Component Interfaces for System Synthesis

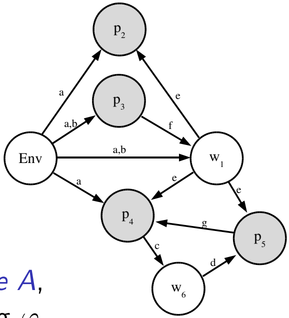
FIT 2008

Sven Schewe
Joint work with Bernd Finkbeiner

Universität des Saarlandes
Reactive Systems Group

5^{th} April 2008

The Problem



Given a *specification* φ and an *architecture* A, find a *distributed implementation* satisfying φ.

Specification: Regular set of trees; e.g., CTL, CTL* or μ-calculus

Architecture: Communication Structure

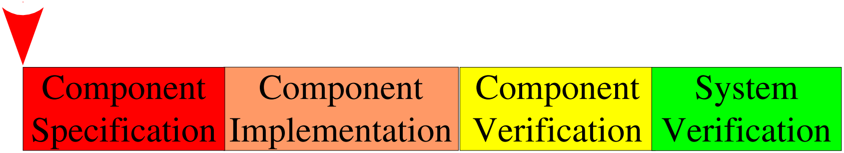
Implementation: Set of programs (Moore machines or trees), one for each component

Push-Button Approach – Automatic Synthesis

- Automatically transforms specifications into implementations for a given architecture
- Works well for single-process architectures
- *Undecidable* for most distributed architectures [PR90,FS05,SF07]

Advantage: Fully automatic
Unrealizable system specifications are detected early

Disadvant.: Works only for a small class of architectures
Extremely expensive (non-elementary lower-bound)



Manual Approach – Implement-and-Verify

1. Manually define component specifications
2. Manually write a *resilient implementation* for each component (independent of other implementations)
3. Automatically or manually *verify* the *correctness* of the distributed implementation

Advantage: Works for all architectures

Disadvant.:
- Mostly manual
- Identifies errors only after implementation
- Does not identify unrealizable requirements

Component Specification	Component Implementation	Component Verification	System Verification

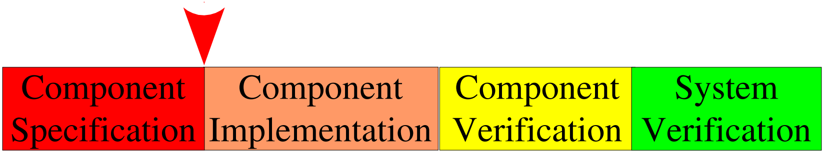
Semi-Automatic Approach – Compositional Synthesis

Trade-Off between both approaches

1. Manually define component specifications
2. Automatically synthesize resilient component implementations

Advantages:

- Mostly automatic
- Works for all architectures
- Reasonable complexity
- Detects unrealizable component specifications



Related Work

Distributed Synthesis

[PR90]: *Distributed Reactive Systems are Hard to Synthesize*
Pnueli and Rosner, FOCS 1990

[KV01]: *Synthesizing Distributed Systems*
Kupferman and Vardi, LICS 2001

[FS05]: *Uniform Distributed Synthesis*
Finkbeiner and Schewe, LICS 2005

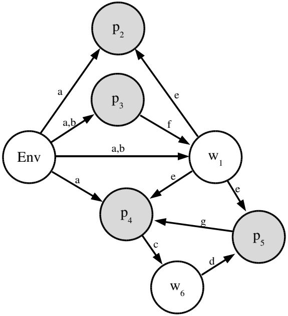
Synthesis in Reactive Environments

[KMTV00]: *Open Systems in Reactive Environments: Control and Synthesis*
Kupferman, Madhusudan, Thiagarajan and Vardi, CONCUR 2000

Overview

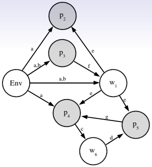
- Setting
    - Architectures
    - Implementations
    - Computations
    - Models
    - Compositional Synthesis Rule
    - Reactive Modules
- The Algorithm
- Conclusion

Architectures



- Architecture ≈ directed graph
  - Nodes ≈ processes
  - Edges ≈ communication structure
- Each process is either
  - a black-box process (sought implementation)
  - a white-box process (fixed implementation)
  - the environment *Env* (unrestricted behavior)
- Each process has a fixed set of input and output variables

Implementation

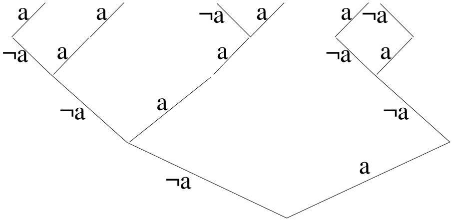


In each step, each process reads the values of its input variables and nondeterministically chooses the value of its output variables.

Implementation

- An implementation contains a strategy for each process.
- A strategy is a mapping from input histories to non-empty sets of possible outputs
  $s_b : (2^{I_b})^* \rightarrow \mathcal{O}_p$, for $\mathcal{O}_p = 2^{2^{O_p}} \setminus \{\emptyset\}$
- Regular strategy trees can be represented as finite-state Moore machines

Computations



Single Computation

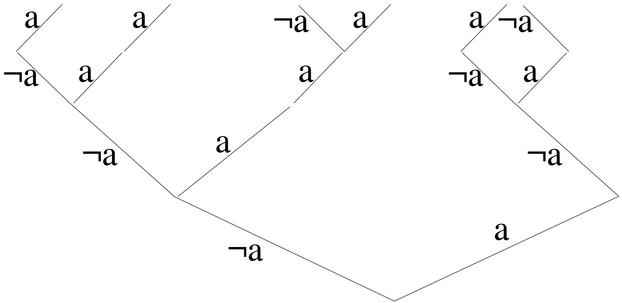
- Sequence of variable assignments $(\in (2^V)^*)$

Computation Tree

- An implementation defines a set of possible computations
- They can be identified with the paths of a total tree
- The set of successors in each node is the product of the individual process decisions $(\bigotimes_{p \in P} \mathcal{O}_p)$

System Models

A temporal or fixed point formula (CTL, CTL*, μ-calculus) φ describes a *regular set* of labeled *total trees*.



The total trees in this set are the system *models of* φ.

The Compositional Synthesis Rule

For a distributed *architecture A*
with set of *black-box processes* $B = \{b_1, \ldots, b_n\}$
and CTL* or μ-calculus *formulas* ψ; $\varphi_{b_1}, \ldots, \varphi_{b_n}$

$$
\begin{array}{lcl}
\text{(ST)} & (A, \emptyset) & \models \bigwedge_{b \in B} \varphi_b \rightarrow \psi \\
\text{(DCI 1)} & (A, \{b_1\}) & \models \varphi_{b_1} \\
\vdots & & \vdots \\
\text{(DCI } n\text{)} & (A, \{b_n\}) & \models \varphi_{b_n} \\
\hline
 & (A, B) & \models \psi
\end{array}
$$

where $(A, \mathcal{B}) \models \varphi$ means that the set $\mathcal{B} \subseteq B$ of black-box processes can guarantee φ against the remaining black-box processes $B \setminus \mathcal{B}$

Implementations as Models

$(A, B) \models \psi$ means that there is an implementation such that the computation tree is a model of ψ.

What is required for $(A, \{b\}) \models \varphi$?

Full-Tree models:

- there is a strategy tree for b that is a model of φ
- suitable for universal specifications

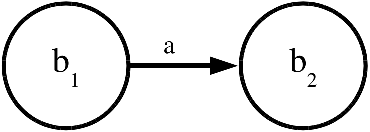
Reactive models:

- there is a strategy tree for b such that *every* total sub-tree is a model of φ [KMTV00]
- suitable for non-distributed systems

$\Rightarrow$ Resilient models

Full-Tree Models are too Weak for $(A, \{b\}) \models \varphi$

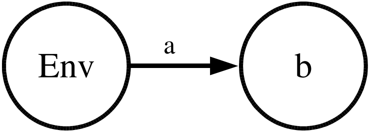
- $\psi = AGa \wedge EF\neg a$ ($= false$),
- $\varphi_1 = AGa$, and
- $\varphi_2 = EF\neg a$



- $s_{b_1} : x \mapsto \{a\} \quad \forall x \in \emptyset^*$ and
- $s_{b_2} : x \mapsto \emptyset \quad \forall x \in (2^{\{a\}})^*$

Reactive Models are too Strong for $(A, \{b\}) \models \varphi$

- $\psi = EFa$,
- $\varphi = \psi = EFa$



- $s_b : x \mapsto \emptyset \qquad \forall x \in (2^{\{a\}})^*$

Resilient Models

Combining Full-Tree Models and Reactive Models

Resilient Models

there is a strategy tree for b such that

- for every behavior of the remaining black-box processes
- the computation tree is a model of φ

Resilient models lead to a sound and complete synthesis rule

- Full-Tree models: Too weak $\longrightarrow$ unsound
- Reactive models: Too strong $\longrightarrow$ incomplete
- Resilient models: Sound and complete

Part II

The algorithm

Outline

1. From specifications to automata
2. Characteristic trees – capturing total trees with full trees
3. Quantification – finding computation trees of resilient models
4. Adjusting for white box processes – treating known components correctly
5. Narrowing – ignoring unavailable information
6. Emptiness check – constructing a strategy

Parity Tree Automata

Alternating Automata

- Run on full Σ-labeled Υ-trees (for finite sets Σ and Υ)
- May send *copies* to multiple states and in multiple directions $\Rightarrow$ run-tree
- Every path in the run tree must satisfy the parity condition

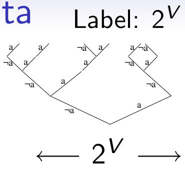
Nondeterministic Automata

- Only *one copy* is sent *in each direction*
- Can be used to simulate alternating automata
- Suited for language *projection* and *emptiness check*

Symmetric Alternating Automata or $\mathcal{ACG}$s

- Only abstract directions $\Box$ (for all successors) and $\Diamond$ (for some successor)
- Suited for *total* trees

From Specifications to Automata



Trees

- Each node in the computation tree is labeled with its direction
- Unlabeled 2^V-trees $\Rightarrow$ 2^V-labeled 2^V-trees
  – we (technically) do not insist on correct labels (for now)

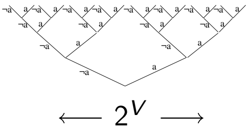
Automata

Specification φ $\Rightarrow$ *symmetric* alternating *automaton* $\mathcal{A}$
such that $\mathcal{A}$ accepts exactly the system models of φ

Characteristic Trees

Make Decisions Explicit

Label: $\bigotimes_{p \in P} \mathcal{O}_p \times 2^V$



Trees

- Each node is additionally labeled with the set of its successors
- 2^V-labeled 2^V-trees $\Rightarrow \bigotimes_{p \in P} \mathcal{O}_p \times 2^V$-labeled 2^V-trees
  - white-box strategies are ignored (for the moment)

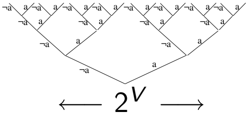
Automata

- Symmetric alternating automata $\Rightarrow$ alternating automata
- Successor set in label used to evaluate $\Box$ and $\Diamond$ transitions

Quantification

∀ Opponent Decisions

Label: $\bigotimes_{p \in W \cup \{Env, b\}} \mathcal{O}_p \times 2^V$



Trees

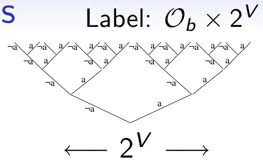
- $\bigotimes_{p \in P} \mathcal{O}_p \times 2^V$-labeled $\Rightarrow \bigotimes_{p \in W \cup \{Env, b\}} \mathcal{O}_p \times 2^V$-labeled 2^V-trees
- "Opponents" can choose the $\bigotimes_{p \in B \setminus \{b\}} \mathcal{O}_p$ part of the label

Automata

- Dualization (Language complementation),
- Nondeterminization,
- Projection (Choice of the $\bigotimes_{p \in B \setminus \{b\}} \mathcal{O}_p$ part of the label), and
- Dualization

White-Box Processes

Use Correct Implementation



Trees

- Trees with incorrect white-box strategies are eliminated
- The white-box decisions are deleted from the label
- $\bigotimes_{p\in W\cup\{Env,b\}} \mathcal{O}_p \times 2^V$-labeled $\Rightarrow \mathcal{O}_b \times 2^V$-labeled 2^V-trees

The *white-box processes* can be represented as a *Moore machine*

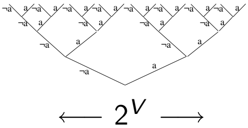
Automata

- Add the Moore machine to the automaton
- Use its output to substitute for the missing input

Direction

Use Correct Direction

Label: $\mathcal{O}_b$



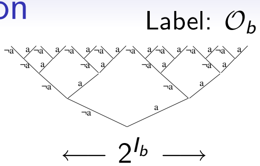
Trees

- Trees with labels that are inconsistent with the directions are eliminated
- The directions are deleted from the label
- $\mathcal{O}_b \times 2^V$-labeled 2^V-trees $\Rightarrow$ $\mathcal{O}_b$-labeled 2^V-trees

Automata

- Add the latest directions to the state of the automaton
- Use it to substitute for the missing input

Incomplete Information



Trees

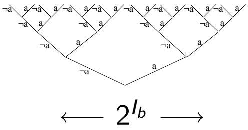
- A process may not react differently on indistinguishable paths
- Trees that violate this condition are eliminated
- Indistinguishable paths are merged into one path
- $\mathcal{O}_b$-labeled 2^V-trees $\Rightarrow$ $\mathcal{O}_b$-labeled 2^{I_b}-trees

Automata

- All copies that were sent in some direction $(d, d') \in 2^{I_p} \times 2^{V \setminus I_p}$ are sent in direction d
- Culmination of obligations

Realizability

Label: $\mathcal{O}_b$



Existence of a strategy is verified by a non-emptiness test

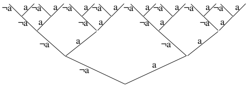
- *Nondeterminization*
- *Emptiness test* for the resulting nondeterministic automaton
- Constructive extension: Synthesis of a *Moore machine*

Complexity

- 2EXPTIME for ACTL* and μ-calculus
- 3EXPTIME for CTL*
- EXPTIME in the size of the Moore machine

Realizability

Label: $\mathcal{O}_b$



$\longleftarrow 2^{I_b} \longrightarrow$

System Tautology ST – $(A, \emptyset) \models \bigwedge_{b \in B} \varphi_b \longrightarrow \psi$

- Alternating word automaton with *single letter alphabet*
- Non-emptiness test directly on the alternating automaton

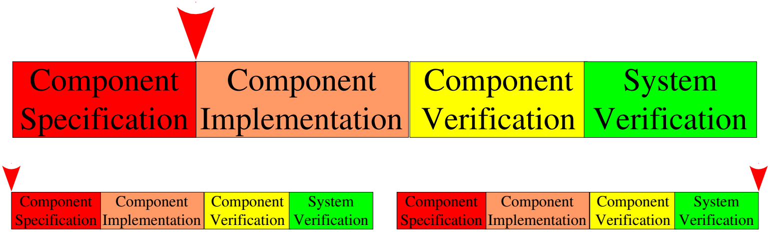
Complexity

- EXPTIME for ACTL* and μ-calculus
- 2EXPTIME for CTL*
- PTIME in the size of the Moore machine

Conclusions

Compositional synthesis

- Detects errors early
- Sound and complete for all distributed architectures
- Automatic (except for component specifications)
- Reasonable complexity (2EXPTIME vs. non-elementary)



Automatic Synthesis

Implement-and-Verify