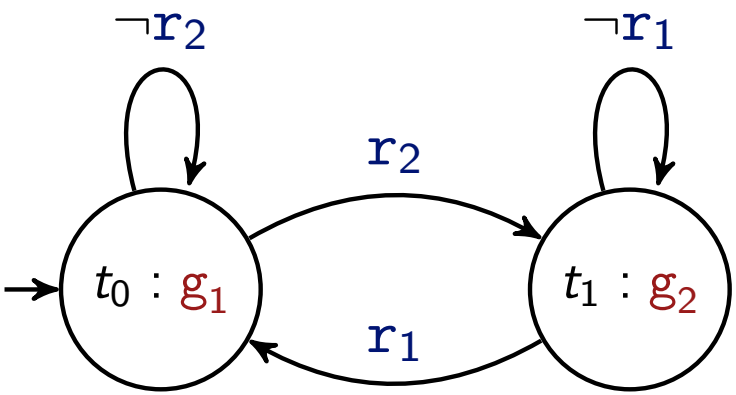


Example run

Env:	r_1, r_2	r_1	r_1	–	–	r_2	–
Sys:	g_1	g_2	g_1	g_1	g_1	g_1	g_2

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

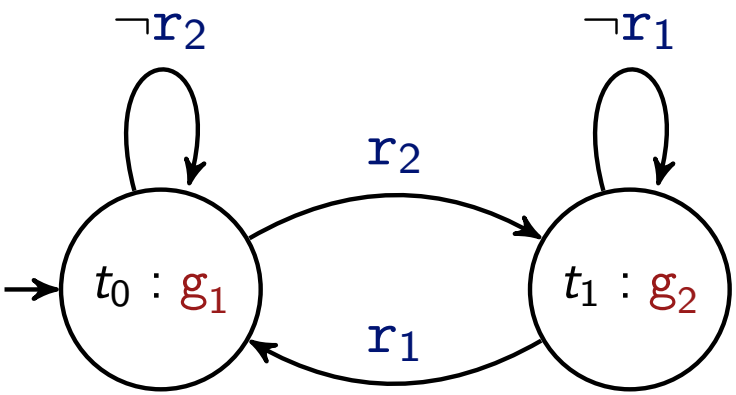


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$		$\mathtt{r}_1$	$\mathtt{r}_1$	–	–	$\mathtt{r}_2$	–
Sys:		$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_2$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

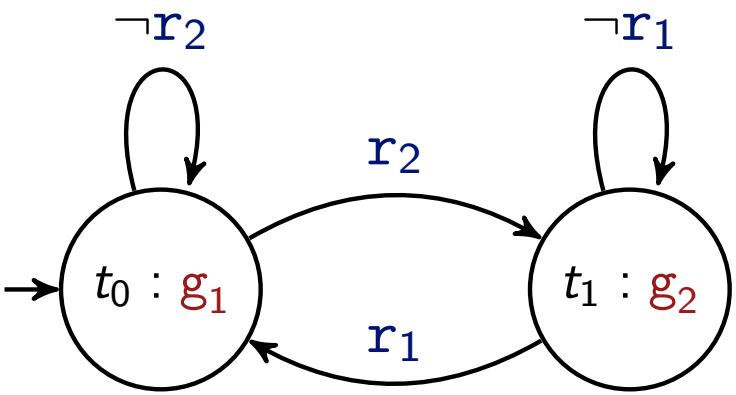


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	$-$	$-$	$\mathtt{r}_2$	$-$	$\mathtt{r}_1, \mathtt{r}_2$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_2$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$

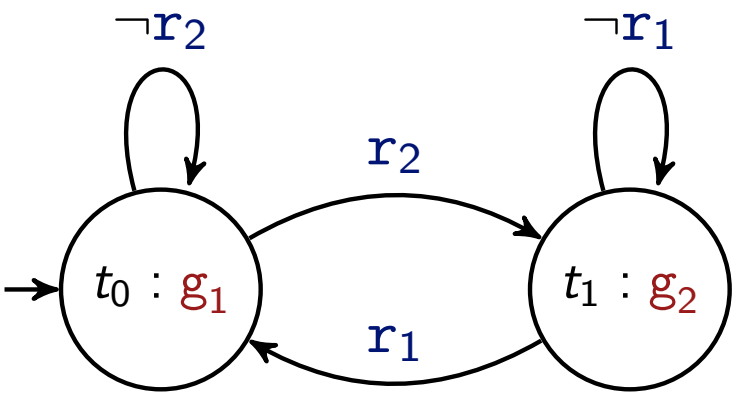


Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	$-$	$-$	$\mathtt{r}_2$	$-$	$\mathtt{r}_1, \mathtt{r}_2$	
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_2$	$\mathtt{g}_1$

Example: Labeled Transition System

Input variables: $I = \{\mathtt{r}_1, \mathtt{r}_2\}$
Output variables: $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
States: $T = \{t_0, t_1\}$
Labeling: $o(t_0) = \{\mathtt{g}_1, \neg\mathtt{g}_2\}, o(t_1) = \{\neg\mathtt{g}_1, \mathtt{g}_2\}$



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1$	–	–	$\mathtt{r}_2$	–	$\mathtt{r}_1, \mathtt{r}_2$	–
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_2$	$\mathtt{g}_1$

Universal Co-Büchi Automata

Let I and O be finite sets of boolean variables. We will run the UCB on an LTS, so O are inputs and I are outputs for the UCB.

Definition

A **universal co-Büchi automaton (UCB)** is a tuple $\mathcal{U} = (Q, q_0, \delta, F)$, where

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\delta : Q \times \mathbb{B}^O \to \mathbb{B}^{(Q \times \mathbb{B}^I)}$ is the transition function (that maps a state and an input to a set of tuples of successor states and outputs), and
- $F \subseteq Q$ is a set of rejecting states.

Universal Co-Büchi Automata

Let I and O be finite sets of boolean variables. We will run the UCB on an LTS, so O are inputs and I are outputs for the UCB.

Definition

A **universal co-Büchi automaton (UCB)** is a tuple $\mathcal{U} = (Q, q_0, \delta, F)$, where

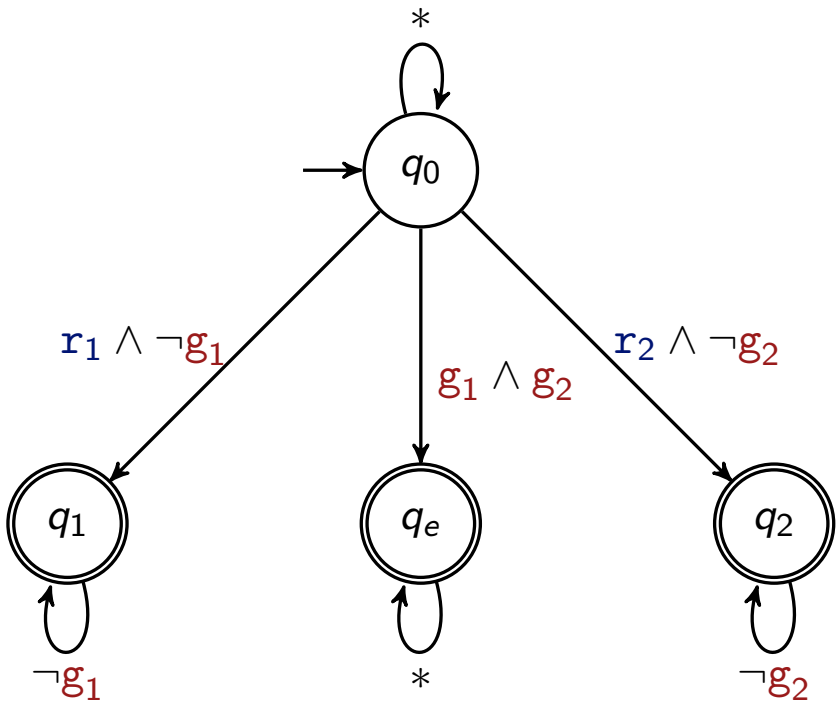
- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\delta : Q \times \mathbb{B}^O \to \mathbb{B}^{(Q \times \mathbb{B}^I)}$ is the transition function (that maps a state and an input to a set of tuples of successor states and outputs), and
- $F \subseteq Q$ is a set of rejecting states.

Example: Universal Co-Büchi Automaton

UCB for basic arbiter specification (with two clients):

$\bigwedge_{i \in \{1,2\}} \mathbf{G}(\mathtt{r}_i \rightarrow \mathbf{F}\, \mathtt{g}_i)$
$\mathbf{G}\, \neg(\mathtt{g}_1 \wedge \mathtt{g}_2)$

Input variables $O = \{\mathtt{g}_1, \mathtt{g}_2\}$
Output variables $I = \{\mathtt{r}_1, \mathtt{r}_2\}$



From LTL to UCB

To obtain a UCB that accepts those LTSs that satisfy a specification φ, do the following:

1. Construct the non-deterministic Büchi automaton $\mathcal{B} = (Q, \Sigma, q_0, \Delta, F)$ for $\neg\varphi$, with $\Sigma = \mathbb{B}^{I \cup O}$.
2. The desired UCB is $\mathcal{U} = (Q, q_0, \delta, F)$, where δ is defined by: $(q', i) \in \delta(q, o)$ if and only if $(q, \sigma, q') \in \Delta$ and $\sigma \cap I = i$ and $\sigma \cap O = o$.

Run Graph, Acceptance of an LTS

Definition

A **run graph** of a UCB $\mathcal{U} = (Q, q_0, \delta, F)$ on a LTS $\mathcal{T} = (T, t_0, \tau, o)$ is the directed graph $G = (V, E)$ that satisfies:

- $V \subseteq Q \times T$,
- $(q_0, t_0) \in V$, and
- if $(q, t) \in V$, then $((q, t), (q', \tau(t, i))) \in E$ if and only if $(q', i) \in \delta(q, o(t))$.

Run Graph, Acceptance of an LTS

Definition

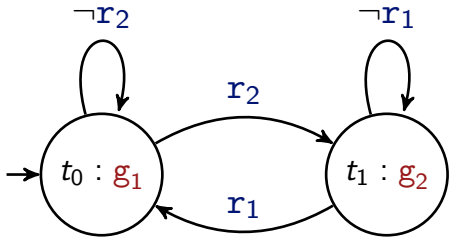
A **run graph** of a UCB $\mathcal{U} = (Q, q_0, \delta, F)$ on a LTS $\mathcal{T} = (T, t_0, \tau, o)$ is the directed graph $G = (V, E)$ that satisfies:

- $V \subseteq Q \times T$,
- $(q_0, t_0) \in V$, and
- if $(q, t) \in V$, then $((q, t), (q', \tau(t, i))) \in E$ if and only if $(q', i) \in \delta(q, o(t))$.

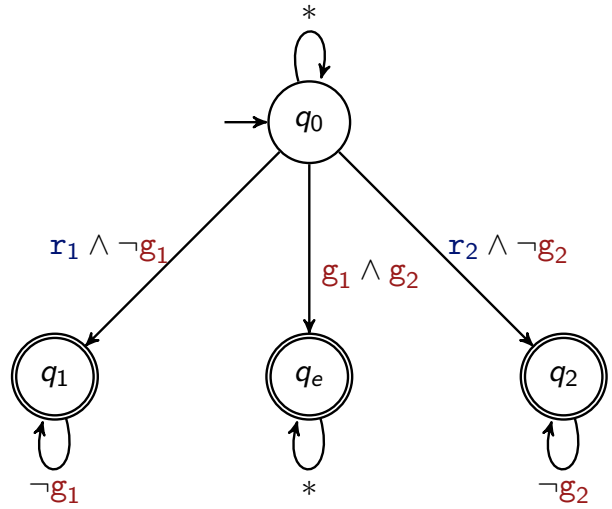
$\mathcal{T}$ is **accepted** by $\mathcal{U}$ if every infinite path in the run graph visits states in F only finitely often.

Example: Run Graph

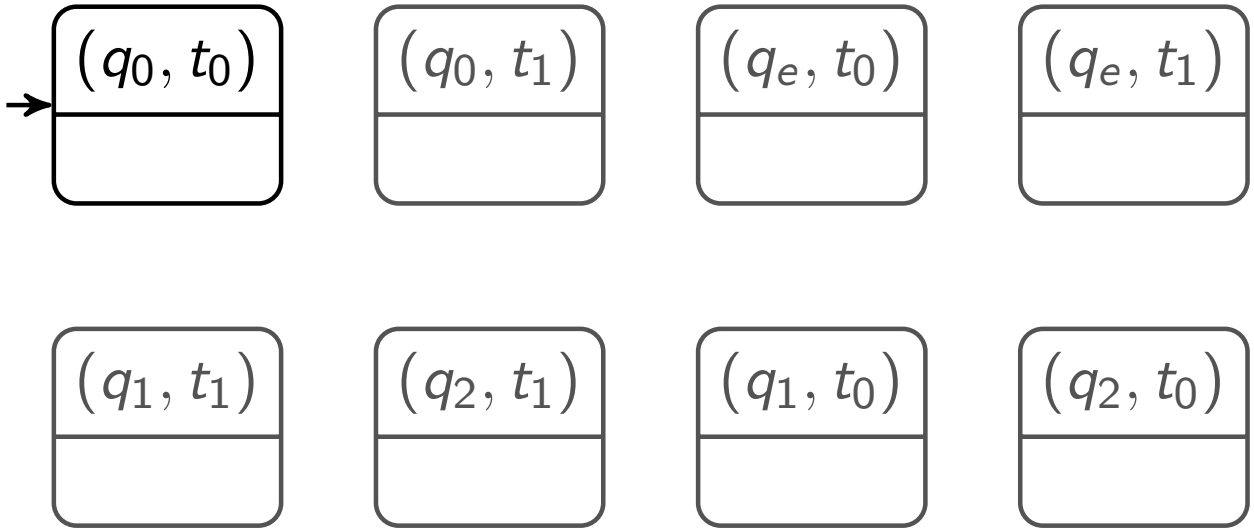
LTS:



UCB:

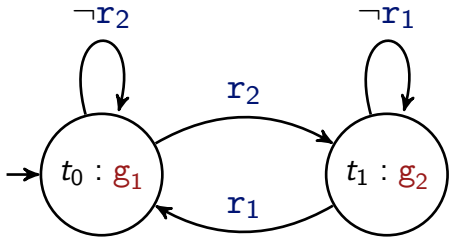


Run graph:

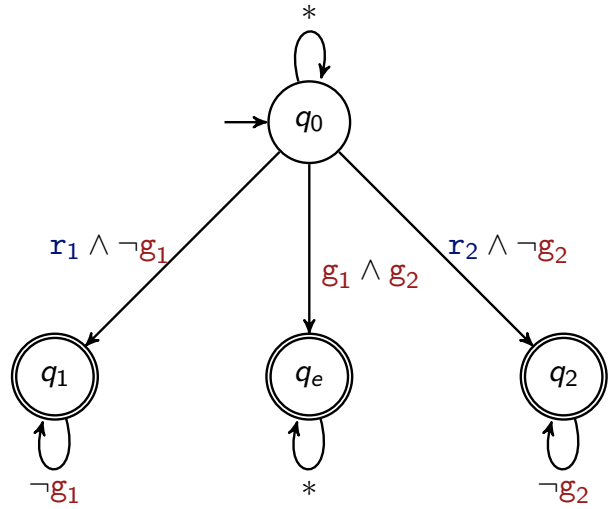


Example: Run Graph

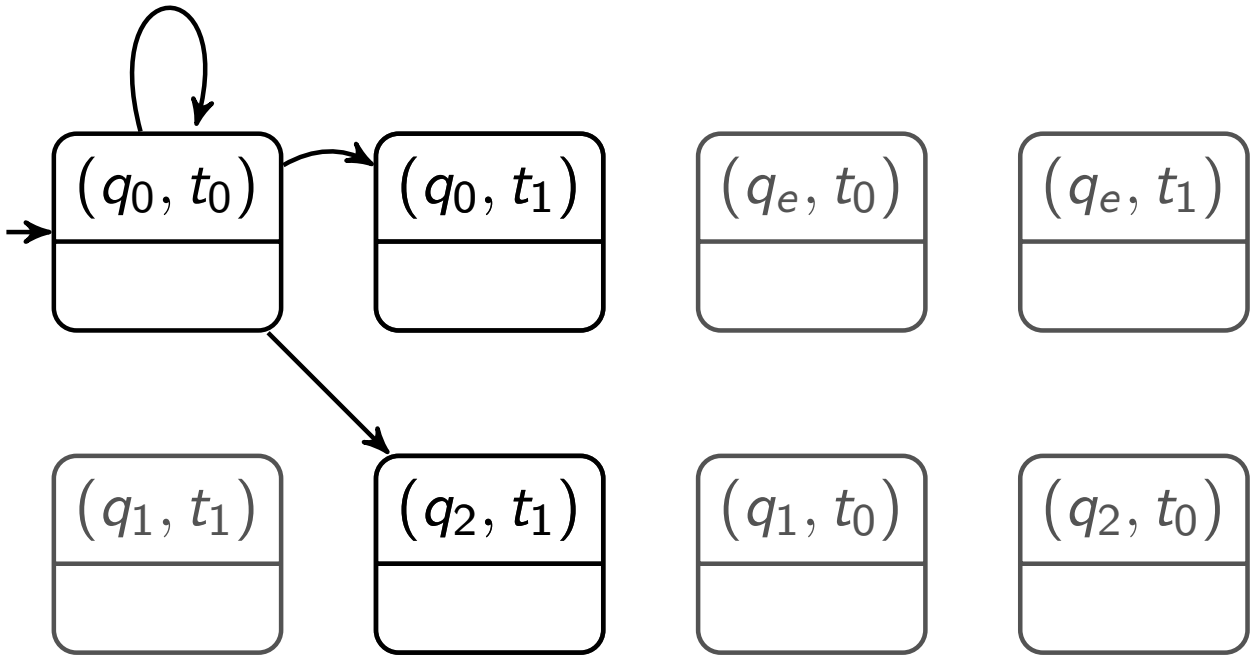
LTS:



UCB:

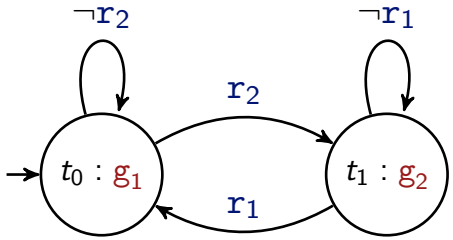


Run graph:

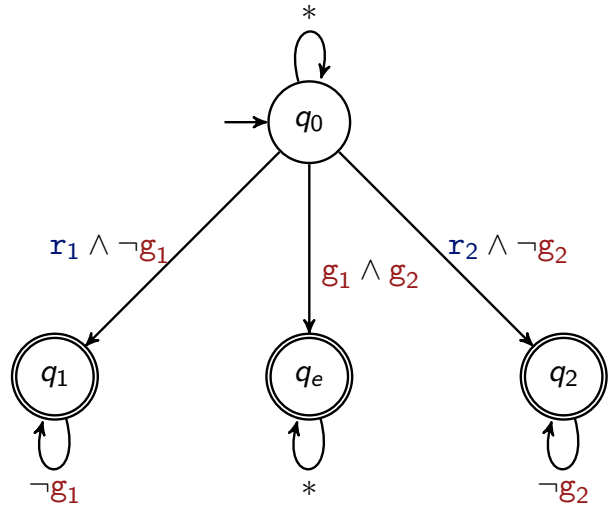


Example: Run Graph

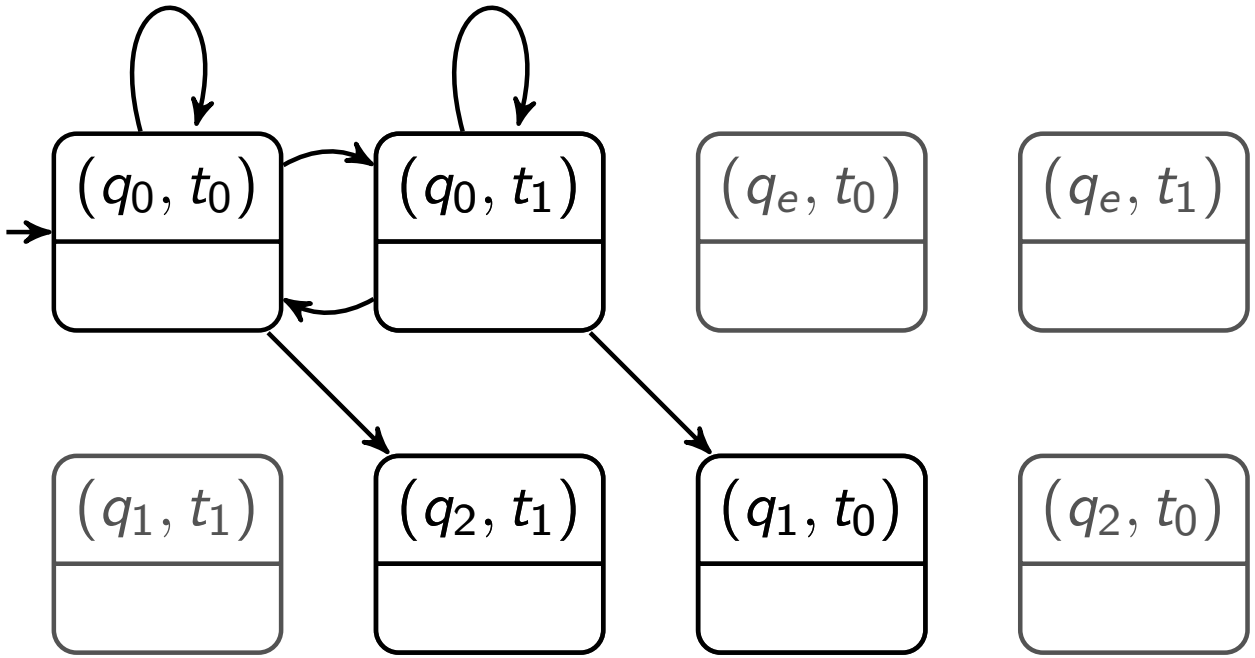
LTS:



UCB:



Run graph:



Valid Annotations for Run Graphs

Definition

For given UCB $\mathcal{U} = (Q, q_0, \delta, F)$ and LTS $\mathcal{T} = (T, t_0, \tau, o)$, an **annotation** is a function $\lambda : Q \times T \to \{\bot\} \cup \mathbb{N}$.

The annotation is **valid** if it satisfies:

1. $\lambda(q_0, t_0) \neq \bot$

Valid Annotations for Run Graphs

Definition

For given UCB $\mathcal{U} = (Q, q_0, \delta, F)$ and LTS $\mathcal{T} = (T, t_0, \tau, o)$, an **annotation** is a function $\lambda : Q \times T \to \{\bot\} \cup \mathbb{N}$.

The annotation is **valid** if it satisfies:

1. $\lambda(q_0, t_0) \neq \bot$
2. if $\lambda(q, t) \neq \bot$ and (q', i) satisfies $\delta(q, o(t))$, then $\lambda(q', \tau(t, i)) \geq \lambda(q, t)$

Valid Annotations for Run Graphs

Definition

For given UCB $\mathcal{U} = (Q, q_0, \delta, F)$ and LTS $\mathcal{T} = (T, t_0, \tau, o)$, an **annotation** is a function $\lambda : Q \times T \to \{\bot\} \cup \mathbb{N}$.

The annotation is **valid** if it satisfies:

1. $\lambda(q_0, t_0) \neq \bot$
2. if $\lambda(q, t) \neq \bot$ and (q', i) satisfies $\delta(q, o(t))$, then $\lambda(q', \tau(t, i)) \geq \lambda(q, t)$
3. if the previous condition holds and additionally $q' \in F$, then $\lambda(q', \tau(t, i)) > \lambda(q, t)$

Valid Annotations for Run Graphs

Definition

For given UCB $\mathcal{U} = (Q, q_0, \delta, F)$ and LTS $\mathcal{T} = (T, t_0, \tau, o)$, an **annotation** is a function $\lambda : Q \times T \to \{\bot\} \cup \mathbb{N}$.

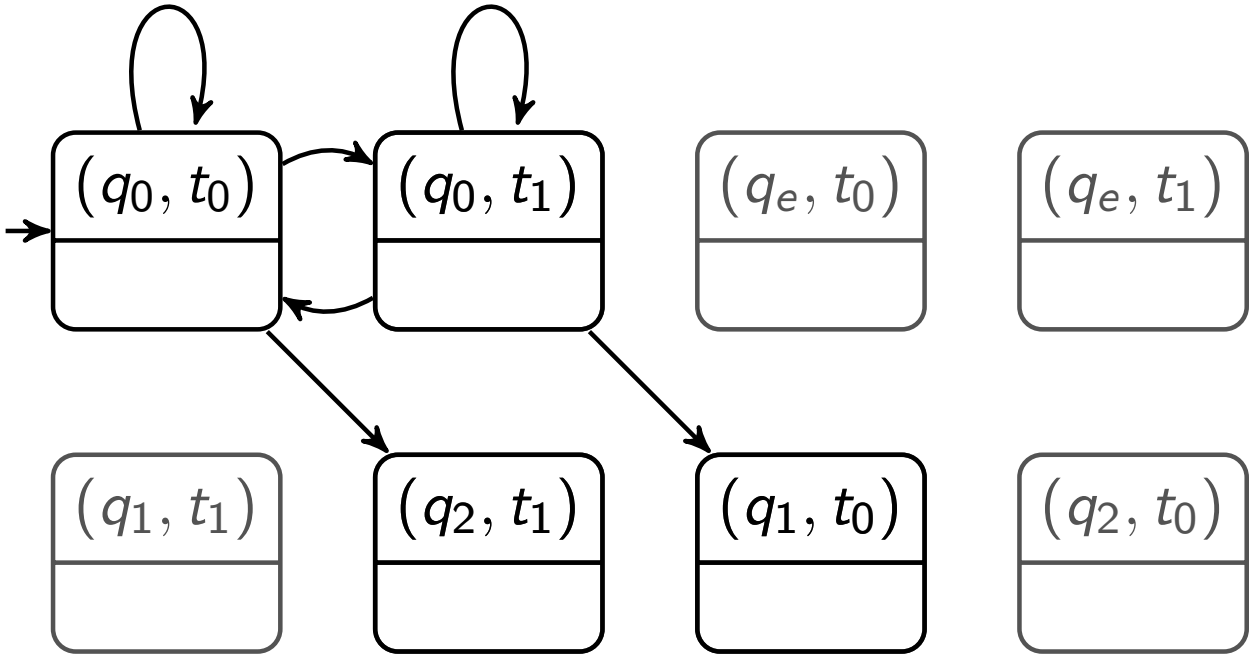
The annotation is **valid** if it satisfies:

1. $\lambda(q_0, t_0) \neq \bot$
2. if $\lambda(q, t) \neq \bot$ and (q', i) satisfies $\delta(q, o(t))$, then $\lambda(q', \tau(t, i)) \geq \lambda(q, t)$
3. if the previous condition holds and additionally $q' \in F$, then $\lambda(q', \tau(t, i)) > \lambda(q, t)$

An annotation is **c-bounded** if $\lambda(q, t) \leq c$ for all q, t (assuming $\bot < c$). An annotation is **bounded** if it is c-bounded for some c.

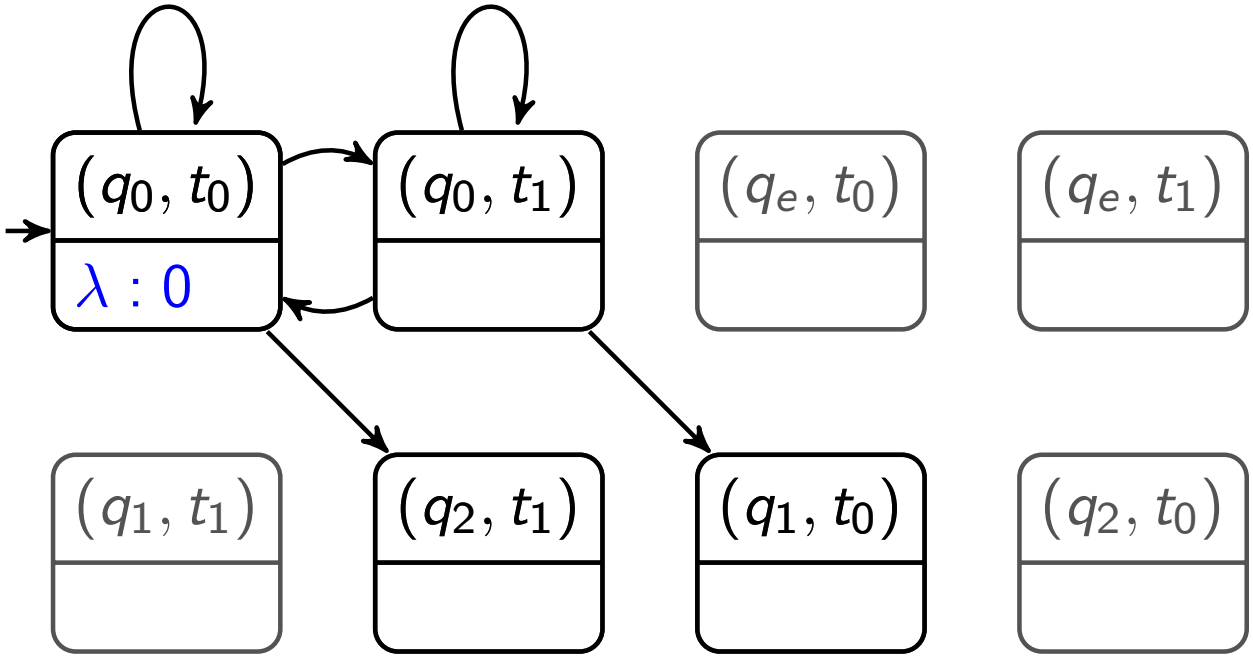
Example: Annotated Run Graph

Run graph with valid annotation λ:



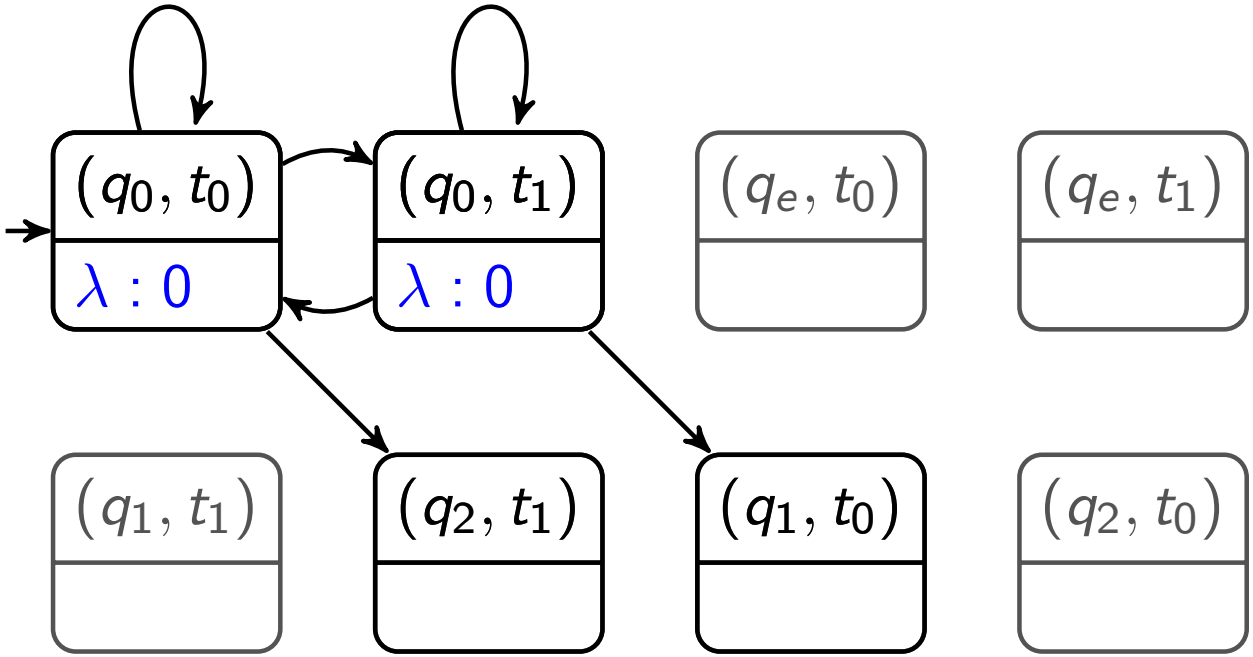
Example: Annotated Run Graph

Run graph with valid annotation λ:



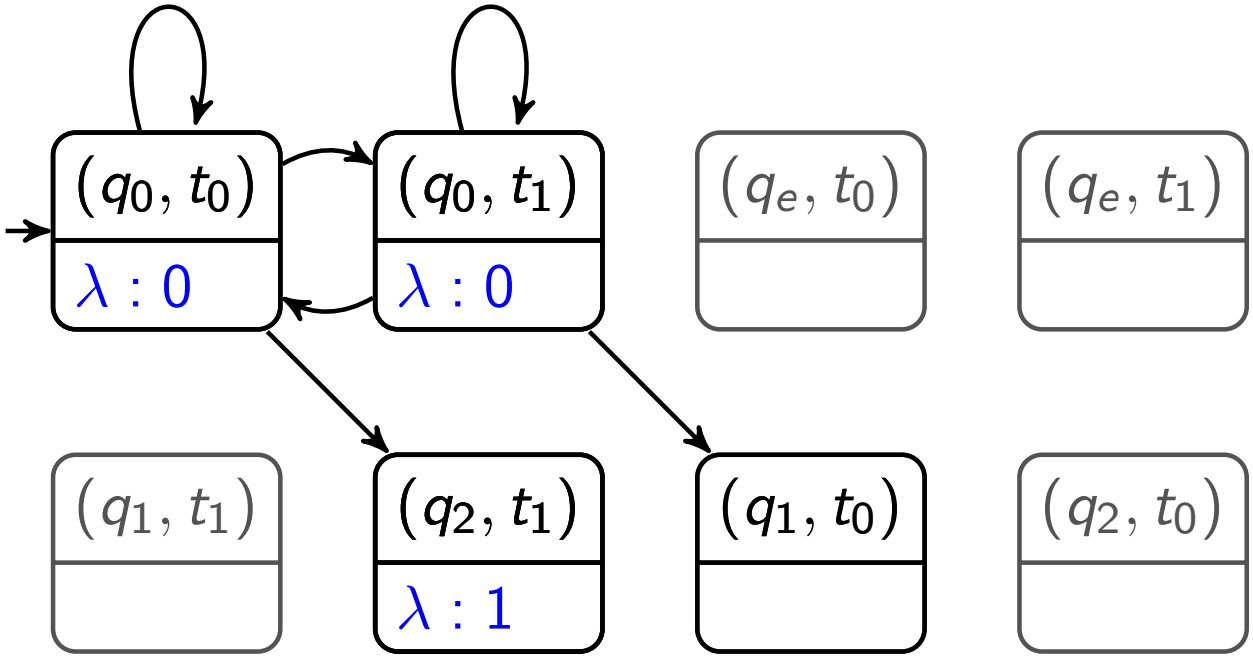
Example: Annotated Run Graph

Run graph with valid annotation λ:



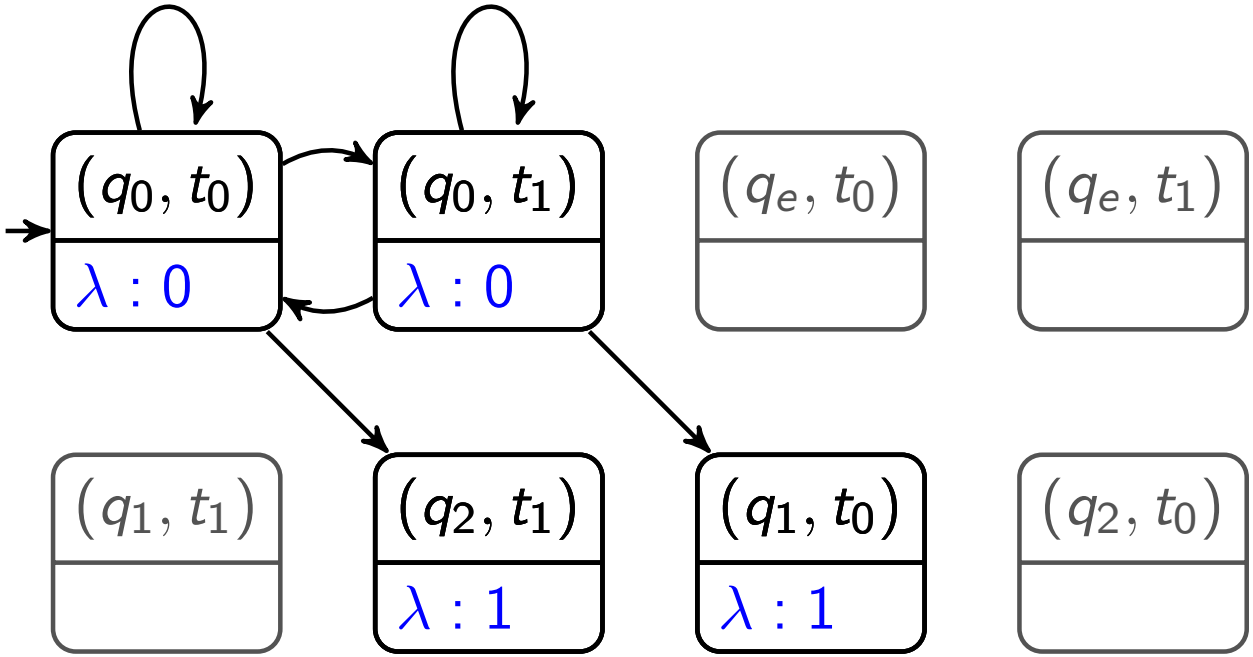
Example: Annotated Run Graph

Run graph with valid annotation λ:



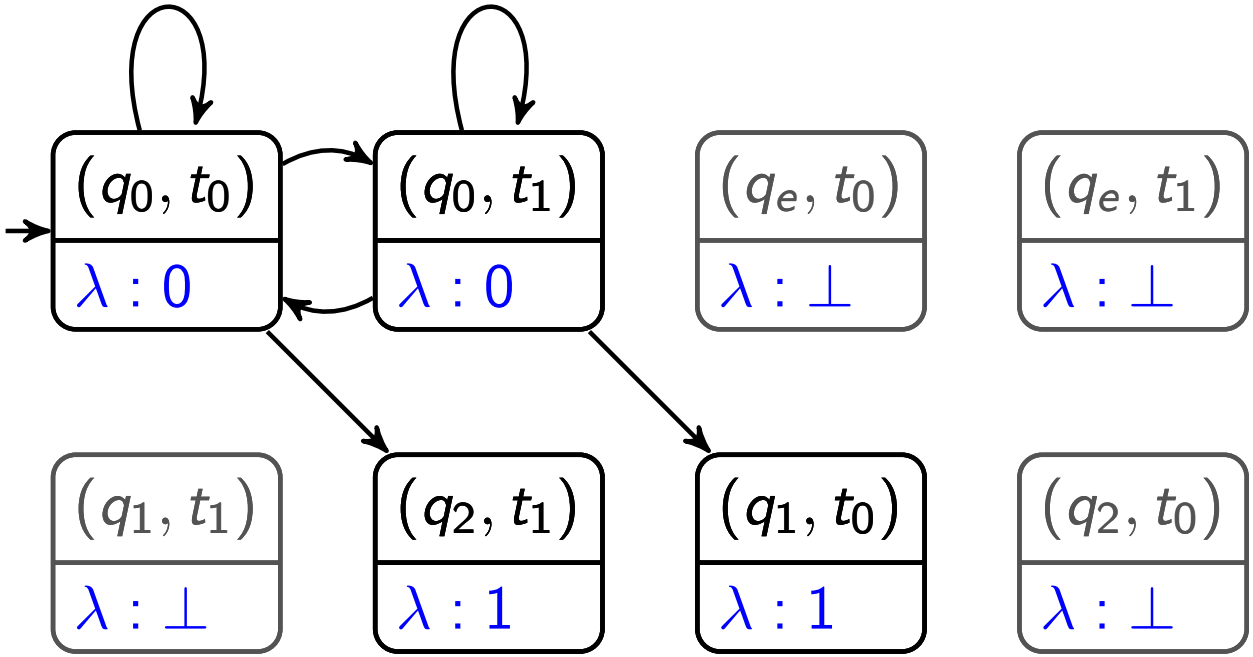
Example: Annotated Run Graph

Run graph with valid annotation λ:



Example: Annotated Run Graph

Run graph with valid annotation λ:



Bounded Valid Annotations and Acceptance

Theorem

A finite-state LTS $\mathcal{T} = (T, t_0, \tau, o)$ *is accepted by a UCB* $\mathcal{U} = (Q, q_0, \delta, F)$ *if and only if it has a valid* $(|T| \cdot |F|)$*-bounded annotation.*

Bounded Valid Annotations and Acceptance

Theorem

A finite-state LTS $\mathcal{T} = (T, t_0, \tau, o)$ *is accepted by a UCB* $\mathcal{U} = (Q, q_0, \delta, F)$ *if and only if it has a valid* $(|T| \cdot |F|)$*-bounded annotation.*

Proof idea: Let G be the run graph of $\mathcal{T}$ and $\mathcal{U}$.

- if G has a lasso with a rejecting state in its loop, then there cannot be a valid bounded annotation (since at some point the value would have to be strictly smaller than itself)

Bounded Valid Annotations and Acceptance

Theorem

A finite-state LTS $\mathcal{T} = (T, t_0, \tau, o)$ is accepted by a UCB $\mathcal{U} = (Q, q_0, \delta, F)$ if and only if it has a valid $(|T| \cdot |F|)$-bounded annotation.

Proof idea: Let G be the run graph of $\mathcal{T}$ and $\mathcal{U}$.

- if G has a lasso with a rejecting state in its loop, then there cannot be a valid bounded annotation (since at some point the value would have to be strictly smaller than itself)
- if G does not have such a lasso, then assign to each $(q, t) \in G$ the highest number of rejecting states that occur on some path from (q_0, t_0), and assign $\bot$ to unreachable vertices. The assigned values will be smaller than $(|T| \cdot |F|)$ (otherwise we would have a loop).

Existence of Bounded Valid Annotations

Definition

An LTS $\mathcal{T} = (T, t_0, \tau, o)$ with output variables O and input variables $I \subseteq O$ is **input-preserving** if for all $t \in T$ and inputs $i \in \mathbb{B}^I$: $o(\tau(t, i)) \cap I = i$.

Existence of Bounded Valid Annotations

Definition

An LTS $\mathcal{T} = (T, t_0, \tau, o)$ with output variables O and input variables $I \subseteq O$ is **input-preserving** if for all $t \in T$ and inputs $i \in \mathbb{B}^I$: $o(\tau(t, i)) \cap I = i$.

Theorem

If a UCB $\mathcal{U} = (Q, q_0, \delta, F)$ accepts an input-preserving transition system, then $\mathcal{U}$ accepts a finite input-preserving transition system $\mathcal{T}$ with $|Q|!^2 \cdot 2^I$ states, and $\mathcal{T}$ has a valid $(|F| \cdot |Q|!^2 \cdot 2^I)$-bounded annotation for $\mathcal{U}$.

Existence of Bounded Valid Annotations

Definition

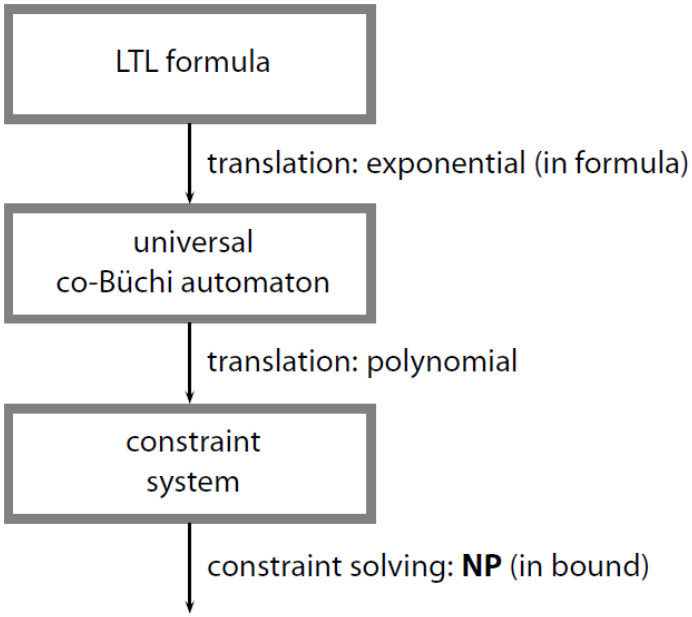
An LTS $\mathcal{T} = (T, t_0, \tau, o)$ with output variables O and input variables $I \subseteq O$ is **input-preserving** if for all $t \in T$ and inputs $i \in \mathbb{B}^I$: $o(\tau(t, i)) \cap I = i$.

Theorem

If a UCB $\mathcal{U} = (Q, q_0, \delta, F)$ accepts an input-preserving transition system, then $\mathcal{U}$ accepts a finite input-preserving transition system $\mathcal{T}$ with $|Q|!^2 \cdot 2^I$ states, and $\mathcal{T}$ has a valid $(|F| \cdot |Q|!^2 \cdot 2^I)$-bounded annotation for $\mathcal{U}$.

How to find such an accepted LTS?

Synthesis of Systems with Bounded Size



Idea: Fix bound c, search for LTS with at most c states and a c-bounded valid annotation.

Approach: Define **constraint system** that specifies existence of LTS and annotation, use a **constraint solver** to find a solution.

Encoding Bounded Synthesis

To encode the transition system, introduce:

- a transition function $\tau : T \times 2^I \to T$
- for each output variable $a \in O$, a function $a : T \to \mathbb{B}$

Encoding Bounded Synthesis

To encode the transition system, introduce:

- a transition function $\tau : T \times 2^I \to T$
- for each output variable $a \in O$, a function $a : T \to \mathbb{B}$

To encode the annotation, introduce for every state $q \in Q$:

- a function $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$
  (Idea: "is (q, t) in run graph?")

Encoding Bounded Synthesis

To encode the transition system, introduce:

- a transition function $\tau : T \times 2^I \to T$
- for each output variable $a \in O$, a function $a : T \to \mathbb{B}$

To encode the annotation, introduce for every state $q \in Q$:

- a function $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$
  (Idea: "is (q, t) in run graph?")
- a function $\lambda_q^{\#} : T \to \{\bot, 0, \ldots, c\}$
  (Idea: "how many rejecting states on path to (q, t)?")

Encoding Bounded Synthesis

To encode the transition system, introduce:

- a transition function $\tau : T \times 2^I \to T$
- for each output variable $a \in O$, a function $a : T \to \mathbb{B}$

To encode the annotation, introduce for every state $q \in Q$:

- a function $\lambda_q^{\mathbb{B}} : T \to \mathbb{B}$
  (Idea: "is (q, t) in run graph?")
- a function $\lambda_q^{\#} : T \to \{\bot, 0, \ldots, c\}$
  (Idea: "how many rejecting states on path to (q, t)?")

These functions are unknown - we will define a constraint system that defines the kind of functions we are looking for, and ask a constraint solver to find a solution.

Constraint System for Bounded Synthesis

$$\lambda^{\mathbb{B}}_{q_0}(t_0) \wedge \lambda^{\#}_{q_0}(t_0) = 0$$

Constraint System for Bounded Synthesis

$$\lambda^{\mathbb{B}}_{q_0}(t_0) \wedge \lambda^{\#}_{q_0}(t_0) = 0$$

$\forall q, q' \in Q, \forall i \in \mathbb{B}^I, \forall t \in \{0, \ldots, c\}$:

$$\lambda^{\mathbb{B}}_{q}(t) \wedge (q', i) \in \delta(q, \vec{a}(t)) \rightarrow \lambda^{\mathbb{B}}_{q'}(\tau(t, i)) \wedge \lambda^{\#}_{q'}(\tau(t, i)) \geq \lambda^{\#}_{q}(t)$$

Constraint System for Bounded Synthesis

$$\lambda^{\mathbb{B}}_{q_0}(t_0) \wedge \lambda^{\#}_{q_0}(t_0) = 0$$

$\forall q, q' \in Q, \forall i \in \mathbb{B}^I, \forall t \in \{0, \ldots, c\}$:

$$\lambda^{\mathbb{B}}_{q}(t) \wedge (q', i) \in \delta(q, \vec{a}(t)) \rightarrow \lambda^{\mathbb{B}}_{q'}(\tau(t, i)) \wedge \lambda^{\#}_{q'}(\tau(t, i)) \geq \lambda^{\#}_{q}(t)$$

$\forall q \in Q, q' \in F, \forall i \in \mathbb{B}^I, \forall t \in \{0, \ldots, c\}$:

$$\lambda^{\mathbb{B}}_{q}(t) \wedge (q', i) \in \delta(q, \vec{a}(t)) \rightarrow \lambda^{\#}_{q'}(\tau(t, i)) > \lambda^{\#}_{q}(t)$$

Constraint System for Bounded Synthesis

$$\lambda^{\mathbb{B}}_{q_0}(t_0) \wedge \lambda^{\#}_{q_0}(t_0) = 0$$

$\forall q, q' \in Q, \forall i \in \mathbb{B}^I, \forall t \in \{0, \ldots, c\}$:

$$\lambda^{\mathbb{B}}_{q}(t) \wedge (q', i) \in \delta(q, \vec{a}(t)) \rightarrow \lambda^{\mathbb{B}}_{q'}(\tau(t, i)) \wedge \lambda^{\#}_{q'}(\tau(t, i)) \geq \lambda^{\#}_{q}(t)$$

$\forall q \in Q, q' \in F, \forall i \in \mathbb{B}^I, \forall t \in \{0, \ldots, c\}$:

$$\lambda^{\mathbb{B}}_{q}(t) \wedge (q', i) \in \delta(q, \vec{a}(t)) \rightarrow \lambda^{\#}_{q'}(\tau(t, i)) > \lambda^{\#}_{q}(t)$$

$\forall a \in I, \forall i \in \mathbb{B}^I, \forall t \in \{0, \ldots, c\}$:

$$a(\tau(t, i)) \leftrightarrow a \in i$$

Correctness of Constraint System

Theorem

The constraint system is satisfiable (modulo a theory with order) if and only if there exists an input-preserving finite-state transition system that is accepted by $\mathcal{U}$.

Moreover, a satisfying assignment to the functions τ and $\vec{a}$ represents such an LTS.

Correctness of Constraint System

Theorem

The constraint system is satisfiable (modulo a theory with order) if and only if there exists an input-preserving finite-state transition system that is accepted by $\mathcal{U}$.

Moreover, a satisfying assignment to the functions τ and $\vec{a}$ represents such an LTS.

How are such constraint systems solved?

SMT Solving

Constraint systems generated by Bounded Synthesis can be solved by SMT (Satisfiability modulo Theories) solvers.

Theories can be equality, uninterpreted functions, (linear/nonlinear, real/integer) arithmetic, etc.

In our case: uninterpreted functions and a (partial) order.

SMT Solving

Constraint systems generated by Bounded Synthesis can be solved by SMT (Satisfiability modulo Theories) solvers.

Theories can be equality, uninterpreted functions, (linear/nonlinear, real/integer) arithmetic, etc.

In our case: uninterpreted functions and a (partial) order.

Basic approach:

- abstract the problem by interpreting all atomic statements as independent boolean variables
- use SAT solving to find a satisfying assignment
- check if satisfying assignment holds in given theory:
  - if it does, return the solution
  - if not, add a theory lemma (excluding this solution at abstract level) and restart

Alternatives: QBF and SAT encoding

Alternative encodings:

- SAT (basic): introduce a boolean variable for every possible **application** of a function symbol to an argument. Universal quantifiers need to be unrolled. Satisfying assignments can be lifted to functions again.

Alternatives: QBF and SAT encoding

Alternative encodings:

- SAT (basic): introduce a boolean variable for every possible **application** of a function symbol to an argument. Universal quantifiers need to be unrolled. Satisfying assignments can be lifted to functions again.
- QBF (input-symbolic): like SAT encoding, but keep universal quantifiers for input variables.

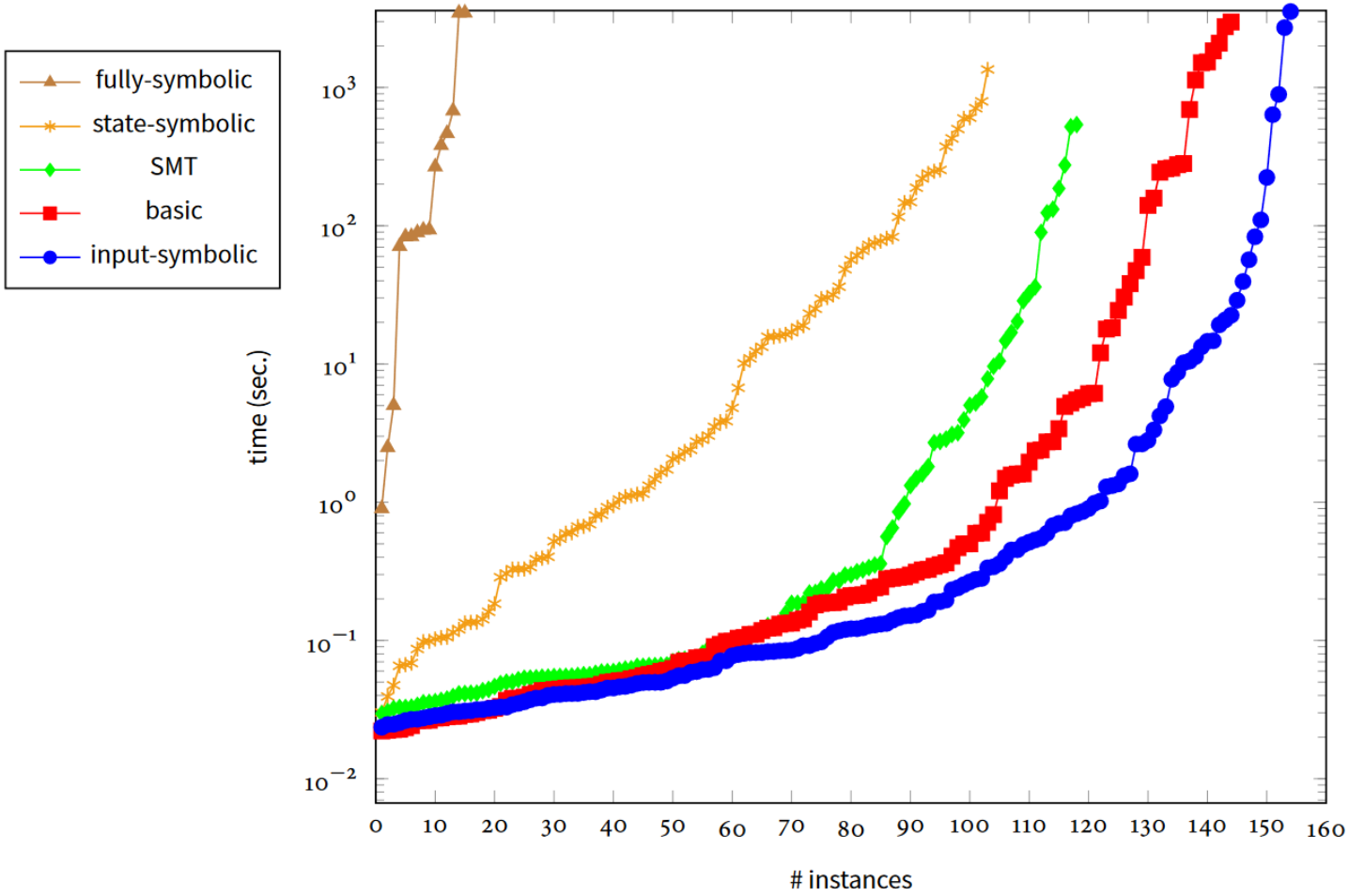
Alternatives: QBF and SAT encoding

Alternative encodings:

- SAT (basic): introduce a boolean variable for every possible **application** of a function symbol to an argument. Universal quantifiers need to be unrolled. Satisfying assignments can be lifted to functions again.
- QBF (input-symbolic): like SAT encoding, but keep universal quantifiers for input variables.
- DQBF: allows to specify which quantifiers another quantifier **depends on**. Like QBF encoding, but keep universal quantifiers for states of LTS (state-symbolic) and possibly also for states of the UCB (fully-symbolic).

Behaviour of different encodings

LTL realizability, SYNTCOMP 2016 experiment set



Alternative: Reduction to Safety Games

For given LTL specification φ and $c \in \mathbb{N}$, the question whether an implementation with a c-bounded annotation exists can be answered by solving a **safety game**.

Alternative: Reduction to Safety Games

For given LTL specification φ and $c \in \mathbb{N}$, the question whether an implementation with a c-bounded annotation exists can be answered by solving a **safety game**.

Approach: expand co-Büchi automaton by adding states that count how often a rejecting state has been visited. If any state is visited more than c times, move to a rejecting sink state.

(other reductions are possible, and are used in practice)

Organization

What remains:

- Next Tuesday: Wrap-up lecture and presentation of competition results. In Seminar Room 15.
- Next Wednesday: Exam. Start at 14:30 (be there at 14:20) in Lecture Hall 3.