Turing Machine

• A Turing machine is: $M = \langle \Sigma, Q, \Delta, q_0, b \rangle$

• Where
  – $\Sigma$ – finite set of symbols (alphabet).
  – $Q$ – finite set of states. Includes special states $acc(empt)$ and $rej(ect)$.
  – $b$ – is a blank symbol (indicating an empty/blank cell).
  – $\Delta: Q \times (\Sigma \cup \{b\}) \rightarrow (\Sigma \times \{\leftarrow, \downarrow, \rightarrow\} \times Q)$
    with every state and letter (where $a$ could be $b$) associate new letter, direction, and new state.
    We assume that $acc$ and $rej$ have no successors!
  – $q_0$ – is an initial state.

• Interesting questions about Turing machines:
  – Does it accept input $x$?
  – Does it halt on input $x$?
A function defines a relation between inputs and outputs.
Lecture 1: Introduction and Background

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.

Specified in terms of Input/Output relations.

**Reactive Programs**

Programs whose role is to maintain an ongoing interaction with their environments. Can be viewed as a 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.
  – 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

- 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.

\[
\begin{align*}
\forall t. \neg o_1(t) \lor \neg o_2(t) \\
\forall t. i(t) \rightarrow (\exists t' > t. o_1(t) \lor o_2(t)) \\
\forall t. o_1(t) \rightarrow (\exists t' < t. (i(t') \land \forall t' < t''. t' < t. (\neg o_1(t') \land \neg o_2(t')))) \\
\forall t. o_2(t) \rightarrow (\exists t' < t. (i(t') \land \forall t' < t''. t' < t. (\neg o_1(t') \land \neg o_2(t')))) \\
\forall t. o_1(t) \rightarrow (\forall t' > t. (\neg (o_1(t) \lor \exists t < t''. t'.o_2(t)))) \\
\forall t. o_1(t) \rightarrow (\forall t' > t. (\neg (o_2(t) \lor \exists t < t''. t'.o_1(t))))
\end{align*}
\]
Causality

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

• The relation \( R = \{(i, o) \mid i: \mathbb{N} \to \{0,1\}, o: \mathbb{N} \to \{0,1\}, o(0) \leftrightarrow (\exists t. i(t))\} \) is not empty.
• The function needs to be causal!
• It cannot be clairvoyant.
Adversarial

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

• There are some input sequences for which this is possible.
• But not all!
• We want a function that can answer all input sequences.
  \[ f: \{ i: \{0, ..., n\} \rightarrow \{0,1\} \mid n \in \mathbb{N} \} \rightarrow \{0,1\} \]
• Furthermore, for every \( i: \mathbb{N} \rightarrow \{0,1\} \) the unique \( o: \mathbb{N} \rightarrow \{0,1\} \) such that \( o(n) = f(i \mid \{0,...,n\}) \) 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.
- Practical issues with **synthesis**:
  - Implication problems.
  - Unrealizability.
  - Building hybrid controllers.
  - Distributed synthesis.
Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Practical Issues with Synthesis
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 sequence of assignments to propositions.
• All possible infinite paths over the automaton should satisfy the specification.
Linear Temporal Logic

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

• Linear Temporal Logic formulae are constructed as follows:
  \[ \varphi ::= p || \varphi \land \varphi || \neg \varphi || O \varphi || E \varphi || \varphi U \varphi || \varphi S \varphi \]

• Other temporal formulae are derived:
  - \( \Diamond \varphi \equiv T U \varphi \) — Eventually.
  - \( \Box \varphi \equiv \neg \Diamond \neg \varphi \) — Always.
  - \( \varphi W \psi \equiv \varphi U \psi \lor \Box \varphi \) — Weak Until.
  - \( \Diamond \varphi \equiv T S \varphi \) — Previously.
  - \( \Box \varphi \equiv \neg \Diamond \neg \varphi \) — Historically.
  - \( \varphi B \psi \equiv \varphi S \psi \lor \Box \varphi \) — BackTo.
LTL Semantics

• A model for an LTL formula $\varphi$ is an infinite sequence $\sigma = \sigma_0, \sigma_1, \ldots$ 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 $j \geq 0$, denoted $(\sigma, j) \models \varphi$, if:
  - If $\varphi$ is a proposition $(\sigma, j) \models \varphi \iff \varphi \in \sigma_j$
  - $(\sigma, j) \models \neg \varphi \iff (\sigma, j) \not\models \varphi$
  - $(\sigma, j) \models \varphi_1 \lor \varphi_2 \iff (\sigma, j) \models \varphi_1 \text{ or } (\sigma, j) \models \varphi_2$
  - $(\sigma, j) \models \Box \varphi \iff (\sigma, j + 1) \models \varphi$
  - $(\sigma, j) \models \Diamond \varphi \iff j > 0 \text{ and } (\sigma, j - 1) \models \varphi$
  - $(\sigma, j) \models \varphi_1 \mathcal{U} \varphi_2 \iff \exists k \geq j . (\sigma, k) \models \varphi_2 \text{ and } \forall j \leq l < k . (\sigma, l) \models \varphi_1$
  - $(\sigma, j) \models \varphi_1 \mathcal{S} \varphi_2 \iff \exists k \leq j . (\sigma, k) \models \varphi_2 \text{ and } \forall j \geq l > k . (\sigma, l) \models \varphi_1$

• Derived:
  - $(\sigma, j) \models \lozenge \varphi \iff \exists k \geq j . (\sigma, k) \models \varphi$
  - $(\sigma, j) \models \square \varphi \iff \forall k \geq j . (\sigma, k) \models \varphi$
LTL Exercises

\[ \square p \]

\[ \square \Diamond p \]

\[ \square (p \rightarrow \bigcirc (q \cup r)) \]

\[ \square (p \rightarrow p \mathcal{W} q) \equiv \square (p \rightarrow (\bigcirc p \lor \bigcirc q)) \]

\[ p \equiv \square (\Theta T \lor p) \]

\[ \square (p \rightarrow \Diamond q) \]

\[ \square (p \rightarrow \Theta (\neg p \mathcal{S} q)) \]

\[ \Diamond (\neg \Theta T \land p) \]

\[ \square (p \rightarrow \Diamond q) \equiv \square \Diamond \neg (\neg q \mathcal{S} p) \]

\[ (p \mathcal{U} (q \mathcal{U} r)) \neq ((p \mathcal{U} q) \mathcal{U} r) \]
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.
  – $\delta: Q \times \Sigma \to 2^Q$ – 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.
  – $\delta: Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with every state and an input letter a set of successor states.
  – $q_0$ – an initial state.
  – $L: Q \times \Sigma \rightarrow \Delta$ – 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 $\Sigma = 2^J$ 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 $Prop = I \cup O$ 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
• Practical Issues with Synthesis
Realizability

• So, given a property $\varphi$ and a partition $Prop = I \cup O$ 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_2

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

• Requirements:
  – $A_1$: leave requests: $\Box(r_1 \land \neg g_1 \rightarrow \diamond r_1) \land \Box(r_2 \land \neg g_2 \rightarrow \diamond r_2)$
  – $G_1$: 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: $\Box(\neg g_1 \lor \neg g_2)$
  – $G_3$: 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 still use graphs with edges for transitions.
• Ownership is by using two types of nodes.
A Play
Arbiter

Arbiter

Client

Arbiter

Client
Lecture 2: Games and Synthesis

N. Piterman
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 \( \pi = v_0, v_1, \ldots \)
  – \( \alpha \) 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 \( v \).
Control Predecessor

- In control it is easier to walk backwards.
Game Analysis
Control Predecessor (for P0)

- Start from an set of nodes $W \subseteq V$.
- We want to say:
  - The system can \textbf{force} the environment to $W$ in \textbf{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:
  \[
  cpre_1(W) = \{ v \in V_1 \mid \exists v' \in W. (v, v') \in E \} \cup \{ v \in V_0 \mid \forall v'. (v, v') \in E \rightarrow v' \in W \} 
  \]
Let’s solve some games!
Reachability Games

- Check that $P_1$ can enforce $\lozenge \neg p$.

1. fix (new := $\neg p$)
2. new := new $\lor cpre_1(new)$
3. end // fix

**Lemma.** The algorithm computes the set of states winning for $P_1$ with objective $\lozenge p$.

**Proof.** Later.

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

• Check that $P_0$ can enforce $\boxdot p$.
  1. $\text{fix } (\text{new} := p)$
  2. $\text{new} := \text{new} \land \text{cpre(new)}$
  3. $\text{end } // \text{fix}$

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

Proof. Later.
Safety vs Reachability Games

• Goals $\square p$ for $P0$ and $\Diamond \neg p$ for $P1$ are complementary.

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

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

  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

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

Games and Synthesis, EATCS Young Researchers School, Telč, Summer 2014
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 $f_i: V^* \cdot V_0 \rightarrow V$ 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: $f^m_i: D \times V_i \rightarrow V$ s.t. $(v, f(d, v)) \in E$.
  – Update function: $f^u_i: D \times V \rightarrow D$. 
Safety Games

- Check that $P_0$ can enforce $\square p$.
  1. $\text{fix} (\text{new} := p)$
  2. $\text{new} := \text{new} \land \text{cpre}(\text{new})$
  3. $\text{end} \ // \text{fix}$
Proof

• Suppose that \( \text{new} \) is not empty.
  Consider \( v \in \text{new} \). Clearly, \( v \in p \). But also \( v \in cpre(\text{new}) \).
  If \( v \in V_0 \), then \( v \) has a successor \( w \) such that \( w \in \text{new} \).
  If \( v \in V_1 \), then for every successor \( w \) of \( v \) we know \( w \in \text{new} \).
• If there is a strategy s.t. every play compliant with it wins \( \square p \).
  Let \( \text{new}_0, \text{new}_1, \text{new}_2, \ldots \) be the series of approximations of \( \text{new} \).
  We prove by induction that for every \( v \) winning for \( P_0 \), \( v \in \text{new}_i \) for every \( i \).
  Clearly, \( v \in p \) implies \( v \in \text{new}_0 \).
  Assume every \( v \) winning for \( P_0 \) is in \( \text{new}_i \) for some \( i \).
  Consider \( v \in V_0 \) winning for \( P_0 \). Then, there is \( w \) such that \((v, w) \in E\) and \( w \) winning for \( P_0 \). Then, \( w \) in \( \text{new}_i \) and \( v \) in \( \text{new}_{i+1} \).
  Consider \( v \in V_1 \) 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 \( \text{new}_i \).
  So \( v \) in \( \text{new}_{i+1} \).

1. fix (\( \text{new} := p \))
2. new := new \and cpre(new)
3. end // fix
Büchi Games

• Check that $P_0$ 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.  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.
By induction, P0 can enforce □□p.

1. fix (greatest := V)
2. fix (least := p ∧ cpre(greatest))
3. least := least ∨ cpre(least);
4. end // fix least
5. greatest := least;
6. end // fix greatest
Proof (Control of Büchi - completeness)

If there is a strategy \( f \) s.t. every play compliant with it wins \( \square \diamond p \).
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. \( \text{fix} (\text{greatest} := V) \)
2. \( \text{fix} (\text{least} := p \land \text{cpre}(\text{greatest}) \) \)
3. \( \text{least} := \text{least} \lor \text{cpre}(\text{least}); \)
4. \( \text{end} // \text{fix least} \)
5. \( \text{greatest} := \text{least}; \)
6. \( \text{end} // \text{fix greatest} \)
Symbolic vs Enumerative

• Algorithms so far have treated sets of states.
• But the proof established a ranking for the winning states – the number of steps until reaching the goal.
  \[ r: V \rightarrow \mathbb{N} \cup \{\infty\} \]

• Can we compute the rank iteratively?
  – A path that visits more than \(|\neg p|\) nodes must be a losing loop.
  – Restrict rank to \( r: V \rightarrow \{0, \ldots, |\neg p|\} \cup \{\infty\} \).
  – \( \text{best}(v) = \begin{cases} \min_{(v,w) \in E} \text{rank}(w) & v \in V_0 \\ \max_{(v,w) \in E} \text{rank}(w) & v \in V_1 \end{cases} \)

– Rank is stable if:
  • \( v \in p \) and \( r(\text{best}(v)) < \infty \).
  • \( v \notin p \) and \( r(\text{best}(v)) < r(v) \).
Compute Rank Directly

1. \( r := \lambda v. 0 \)
2. while (\( \exists v. v \) not stable)
3. \( \quad \text{if} (v \in p) \)
4. \( \quad r(v) := \infty; \)
5. \( \quad \text{else} \)
6. \( \quad r(v) := best(v)+1; \)
7. end // while

• Each \( v \) can be increased at most \( |\neg p| \) times.
• By evaluating loop condition (and \( best \)) efficiently, all work can be restricted to \( O(|V| \cdot |E|) \).
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 $2^{I \cup O}$.
• We can define the winning condition with an LTL formula over $I \cup O$. A play naturally corresponds to a possible model.
• For a set of nodes $W$, define
  $$cpre(W) = \{v \mid \forall x \in 2^I. \exists y \in 2^O. (x \cup y) \in W\}$$
• When computing the set of winning states, check that for every $x \in 2^I$ there is $y \in 2^O$ 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 $f_i : (2^{\mathcal{J} \cup \mathcal{O}})^* \cdot 2^\mathcal{J} \to 2^\mathcal{O}$.

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

• The strategy becomes $f_i : D \times 2^\mathcal{J} \to D \times 2^\mathcal{O}$
Consider a strategy $f_0 : D \times 2^j \to D \times 2^\omega$ 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:

$\Sigma = 2^j$

$\Delta = 2^\omega$

$\delta(d, i) = f_0(d, i) \downarrow_1$

$L(d, i) = f_0(d, i) \downarrow_2$

What’s the memory domain in the cases we’ve seen?
Winning $\rightarrow$ 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 $\rightarrow$ 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 := $p \land cpre(greatest)$)
3. least := least $\lor$ cpre(least)
4. end // fix least
5. end // fix greatest

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

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

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

• Starting from an LTL formula $\varphi$, construct the game $G = \langle 2^I \cup (2^I \times 2^J), 2^I \times 2^J, 2^I, E, \varphi \rangle$.

• Compute the set $\text{win}$. 

• If for every $i \in 2^J$ there is $o \in 2^O$ 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 and Büchi games.

• What about general LTL?
Bibliography

Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Practical Issues with Synthesis
From Logic to Graphs?

How to embed the logical winning condition into the graph notation?
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.
  – \( \delta : Q \times \Sigma \to 2^Q \) – 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 \( w = \sigma_0, \sigma_1, ... \) is a sequence of letters from \( \Sigma \).
• A run \( r = q_0, q_1, ... \) 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 $\mathcal{L}(M_\varphi) = \mathcal{L}(\varphi)$. 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 $\Diamond \square p$

Environment

System
NBW for $\Diamond \square p$

- NBW for $\varphi = \Diamond \square p$: 

\[ \begin{array}{c}
\text{p, } \neg \text{p} \\
\text{p, p}
\end{array} \]
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.
Nondeterministic parity 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.
  – $\delta: Q \times \Sigma \to 2^Q$ – a transition function. Associates with state and an input letter a set of successor states.
  – $q_0$ – an initial state.
  – $\alpha: Q \to \mathbb{N}$ – 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 $A$. 
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 determine it!
- Starting from an automaton with $n$ states:
  - Create an automaton with $O((n!)^2)$ states and $2n$ rank.
- Wolfgang showed one such construction.
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(greatest) )
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 \textit{win} is not empty. Have the intermediate least fixpoint approximations: \(\text{least}^p_0, \text{least}^p_1, \text{least}^p_2, \ldots\) for an odd parity \(p\).

Consider \(v \in \text{win}\). Let \(i_1, i_3, \ldots, i_m\) be the indices such that \(v \in \text{least}^j_{i_j}\). By definition of \(\text{cpre}(\cdot)\), \(P_0\) 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.
Enumerative Again

- The proof established a ranking for the winning states – the odd nodes until an even node.
  \[ r: V \rightarrow \mathbb{N}^{\text{odd}} \cup \{\infty\} \]

- Can we compute the rank iteratively?
  - For odd \( j \) a path that visits more than \( |\alpha^{-1}(j)| \) nodes must be a losing loop.
  - Restrict rank to \( r: V \rightarrow (|\alpha^{-1}(1) \times \cdots \times |\alpha^{-1}(m)|) \cup \{\infty\} \).
  - \( \text{best}(v) = \min_{(v,w) \in E} \text{rank}(w) \quad v \in V_0 \)
  - \( \text{best}(v) = \max_{(v,w) \in E} \text{rank}(w) \quad v \in V_1 \)

- Rank is stable if:
  - \( \alpha(v) \) is even and \( r(\text{best}(v)) \leq_{\alpha(v)} r(v) \).
  - \( \alpha(v) \) is odd and \( r(\text{best}(v)) <_{\alpha(v)} r(v) \).
Compute Rank Directly

1. $r := \lambda v. (0,0,\ldots,0)$
2. while ($\exists v. v$ not stable)
3. \hspace{1em} $r(v) := \text{inc}_{\alpha(v)}(\text{best}(v))$;
4. end \ // \ while

- Each $v$ can be increased at most $|V|^{|\text{odd}|}$ times.
- By evaluating loop condition (and $\text{best}$) efficiently, all work can be restricted to $|V|^{|\text{odd}|} \cdot |E|$.
- By counting better the exponent can be halved.
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 does not scale.
- More practice required for solution of parity games (Martin?).

$|\varphi| = n$

$|A_\varphi| = 2^{O(n \log n)}$

$|\alpha| = 2^n$

$2^{O(n^2 \log n)}$
Bibliography


Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Practical Issues with Synthesis
Two Ways to Avoid Determinization

• Replace by counting:
  – Search for bounded strategy.
  – Express winning through universal co-Büchi automata.
  – Limited determinization through counting.

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

- 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 \( \pi = v_0, v_1, ... \)
  - \( \alpha \) is a set of winning plays:
    - Safety: \( \alpha = W^\omega \) for some \( W \subseteq V \).
    - Büchi: \( \alpha = (V^*W)^\omega \) for some \( W \subseteq V \).
    - LTL: \( \alpha = \{ \pi \in V^\omega \mid \pi \models \varphi \} \).
    - NBW: \( \alpha = \langle V, Q, \delta, q_0, \beta \rangle \), where \( \delta: Q \times V \to 2^V \) and \( \beta \subseteq Q \).
    - DPW: \( \alpha = \langle V, Q, \delta, q_0, c \rangle \), where \( \delta: Q \times V \to V \) and \( c: Q \to \mathbb{N} \).
    - UCW: \( \alpha = \langle V, Q, \delta, q_0, \beta \rangle \), where \( \delta: Q \times V \to V \) and \( \beta \subseteq Q \).
Universal co-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 rejecting 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 finitely many $i \in \mathbb{N}$ we have $q_0 \in \alpha$.
• A word is accepted if all runs over it is accepting.
• The language of $A$, denoted $\mathcal{L}(A)$, is the set of words accepted by $A$. 
UCW for $\Box \Diamond \neg p$

- UCW for $\varphi = \Box \Diamond \neg p$:
UCW for $\Diamond \Box p$

- UCW for $\varphi = \Diamond \Box p$:
Runs of UCW on (W.P.B.) Mealy Machines

• Consider a UCW \( U_\varphi = \langle 2^\mathbb{J} \cup \mathbb{O}, Q, \delta, q_0, \alpha \rangle \), where \(|Q| = n\).

• Suppose we get a machine \( M = \langle 2^\mathbb{J}, 2^\mathbb{O}, S, \rho, s_0, L \rangle \) such that \( \mathcal{L}(M) \subseteq \mathcal{L}(\varphi) \), where \(|S| = m\).

• We run \( U \) on words “produced” by \( M \). What is the maximum number of visits to \( \alpha \)?
  
  – A word produced by \( M \) corresponds to a run \( r = s_0, s_1, \ldots \) on \( w = \sigma_0, \sigma_1, \ldots \) by taking \( c = (\sigma_0, L(s_0, \sigma_0)), (\sigma_1, L(s_1, \sigma_1)), \ldots \).
  
  – Let the run of \( U_\varphi \) on \( c \) be \( \pi = q_0, q_1, \ldots \). Here, \( q_{i+1} \in \delta(q_i, (\sigma_i, L(s_i, \sigma_i))) \).
  
  – If the number of visits to \( \alpha \) is more than \( n \cdot m \), there must exist \( i \) and \( j \) such that \( q_i = q_j, s_i = s_j \), and \( q_i \in \alpha \).
  
  – But then \( q_0, \ldots, (q_i, \ldots, q_{j-1})^\omega \) is a rejecting run of \( U_\varphi \) on \( (\sigma_0, L(s_0, \sigma_0)), \ldots, ((\sigma_i, L(s_i, \sigma_i)), \ldots, (\sigma_{j-1}, L(s_{j-1}, \sigma_{j-1})))^\omega \).
Bounded Runs of UCW

A UCW $U$ accepts all computations of $M$ iff for every computation of $M$ every run of $U$ visits at most $|U| \cdot |M|$ rejecting states.

⇒ Can’t be any other way (as we’ve seen).

⇐ Well …
   The number of visits to rejecting states is finite …
Specialized (Bounded) Determinization

• Follow all runs simultaneously.
• For each state count the number of visits to rejecting states it has seen so far.
• If number of visits exceeds maximum – abort.
Consider a UCW $U = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$. Determinize $U$ for bound $m$ by taking $D = \langle \Sigma, F, \rho, f_0, \beta \rangle$, where:

- $F = \{ f: Q \to (\{0, \ldots, m\} \cup \{\bot, \infty\}) \}$

- $f_0(q) = \begin{cases} 0 & q = q_0 \\ \bot & \text{Otherwise} \end{cases}$

- $\rho(f, \sigma)(q) = \begin{cases} \max\{q'|q \in \delta(q', \sigma)\} f(q') & q \notin \alpha \\ \max\{q'|q \in \delta(q', \sigma)\} 1 + f(q') & q \in \alpha \end{cases}$

Here $\forall v. \bot \leq v$, $m + 1 = \infty$ and $\infty + 1 = \infty$

- $\beta$ is a safety condition: $\{ f \in F | \forall q \in Q. f(q) < \infty \}$
Synchronous Composition

• Consider a game $G = \langle V, V_0, V_1, E, \varphi \rangle$. Let $D_\varphi = \langle \Sigma, D, \delta, d_0, \beta \rangle$ be the safety automaton constructed from $U_\varphi$ for size $m$.
• Their synchronous parallel composition $(G \parallel D_\varphi)$ is the safety 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.
Deterministic Automata Work (again)!

**Theorem.** If there is a machine of size $m$ realizing $\varphi$ then P0 wins $G \parallel D_\varphi$, where $D_\varphi$ is the deterministic automaton obtained by bounded determinization for size $m$.

⇒ If there is a machine of size $m$ realizing $\varphi$ then this machine can be used as memory for winning $G$ (with aim $\varphi$). All plays in the resulting game satisfy $\varphi$. The runs of $U_\varphi$ on them cannot visit more than $m$ rejecting states. Adding the states of $D_\varphi$ alongside does not change their behavior. This is a winning strategy in $G \parallel D_\varphi$. 

Games and Synthesis, EATCS Young Researchers School, Telč, Summer 2014
Deterministic Automata Work (again)!

Theorem. If $P_0$ wins $G \parallel D_\varphi$, where $D_\varphi$ is the deterministic automaton obtained by bounded determinization for size $m$ then

⇒ If there is a machine of size $m$ realizing $\varphi$ then this machine can be used as memory for winning $G$ (with aim $\varphi$). All plays in the resulting game satisfy $\varphi$. The runs of $U_\varphi$ on them cannot visit more than $m$ rejecting states. Adding the states of $D_\varphi$ alongside does not change their behavior. This is a winning strategy in $G \parallel D_\varphi$.
Theorem. If P0 wins $G \parallel D_\varphi$, where $D_\varphi$ is the deterministic automaton obtained by bounded determinization for size $m$, then $\varphi$ is realizable.

⇒ Use the states of $D_\varphi$ as memory for $G$ with winning condition $\varphi$. Every path is accepted by $D_\varphi$, which implies that it satisfies $\varphi$. 
Summarize

• Start from an LTL formula $\varphi$.
• Obtain a UCW $U_\varphi$ for $\varphi$.
• Construct $D_\varphi$ for increasing sizes.
• If one of the games $G \parallel D_\varphi$ is winning for P0 the formula is realizable.
  – Where do I get a UCW $U_\varphi$ for $\varphi$?
  – When do I stop?
From LTL to co-Büchi Automata

Theorem. Given an LTL formula $\varphi$ we can construct a universal co-Büchi automaton $U_\varphi$ such that $\mathcal{L}(U_\varphi) = \mathcal{L}(\varphi)$. The size of $U_\varphi$ is exponential in the length of $\varphi$.

Construct the nondeterministic Büchi automaton $N_{\neg \varphi}$. Think about $N_{\neg \varphi}$ as a universal co-Büchi automaton $U_\varphi$. 

Games and Synthesis, EATCS Young Researchers School, Telč, Summer 2014
How far should I go?

• We had a different solution with deterministic automaton:

Theorem. P0 wins $G$ with winning condition $\varphi$ iff P0 wins $G \parallel A_{\varphi}$, where $A_{\varphi}$ is a deterministic automaton for $\varphi$.

• In case that $\varphi$ is realizable there is a machine realizing it with at most $|G| \times |A_{\varphi}|$ states.
• We can use $|G| \times |A_{\varphi}|$ as the bound (but start searching with smaller bounds ...).
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.
Two Ways to Avoid Determinization

• Replace by counting:
  – Search for bounded strategy.
  – Express winning through universal co-Büchi automata.
  – Limited determinization through counting.

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

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

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

Client

\[ r_i, g_i, \overline{r_i}, \overline{g_i} \]
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:
  \[
  \begin{align*}
  & (-r_1 \land \neg g_1 \land \Box r_1 \rightarrow (r_2 \leftrightarrow \Box r_2)) \\
  & (r_1 \land g_1 \land \Box \neg r_1 \rightarrow (r_2 \leftrightarrow \Box r_2)) \\
  & (-r_2 \land \neg g_2 \land \Box r_2 \rightarrow (r_1 \leftrightarrow \Box r_1)) \\
  & (r_2 \land g_2 \land \Box \neg r_2 \rightarrow (r_1 \leftrightarrow \Box r_1)) \\
  & (-g_1 \lor \neg g_2) \\
  & (g_1 \neq \Box r_1 \land (g_2 \leftrightarrow \Box g_2)) \\
  & (g_2 \neq \Box r_2 \land (g_1 \leftrightarrow \Box g_1))
  \end{align*}
  \]

• Good things:
  \( \Box \Diamond (g_1 = r_1) \land \Box \Diamond (g_2 = r_2) \)
Separate to Assumptions and Guarantees

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

System:
• Initially:
  \( \neg g_1 \land \neg g_2 \)
• Transition:
  \[
  (\neg g_1 \lor \neg g_2) \land \\
  \left( (g_1 \neq \bigcirc r_1 \land (g_2 \leftrightarrow \bigcirc g_2)) \right) \lor \\
  \left( g_2 \neq \bigcirc r_2 \land (g_1 \leftrightarrow \bigcirc g_1) \right) \lor \\
  \left( g_1 \leftrightarrow \bigcirc g_1 \right) \land \left( g_2 \leftrightarrow \bigcirc g_2 \right)
  \]
• 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 \(P0:\)
    For every initial input there is initial output satisfying \(\theta_s\) …
  – \(\Box \rho_s\) can be used to restrict the transitions of \(P0.\)
  – What if we use \(\theta_e\) and \(\Box \rho_e\) to restrict the moves of \(P1?\)
Lecture 4: Bypassing Determinization

N. Piterman

Games and Synthesis, EATCS Young Researchers Sch
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 := $G \land cpre$ (greatest))
3. least := least $\lor$ cpre (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 := $G_i \land cpre$ (greatest))
4. least := least $\lor$ cpre (least);
5. end // fix least
6. greatest := least;
7. end // foreach
8. end // fix greatest
Proof (Generalized Büchi–Soundness)

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

Let $\text{least}_0^i, \text{least}_1^i, \text{least}_2^i, \ldots$ be the sequence of values that least has through the computation of this last iteration for $G_i$.

Consider $v \in \text{greatest}$. Let $j_0$ be the index such that $v \in \text{least}_{j_0}^i$. By definition of $\text{cpre}(\cdot)$, $P_0$ can force a successor $w$ of $v$. But then, $w \in \text{least}_{j_1}^i$ for some $j_1 < j_0$. This shows that $P_0$ can ensure to reach $\text{least}_0^i = G_0 \land \text{cpre}(\text{greatest})$. So it ensures a visit $G_i$.

But now $\text{least}_0^i = G_i \land \text{cpre}(\text{greatest})$. So next $P_0$ forces $\text{least}_{k+1}^i$, for some $k$ and repeat this process.

By induction, $P_0$ can enforce $\land_i \Box \Diamond G_i$.

---

Generalized Büchi:
1. fix (greatest := $V$)
2. foreach ($G_i$)
3. fix (least := $G_i \land \text{cpre}(\text{greatest})$
4. least := least $\lor$ cpre(least)
5. end // fix least
6. greatest := 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 $\land_i \square \Diamond G_i$.

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 $\text{win} \subseteq \text{greatest}_i$. Then for every node $v \in \text{win}$ it must be that $v \in \text{least}_j$ for the distance to reach $G_i \land \text{cpre}(\text{win})$.

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
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: $\bigwedge_j \Box \Diamond A_j \rightarrow \bigwedge_i \Box \Diamond G_i$
• We call this Generalized Reactivity (1) or GR(1).
Solving GR(1) Games

Generalized Reactivity (1):
1. fix (greatestZ := V)
2. foreach (Gi)
3. fix (leastY := Gi ∧ cpred(greatestZ))
4. leastY := leastY ∨ cpred(leastY);
5. foreach (Ai)
6. fix (greatestX := V)
7. greatestX := least X (¬Aj ∧ cpred(greatestX))
8. end // fix greatestX
9. leastY := leastY ∨ 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 greatest\(Z\) is not empty. For each \(G_i\) the inner fixpoint starting from greatest\(Z\) recomputes greatest\(Z\).

Let least\(Y^i_0\), least\(Y^i_1\), least\(Y^i_2\), ... be the sequence of values that least\(Y\) has during the last iteration. Each least\(Y^i_k\) is equal to the union of greatest\(X^i_{k,1}\), greatest\(X^i_{k,2}\), ..., greatest\(X^i_{k,m}\).

Consider \(v \in\) greatest\(Z\). Let \(k_0\) be the minimal index such that \(v \in\) least\(Y^i_{k_0}\) and let \(j_0\) be the minimal such that \(v \in\) greatest\(X^i_{k_0,j_0}\).

By definition of \(cpre\), P0 can control to reach in one move greatest\(X^i_{k_1,j_1}\) 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 \(v \models \neg A_{j_0}\). So by playing this strategy, P0 can ensure that either some \(A\) is visited finitely often, or reach least\(Y^i_0 \land cpre(greatestZ)\).

By repeating the same for all \(G_i\) P0 can enforce \((\bigwedge_j \square \Diamond A_j) \rightarrow (\bigwedge_i \square \Diamond G_i)\).
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 $greatestZ$:

As before, consider some $G_i$. From $v$ there is a maximum on the number of visits to $A_j$ before arriving to $G_i \land cpre(\text{win})$ (König’s lemma).

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

For $greatestZ_0 = V$ this is clear. Assume $\text{win} \subseteq greatestZ_l$. Then for every $v \in \text{win}$ it must be that $v \in leastY^i_k$ for some $k$.

1. fix ($greatestZ := V$
2. foreach ($G_i$
3.   fix ($leastY := G_i \land cpre(greatestZ)$
4.   leastY := leastY $\lor cpre(leastY)$
5.   foreach ($A_j$
6.     fix ($greatestX := V$
7.     greatestX := leastY $\lor cpre(greatestX)$
8.     end // fix greatestX
9.     leastY := leastY $\lor greatestX$
10.   end // foreach $A_j$
11.   end // fix leastY
12. greatestZ := leastY
13. end // foreach $G_i$
14. end // fix greatestZ

Games and Synthesis, EATCS Young Researchers School, Telč, Summer 2014

N. Piterman
Ranking (again)

• The proof established a set of rankings for the winning states – the number of assumptions met until reaching the goal.
  \[ r_i : V \rightarrow (\mathbb{N} \times |\{A_j\}|) \cup \{\infty\} \]

• Can we compute the rank iteratively?
  – A path that visits more than \(|A_j|\) nodes must be a losing loop.
  – Restrict rank to \( r_i : V \rightarrow (\{0, \ldots, \max|A_j|\} \times |\{A_j\}|) \cup \{\infty\} \).
  – \( \text{best}_i(v) = \begin{cases} \min_{(v,w)\in E} r_i(w) & v \in V_0 \\ \max_{(v,w)\in E} r_i(w) & v \in V_1 \end{cases} \)
  – Rank is stable if:
    • \( v \in G_i \) and \( \forall i. r_i(\text{best}_i(v)) < \infty \).
    • \( v \notin G_i, v \in A_j, r_i(v) = (l,j), \) and \( r_i(\text{best}_i(v)) < r_i(v) \).
    • \( v \notin G_i, v \notin A_j, r_i(v) = (l,j), \) and \( r_i(\text{best}_i(v)) \leq r_i(v) \).
Compute Rank Directly

1. $r_i := \lambda v. 0$
2. while ($\exists v, i. r_i(v)$ not stable)
3. \[ r_i(v) := \text{inc}_i(\text{best}(v)) \]
4. end // while

- Each $v$ in each rank can be increased at most $|V||\{A_i\}|$ times.
- By evaluating loop condition (and $\text{best}$) efficiently, all work can be restricted to $O(|\{A_i\}| \cdot |\{G_i\}| \cdot |V| \cdot |E|)$. 

Memorizing Intermediate Values

Generalized Reactivity (1):
1. fix (greatestZ := V)
2. foreach (Gi)
3. cY := 0;
4. fix (leastY := Gi ∧ cpre(greatestZ))
5. leastY := leastY ∨ cpre(leastY);
6. foreach (Aj)
7. fix (greatestX := V)
8. greatestX := least ∀ (¬Aj ∧ cpre(greatestX))
9. end // fix greatestX
10. x[Gi][cY][Aj] := greatestX;
11. leastY := leastY ∨ greatestX;
12. end // foreach A
13. y[Gi][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

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

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

- Set let \(m = |\{G_i\}|\) and \(n = |\{A_i\}|.\)

- Construct a machine \(M\) realizing \(\varphi:\)

\[M = \langle 2^j, 2^o, 2^{j \cup o} \times [1..m] \cup \{s_0\}, \rho, s_0, L \rangle:\]

\[\rho(s_0, i) = \begin{cases} \theta_s & i \models \theta_e \\ T & i \models \neg \theta_e \end{cases}\]

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

Generalized Reactivity (1):
1. fix (greatestZ := \( V \))
2. foreach (\( G_i \))
3. \( cY := 0 \);
4. fix (leastY := \( G_i \land \text{cpre}(\text{greatestZ}) \))
5. leastY := leastY \lor \text{cpre}(\text{leastY})
6. foreach (\( A_j \))
7. fix (greatestX := y[\( G_i \)][maxprev])
8. greatestX := least \lor (\neg A_j \land \text{cpre}(\text{greatestX}))
9. end // fix greatestX
10. \( x[\( G_i \)][cY][A_j] := \text{greatestX} \)
11. leastY := leastY \lor greatestX
12. end // foreach A
13. \( y[\( G_i \)][cY] := \text{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:
  \[-r_1 \land \neg r_2\]
• Transition:
  \(\neg r_1 \land \neg g_1 \land \bigcirc r_1 \rightarrow (r_2 \leftrightarrow \bigcirc r_2)\) \lor
  \(r_1 \land g_1 \land \bigcirc \neg r_1 \rightarrow (r_2 \leftrightarrow \bigcirc r_2)\) \lor
  \(\neg r_2 \land \neg g_2 \land \bigcirc r_2 \rightarrow (r_1 \leftrightarrow \bigcirc r_1)\) \lor
  \(r_2 \land g_2 \land \bigcirc \neg r_2 \rightarrow (r_1 \leftrightarrow \bigcirc r_1)\) \lor
  \((r_1 \leftrightarrow \bigcirc r_1) \land (r_2 \leftrightarrow \bigcirc r_2)\)
• 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:
  \((\neg g_1 \lor \neg g_2) \land
  \left((g_1 \neq \bigcirc r_1 \land (g_2 \leftrightarrow \bigcirc g_2)) \lor
  (g_2 \neq \bigcirc r_2 \land (g_1 \leftrightarrow \bigcirc g_1)) \lor
  (g_1 \leftrightarrow \bigcirc g_1) \land (g_2 \leftrightarrow \bigcirc g_2)\right)\)
• Good things:
  \(\Box \Diamond (g_1 = r_1) \land \Box \Diamond (g_2 = r_2)\)
Result of Synthesis
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 \(2^{J \cup \emptyset}\).
- 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.
What we actually do

\[
(\theta_e \rightarrow \theta_s) \land (\theta_e \rightarrow \Box((\Box \rho_e) \rightarrow \rho_s)) \land (\theta_e \land \Box \rho_e \rightarrow ((\land j \Box \Diamond A_j) \rightarrow (\land i \Box \Diamond G_i))
\]

Lemma. If the above is realizable then the implication is realizable.

Consider a computation $c$ satisfying the above. Then, $c$ satisfies the implication as well.
1. If $c \models \theta_e$ then clearly $c$ satisfies the implication.
2. If $c \models \Box \rho_e$ then clearly $c$ satisfies the implication.
3. If $c \models (\land j \Box \Diamond A_j)$ then clearly $c$ satisfies the implication.
4. If $c \models \theta_e$, $c \models \Box \rho_e$, and $c \models (\land j \Box \Diamond A_j)$ then from the first conjunct $c \models \theta_s$, from the second conjunct $c \models \Box \rho_s$, and from the third conjunct $c \models (\land j \Box \Diamond G_i)$. 
What about the other direction?

Well, the other direction does not hold.

Example. Let $x$ and $y$ be Boolean input and output variables. Consider the specification:

$$(\square(\bigcirc x) \land \square \Diamond (x \leftrightarrow y)) \rightarrow (\square(\bigcirc x \leftrightarrow \bigcirc y) \land \square \Diamond \neg y)$$

It is clearly realizable (just set $y$ to false ...).

But

$$((\square((\square \Box x) \rightarrow (\bigcirc x \leftrightarrow \bigcirc y)))$$

\[\square \bigcirc \square \Diamond \square \Diamond\]
Are such systems interesting?

- The only way to realize such a system is by violating the system’s safety requirement.
- The implication creates a dependency between the system’s safety and the environment’s liveness.
- Intuitively, the specification should not be realizable. But it is!
- Actually, the second one is more natural.
- They are different only if the environment can be forced to violate its specification.

- There is a way to handle the implication.
- It effectively reduces safety to liveness.
- Does not benefit from the embedding of the safety as transitions.
- Not going to cover.
What about saying more?

• The specification is composed of many parts. Conjunction on the left and conjunction on the right.
• What if some of these conjuncts are deterministic Buchi automata?
  – Everything should work the same!
  – Liveness added to the interested party.
Easy way to get Deterministic Buchi Automata

• The past is naturally deterministic.

\[ \varphi ::= p \| \varphi \land \varphi \| \neg \varphi \| \lozenge \varphi \| \varphi S \varphi \]

• Automata for past formulas are deterministic and acceptance free.

• Given a past formula \( \psi \), \( \square \psi \), and \( \square \lozenge \psi \) are easily converted to deterministic Buchi automata.

• Examples:
  - \( \square (r \rightarrow \lozenge g) \equiv \square \lozenge \neg (\neg g S (\neg g \land r)) \)
  - \( \square (a \land \bigcirc b \rightarrow \bigcirc \bigcirc c) \equiv \square (\neg a \land b \rightarrow \bigcirc c) \)
  - \( \square (a \rightarrow a U b) \equiv \square \Box (a \rightarrow \bigcirc (a \lor b)) \land \square \lozenge (\neg a \lor b) \)
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)

From BGJPPW07
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

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
Fig. 1. *Left.* Alice, Team Caltech’s entry in the 2007 DARPA Urban Challenge. *Right.* Alice’s planner-controller subsystem.
Bibliography

Lectures Outline

• Introduction
• Automata and Linear Temporal Logic
• Games and Synthesis
• General LTL Synthesis
• Bypassing Determinization
• Practical Issues with Synthesis
Is Implication the Right Thing?

• We’ve seen that

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

is handled by restricting permitted moves and solving

\[
(\land_j \Box \Diamond A_j) \rightarrow (\land_i \Box \Diamond G_i)
\]

Example. Let \(x\) and \(y\) be Boolean input and output variables. Consider the specification:

\[
(\Box(\Diamond x) \land \Box \Diamond (x \leftrightarrow y)) \rightarrow (\Box(\Diamond x \leftrightarrow \Diamond y) \land \Box \Diamond \neg y)
\]

It is clearly realizable (just set \(y\) to false …).

But

\[
\left(\left(\Box((\Box \Box \Box) \rightarrow (\Box x \leftrightarrow \Box y))\right) \land \Box \Diamond \Box \Diamond \Box \Diamond
\]

Games and Synthesis, EATCS Young Researchers School, Telč, Summer 2014
Compatible Environment

• An environment is **compatible** if it does not need help to fulfil the assumptions.

• Stated in games form:
  - **Remove** all obligations of the system.
  - Make sure it can win only by **forcing** environment to **lose**.
    
    \[
    (\theta_e \land \square \rho_e \land (\land_j \square \Diamond A_j)) \rightarrow (T \land \square T \land (\land_i \square \Diamond F))
    \]
  - Or the embedded version:
    
    \[
    (\theta_e \rightarrow T) \land (\theta_e \rightarrow \square ((\square \rho_e) \rightarrow T)) \land (\theta_e \land \square \rho_e \rightarrow ((\land_j \square \Diamond A_j) \rightarrow (\land_i \square \Diamond F)))
    \]
  - Both equivalent to:
    
    \[
    (\theta_e \land \square \rho_e \land (\land_j \square \Diamond A_j)) \rightarrow \square \Diamond F
    \]

• So we can use the same game settings. But:
  - The environment is **compatible** if from **every** state of the modified game the system **loses**.
  - For compatible environments the two are the **same**.
Compatibility $\rightarrow$ Same

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

• Suppose that you win for the game with implication but lose the embedded.
• The winning strategy in implication uses an unsafe transition.
• Once the unsafe transition is crossed, the environment has a strategy that shows compatibility.
• Using that strategy, the environment realizes

\[
\theta_e \land \square \rho_s \land \land_i \square \diamond A_i
\]

• But then the play satisfies:

\[
(\theta_e \land \square \rho_s \land \land_i \square \diamond A_i) \land (\square \neg \rho_e \lor \neg \theta_e)
\]
\[\square(\neg\text{overheated} \land (\text{moveToFelt} \Rightarrow \text{CookedTwice})) \land \square \lozenge \text{finishedCooking} \Rightarrow \square \lozenge \text{moveToFelt}\]
Good Features

• **Best Effort Controller:**
  – Will avoid assumptions if this is the only way to guarantee goals.

• **Assumption Preserving:**
  – Will only avoid assumptions if it is impossible to fulfill them.

• In compatible environments all possible controllers are both.
Abstracting Real Time

• **Discrete controllers** are augmented with **continuous controllers**.
• The **discrete** model does not capture **time** it takes to cross a transition.
• How to combine?
Direct Mapping

- **States** correspond to **world status**.
- A **change in state** corresponds to activation of continuous controller.
  - Sense for environment actions.
  - Upon change, activate “slow controllers”.
  - Upon completion of “slow controllers” activate “fast controllers”.
  - If something happens during execution of controller “change your mind.”
“Robot starts in region r1 with the camera off”
\[ r_1 \land \neg \text{camera} \]

“Activate the camera if and only if you see a person”
\[ \land \Box (\Diamond \text{person} \leftrightarrow \Diamond \text{camera}) \]

“Visit r2”
\[ \land \Box \Diamond (\varphi_{r_2}) \]
Lecture 5: Practical Issues

Games and Synthesis, EATCS Young Researchers School, Telč, Summer 2014
Problems with Direct Mapping

- **Delayed response** to environment changes.
  - Fast actions wait until completion of slow actions.
- **Unsafe** continuous execution even though discrete controller is provably correct.
- **Fast actions first:**
  - Change synthesis algorithm to ensure safety:
    - Intermediate states in transitions.
    - OK to complete fast before slow?
  - **Committed** to a transition once part of it complete.
    - Unable to change your mind.
    - Even longer response time.
“Do not activate the camera in r1”

$$\Box(\neg(\pi_{\text{camera}} \land \pi_{r_1}))$$
Embedding “Actions”

- **Transitions** are **instantaneous** and **states** allow for **time pass** (timed automata ...).
- Robot **starts** all continuous controllers.
- Environment **terminates** them!
  - Every action needs a sensor for termination.
  - All controllers start immediately.
  - Controller active but not completed captured in the state.
- **Discrete model** captures the **passage** of time!
camera and motion to $r_2$ not activated
\[ \neg \text{camera}, \neg r_2, \neg c_{\text{camera}}, \neg c_{r_2} \]
camera and motion activated, neither completed
\[ \text{camera}, r_2, \neg c_{\text{camera}}, \neg c_{r_2} \]
camera completed (on), motion activated
\[ \text{camera}, r_2, c_{\text{camera}}, \neg c_{r_2} \]
camera and motion both complete
\[ \text{camera}, r_2, c_{\text{camera}}, c_{r_2} \]
Problems

• How to ensure continuous controllers terminate?
  – Fairness limiting environment.
  – Can be captured by extra state variables or ...
  – Allow transition fairness!
• Increased state space.
  – Mitigated by handling transition fairness in addition to state fairness.
Changing the Control Predecessor

For a set of transitions $R \subseteq V \times V$, define

$$cpre(R) = \{ v | \forall x \in 2^J . \exists y \in 2^O . (v, x \cup y) \in R \}$$

For a set of nodes $W$, define

$$next(W) = \{(v, w)|w \in W\}$$

Generalized Reactivity (1):

1. $\text{fix } (\text{greatestZ} := V)$
2. $\text{foreach } (G_i)$
3. $\text{fix } (\text{leastY} := G_i \land next(\text{greatestZ}))$
4. $\text{leastY} := \text{leastY} \lor next(\text{leastY})$
5. $\text{foreach } (A_j)$
6. $\text{fix } (\text{greatestX} := V)$
7. $\text{greatestX} := cpre(\text{least} \lor (\neg A_j \land next(\text{greatestX})))$
8. $\text{end } // \text{fix greatestX}$
9. $\text{leastY} := \text{leastY} \lor \text{greatestX}$
10. $\text{end } // \text{foreach } A$
11. $\text{end } // \text{fix leastY}$
12. $\text{greatestZ} := \text{leastY}$
13. $\text{end } // \text{foreach } G$
14. $\text{end } // \text{fix greatestZ}$
Advantages

- **Multiple scales** for continuous control time.
- **Abort** of action **explicitly** in the model!
- Environment changes leading to **hesitation** result in **unrealizability**!
Related Work / Open Problems

• Partial information [Chatterjee, ...].
• Stochastic elements [Chatterjee, Kucera, ...].
• Real time [Alur, Maler, Larsen, ...].
• SMT [Rybalchnko, ...].
• Distributed Synthesis [Finkbeiner, ...].
• Implication?? [Bloem, ...].
• Asynchronous synthesis [Pnueli, ...].
• Quantitative Objectives [Henzinger, Raskin, ...].
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.
Bibliography