Synthesis from temporal logic

Guest Lecture
219C, Sanjit A. Seshia

26th November 2012
Verification and synthesis of reactive systems

Verification:

\[ G(u = 0 \rightarrow X(v = 1)) \]
Verification and synthesis of reactive systems

Synthesis:

\[ G(u = 0 \rightarrow X(v = 1)) \]

\[ \text{Input} = \{u, \ldots\} \]
\[ \text{Output} = \{v, \ldots\} \]

Realisable

Not realisable
Atomic propositions

- $\text{AP}_I = \{\text{button}\}$
- $\text{AP}_O = \{\text{grind}, \text{brew}\}$
Synthesis of reactive systems - example

Atomic propositions
- $AP_I = \{button\}$
- $AP_O = \{grind, brew\}$

A run of the system
$$\rho = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \ldots$$
Synthesis of reactive systems - example

Atomic propositions
- $\text{AP}_I = \{\text{button}\}$
- $\text{AP}_O = \{\text{grind}, \text{brew}\}$

A run of the system

\[
\rho = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{pmatrix} \ldots
\]

Specification
Whenever the user presses the button, the grinding unit should be activated for the next 2 steps. After that, the grinding module should be inactive while the brewing unit brews for the next 3 steps.
Informal specification

Whenever the user presses the button, the grinding unit should be activated for the next 2 steps. After that, the grinding module should be inactive while the brewing unit brews for the next 3 steps.
Informal specification

Whenever the user presses the button, the grinding unit should be activated for the next 2 steps. After that, the grinding module should be inactive while the brewing unit brews for the next 3 steps.

 Formal specification in linear-time temporal logic (LTL)

\[ G (button) \rightarrow (grind \land X grind \land XX (brew \land \neg grind) \land XXXX (brew \land \neg grind)) \]
Formal specification in linear-time temporal logic (LTL)

\[ G(button) \rightarrow (\text{grind} \land \mathbf{X} \text{grind} \land \mathbf{XX}(\text{brew} \land \neg \text{grind}) \land \mathbf{XXX}(\text{brew} \land \neg \text{grind}) \land \mathbf{XXXX}(\text{brew} \land \neg \text{grind})) \]

A surprise

The specification is *unrealisable*.

Example:

\[
\rho = \begin{pmatrix}
0 \\
0 \\
0 \\
0
\end{pmatrix}
\begin{pmatrix}
1 \\
1 \\
1 \\
1
\end{pmatrix}
\begin{pmatrix}
0 \\
0 \\
0 \\
1
\end{pmatrix}
\begin{pmatrix}
??? \\
??? \\
??? \\
1
\end{pmatrix}
\ldots
\]
What do we want as a result?

The computed implementation should be...

- ...finite-state,
- ...deterministic, and
- ...non-terminating and input-responsive
What do we want as a result?

The computed implementation should be...

- finite-state,
- deterministic, and
- non-terminating and input-responsive

Kripke structures
What do we want as a result?

The computed implementation should be...

- finite-state,
- deterministic, and
- non-terminating and input-responsive

Kripke structures
Adapted computation models

### Moore machines
\[ M = (S, \Sigma^I, \Sigma^O, s_0, \delta, L) \] with:
- Set of states \( S \)
- Input/output alphabets \( \Sigma^I/\Sigma^O \)
- Initial state \( s_0 \)
- Transition function \( \delta : S \times \Sigma^I \rightarrow S \)
- State labeling: \( L : S \rightarrow \Sigma^O \)

### Mealy machines
\[ M = (S, \Sigma^I, \Sigma^O, s_0, \delta) \] with:
- Set of states \( S \)
- Input/output alphabets \( \Sigma^I/\Sigma^O \)
- Initial state \( s_0 \)
- Transition function \( \delta : S \times \Sigma^I \rightarrow S \times \Sigma^O \)
General synthesis workflow

\[ G(r \rightarrow Xg) \]

Specification

\[ q_{0} \quad q_{1} \]

Mealy machine
General synthesis workflow

Word automaton

\[ G(r \rightarrow Xg) \]

Specification

Mealy machine

\[ G(r \rightarrow Xg) \]
General synthesis workflow

Word automaton

\[ G(r \rightarrow Xg) \]

Strategy / Mealy machine

Game
Games

Definition

Every player in a (two-player) game $G = (V_0, V_1, \Sigma_0, \Sigma_1, E_0, E_1, v_0, \mathcal{F})$ has:

- Positions
- Actions
- Transitions
- A goal

Additionally, there is some initial position.
One player is the **system player**, whereas the other player is the **environment player**. If player $p \in \{0, 1\}$ has a **strategy** to win, then she can enforce to win a play by playing the strategy. We say that player $p$ wins the game in such a case.
Strategies

One player is the **system player**, whereas the other player is the **environment player**. If player $p \in \{0, 1\}$ has a **strategy** to win, then she can enforce to win a play by playing the strategy. We say that player $p$ wins the game in such a case.
Games for synthesis

Strategies

One player is the **system player**, whereas the other player is the **environment player**. If player $p \in \{0, 1\}$ has a **strategy** to win, then she can enforce to win a play by playing the strategy. We say that player $p$ wins the game in such a case.

This is a Mealy Machine!
Games for synthesis

Strategies

One player is the **system player**, whereas the other player is the **environment player**. If player \( p \in \{0, 1\} \) has a **strategy** to win, then she can enforce to win a play by playing the strategy. We say that player \( p \) wins the game in such a case.

Strategies in Synthesis games

In games that correspond to a specification, winning strategies for the system player represent Mealy or Moore machines that satisfy the specification.
Main questions

1. How do we solve a game (determine the winner and a winning strategy for her)?
2. What winning condition do we need to use for general LTL?
3. How do we build games that correspond to specifications?
A more complicated (safety) game

00  01

10  11
A more complicated (safety) game

\[ \overline{g_1} \lor \overline{g_2} \]

\[ g_1 g_2 \]

\[ \overline{g_1} \overline{g_2} \]

\[ g_1 \overline{g_2} \]

\[ \overline{g_1} g_2 \]

\[ \overline{g_1} \overline{g_2} \]

\[ \overline{g_1} \overline{g_2} \]

\[ g_1 g_2 \]

\[ \overline{g_1} g_2 \]

\[ g_1 \overline{g_2} \]

\[ \overline{g_1} \overline{g_2} \]

\[ g_1 g_2 \]

\[ \overline{g_1} \overline{g_2} \]

\[ \overline{g_1} \overline{g_2} \]
A more complicated (safety) game

\[ (g_2 \rightarrow g_1) \]

\[ \overline{g_1} g_2 \]

\[ r \]

\[ \overline{r} \]

\[ g_1 \rightarrow g_2 \]

\[ \perp \]

\[ g_1 \rightarrow g_2 \]

\[ \perp \]
A more complicated (safety) game
A more complicated (safety) game
A more complicated (safety) game

\[ g_1 \lor \overline{g_2} \quad (g_2 \rightarrow g_1) \]

\[ g_1 \overline{g_2} \quad \overline{g_1} \overline{g_2} \]

\[ \overline{g_1} g_2 \quad \overline{g_1} g_2 \]

\[ g_1 \overline{g_2} \quad (g_2 \rightarrow g_1) \]

\[ (g_1 \rightarrow g_2) \]

\[ \overline{g_1} g_2 \]

\[ (g_1 \rightarrow g_2) \]

\[ \overline{g_1} g_2 \]

\[ \overline{g_1} g_2 \]

\[ (g_2 \rightarrow g_1) \]

\[ (g_2 \rightarrow g_1) \]

\[ \overline{r} \]

\[ r \]

\[ \overline{r} \]

\[ r \]

\[ \overline{r} \]

\[ * \]

\[ * \]
A more complicated (safety) game
A more complicated (safety) game
A more complicated (safety) game

\[
\begin{align*}
&\overline{g_1} \lor \overline{g_2} \\
&\overline{g_1g_2} \\
&\overline{r} \\
\end{align*}
\]
A more complicated (safety) game
A more complicated (safety) game
Building safety games from deterministic safety automata

\[\begin{align*}
q_0 &\rightarrow \neg r \\
q_0 &\rightarrow r \\
q_0 &\rightarrow r \land g \\
q_1 &\rightarrow \neg r \\
q_1 &\rightarrow r \\
q_1 &\rightarrow r \land g \\
q_0 &\rightarrow \neg r \\
q_1 &\rightarrow r
\end{align*}\]
Building deterministic safety automata

\[ \psi = (a \land Xb) \mathcal{R} c, \quad \text{AP} = \{a, b, c\} \]
To get from safety to the general LTL case, we need to ...

... scale up from the safety winning condition to something more complex

Upcoming:
- Büchi games
- Parity games
Example deterministic Büchi automaton and game

**LTL Specification**

\[ G(r \rightarrow Fg) \]

**Büchi automaton**

- **Start state:** `ok`
- **States:** `ok`, `wait`
- **Transitions:**
  - `r \rightarrow \neg r \lor g`
  - `\neg r \rightarrow g`
  - `g \rightarrow \neg g`
  - `\neg g \rightarrow g`

**Büchi game**

- **Start state:** `start`
- **States:** `ok`, `wait`
- **Transitions:**
  - `* \rightarrow \neg r`
  - `r \rightarrow g`
  - `g \rightarrow \neg g`
  - `\neg g \rightarrow g`

4

Ruediger Ehlers  Synthesis from temporal logic  26th November 2012  16 / 24
On using non-deterministic Büchi automata

LTL

\[(GF_r \land GF_g) \lor (FG\neg r \land FG\neg g)\]

Büchi automaton

Büchi game

Ruediger Ehlers

Synthesis from temporal logic

26\textsuperscript{th} November 2012
Deterministic vs. non-deterministic Büchi automata

Properties of Büchi automata

- For every LTL formula, there exists a non-deterministic Büchi automaton
- For some LTL formulas, there exist no deterministic Büchi automata

Problem

The automaton→game construction only works for deterministic automata

Solution

Use a richer automaton model/game winning condition: parity automata
Parity automata and games

\[ GF \rightarrow GFg \]
Building parity automata

Classical approach

- Construct a non-deterministic Büchi automaton from the LTL specification
- Translate the Büchi automaton to a deterministic parity automaton (Piterman, 2006)

→ doubly-exponential blow-up!

More practical approach

Many LTL subsets allow building the deterministic parity automaton directly (see, e.g., Bloem et al., 2012, where this is done implicitly)
How to solve games, the general case

Safety: \( W_{sys} = \nu X. (V_{env} \land \overline{\Delta} \land \Box X) \lor (V_{sys} \land \overline{\Delta} \land \Diamond X) \)

Büchi:

Parity:
How to solve games, the general case

**Safety:** \[ W_{sys} = \nu X.(V_{env} \cap \square \cap X) \cup (V_{sys} \cap \square \cap \diamond X) \]

**Büchi:** \[ W_{sys} = \nu X.\mu Y.(V_{env} \cap \overline{F} \cap \square Y) \cup (V_{sys} \cap \overline{F} \cap \diamond Y) \]
\[ \cup (V_{env} \cap F \cap \square X) \cup (V_{sys} \cap F \cap \diamond X) \]

**Parity:**
How to solve games, the general case

Safety: \( W_{sys} = \nu X. (V_{env} \cap \Box X) \cup (V_{sys} \cap \Box X) \)

Büchi: \( W_{sys} = \nu X. \mu Y. (V_{env} \cap \Box Y) \cup (V_{sys} \cap \Box Y) \)
\( \cup (V_{env} \cap \Box X) \cup (V_{sys} \cap \Box X) \)

Parity: \( W_{sys} = \gamma X_{c-1} \ldots \nu X_2. \mu X_1. \nu X_0. \cup i \in \{c-1, \ldots, 0\} (V_{env} \cap C_i \cap \Box X_i) \cup (V_{sys} \cap C_i \cap \Box X_i) \)
How to solve games, the general case

Safety: \[ W_{\text{sys}} = \nu X. (V_{\text{env}} \cap \neg\neg \neg \neg X) \cup (V_{\text{sys}} \cap \neg \neg \neg \neg \diamond X) \]

Büchi: \[ W_{\text{sys}} = \nu X. \mu Y. (V_{\text{env}} \cap \neg F \cap \neg X) \cup (V_{\text{sys}} \cap \neg F \cap \diamond Y) \]
\[ \cup (V_{\text{env}} \cap F \cap \neg X) \cup (V_{\text{sys}} \cap F \cap \diamond X) \]

Parity: \[ W_{\text{sys}} = \gamma X_{c-1} \ldots \nu X_2. \mu X_1. \nu X_0. \]
\[ \cup_{i \in \{c-1, \ldots, 0\}} (V_{\text{env}} \cap C_i \cap \neg X_i) \cup (V_{\text{sys}} \cap C_i \cap \diamond X_i) \]

All games have positional winning strategies.

All complexities are polynomial (for some constant \( c \))
Conceptual summary

Word automaton

\[ G(r \rightarrow Xg) \]

Game

Strategy / Mealy machine
Efficiently implementing reactive synthesis

Symbolic data structures
- Binary decision diagrams
- Safety games: list of worst-case positions → antichains

Targeting specific specification classes - Example 1/2

Specification form:

\[(a_1 \land a_2 \land \ldots \land a_n) \rightarrow (g_1 \land g_2 \land \ldots \land g_m)\]

for which every \(a_i\) and \(g_i\) is of one of the following forms:

1. \(\psi\)
2. \(G(\psi_1 \rightarrow X(\psi_2))\)
3. \(GF(\psi)\)
Symbolic data structures

- Binary decision diagrams
- Safety games: list of worst-case positions → antichains

Targeting specific specification classes - Example 2/2

Specification form:

\[(a_1 \land a_2 \land \ldots \land a_n) \rightarrow (g_1 \land g_2 \land \ldots \land g_m)\]

for which most of \(a_1, \ldots, g_m\) are safety properties.
Reactive Synthesis - summary

Main concepts
- Mealy/Moore machines
- Deterministic word automata
- Games with \( \omega \)-regular winning condition

Main difficulties
- Complexity! It is 2EXPTIME for LTL specifications
- Complicated constructions (e.g., Büchi \( \rightarrow \) parity)

Main applications
- Automatic system construction
- Fast prototyping
- Specification debugging