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

But:

Specifying real-life properties with these winning conditions is cumbersome (and in some cases impossible).

⇒ we need an expressive high-level specification language.
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.

⇒ we need an expressive high-level specification language.
Motivating Example

Setting

- An arbiter for a shared resource and \( n \) clients.
- Requests \( r_i \) from client \( i \) controlled by the environment.
- Grants \( g_i \) 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: $r_1, r_2, r_1$
Sys: $g_1$
Motivating Example

Example run

Env: $r_1, r_2, r_1$

Sys: $g_1, g_2$
Motivating Example

Example run

Env: \( r_1, r_2 \quad r_1 \quad r_1, r_4 \)
Sys: \( g_1 \quad g_2 \)
Motivating Example

Example run

Env: \( r_1, r_2 \), \( r_1 \), \( r_1, r_4 \)
Sys: \( g_1 \), \( g_2 \), \( g_3 \)
Motivating Example

Example run

Env: \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad r_4 \)

Sys: \( g_1 \quad g_2 \quad g_3 \)
Motivating Example

Example run

Env: $r_1, r_2 \quad r_1 \quad r_1, r_4 \quad -$ 

Sys: $g_1 \quad g_2 \quad g_3 \quad g_4$
Motivating Example

Example run

Env: \( r_1, r_2 \)  \( r_1 \)  \( r_1, r_4 \)  \( - \)  \( - \)

Sys: \( g_1 \)  \( g_2 \)  \( g_3 \)  \( g_4 \)
Motivating Example

Example run

Env:  \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \)

Sys:  \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \)
Motivating Example

Example run

Env: $r_1, r_2 \quad r_1 \quad r_1, r_4 \quad -$ $-$ $-$

Sys: $g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1$
Motivating Example

Example run

Env:  \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \)

Sys:  \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \)
Motivating Example

Example run

Env:  \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \)

Sys:  \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \)
Motivating Example

Example run

Env:  \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \)
Sys:  \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \)
Motivating Example

Example run

Env: \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \quad r_1 \)

Sys: \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \)
Motivating Example

Example run

Env: $r_1, r_2$ $r_1$ $r_1, r_4$ $-$ $-$ $-$ $r_2$ $r_1$
Sys: $g_1$ $g_2$ $g_3$ $g_4$ $g_1$ $g_2$ $g_3$ $g_4$
Motivating Example

Example run

Env: \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \quad r_1 \quad - \)

Sys: \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \quad g_4 \)
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:  \( g_1 \)  \( g_2 \)  \( g_3 \)  \( g_4 \)  \( g_1 \)  \( g_2 \)  \( g_3 \)  \( g_4 \)  \( g_1 \)
Motivating Example

Example run

Env: \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \quad r_1 \quad - \quad - \)

Sys: \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \)
Motivating Example

Example run

Env:  $r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \quad r_1 \quad - \quad - \quad -$

Sys:  $g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2$
Motivating Example

Example run

Env: \( r_1, r_2 \quad r_1 \quad r_1, r_4 \quad - \quad - \quad - \quad r_2 \quad r_1 \quad - \quad - \quad - \)

Sys: \( g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_1 \quad g_2 \quad g_3 \)
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.
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., $r_i$, $g_i$, 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., $r_i$, $g_i$, 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

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \]

where \( p \) ranges over a finite set \( P \) of atomic propositions.
**LTL: Syntax and Semantics**

**Syntax**

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \]

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

**Semantics**

Notation: \( (\pi, n) \models \varphi \iff \varphi \) holds at position \( n \) of \( \pi \in (2^P)\omega \)
LTL: Syntax and Semantics

Syntax

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \]

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

Semantics

Notation: \( (\pi, n) \models \varphi \iff \varphi \) 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 \( (\pi, n) \models \varphi \).
- \( (\pi, n) \models \varphi \land \psi \iff (\pi, n) \models \varphi \) and \( (\pi, n) \models \psi \).
- \( (\pi, n) \models \varphi \lor \psi \iff (\pi, n) \models \varphi \) or \( (\pi, n) \models \psi \).
LTL: Syntax and Semantics

Syntax

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \]

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

Semantics

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

\[ (\pi, n) \models X \varphi \iff (\pi, n + 1) \models \varphi. \]

\[ \pi \quad \varphi \]
\[ n \quad n + 1 \]
LTL: Syntax and Semantics

Syntax

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid X \varphi \mid \varphi U \varphi \]

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

Semantics

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

\[ (\pi, n) \models \varphi U \psi \text{ iff there is a } k \geq 0 \text{ such that } (\pi, n + k) \models \psi \]

and \((\pi, n + j) \models \varphi \) for all \( 0 \leq j < k \).
Examples

Use shorthands:

- $tt = p \lor \neg p$ and $ff = p \land \neg p$ for some $p \in P$
- $\varphi \rightarrow \psi = \neg \varphi \lor \psi$ and $\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi)$
Examples

Use shorthands:

- $\mathsf{tt} = p \lor \neg p$ and $\mathsf{ff} = p \land \neg p$ for some $p \in P$
- $\varphi \rightarrow \psi = \neg \varphi \lor \psi$ and $\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi)$
- $\mathbf{F} \varphi = \mathsf{tt} \mathbf{U} \varphi$: eventually $\varphi$ holds
Use shorthands:

- \( \text{tt} = p \lor \neg p \) and \( \text{ff} = p \land \neg p \) for some \( p \in P \)
- \( \varphi \to \psi = \neg \varphi \lor \psi \) and \( \varphi \leftrightarrow \psi = (\varphi \to \psi) \land (\psi \to \varphi) \)
- \( \mathbf{F} \varphi = \text{tt } \mathbf{U} \varphi \): eventually \( \varphi \) holds

\[ \varphi \]

\[ \pi \quad \vdash \quad n \quad \vdash \quad n + k \]

- \( \mathbf{G} \varphi = \neg \mathbf{F} \neg \varphi \): \( \varphi \) always holds

\[ \varphi \]

\[ \pi \quad \vdash \quad n \quad \vdash \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \]
Examples

- reachability: $F r$
Examples

- reachability: $F r$
- safety: $G s$
Examples

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

- reachability: $F r$
- safety: $G s$
- Büchi: $G F f$
- co-Büchi: $F G c$
Examples

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

- reachability: $F r$
- safety: $G s$
- Büchi: $G F f$
- co-Büchi: $F G c$

parity: $\bigvee_{c \in \Omega(V)} \left( G F c \land F G \bigwedge_{c' \in \Omega(V)} \neg c' \right)_{c \text{ even}}$

weak parity: $\bigvee_{c \in \Omega(V)} \left( F c \land G \bigwedge_{c' \in \Omega(V)} \neg c' \right)_{c \text{ even}}$
Example specifications

1. Answer every request: $\bigwedge_i G(r_i \rightarrow F g_i)$
Example specifications

1. Answer every request: $\bigwedge_i G(r_i \rightarrow F g_i)$
2. At most one grant at a time: $G \bigwedge_{i \neq j} \neg (g_i \land g_j)$
Example specifications

1. Answer every request: $\bigwedge_i \mathbf{G}(r_i \rightarrow \mathbf{F} g_i)$
2. At most one grant at a time: $\mathbf{G} \bigwedge_{i \neq j} \neg (g_i \land g_j)$
3. No spurious grants:

$$\bigwedge_i \neg [(\neg r_i \mathbf{U} (\neg r_i \land g_i)) \land \neg (\mathbf{F} (g_i \land \mathbf{X} (\neg r_i \mathbf{U} (\neg r_i \land g_i))))]$$
Example specifications

1. Answer every request: $\bigwedge_i G(r_i \rightarrow F g_i)$
2. At most one grant at a time: $G \bigwedge_{i \neq j} \neg (g_i \land g_j)$
3. No spurious grants:

   $$\bigwedge_i \neg[(\neg r_i U (\neg r_i \land g_i))] \land \neg [F (g_i \land X (\neg r_i U (\neg r_i \land g_i)))]$$

4. No new requests while request is open:
   $$\bigwedge_i G(r_i \rightarrow X(\neg r_i U g_i))$$
**Notation**
Given a labeling $\ell : V \rightarrow 2^P$ 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 $\ell : V \rightarrow 2^P$ be a labeling of $\mathcal{A}$’s vertices by atomic propositions. Then, the LTL condition $LTL(\varphi, \ell)$ is defined as

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

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

\[ \varphi = \bigwedge_{i=1,\ldots,n} (F p_i) \leftrightarrow (F q_i) \]
Example

\[ \varphi = \bigwedge_{i=1,...,n} (F p_i) \leftrightarrow (F q_i) \]

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

Idea

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.
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.
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 \( \mathcal{B} = (Q, \Sigma, q_I, \Delta, F) \) consists of

- a finite set \( Q \) of states,
- an alphabet \( \Sigma \),
- 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 $\mathcal{B} = (Q, \Sigma, q_I, \Delta, F)$ consists of

- a finite set $Q$ of states,
- an alphabet $\Sigma$,
- 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 $\mathcal{B} = (Q, \Sigma, q_I, \Delta, F)$ consists of

- a finite set $Q$ of states,
- an alphabet $\Sigma$,
- 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 $\mathcal{B}$ on an $\omega$-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 $q_0 = q_I$ 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 $\text{Inf}(q_0q_1q_2 \cdots) \cap F \neq \emptyset$.

The language $L(\mathcal{B})$ contains all $\alpha \in \Sigma^\omega$ such that $\mathcal{B}$ has an accepting run on $\alpha$. 
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$. 
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

Every request is answered
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

Some request is not answered
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

No spurious grants

![Diagram with nodes and edges representing logical formulas]

- $r_1 \leftrightarrow g_1$
- $\neg g_1$
- $r_1 \land \neg g_1$
- $g_1$
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

There is a spurious grant

\[
r_1 \leftrightarrow g_1 \\
\neg g_1
\]

\[
r_1 \land \neg g_1
\]

\[
g_1
\]

\[
\neg r_1 \land g_1
\]

\[
tt
\]
Theorem
For every LTL formula $\varphi$ there is a non-deterministic Büchi automaton $\mathcal{B}_\varphi$ with

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

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

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

**Definition**
A deterministic parity automaton \( \mathcal{A} = (Q, \Sigma, q_I, \delta, \Omega) \) consists of

- a finite set \( Q \) of states,
- an alphabet \( \Sigma \),
- an initial state \( q_I \),
- a transition function \( \delta: Q \times \Sigma \to Q \), and
- a coloring \( \Omega: Q \to \mathbb{N} \) of the states.
Parity Automata

Definition
A deterministic parity automaton $\mathcal{A} = (Q, \Sigma, q_I, \delta, \Omega)$ consists of

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

Definition
A deterministic parity automaton $\mathcal{P} = (Q, \Sigma, q_I, \delta, \Omega)$ consists of

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

The unique run of $\mathcal{P}$ on an $\omega$-word $\alpha = \alpha_0\alpha_1\alpha_2 \cdots \in \Sigma^\omega$ is the infinite sequence $q_0q_1q_2 \cdots$ of states with $q_0 = q_I$ and $q_{n+1} = \delta(q_n, \alpha_n)$ for every $n \in \mathbb{N}$.

- It is accepting if $\max \text{Inf}(\Omega(q_0)\Omega(q_1)\Omega(q_2)\cdots)$ is even.

- The language $L(\mathcal{P})$ contains all $\alpha \in \Sigma^\omega$ such that the run of $\mathcal{P}$ on $\alpha$ is accepting.
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$. 
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

Every request is answered (Büchi)
Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

Every request is answered (parity)
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

Some request is not answered (parity)

![Diagram showing a state transition graph with nodes labeled 3 and 2, and transitions labeled with logical expressions: $\neg r_1 \lor g_1$, $r_1 \land \neg g_1$, $\neg g_1$, and $g_1$.](image)
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

No spurious grants (Büchi)

\[ r_1 \leftrightarrow g_1 \]
\[ r_1 \land \neg g_1 \]
\[ \neg g_1 \]
Examples

Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ 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 $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

There is a spurious grant (parity)
Use alphabet $2^P$ and propositional formulas to represent sets of letters, e.g., if $P = \{r_1, g_1\}$, then $r_1 \leftrightarrow g_1$ represents $\{\emptyset, P\}$.

No spurious grants (parity)
Büchi Automata to Parity Automata

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

**Corollary**
For every LTL formula $\varphi$ there is a deterministic parity automaton $P_\varphi$ with

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

and $2^{2^{O(|\varphi|)}}$ states and $2^{O(|\varphi|)}$ colors.
The product $A \times M = (V', V_0', V_1', E')$ of $A = (V, V_0, V_1, E)$ and a memory structure $M = (M, \text{init}, \text{upd})$ for $A$ 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 $\text{upd}(m, v') = m'$.
The product $A \times M = (V', V'_0, V'_1, E')$ of $A = (V, V_0, V_1, E)$ and a memory structure $M = (M, \text{init}, \text{upd})$ for $A$ 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 $\text{upd}(m, v') = m'$.

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

- $A' = A \times M$, and
- for each play $\rho$ in $A$: $\rho \in \text{Win}$ if and only if $\text{ext}(\rho) \in \text{Win'}$. 
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 $\mathcal{M} = (M, \text{init}, \text{upd})$ 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 $\text{upd}(m, v') = m'$.

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

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

**Corollary**

Let $\mathcal{G} \leq_M \mathcal{G}'$. 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 $\mathcal{G}$. 
Fix an LTL game \((\mathcal{A}, \text{LTL}(\varphi, \ell))\).

1. Construct deterministic parity automaton \(\mathcal{P}_\varphi = (Q, 2^P, q_I, \delta, \Omega)\) as before.
Fix an LTL game $(\mathcal{A}, \text{LTL}(\varphi, \ell))$.

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

2. Define memory structure $\mathcal{M} = (Q, \text{init}, \text{upd})$ with
   - $\text{init}(v) = \delta(q_I, \ell(v))$ and
   - $\text{upd}(q, v) = \delta(q, \ell(v))$. 
Reducing LTL Games to Parity Games

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

1. Construct deterministic parity automaton \\
   \(\mathcal{P}_\varphi = (Q, 2^P, q_I, \delta, \Omega)\) as before.

2. Define memory structure \(\mathcal{M} = (Q, \text{init}, \text{upd})\) with \\
   - \(\text{init}(\nu) = \delta(q_I, \ell(\nu))\) and \\
   - \(\text{upd}(q, \nu) = \delta(q, \ell(\nu))\).

3. Consider parity game \(G' = (\mathcal{A} \times \mathcal{M}, \text{PARITY}(\Omega'))\) with \\
   \(\Omega'(\nu, q) = \Omega(q)\).
Reducing LTL Games to Parity Games

Fix an LTL game \((A, \text{LTL}(\varphi, \ell))\).

1. Construct deterministic parity automaton 
   \(\mathcal{P}_\varphi = (Q, 2^P, q_I, \delta, \Omega)\) as before.

2. Define memory structure \(M = (Q, \text{init}, \text{upd})\) with
   - \(\text{init}(v) = \delta(q_I, \ell(v))\) and
   - \(\text{upd}(q, v) = \delta(q, \ell(v))\).

3. Consider parity game \(G' = (A \times M, \text{PARITY}(\Omega'))\) with
   \(\Omega'(v, q) = \Omega(q)\).

Lemma

1. \(G'\) has doubly-exponential size and exponentially many colors
   (both in \(|A| + |\varphi|\)).

2. \(G \leq_M G'\).
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 $G'$ and solve it in time

$$\left(|A| \cdot 2^{O(|\varphi|)}\right) \log(2^{O(|\varphi|)}).$$

Remark

Both lower bounds are tight.
Main Result

Theorem

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

Proof.

Construct parity game $G'$ and solve it in time

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

Remark

\textit{Both lower bounds are tight.}
Numerous variants of LTL have been considered:
- LTL with past operators: more succinct (but not more expressive!) and thus more expensive.
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 $\omega$-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 $\omega$-regular languages: (typically) same complexity.
- Extensions of LTL that capture the visibly $\omega$-contextfree languages: (typically) slightly higher complexity.
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 $\omega$-regular languages: (typically) same complexity.
- Extensions of LTL that capture the visibly $\omega$-contextfree languages: (typically) slightly higher complexity.
- Parameterized LTL with bounds on temporal operators: same complexity.
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 $\omega$-regular languages: (typically) same complexity.
- Extensions of LTL that capture the visibly $\omega$-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 \( \omega \)-regular languages: (typically) same complexity.
- Extensions of LTL that capture the visibly \( \omega \)-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.