Synthesis

Sven Schewe

University of Liverpool

AVACS Autumn School, October 2\textsuperscript{nd}, 2015
A Modest Goal

- obtain correct systems ...
A Modest Goal

- obtain correct systems . . .
- . . . without doing anything.

Specification \rightarrow Verification
Implementation \rightarrow Verification
Specification \rightarrow Synthesis

Verifiable

correct
incorrect
implementation
unrealisable
Applications

- Detection of inconsistent specifications
- Partial design verification
  (Early error detection)
- Error localisation
- Automated prototyping
Synthesis

Specification

ATL, ATL*, CL
alternating-time $\mu$-calculus

Specification & Architecture

+ CTL, LTL, CTL*
+ $\mu$-calculus

Requirement: The scientist can get as much coffee as she likes.

ATL*: $\langle\text{scientist}\rangle\square \Diamond \text{get}_{\text{coffee}}$
Synthesis

Specification

ATL, ATL*, CL
alternating-time $\mu$-calculus

Speciﬁcation
& Architecture

+ CTL, LTL, CTL*
+ $\mu$-calculus

Requirement: The scientist can get as much coffee as she likes.

LTL: $\square (\text{want}_{\text{coffee}} \rightarrow \Diamond \text{get}_{\text{coffee}})$

CTL: $\forall \square (\text{want}_{\text{coffee}} \rightarrow \forall \Diamond \text{get}_{\text{coffee}})$
Synthesis

Specification

ATL, ATL*, CL
alternating-time $\mu$-calculus

Requirement: The scientist can get as much coffee as she likes.

$LTL$: $\Box (want_{coffee} \rightarrow \Diamond get_{coffee})$

$CTL$: $\forall \Box (want_{coffee} \rightarrow \forall \Diamond get_{coffee})$

Specification & Partial Design

+ CTL, LTL, CTL*
+ $\mu$-calculus

controller

brew!

grunb!

doone

g_info

sensor

get_{coffee}

want_{coffee}

environment
Synthesis

Specification

ATL, ATL*, CL
alternating-time $\mu$-calculus

Specification & Partial Design

+ CTL, LTL, CTL*
  + $\mu$-calculus

Automata-Theory

Constructive Non-Emptyness Games
The birth of the synthesis problem

<table>
<thead>
<tr>
<th>Alonzo Church</th>
<th>Summer Institute of Symbolic Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cornell University</td>
<td>1957</td>
</tr>
</tbody>
</table>

Given a requirement which a circuit is to satisfy, we may suppose the requirement expressed in some suitable logistic system which is an extension of restricted recursive arithmetic. The synthesis problem is then to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit).
Church’s Solvability Problem

Given: an interface specification
(identification of input and output variables)
and a behavioural specification \( \varphi \)

Sought: an implementation \( (\text{Input}^* \rightarrow \text{Output}) \), satisfying \( \varphi \)
Church’s Solvability Problem

Given: an interface specification
(identification of input and output variables)
and a behavioural specification $\varphi$

Sought: an implementation ($\text{Input}^* \rightarrow \text{Output}$), satisfying $\varphi$

$$\text{Env} \parallel \text{Proc} \models \varphi$$
Church’s Solvability Problem

Given: an interface specification
   (identification of input and output variables)
   and a behavioural specification $\varphi$

Sought: an implementation ($Input^* \rightarrow Output$), satisfying $\varphi$

$\exists \text{Proc} \ \forall \text{Env}. \ \text{Env} \parallel \text{Proc} \models \varphi$
Part I

But how? History and Simplicity of Synthesis
Synthesis through the ages

1963  Church’s solvability problem
1969  Büchi and Landweber, finite games of infinite duration
1969  Rabin’s solution based on determinising \( \omega \)-automata

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>LTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1989  Pnueli and Rosner</td>
<td></td>
</tr>
<tr>
<td>2005  Kupferman and Vardi “Safraless”</td>
<td></td>
</tr>
<tr>
<td>2007  S and Finkbeiner “Büchiless”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tools</th>
<th>LTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009  Filiot, Jin, and Raskin (Antichain)</td>
<td></td>
</tr>
<tr>
<td>2010  Ehlers (BDD)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Algorithms
in the vicinity of synthesis

Tree Automata
1. Projection-simple
2. Narrowing / information hiding-simple

Word Automata
1. determinisation-difficult
Algorithms
in the vicinity of synthesis

Tree Automata
1. Projection simple
2. Narrowing / information hiding simple

Word Automata
1. determinisation difficult

But why is it difficult?
Finite and Büchi Automata

Finite Automata
interpreted over **finite words**
here: over $\Sigma = \{a, b\}$

**run:** start at some **initial state**
stepwise: read an **input** letter, and
traverse the automaton respectively

**accepting:** is in a **final state** after processing the complete word

**language:** words with accepting runs
here: $\Sigma^* \setminus \{\varepsilon\}$
Finite and Büchi Automata

Büchi Automata

interpreted over **infinite words**

here: over $\Sigma = \{a, b\}$

run: start at some **initial state**

stepwise: read an **input** letter, and

traverse the automaton respectively

accepting: is **infinitely often** in a **final state** while processing

the complete $\omega$-word

language: words with accepting runs

here: $\omega$-words with **finitely many** $a$'s
Determinisation of Finite Automata
Determinisation of Büchi Automata

![Graph of Büchi Automata](image)

**Deterministic Büchi Automata** ...  
...are **less expressive** than nondeterministic Büchi automata.

**Example Language:** All words with finitely many $a$’s

Construct an input word by repeatedly
- choosing $b$’s until a final state is reached
- choosing an $a$ once.

$\Rightarrow$ determinisation requires **more involved acceptance condition**
Determinisation of Büchi Automata

Muller Automata

Table of acceptable infinity sets.

finitely many a’s: \( \{ \{ \beta \} \} \)
**Determinisation of Büchi Automata**

![Büchi Automata Diagram](image)

<table>
<thead>
<tr>
<th>Muller Automata</th>
<th>normal forms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rabin:</strong></td>
<td>list of pairs ((A_i, R_i)) of accepting and rejecting states</td>
</tr>
<tr>
<td></td>
<td>for <strong>some pair</strong>, some accepting and <strong>no rejecting</strong> state occurs <strong>infinitely often</strong></td>
</tr>
<tr>
<td><strong>Streett:</strong></td>
<td>list of pairs ((A_i, R_i)) of accepting and rejecting states</td>
</tr>
<tr>
<td></td>
<td>(dual case)</td>
</tr>
<tr>
<td></td>
<td>for <strong>all pairs</strong>, some accepting or <strong>no rejecting</strong> state occurs <strong>infinitely often</strong></td>
</tr>
<tr>
<td><strong>Parity:</strong></td>
<td>priority function states (\rightarrow \mathbb{N})</td>
</tr>
<tr>
<td></td>
<td>lowest priority occurring infinitely often is <strong>even</strong></td>
</tr>
<tr>
<td></td>
<td>Rabin chain or Streett double chain condition</td>
</tr>
</tbody>
</table>
### Determinisation of Büchi Automata

![Diagram of Büchi Automata]

#### Muller Automata

<table>
<thead>
<tr>
<th>Muller Automata</th>
<th>normal forms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rabin:</strong></td>
<td>- list of pairs ((A_i, R_i)) of accepting and <strong>rejecting</strong> states</td>
</tr>
<tr>
<td></td>
<td>- for <strong>some pair</strong>, some accepting <strong>and</strong> no rejecting state occurs <strong>infinitely often</strong></td>
</tr>
<tr>
<td><strong>Streett:</strong></td>
<td>- list of pairs ((A_i, R_i)) of accepting and <strong>rejecting</strong> states <strong>(dual case)</strong></td>
</tr>
<tr>
<td></td>
<td>- for <strong>all pairs</strong>, some accepting <strong>or</strong> no rejecting state occurs <strong>infinitely often</strong></td>
</tr>
<tr>
<td><strong>Parity:</strong></td>
<td>- priority function states (\rightarrow \mathbb{N})</td>
</tr>
<tr>
<td></td>
<td>- lowest priority occurring infinitely often is <strong>even</strong></td>
</tr>
<tr>
<td></td>
<td>- Rabin chain or Streett double chain condition</td>
</tr>
</tbody>
</table>
### Determinisation of Büchi Automata

![Determinisation of Büchi Automata](image)

<table>
<thead>
<tr>
<th>Muller Automata</th>
<th>normal forms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rabin:</strong></td>
<td>list of pairs ((A_i, R_i)) of accepting and rejecting states</td>
</tr>
<tr>
<td></td>
<td>for <strong>some pair</strong>, some accepting and <strong>no rejecting state</strong> occurs <strong>infinitely often</strong></td>
</tr>
<tr>
<td><strong>Streett:</strong></td>
<td>list of pairs ((A_i, R_i)) of accepting and rejecting states</td>
</tr>
<tr>
<td></td>
<td>(dual case) for <strong>all pairs</strong>, some accepting or <strong>no rejecting state</strong> occurs <strong>infinitely often</strong></td>
</tr>
<tr>
<td><strong>Parity:</strong></td>
<td>priority function states (\rightarrow \mathbb{N})</td>
</tr>
<tr>
<td></td>
<td>lowest priority occurring infinitely often is <strong>even</strong></td>
</tr>
<tr>
<td></td>
<td>Rabin chain or Streett double chain condition</td>
</tr>
</tbody>
</table>
Algorithms

determinising $\omega$-automata

1969 Rabin’s solution based on determinising $\omega$-automata
1988 **Safra** $O(n)$
1988 Michel $\theta(n)$
2006 Piterman $O(n!^2)$ (bound by [S08])
2008 S $O((cn)^n)$ with $c \approx 1.65$ Rabin
2008 Colcombet and Zdanowski $\theta((cn)^n)$ Rabin
2012 S and Varghese determinising GBA
2014 S and Varghese $\theta(n!^2)$ parity and Streett
Part II

Warm-Up: LTL – Automata & Simple Cases
Automata & Games

- LTL
- LTL $\Rightarrow$ alternating word automata ($\mathcal{AA}$)
- $\mathcal{AA} \Rightarrow$ acceptance game for traces
- $\mathcal{AA} \not\Rightarrow$ existence game for traces
- $\mathcal{NBA} \Rightarrow$ existence game for traces
- $\mathcal{NBA}$ and model checking
Linear-Time Temporal Logics
– as a word language –

**LTL formulas**

\[ \varphi ::= \text{true} \mid p \mid \lnot \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid \bigcirc \varphi \mid \square \varphi \mid \Diamond \varphi \mid \varphi U \varphi \]

| \(p\): | \(p\) | \(p\) | \(p\) | \(p\) | \(p\) | \(p\) | \(p\) | \(p\) | \(p\) |
| \(\bigcirc \varphi\): | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) |
| \(\Diamond \varphi\): | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) |
| \(\square \varphi\): | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) |
| \(\varphi U \psi\): | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\varphi\) | \(\psi\) |
Linear-Time Temporal Logics

- a backwards aproach -

$\Box \Diamond \bigcirc p \lor \bigcirc \neg p$

a harmless tautology

<table>
<thead>
<tr>
<th>$p$:</th>
<th>$p$</th>
<th>$p$</th>
<th>$p$</th>
<th>$p$</th>
<th>$p$</th>
<th>$p$</th>
<th>$p$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc p$:</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\bigcirc \neg p$:</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\bigcirc p \lor \bigcirc \neg p$:</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\Box \bigcirc p \lor \bigcirc \neg p$:</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\Box \Diamond \bigcirc p \lor \bigcirc \neg p$:</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The truth table shows that $\bigcirc p \lor \bigcirc \neg p$ is a tautology, meaning it is true under all possible scenarios.
Acceptance Game

\[\Box \Diamond \bigcirc p \lor \bigcirc \neg p\]

1. \[\Box \Diamond \bigcirc p \lor \bigcirc \neg p \rightarrow \Box \Diamond \bigcirc p \lor \bigcirc \neg p \land \Diamond \bigcirc p \lor \bigcirc \neg p\]
2. \[\Diamond \bigcirc p \lor \bigcirc \neg p \rightarrow \Box \Diamond \bigcirc p \lor \bigcirc \neg p \lor \bigcirc p \lor \bigcirc \neg p\]
3. \[\bigcirc p \lor \bigcirc \neg p \rightarrow \bigcirc p \lor \bigcirc \neg p\]
4. \[\bigcirc p \rightarrow \bigcirc p\]
5. \[\bigcirc \neg p \rightarrow \bigcirc \neg p\]

\[p: \quad \begin{array}{cccccccc} p & p & p & p & p & p & p & p & \ldots \end{array}\]
## Emptiness Game

| 1 | □ ◇ ◇ ○ p ∨ ○ ¬p → ○ □ ◇ ○ p ∨ ○ ¬p ∧ ◇ ○ p ∨ ○ ¬p |
| 2 | ◇ ○ p ∨ ○ ¬p → ○ ◇ ○ p ∨ ○ ¬p ∨ ○ p ∨ ○ ¬p |
| 3 | ○ p ∨ ○ ¬p → ○ p ∨ ○ ¬p |
| 4 | ○ p → ○ p |
| 5 | ○ ¬p → ○ ¬p |
## Emptiness Game

<table>
<thead>
<tr>
<th></th>
<th>□ ◐ ◐ p ∧ ◐ ¬p</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>□ ◐ ◐ p ∧ ◐ ¬p → ◐ ◐ ◐ p ∧ ◐ ¬p ∧ ◐ ◐ p ∧ ◐ ¬p</td>
</tr>
<tr>
<td>2</td>
<td>◐ ◐ p ∧ ◐ ¬p → ◐ ◐ p ∧ ◐ ¬p ∧ ◐ p ∧ ◐ ¬p</td>
</tr>
<tr>
<td>3</td>
<td>◐ p ∧ ◐ ¬p → ◐ p ∧ ◐ ¬p</td>
</tr>
<tr>
<td>4</td>
<td>◐ p → ◐ p</td>
</tr>
<tr>
<td>5</td>
<td>◐ ¬p → ◐ ¬p</td>
</tr>
</tbody>
</table>

- the acceptance player can cheat
  by using previous choices of the rejection player when constructing a “model”
The acceptance player can cheat by using previous choices of the rejection player when constructing a "model".
Acceptance Game
– non-deterministic automata –

\[ \Box \Diamond \Box p \lor \Box \neg p \]

\begin{enumerate}
\item \{\Box, \Diamond, \Box p \lor \Box \neg p, \Box p, p\}
\item \{\Box, \Diamond, \Box p \lor \Box \neg p, \Box p, \neg p\}
\item \{\Box, \Diamond, \Box p \lor \Box \neg p, \Box \neg p, p\}
\item \{\Box, \Diamond, \Box p \lor \Box \neg p, \Box \neg p, \neg p\}
\item \{\Box, \Diamond, p\}
\item \{\Box, \Diamond, \neg p\}
\end{enumerate}

\[ p: \quad \begin{array}{cccccccccc}
p & p & p & p & p & p & p & p & \ldots \\
\end{array} \]

\[ GBA: \quad \begin{array}{cccccccccc}
3 & 2 & 1 & 1 & 3 & 2 & 1 & 1 & 1 & 1 \end{array} \]
Emptiness Game
– non-deterministic automata –

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$\Box \Diamond \Diamond \quad \Box p \lor \Diamond \neg p$</th>
<th>$GBA$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\Box \Diamond \Diamond \Box p \lor \Diamond \neg p, \Box p, p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$\Box \Diamond \Diamond \Box p \lor \Diamond \neg p, \Box p, \neg p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$\Box \Diamond \Diamond \Box p \lor \Diamond \neg p, \Box \neg p, p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$\Box \Diamond \Diamond \Box p \lor \Diamond \neg p, \Box \neg p, \neg p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$\Box \Diamond \Diamond \Box p$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$\Box \Diamond \Diamond \Box \neg p$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Model Checking Game
– non-deterministic automata –

\[ \square \Diamond \Box p \lor \Box \neg p \]

1. \{\square, \Diamond, \Box p \lor \Box \neg p, \Box p, p\}
2. \{\square, \Diamond, \Box p \lor \Box \neg p, \Box p, \neg p\}
3. \{\square, \Diamond, \Box p \lor \Box \neg p, \Box \neg p, p\}
4. \{\square, \Diamond, \Box p \lor \Box \neg p, \Box \neg p, \neg p\}
5. \{\square, \Diamond, p\}
6. \{\square, \Diamond, \neg p\}

\[ \models \varnothing \]
Model Checking Game
– non-deterministic automata –

\( \square \Diamond \bigcirc p \lor \bigcirc \neg p \)

1. \( \{ \square, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, p \} \)
2. \( \{ \square, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, \neg p \} \)
3. \( \{ \square, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, p \} \)
4. \( \{ \square, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, \neg p \} \)
5. \( \{ \square, \Diamond, p \} \)
6. \( \{ \square, \Diamond, \neg p \} \)

---

\( \nabla \neq \emptyset \)
Model Checking Game
– non-deterministic automata –

\( \Box \Diamond \bigcirc p \lor \bigcirc \neg p \)

1. \{\( \Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, p \}\)
2. \{\( \Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, \neg p \}\)
3. \{\( \Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, p \}\)
4. \{\( \Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, \neg p \}\)
5. \{\( \Box, \Diamond, p \}\)
6. \{\( \Box, \Diamond, \neg p \}\)

\( \Gamma \not\models \neg \psi \)
Model Checking Game
– non-deterministic automata –

\( \Box \Diamond \Box p \lor \Diamond \neg p \)

1 \( \{ \Box, \Diamond, \Box p \lor \Diamond \neg p, \Box p, p \} \)
2 \( \{ \Box, \Diamond, \Box p \lor \Diamond \neg p, \Box p, \neg p \} \)
3 \( \{ \Box, \Diamond, \Box p \lor \Diamond \neg p, \Box \neg p, p \} \)
4 \( \{ \Box, \Diamond, \Box p \lor \Diamond \neg p, \Box \neg p, \neg p \} \)
5 \( \{ \Box, \Diamond, p \} \)
6 \( \{ \Box, \Diamond, \neg p \} \)

\( \nabla \neq \varnothing \)
Model Checking Game

\[ \neg \Box\Diamond \bigcirc p \lor \bigcirc \neg p \]

\( UCA \)

1. \[ \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, p\} \]
2. \[ \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, \neg p\} \]
3. \[ \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, p\} \]
4. \[ \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, \neg p\} \]
5. \[ \{\Box, \Diamond, p\} \]
6. \[ \{\Box, \Diamond, \neg p\} \]

\[ \models \varnothing \]
Model Checking Game
– universal automata –

\[ \square \Diamond \Diamond \Diamond p \lor \Diamond \neg p \]
\[ \mathcal{N}\mathcal{B}\mathcal{A} \text{ for } \Diamond \Box \Diamond \neg p \land \Diamond p \]

1. \{\Diamond\}
2. \{\Diamond, \Box, \Diamond p, \Diamond \neg p\}

(blocks as \(\mathcal{N}\mathcal{B}\mathcal{A}\), accepts immediately as \(\mathcal{U}\mathcal{C}\mathcal{A}\))
Part III

Automata & Solvability
Church’s Solvability Problem

Church’s Solvability Problem – 1963

**Given:** an interface specification
(identification of input and output variables)
and a behavioural specification $\varphi$

**Sought:** an implementation ($Input^* \rightarrow Output$), satisfying $\varphi$
Automata & Games for Synthesis

Implementation

Computation Tree

\[ \approx \]

Automata-theoretic approach

<table>
<thead>
<tr>
<th>Specification $\varphi$</th>
<th>$\sim$</th>
<th>Automaton $A_\varphi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models of $\varphi$</td>
<td>$\sim$</td>
<td>Language of $A_\varphi$</td>
</tr>
<tr>
<td>Realisability of $\varphi$</td>
<td>$\sim$</td>
<td>Language Non-Emptiness of $A_\varphi$</td>
</tr>
</tbody>
</table>
Church’s Solvability Problem

– on the running example –

as before – $AA$ won’t do

how about $\neg A$?
Church’s Solvability Problem
– on the running example –

as before – \( AA \) won’t do

how about \( NA \)?
Church’s Solvability Problem

– on the running example –

\[ \Box \quad \Diamond \quad \bigcirc p \lor \bigcirc \neg p \\]

gba

1. \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, p\}
2. \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc p, \neg p\}
3. \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, p\}
4. \{\Box, \Diamond, \bigcirc p \lor \bigcirc \neg p, \bigcirc \neg p, \neg p\}
5. \{\Box, \Diamond, p\}
6. \{\Box, \Diamond, \neg p\}
Church’s Solvability Problem

– on the running example –

\[ \text{Env} \xrightarrow{p} \text{Proc} \xrightarrow{\emptyset} \]

<table>
<thead>
<tr>
<th>( \square \diamond  \bigcirc p \lor \bigcirc \neg p )</th>
<th>( \text{NBA for} \ \diamond \ \square \bigcirc \neg p \land \bigcirc p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 {( \diamond )}</td>
<td>( UCA )</td>
</tr>
<tr>
<td>2 {( \neg ), ( \square ), ( \bigcirc p ), ( \bigcirc \neg p )}</td>
<td></td>
</tr>
</tbody>
</table>

(blocks as NBA, accepts immediately as UCA)
Church’s Solvability Problem
– incomplete information –

\[ \text{Env} \rightarrow \text{Input Variables} \rightarrow \text{Proc} \rightarrow \text{Output Variables} \]

\[ AA \Rightarrow UA / NA \Rightarrow DA \] (expensive)

\[ \models \emptyset \]
Extension: Incomplete Information

\[ UA \land NA \Rightarrow DA \]

"narrowing operation" \( AA \Rightarrow AA, UA \Rightarrow UA, NA \not\Rightarrow NA \)

- if \( dir_1 \) and \( dir_2 \) are indistinguishable and
- you'd send \( s_1 \) to \( dir_1 \) and \( s_2 \) to \( dir_2 \)

\( \sim \) send \( s_1 \) and \( s_2 \) to \( dir_{12} \)

(expensive)
Extension: Incomplete Information

\[ UA \lor NA \Rightarrow DA \]

“narrowing operation” \( AA \Rightarrow AA, UA \Rightarrow UA, NA \not\Rightarrow NA \)

- if \( dir_1 \) and \( dir_2 \) are indistinguishable and
- you’d send \( s_1 \) to \( dir_1 \) and \( s_2 \) to \( dir_2 \)

\[ \sim \text{ send } s_1 \text{ and } s_2 \text{ to } dir_{12} \]
Part IV

Distributed Strategies
Distributed Synthesis

– classic results –

Decidable Architectures

Pipelines [Pnueli+Rosner 90]

Two-Way Chains [Kupferman+Vardi 01]

One-Way Rings [Kupferman+Vardi 01]

Undecidable Architecture

[Pnueli+Rosner 90]
What does a Process Know?

- process b2 knows its input
- process b2 knows its output
  ⇒ process b2 knows the input to process b3
  ⇒ process b2 knows the output of process b3
  ... least fixed point ⇒ knowledge of b2
What does a Process Know?

- process b2 knows its **input**
- process b2 knows its **output**
  - process b2 knows the input to process b3
  - process b2 knows the output of process b3
  ... least fixed point ⇒ knowledge of b2
What does a Process Know?

- process b2 knows its **input**
- process b2 knows its **output**
  ⇒ process b2 knows the **input to process b3**
  ⇒ process b2 knows the **output of process b3**
  ... least fixed point ⇒ knowledge of b2
What does a Process Know?

- process b2 knows its **input**
- process b2 knows its **output**
  ⇒ process b2 knows the **input to process b3**
  ⇒ process b2 knows the **output of process b3**
  ... least fixed point ⇒ knowledge of b2
What does a Process Know?

- process b2 knows its **input**
- process b2 knows its **output**
  \[ \Rightarrow \text{ process b2 knows the input to process b3} \]
  \[ \Rightarrow \text{ process b2 knows the output of process b3} \]
  \[ \ldots \text{ least fixed point } \Rightarrow \text{ knowledge of b2} \]
Super-Processes

- b2 is **better informed** than b3 and b4 ($b_2 \succeq b_3, b_4$)
- b2 can simulate b3 and b4

$\Rightarrow$ b2 can be used as a super-process
Decidability of Architectures

≥ is an order

processes incomparable by ≥
Information Fork

Env \rightarrow w1 \rightarrow b2 \rightarrow b3 \rightarrow b4 \rightarrow b5

b2 \sim b5 \succeq b3, b4

Undecidable
Information Fork

b1 ⊨ b5 ⊨ b3, b4

Undecidable
No Information Fork

\[ b_1 \succ b_2 \simeq b_5 \succ b_4 \]

Decidable
Decision Procedure

– ordered architecture –

- $A_\varphi$ accepts strategies for super-process
  - Automata transformation: $A_\varphi \rightarrow B_\varphi$
    - $B_\varphi$ accepts a strategy for process $b2$ iff
      - there is a strategy for process $b1$ such that
      - their composition is accepted by $A_\varphi$
  - test non-emptiness of $B_\varphi$
  - $A'_\varphi$ – accepts proper strategies for $b1$
  - test non-emptiness of $A'_\varphi$
Decision Procedure
– ordered architecture –

- $A_\varphi$ accepts strategies for super-process
- Automata transformation: $A_\varphi \rightarrow B_\varphi$
  - $B_\varphi$ accepts a strategy for process $b2$ iff
    - there is a strategy for process $b1$ such that
    - their composition is accepted by $A_\varphi$
- test non-emptiness of $B_\varphi$
- $A'_\varphi$ – accepts proper strategies for $b1$
- test non-emptiness of $A'_\varphi$
Decision Procedure
– ordered architecture –

- $A_\varphi$ accepts strategies for super-process
- **Automata transformation:** $A_\varphi \rightarrow B_\varphi$
  - $B_\varphi$ accepts a strategy for process $b2$ iff
    - there is a strategy for process $b1$ such that
    - their composition is accepted by $A_\varphi$
- test non-emptiness of $B_\varphi$
  - $A'_\varphi$ – accepts proper strategies for $b1$
  - test non-emptiness of $A'_\varphi$
Decision Procedure
– ordered architecture –

\[ \mathcal{A}_\varphi \] accepts strategies for super-process

Automata transformation: \[ \mathcal{A}_\varphi \rightarrow \mathcal{B}_\varphi \]

\[ \mathcal{B}_\varphi \] accepts a strategy for process b2 iff

- there is a strategy for process b1 such that
- their composition is accepted by \[ \mathcal{A}_\varphi \]

test non-emptiness of \[ \mathcal{B}_\varphi \]

\[ \mathcal{A}'_\varphi \] – accepts proper strategies for b1

test non-emptiness of \[ \mathcal{A}'_\varphi \]
Decision Procedure
– ordered architecture –

- $A_\phi$ accepts strategies for super-process
- **Automata transformation:** $A_\phi \rightarrow B_\phi$
  - $B_\phi$ accepts a strategy for process b2 iff
    - there is a strategy for process b1 such that
    - their composition is accepted by $A_\phi$

- test non-emptiness of $B_\phi$
- $A'_\phi$ – accepts proper strategies for b1
- test non-emptiness of $A'_\phi$
Decision Procedure
– ordered architecture –

- LTL, $UWA / UTA$, $DWA / DTA$
  - projection ($N^TA$), narrowing ($ATA$), non-determinisation $N^TA$
    - “annotate” strategy – $UTA$
    - determinise – $DTA$
    - project strategy – $N^TA$
  - test non-emptiness of $N^TA$ – TS / $DTA$
  - intersect
  - test non-emptiness
Decision Procedure
– ordered architecture –

- LTL, $UWA / UTA$, $DWA / DTA$
- projection ($NTA$), narrowing ($ATA$), non-determinisation $NTA$
  - “annotate” strategy – $UTA$
  - determinise – $DTA$
  - project strategy – $NTA$
- test non-emptiness of $NTA$ – $TS / DTA$
- intersect
- test non-emptiness
Decision Procedure
– ordered architecture –

- LTL, $UWA / UTA$, $DWA / DTA$
- projection ($N \cdot TA$), narrowing ($ATA$), non-determinisation $N \cdot TA$
  - “annotate” strategy – $UTA$
  - determinise – $DTA$
  - project strategy – $N \cdot TA$
- test non-emptiness of $N \cdot TA$ – $TS / DTA$
- intersect
- test non-emptiness
Decision Procedure

- ordered architecture -

- LTL, \textit{UWA / UTA}, \textit{DWA / DTA}
- projection (\textit{NTA}), narrowing (\textit{ATA}), non-determinisation \textit{NTA}
  - "annotate" strategy – \textit{ATA}
  - determinise – \textit{DTA}
  - project strategy – \textit{NTA}
- test non-emptiness of \textit{NTA} – TS / \textit{DTA}
- intersect
- test non-emptiness
Decision Procedure
– ordered architecture –

- LTL, $\mathcal{UWA} / \mathcal{UTA}$, $\mathcal{DWA} / \mathcal{DTA}$
- projection ($\mathcal{NTA}$), narrowing ($\mathcal{ATA}$),
  non-determinisation $\mathcal{NTA}$
  - “annotate” strategy – $\mathcal{UTA}$
  - determinise – $\mathcal{DTA}$
  - project strategy – $\mathcal{NTA}$
- test non-emptiness of $\mathcal{NTA}$ – TS / $\mathcal{DTA}$
- intersect
- test non-emptiness
Decision Procedure

– ordered architecture –

- LTL, \( UWA \) / \( UTA \), \( DWA \) / \( DTA \)
- projection \( (\mathcal{N}TA) \), narrowing \( (\mathcal{A}TA) \),
  non-determinisation \( \mathcal{N}TA \)
  - “annotate” strategy – \( \mathcal{U}TA \)
  - determinise – \( \mathcal{D}TA \)
  - project strategy – \( \mathcal{N}TA \)
- test non-emptiness of \( \mathcal{N}TA \) – TS / \( \mathcal{D}TA \)
- intersect
- test non-emptiness
Decision Procedure

Architecture transformation $\Leftrightarrow$ linear on black-box processes

- merge equivalent processes
- attach white-box processes to better informed process
- remove feedback
Decision Procedure

Architecture transformation $\Rightarrow$ linear on black-box processes

- merge equivalent processes
- attach white-box processes to better informed process
- remove feedback
Decision Procedure

Architecture transformation $\Rightarrow$ linear on black-box processes

- merge equivalent processes
- attach white-box processes to better informed process
- remove feedback
Decision Procedure

Architecture transformation \(\Rightarrow\) linear on black-box processes

- merge equivalent processes
- attach white-box processes to better informed process
- remove feedback
Decision Procedure

Ordered Architecture
- decision procedure
- adds one exponent / level of knowledge
- hardness result
Perfect – But Something ’s Wrong

**Interfaces – friend or foe?**

- **theory:** restricted information can be abused
- **practice:** then it is a specification error

**Infeasible complexity**

- **theory:** completeness result
  - maximal size of minimal model
- **practice:** no small model \( \Rightarrow \) specification error

**Redefine realisability**

- there is a feasible model
- predefined bounds on the implementation

\( \Rightarrow \) Bounded Synthesis
Perfect – But Something’s Wrong

Interfaces – friend or foe?

theory: restricted information can be abused

practice: then it is a specification error

Infeasible complexity

theory: completeness result
maximal size of minimal model

practice: no small model \(\Rightarrow\) specification error

Redefine realisability

- there is a feasible model
- predefined bounds on the implementation

\(\Rightarrow\) Bounded Synthesis
Perfect – But Something ’s Wrong

Interfaces – friend or foe?

theory: restricted information can be abused
practice: then it is a specification error

Infeasible complexity

theory: completeness result
maximal size of minimal model
practice: no small model \(\Rightarrow\) specification error

Redefine realisability

- there is a feasible model
- predefined bounds on the implementation

\(\Rightarrow\) Bounded Synthesis
Part V

Two Steps Towards Practice
Overview

Specification $\varphi$

Universal Co-Büchi Automaton

Deterministic Parity Automaton

Parameterised Deterministic Safety Automaton

Parity Emptiness Game

Parameterised Safety Emptiness Game

SMT Problem

SMT Solving

Sequence of Automata Transformations

Safra-Constructions – Exponential

Locality Constraints

Small – Usually Cheap
Example – Simplified Arbiter

Synthesis with Complete Information
From Co-Büchi to Safety

Realisable specification

- finite implementation – size $s$
- bound $b$ on the number of rejecting states – $b \leq s \cdot |F|$
- safety condition

$s$ can be bounded by the size of the resp. deterministic automaton
Parameterised Emptiness Game

\[
\text{Input: } r_1, r_2 \quad \text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, _)} \\
\text{(0, 0, 0)} \\
\text{(0, 1, _)} \\
\text{(0, 1, 0)} \\
\end{array}
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 0)} \\
\text{(0, 1, 0)} \\
\text{(0, 0, 0)} \\
\text{(0, 0, 1)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{(0, _, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, _)} \\
\text{(0, 0, 0)} \\
\text{(0, 1, _)} \\
\text{(0, 1, 0)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 0)} \\
\text{(0, 1, 0)} \\
\text{(0, 0, 0)} \\
\text{(0, 0, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, _)} \\
\text{(0, 0, 0)} \\
\text{(0, 1, _)} \\
\text{(0, 1, 0)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 0)} \\
\text{(0, 1, 0)} \\
\text{(0, 0, 0)} \\
\text{(0, 0, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, _)} \\
\text{(0, 0, 0)} \\
\text{(0, 1, _)} \\
\text{(0, 1, 0)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 0)} \\
\text{(0, 1, 0)} \\
\text{(0, 0, 0)} \\
\text{(0, 0, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, _)} \\
\text{(0, 0, 0)} \\
\text{(0, 1, _)} \\
\text{(0, 1, 0)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 0)} \\
\text{(0, 1, 0)} \\
\text{(0, 0, 0)} \\
\text{(0, 0, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, _)} \\
\text{(0, 0, 0)} \\
\text{(0, 1, _)} \\
\text{(0, 1, 0)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 0)} \\
\text{(0, 1, 0)} \\
\text{(0, 0, 0)} \\
\text{(0, 0, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]

\[
\begin{array}{c}
\text{(0, _, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\text{(0, 1, 1)} \\
\end{array}
\]

\[
\text{Input: } r_1, r_2
\]

\[
\text{Output: } g_1, g_2
\]
Parameterised Emptiness Game

Semantics of a Game Positon

- **collects** the paths of the run tree
- $i$-th position in the annotation:
  - $\_\_\_$: no path ends in automaton-state $i$
  - $n \in \mathbb{N}$: a path may end in automaton-state $i$
    - each such path has $\leq n$ previous visits to rejecting states
Parameterised Emptiness Game

\[ \text{env} \xrightarrow{\text{Input } r_1, r_2} p_1 \xrightarrow{\text{Output } g_1, g_2} \]

- \((0, _, _)\)
- \((0, 0, 0)\)
- \((0, _, 0)\)
- \((0, 0, 1)\)
- \((0, 1, 0)\)
- \((0, 1, 1)\)
- \((0, 1, _)\)
- \((0, _, 1)\)
- \((0, 0, 1)\)
- \((0, 0, 0)\)

Transition labels:
- \(g_1g_2\)
- \(g_1\)
- \(g_2\)
Parameterised Emptiness Game

\( env \) \( \xrightarrow{r_1, r_2} \) \( p_1 \) \( \xrightarrow{g_1, g_2} \) Output

\( (0, _, _) \)

\( \overline{g_1 g_2} \)

\( (0, 0, _) \)
\( (0, 0, 0) \)
\( (0, _, 0) \)

\( \overline{g_1 g_2} \)

\( (0, 1, _) \)
\( (0, 1, 1) \)

\( (0, 1, 0) \)
\( (0, 0, 1) \)

\( 2 \)
\( 3 \)

\( r_1 \) \( g_1 g_2 \) \( r_2 \)

\( 1 \)

\( * \)
Theorem – Completeness

An (input preserving) transition system is accepted by a UCB if and only if it has a valid annotation.

Proof idea

Cycle with rejecting state reachable in the run graph if and only if no valid annotation.
Progress Constraints – Automaton Transitions

output complexity: NP-complete

\[ \forall t. \lambda_1^B(t) \rightarrow \lambda_1^B(\tau_{r_1 r_2}(t)) \land \lambda_1^#(\tau_{r_1 r_2}(t)) \geq \lambda_1^#(t) \]
\[ \land \lambda_1^B(\tau_{r_1 r_2}(t)) \land \lambda_1^#(\tau_{r_1 r_2}(t)) \geq \lambda_1^#(t) \]
\[ \land \lambda_1^B(\tau_{r_1 r_2}(t)) \land \lambda_1^#(\tau_{r_1 r_2}(t)) \geq \lambda_1^#(t) \]
\[ \land \lambda_1^B(\tau_{r_1 r_2}(t)) \land \lambda_1^#(\tau_{r_1 r_2}(t)) \geq \lambda_1^#(t) \]

\[ \forall t. \lambda_1^B(t) \rightarrow \neg g_1(t) \lor \neg g_2(t) \]

\[ \forall t. \lambda_2^B(t) \land r_1(t) \rightarrow \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) \geq \lambda_2^#(t) \]
\[ \land \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) \geq \lambda_2^#(t) \]
\[ \land \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) \geq \lambda_2^#(t) \]
\[ \land \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) \geq \lambda_2^#(t) \]

\[ \forall t. \lambda_2^B(t) \land \neg g_1(t) \rightarrow \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) > \lambda_2^#(t) \]
\[ \land \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) > \lambda_2^#(t) \]
\[ \land \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) > \lambda_2^#(t) \]
\[ \land \lambda_2^B(\tau_{r_1 r_2}(t)) \land \lambda_2^#(\tau_{r_1 r_2}(t)) > \lambda_2^#(t) \]
Explicit Synthesis

Church’s Solvability Problem – 1963

Given: an interface specification
   (identification of input and output variables)
   and a behavioural specification $\varphi$

Sought: a circuit s.t. $(Input^* \rightarrow Output)$ satisfies $\varphi$

\[
TS \models \varphi
\]

CTL: EXPTIME-complete, exponential transition system

LTL: 2EXPTIME-complete, doubly exponential $TS$
Explicit vs. Succinct

counter: \( \bigwedge_{1<i \leq n} \forall \Box (p_i \iff \forall \diamond p_i) \iff \bigwedge_{j<i} p_j \)

\( \land (p_i \iff \exists \lozenge p_i) \iff \bigwedge_{j<i} p_j \)

**explicit**

transition system
Kripke structure
\( 2^n \) states

**succinct**

circuit
program
online Turing machine
tape size \( n \)
Succinct Synthesis

min-output PSPACE-complete

Is there always a small oTM?

if and only if \[ \text{PSPACE} = \text{EXPTIME} \]

Is there always a small & fast circuit?

if and only if \[ \text{EXPTIME} \subseteq \text{P/poly} \]

LTL: ditto for intermediate automata
Succinct Synthesis
min-output PSPACE-complete

Is there always a small oTM?

if and only if \( \text{PSPACE} = \text{EXPTIME} \)

Is there always a small & fast circuit?

if and only if \( \text{EXPTIME} \subseteq \text{P}/\text{poly} \)

LTL: dito for intermediate automata
Succinct Synthesis
min-output PSPACE-complete

\[
\text{Env} \quad \xrightarrow{\text{Input Variables}} \quad \text{Proc} \quad \xrightarrow{\text{Output Variables}}
\]

Is there always a small oTM?  
**CTL**

if and only if  
PSPACE = \text{EXPTIME}

Is there always a small & fast circuit?  
**CTL**

if and only if  
\text{EXPTIME} \subseteq \text{P}/\text{poly}

LTL: ditto for intermediate automata
Succinct Synthesis

Is there always a small oTM?

\[
\text{if and only if} \quad \text{PSPACE} = \text{EXPTIME}
\]

only if: guess & verify

hence: PSPACE in the minimal succinct solution

if: much harder

uses the \( DSA \) from bounded synthesis
Succinct Synthesis

Is there always a small oTM?  CTL

if and only if  PSPACE = EXPTIME

only if: guess & verify  EXPTIME-complete

hence: PSPACE in the minimal succinct solution

if: much harder
uses the DSA from bounded synthesis
Succinct Synthesis

Is there always a small oTM?

if and only if

PSPACE = EXPTIME

only if: guess & verify

hence: PSPACE in the minimal succinct solution

if: much harder

uses the DSA from bounded synthesis
PSPACE = EXPTIME \implies \text{Small Model}
Succinct & Fast Synthesis

Is there always a small & fast circuit?  
if and only if  
$\text{EXPTIME} \subseteq P/\text{poly}$

if: as before

only if: construction not enough  
encode universal space bounded ATM  
environment provides initial tape  
immediate answer to the halting problem
Succinct & Fast Synthesis

Is there always a small & fast circuit?  

if and only if \( \text{EXPTIME} \subseteq \text{P/poly} \)

if: as before

only if: construction not enough
encode universal space bounded ATM
environment provides initial tape
immediate answer to the halting problem
Succinct & Fast Synthesis

Is there always a small & fast circuit?

if and only if \( \text{EXPTIME} \subseteq \text{P/poly} \)

if: as before

only if: construction not enough
encode universal space bounded ATM
environment provides initial tape
immediate answer to the halting problem
Implementations

**General Search: Genetic Programing & Co**

- Colin G. Johnson: Genetic Programming with Fitness Based on Model Checking. EuroGP 2007: 114-124
<table>
<thead>
<tr>
<th>Search Technique</th>
<th>mutex</th>
<th>leader election</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 shared bits</td>
<td>3 shared bits</td>
</tr>
<tr>
<td>simulated annealing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution time</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>success rate</td>
<td>19</td>
<td>23</td>
</tr>
<tr>
<td>overall time</td>
<td><strong>105.26</strong></td>
<td><strong>100</strong></td>
</tr>
<tr>
<td>hybrid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution time</td>
<td>113</td>
<td>171</td>
</tr>
<tr>
<td>success rate</td>
<td><strong>31</strong></td>
<td>17</td>
</tr>
<tr>
<td>overall time</td>
<td>364.51</td>
<td>1,005.88</td>
</tr>
<tr>
<td>genetic programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution time</td>
<td>583</td>
<td>615</td>
</tr>
<tr>
<td>success rate</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>overall time</td>
<td>8,328.57</td>
<td>8,785.71</td>
</tr>
</tbody>
</table>
Part VII

summary
Automata Theoretic Approach

pro: simple
- narrowing
- projection
- determinisation (word)

pro: clean
- introduction of Partial Designs
- characterisation of the class of decidable architectures
- uniform synthesis algorithm

con: beyond price
con: does not benefit from small solutions
Bounded Synthesis

- guess implementation & verify
- NP complete in minimal transition system & $A_{\varphi}$

**pro:** complexity closer to model checking

**pro:** applicable to distributed systems

**con:** transition system vs. program / circuit
Succinct Synthesis

— is good news —

Is there always a small oTM?

if and only if

PSPACE = EXPTIME

Is there always a small & fast circuit?

if and only if

EXPTIME ⊆ P/poly
Synthesis vs. Model Checking

Bounded Synthesis of Succinct Systems

- construct a correct program / circuit
- PSPACE complete in minimal program / circuit

pro: complexity equal to model checking

pro: applicable to distributed systems
Summary

- **Distributed Synthesis**
  - decidability
  - complexity

- **Bounded Synthesis**
  - decidability
  - complexity

- **Succinct Synthesis**
  - decidability
  - complexity