Reactive Synthesis

Nir Piterman
University of Gothenburg
Aalborg, June 13, 2022
A function defines a relation between inputs and outputs.
Doesn’t quite work …
**Computation vs. Reactivity**

**Computational Programs:** Run in order to produce a final result on termination.

Can be modeled as a black box.

\[ x \rightarrow \text{Box} \rightarrow y \]

Specified in terms of \textit{Input/Output} relations.

**Reactive Programs**

Programs whose role is to \textit{maintain an ongoing interaction} with their environments.

Can be viewed as a \textit{green cactus} (?)
Reactive Systems

• Systems whose main aim is to interact rather than compute (OS, driver, CPU, car controller).
• Main complexity is in maintaining communication with a user / another program / the environment.
• Reactive systems are notoriously hard to design.
• Major efforts are invested in development and validation of reactive systems.
The Requirement Language

- Correctness of computational programs is expressed as Hoare triples: \( \{P\}C\{Q\} \).
- Correctness of reactive programs is expressed as behavioral specifications:
  - The behavior of a system is a sequence of system states.
  - Specification should tell us when a sequence is good/bad.
  - We use temporal logic: connect states through time.
Validating Reactive Systems

- **Simulations**: 
  - Run the system and check whether behavior satisfies specifications.

- **Model checking**: 
  - Create a comprehensive model of the system and check whether all behaviors satisfy specifications.

- **Model checking research**: 
  - Automatic construction of models.
    - Predicate extraction.
    - Heap analysis.
    - Counter-example guided abstraction refinement.
  - Techniques for model exploration.
    - Efficient enumerative graph exploration.
    - Symbolic representation of states.
    - Bounded model checking and SAT/SMT solving.
  - Specification.
    - Expressive specification languages.
    - Translation to model exploration.
Synthesis

• Developing systems is **hard**, **expensive**, and **error prone**.
• The common solution is **extensive testing** and **verification**.
• If we can **verify**, why not go directly from **specification** to **correct-by-construction** systems by **synthesis**?
• **Church’s synthesis problem:**
  Given a **circuit interface** specification and a **behavioral specification**:
  – **Determine** if there is an **automaton** that **realizes** the **specification**.
  – If the specification is **realizable**, **construct** an implementing **automaton**.
• **Circuit interface** – **partition** to **inputs** and **outputs**.
• **Behavioral specification** – description in **first order logic**.
Synthesis from Temporal Specifications

\[ \forall t. \neg o_1(t) \lor \neg o_2(t) \]
\[ \forall t. i(t) \rightarrow (\exists t' > t. o_1(t) \lor o_2(t)) \]
\[ \forall t. o_1(t) \rightarrow \left( \exists t' < t. \left( i(t') \land \forall t' < t'' < t. \left( \neg o_1(t'') \land \neg o_2(t'') \right) \right) \right) \]
\[ \forall t. o_2(t) \rightarrow \left( \exists t' < t. \left( i(t') \land \forall t' < t'' < t. \left( \neg o_1(t'') \land \neg o_2(t'') \right) \right) \right) \]
\[ \forall t. o_1(t) \rightarrow \left( \forall t' > t. \left( \neg \left( o_1(t') \lor \exists t < t'' < t'. o_2(t'') \right) \right) \right) \]
\[ \forall t. o_1(t) \rightarrow \left( \forall t' > t. \left( \neg \left( o_2(t') \lor \exists t < t'' < t'. o_1(t'') \right) \right) \right) \]

- Is it possible to realize this specification?
- The formula defines a relation between \( i: \mathbb{N} \rightarrow \{0,1\} \) and \( o_1, o_2: \mathbb{N} \rightarrow \{0,1\} \)
- We want a function that is a subset.
Causal

\[ o(0) \leftrightarrow (\exists t. i(t)) \]

• The relation \( R = \{(i, o) | i: \mathbb{N} \to \{0,1\}, o: \mathbb{N} \to \{0,1\}, o(0) \leftrightarrow (\exists t. i(t))\} \) is not empty.
• Find a function that implements it.
• The function cannot be clairvoyant.
• It needs to be causal: \( o(n) = f(i|\{0,\ldots,n\}) \)
Adversarial

\[ \forall t. i(t) \rightarrow \neg o(t) \]
\[ \forall t. i(t) \rightarrow \exists t' > t. o(t') \]

• There are \textbf{some} input sequences for which this is possible.
• But \textbf{not all}!
• We want a function that can answer \textbf{all} input sequences.
  \[ f: \{ i: \{0, \ldots, n\} \rightarrow \{0,1\} \mid n \in \mathbb{N} \} \rightarrow \{0,1\} \]
• Furthermore, for every \( i: \mathbb{N} \rightarrow \{0,1\} \) the unique \( o: \mathbb{N} \rightarrow \{0,1\} \) such that \( o(n) = f(i \mid \{0, \ldots, n\}) \) for every \( n \in \mathbb{N} \) satisfies the specification.
Brief History

• **Church**’s problem [1965].
• **Rabin** introduces **automata** on **infinite trees**. Effectively, generalizing **Büchi**’s work on **ω-automata** to trees [1969].
• **Büchi** and **Landweber** define **two-player games** of infinite duration [1969].
• We now know that the two are **effectively** the same. These are still the techniques we use to **solve** the problem.
Modern Times

- Pnueli introduces linear temporal logic [1977].
- Emerson and Clarke and Quielle and Sifakis invent model checking [1981].
- Emerson and Clarke and Manna and Wolper ignore adversarial nature and propose reduction to satisfiability [1984].
- Pnueli and Rosner establish LTL realizability to be 2EXPTIME-complete.
  – This result established realizability and synthesis as highly intractable.
In these Lectures

- **Synthesis** as a game.
- Simple games (safety, reachability, Büchi).
- **LTL Synthesis** reduced to solution of parity games.
- Bypassing determinization:
  - Safraless approach.
  - Restricting the specification language.
  - Usage of synthesis in robotics.
- Current research directions:
  - Distributed synthesis.
  - Safety of learned behaviour.
  - Strategic reasoning.
Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Current Research Directions
A More Formal Context

• A specification in linear temporal logic over input and output propositions.
• A system will be an automaton with output.
• Input and output are combined to create a sequences of assignments to propositions.
• All possible infinite paths over the automaton should satisfy the specification.
Linear Temporal Logic

• A set of propositions \( \mathcal{P} \) denoting the basic facts about the world. Set \( \mathcal{P} \) is partitioned to inputs \( \mathcal{I} \) and outputs \( \mathcal{O} \).

• Linear Temporal Logic formulae are constructed as follows:

\[
\varphi ::= \phi \mid \varphi \land \varphi \mid \neg \varphi \mid O \varphi \mid E \varphi \mid \varphi U \varphi \mid \varphi S \varphi
\]

• Other temporal formulae are derived:

- \( \lozenge \varphi \equiv T \varphi \) – Eventually.
- \( \square \varphi \equiv \neg \lozenge \neg \varphi \) – Always.
- \( \varphi W \psi \equiv \varphi U \psi \lor \square \psi \) – Weak Until.
- \( \lozenge \varphi \equiv T \varphi \) – Previously.
- \( \square \varphi \equiv \neg \lozenge \neg \varphi \) – Historically.
- \( \varphi B \psi \equiv \varphi S \psi \lor \square \psi \) – BackTo.
LTL Semantics

• A **model** for an LTL formula is an infinite sequence $\sigma = \sigma_0, \sigma_1, ...$ with a designated location $j \geq 0$.
• Each letter $\sigma_i$ is a set of propositions true at time $i$.
• Formula $\varphi$ holds over sequence $\sigma$ in location $i \geq 0$, denoted $(\sigma, i) \models \varphi$, if:
  – If $\varphi$ is a proposition $(\sigma, i) \models \varphi \iff \varphi \in \sigma_i$
  – $(\sigma, i) \models \neg \varphi \iff (\sigma, i) \not\models \varphi$
  – $(\sigma, i) \models \varphi_1 \lor \varphi_2 \iff (\sigma, i) \models \varphi_1 \text{ or } (\sigma, i) \models \varphi_2$
  – $(\sigma, i) \models \Diamond \varphi \iff (\sigma, i + 1) \models \varphi$
  – $(\sigma, i) \models \Box \varphi \iff i > 0 \text{ and } (\sigma, i - 1) \models \varphi$
  – $(\sigma, i) \models \varphi_1 U \varphi_2 \iff \exists k \geq i. (\sigma, k) \models \varphi_2 \text{ and } \forall i \leq j < k. (\sigma, j) \models \varphi_1$
  – $(\sigma, i) \models \varphi_1 S \varphi_2 \iff \exists k \leq i. (\sigma, k) \models \varphi_2 \text{ and } \forall i \geq j > k. (\sigma, j) \models \varphi_1$
• Derived:
  – $(\sigma, i) \models \Diamond \varphi \iff \exists k \geq i. (\sigma, k) \models \varphi$
  – $(\sigma, i) \models \Box \varphi \iff \forall k \geq i. (\sigma, k) \models \varphi$
LTL Exercises

□ p

□ □ p

□ (p → □ (q U r))

□ (p → p ⋀ q) \equiv □ (p → (□ p ∨ □ q))

p \equiv □ (□ T ∨ p)

□ (p → □ q)

□ (p → □ (¬p S q))

□ (¬□ T ∧ p)

□ (p → □ q) \equiv □ □ □ (¬q S p)

(p U (q U r)) \neq ((p U q) U r)
Automata

• Systems with **discrete states**.
• Formally, $A = \langle \Sigma, Q, \delta, q_0 \rangle$, where
  – $\Sigma$ – a **finite input alphabet**.
  – $Q$ – a **finite set of states**.
  – $\delta: Q \times \Sigma \rightarrow 2^Q$ – a **transition function**. Associates with **state** and an **input letter** a set of **successor states**.
  – $q_0$ – an **initial state**.
• An **input word** $w = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from $\Sigma$.
• A **run** $r = q_0, q_1, \ldots$ over $w$ is a sequence of states starting from $q_0$ such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
• An automaton is **deterministic** if for every $q \in Q$ and $\sigma \in \Sigma$ we have $|\delta(q, \sigma)| \leq 1$. 
Mealy Machines

• Systems with discrete states.
• Formally, $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, where
  – $\Sigma$ – a finite input alphabet.
  – $\Delta$ – a finite output alphabet.
  – $Q$ – a finite set of states.
  – $\delta: Q \times \Sigma \to 2^Q$ – a transition function. Associates with every state and an input letter a set of successor states.
  – $q_0$ – an initial state.
  – $L: Q \times \Sigma \to \Delta$ – an output function. Associates with every transition an output letter.
• A run $r = q_0, q_1, \ldots$ over $w$ is a sequence of states starting from $q_0$ such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
• The computation corresponding to $r = q_0, q_1, \ldots$ over $w$ is $c = (\sigma_0, L(q_0, \sigma_0)), (\sigma_1, L(q_1, \sigma_1)), \ldots$. 
Mealy Machines and LTL

• The set of computations of a machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$ is denoted $\mathcal{L}(M)$.
• Assume $\Sigma = 2^I$ and $\Delta = 2^O$. So input letters are assignments to input propositions and outputs are assignments to output propositions.
• A machine $M$ satisfies a formula $\varphi$, denoted $M \models \varphi$, if every computation in $\mathcal{L}(M)$ satisfies $\varphi$.
• Given an LTL formula $\varphi$ over propositions $\mathcal{P}rop = I \cup O$ we say that $\varphi$ is realizable if there is a Mealy machine that satisfies it.
• Our task is going to be to find such a Mealy machine or say that it does not exist.
• We will mostly be interested in deterministic machines.
Bibliography

Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Current Research Directions
Realizability

• So, given a property $\varphi$ and a partition $\text{Prop} = I \cup O$ find a system $M$ such that $M \models \varphi$.
• For every possible input, decide on an output …
• All paths through the machine should satisfy the property.
Arbiter

\[
\begin{align*}
&\text{Arbiter} \\
&\quad r_1 \\
&\quad r_2 \\
&\quad \ldots \\
&\quad r_n \\
&\quad g_1 \\
&\quad g_2 \\
&\quad \ldots \\
&\quad g_n \\
\end{align*}
\]

Client

\[
\begin{align*}
&\text{Client} \\
&\quad r_i \\
&\quad \overline{r_i}, \overline{g_i} \\
&\quad r_i, \overline{g_i} \\
&\quad \overline{r_i}, g_i \\
&\quad r_i, g_i \\
&\quad g_i \\
\end{align*}
\]
Arbiter$^2$

• Propositions $Prop = \{r_1, r_2, g_1, g_2\}$, where $I = \{r_1, r_2\}$ and $O = \{g_1, g_2\}$.

• Requirements:
  – $A_1$: leave requests: $\Box (r_1 \land !g_1 \rightarrow \Diamond r_1) \land \Box (r_2 \land !g_2 \rightarrow \Diamond r_2)$
  – $G_1$: leave grants: $\Box (r_1 \land g_1 \rightarrow \Diamond g_1) \land \Box (r_2 \land g_2 \rightarrow \Diamond g_2)$
  – $G_2$: mutual exclusion: $\Box (!g_1 \lor !g_2)$
  – $G_3$: deliver and remove grants: $\Box \Diamond (g_1 \leftrightarrow r_1) \land \Box \Diamond (g_2 \leftrightarrow r_2)$

• Or together: $A_1 \rightarrow (G_1 \land G_2 \land G_3)$
What’s the idea?

• Think about control:
  – Some things are under our control.
  – Some things are not.
• We want to exercise our control so that to achieve certain goals.
• In some cases the environment is hostile.
• What we want:
  – Find a strategy that will guide our actions based on our view of the world.
• This leads to viewing the world as an opponent:
  – Exercise control so that uncontrollable events do not lead to damage.
• We model this as two-player games.
Example: Nim

- Some rows of matches.
- Every player removes in turn at least one match from one row.
- The one to remove last match wins.
- Can you win?
Whose in Control?

- We use **graphs** with **vertices** for **states** and **edges** for **transitions**.
- **Ownership** is by using **two types of vertices**.

![Graph with vertices and edges]
A Play
Arbiter

\[ r_1 \rightarrow g_1 \]
\[ r_2 \rightarrow g_2 \]
\[ \ldots \]
\[ r_n \rightarrow g_n \]

Client

\[ r_i \rightarrow r_i, \overline{g_i} \]
\[ r_i, \overline{g_i} \rightarrow \overline{r_i, g_i} \]
\[ \overline{r_i, g_i} \rightarrow r_i, g_i \]
\[ r_i, g_i \rightarrow g_i \]
Games

- Formally, a game is $G = \langle V, V_0, V_1, E, \alpha \rangle$, where:
  - $V$ is a set of nodes.
  - $V_0$ and $V_1$ form a partition of $V$.
  - $E \subseteq V \times V$ is a set of edges.
- A play is $\pi = v_0, v_1, \ldots$ where $\alpha$ is a set of winning plays.
- A strategy for player $i$ is a function $f_i : V^* \cdot V_i \rightarrow V$ such that $(v, f_i(w \cdot v)) \in E$.
- A play $\pi = v_0, v_1, \ldots$ is compatible with $f_i$ if for every $j \geq 0$ such that $v_j \in V_i$ we have $v_{j+1} = f_i(v_0 \cdots v_j)$.
- A strategy for player 0 is winning if every play compatible with it is in $\alpha$. A strategy for player 1 is winning if every play compatible with it is not in $\alpha$.
- A node $v$ is won by player $i$ if she has a winning strategy for all plays starting from $v$.

From now on we mostly care about player 0!
• In control it is easier to walk backwards.
Control Predecessor (for P0)

• Start from an set of nodes $W \subseteq V$.
• We want to say:
  – The system can force the environment to $W$ in one move.
• That is:
  – Nodes $v \in V_0$ for which some successor is in $W$.
  – Nodes $v \in V_1$ for which all successors are in $W$.
• Formally:

$$
cpre(W) = \{ v \in V_0 \mid \exists v' \in W. (v, v') \in E \} \cup \{ v \in V_1 \mid \forall v'. (v, v') \in E \rightarrow v' \in W \} 
$$
Control Predecessor (for P1)

- Start from an set of nodes $W \subseteq V$.
- We want to say:
  - The environment can force the system to $W$ in one move.
- That is:
  - Nodes $v \in V_1$ for which some successor is in $W$.
  - Nodes $v \in V_0$ for which all successors are in $W$.
- Formally:

  $$\text{cpre}_1(W) = \{ v \in V_1 \mid \exists v' \in W. (v, v') \in E \} \cup \{ v \in V_0 \mid \forall v'. (v, v') \in E \rightarrow v' \in W \}$$
Let’s solve some games!
Safety Games

• Check that $P_0$ can enforce $\Box p$.

1. $\text{fix } (\text{new} := p)$
2. $\text{new} := \text{new} \land cpre(\text{new})$
3. $\text{end } // \text{fix}$

Lemma. The algorithm computes the set of states winning for $P_0$ with objective $\Box p$.

Proof. Later.
Reachability Games

• Check that $P1$ can enforce $\lozenge \neg p$.

1. $\text{fix (new := } \neg p\text{)}$
2. $\text{new := new } \lor \text{ cpre}_1(\text{new})$
3. $\text{end } // \text{ fix}$

**Lemma.** The algorithm computes the set of states winning for $P1$ with objective $\lozenge p$.

**Proof.** Later.

$Attr_i(W)$ the set of nodes from which player $i$ can force reaching $W$. 
Safety vs Reachability Games

• Goals $\square p$ for $P_0$ and $\Diamond \neg p$ for $P_1$ are complementary.

1. fix $\text{new} := p$
2. $\text{new} := \text{new} \land cpre(\text{new})$
3. end // fix

1. fix $\text{new} := \neg p$
2. $\text{new} := \text{new} \lor cpre_1(\text{new})$
3. end // fix
Safety Games

• Check that $P_0$ can enforce $\square p$.

1. fix (new := p)
2. new := new $\land$ cpre(new)
3. end // fix
Proof

- Suppose that \texttt{new} is not empty. Consider \( v \in \texttt{new} \). Clearly, \( v \in p \). But also \( v \in \texttt{cpre(new)} \).
  If \( v \in \mathcal{V}_0 \), then \( v \) has a successor \( w \) such that \( w \in \texttt{new} \).
  If \( v \in \mathcal{V}_1 \), then for every successor \( w \) of \( v \) we know \( w \in \texttt{new} \).
- If there is a strategy s.t. every play compliant with it wins \( \square p \).

Let \( \texttt{new}_0, \texttt{new}_1, \texttt{new}_2, \ldots \) be the series of approximations of \( \texttt{new} \). We prove by induction that for every \( v \) winning for \( P_0 \), \( v \in \texttt{new}_i \) for every \( i \).

Clearly, \( v \in p \) implies \( v \in \texttt{new}_0 \).

Assume every \( v \) winning for \( P_0 \) is in \( \texttt{new}_i \) for some \( i \). Consider \( v \in \mathcal{V}_0 \) winning for \( P_0 \). Then, there is \( w \) such that \( (v, w) \in E \) and \( w \) winning for \( P_0 \). Then, \( w \) in \( \texttt{new}_i \) and \( v \) in \( \texttt{new}_{i+1} \). Consider \( v \in \mathcal{V}_1 \) winning for \( P_0 \). Then, for every \( w \) such that \( (v, w) \in E \) we have \( w \) winning for \( P_0 \). Then, every \( w \) such that \( (v, w) \in E \) is in \( \texttt{new}_i \).

So \( v \) in \( \texttt{new}_{i+1} \).

---

1. \texttt{fix (new := p)}
2. \texttt{new := new \land cpre(new)}
3. \texttt{end // fix}
Lecture 2: Games and Synthesis

Reactive Synthesis

MOVEP Summer School, Aalborg, 2022
Lecture 2: Games and Synthesis

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Lecture 2: Games and Synthesis

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Büchi Games

• Check that P0 can enforce $\Box \Diamond p$.

1. fix (greatest := $V$)
2. fix (least := $p \land \text{cpre}(\text{greatest})$
3. least := least $\lor \text{cpre}(\text{least})$
4. end // fix least
5. greatest := least;
6. end // fix greatest

Lemma. The algorithm computes the set of nodes winning for P0 with objective $\Box \Diamond p$. 
Büchi Games

- Check that $P0$ can enforce $\square \Diamond p$.

1. fix (greatest := $V$)
2. fix (least := $p \land cpre$(greatest))
3. least := least $\lor cpre$(least);
4. end // fix least
5. greatest := least;
6. end // fix greatest
Proof (Control of Büchi – Soundness)

• Suppose that greatest is not empty. For the fixpoint to terminate, the inner fixpoint starting from this value recomputes it.

• Let least_0, least_1, least_2, … be the sequence of values that least has through the computation of this last iteration.

• Consider v ∈ greatest. Let i_0 be the index such that v ∈ least_i_0. By definition of cpre(·), P0 can force a successor w of v. But then, w ∈ least_i_1 for some i_1 < i_0.

• This shows that P0 can ensure to reach least_0 = p ∧ cpre(greatest). So it ensures a visit p.

• But now least_0 = p ∧ cpre(greatest). So in the next step P0 forces least_j for some j and repeat this process.

• P0 can enforce □□ p.
Proof (Control of Büchi - completeness)

• If there is a strategy $f$ s.t. every play compliant with it wins $\Box\Diamond p$.

• Every node $v$ from which $f$ is winning remains in every approximation of the fixpoint $\text{greatest}$:
  – From $v$ there is a maximum on the length of paths to reach $p$ (König’s lemma).
  – Prove by induction on the number of iterations in the first fixpoint that $\text{win} \subseteq \text{greatest}$.
  – For $\text{greatest}_0 = V$ this is clear.
  – Assume $\text{win} \subseteq \text{greatest}_i$. Then for every node $v \in \text{win}$ it must be that $v \in \text{least}_j$ for the distance to reach $p \land \text{win}$.

1. $\text{fix (greatest} := V)$
2. $\text{fix (least} := p \land \text{cpre(greatest)}$
3. $\text{least} := \text{least } \lor \text{cpre(least)}$
4. $\text{end // fix least}$
5. $\text{greatest} := \text{least}$
6. $\text{end // fix greatest}$
Strategy

• A strategy is the way of enforcing the goal.
• Let $D$ be some memory domain and let $d_0$ be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
• A strategy for player $i$ is a function $f_i: V^* \cdot V_0 \to V$ such that $(v, f_i(w \cdot v)) \in E$.
• We look to replace $V^*$ by some (finite) domain $D$. Then, instead of considering $V$ we could consider $D \times V$.
• The strategy is replaced by two functions:
  – Move function: $f_i^m: D \times V_i \to V$ s.t. $(v, f(d, v)) \in E$.
  – Update function: $f_i^u: D \times V \to D$. 
What about Synthesis?

• Our goal is to construct a Mealy machine that realizes the specification.
  – A Mealy machine from every state reads input and answers with output.
• A node in the game corresponding to choice of input will be followed by node corresponding to choice of output.
• We can define a specialized game with nodes in $2^{\mathcal{I} \cup \mathcal{O}}$.
• We can define the winning condition with an LTL formula over $\mathcal{I} \cup \mathcal{O}$. A play naturally corresponds to a possible model.
• For a set of nodes $W$, define
  $$cpre(W) = \{ v \mid \forall x \in 2^\mathcal{I}. \exists y \in 2^\mathcal{O}. (x \cup y) \in W \}$$
• When computing the set of winning states, check that for every $x \in 2^\mathcal{I}$ there is $y \in 2^\mathcal{O}$ such that $x \cup y$ is winning.
Further Specialize Strategy

- Let $D$ be some memory domain and let $d_0$ be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
- A strategy for player $i$ is a function $f_i : V^* \cdot V_0 \rightarrow V$ such that $(v, f_i(w \cdot v)) \in E$.
- We look to replace $V^*$ by some (finite) domain $D$. Then, instead of considering $V$ we could consider $D \times V$.
- The strategy is replaced by two functions:
  - Move function: $f_i^m : D \times V_i \rightarrow V$ s.t. $(v, f(d, v)) \in E$.
  - Update function: $f_i^u : D \times V \rightarrow D$. 
Further Specialize Strategy

• Let \( D \) be some memory domain and let \( d_0 \) be an initial memory value. Elements in the memory domain recall facts about the history of play so far.

• A strategy for player \( i \) is a function \( f_i : (2^{J \cup O})^* \cdot 2^J \rightarrow 2^O \).

• We look to replace \( V^* \) by some (finite) domain \( D \). Then, instead of considering \( V \) we could consider \( D \times V \).

• The strategy is replaced by two functions:
  – Move function: \( f_i^m : D \times V_i \rightarrow V \) s.t. \( (v, f(d, v)) \in E \).
  – Update function: \( f_i^u : D \times V \rightarrow D \).
Further Specialize Strategy

• Let $D$ be some memory domain and let $d_0$ be an initial memory value. Elements in the memory domain recall facts about the history of play so far.

• A strategy for player $i$ is a function $f_i : (2^{J \cup O})^* \cdot 2^J \rightarrow 2^O$.

• We look to replace $(2^{J \cup O})^*$ by some (finite) domain $D$. Then, instead of considering $(2^{J \cup O})^*$ we could consider $D \times 2^{J \cup O}$.

• The strategy is replaced by two functions:
  – Move function: $f_i^m : D \times V_i \rightarrow V$ s.t. $(v, f(d, v)) \in E$.
  – Update function: $f_i^u : D \times V \rightarrow D$. 
Further Specialize Strategy

• Let $D$ be some memory domain and let $d_0$ be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
• A strategy for player $i$ is a function $f_i : (2^{I \cup O})^* \cdot 2^I \rightarrow 2^O$.
• We look to replace $(2^{I \cup O})^*$ by some (finite) domain $D$. Then, instead of considering $(2^{I \cup O})^*$ we could consider $D \times 2^{I \cup O}$.
• The strategy becomes $f_i : D \times 2^I \rightarrow D \times 2^O$. 
From Strategy to System

Consider a strategy $f_0 : D \times 2^j \rightarrow D \times 2^o$ and let $d_0 \in D$ be the initial memory value. Construct the machine $M = \langle \Sigma, \Delta, D, \delta, d_0, L \rangle$ with:

- $\Sigma = 2^j$
- $\Delta = 2^o$
- $\delta(d, i) = f_0(d, i) \Downarrow_1$
- $L(d, i) = f_0(d, i) \Downarrow_2$

What’s the memory domain in the cases we’ve seen?
Winning \rightarrow \text{Realizability}

Consider a run \( r = q_0, q_1, \ldots \) over \( w = \sigma_0, \sigma_1, \ldots \) and the corresponding computation \( c = (\sigma_0, L(q_0, \sigma_0)), (\sigma_1, L(q_1, \sigma_1)), \ldots \) of \( M \).

i. For every \( i \in 2^j \) there is \( o \in 2^o \) s.t. \((i, o)\) is winning.

ii. By \( f \) winning \( c \) satisfies the formula.

Realizability \rightarrow \text{Winning}

Take a machine \( M \) and use it to construct the winning strategy.
A play in the game is a computation of the machine.
Memorize Intermediate Values

1. fix (greatest := \( V \))
2. fix (least := \( p \land cpref(greatest) \))
3. least := least \( \lor \) cpref(least)
4. end // fix least
5. greatest := least
6. end // fix greatest

1. fix (greatest := \( V \))
2. \( cY := 0 \);
3. fix (least := \( p \land cpref(greatest) \))
4. \( y[cY] := \) least;
5. least := least \( \lor \) cpref(least)
6. \( cY := cY + 1 \);
7. end // fix least
8. greatest := least
9. end // fix greatest
Construct the Realizing Machine

• Given \( G = \langle 2^{\mathcal{J} \cup \mathcal{O}} \cup (2^{\mathcal{J} \cup \mathcal{O}} \times 2^{\mathcal{I}}), 2^{\mathcal{J} \cup \mathcal{O}} \times 2^{\mathcal{I}}, 2^{\mathcal{J} \cup \mathcal{O}}, E, \square \diamond p \rangle \).
  \[ E = \{((i, o), (i, o, i')) , ((i, o, i'), (i', o'))\}\]

• Construct a \( M = \langle 2^{\mathcal{I}} , 2^{\mathcal{O}} , 2^{\mathcal{J} \cup \mathcal{O}} , \delta , s_0 , L \rangle \):
  \[ \delta((i, o), i') = \begin{cases} \{ (i', o') \mid (i', o') \text{ is winning} \} & (i, o) \in p \\ \{ (i', o') \mid (i', o') \in y[ \leq j] \} & (i, o) \in y[j + 1] \end{cases} \]
Summary

• Starting from an LTL formula $\varphi$, construct the game $G = \langle 2^{\text{I}} \cup (2^{\text{I}} \times 2^\text{O}), 2^{\text{I}} \times 2^\text{O}, 2^{\text{I}} \cup 2^\text{O}, E, \varphi \rangle$.
• Compute the set $\text{win}$.
• If for every $i \in 2^\text{I}$ there is $o \in 2^\text{O}$ such that $(i, o) \in \text{win}$ then declare $\varphi$ realizable.
• Extract from the winning strategy a realizing Machine.

• But we only know to solve reachability/safety and Büchi games.
• What about general LTL?
Bibliography

Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Current Research Directions
From Logic to Graphs?

How to embed the logical winning condition into the graph notation?
• Systems with discrete states.
• Formally, $A = (\Sigma, Q, \delta, q_0, \alpha)$, where
  – $\Sigma$ – a finite input alphabet.
  – $Q$ – a finite set of states.
  – $\delta: Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with state and input letter a set of successor states.
  – $q_0$ – an initial state.
  – $\alpha \subseteq Q$ – a set of accepting states.
• An input word $w = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from $\Sigma$.
• A run $r = q_0, q_1, \ldots$ over $w$ is a sequence of states starting from $q_0$ such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
• A run is accepting if for infinitely many $i \in \mathbb{N}$ we have $q_i \in \alpha$.
• A word is accepted if some run over it is accepting.
• The language of $A$, denoted $\mathcal{L}(A)$, is the set of words accepted by $A$. 
From LTL to Büchi Automata

**Theorem.** Given an LTL formula $\varphi$ we can construct a **nondeterministic Büchi automaton** $N_\varphi$ such that $\mathcal{L}(N_\varphi) = \mathcal{L}(\varphi)$.

The size of $N_\varphi$ is exponential in the length of $\varphi$.

Intuitively, if $\text{sub}(\varphi)$ is the set of **subformulas** of $\varphi$, a state of $N_\varphi$ corresponds to a **set** of subformulas that are true (in an accepting run).
Control with Automaton Observer

Visit finitely many not-\(p\)'s \(\Diamond \square \neg p\)

- Environment
- System

Diagram:
```
  p     !p     p
     ↘    ↗     ↘
     p     !p   !p
```

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
NBW for $\Diamond \Box p$

- NBW for $\varphi = \Diamond \Box p$:
Nondeterminism is bad
What went wrong?

• The automaton is **nondeterministic**.
• It makes **predictions** regarding the **future** and aborts runs that do not match these predictions.
• In the context of **games** **nondeterminism** is added as choice of one side:
  – If the **system** resolves **nondeterminism**, it has to find a solution that matches all possible futures.
  – If the **environment** resolves **nondeterminism**, the system must force all runs to be accepting.
Solution: Determinism

- If the automaton were deterministic, there would be no added choice!
- We create a synchronous parallel composition of the automaton with the game.
- Solve the resulting game.
- Extract system from winning strategy.
Automata as Acceptors

- Systems with **discrete states**.
- Formally, $A = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where
  - $\Sigma$ – a **finite input alphabet**.
  - $Q$ – a **finite set of states**.
  - $\delta: Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with state and input letter a set of successor states.
  - $q_0$ – an **initial state**.
  - $\alpha: Q \rightarrow \mathbb{N}$ – a ranking of **states**.
- An **input word** $w = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from $\Sigma$.
- A **run** $r = q_0, q_1, \ldots$ over $w$ is a sequence of states starting from $q_0$ such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
- A **run** is **accepting** if for the minimum rank to occur infinitely often is **even**.
- The **language** of $A$, denoted $\mathcal{L}(M)$, is the set of words accepted by $A$. 

Nondeterministic parity Automata
Synchronous Composition of Games

• Consider a game $G = \langle V, V_0, V_1, E, \varphi \rangle$ and a deterministic (with respect to entire alphabet $\Sigma$) automaton $A_\varphi = \langle \Sigma, D, \delta, d_0, \beta \rangle$.

• Their synchronous parallel composition $(G \parallel A_\varphi)$ is the game, 
  $$\hat{G} = \langle \hat{V}, \hat{V}_0, \hat{V}_1, \hat{E}, \gamma \rangle$$ where:
  – $\hat{V} = D \times V$ – a new node holds a game node and an automaton state.
  – $\hat{E} = \{ (d, v), (d', v') \mid (v, v') \in E \text{ and } d' = \delta(d, L(v)) \}$ – the transitions of the automaton are updated.
  – $\gamma(d, v) = \beta(d)$ – acceptance only considers the acceptance of the automaton.

• The results is a parity game.
Deterministic Automata Work!

Theorem. \( P_0 \) wins \( G \) with winning condition \( \varphi \) iff \( P_0 \) wins \( G \parallel A_\varphi \), where \( A_\varphi \) is a deterministic automaton for \( \varphi \).

⇒ If \( P_0 \) wins \( G \) all she has to do in \( G \parallel A_\varphi \) is to use the same strategy. Every play in \( G \parallel A_\varphi \) corresponds to a play in \( G \) and the unique run of \( A_\varphi \) that reads this play. But the play satisfies \( \varphi \), so the run must be accepting. So the play in \( G \parallel A_\varphi \) is winning for \( P_0 \) as well.

⇐ If \( P_0 \) wins \( G \parallel A_\varphi \) she can use the states of \( A_\varphi \) as (part of) the memory in \( G \). She will then be able to use the winning strategy from \( G \parallel A_\varphi \). Now, a play in \( G \) corresponds to an accepting run of \( A_\varphi \). But then the play satisfies \( \varphi \), which means that \( P_0 \) wins.
Two tiny issues ...

• How do we get a deterministic parity automata for LTL?
• How do we solve a parity games?
Deterministic Automata

• Well, the answer is simple: construct a **nondeterministic automaton** and **determinize** it!
• Starting from an automaton with \( n \) states:
  – Create an automaton with \( O((n!)^2) \) states and \( 2n \) rank.
• **Subset construction** augmented with a **tree structure**. Will not be shown.
Solving parity Games

Func main()
1. Return even_parity(0, ∅);
End // Func main

Func even_parity(i, win)
1. fix (greatest := V)
2. greatest := win V {v | α(v) = i} ∧ cpre(greatest)
3. if (i!=max)
4. greatest := odd_parity(i+1, greatest)
5. end // fix greatest
6. Return greatest;
End // Func even_parity

Func odd_parity(i, win)
1. fix (least := ∅)
2. least := win V {v | α(v) ≥ i} ∧ cpre(least)
3. if (i!=max)
4. least := even_parity(i+1, least)
5. end // fix least
6. Return least;
End // Func odd_parity
Proof (Soundness)

- Suppose that \( \text{win} \) is not empty. Have the intermediate least fixpoint approximations: \( \text{least}_0^p, \text{least}_1^p, \text{least}_2^p, \ldots \) for an odd parity \( p \).

- Consider \( v \in \text{win} \). Let \( i_1, i_3, \ldots, i_m \) be the indices such that \( v \in \text{least}_{i_j}^j \). By definition of \( cpre(\cdot) \), \( P_0 \) can force a successor \( w \) of \( v \). But then, either (a) for some even \( j \) we have \( v \in \alpha(j) \) and \( w \) has \( i_1', i_3', \ldots, i_m' \) such that for \( j' < j \) we have \( i_{j'} \leq i_j' \), or (b) there is some \( j \) such that \( w \) has \( i_1', i_3', \ldots, i_m' \), for \( j' < j \) we have \( i_j' = i_{j'} \), and for \( j \) we have \( i_j' < i_j \).

- Consider an infinite path and what happens to these numbers. There must be an even priority that is “reset” infinitely often, showing that \( P_0 \) wins.

```
Func oddParity(i, win)
1.   fix (least := ∅)
2.   least := win ∨ (\{v | α(v) ≥ i \} ∧ cpre(least))
3.   if (i!=max)
4.     least := evenParity(i+1, least)
5. end // fix least
6. Return least;
End // Func oddParity

Func evenParity(i, win)
1.   fix (greatest := V)
2.   greatest := win ∨ (\{v | α(v) = i \} ∧ cpre(greatest))
3.   if (i!=max)
4.     greatest := oddParity(i+1, greatest)
5. end // fix greatest
6. Return greatest;
End // Func evenParity
```
To Summarize

- Start with a game structure $G$ with winning condition $\varphi$.
- Construct a deterministic automaton $A_\varphi$ for $\varphi$.
- Construct the product $G \parallel A_\varphi$.
- Solve the game $G \parallel A_\varphi$.
- Construct a winning strategy for $G \parallel A_\varphi$.
- Construct from the winning strategy a Mealy machine realizing $\varphi$.

The problem is $2\text{EXPTIME}$-complete.

- Determinization is an issue.
- Practical solutions of parity games.
Bibliography


Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Current Research Directions
Two Ways to Avoid Determinization

• Replace by counting:
  – Search for bounded strategy.
  – Express winning through safety games.
  – Limited determinization through counting.
  – Translate to an SMT problem.

• Concentrate on simpler specifications:
  – Both system and environment are Büchi automata.
  – Enforce “deterministic” specification.
  – State-space exponential. Exponent linear.

- Search for bounded strategy.
- Express winning through safety games.
- Limited determinization through counting.
- Translate to an SMT problem.
- Concentrate on simpler specifications:
  - Both system and environment are Büchi automata.
  - Enforce “deterministic” specification.
  - State-space exponential. Exponent linear.
The Automata Theoretic Approach to LTL Model Checking

• Given a Mealy machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, $M$ satisfies a formula $\varphi$, denoted $M \models \varphi$, if every computation in $\mathcal{L}(M)$ satisfies $\varphi$.
• Dually, $M$ satisfies a formula $\varphi$ if no computation in $\mathcal{L}(M)$ satisfies $\neg \varphi$.
• Use automata for model checking:
  – Construct a nondet Büchi automaton $N_{\neg \varphi}$ such that $\mathcal{L}(N_{\varphi}) = (\Sigma \times \Delta)^\omega \setminus \mathcal{L}(\varphi)$.
  – Take the product of $M$ and $N_{\neg \varphi}$ as a nondet Büchi automaton.
  – If $M \times N_{\neg \varphi}$ accepts some word, the word corresponds to a computation in $\mathcal{L}(M)$ not satisfying $\varphi$.
• Our goal:
  – Find a Mealy machine $M$ and show that $M \times N_{\neg \varphi}$ is empty.
Nondeterministic Büchi Automata

• Systems with discrete states.
• Formally, $A = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where
  – $\Sigma$ – a finite input alphabet.
  – $Q$ – a finite set of states.
  – $\delta: Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with state and an input letter a set of successor states.
  – $q_0$ – an initial state.
  – $\alpha \subseteq Q$ – a set of accepting states.
• An input word $w = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from $\Sigma$.
• A run $r = q_0, q_1, \ldots$ over $w$ is a sequence of states starting from $q_0$ such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
• A run is accepting if for infinitely many $i \in \mathbb{N}$ we have $q_i \in \alpha$.
• A word is accepted if some run over it is accepting.
• The language of $A$, denoted $\mathcal{L}(A)$, is the set of words accepted by $A$. 
LTL Model Checking

**Theorem.** Given an LTL formula $\varphi$ over propositions $I \cup O$ we can construct a **nondet Büchi automaton** $N_{\neg \varphi}$ over alphabet $2^{I \cup O}$ such that $\mathcal{L}(N_{\neg \varphi}) = (2^{I \cup O})^\omega \setminus \mathcal{L}(\varphi)$.

- We have:
  - Mealy machine $M = \langle 2^I, 2^I, Q, \delta, q_0, L \rangle$
  - Büchi automaton $N_{\neg \varphi} = \langle 2^{I \cup O}, S, \rho, s_0, \alpha \rangle$

- Construct:
  - $M \times N_{\neg \varphi} = \langle 2^{I \cup O}, Q \times S, \delta', (q_0, s_0), Q \times \alpha \rangle$, where
    \[ \delta'( (q, s), (i, o) ) = \{(q', s') | \delta(s, i) = s', L(s, i) = o, \text{ and } q' \in \rho(q, (i, o)) \} \]
  - An accepting run $r = (q_0, s_0), (q_1, s_1), \ldots$ on word $w = \sigma_0, \sigma_1, \ldots$ is exactly a computation of $M$ accepted by $N_{\neg \varphi}$.
  - But we are interested in the case that $M \models \varphi$ …
Analyze the Graph

• Assume that $M \times N_{\neg \varphi} = \langle 2^{\mathbb{J} \cup \mathbb{O}}, Q \times S, \delta', (q_0, s_0), Q \times \alpha \rangle$ is empty ($M \models \varphi$).

• Every run of $M \times N_{\neg \varphi}$ contains finitely many accepting states in $Q \times \alpha$.

• But how many?
  – Think about $M \times N_{\neg \varphi}$ as a graph.
  – If there are more than $|\alpha| \cdot |S|$ accepting states on a path then this is an accepting loop.
  – Create a proof that $M \times N_{\neg \varphi}$ is empty by adding a function $f: Q \times S \to \mathbb{N}$ such that:
    • $f(q_0, s_0) = |\alpha| \cdot |S|$
    • If for some $(i, o)$ we have $(q', s') \in \delta'((q, s), (i, o))$ then:
      – If $s \in \alpha$ then $f(q, s) > f(q', s')$.
      – If $s \notin \alpha$ then $f(q, s) \geq f(q', s')$. 
Bounded Synthesis

• Remember, given \( \varphi \) (and \( N_\varphi = (2^{\|O\|}, S, \rho, s_0, \alpha) \)) we want a machine \( M \) s.t. \( M \models \varphi \).
• What if we search for a machine with at most \( m \) states?
• We can just "nondeterministically guess" its structure along with the proof that it satisfies \( \varphi \).

• Create an SMT instance \( \Gamma \):
  – Variables encoding transitions:
    For \( j \in \{1, \ldots, m\} \) and \( \sigma \in 2^J \) have \( tr_{j,\sigma} \in \{1, \ldots, m\} \).
  – Variables encoding outputs:
    For \( j \in \{1, \ldots, m\} \) and \( \sigma \in 2^J \) have \( l_{j,\sigma} \in 2^O \).
  – Variables encoding Büchi proof:
    For \( j \in \{1, \ldots, m\} \) and \( s \in S \) have \( f_{j,s} \in \{0, \ldots, m \cdot |S|, T\} \) (\( T > T \) and for all \( k, T > k \)).
  – Add constraints:
    \[ f_{0,s_0} \neq T \]
    If \( s' \in \rho(s, \sigma, l_{j,\sigma}) \) and \( s \in \alpha \) then \( f_{j,s} > f_{tr_{j,\sigma},s'} \).
    If \( s' \in \rho(s, \sigma, l_{j,\sigma}) \) and \( s \notin \alpha \) then \( f_{j,s} \geq f_{tr_{j,\sigma},s'} \).

• If \( \Gamma \) is satisfiable there exists a machine of size at most \( m \) realizing \( \varphi \) and it can be extracted from the satisfying assignment.
Advantages

• **Simple** structure of states.
  – Replace the tree structure over sets of states by a function from states to ranks.
  – Determinization is a challenge for implementation.

• **Safety** games compared with **parity** games.
  – Solution of safety games is much simpler.
  – Exact complexity and practical solving of parity games are interesting open problems.

• Search for **small machines** first.
  – By increasing the bound gradually we can ensure to find small implementations first (and compute less).
  – Information from failed search for small sizes can be reused for searching for larger sizes.
  – Worst case complexity is as the general technique.

• Add **additional quality constraints**.
  – Low number of loops …
Take Another Look at Machines

- A machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, where
  - $\Sigma = 2^I$ – a finite input alphabet.
  - $\Delta = 2^O$ – a finite output alphabet.
  - $Q = 2^X$ – a finite set of states.
- Express as an LTL formula over $I \cup O \cup X$:
  - $q_0$:
    $$ \theta = \forall x \in 2^I (x, L(q_0, x)) \land \delta(q_0, x) $$
  - $\delta$: $Q \times \Sigma \rightarrow 2^Q$:
    $$ \rho = \left( \land_{q \in Q, x \in 2^I} (q \land \Box x \rightarrow \bigcirc L(q, x) \lor \forall q \in \delta(q, \sigma) \bigcirc q) \right) $$
- We may want to add some “good things” happen often enough:
  $$ \land_i \square \Diamond (\forall q \in G_i \ q) $$
- Overall:
  $$ \theta \land \Box \rho \land \land_i \Box \Diamond (\forall q \in G_i \ q) $$
Arbiter

Arbiter

Client

Client

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Translate to LTL

- Variables:
  \[ I = \{r_1, r_2\} \]
  \[ O = \{g_1, g_2\} \]

- Initially:
  \[ \neg r_1 \land \neg r_2 \land \neg g_1 \land \neg g_2 \]

- Transition:
  \[ (r_1 \land \neg g_1) \rightarrow O r_1 \]
  \[ (\neg r_1 \land g_1) \rightarrow O \neg r_1 \]
  \[ (r_2 \land \neg g_2) \rightarrow O r_2 \]
  \[ (\neg r_2 \land g_2) \rightarrow O \neg r_2 \]
  \[ \neg g_1 \lor \neg g_2 \]
  \[ (g_1 \leftrightarrow O r_1) \rightarrow (g_1 \leftrightarrow O g_1) \]
  \[ (g_2 \leftrightarrow O r_2) \rightarrow (g_2 \leftrightarrow O g_2) \]

- Good things:
  \[ \Box \Diamond (g_1 = r_1) \land \Box \Diamond (g_2 = r_2) \]

If requesting, stay until granted
Don’t reuse grants

Mutual exclusion
Don’t grant w.o. request
Don’t take away used grants

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Separate to Assumptions and Guarantees

Environment:
• Initially:
  \( \neg r_1 \land \neg r_2 \)
• Transition:
  \[
  \left( (r_1 \land \neg g_1) \rightarrow \bigcirc r_1 \right) \land \\
  \left( (\neg r_1 \land g_1) \rightarrow \bigcirc \neg r_1 \right) \land \\
  \left( (r_2 \land \neg g_2) \rightarrow \bigcirc r_2 \right) \land \\
  \left( (\neg r_2 \land g_2) \rightarrow \bigcirc \neg r_2 \right)
  \]

System:
• Initially:
  \( \neg g_1 \land \neg g_2 \)
• Transition:
  \[
  \left( \neg g_1 \lor \neg g_2 \right) \land \\
  \left( (g_1 \leftrightarrow \bigcirc r_1) \rightarrow (g_1 \leftrightarrow \bigcirc g_1) \right) \land \\
  \left( (g_2 \leftrightarrow \bigcirc r_2) \rightarrow (g_2 \leftrightarrow \bigcirc g_2) \right)
  \]
• Good things:
  \( \Box \Diamond (g_1 = r_1) \land \Box \Diamond (g_2 = r_2) \)
The Goal for Synthesis

\[(\theta_e \land \Box \rho_e) \rightarrow (\theta_s \land \Box \rho_s \land (\land_i \Box \Diamond G_i))\]

- This still does not look very simple …

- Can we do anything with the bits \(\theta_e, \theta_s, \Box \rho_e,\) and \(\Box \rho_s\)?
  - \(\theta_s\) can be used to restrict the initial moves of \(P_0\):
    - For every initial input there is initial output satisfying \(\theta_s\) …
  - \(\Box \rho_s\) can be used to restrict the transitions of \(P_0\).
  - What if we use \(\theta_e\) and \(\Box \rho_e\) to restrict the moves of \(P_1\)?
Lecture 4: Bypassing Determinization

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Lecture 4: Bypassing Determinization

N. Piterman

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
What’s left?

\[(\theta_e \land \Box \rho_e) \rightarrow (\theta_s \land \Box \rho_s \land (\land_i \Box \Diamond G_i))\]

• This is slightly more complicated than response. We call it generalized Büchi.

Büchi:
1. fix (greatest := V)
2. fix (least := G \land \text{cpre}(\text{greatest})
3. least := least \lor \text{cpre}(\text{least});
4. end // fix least
5. greatest := least;
6. end // fix greatest

Generalized Büchi:
1. fix (greatest := V)
2. foreach (G_i)
3. fix (least := G_i \land \text{cpre}(\text{greatest})
4. least := least \lor \text{cpre}(\text{least});
5. end // fix least
6. greatest := least;
7. end // foreach
8. end // fix greatest
Proof (Generalized Büchi–Soundness)

• Suppose that \texttt{greatest} is not empty. For the fixpoint to terminate, for each \(G_i\) the inner fixpoint starting from this value recomputes it.

• Let \(\texttt{least}_0, \texttt{least}_1, \texttt{least}_2, \ldots\) be the sequence of values that \texttt{least} has through the computation of this last iteration for \(G_i\).

• Consider \(v \in \texttt{greatest}\). Let \(j_0\) be the index such that \(v \in \texttt{least}_{j_0}\).

By definition of \(\texttt{cpre}()\), \(P_0\) can force a successor \(w\) of \(v\). But then, \(w \in \texttt{least}_{j_1}\) for some \(j_1 < j_0\). This shows that \(P_0\) can ensure to reach \(\texttt{least}_{0} = G_0 \land \texttt{cpre}(\texttt{greatest})\). So it ensures a visit \(G_i\).

• But now \(\texttt{least}_{0} = G_i \land \texttt{cpre}(\texttt{greatest})\).

So next \(P_0\) forces \(\texttt{least}_{k+1}\), for some \(k\) and repeat this process.

• By induction, \(P_0\) can enforce \(\land_i \square \Diamond G_i\).

Generalized Büchi:
1. fix (\texttt{greatest} := \(V\))
2. foreach (\(G_i\))
3. fix (\texttt{least} := \(G_i \land \texttt{cpre}(\texttt{greatest})\))
4. \texttt{least} := \texttt{least} \lor \texttt{cpre}(\texttt{least});
5. end // fix least
6. greatest := \texttt{least};
7. end // foreach
8. end // fix greatest
Proof (Control of Büchi - completeness)

If there is a strategy $f$ s.t. every play compliant with it wins $\bigwedge_i \square \Diamond G_i$.

Every node $v$ from which $f$ is winning remains in every approximation of the fixpoint $\text{greatest}$:

Consider some $G_i$. From $v$ there is a maximum on the length of paths to reach $G_i \land \text{cpre}(\text{greatest})$ (König’s lemma).

Prove by induction on the number of iterations in the first fixpoint that $\text{win} \subseteq \text{greatest}$.

For $\text{greatest}_0 = V$ this is clear. Assume every node $v \in \text{win}$ it must be that $v$ reach $G_i \land \text{cpre}(\text{win})$.

Generalized Büchi:

1. fix ($\text{greatest} := V$)
2. foreach ($G_i$)
3. fix ($\text{least} := G_i \land \text{cpre}(\text{greatest})$)
4. $\text{least} := \text{least} \lor \text{cpre}(\text{least})$
5. end // fix least
6. $\text{greatest} := \text{least}$
7. end // foreach
8. end // fix greatest
Lecture 4: Bypassing Determinization

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Oops …

- The clients do not release the bus!
- It’s not only the system that has to do good things.
- The environment has to do good things as well!
- We need: \((\land_j \Box \Diamond A_j) \rightarrow (\land_i \Box \Diamond G_i)\)
- We call this Generalized Reactivity (1) or GR(1).
Solving GR(1) Games

Generalized Reactivity (1):
1. fix (greatestZ := \( V \))
2. foreach \( (G_i) \)
3. fix (leastY := \( G_i \land \text{cpre}(\text{greatestZ}) \))
4. leastY := leastY \lor \text{cpre}(\text{leastY});
5. foreach \( (A_j) \)
6. fix (greatestX := \( V \))
7. greatestX := least \lor (\lnot A_j \land \text{cpre}(\text{greatestX}))
8. end // fix greatestX
9. leastY := leastY \lor \text{greatestX};
10. end // foreach A
11. end // fix leastY
12. greatestZ := leastY;
13. end // foreach G
14. end // fix greatestZ
Proof (Control of GR(1) – Soundness)

Suppose that \( \text{greatestZ} \) is not empty. For each \( G_i \) the inner fixpoint starting from \( \text{greatestZ} \) recomputes \( \text{greatestZ} \).

Let \( \text{leastY}_0^i, \text{leastY}_1^i, \text{leastY}_2^i, \ldots \) be the sequence of values that \( \text{leastY} \) has during the last iteration. Each \( \text{leastY}_k^i \) is equal to the union of \( \text{greatestX}_{k_1}^{i,1}, \text{greatestX}_{k_1}^{i,2}, \ldots, \text{greatestX}_{k_1}^{i,m} \).

Consider \( v \in \text{greatestZ} \). Let \( k_0 \) be the minimal index such that \( v \in \text{leastY}_{k_0}^i \) and let \( j_0 \) be the minimal such that \( v \in \text{greatestX}_{k_0}^{i,j_0} \).

By definition of \( \text{cpre} \), \( P_0 \) can control to reach in one move \( \text{greatestX}_{k_1}^{i,j_1} \) such that either (A) \( k_1 < k_0 \) or (B) \( k_1 = k_0 \) and \( j_1 = j_0 \).

In case (B), we know that \( v \models \neg A_j \). So by playing this strategy, \( P_0 \) can ensure that either some \( A \) is visited finitely often, or reach \( \text{leastY}_0^i \land \text{cpre}(\text{greatestZ}) \).

By repeating the same for all \( G_i \), \( P_0 \) can enforce

\[
(\bigwedge_j \Box \Box A_j) \rightarrow (\bigwedge_i \Box \Box G_i)
\]
Proof (Control of GR(1) – completeness sketch)

If there is a strategy \( f \) s.t. every play compliant with it wins
\[
(\land_j \Box \Diamond A_j) \rightarrow (\land_i \Box \Diamond G_i)
\]
Every \( v \) from which \( f \) is winning remains in every approximation of the fixpoint \( \text{greatestZ} \):
As before, consider some \( G_i \). From \( v \) there is a maximum on the number of visits to \( A_j \) before arriving to \( G_i \land \text{cpre}(\text{win}) \) (König’s lemma).
Prove by induction on the number of iterations in the first fixpoint that \( \text{win} \subseteq \text{greatestZ} \).
For \( \text{greatestZ}_0 = V \) this is clear. Assume \( \text{win} \subseteq \text{greatestZ}_l \). Then for every \( v \in \text{win} \) it must be that \( v \in \text{leastY}_k \) for some \( k \).
Memorizing Intermediate Values

Generalized Reactivity (1):
1. fix (greatestZ := V)
2. foreach (G_i)
3. cY := 0;
4. fix (leastY := G_i ∧ cpre(greatestZ))
5. leastY := leastY ∨ cpre(leastY);
6. foreach (A_j)
7. fix (greatestX := V)
8. greatestX := least ∨ (¬A_j ∧ cpre(greatestX))
9. end // fix greatestX
10. x[G_i][cY][A_j] := greatestX;
11. leastY := leastY ∨ greatestX;
12. end // foreach A
13. y[G_i][cY] := leastY;
14. cY := cY + 1;
15. end // fix leastY
16. greatestZ := leastY;
17. end // foreach G
18. end // fix greatestZ
Construct the Realizing Machine

\[(\theta_e \land \square \rho_e \land (\land_j \square \Diamond A_j)) \rightarrow (\theta_s \land \square \rho_s \land (\land_i \square \Diamond G_i))\]

- Embed \(\theta_e, \rho_e, \theta_s,\) and \(\rho_s\) into \(G = \langle V, V_0, V_1, E, \varphi \rangle,\) where
  \[\varphi = (\land_j \square \Diamond A_j) \rightarrow (\land_i \square \Diamond G_i)\]

- Set let \(m = |\{G_i\}|\) and \(n = |\{A_i\}|.\)
- Construct a machine \(M\) realizing \(\varphi:\)
  \[M = \langle 2^J, 2^O, 2^{J\cup O} \times [1..m] \cup \{s_0\}, \rho, s_0, L\rangle:\]
  \[
  \rho(s_0, i) = \begin{cases} 
  \theta_s & i = \theta_e \\
  T & i = \neg \theta_e 
  \end{cases}
  \]

\[\rho((i, o, l), i') = \begin{cases} 
(i', o', l \oplus 1) & (i, o) \models G_l \land (i', o') \in \text{win} \\
(i', o', l) & (i, o) \models \neg A_j \land (i', o') \in x[G_l][cY][A_j] \land (i', o') \in y[G_l][\leq cY][\leq A_j] \\
(i', o', l) & (i, o) \models G_l \land (i', o') \in y[G_l][cY][< cY] \\
\end{cases}\]
Optimizing Symbolic Runtime

Generalized Reactivity (1):
1. fix (greatestZ := $V$)
2. foreach ($G_i$)
3. \( cY := 0; \)
4. fix (leastY := $G_i \land cpref(greatestZ))
5. leastY := leastY \lor cpref(leastY);
6. foreach ($A_j$)
7. fix (greatestX := $y[G_i][maxprev]$)
8. greatestX := least \lor (\neg A_j \land cpref(greatestX))
9. end // fix greatestX
10. $x[G_i][cY][A_j]$ := greatestX;
11. leastY := leastY \lor greatestX;
12. end // foreach $A$
13. $y[G_i][cY]$ := leastY;
14. $cY$ := $cY + 1$;
15. end // fix leastY
16. greatestZ := leastY;
17. end // foreach $G$
18. end // fix greatestZ
Back to the Arbiter

Environment:
• Initially:
  \( \neg r_1 \land \neg r_2 \)
• Transition:
  \((r_1 \land \neg g_1) \rightarrow O r_1\) \land
  \((\neg r_1 \land g_1) \rightarrow O \neg r_1\) \land
  \((r_2 \land \neg g_2) \rightarrow O r_2\) \land
  \((\neg r_2 \land g_2) \rightarrow O \neg r_2\)
• Good things:
  \(\Box \Diamond (\neg r_1 \lor \neg g_1) \land \Box \Diamond (\neg r_2 \lor \neg g_2)\)

System:
• Initially:
  \( \neg g_1 \land \neg g_2 \)
• Transition:
  \((\neg g_1 \lor \neg g_2) \land
  ((g_1 \leftrightarrow O r_1) \rightarrow (g_1 \leftrightarrow O g_1)) \land
  ((g_2 \leftrightarrow O r_2) \rightarrow (g_2 \leftrightarrow O g_2))\)
• Good things:
  \(\Box \Diamond (g_1 = r_1) \land \Box \Diamond (g_2 = r_2)\)
Result of Synthesis

\[ r_1:0, r_2:0, g_1:0, g_2:0 \] → \[ r_1:0, r_2:0, g_1:0, g_2:0 \] → \[ r_1:0, r_2:1, g_1:0, g_2:0 \] → \[ r_1:0, r_2:1, g_1:0, g_2:1 \] → \[ r_1:0, r_2:0, g_1:0, g_2:1 \] → \[ r_1:1, r_2:0, g_1:0, g_2:0 \] → \[ r_1:1, r_2:1, g_1:0, g_2:0 \] → \[ r_1:1, r_2:1, g_1:0, g_2:1 \] → \[ r_1:1, r_2:0, g_1:0, g_2:1 \] → \[ r_1:1, r_2:0, g_1:1, g_2:0 \] → \[ r_1:1, r_2:1, g_1:1, g_2:0 \] → \[ r_1:1, r_2:1, g_1:1, g_2:1 \] → \[ r_1:1, r_2:0, g_1:1, g_2:1 \] → \[ r_1:0, r_2:0, g_1:1, g_2:0 \] → \[ r_1:0, r_2:1, g_1:1, g_2:0 \] → \[ r_1:0, r_2:1, g_1:1, g_2:1 \] → \[ r_1:0, r_2:0, g_1:1, g_2:1 \]
But why do you embed safety?

• We started from:
  \[(\theta_e \land \Box \rho_e \land (\land_j \Box \Diamond A_j)) \rightarrow (\theta_s \land \Box \rho_s \land (\land_i \Box \Diamond G_i))\]

• And ended up with:
  \[(\land_j \Box \Diamond A_j) \rightarrow (\land_i \Box \Diamond G_i)\]

with some modifications to permitted moves in \(2^{\mathcal{I} \cup \mathcal{O}}\).

• Are the two the same?
  • No!

• What’s the difference?
  – Realizability in our game implies realizability of the general formula.
  – Other direction is not true.
Some applications

**AMBA Bus**
- Industrial standard
- ARM's AMBA AHB bus
  - High performance on-chip bus
  - Data, address, and control signals
  - Up to 16 masters and 16 clients
  - Arbiter part of bus (determines control signals)

**Generalized Buffer**
- Tutorial model checking design from IBM
  - Parameterized buffer
    - Transfer data from n senders to 2 receivers
    - Senders arbitrary order
    - Receivers round robin

---

**Valet Parking Without a Valet**

*Where's Waldo? Sensor-Based Temporal Logic Motion Planning*

M. Brandt, G. Cremers, T. Honari, and L. M. Rodrigues

---

**UBA**

Universidad de Buenos Aires

---

Lecture 4: Bypassing Determinization

N. Piterman

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Lecture 4: Bypassing Determinization

Valet Parking Without a Valet

David C. Conner, Hadas Kress-Gazit, Howie Choset, Alfred A. Rizzi, and George J. Pappas

Where’s Waldo?
Sensor-Based Temporal Logic Motion Planning

Hadas Kress-Gazit, Georgios E. Fainekos and George J. Pappas
Automatic Synthesis of Robust Embedded Control Software

Tichakorn Wongpiromsarn, Utuk Topcu and Richard M. Murray
California Institute of Technology
Pasadena, California 91125

vehicle & environment states
mission planner

vehicle & environment states
route response

vehicle & environment states
traffic planner

path planning problem
response

destination planner

vehicle state

obstacle follower

activation commands

emergency stop

activation commands

actuator state

vehicle

Fig. 1. Left: Alice, Team Cabeche’s entry in the 2007 DARPA Urban Challenge. Right: Alice’s planner-controller subsystem.
Robotics Approach Overview

- Specification
  - Synthesis
    - Realizable
    - Unrealizable
  - Specification Analysis

FSM

Hybrid Controller

Realizable

Unrealizable

Simulation

Physical Robot
\[ \varphi_{t,\text{pre}}^s = \bigwedge_{a \in A} \square \left[ \neg \left( \bigvee_{\sigma_p \in \sigma_{\text{pre}(a)}} \left( \bigwedge_{\sigma \in \sigma_p} \right) \right) \rightarrow \neg a \right] \]

\[ \varphi_{t,\text{eff}}^e = \bigwedge_{a \in A} \square \left[ a \rightarrow \bigvee_{j \in \{1, \ldots, k(a)\}} \left( \bigwedge_{\sigma \in \sigma_{\text{eff}^j(a)}} \bigcirc \sigma \right) \bigwedge \left( \bigwedge_{\sigma \in \sigma_{\text{eff}^j(a)}} \bigcirc \neg \sigma \right) \bigwedge \left( \bigwedge_{\sigma \in \sigma_{\text{stay}_{\text{eff}^j(a)}}} \sigma \leftrightarrow \bigcirc \sigma \right) \right] \]

\[ \varphi_{t,\text{no act}}^e = \square \left[ \left( \bigwedge_{a \in A} \neg a \right) \rightarrow \left( \bigwedge_{\sigma \in \Sigma} \sigma \leftrightarrow \bigcirc \sigma \right) \right] \]
Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks

\[ \Psi_{\text{host}} = G\left( \text{lead} \Rightarrow \left( \left( F_{[0,25]}(\|x_{1,t} - x_{\text{cstmr},1,t}\| < 1) \lor \left( \|x_{1,t} - x_{\text{cstmr},1,t}\| < 1 \right) \land U_{[25,60]}(\|x_{1,t} - [1.75, -1]\| < 1) \right)^\wedge \right) \right) \]

\[ \Psi_{\text{request}_k} = G\left( \text{request}_k \Rightarrow F_{[0,20]} \left( \left( \|x_{2,t} - x_{\text{cstmr},k,t}\| < 1 \right) \lor \left( \|x_{3,t} - x_{\text{cstmr},k,t}\| < 1 \right) \lor \left( \|x_{4,t} - x_{\text{cstmr},k,t}\| < 1 \right) \wedge \right) \right) \]

\[ \Psi_{\text{collision}} = G_{[0,\infty]}(\|x_{i,t} - x_{j,t}\| > 0.05), \forall i \neq j \]

\[ \Psi_{\text{wallAvoid}_i} = G_{[0,\infty]}(\min(\|x_{i,t} - M\|) > 0.1), i = (1, 2, \ldots, 5) \]
Iterator-Based Temporal Logic Task Planning
Sebastián A. Zudaire; Martin Garrett; Sebastián Uchitel

IEEE International Conference on Robotics and Automation (ICRA)
Lecture 4: Bypassing Determinization
N. Piterman
Reactive Synthesis, MOVEP Summer School, Aalborg, 2022
Lecture 4: Bypassing Determinization

MovementModel = (go[Rooms][Locations] → GoModel),
GoModel = (arrived[Rooms][Locations] → MovementModel).

Adjacency = InRoom1, //Start in room 0
InRoom1 = (go[1][Locations] → InRoom1 | go[2][Locations] → InRoom2), //From Room1 we can go to Room1 or Room2
InRoom2 = (go[2][Locations] → InRoom2 | go[3][Locations] → InRoom3), //From Room2 we can go to Room2 or Room3
InRoom3 = (go[3][Locations] → InRoom3 | go[1][Locations] → InRoom1). //From Room3 we can go to Room3 or Room1

Room(Id=1) = Elem[0],
Elem[i:Lociations] = (when (i<1) go[Id][i+1] → arrived[Id][i+1] → Elem[i+1] |
  when (i>0) go[Id][i-1] → arrived[Id][i-1] → Elem[i-1]).

PersonSensor = (sense → Sensing),
Sensing = ({yes.person,no.person} → PersonSensor).

\texttt{ltl_property SenseAtEachLoc} = [(\neg arrived[Rooms][Locations] W \{yes.person,no.person\})]

\texttt{fluent WentLocRoom[j:1..N][i:0..M] = \langle arrived[j][i],yes.person\rangle}
\texttt{fluent FoundPerson = \langle yes.person, Alphabet\{yes.person\}\rangle}

assert VisitedRoom1 = ((WentLocRoom[1][0] && WentLocRoom[1][1] && WentLocRoom[1][2]) \lor FoundPerson)
assert VisitedRoom2 = ((WentLocRoom[2][0] && WentLocRoom[2][1] && WentLocRoom[2][2]) \lor FoundPerson)
assert VisitedRoom3 = ((WentLocRoom[3][0] && WentLocRoom[3][1] && WentLocRoom[3][2]) \lor FoundPerson)

controllerSpec ControlSpec = {
  safety = {};
  assumption = {};
  liveness = {VisitedRoom1, VisitedRoom2} //, VisitedRoom3
  controllable = {Controlables};
}
Bibliography

Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Current Research Directions
Distributed Synthesis

- We want to co-synthesize controllers that will control different variables and collaborate.
- An architecture $A = (P, x, P, I, O)$, where:
  - $P$ is a set of processes.
  - $x \subseteq P$ the environment.
  - $I$ set of (Boolean) variables.
  - $O = P \rightarrow 2^I$ input connectivity function.
  - $O = P \rightarrow 2^O$ output connectivity function.
  - $\delta_p: I \rightarrow \{0, 1\}$ action of $p$.
  - $\exists p, p \in P^2$. $\forall p \in P^2$.

A reactive implementation for $p \circ p$ is $\langle 0, 0 \rangle$.

As before, we would like to replace $\langle 0, 0 \rangle$ by some (finite) domain $\delta_p$.

- Given implementations $\delta_p$ for all processes, their composition $\delta_p \delta_p$ includes all possible matching interactions.

Abstractive Real Time

- Discrete controllers are augmented with continuous controllers.
- The controller model does not capture time it takes to cross a transition.
- How to combine?

Safety of Learned Behaviour

- Use formal specifications at learning and at runtime:
  - Shared synthesis - create controllers that accompany a learner and restrict attention to safe actions.

Unrealisability

- What feedback do you give when the specification is unrealizable?
  - Environment counter strategy.
  - Build a strategy for the environment and let the user play against it.
  - Unrealisability core.
  - Compute a minimal set of guarantees that is still unrealizable.
  - Suggest additional assumptions that make guarantees possible.

Asynchronous (fully observable) Composition

- So far: system and environment were strictly synchronous.
- This caused some problems with hybrid control that we need to circumvent.
- How to allow both system and environment multiple (unbounded) number of actions without either freezing?
  - Add a "who is control" mechanism.
  - Add an additional clause to specification forcing the system to give back control.
  - Games become more complicated.
  - "Modelling benefit" justifies?

Strategic Reasoning

- Using games and reasoning about strategies for designing multi-agent systems.
- Connections to algorithmic game theory.
- Logics, games, equilibrium.
Distributed Synthesis

• We want to co-synthesize controllers that will control different variables and collaborate.
• An architecture $A = (P, e, \mathcal{V}, I, O)$, where:
  – $P$ is a set of processes.
  – $e \in P$ the environment.
  – $\mathcal{V}$ set of (Boolean) variables.
  – $I: P \to 2^\mathcal{V}$ input connectivity function.
  – $O: P \to 2^\mathcal{V}$ output connectivity function.
    $\forall p_1, p_2. O(p_1) \cap O(p_2) = \emptyset$
    $\mathcal{V} = \bigcup_{p \in P} O(p)$

• An implementation for $p \in P$ is $(2^{I(p)})^+ \to 2^{O(p)}$.
  As before, we would like to replace $(2^{I(p)})^+$ by some (finite) domain $D_p$.
• Given implementations $\{T_p\}_{p \in P}$ for all processes, their composition $\|_p T_p$ includes all possible matching interactions.
The Synthesis Problem

• Given an architecture $A = (P, e, \mathcal{V}, I, O)$ and a specification $\varphi$ over $\mathcal{V}$, do there exists implementations $\{T_p\}_{p \in P}$ such that $\parallel_p T_p$ satisfies $\varphi$?
• In general the problem is undecidable.
  – It is enough to have an architecture with two processes with separate inputs.
  – If the architecture contains an information fork, synthesis for it is undecidable.
• Some architectures are possible:

But complexity is non-elementary.
What are my options?

- **Bounded synthesis:**
  - Use the bounded synthesis for each process separately.
  - Synthesize all the processes together.

- **Construct dominant strategies inductively:**
  - For a process construct a dominant strategy for the full specification.
  - Extract from the dominant strategy the assumptions for other processes.
  - Synthesize a dominant strategy for specification and new assumptions for all others.

- Use Zielonka/Asynchronous Automata.
  - Communication by synchronous message passing (blocking multicast).
  - More architectures are decidable.
  - Sending of Full information leads to algorithmic distribution.
Safety of Learned Behaviour

- Use formal specifications at learning and at runtime:
  - Shield synthesis – create controllers that accompany a learner and restrict attention to safe actions.
Shield Synthesis for Reinforcement Learning

Bettina Könighofer, Florian Lorber, Nils Jansen & Roderick Bloem
Good-for-MDPs Automata for Probabilistic Analysis and Reinforcement Learning

Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, & Dominik Wojtczak
Strategic Reasoning

• Using *games* and *reasoning* about *strategies* for designing *multi-agent systems*.
• Connections to *algorithmic game theory*.
• Logics, games, equilibria, …
Concurrent Game Structures

- A concurrent game structure $G = \langle AP, Ag, Ac, St, \lambda, \tau, s_0 \rangle$:
  - $AP$ – atomic set of propositions.
  - $Ag$ – set of agents.
  - $Ac$ – set of actions.
  - $St$ – set of states.
  - $\lambda: St \rightarrow 2^{AP}$ - labeling function.
  - $\tau: St \times Ac^{Ag} \rightarrow St$ – transition function.
- History / track: $\rho \in St^*$.
- Strategy: $f: St^* \rightarrow Ac$.
- Strategy profile: $\{f_{ag}\}_{ag \in Ag}$.
- A strategy profile defines exactly one infinite run.
Logics and Equilibria

• **Alternating Temporal Logic** – quantify existentially and universally about abilities of coalitions.

\[
\langle \langle X \rangle \rangle \Diamond P
\]

• **Strategy logic** – quantify existentially and universally about individual strategies.

\[
\exists x_1, x_2 \forall x_3, x_4 \Diamond P(x_1, x_2, x_3, x_4)
\]

\[
\exists x_1, x_2 \forall x_3, x_4 \Diamond P(x_1, x_2) \land \Box Q_1(x_1, x_4) \land \Box Q_2(x_2, x_3)
\]

• **Nash equilibrium** – a strategy profile such that if a player deviates, other players can join forces to punish them.

• **Subgame perfect equilibrium** – a strategy profile that is optimal from every location in the game.
Rationality

- What does it mean for an agent to be rational?
- Nash equilibrium in Boolean context?
- Rational synthesis …
- Dominant strategies …
- Good-enough synthesis …
Related Work / Open Problems

- Other determinization [Křetínský, Esparza, …].
- History Determinization (GFG) [HP06, Boker, Lehtinen, …]
- Partial information [Chatterjee, Doyen, Raskin, …].
- Stochastic elements [Chatterjee, Kucera, …].
- Real time [Alur, Maler, Larsen, …].
- Quantitative Objectives [Henzinger, Kupferman, Raskin, …].
- Distributed Synthesis [Muschol, Finkbeiner, Raskin, Walukiewicz, …].
Summary

• Theoretical solution well known since 1969/1989.
• Still provides motivation for a lot of theoretical and practical work.
• In theory, theory and practice are the same.
• Thank you.