Verifying Partial Designs
— AVACS S1 —
German Verification Day

Sven Schewe
Joint Work with Bernd Finkbeiner

Universität des Saarlandes

31st August 2006
Partial Designs

Complete Design

Partial Design

Black-Box Components

- Components may be unknown
  - early design phase
  - external vendor
  - legacy products
- Complete design may be too expensive to be considered
  - state space explosion
Verification of Complete Designs

Outline

Temporary Specification

Motivation

Verification:

valid

invalid
Partial Designs

Motivation

Partial Designs

Validity

∀ Implementations

Realizability

∃ Implementation

Dual Verification Problems

ϕ realizable ⇔ ¬φ not valid

Valid: Specification satisfied for all implementations of the “black-box” processes

Realizable: Specification satisfied for some implementation of the “black-box” processes
Verification of Partial Designs

Temporal Specification

Synthesis

valid

unrealizable

completion

Outline

Motivation
Nodes:  
- white-box processes – **known** implementation
- black-box processes – **unknown** implementation
- environment – unconstrained behavior

Edges:  
- communication structure
- variables
Correct Implementations

Implementation correct ⇔ Computation tree satisfies specification
History of Synthesis

1962: Church’s problem
1969: Rabin; Büchi, Landweber – open systems, S1S

1981: Manna, Wolper – closed systems, LTL
      Clarke, Emerson – closed systems, CTL

1989: Pnueli, Rosner
      – synchronous and asynchronous open systems, LTL

1990: – distributed systems, LTL
      undecidable, decidable for pipelines

1995: Bernholtz, Vardi – open systems, CTL*

2001: Kupferman, Vardi
      – one-way rings / two-way chains, CTL*
## Completion of Partial Designs

**Open Questions – Pre-AVACS**

<table>
<thead>
<tr>
<th></th>
<th>Synchronous Systems</th>
<th>Asynchronous Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Open Systems</strong></td>
<td>CTL – EXPTIME</td>
<td>LTL – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td>LTL – 2EXPTIME</td>
<td>LTL – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td>CTL* – 2EXPTIME</td>
<td>Branching-Time ?</td>
</tr>
<tr>
<td></td>
<td>$\mu$-calculus – EXPTIME</td>
<td></td>
</tr>
<tr>
<td><strong>Distributed Systems</strong></td>
<td>General Undecidability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pipelines – non-elementary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Two-Way Chains – non-elementary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>One-Way Rings – non-elementary</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decidable Architectures ?</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uniform Approach ?</td>
<td>?</td>
</tr>
</tbody>
</table>
Synchronous Systems
Church’s Problem – Complete Information

Implementation

Computation Tree – $\mathcal{A}_\varphi$
Automata-Theoretic Synthesis

Specification $\varphi \sim$ Automaton $A_\varphi$

Models of $\varphi \sim$ Language of $A_\varphi$

Realizability of $\varphi \sim$ Language Non-Emptiness of $A_\varphi$

Implementation

Computation Tree – $A_\varphi$
Asynchronous Systems
One Process, Full Scheduler

Implementation – $B_{\varphi}$

Computation Tree – $A_{\varphi}$

injective $\Rightarrow$ surjective
Asynchronous Systems
One Process, Full Scheduler

Implementation – $B_\varphi$

Computation Tree – $A_\varphi$

Simulation $\Rightarrow$ store $O_{Env}$

Environment $\Rightarrow$ Process
Asynchronous Systems
One Process, Full Scheduler

Implementation – $B_\varphi$

Computation Tree – $A_\varphi$

$\text{Env} \xrightarrow{\text{Input Variables}} \text{Proc} \xrightarrow{\text{Output Variables}}$

Simulation $\Rightarrow$ input $\leftarrow$ store

Proc, $\neg$Env
Scheduler-Independent Synthesis
Scheduler-Independent Synthesis

Scheduler

- automaton $A^h_\varphi$ guesses a hostile scheduling policy

Asynchronous Composition + Scheduler

Implementation – $B_\varphi$

Computation Tree – $A^h_\varphi$

scheduler$_2$
## Single-Process Synthesis

### Summary

<table>
<thead>
<tr>
<th>Open Systems</th>
<th>Synchronous Systems</th>
<th>Asynchronous Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CTL – EXPTIME</td>
<td>LTL – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td>LTL – 2EXPTIME</td>
<td>Full Scheduler</td>
</tr>
<tr>
<td></td>
<td>CTL* – 2EXPTIME</td>
<td>CTL – EXPTIME</td>
</tr>
<tr>
<td></td>
<td>µ-calculus – EXPTIME</td>
<td>CTL* – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>µ-calculus – EXPTIME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distributed Systems</th>
<th>General Undecidability</th>
<th>Decidable Architectures ?</th>
<th>Uniform Approach ?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pipelines – non-elementary</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Two-Way Chains – non-elementary</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>One-Way Rings – non-elementary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pipelines – non-elementary
Two-Way Chains – non-elementary
One-Way Rings – non-elementary

**Decidable Architectures ?**
**Uniform Approach ?**
Distributed Synthesis

Synthesis

Decision Procedure

Approximation Technique

Compositional Method

Distributed

Approximation

Compositional

Single-Process Distributed

Synchronous Asynchronous

Single-Process Async.

Full-Scheduler Sched.-Indep.

√ √ √
Theorem
Asynchronous synthesis is decidable iff $< 2$ black-box processes

Decision Procedure for one black-box process
- states of the Moore machines are stored (like environment decisions)
- state of Moore machine updated iff process scheduled
Synchronous Synthesis
Previous Results

Decidable Architectures

- Pipelines [Pnueli+Rosner 90]
- Two-Way Chains [Kupferman+Vardi 01]
- One-Way Rings [Kupferman+Vardi 01]

Undecidable Architecture

- [Pnueli+Rosner 90]
Information Fork

An information fork consists of two black-box processes $b_2$, $b_3$ and a subgraph $A'$ rooted in the environment such that:

- each edge in $A'$ has at least one variable invisible to $b_2$ and $b_3$
- there is an edge from $A'$ to $b_2$ with at least one variable invisible to $b_3$
- there is an edge from $A'$ to $b_3$ with at least one variable invisible to $b_2$

Theorem

An architecture is **undecidable** iff it contains an **information fork**
Information Fork
Detecting Undecidability

Undecidable
Information Fork
Detecting Undecidability

Undecidable
Information Fork
Detecting Undecidability

Decidable
Uniform Synthesis Algorithm
Overview

Deciding Decidability – Completeness

Is there an information fork?
- information fork $\Rightarrow$ undecidable
- no information fork $\Rightarrow$ decidable

For decidable Architectures
- Order the black-box processes w.r.t. their informedness
- Transform the architecture $\approx$ Pipeline
- Solve the synthesis problem on the simplified architecture
Informedness

Preorder "\( \preceq \)" on the processes (is better informed than)

**Intuition**
A process \( p \) can predict the decisions of all less informed processes \( q \) \( (p \preceq q) \)

**Construction**
Set \( E_p \) of edges that carry information invisible to \( q \)
\[ p \preceq q \iff \text{there is no directed path from Env to } q \text{ in } E_p \]

If all processes are comparable by \( \preceq \),
\( \preceq \) defines equivalence classes and an order on them
Informedness Relation
Uniform Synthesis Algorithm

1. Architecture transformation
2. Along the order of informedness:
   - shape transformation (reducing information)
   - nondeterminization (exponential blow-up)
## Distributed Synthesis

**Summary**

<table>
<thead>
<tr>
<th></th>
<th>Synchronous Systems</th>
<th>Asynchronous Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Open Systems</strong></td>
<td>CTL – EXPTIME</td>
<td>LTL – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td>LTL – 2EXPTIME</td>
<td>Full Scheduler</td>
</tr>
<tr>
<td></td>
<td>CTL* – 2EXPTIME</td>
<td>CTL – EXPTIME</td>
</tr>
<tr>
<td></td>
<td>μ-calculus – EXPTIME</td>
<td>CTL* – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>μ-calculus – 2EXPTIME</td>
</tr>
<tr>
<td><strong>Distributed Systems</strong></td>
<td>Information Fork Criterion</td>
<td>Single-Process: Decidable</td>
</tr>
<tr>
<td></td>
<td>Uniform Synthesis Algorithm (non-elementary)</td>
<td>Multi-Process: Undecidable</td>
</tr>
</tbody>
</table>
Approximation Technique

Synthesis

Compositional Method

Approximation Technique

Single-Process

Distributed

Approximation

Compositional

Single-Process

Distributed

Approximation

Compositional

Single-Process

Distributed

Approximation

Compositional

Single-Process

Distributed

Approximation

Compositional
Verification of Partial Designs
Approximation Technique

Temporal Specification

Verification
valid
completion
don’t know
unrealizable
Approximation Technique
Undecidable Architectures

Diagram:
- Env
- w1
- b1
- b2
- b3
- w2

Nodes:
- Env
- w1
- b1
- b2
- b3
- w2

Edges:
- a
- b
- c
- d
- e
- f
- g
- h
- i
- j
- k
- l
Proving Realizability – Underapproximation
Hiding Information

\[ \text{Env} \rightarrow a, b \rightarrow \text{b2} \rightarrow d \rightarrow \text{b1} \]

\[ \text{Env} \rightarrow a, b \rightarrow \text{b3} \rightarrow h, i \rightarrow \text{w2} \]

\[ \text{Env} \rightarrow d, j, k \rightarrow \text{w1} \rightarrow a \rightarrow \text{b1} \]

\[ \text{Env} \rightarrow \text{b2} \rightarrow e \rightarrow \text{b1} \]

\[ \text{Env} \rightarrow \text{b3} \rightarrow f, g \rightarrow \text{w2} \]

\[ \text{Env} \rightarrow j, k \rightarrow \text{b1} \]

\[ \text{Env} \rightarrow \text{w2} \rightarrow l \]
Disproving Realizability – Overapproximation

Adding Information
Disproving Realizability – Coarse Overapproximation
No Exponential Blow-Up
Semi-Automatic Approach
The Compositional Realizability Rule

Input
- black-box processes \( B = \{b_1, \ldots, b_n\} \),
- system specification \( \psi \),
- auxiliary specifications \( \varphi_{b_1}, \ldots, \varphi_{b_n} \)

\[
\begin{align*}
(R0) \quad (A, \emptyset) & \models \bigwedge_{b \in B} \varphi_b \rightarrow \psi \\
(R1) \quad (A, \{b_1\}) & \models \varphi_{b_1} \\
& \vdots \\
(Rn) \quad (A, \{b_n\}) & \models \varphi_{b_n} \\
\hline
(A, B) & \models \psi
\end{align*}
\]
Resilient Realizability – \((A, \{b_2\}) \models \varphi\)

- \(b_2\) can guarantee \(\varphi\) against \(b_1\) and \(b_3\)
  - \(b_1\) and \(b_3\) hostile
  - use automaton to guess hostile decisions of \(b_1\) and \(b_3\)
Decision Procedures

Synthesis

Compositional Method

Approximation Technique

Decision Procedure

Distributed

Synchronous

Async.

Single-Process

Full-Scheduler

Sched.-Indep.

Synchronous

Asynchronous

√

√ √

√

√ √

√

√
## Conclusions

Completion of Partial Designs  
Pre-AVACS Questions Solved

<table>
<thead>
<tr>
<th>Open Systems</th>
<th>Synchronous Systems</th>
<th>Asynchronous Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CTL – EXPTIME</td>
<td>LTL – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td>LTL – 2EXPTIME</td>
<td>Full Scheduler</td>
</tr>
<tr>
<td></td>
<td>CTL* – 2EXPTIME</td>
<td>CTL – EXPTIME</td>
</tr>
<tr>
<td></td>
<td>( \mu )-calculus – EXPTIME</td>
<td>CTL* – 2EXPTIME</td>
</tr>
<tr>
<td></td>
<td>CTL* – 2EXPTIME</td>
<td>( \mu )-calculus – EXPTIME</td>
</tr>
<tr>
<td></td>
<td>( \mu )-calculus – EXPTIME</td>
<td>Scheduler-Independent</td>
</tr>
<tr>
<td>Distributed Systems</td>
<td>Information Fork Criterion</td>
<td>Single-Process: Decidable</td>
</tr>
<tr>
<td></td>
<td>Uniform Synthesis Algorithm (non-elementary)</td>
<td>Multi-Process: Undecidable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Decidability of Partial Designs is decidable
- Decidable architectures can be solved uniformly
- Undecidable architectures can be approximated by decidable architectures
- Compositional method works for all architectures

References:
- Bernd Finkbeiner and Sven Schewe, Uniform Distributed Synthesis, LICS 2005
- Bernd Finkbeiner and Sven Schewe, Semi-Automatic Distributed Synthesis, ATVA 2005
- Sven Schewe and Bernd Finkbeiner, Synthesis of Asynchronous Systems, LOPSTR 2006