

# Verifying Partial Designs

## — AVACS S1 —

### German Verification Day

Sven Schewe  
Joint Work with Bernd Finkbeiner

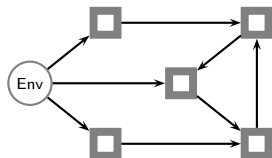
Universität des Saarlandes

31<sup>st</sup> August 2006



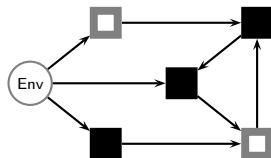
# Partial Designs

## Complete Design



VS.

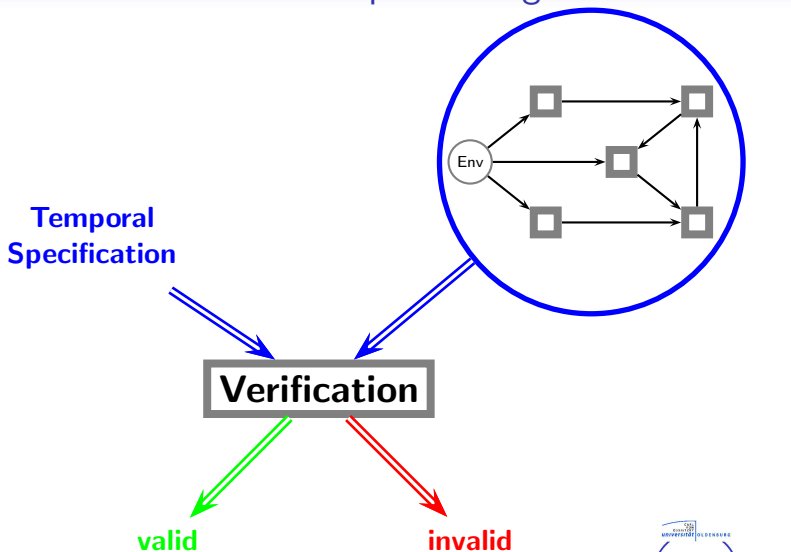
## Partial Design



## 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



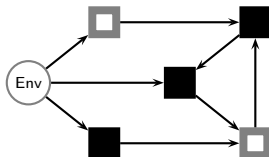
# Partial Designs

Validity



Implementations

Partial Design



Realizability



Implementation

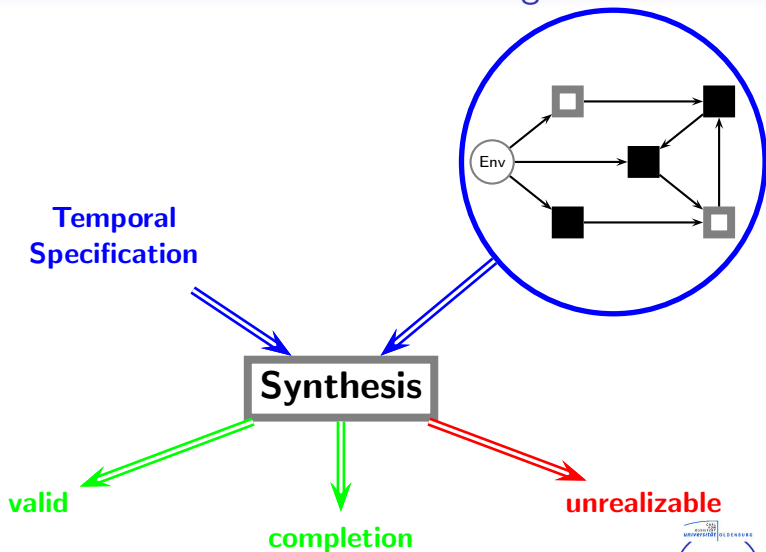
## Dual Verification Problems

$\varphi$  realizable  $\Leftrightarrow \neg\varphi$  not valid

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

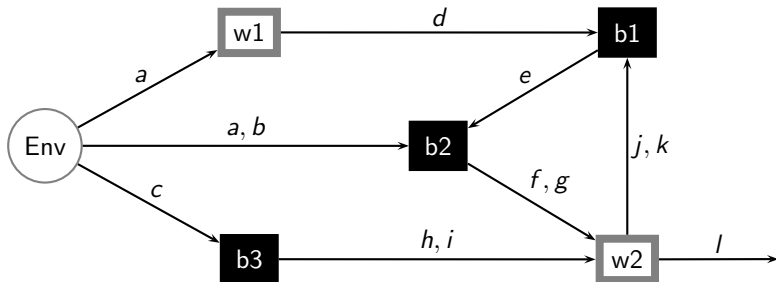
**Realizable:** Specification satisfied **for some** implementation of the “black-box” processes

# Verification of Partial Designs



# System-Architecture

## Directed Graph

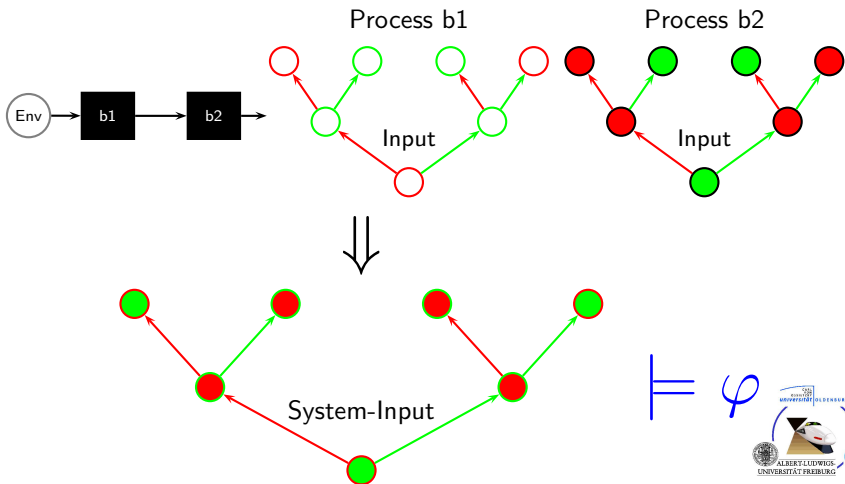


- Nodes:**
- white-box processes – **known** implementation
  - black-box processes – **unknown** implementation
  - environment – unconstrained behavior
- Edges:**
- communication structure
  - variables

# System Behavior

## Correct Implementations

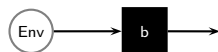
Implementation correct  $\Leftrightarrow$  Computation tree satisfies specification



# 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

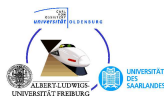
undecidable, decidable for pipelines



1995: Bernholtz, Vardi – open systems, CTL\*

2001: Kupferman, Vardi

– one-way rings / two-way chains, CTL\*



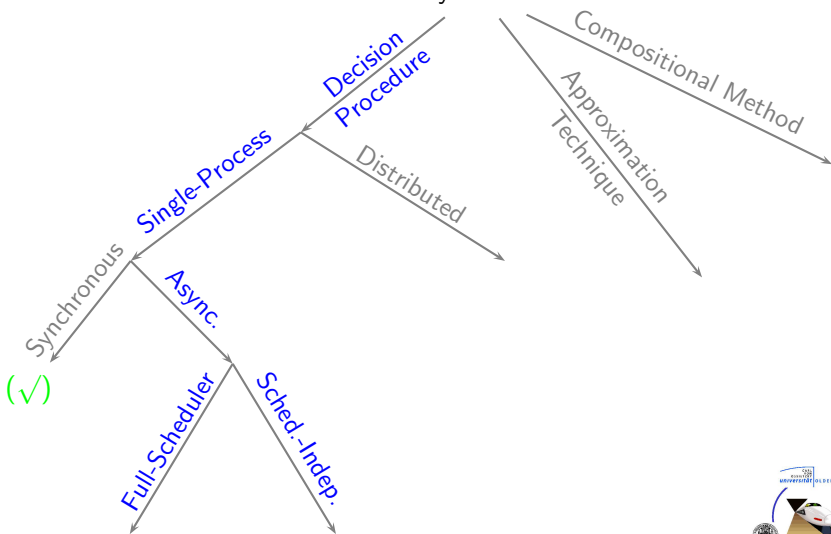
# Completion of Partial Designs

## Open Questions – Pre-AVACS

	Synchronous Systems	Asynchronous Systems
Open Systems	CTL – EXPTIME LTL – 2EXPTIME CTL* – 2EXPTIME $\mu$ -calculus – EXPTIME	LTL – 2EXPTIME <b>Branching-Time ?</b>
Distributed Systems	General Undecidability Pipelines – non-elementary Two-Way Chains – non-elementary One-Way Rings – non-elementary <b>Decidable Architectures ?</b> <b>Uniform Approach ?</b>	<span style="font-size: 2em; color: red;">?</span>

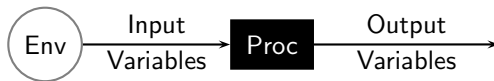
# Outline

Synthesis

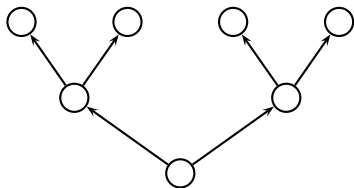


# Synchronous Systems

## Church's Problem – Complete Information

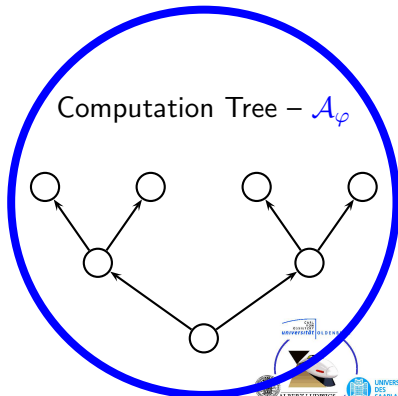


Implementation



$\approx$

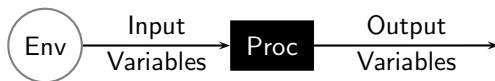
Computation Tree –  $\mathcal{A}_\varphi$



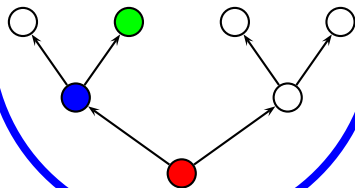


# Asynchronous Systems

## One Process, Full Scheduler

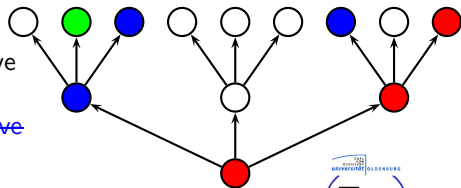


Implementation –  $\mathcal{B}_\varphi$



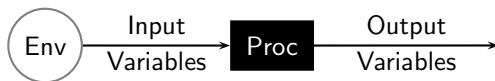
Computation Tree –  $\mathcal{A}_\varphi$

injective  
 $\Rightarrow$   
 surjective

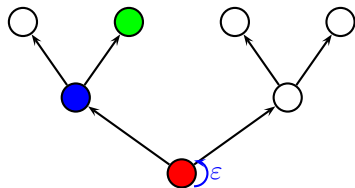


# Asynchronous Systems

## One Process, Full Scheduler

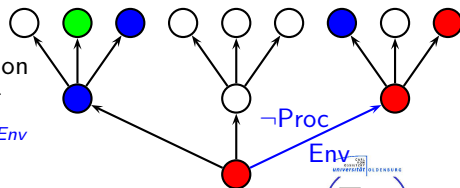


Implementation –  $\mathcal{B}_\varphi$



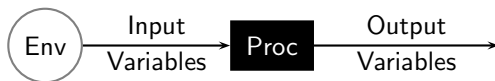
Computation Tree –  $\mathcal{A}_\varphi$

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

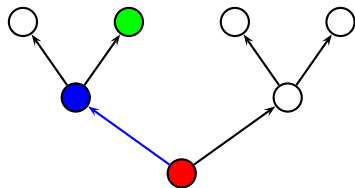


# Asynchronous Systems

## One Process, Full Scheduler

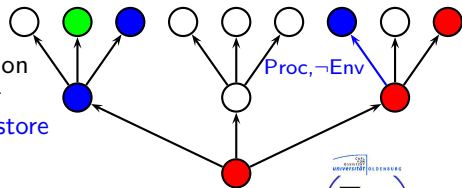


Implementation –  $\mathcal{B}_\varphi$



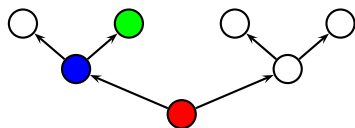
Computation Tree –  $\mathcal{A}_\varphi$

simulation  
 $\Rightarrow$   
 input  $\leftarrow$  store

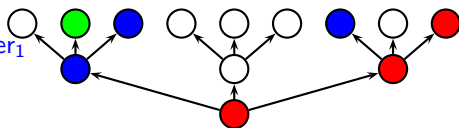


# Scheduler-Independent Synthesis

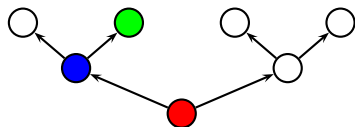
Implementation

scheduler<sub>1</sub>

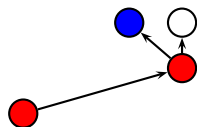
Computation Tree



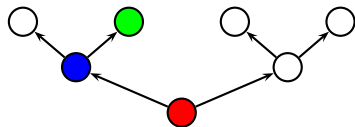
Implementation

scheduler<sub>2</sub>

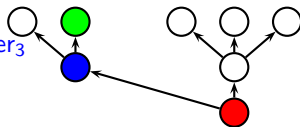
Computation Tree



Implementation

scheduler<sub>3</sub>

Computation Tree



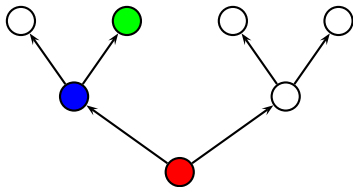
# Scheduler-Independent Synthesis

## Scheduler

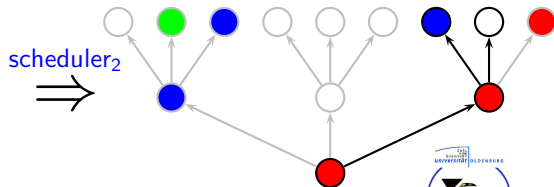
- automaton  $\mathcal{A}_\varphi^h$  guesses a hostile scheduling policy

## Asynchronous Composition + Scheduler

### Implementation – $\mathcal{B}_\varphi$




### Computation Tree – $\mathcal{A}_\varphi^h$

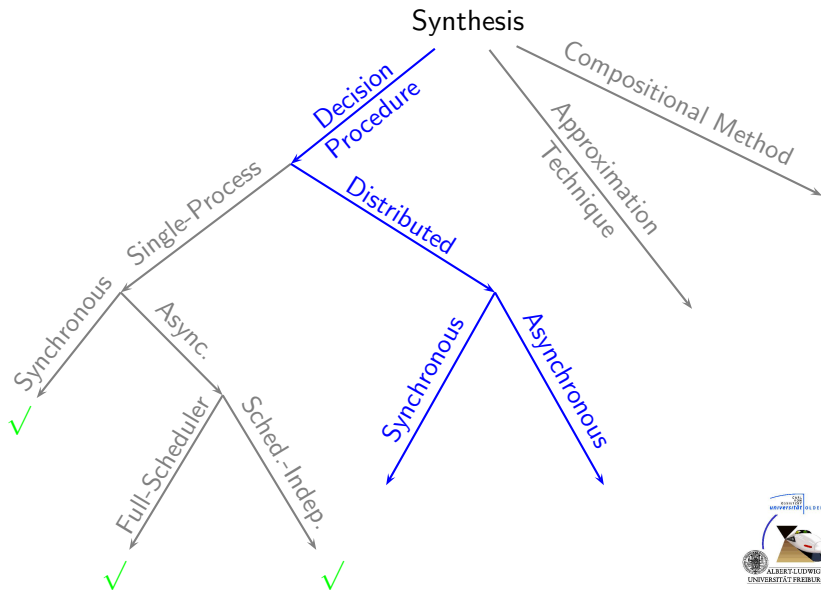


# Single-Process Synthesis

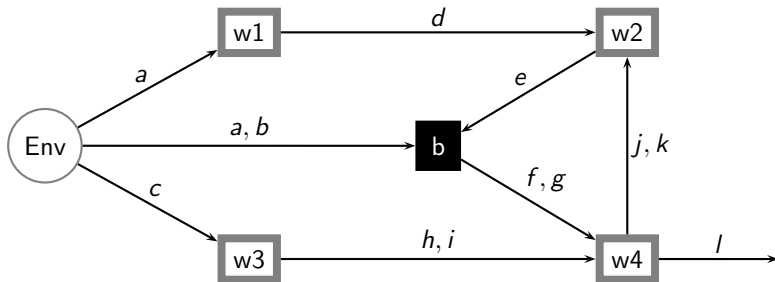
## Summary

	Synchronous Systems	Asynchronous Systems
Open Systems	CTL – EXPTIME LTL – 2EXPTIME CTL* – 2EXPTIME $\mu$ -calculus – EXPTIME	LTL – 2EXPTIME <hr/> Full Scheduler CTL – EXPTIME CTL* – 2EXPTIME $\mu$ -calculus – EXPTIME <hr/> Scheduler-Independent CTL – 2EXPTIME CTL* – 3EXPTIME $\mu$ -calculus – 2EXPTIME
Distributed Systems	General Undecidability Pipelines – non-elementary Two-Way Chains – non-elementary One-Way Rings – non-elementary Decidable Architectures ? Uniform Approach ?	

# Distributed Synthesis



# Asynchronous Synthesis



## 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

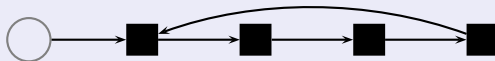
### Decidable Architectures



Pipelines [Pnueli+Rosner 90]

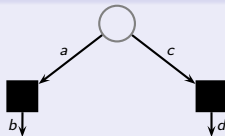


Two-Way Chains [Kupferman+Vardi 01]



One-Way Rings [Kupferman+Vardi 01]

### Undecidable Architecture



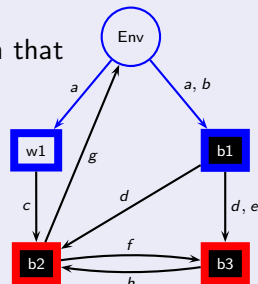
[Pnueli+Rosner 90]

# Information Fork

## Information Fork

An information fork consists of two black-box processes  $b2$ ,  $b3$  and a subgraph  $A'$  rooted in the environment such that

- each edge in  $A'$  has at least one variable invisible to  $b2$  and  $b3$
- there is an edge from  $A'$  to  $b2$  with at least one variable invisible to  $b3$
- there is an edge from  $A'$  to  $b3$  with at least one variable invisible to  $b2$

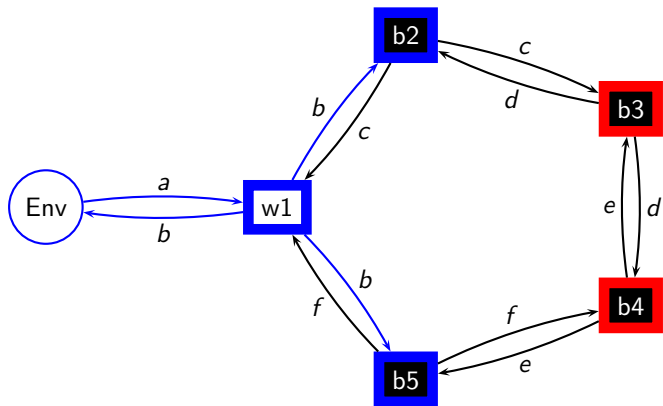


## Theorem

An architecture is **undecidable** iff it contains an **information fork**

# Information Fork

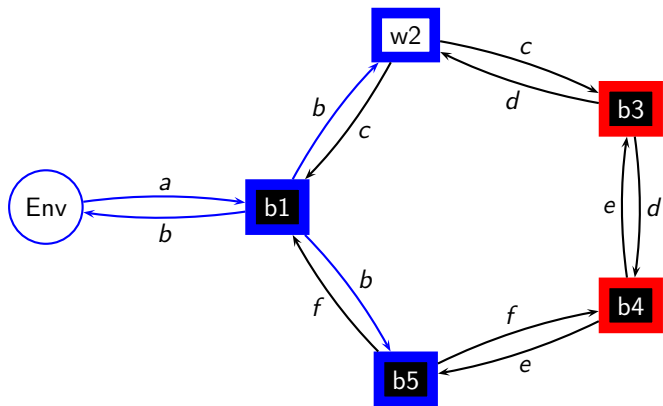
## Detecting Undecidability



# Undecidable

# Information Fork

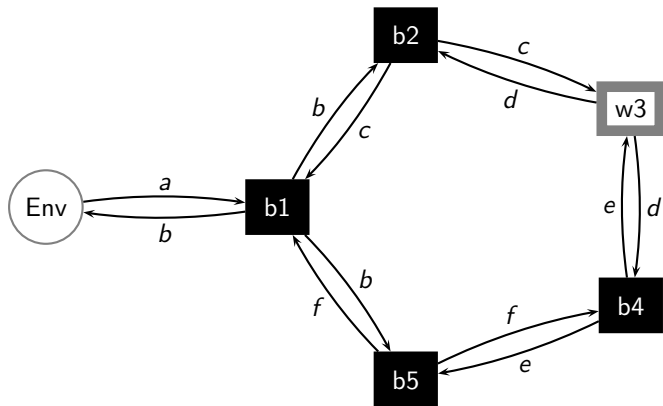
## Detecting Undecidability



# Undecidable

# Information Fork

## Detecting Undecidability



Decidable

# Uniform Synthesis Algorithm

## Overview

### Deciding Decidability – Completeness

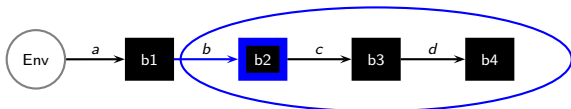
Is there an information fork?

- 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  $q$  ( $p \preceq q$ )

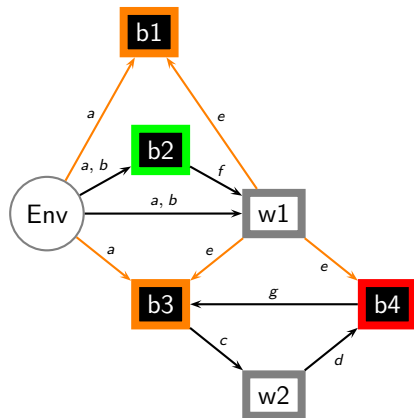
## Construction

Set  $E_p$  of edges that carry information invisible to  $p$   
 $p \preceq q \Leftrightarrow$  there is no directed path from  $Env$  to  $q$  in  $E_p$

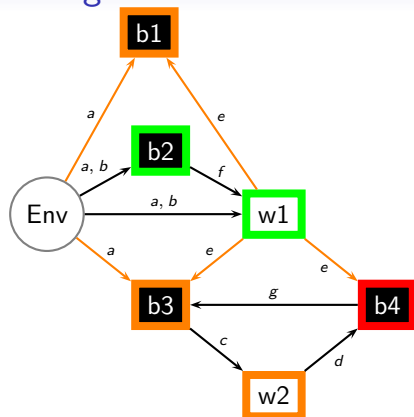
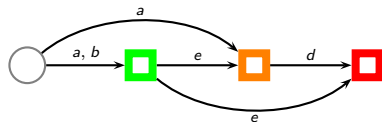
If all processes are comparable by  $\preceq$ ,

$\preceq$  defines **equivalence classes** and an **order** on them

# Informedness Relation



# Uniform Synthesis Algorithm



## Synthesis Algorithm

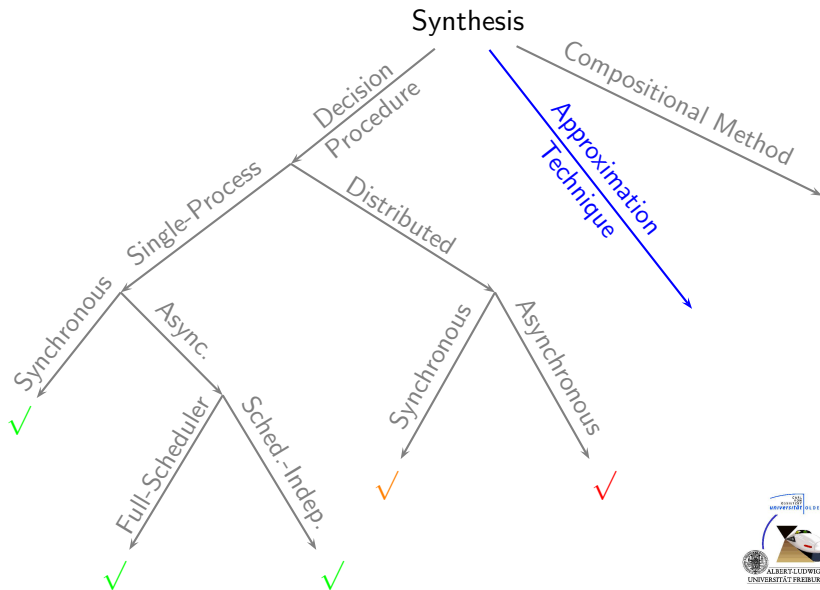
- 1 Architecture transformation
- 2 Along the order of informedness:
  - shape transformation (reducing information)
  - nondeterminization (exponential blow-up)

# Distributed Synthesis

## Summary

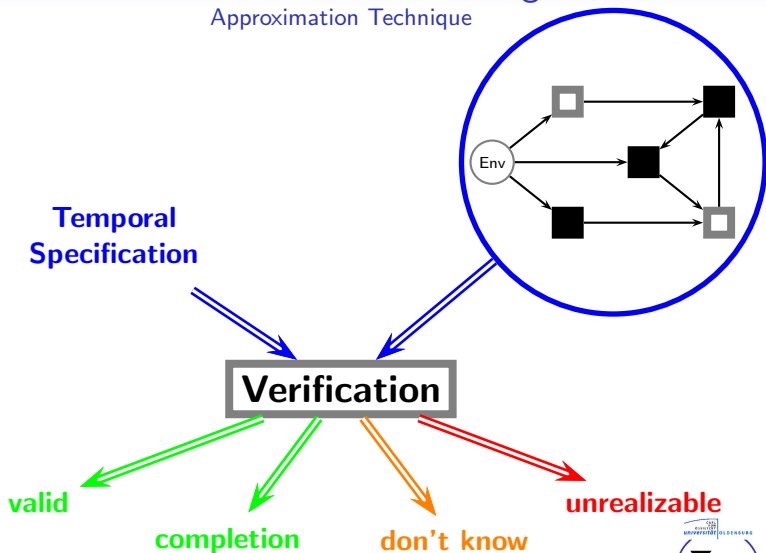
	Synchronous Systems	Asynchronous Systems
Open Systems	CTL – EXPTIME LTL – 2EXPTIME CTL* – 2EXPTIME $\mu$ -calculus – EXPTIME	LTL – 2EXPTIME <hr/> Full Scheduler CTL – EXPTIME CTL* – 2EXPTIME $\mu$ -calculus – EXPTIME <hr/> Scheduler-Independent CTL – 2EXPTIME CTL* – 3EXPTIME $\mu$ -calculus – 2EXPTIME
Distributed Systems	Information Fork Criterion Uniform Synthesis Algorithm (non-elementary)	Single-Process: Decidable Multi-Process: Undecidable

# Approximation Technique



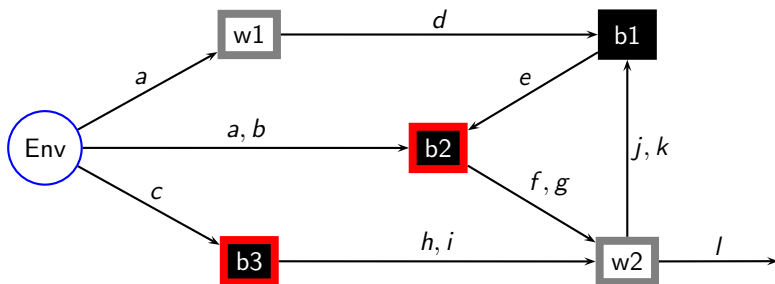
# Verification of Partial Designs

## Approximation Technique



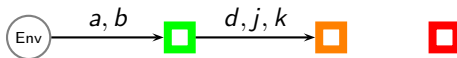
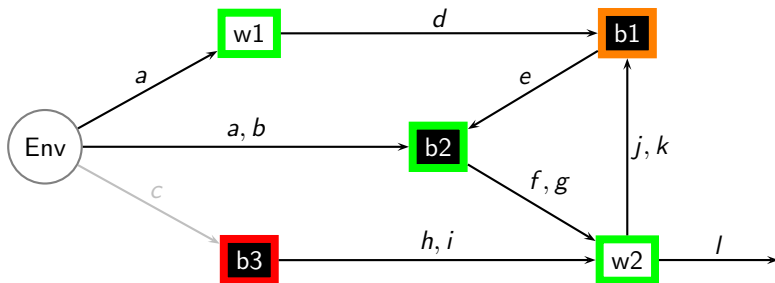
# Approximation Technique

## Undecidable Architectures



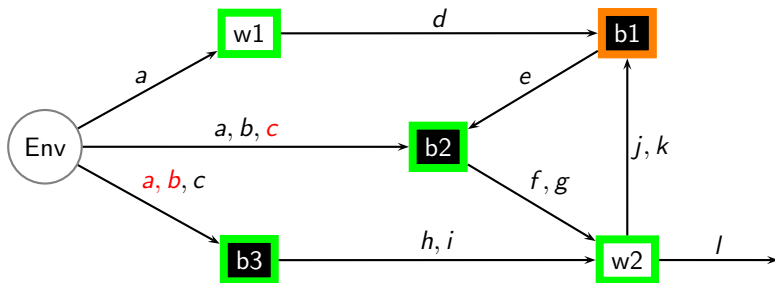
# Proving Realizability – Underapproximation

## Hiding Information



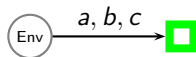
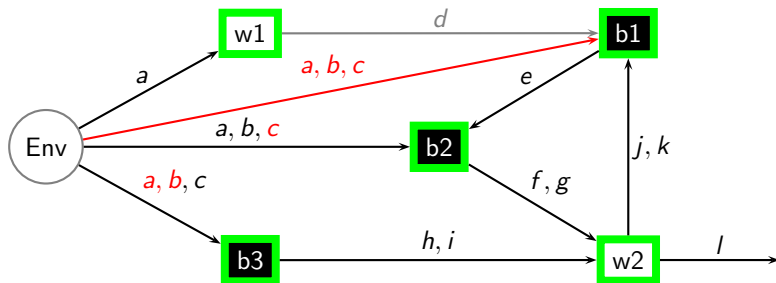
# Disproving Realizability – Overapproximation

## Adding Information

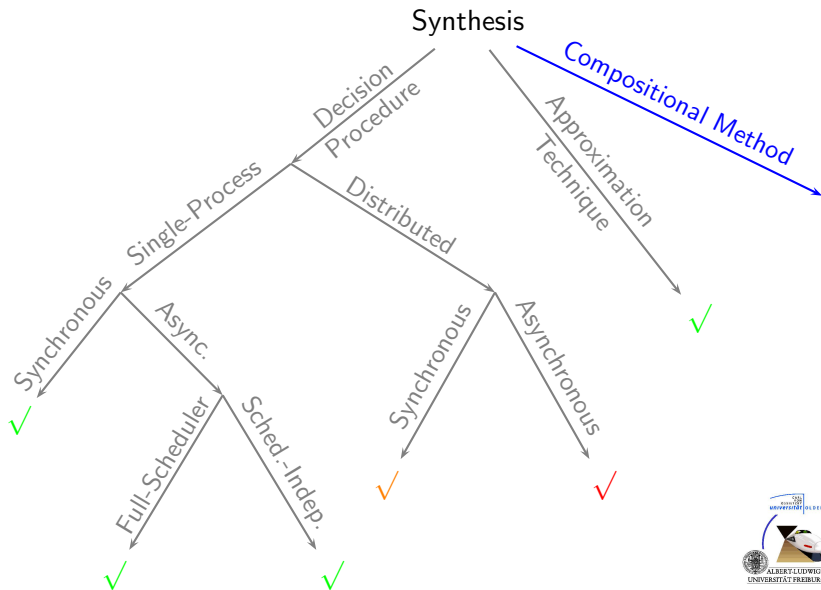


# Disproving Realizability – Coarse Overapproximation

## No Exponential Blow-Up



# Compositional Method



# Semi-Automatic Approach

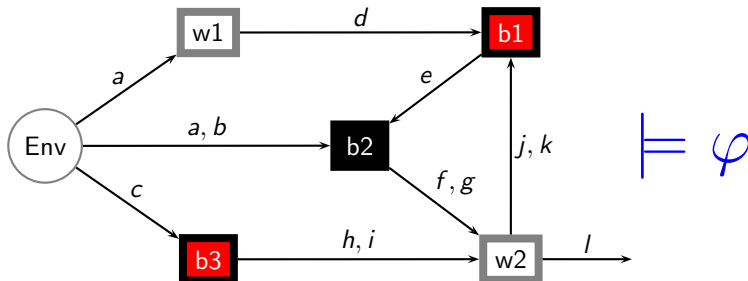
## The Compositional Realizability Rule

### Input

- black-box processes  $B = \{b_1, \dots, b_n\}$ ,
- system specification  $\psi$
- auxiliary specifications  $\varphi_{b_1}, \dots, \varphi_{b_n}$

$$\begin{array}{rcl}
 \text{(R0)} & (A, \emptyset) & \models \bigwedge_{b \in B} \varphi_b \rightarrow \psi \\
 \text{(R1)} & (A, \{b_1\}) & \models \varphi_{b_1} \\
 & \vdots & \vdots \\
 \text{(Rn)} & (A, \{b_n\}) & \models \varphi_{b_n} \\
 \hline
 & (A, B) & \models \psi
 \end{array}$$

# Compositional Method



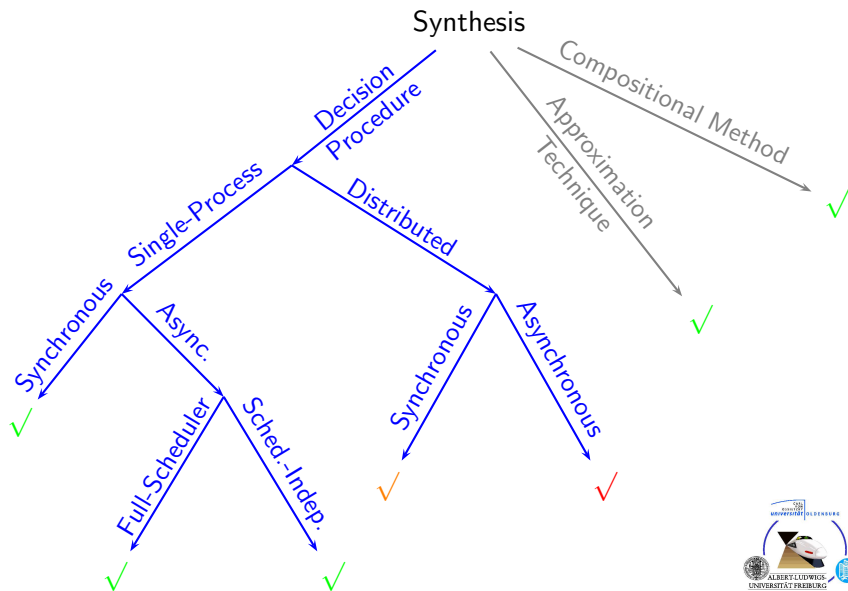
Resilient Realizability –  $(A, \{b2\}) \models \varphi$

$b2$  can guarantee  $\varphi$  against  $b1$  and  $b3$

- $b1$  and  $b3$  hostile
- use automaton to **guess** hostile decisions of  $b1$  and  $b3$

# Summary

# Decision Procedures

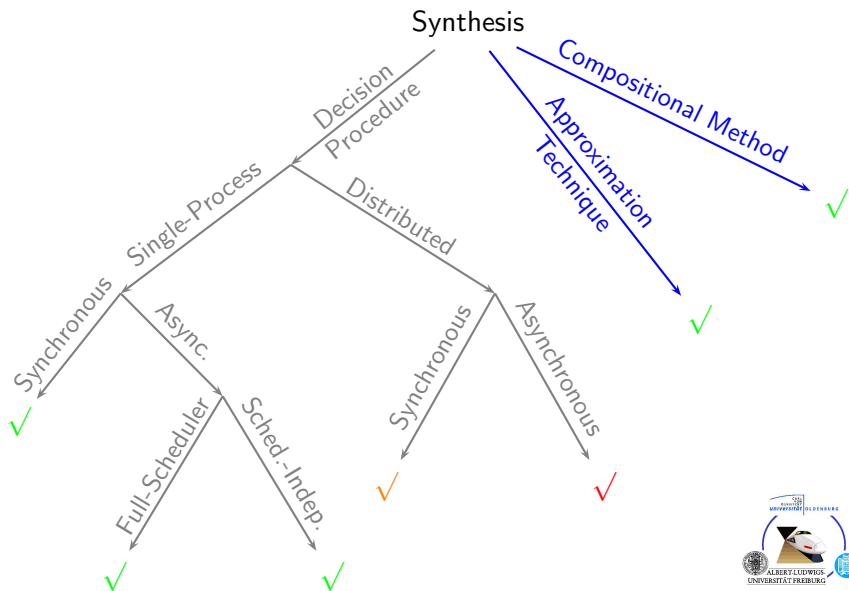


# Completion of Partial Designs

## Pre-AVACS Questions Solved

	Synchronous Systems	Asynchronous Systems
Open Systems	CTL – EXPTIME LTL – 2EXPTIME CTL* – 2EXPTIME $\mu$ -calculus – EXPTIME	LTL – 2EXPTIME <hr/> Full Scheduler CTL – EXPTIME CTL* – 2EXPTIME <hr/> $\mu$ -calculus – EXPTIME <hr/> Scheduler-Independent CTL – 2EXPTIME CTL* – 3EXPTIME $\mu$ -calculus – 2EXPTIME
Distributed Systems	Information Fork Criterion Uniform Synthesis Algorithm (non-elementary)	Single-Process: Decidable Multi-Process: Undecidable

# Approximation Technique & Compositional Method



## 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

