Synthesis

Sven Schewe

University of Liverpool

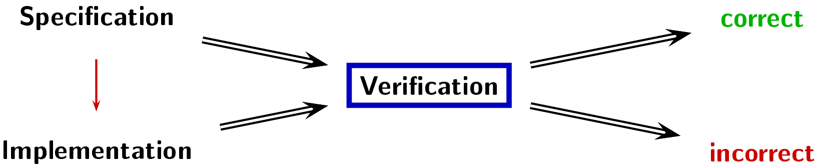
AVACS Autumn School, October 2^{nd}, 2015

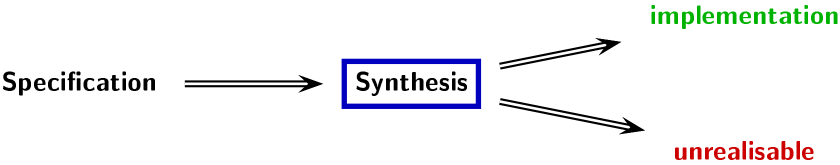
A Modest Goal

- obtain correct systems . . .

A Modest Goal

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





Applications

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

Synthesis

Specification

ATL, ATL*, CL
alternating-time μ-calculus

Specification
& Architecture

+ CTL, LTL, CTL*
+ μ-calculus

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

ATL*: $\langle\langle \mathit{scientist} \rangle\rangle \Box \Diamond \mathrm{get}_{\mathit{coffee}}$

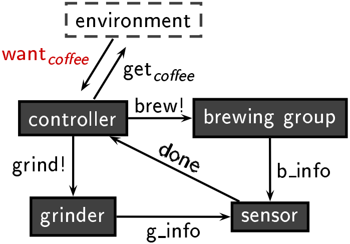
Synthesis

Specification

ATL, ATL*, CL
alternating-time μ-calculus

Specification
& Architecture

+ CTL, LTL, CTL*
+ μ-calculus



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

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

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

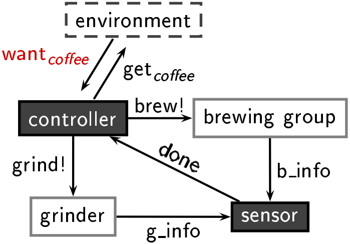
Synthesis

Specification

ATL, ATL*, CL
alternating-time μ-calculus

Specification
& Partial Design

+ CTL, LTL, CTL*
+ μ-calculus



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

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

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

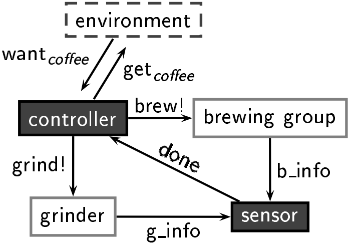
Synthesis

Specification

ATL, ATL*, CL
alternating-time μ-calculus

Specification
& Partial Design

+ CTL, LTL, CTL*
+ μ-calculus



Automata-Theory

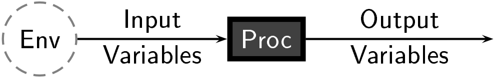
Constructive Non-Emptiness Games

The birth of the synthesis problem

Alonzo Church — Summer Institute of Symbolic Logic
Cornell University — 1957

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).

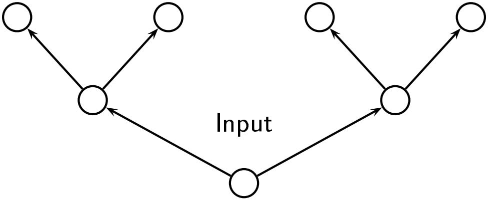
Church's Solvability Problem



Church's Solvability Problem 1963

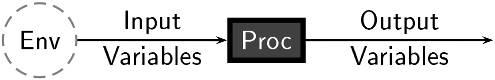
Given: an interface specification
(identification of input and output variables)
and a behavioural specification φ

Sought: an implementation ($\mathit{Input}^* \to \mathit{Output}$), satisfying φ



$\models \varphi$

Church's Solvability Problem



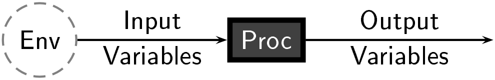
Church's Solvability Problem 1963

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

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

$$\text{Env} \parallel \text{Proc} \models \varphi$$

Church's Solvability Problem



Church's Solvability Problem — 1963

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

Sought: an implementation ($\mathit{Input}^* \to \mathit{Output}$), satisfying φ

$$\exists \mathsf{Proc}\ \forall \mathsf{Env}.\ \ \mathsf{Env} \mid\mid \mathsf{Proc} \models \varphi$$

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 infinte duration

1969 Rabin's solution based on deterministing ω-automata

Algorithms — LTL

1989 Pnueli and Rosner

2005 Kupferman and Vardi "Safraless"

2007 S and Finkbeiner "Büchiless"

Tools — LTL

2009 Filiot, Jin, and Raskin (Antichain)

2010 Ehlers (BDD)

...

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

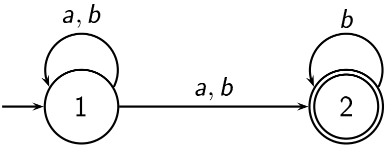
1. Projection — simple
2. Narrowing / information hiding — simple

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**

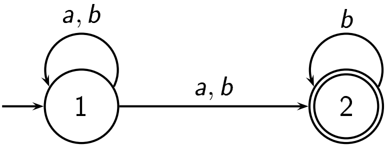
stepwise: read an **input** letter, and traverse the automaton respectively

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 $\Sigma = \{a, b\}$

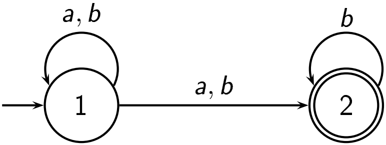
run: start at some **initial state**

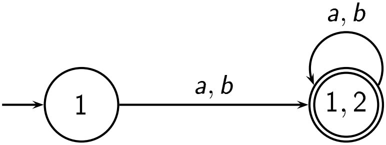
stepwise: read an **input** letter, and
traverse the automaton respectively

accepting: is **infinitely often** in a **final state** while processing the complete ω-word

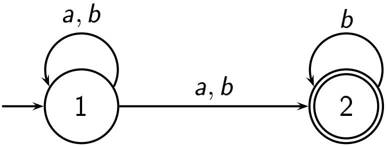
language: words with accepting runs
here: ω-words with **finitely many a's**

Determinisation of Finite Automata





Determinisation of Büchi Automata



Deterministic Büchi Automata ...

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

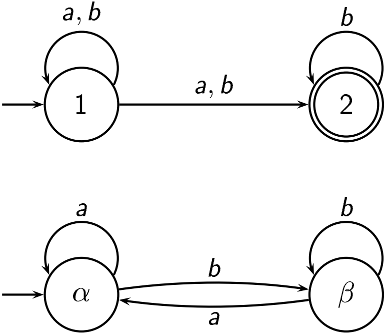
Example Language: All words with finitely many *a*'s

Construct an input word by repeatedly

- choosing *b*'s until a final state is reached
- choosing an *a* once.

⇒ determinisation requires **more involved acceptance condition**

Determinisation of Büchi Automata

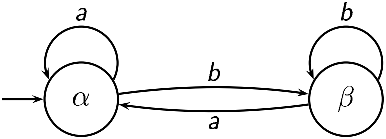


Muller Automata

Table of acceptable infinity sets.

finitely many *a*'s: $\{\{\beta\}\}$

Determinisation of Büchi Automata



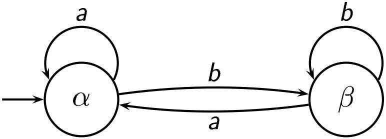
Muller Automata — normal forms

Rabin:
- list of pairs (A_i, R_i) of accepting and rejecting states
- for **some pair**, some accepting **and** no rejecting state occurs **infinitely often**

Streett:
- list of pairs (A_i, R_i) of accepting and rejecting states (dual case)
- for **all pairs**, some accepting **or** no rejecting state occurs **infinitely often**

Parity:
- priority function states $\to \mathbb{N}$
- lowest priority occurring infinitely often is **even**
- Rabin chain or Streett double chain condition

Determinisation of Büchi Automata



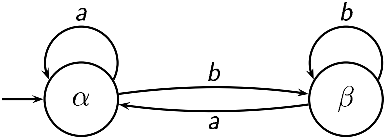
Muller Automata — normal forms

Rabin:
- list of pairs (A_i, R_i) of accepting and rejecting states
- for **some pair**, some accepting **and** no rejecting state occurs **infinitely often**

Streett:
- list of pairs (A_i, R_i) of accepting and rejecting states (dual case)
- for **all pairs**, some accepting **or** no rejecting state occurs **infinitely often**

Parity:
- priority function states $\to \mathbb{N}$
- lowest priority occurring infinitely often is **even**
- Rabin chain or Streett double chain condition

Determinisation of Büchi Automata



Muller Automata — normal forms

Rabin:
- list of pairs (A_i, R_i) of accepting and rejecting states
- for **some pair**, some accepting **and** no rejecting state occurs **infinitely often**

Streett:
- list of pairs (A_i, R_i) of accepting and rejecting states (dual case)
- for **all pairs**, some accepting **or** no rejecting state occurs **infinitely often**

Parity:
- priority function states $\to \mathbb{N}$
- lowest priority occurring infinitely often is **even**
- Rabin chain or Streett double chain condition

Algorithms

determinising ω-automata

1969	Rabin's solution based on deterministing ω-automata	
1988	**Safra** $n^{O(n)}$	
1988	Michel $n^{\theta(n)}$	
2006	Piterman $O(n!^2)$	(bound by [S08])
2008	S $O\big((cn)^n\big)$ with $c \approx 1.65$	Rabin
2008	Colcombet and Zdanowski $\theta\big((cn)^n\big)$	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

$$\varphi ::= \mathit{true} \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\,\varphi \mid \Box\,\varphi \mid \Diamond\,\varphi \mid \varphi\,\mathcal{U}\,\varphi$$

p:	p									
$\bigcirc\,\varphi$:		φ								
$\Diamond\,\varphi$:								φ		
$\Box\,\varphi$:	φ	φ	φ	φ	φ	φ	φ	φ	φ	φ
$\varphi\,\mathcal{U}\,\psi$:	φ	φ	φ	φ	φ	φ	φ	φ	ψ	

Linear-Time Temporal Logics

– a backwards aproach –

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

a harmless tautology

p:	p		p	p	p		p	p	p	...
$\bigcirc p$:		√	√	√		√	√	√	√	√
$\bigcirc \neg p$:	√				√					
$\bigcirc p \lor \bigcirc \neg p$:	√	√	√	√	√	√	√	√	√	√
$\Diamond \bigcirc p \lor \bigcirc \neg p$:	√	√	√	√	√	√	√	√	√	√
$\Box \Diamond \bigcirc p \lor \bigcirc \neg p$:	√	√	√	√	√	√	√	√	√	

Acceptance Game

$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{AA}$

1. $\underline{\Box \Diamond \bigcirc p \vee \bigcirc \neg p} \rightarrow \bigcirc \underline{\Box \Diamond \bigcirc p \vee \bigcirc \neg p} \wedge \underline{\Diamond \bigcirc p \vee \bigcirc \neg p}$
2. $\underline{\Diamond \bigcirc p \vee \bigcirc \neg p} \rightarrow \bigcirc \underline{\Diamond \bigcirc p \vee \bigcirc \neg p} \vee \underline{\bigcirc p \vee \bigcirc \neg p}$
3. $\underline{\bigcirc p \vee \bigcirc \neg p} \rightarrow \underline{\bigcirc p} \vee \underline{\bigcirc \neg p}$
4. $\underline{\bigcirc p} \rightarrow \bigcirc \underline{p}$
5. $\underline{\bigcirc \neg p} \rightarrow \bigcirc \underline{\neg p}$

p:

p		p	p	p		p	p	p	...

Emptiness Game

$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{AA}$

1. $\underline{\Box \Diamond \bigcirc p \vee \bigcirc \neg p} \rightarrow \bigcirc \underline{\Box \Diamond \bigcirc p \vee \bigcirc \neg p} \wedge \underline{\Diamond \bigcirc p \vee \bigcirc \neg p}$
2. $\underline{\Diamond \bigcirc p \vee \bigcirc \neg p} \rightarrow \bigcirc \underline{\Diamond \bigcirc p \vee \bigcirc \neg p} \vee \underline{\bigcirc p \vee \bigcirc \neg p}$
3. $\underline{\bigcirc p \vee \bigcirc \neg p} \rightarrow \underline{\bigcirc p} \vee \underline{\bigcirc \neg p}$
4. $\underline{\bigcirc p} \rightarrow \bigcirc \underline{p}$
5. $\underline{\bigcirc \neg p} \rightarrow \bigcirc \underline{\neg p}$

Emptiness Game

$\Box \Diamond \bigcirc p \wedge \bigcirc \neg p$ $\mathcal{AA}$

1. $\underline{\Box \Diamond \bigcirc p \wedge \bigcirc \neg p} \rightarrow \bigcirc \underline{\Box \Diamond \bigcirc p \wedge \bigcirc \neg p} \wedge \underline{\Diamond \bigcirc p \wedge \bigcirc \neg p}$
2. $\underline{\Diamond \bigcirc p \wedge \bigcirc \neg p} \rightarrow \bigcirc \underline{\Diamond \bigcirc p \wedge \bigcirc \neg p} \vee \underline{\bigcirc p \wedge \bigcirc \neg p}$
3. $\underline{\bigcirc p \wedge \bigcirc \neg p} \rightarrow \underline{\bigcirc p} \wedge \underline{\bigcirc \neg p}$
4. $\underline{\bigcirc p} \rightarrow \bigcirc \underline{p}$
5. $\underline{\bigcirc \neg p} \rightarrow \bigcirc \underline{\neg p}$

- the acceptance player can cheat

  by using previous choices of the rejection player when constructing a "model"

Emptiness Game

$\Box \Diamond \bigcirc p \wedge \bigcirc \neg p$ $\mathcal{AA}$

1. $\underline{\Box \Diamond \bigcirc p \wedge \bigcirc \neg p} \rightarrow \bigcirc \underline{\Box \Diamond \bigcirc p \wedge \bigcirc \neg p} \wedge \underline{\Diamond \bigcirc p \wedge \bigcirc \neg p}$
2. $\underline{\Diamond \bigcirc p \wedge \bigcirc \neg p} \rightarrow \bigcirc \underline{\Diamond \bigcirc p \wedge \bigcirc \neg p} \vee \underline{\bigcirc p \wedge \bigcirc \neg p}$
3. $\underline{\bigcirc p \wedge \bigcirc \neg p} \rightarrow \underline{\bigcirc p} \wedge \underline{\bigcirc \neg p}$
4. $\underline{\bigcirc p} \rightarrow \bigcirc \underline{p}$
5. $\underline{\bigcirc \neg p} \rightarrow \bigcirc \underline{\neg p}$

- the acceptance player can cheat

  by using previous choices of the rejection player when constructing a "model"

Acceptance Game

– non-deterministic automata –

$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ — $\mathcal{GBA}$

1. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{p}\}$
2. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{\neg p}\}$
3. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{p}\}$
4. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{\neg p}\}$
5. $\{\underline{\Box}, \underline{\Diamond}, \underline{p}\}$
6. $\{\underline{\Box}, \underline{\Diamond}, \underline{\neg p}\}$

p:	p		p	p	p		p	p	p	...
$\mathcal{GBA}$:	3	2	1	1	3	2	1	1	1	1

Emptiness Game

– non-deterministic automata –

$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{GBA}$

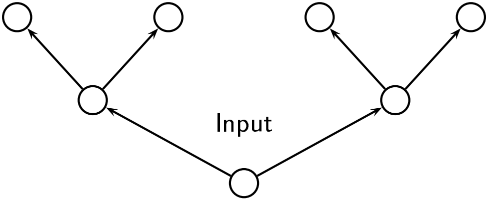
1. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{p}\}$
2. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{\neg p}\}$
3. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{p}\}$
4. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{\neg p}\}$
5. $\{\underline{\Box}, \underline{\Diamond}, \underline{p}\}$
6. $\{\underline{\Box}, \underline{\Diamond}, \underline{\neg p}\}$

Model Checking Game

– non-deterministic automata –

$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{GBA}$

1. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{p}\}$
2. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{\neg p}\}$
3. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{p}\}$
4. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{\neg p}\}$
5. $\{\underline{\Box}, \underline{\Diamond}, \underline{p}\}$
6. $\{\underline{\Box}, \underline{\Diamond}, \underline{\neg p}\}$



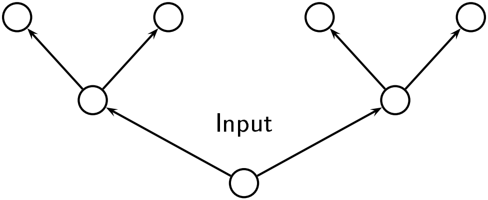
$\models \varphi$

Model Checking Game

– non-deterministic automata –

$\Box \Diamond \bigcirc p \lor \bigcirc \neg p$ $\mathcal{GBA}$

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



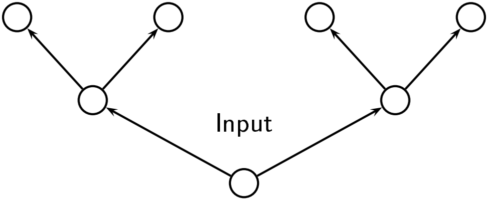
$\not\models \varphi$

Model Checking Game

– non-deterministic automata –

$\square \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{GBA}$

1. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{p}\}$
2. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{\neg p}\}$
3. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{p}\}$
4. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{\neg p}\}$
5. $\{\underline{\square}, \underline{\Diamond}, \underline{p}\}$
6. $\{\underline{\square}, \underline{\Diamond}, \underline{\neg p}\}$



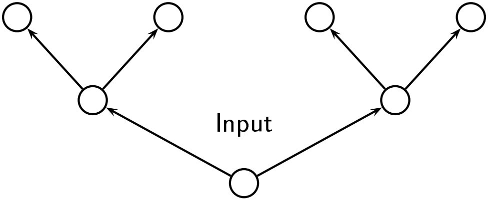
$\not\models \neg\varphi$

Model Checking Game

– non-deterministic automata –

$\Box \Diamond \bigcirc p \lor \bigcirc \neg p$ $\qquad \mathcal{GBA}$

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



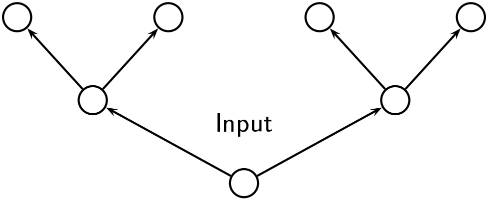
$\not\models \varphi$

Model Checking Game

– universal automata –

$\neg\Box\Diamond\bigcirc p \vee \bigcirc\neg p$ $\mathcal{UCA}$

1. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{p}\}$
2. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{\neg p}\}$
3. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{p}\}$
4. $\{\underline{\Box}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{\neg p}\}$
5. $\{\underline{\Box}, \underline{\Diamond}, \underline{p}\}$
6. $\{\underline{\Box}, \underline{\Diamond}, \underline{\neg p}\}$



$\models \varphi$

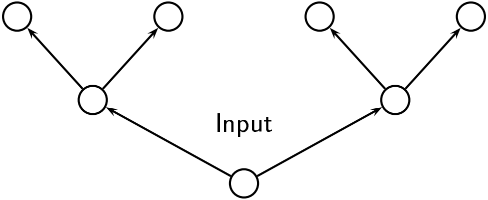
Model Checking Game

– universal automata –

$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{UCA}$

$\mathcal{NBA}$ for $\Diamond \Box \bigcirc \neg p \wedge \bigcirc p$

1. $\{\underline{\Diamond}\}$
2. $\{\underline{\Diamond}, \underline{\Box}, \underline{\bigcirc p}, \underline{\bigcirc \neg p}\}$
   (blocks as $\mathcal{NBA}$, accepts immediately as $\mathcal{UCA}$)

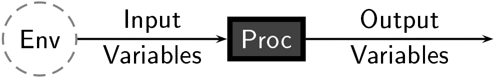


$\models \varphi$

Part III

Automata & Solvability

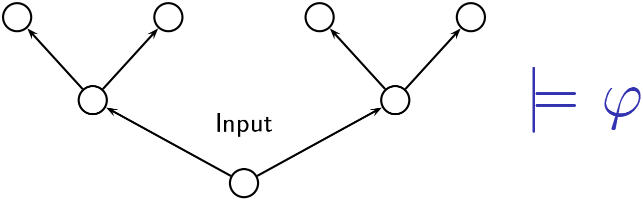
Church's Solvability Problem



Church's Solvability Problem – 1963

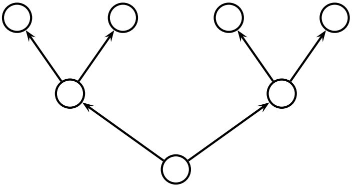
Given: an interface specification
(identification of input and output variables)
and a behavioural specification φ

Sought: an implementation ($\mathit{Input}^* \to \mathit{Output}$), satisfying φ



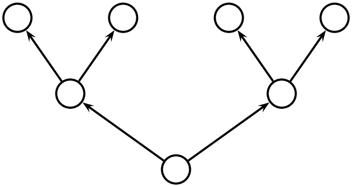
Automata & Games for Synthesis

Implementation



$\approx$

Computation Tree

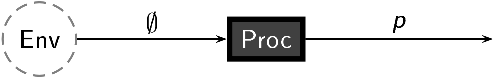


Automata-theoretic approach

Specification φ	$\sim$	Automaton $\mathcal{A}_\varphi$
Models of φ	$\sim$	Language of $\mathcal{A}_\varphi$
Realisability of φ	$\sim$	Language Non-Emptiness of $\mathcal{A}_\varphi$

Church's Solvability Problem

– on the running example –

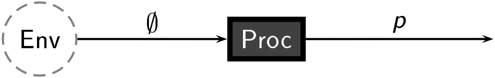


as before – $\mathcal{AA}$ won't do

how about $\mathcal{NA}$?

Church's Solvability Problem

– on the running example –

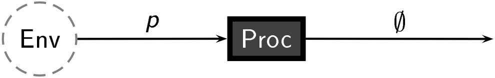


as before – $\mathcal{A}\mathcal{A}$ won't do

how about $\mathcal{N}\mathcal{A}$?

Church's Solvability Problem

– on the running example –

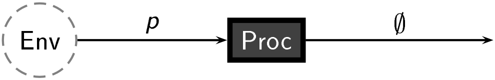


$\square \, \Diamond \, \bigcirc p \vee \bigcirc \neg p$ — $\mathcal{GBA}$

1. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{p}\}$
2. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc p}, \underline{\neg p}\}$
3. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{p}\}$
4. $\{\underline{\square}, \underline{\Diamond}, \underline{\bigcirc p \vee \bigcirc \neg p}, \underline{\bigcirc \neg p}, \underline{\neg p}\}$
5. $\{\underline{\square}, \underline{\Diamond}, \underline{p}\}$
6. $\{\underline{\square}, \underline{\Diamond}, \underline{\neg p}\}$

Church's Solvability Problem

– on the running example –



$\Box \Diamond \bigcirc p \vee \bigcirc \neg p$ $\mathcal{UCA}$

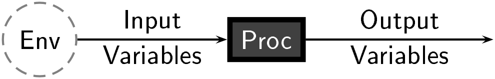
$\mathcal{NBA}$ for $\Diamond \Box \bigcirc \neg p \wedge \bigcirc p$

1. $\{\underline{\Diamond}\}$
2. $\{\underline{\Diamond}, \underline{\Box}, \underline{\bigcirc p}, \underline{\bigcirc \neg p}\}$

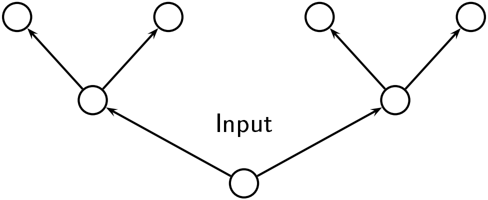
   (blocks as $\mathcal{NBA}$, accepts immediately as $\mathcal{UCA}$)

Church's Solvability Problem

– incomplete information –

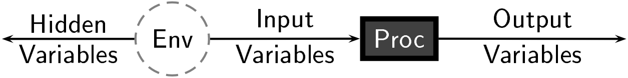


$\mathcal{AA} \Rightarrow \mathcal{UA}$ / $\mathcal{NA} \Rightarrow \mathcal{DA}$ **(expensive)**



$\models \varphi$

Extension: Incomplete Information



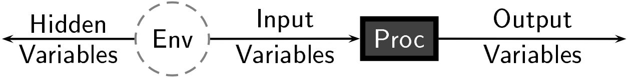
$\mathcal{UA}$ / $\mathcal{NA} \Rightarrow \mathcal{DA}$ (expensive)

"narrowing operation" $\mathcal{AA} \Rightarrow \mathcal{AA}$, $\mathcal{UA} \Rightarrow \mathcal{UA}$, $\mathcal{NA} \not\Rightarrow \mathcal{NA}$

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

$\rightsquigarrow$ send s_1 and s_2 to dir_{12}

Extension: Incomplete Information



$\mathcal{UA}$ / $\mathcal{NA} \Rightarrow \mathcal{DA}$ (expensive)

"narrowing operation" $\mathcal{AA} \Rightarrow \mathcal{AA}$, $\mathcal{UA} \Rightarrow \mathcal{UA}$, $\mathcal{NA} \not\Rightarrow \mathcal{NA}$

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

$\rightsquigarrow$ send s_1 and s_2 to dir_{12}

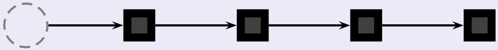
Part IV

Distributed Strategies

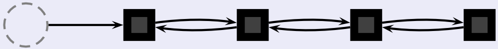
Distributed Synthesis

– classic results –

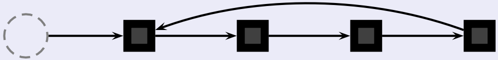
Decidable Architectures



Pipelines [Pnueli+Rosner 90]

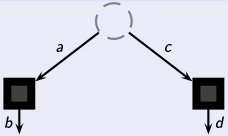


Two-Way Chains [Kupferman+Vardi 01]



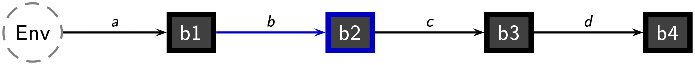
One-Way Rings [Kupferman+Vardi 01]

Undecidable Architecture



[Pnueli+Rosner 90]

What does a Process Know?



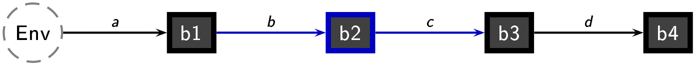
- process b2 knows its input
- process b2 knows its output

⇒ 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?



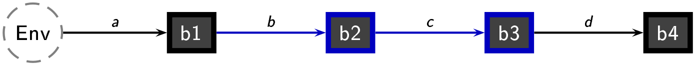
- process b2 knows its input
- process b2 knows its output

⇒ 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?



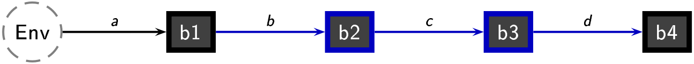
- process b2 knows its input
- process b2 knows its output

⇒ 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?



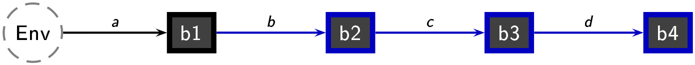
- process b2 knows its input
- process b2 knows its output

⇒ 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?



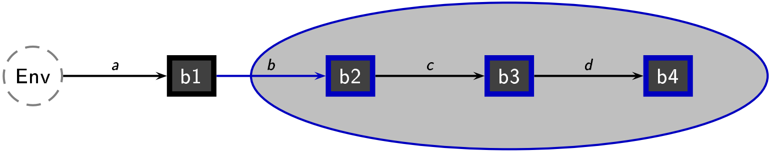
- process b2 knows its input
- process b2 knows its output

⇒ process b2 knows the input to process b3

⇒ process b2 knows the output of process b3

. . . least fixed point ⇒ knowledge of b2

Super-Processes



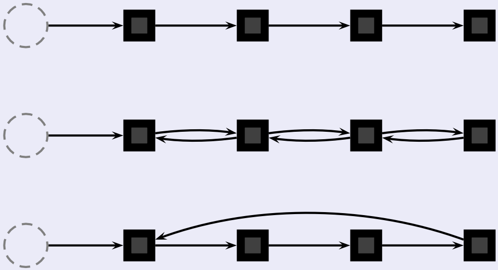
- b2 is better informed than b3 and b4 (b2 $\succeq$ b3, b4)
- b2 can simulate b3 and b4

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

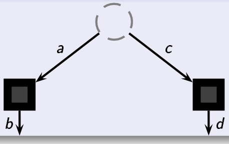
Decidability of Architectures

– particularities –

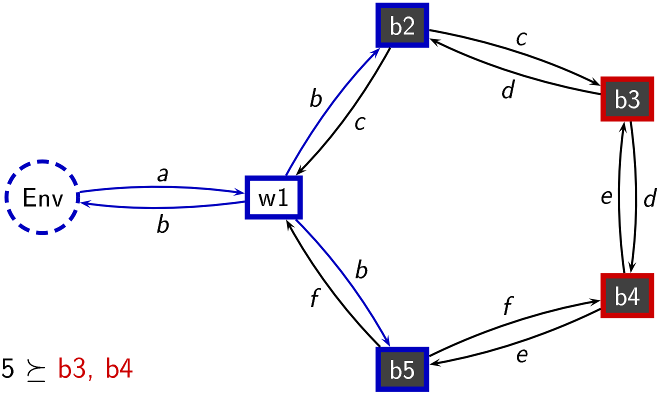
$\succeq$ is an order



processes incomparable by $\succeq$



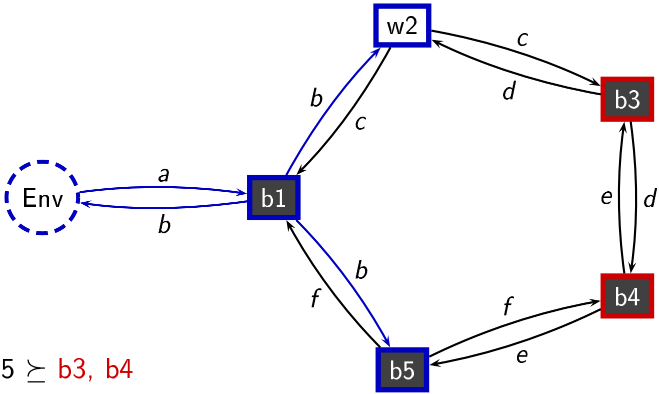
Information Fork



b2 $\simeq$ b5 $\succeq$ b3, b4

Undecidable

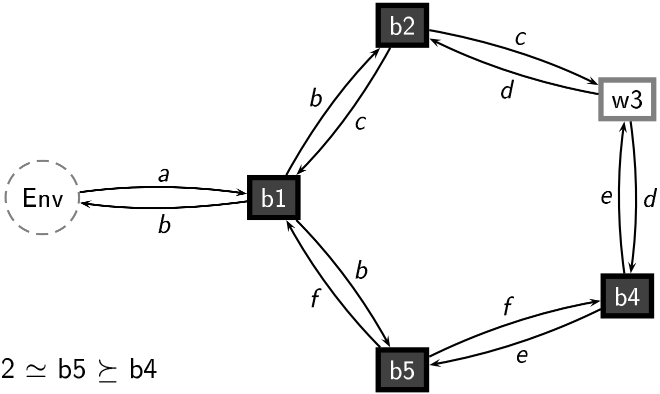
Information Fork



b1 $\succeq$ b5 $\succeq$ b3, b4

Undecidable

No Information Fork

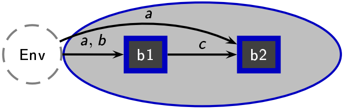


b1 $\succeq$ b2 $\simeq$ b5 $\succeq$ b4

Decidable

Decision Procedure

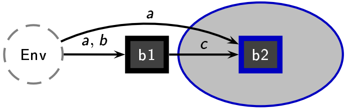
– ordered architecture –



- $\mathcal{A}_\varphi$ accepts strategies for super-process
- Automata transformation: $\mathcal{A}_\varphi$ – $\mathcal{B}_\varphi$
  $\mathcal{B}_\varphi$ 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

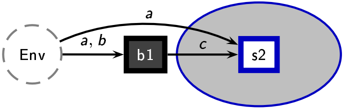
– ordered architecture –



- $\mathcal{A}_\varphi$ accepts strategies for super-process
- Automata transformation: $\mathcal{A}_\varphi$ — $\mathcal{B}_\varphi$
  $\mathcal{B}_\varphi$ 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

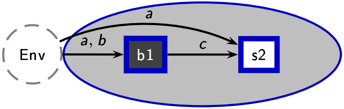
– ordered architecture –



- $\mathcal{A}_\varphi$ accepts strategies for super-process
- Automata transformation: $\mathcal{A}_\varphi$ – $\mathcal{B}_\varphi$
  $\mathcal{B}_\varphi$ 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

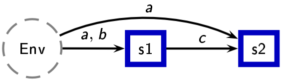
– ordered architecture –



- $\mathcal{A}_\varphi$ accepts strategies for super-process
- Automata transformation: $\mathcal{A}_\varphi$ – $\mathcal{B}_\varphi$
  $\mathcal{B}_\varphi$ 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

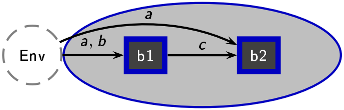
– ordered architecture –



- $\mathcal{A}_\varphi$ accepts strategies for super-process
- Automata transformation: $\mathcal{A}_\varphi$ – $\mathcal{B}_\varphi$
  $\mathcal{B}_\varphi$ 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

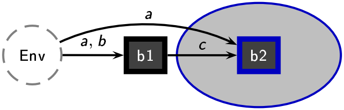
– ordered architecture –



- 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

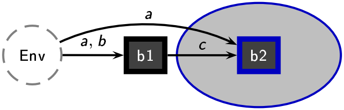
– ordered architecture –



- LTL, $\mathcal{UWA}$ / $\mathcal{UTA}$, $\mathcal{DWA}$ / $\mathcal{DTA}$
- 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

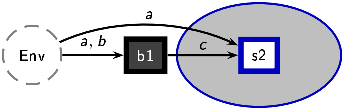
– ordered architecture –



- LTL, $\mathcal{UWA}$ / $\mathcal{UTA}$, $\mathcal{DWA}$ / $\mathcal{DTA}$
- 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

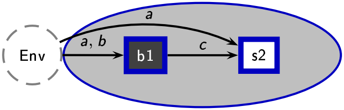
– ordered architecture –



- LTL, $\mathcal{UWA}$ / $\mathcal{UTA}$, $\mathcal{DWA}$ / $\mathcal{DTA}$
- 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

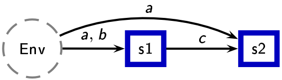
– ordered architecture –



- LTL, $\mathcal{UWA}$ / $\mathcal{UTA}$, $\mathcal{DWA}$ / $\mathcal{DTA}$
- 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

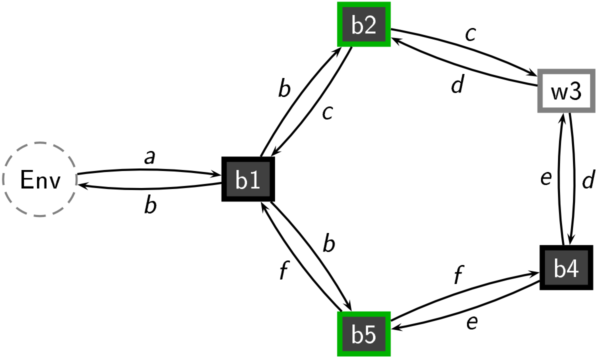
Decision Procedure

– ordered architecture –



- LTL, $\mathcal{UWA}$ / $\mathcal{UTA}$, $\mathcal{DWA}$ / $\mathcal{DTA}$
- 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

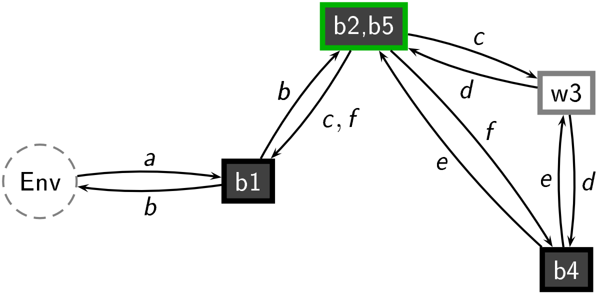
Decision Procedure



Architecture transformation ⪰ linear on black-box processes

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

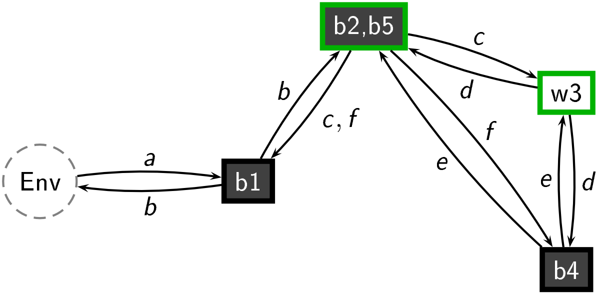
Decision Procedure



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

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

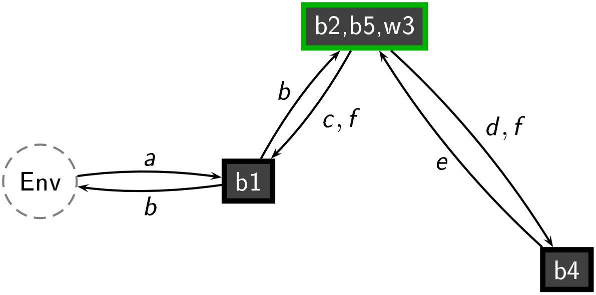
Decision Procedure



Architecture transformation ≽ linear on black-box processes

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

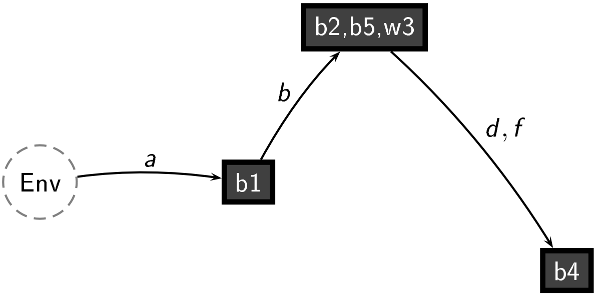
Decision Procedure



Architecture transformation ≽ 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 non-elementary, undecidable

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

non-elementary, undecidable

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

non-elementary, undecidable

theory: completeness result

maximal size of minimal model

practice: no small model ⇒ specification error

Redefine realisability

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

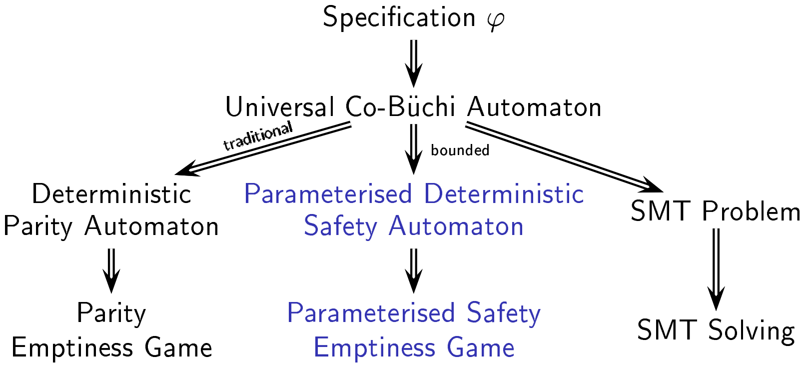
⇒ Bounded Synthesis

Part V

Two Steps Towards Practice

Overview

Complete Information



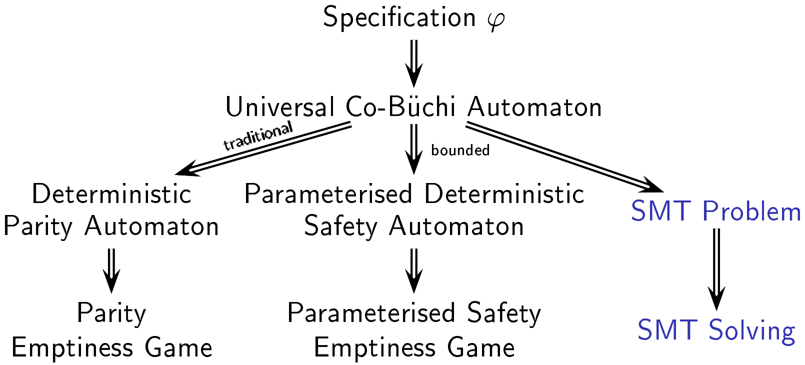
Distributed

Sequence of Automata Transformations
Safra-Constructions – Exponential

Locality Constraints
Small – Usually Cheap

Overview

Complete Information



Distributed

Sequence of Automata Transformations
Safra-Constructions – Exponential

Locality Constraints
Small – Usually Cheap

Overview

Complete Information

Specification φ

⇓

Universal Co-Büchi Automaton

traditional ⇙ | bounded ⇓ | ⇘

Deterministic Parity Automaton	Parameterised Deterministic Safety Automaton	SMT Problem
⇓	⇓	⇓
Parity Emptiness Game	Parameterised Safety Emptiness Game	SMT Solving

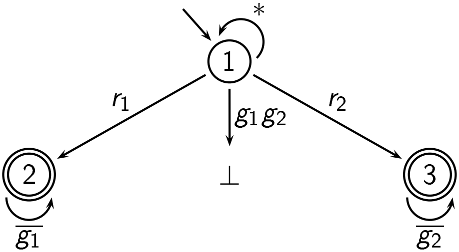
---

Distributed

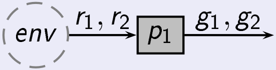
Sequence of Automata Transformations
Safra-Constructions – Exponential

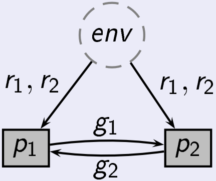
Locality Constraints
Small – Usually Cheap

Example – Simplified Arbiter

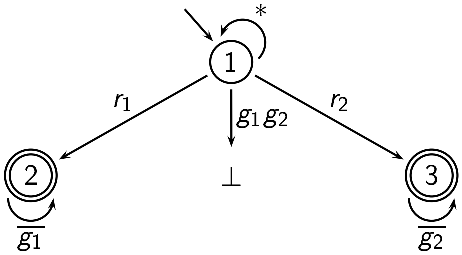


Synthesis with Complete Information





From Co-Büchi to Safety

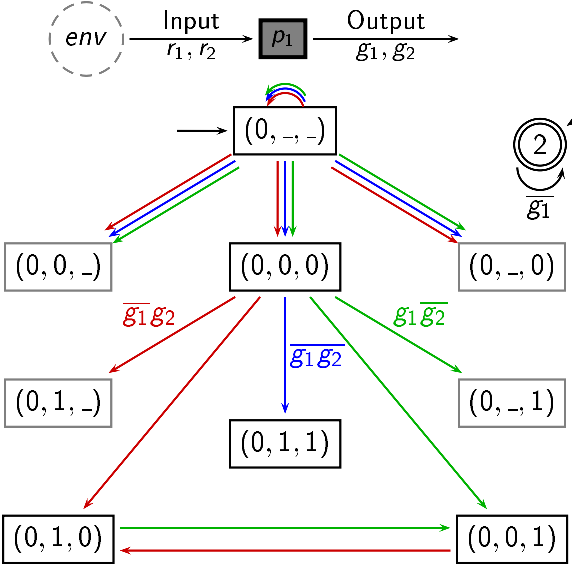


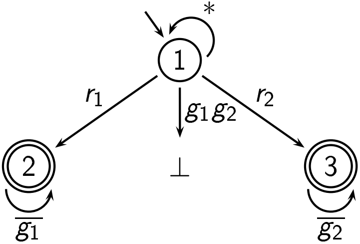
Realisable specification

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

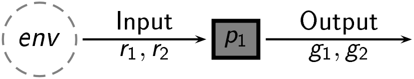
s can be bounded by the size of the resp. deterministic automaton

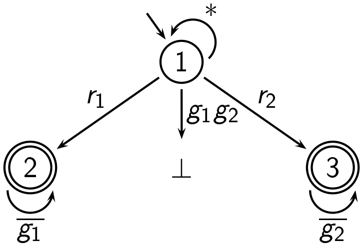
Parameterised Emptiness Game





Parameterised Emptiness Game





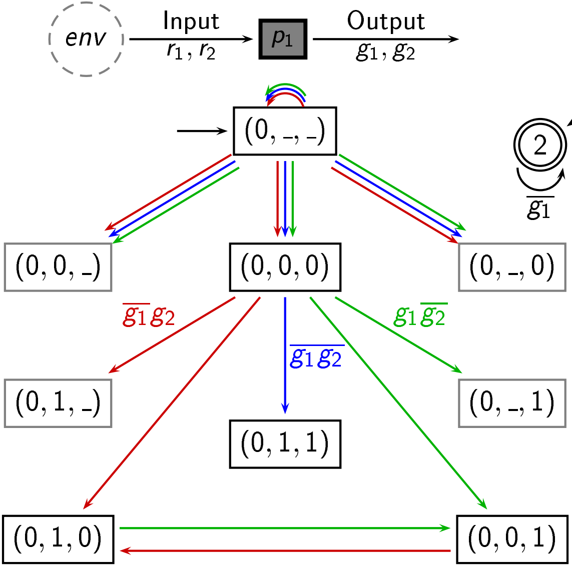
$(0, 1, \_)$

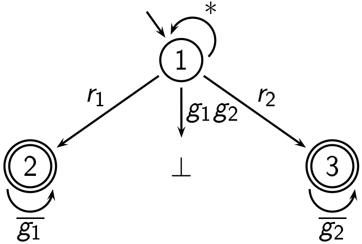
Semantics of a Game Positon

- collects the paths of the run tree
- i-th position in the annotation:

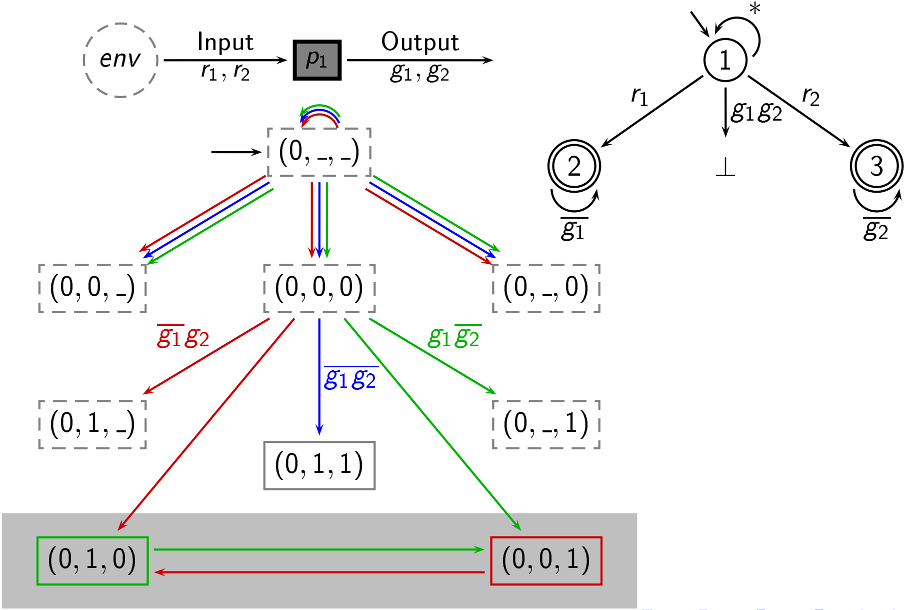
$\_$:	no path ends in automaton-state i
$n \in \mathbb{N}$:	a path may end in automaton-state i each such path has $\leq n$ previous visits to rejecting states

Parameterised Emptiness Game

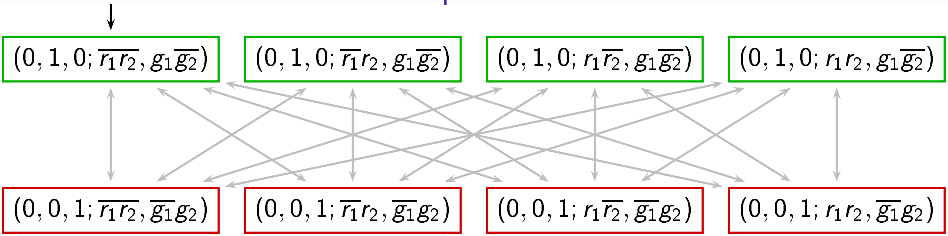




Parameterised Emptiness Game



Completeness



Theorem – Completeness

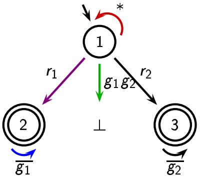
An (input preserving) transition system is accepted by a $\mathcal{UCB}$
$\Leftrightarrow$ it has a valid annotation.

Proof idea

Cycle with rejecting state reachable in the run graph
$\Leftrightarrow$ no valid annotation.

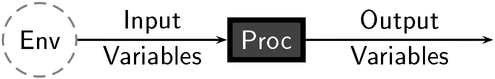
Progress Constraints – Automaton Transitions

output complexity: NP-complete



- $\forall t.\, \lambda_1^{\mathbb{B}}(t) \rightarrow \lambda_1^{\mathbb{B}}(\tau_{\bar{r}_1 \bar{r}_2}(t)) \wedge \lambda_1^{\#}(\tau_{\bar{r}_1 \bar{r}_2}(t)) \geq \lambda_1^{\#}(t)$
  $\wedge\ \lambda_1^{\mathbb{B}}(\tau_{\bar{r}_1 r_2}(t)) \wedge \lambda_1^{\#}(\tau_{\bar{r}_1 r_2}(t)) \geq \lambda_1^{\#}(t)$
  $\wedge\ \lambda_1^{\mathbb{B}}(\tau_{r_1 \bar{r}_2}(t)) \wedge \lambda_1^{\#}(\tau_{r_1 \bar{r}_2}(t)) \geq \lambda_1^{\#}(t)$
  $\wedge\ \lambda_1^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_1^{\#}(\tau_{r_1 r_2}(t)) \geq \lambda_1^{\#}(t)$
- $\forall t.\, \lambda_1^{\mathbb{B}}(t) \rightarrow \neg g_1(t) \vee \neg g_2(t)$
- $\forall t.\, \lambda_1^{\mathbb{B}}(t) \wedge r_1(t) \rightarrow \lambda_2^{\mathbb{B}}(\tau_{\bar{r}_1 \bar{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{\bar{r}_1 \bar{r}_2}(t)) \geq \lambda_1^{\#}(t)$
  $\wedge\ \lambda_2^{\mathbb{B}}(\tau_{\bar{r}_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{\bar{r}_1 r_2}(t)) \geq \lambda_1^{\#}(t)$
  $\wedge\ \lambda_2^{\mathbb{B}}(\tau_{r_1 \bar{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 \bar{r}_2}(t)) \geq \lambda_1^{\#}(t)$
  $\wedge\ \lambda_2^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 r_2}(t)) \geq \lambda_1^{\#}(t)$
- $\forall t.\, \lambda_2^{\mathbb{B}}(t) \wedge \neg g_1(t) \rightarrow \lambda_2^{\mathbb{B}}(\tau_{\bar{r}_1 \bar{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{\bar{r}_1 \bar{r}_2}(t)) > \lambda_2^{\#}(t)$
  $\wedge\ \lambda_2^{\mathbb{B}}(\tau_{\bar{r}_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{\bar{r}_1 r_2}(t)) > \lambda_2^{\#}(t)$
  $\wedge\ \lambda_2^{\mathbb{B}}(\tau_{r_1 \bar{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 \bar{r}_2}(t)) > \lambda_2^{\#}(t)$
  $\wedge\ \lambda_2^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 r_2}(t)) > \lambda_2^{\#}(t)$

Explicit Synthesis



Church's Solvability Problem – 1963

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

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

$$TS \models \varphi$$

CTL: EXPTIME-complete, exponential transition system

LTL: 2EXPTIME-complete, doubly exponential TS

Explicit vs. Succinct

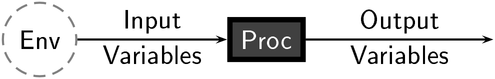
$$\text{counter:}\quad \bigwedge_{1<i\leq n} \forall\Box\,(p_i \not\leftrightarrow \forall\bigcirc\, p_i) \leftrightarrow \bigwedge_{j<i} p_j$$

$$\wedge\ (p_i \not\leftrightarrow \exists\bigcirc\, p_i) \leftrightarrow \bigwedge_{j<i} p_j$$

explicit	**succinct**
	circuit
transition system	program
Kripke structure	online Turing machine
2^n states	tape size n

Succinct Synthesis

min-output PSPACE-complete



Is there always a small oTM? CTL

if and only if $\mathsf{PSPACE} = \mathsf{EXPTIME}$

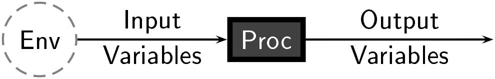
Is there always a small & fast circuit? CTL

if and only if $\mathsf{EXPTIME} \subseteq \mathsf{P/poly}$

LTL: dito for intermediate automata

Succinct Synthesis

min-output PSPACE-complete



Is there always a small oTM? — CTL

if and only if PSPACE = EXPTIME

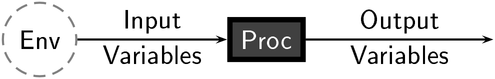
Is there always a small & fast circuit? — CTL

if and only if EXPTIME ⊆ P/poly

LTL: dito for intermediate automata

Succinct Synthesis

min-output PSPACE-complete



Is there always a small oTM? CTL

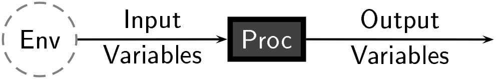
if and only if PSPACE = EXPTIME

Is there always a small & fast circuit? CTL

if and only if EXPTIME ⊆ P/poly

LTL: dito for intermediate automata

Succinct Synthesis



Is there always a small oTM? CTL

if and only if $\quad \mathrm{PSPACE} = \mathrm{EXPTIME}$

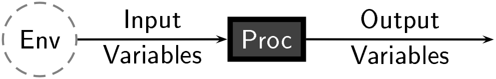
only if: guess & verify EXPTIME-complete

hence: PSPACE in the minimal succinct solution

if: much harder

uses the $\mathcal{DSA}$ from bounded synthesis

Succinct Synthesis



Is there always a small oTM? CTL

if and only if PSPACE = EXPTIME

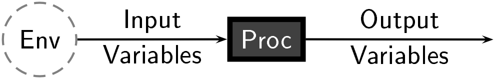
only if: guess & verify EXPTIME-complete

hence: PSPACE in the minimal succinct solution

if: much harder

uses the $\mathcal{DSA}$ from bounded synthesis

Succinct Synthesis



Is there always a small oTM? CTL

if and only if $\mathsf{PSPACE} = \mathsf{EXPTIME}$

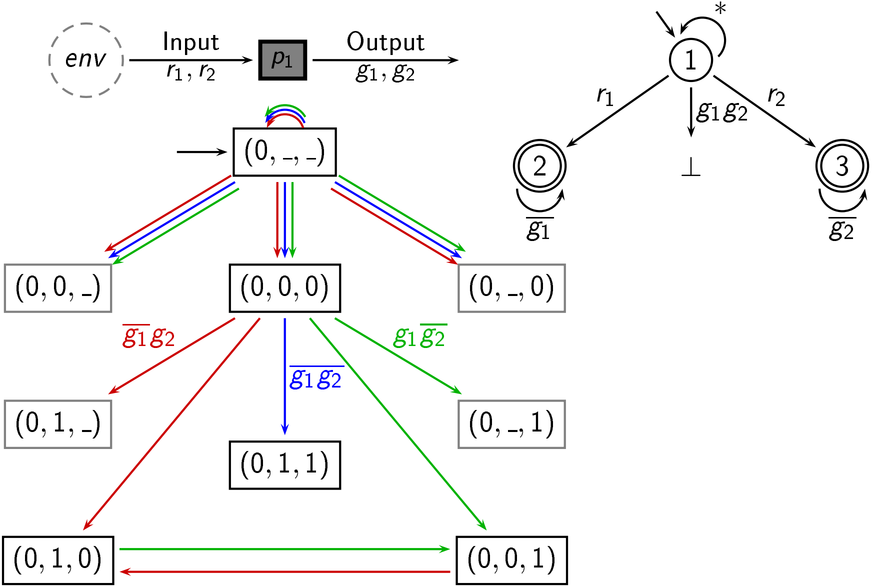
only if: guess & verify EXPTIME-complete

hence: PSPACE in the minimal succinct solution

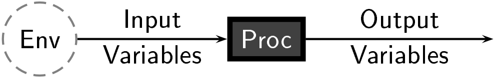
if: much harder

uses the $\mathcal{DSA}$ from bounded synthesis

PSPACE=EXPTIME ⇒ Small Model



Succinct & Fast Synthesis



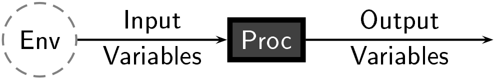
Is there always a small & fast circuit? CTL

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

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? CTL

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

if: as before

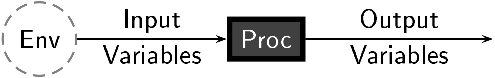
only if:
- construction not enough

  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? CTL

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 Progamming & Co

- Gal Katz, Doron Peled: MCGP: A Software Synthesis Tool Based on Model Checking and Genetic Programming. ATVA 2010: 359-364
- Gal Katz, Doron Peled: Code Mutation in Verification and Automatic Code Correction. TACAS 2010: 435-450
- Gal Katz, Doron Peled: Model Checking-Based Genetic Programming with an Application to Mutual Exclusion. TACAS 2008: 141-156
- Colin G. Johnson: Genetic Programming with Fitness Based on Model Checking. EuroGP 2007: 114-124

Sneak Preview

Search Technique		mutex		leader election	
		2 shared bits	3 shared bits	3 nodes	4 nodes
simulated annealing	execution time	**20**	**23**	**84**	**145**
	success rate	19	**23**	**19**	**17**
	overall time	**105.26**	**100**	**442.1**	**852.94**
hybrid	execution time	113	171	418	536
	success rate	**31**	17	15	11
	overall time	364.51	1,005.88	2,786.66	4,872.72
genetic programming	execution time	583	615	1120	1311
	success rate	7	7	3	3
	overall time	8,328.57	8,785.71	37,333.33	43,700.00

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

$\succeq$

con: beyond price

con: does not benefit from small solutions

Bounded Synthesis

Bounded Synthesis

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

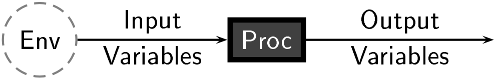
pro: complexity closer to model checking

pro: applicable to distributed systems

con: transition system vs. program / circuit

Succinct Synthesis

— is **good** news —



Is there always a small oTM? CTL

if and only if PSPACE = EXPTIME

Is there always a small & fast circuit? CTL

if and only if EXPTIME $\subseteq$ P/poly

Synthesis vs. Model Checking

Bounded Synthesis of Succinct Systems

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

pro: complexity **equal** to **model checking**

pro: applicable to distributed systems

Summary

- Distributed Synthesis
  - decidability
  - complexity
- Bounded Synthesis
  - decidability
  - complexity
- Succinct Synthesis
  - decidability
  - complexity