Reactive Synthesis

Lecture 10

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
  - LTL Synthesis

The Need for More Expressiveness

Winning conditions introduced thus far:

Reachability reaching a goal (at least once)

Safety staying safe (at all times)

Recurrence reaching a goal infinitely often

Persistence staying safe from some point onwards

Parity maximal color visited infinitely often is odd

The Need for More Expressiveness

Winning conditions introduced thus far:

Reachability reaching a goal (at least once)

Safety staying safe (at all times)

Recurrence reaching a goal infinitely often

Persistence staying safe from some point onwards

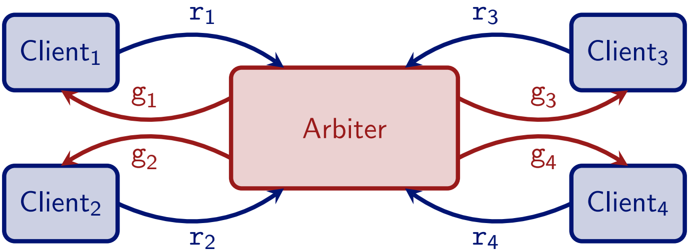
Parity maximal color visited infinitely often is odd

But:

- Specifying real-life properties with these winning conditions is cumbersome (and in some cases impossible).
- Such properties are typically modular, but our winning conditions are (in general) not even closed under boolean combinations.

$\Rightarrow$ we need an expressive high-level specification language.

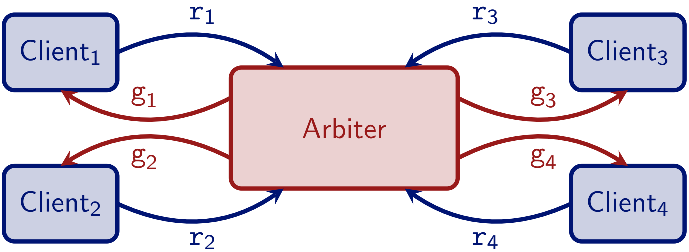
Motivating Example



Setting

- An arbiter for a shared resource and n clients.
- Requests $\mathtt{r}_i$ from client i controlled by the environment.
- Grants $\mathtt{g}_i$ for client i controlled by the system.

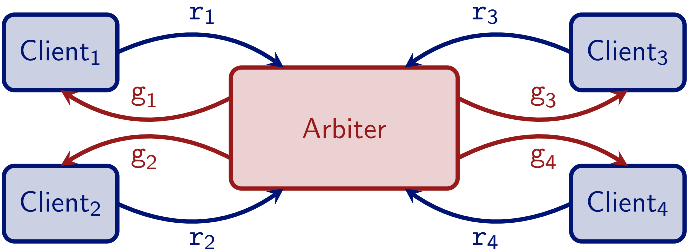
Motivating Example



Example run

Env:
Sys:

Motivating Example

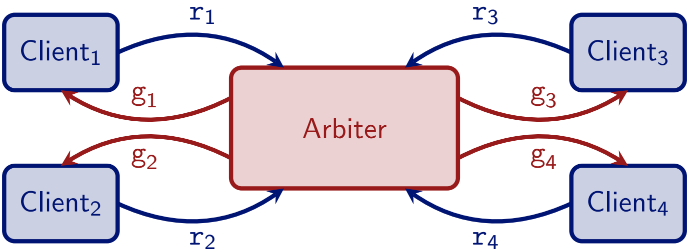


Example run

Env: r_1, r_2

Sys:

Motivating Example

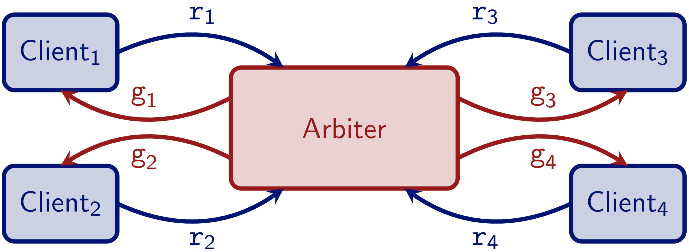


Example run

Env: r_1, r_2

Sys: g_1

Motivating Example

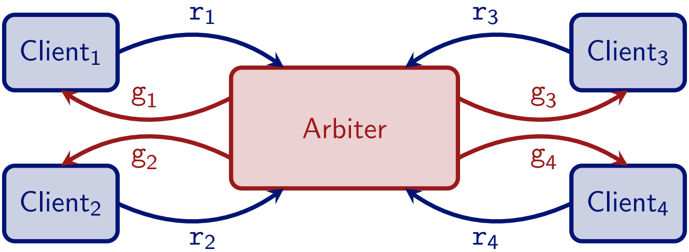


Example run

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

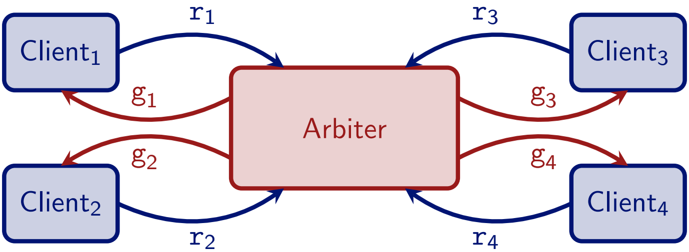
Sys: $\mathtt{g}_1$

Motivating Example



Example run

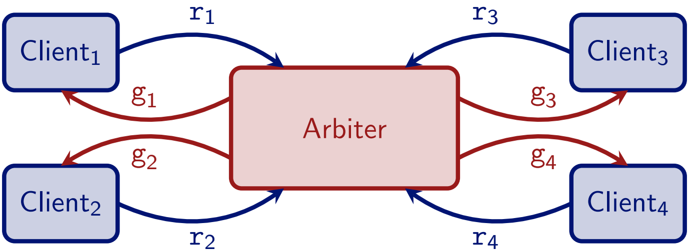
Env:	r_1, r_2	r_1
Sys:	g_1	g_2



Example run

Env:	r_1, r_2	r_1	r_1, r_4
Sys:	g_1	g_2	

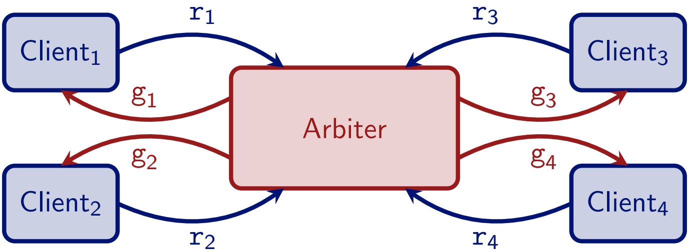
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$

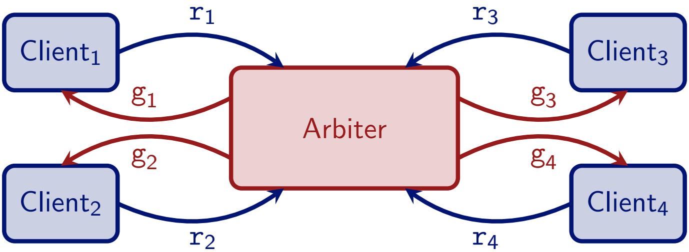
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	–
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	

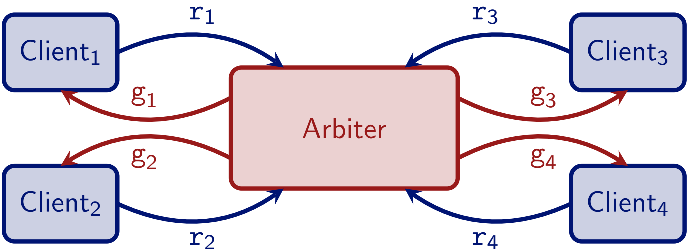
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	$-$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$

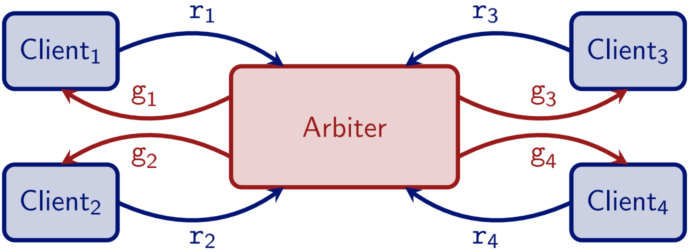
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	–	–
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	

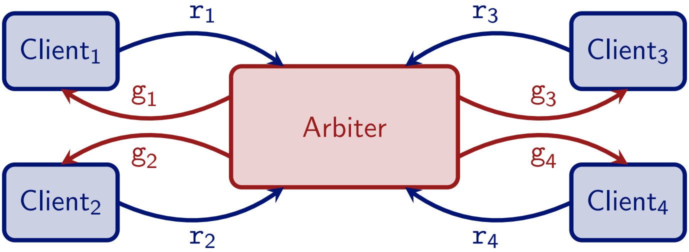
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$

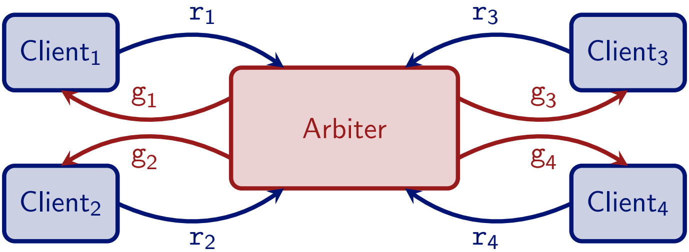
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	–	–	–
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	

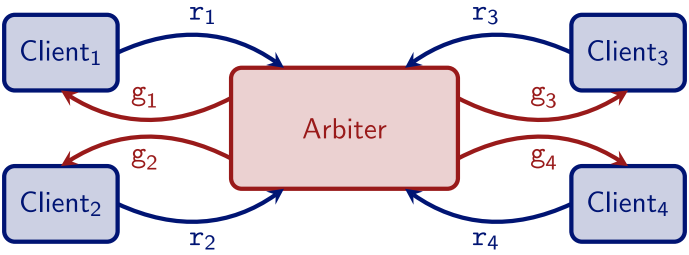
Motivating Example



Example run

Env:	r_1, r_2	r_1	r_1, r_4	—	—	—
Sys:	g_1	g_2	g_3	g_4	g_1	g_2

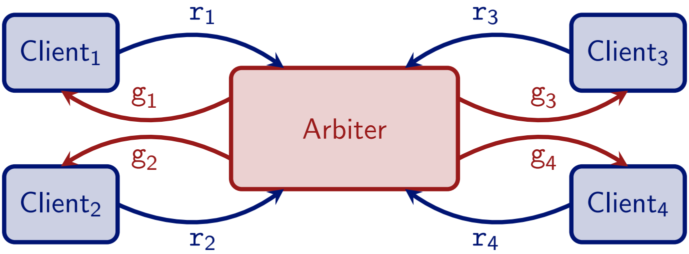
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—	—	$\mathtt{r}_2$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	

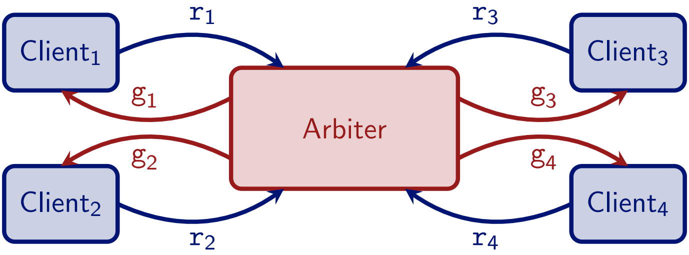
Motivating Example



Example run

Env:	r_1, r_2	r_1	r_1, r_4	—	—	—	r_2
Sys:	g_1	g_2	g_3	g_4	g_1	g_2	g_3

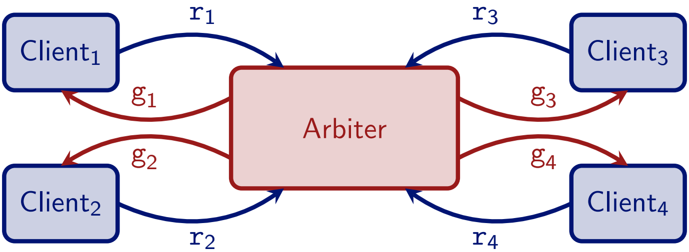
Motivating Example



Example run

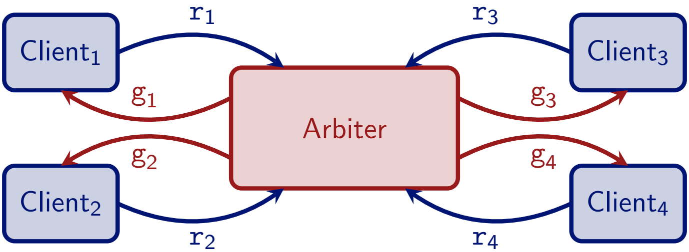
Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—	—	$\mathtt{r}_2$	$\mathtt{r}_1$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	

Motivating Example



Example run

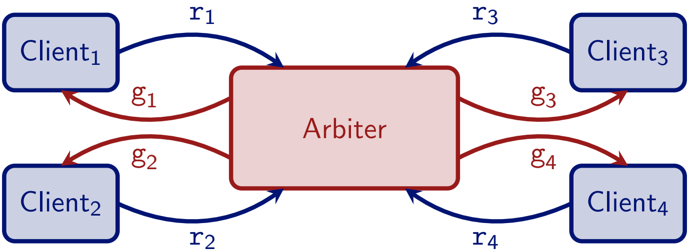
Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—	—	$\mathtt{r}_2$	$\mathtt{r}_1$
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—	—	$\mathtt{r}_2$	$\mathtt{r}_1$	—
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	

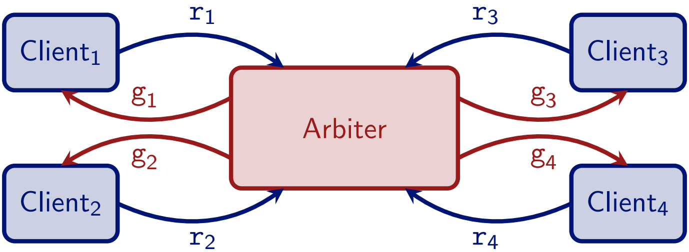
Motivating Example



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—	—	$\mathtt{r}_2$	$\mathtt{r}_1$	—
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$

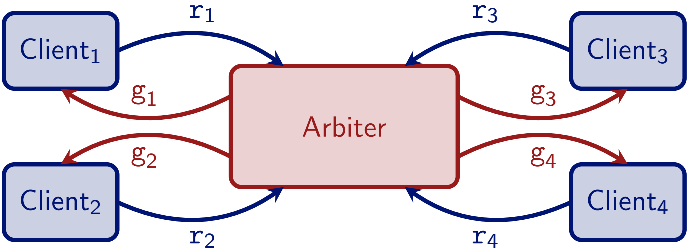
Motivating Example



Example run

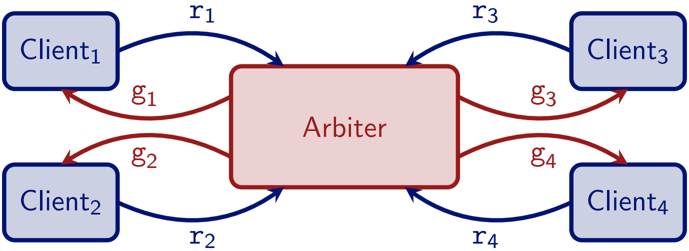
Env:	r_1, r_2	r_1	r_1, r_4	—	—	—	r_2	r_1	—	—
Sys:	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4	g_1	

Motivating Example



Example run

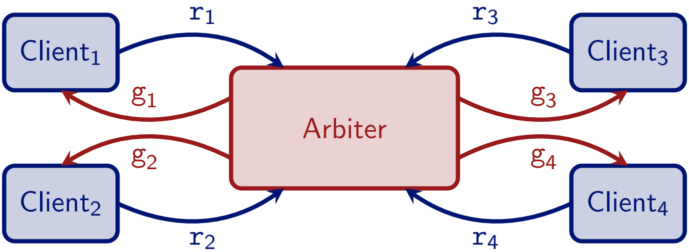
Env:	r_1, r_2	r_1	r_1, r_4	—	—	—	r_2	r_1	—	—
Sys:	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4	g_1	g_2



Example run

Env:	$\mathtt{r}_1, \mathtt{r}_2$	$\mathtt{r}_1$	$\mathtt{r}_1, \mathtt{r}_4$	—	—	—	$\mathtt{r}_2$	$\mathtt{r}_1$	—	—	—
Sys:	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	$\mathtt{g}_3$	$\mathtt{g}_4$	$\mathtt{g}_1$	$\mathtt{g}_2$	

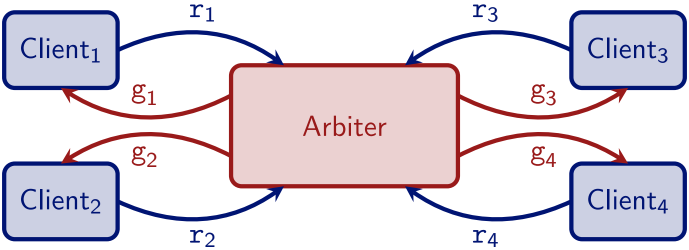
Motivating Example



Example run

Env:	r_1, r_2	r_1	r_1, r_4	—	—	—	r_2	r_1	—	—	—
Sys:	g_1	g_2	g_3	g_4	g_1	g_2	g_3	g_4	g_1	g_2	g_3

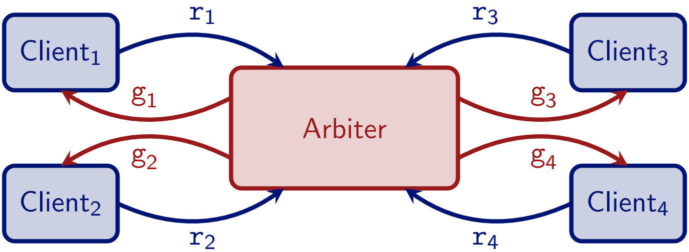
Motivating Example



Typical requirements on an arbiter:

- Every request of a client is eventually granted by the arbiter.

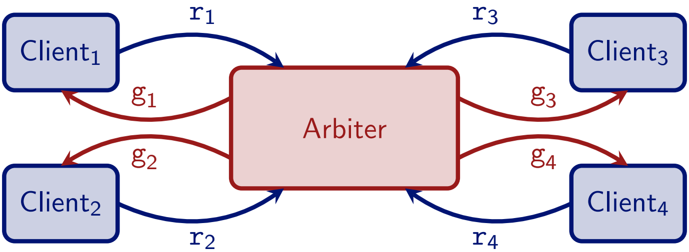
Motivating Example



Typical requirements on an arbiter:

- Every request of a client is eventually granted by the arbiter.
- Never two grants at the same time.

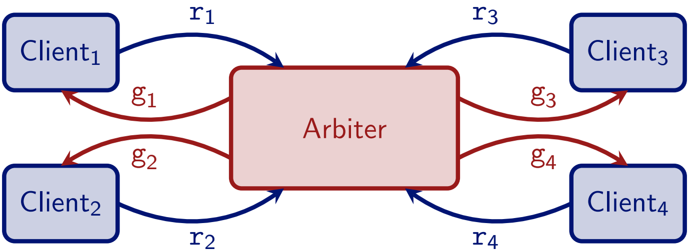
Motivating Example



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.

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.

Linear Temporal Logic (LTL)

A logic to reason about execution traces of reactive systems.

- Vertices are labeled by atomic propositions, e.g., $\mathtt{r}_i$, $\mathtt{g}_i$, `error`, etc.
- Thus, an (infinite) execution of the program induces an infinite sequence of truth values for the atomic propositions.
- LTL has modalities referring to the temporal flow of these truth values.

Linear Temporal Logic (LTL)

A logic to reason about execution traces of reactive systems.

- Vertices are labeled by atomic propositions, e.g., $\mathtt{r}_i$, $\mathtt{g}_i$, `error`, etc.
- Thus, an (infinite) execution of the program induces an infinite sequence of truth values for the atomic propositions.
- LTL has modalities referring to the temporal flow of these truth values.
- The most important specification language for reactive systems.
- "Linear time": consider each execution trace in isolation.

LTL: Syntax and Semantics

Syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi$$

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

LTL: Syntax and Semantics

Syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi$$

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

Semantics

Notation: $(\pi, n) \models \varphi \Leftrightarrow \varphi$ holds at position n of $\pi \in (2^P)^\omega$

LTL: Syntax and Semantics

Syntax

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi$$

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

Semantics

Notation: $(\pi, n) \models \varphi \Leftrightarrow \varphi$ holds at position n of $\pi \in (2^P)^\omega$

- $(\pi, n) \models p$ 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: Syntax and Semantics

Syntax

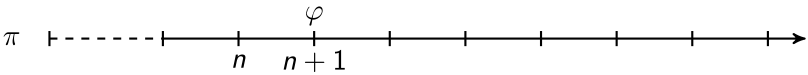
$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi$$

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

Semantics

Notation: $(\pi, n) \models \varphi \Leftrightarrow \varphi$ holds at position n of $\pi \in (2^P)^\omega$

- $(\pi, n) \models \mathbf{X}\,\varphi$ iff $(\pi, n+1) \models \varphi$.



LTL: Syntax and Semantics

Syntax

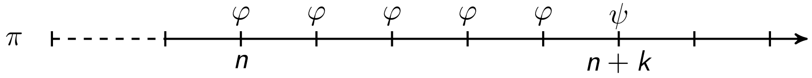
$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi$$

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

Semantics

Notation: $(\pi, n) \models \varphi \Leftrightarrow \varphi$ holds at position n of $\pi \in (2^P)^\omega$

- $(\pi, n) \models \varphi\,\mathbf{U}\,\psi$ iff there is a $k \geq 0$ such that $(\pi, n+k) \models \psi$ and $(\pi, n+j) \models \varphi$ for all $0 \leq j < k$.



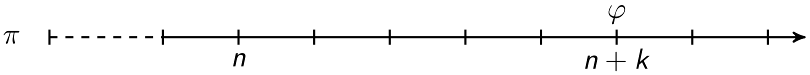
Examples

Use shorthands:

- $\mathtt{tt} = p \vee \neg p$ and $\mathtt{ff} = p \wedge \neg p$ for some $p \in P$
- $\varphi \rightarrow \psi = \neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

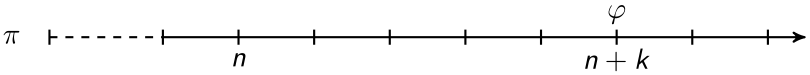
Use shorthands:

- $\texttt{tt} = p \vee \neg p$ and $\texttt{ff} = p \wedge \neg p$ for some $p \in P$
- $\varphi \rightarrow \psi = \neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\mathbf{F}\, \varphi = \texttt{tt}\ \mathbf{U}\ \varphi$: eventually φ holds

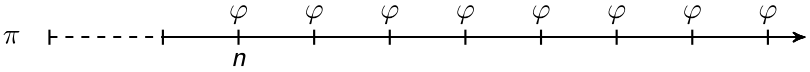


Use shorthands:

- $\mathtt{tt} = p \vee \neg p$ and $\mathtt{ff} = p \wedge \neg p$ for some $p \in P$
- $\varphi \to \psi = \neg\varphi \vee \psi$ and $\varphi \leftrightarrow \psi = (\varphi \to \psi) \wedge (\psi \to \varphi)$
- $\mathbf{F}\,\varphi = \mathtt{tt}\,\mathbf{U}\,\varphi$: eventually φ holds



- $\mathbf{G}\,\varphi = \neg\,\mathbf{F}\,\neg\varphi$: φ always holds



Examples

- reachability: $\mathbf{F}\, r$

Examples

- reachability: $\mathbf{F}\, r$
- safety: $\mathbf{G}\, s$

Examples

- reachability: $\mathbf{F}\, r$
- safety: $\mathbf{G}\, s$
- Büchi: $\mathbf{G}\,\mathbf{F}\, f$

Examples

- reachability: $\mathbf{F}\, r$
- safety: $\mathbf{G}\, s$
- Büchi: $\mathbf{G}\,\mathbf{F}\, f$
- co-Büchi: $\mathbf{F}\,\mathbf{G}\, c$

Examples

- reachability: $\mathbf{F}\, r$
- safety: $\mathbf{G}\, s$
- Büchi: $\mathbf{G}\,\mathbf{F}\, f$
- co-Büchi: $\mathbf{F}\,\mathbf{G}\, c$
- parity: $\bigvee_{\substack{c \in \Omega(V) \\ c \text{ even}}} \left(\mathbf{G}\,\mathbf{F}\, c \wedge \mathbf{F}\,\mathbf{G} \bigwedge_{\substack{c' \in \Omega(V) \\ c' > c}} \neg c' \right)$

Examples

- reachability: $\mathbf{F}\, r$
- safety: $\mathbf{G}\, s$
- Büchi: $\mathbf{G}\,\mathbf{F}\, f$
- co-Büchi: $\mathbf{F}\,\mathbf{G}\, c$
- parity: $\bigvee_{\substack{c\in\Omega(V)\\ c \text{ even}}} \left(\mathbf{G}\,\mathbf{F}\, c \wedge \mathbf{F}\,\mathbf{G} \bigwedge_{\substack{c'\in\Omega(V)\\ c'>c}} \neg c' \right)$
- weak parity: $\bigvee_{\substack{c\in\Omega(V)\\ c \text{ even}}} \left(\mathbf{F}\, c \wedge \mathbf{G} \bigwedge_{\substack{c'\in\Omega(V)\\ c'>c}} \neg c' \right)$

Arbiter Specification in LTL

Example specifications

1. Answer every request: $\bigwedge_i \mathbf{G}(\mathtt{r}_i \rightarrow \mathbf{F}\, \mathtt{g}_i)$

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: $\mathbf{G} \bigwedge_{i \neq j} \neg(\mathtt{g}_i \wedge \mathtt{g}_j)$

Arbiter Specification in LTL

Example specifications

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

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

Arbiter Specification in LTL

Example specifications

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

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

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

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

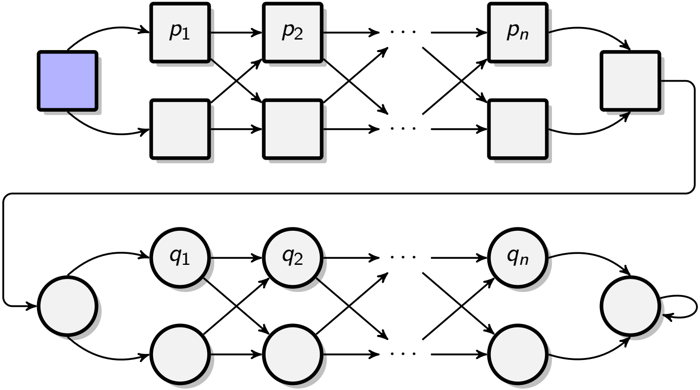
Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $\ell \colon V \to 2^P$ be a labeling of $\mathcal{A}$'s vertices by atomic propositions. Then, the LTL condition $\mathrm{LTL}(\varphi, \ell)$ is defined as

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

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

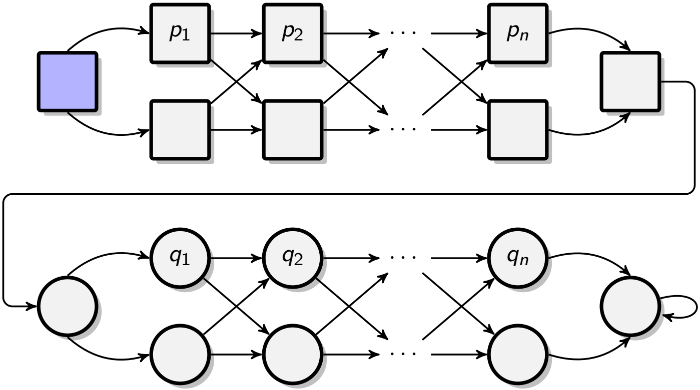
Example

$$\varphi = \bigwedge_{i=1,\ldots,n} (\mathbf{F}\, p_i) \leftrightarrow (\mathbf{F}\, q_i)$$



Example

$$\varphi = \bigwedge_{i=1,\ldots,n} (\mathbf{F}\, p_i) \leftrightarrow (\mathbf{F}\, q_i)$$



Player 0 has a finite-state winning strategy from the blue vertex, but none with less than 2^n memory states.

Solving LTL Games

Solving LTL Games

Idea

- Use game reductions.
- To this end, we need to translate an LTL formula into a deterministic automaton on infinite words.
- Then, the game obtained in the reduction *inherits* the acceptance condition of the automaton as winning condition.

Solving LTL Games

Idea

- Use game reductions.
- To this end, we need to translate an LTL formula into a deterministic automaton on infinite words.
- Then, the game obtained in the reduction *inherits* the acceptance condition of the automaton as winning condition.
- We use the following chain of translations:
    - Every LTL formula can be translated into an equivalent non-deterministic Büchi automaton.

Solving LTL Games

Idea

- Use game reductions.
- To this end, we need to translate an LTL formula into a deterministic automaton on infinite words.
- Then, the game obtained in the reduction *inherits* the acceptance condition of the automaton as winning condition.
- We use the following chain of translations:
  - Every LTL formula can be translated into an equivalent non-deterministic Büchi automaton.
  - Every non-deterministic Büchi automaton can be translated into an equivalent deterministic parity automaton.

Solving LTL Games

Idea

- Use game reductions.
- To this end, we need to translate an LTL formula into a deterministic automaton on infinite words.
- Then, the game obtained in the reduction *inherits* the acceptance condition of the automaton as winning condition.
- We use the following chain of translations:
  - Every LTL formula can be translated into an equivalent non-deterministic Büchi automaton.
  - Every non-deterministic Büchi automaton can be translated into an equivalent deterministic parity automaton.
- Using this automaton, we can indeed reduce LTL games to parity games.

Büchi Automata

Definition

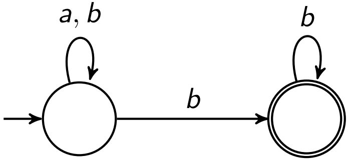
A non-deterministic Büchi automaton $\mathfrak{B} = (Q, \Sigma, q_{\mathrm{I}}, \Delta, F)$ consists of

- a finite set Q of states,
- an alphabet Σ,
- an initial state q_{I},
- a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and
- a set $F \subseteq Q$ of accepting states.

Definition

A non-deterministic Büchi automaton $\mathfrak{B} = (Q, \Sigma, q_{\mathrm{I}}, \Delta, F)$ consists of

- a finite set Q of states,
- an alphabet Σ,
- an initial state q_{I},
- a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and
- a set $F \subseteq Q$ of accepting states.



Büchi Automata

Definition

A non-deterministic Büchi automaton $\mathfrak{B} = (Q, \Sigma, q_{\mathrm{I}}, \Delta, F)$ consists of

- a finite set Q of states,
- an alphabet Σ,
- an initial state q_{I},
- a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and
- a set $F \subseteq Q$ of accepting states.

- A run of $\mathfrak{B}$ on an ω-word $\alpha = \alpha_0\alpha_1\alpha_2 \cdots \in \Sigma^\omega$ is an infinite sequence $q_0q_1q_2 \cdots$ of states such that $q_0 = q_{\mathrm{I}}$ and $(q_n, \alpha_n, q_{n+1}) \in \Delta$ for every $n \in \mathbb{N}$.
- A run $q_0q_1q_2 \cdots$ is accepting if $\mathrm{Inf}(q_0q_1q_2 \cdots) \cap F \neq \emptyset$.
- The language $L(\mathfrak{B})$ contains all $\alpha \in \Sigma^\omega$ such that $\mathfrak{B}$ has an accepting run on α.

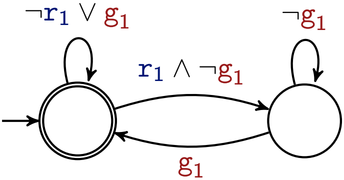
Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathrm{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathrm{g}_1$ represents $\{\emptyset, P\}$.

Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathrm{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathrm{g}_1$ represents $\{\emptyset, P\}$.

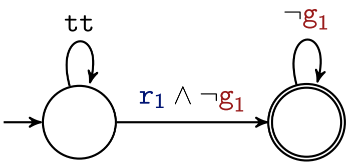
Every request is answered



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

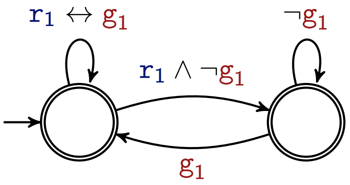
Some request is not answered



Examples

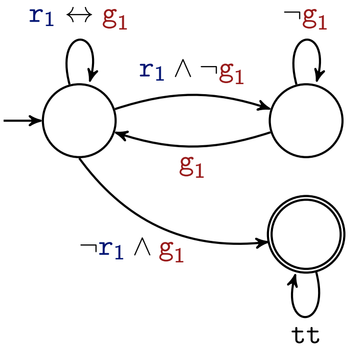
Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

No spurious grants



Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

There is a spurious grant



Theorem

For every LTL formula φ there is a non-deterministic Büchi automaton $\mathfrak{B}_\varphi$ with

$$L(\mathfrak{B}_\varphi) = \{\pi \in (2^P)^\omega \mid (\pi, 0) \models \varphi\}$$

and $2^{\mathcal{O}(|\varphi|)}$ states.

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

Parity Automata

Definition

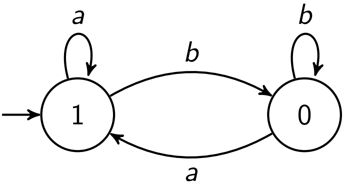
A deterministic parity automaton $\mathfrak{P} = (Q, \Sigma, q_\mathrm{I}, \delta, \Omega)$ consists of

- a finite set Q of states,
- an alphabet Σ,
- an initial state q_I,
- a transition function $\delta \colon Q \times \Sigma \to Q$, and
- a coloring $\Omega \colon Q \to \mathbb{N}$ of the states.

Definition

A deterministic parity automaton $\mathfrak{P} = (Q, \Sigma, q_{\mathrm{I}}, \delta, \Omega)$ consists of

- a finite set Q of states,
- an alphabet Σ,
- an initial state q_{I},
- a transition function $\delta \colon Q \times \Sigma \to Q$, and
- a coloring $\Omega \colon Q \to \mathbb{N}$ of the states.



Definition

A deterministic parity automaton $\mathfrak{P} = (Q, \Sigma, q_\mathrm{I}, \delta, \Omega)$ consists of

- a finite set Q of states,
- an alphabet Σ,
- an initial state q_I,
- a transition function $\delta \colon Q \times \Sigma \to Q$, and
- a coloring $\Omega \colon Q \to \mathbb{N}$ of the states.

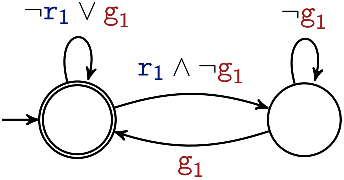
- The unique run of $\mathfrak{P}$ on an ω-word $\alpha = \alpha_0\alpha_1\alpha_2 \cdots \in \Sigma^\omega$ is the infinite sequence $q_0 q_1 q_2 \cdots$ of states with $q_0 = q_\mathrm{I}$ and $q_{n+1} = \delta(q_n, \alpha_n)$ for every $n \in \mathbb{N}$.
- It is accepting if $\max \mathrm{Inf}(\Omega(q_0)\Omega(q_1)\Omega(q_2)\cdots)$ is even.
- The language $L(\mathfrak{P})$ contains all $\alpha \in \Sigma^\omega$ such that the run of $\mathfrak{P}$ on α is accepting.

Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathrm{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathrm{g}_1$ represents $\{\emptyset, P\}$.

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

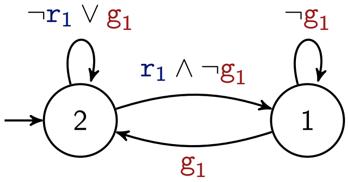
Every request is answered (Büchi)



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathrm{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathrm{g}_1$ represents $\{\emptyset, P\}$.

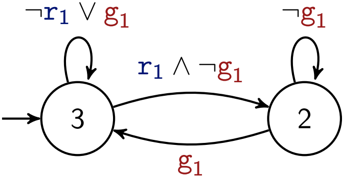
Every request is answered (parity)



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

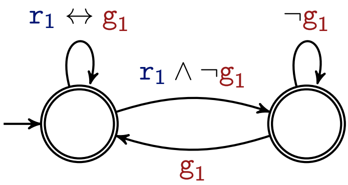
Some request is not answered (parity)



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

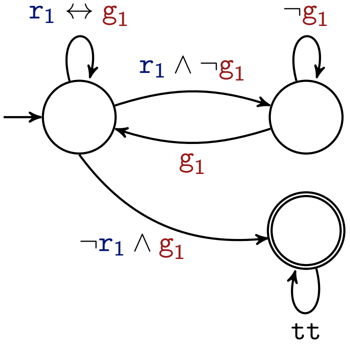
No spurious grants (Büchi)



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathrm{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathrm{g}_1$ represents $\{\emptyset, P\}$.

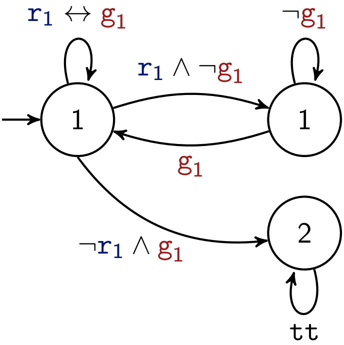
There is a spurious grant (Büchi)



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

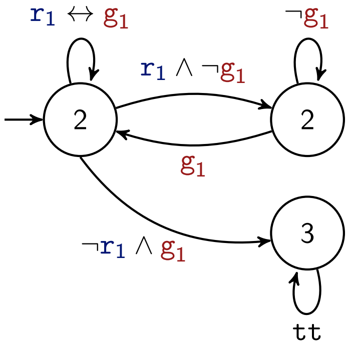
There is a spurious grant (parity)



Examples

Use alphabet 2^P and propositional formulas to represent sets of letters, e.g., if $P = \{\mathtt{r}_1, \mathtt{g}_1\}$, then $\mathtt{r}_1 \leftrightarrow \mathtt{g}_1$ represents $\{\emptyset, P\}$.

No spurious grants (parity)



Theorem

For every non-deterministic Büchi automaton $\mathfrak{B}$ there is a deterministic parity automaton $\mathfrak{P}$ recognizing the same language with $2^{\mathcal{O}(|\mathfrak{B}| \log |\mathfrak{B}|)}$ states and $\mathcal{O}(|\mathfrak{B}|)$ colors.

Corollary

For every LTL formula φ there is a deterministic parity automaton $\mathfrak{P}_\varphi$ with

$$L(\mathfrak{P}_\varphi) = \{\pi \in (2^P)^\omega \mid (\pi, 0) \models \varphi\}$$

and $2^{2^{\mathcal{O}(|\varphi|)}}$ states and $2^{\mathcal{O}(|\varphi|)}$ colors.

Back in Lecture 8

The product $\mathcal{A} \times \mathcal{M} = (V', V'_0, V'_1, E')$ of $\mathcal{A} = (V, V_0, V_1, E)$ and a memory structure $\mathcal{M} = (M, \mathrm{init}, \mathrm{upd})$ for $\mathcal{A}$ is defined as

- $V' = V \times M$,
- $V'_i = V_i \times M$, and
- $((v, m), (v', m')) \in E'$ if and only if $(v, v') \in E$ and $\mathrm{upd}(m, v') = m'$.

Back in Lecture 8

The product $\mathcal{A} \times \mathcal{M} = (V', V_0', V_1', E')$ of $\mathcal{A} = (V, V_0, V_1, E)$ and a memory structure $\mathcal{M} = (M, \mathrm{init}, \mathrm{upd})$ for $\mathcal{A}$ is defined as

- $V' = V \times M$,
- $V_i' = V_i \times M$, and
- $((v, m), (v', m')) \in E'$ if and only if $(v, v') \in E$ and $\mathrm{upd}(m, v') = m'$.

A game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ is reducible to a game $\mathcal{G}' = (\mathcal{A}', \mathrm{Win}')$ via $\mathcal{M}$, written $\mathcal{G} \leq_{\mathcal{M}} \mathcal{G}'$, if

- $\mathcal{A}' = \mathcal{A} \times \mathcal{M}$, and
- for each play ρ in $\mathcal{A}$: $\rho \in \mathrm{Win}$ if and only if $\mathrm{ext}(\rho) \in \mathrm{Win}'$.

Back in Lecture 8

The product $\mathcal{A} \times \mathcal{M} = (V', V_0', V_1', E')$ of $\mathcal{A} = (V, V_0, V_1, E)$ and a memory structure $\mathcal{M} = (M, \mathrm{init}, \mathrm{upd})$ for $\mathcal{A}$ is defined as

- $V' = V \times M$,
- $V_i' = V_i \times M$, and
- $((v, m), (v', m')) \in E'$ if and only if $(v, v') \in E$ and $\mathrm{upd}(m, v') = m'$.

A game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ is reducible to a game $\mathcal{G}' = (\mathcal{A}', \mathrm{Win}')$ via $\mathcal{M}$, written $\mathcal{G} \leq_{\mathcal{M}} \mathcal{G}'$, if

- $\mathcal{A}' = \mathcal{A} \times \mathcal{M}$, and
- for each play ρ in $\mathcal{A}$: $\rho \in \mathrm{Win}$ if and only if $\mathrm{ext}(\rho) \in \mathrm{Win}'$.

Corollary

Let $\mathcal{G} \leq_{\mathcal{M}} \mathcal{G}'$. If $\mathcal{G}'$ is determined with positional strategies, then $\mathcal{G}$ is determined with finite-state strategies implemented by $\mathcal{M}$ and solving $\mathcal{G}'$ solves $\mathcal{G}$.

Reducing LTL Games to Parity Games

Fix an LTL game $(\mathcal{A}, \mathrm{LTL}(\varphi, \ell))$.

1. Construct deterministic parity automaton $\mathfrak{P}_\varphi = (Q, 2^P, q_\mathrm{I}, \delta, \Omega)$ as before.

Reducing LTL Games to Parity Games

Fix an LTL game $(\mathcal{A}, \mathrm{LTL}(\varphi, \ell))$.

1. Construct deterministic parity automaton $\mathfrak{P}_\varphi = (Q, 2^P, q_\mathrm{I}, \delta, \Omega)$ as before.
2. Define memory structure $\mathcal{M} = (Q, \mathrm{init}, \mathrm{upd})$ with
   - $\mathrm{init}(v) = \delta(q_\mathrm{I}, \ell(v))$ and
   - $\mathrm{upd}(q, v) = \delta(q, \ell(v))$.

Reducing LTL Games to Parity Games

Fix an LTL game $(\mathcal{A}, \mathrm{LTL}(\varphi, \ell))$.

1. Construct deterministic parity automaton $\mathfrak{P}_\varphi = (Q, 2^P, q_\mathrm{I}, \delta, \Omega)$ as before.
2. Define memory structure $\mathcal{M} = (Q, \mathrm{init}, \mathrm{upd})$ with
   - $\mathrm{init}(v) = \delta(q_\mathrm{I}, \ell(v))$ and
   - $\mathrm{upd}(q, v) = \delta(q, \ell(v))$.
3. Consider parity game $\mathcal{G}' = (\mathcal{A} \times \mathcal{M}, \textsc{Parity}(\Omega'))$ with $\Omega'(v, q) = \Omega(q)$.

Reducing LTL Games to Parity Games

Fix an LTL game $(\mathcal{A}, \mathrm{LTL}(\varphi, \ell))$.

1. Construct deterministic parity automaton $\mathfrak{P}_\varphi = (Q, 2^P, q_\mathrm{I}, \delta, \Omega)$ as before.
2. Define memory structure $\mathcal{M} = (Q, \mathrm{init}, \mathrm{upd})$ with
   - $\mathrm{init}(v) = \delta(q_\mathrm{I}, \ell(v))$ and
   - $\mathrm{upd}(q, v) = \delta(q, \ell(v))$.
3. Consider parity game $\mathcal{G}' = (\mathcal{A} \times \mathcal{M}, \textsc{Parity}(\Omega'))$ with $\Omega'(v, q) = \Omega(q)$.

Lemma

1. *$\mathcal{G}'$ has doubly-exponential size and exponentially many colors (both in $|\mathcal{A}| + |\varphi|$).*
2. *$\mathcal{G} \leq_\mathcal{M} \mathcal{G}'$.*

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.

Proof.

Construct parity game $\mathcal{G}'$ and solve it in time

$$(|\mathcal{A}| \cdot 2^{2^{\mathcal{O}(|\varphi|)}})^{\log(2^{\mathcal{O}(|\varphi|)})}. \qquad \square$$

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.

Proof.

Construct parity game $\mathcal{G}'$ and solve it in time

$$(|\mathcal{A}| \cdot 2^{2^{\mathcal{O}(|\varphi|)}})^{\log(2^{\mathcal{O}(|\varphi|)})}. \qquad \square$$

Remark

Both lower bounds are tight.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
- GR(1): syntactic fragment with polynomial complexity.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
- GR(1): syntactic fragment with polynomial complexity.
- extensions of LTL that capture the ω-regular languages: (typically) same complexity.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
- GR(1): syntactic fragment with polynomial complexity.
- extensions of LTL that capture the ω-regular languages: (typically) same complexity.
- extensions of LTL that capture the visibly ω-contextfree languages: (typically) slightly higher complexity.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
- GR(1): syntactic fragment with polynomial complexity.
- extensions of LTL that capture the ω-regular languages: (typically) same complexity.
- extensions of LTL that capture the visibly ω-contextfree languages: (typically) slightly higher complexity.
- parameterized LTL with bounds on temporal operators: same complexity.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
- GR(1): syntactic fragment with polynomial complexity.
- extensions of LTL that capture the ω-regular languages: (typically) same complexity.
- extensions of LTL that capture the visibly ω-contextfree languages: (typically) slightly higher complexity.
- parameterized LTL with bounds on temporal operators: same complexity.
- branching time logics: a whole new world.

Beyond LTL

Numerous variants of LTL have been considered:

- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
- GR(1): syntactic fragment with polynomial complexity.
- extensions of LTL that capture the ω-regular languages: (typically) same complexity.
- extensions of LTL that capture the visibly ω-contextfree languages: (typically) slightly higher complexity.
- parameterized LTL with bounds on temporal operators: same complexity.
- branching time logics: a whole new world.
- real-time logics: higher complexity, often undecidability.

Exam

Some information about the exam:

- Date: January 31st, 2018 at 14:30 (slot of the last lecture). Please be there ten minutes earlier.
- Duration: 90 minutes.
- Closed book!
- You need 40 points from the problem sets to be admitted.

Save the date: on January 30th we present the results of the competition.