Synthesizing Structured Reactive Programs via Deterministic Tree Automata

Benedikt Brütsch

1st International Workshop on Strategic Reasoning (SR 2013)
March 17, 2013, Rome
Outline

1. Introduction
2. Structured Reactive Programs
3. Synthesizing Structured Reactive Programs
4. Complexity Considerations
5. Conclusion
1 Introduction

2 Structured Reactive Programs

3 Synthesizing Structured Reactive Programs

4 Complexity Considerations

5 Conclusion
Reactive Systems

Input sequence: 1 0 1 1 ... \{ (0, 1) \times (0, 1) \}^\omega

Output sequence: 0 1 0 0 ... \{ (0, 1) \times (0, 1) \}^\omega
Synthesis Problem for Reactive Systems

Given: \( \omega \)-regular specification \( R \subseteq (\{0, 1\} \times \{0, 1\})^\omega \) (= admissible system behavior)

Task: Construct an operator \( F: \{0, 1\}^\omega \rightarrow \{0, 1\}^\omega \) that works “on-line” (ith output only depends on first \( i \) inputs) such that for all \( \alpha \in \{0, 1\}^\omega \):

\[
\begin{pmatrix}
\alpha \\
F(\alpha)
\end{pmatrix} \in R
\]

(Or detect that no such operator exists.)

Game-theoretic formulation:
Find winning strategy for System against Environment.
Representation of the Winning Strategy

- Usual format: transition systems (e.g., Mealy/Moore automata)
- Desirable: more succinct representations, e.g.:
  - logic circuits
  - programs
Madhusudan’s Proposal (2011)

- Format of strategies: structured reactive programs.
- Synthesize these programs directly (no transition systems).

Advantages:
- Programs can be exponentially more succinct than transition systems.
- Synthesis procedure allows to obtain shortest program for given specification.
1 Introduction

2 Structured Reactive Programs

3 Synthesizing Structured Reactive Programs

4 Complexity Considerations

5 Conclusion
Example (Program over Boolean variables $B = \{b_1, b_2, b_3\}$)

```plaintext
input b_1;
input b_2;
if b_1 then {output b_2} else { b_1 := b_2 };  
while true do {
    input b_2;
    b_3 := b_1 \land b_2;
    output b_3
}
```
Let $B$ be a finite set of Boolean variables.

**Programs over $B$:**

- input $b$
- output $b$
- $b := \langle expr \rangle$

\[
\langle prog \rangle ; \langle prog \rangle
\]

- if $\langle expr \rangle$ then $\langle prog \rangle$ else $\langle prog \rangle$
- while $\langle expr \rangle$ do $\langle prog \rangle$

**Expressions:** Boolean expressions over variables in $B$. 

\[
\langle prog \rangle
\]
Program Computations

Example (Computation of a program over $B = \{b_1, b_2\}$)

$\begin{bmatrix} b_1=0 \\ b_2=0 \end{bmatrix} \xrightarrow{(1,\varepsilon)} \begin{bmatrix} b_1=1 \\ b_2=0 \end{bmatrix} \xrightarrow{(\varepsilon,1)} \begin{bmatrix} b_1=1 \\ b_2=0 \end{bmatrix} \xrightarrow{(0,\varepsilon)} \begin{bmatrix} b_1=0 \\ b_2=0 \end{bmatrix} \xrightarrow{(\varepsilon,\varepsilon)} \begin{bmatrix} b_1=0 \\ b_2=1 \end{bmatrix} \xrightarrow{(1,\varepsilon)} \begin{bmatrix} b_1=0 \\ b_2=1 \end{bmatrix}$

Input sequence: 1 0 1 ...

Output sequence: 1 ...

- Note: Delay between input sequence and output sequence possible.
- Delay of computation = largest length difference between input and output sequence (in example: delay is 2).
- Delay of program = largest delay of all its computations.
Definition (Behavior of program $p$)

$\langle\langle p \rangle\rangle = \text{set of infinite I/O sequences produced by computations that start with initial variable valuation}$
A program is called reactive if
- it is non-terminating and
- all infinite computations yield infinite input and output sequences.
Part 3: Synthesizing Structured Reactive Programs

1 Introduction

2 Structured Reactive Programs

3 Synthesizing Structured Reactive Programs

4 Complexity Considerations

5 Conclusion
New Synthesis Problem

Synthesis Problem for Structured Reactive Programs

**Given:**
- Specification $R \subseteq (\{0, 1\} \times \{0, 1\})^\omega$, represented by nondeterministic Büchi automaton $\mathcal{A}_R^\neg$ recognizing the complement of $R$.
- Finite set of Boolean variables $B$.
- Delay bound $k \in \mathbb{N}$.

**Task:** Construct a structured reactive program $p$ over $B$ with delay $\leq k$ such that $\langle \langle p \rangle \rangle \subseteq R$.

(Or detect that no such program exists.)
Basis for Synthesis Procedure

Programs are finite trees!

Example

```
while true
  input b1
  b2 := b1 \land true
  output b2
```

Diagram: [Diagram of the example program with nodes labeled `while`, `true`, `input b1`, `b2 :=`, `output b2`, `\land`, `b1`, `true`]

Madhusudan’s Approach

Madhusudan (2011):

- Construct two-way alternating tree automaton with co-Büchi acceptance condition, accepting exactly the desired programs.
- Perform emptiness check to obtain such a program.

**Note:** Only programs with strict alternation between input and output were considered.
Now:
- Direct construction of a deterministic (bottom-up) tree automaton (DTA).
- Lift restriction of strict input/output alternation: Consider programs with delay $\leq k$. 
Basic Concept (1)

Run of the DTA on a program $p$:
- Starts at leaf nodes.
- Assigns to each subprogram of $p$ (= to each node) a “description” of its behavior.
- Description at root indicates whether $p$ satisfies $R$. Final states = descriptions of “correct” programs.
Basic Concept (1)

Run of the DTA on a program $\rho$:

- Starts at leaf nodes.
- Assigns to each subprogram of $\rho$ (= to each node) a “description” of its behavior.
- Description at root indicates whether $\rho$ satisfies $R$. Final states = descriptions of “correct” programs.
Basic Concept (1)

Run of the DTA on a program $\rho$:

- Starts at leaf nodes.
- Assigns to each subprogram of $\rho$ (= to each node) a "description" of its behavior.
- Description at root indicates whether $\rho$ satisfies $R$. Final states = descriptions of "correct" programs.
Basic Concept (2)

- How to capture behavior of programs in “descriptions” of bounded size?
- Basic idea: Indicate possible pairs of program computations and runs of $\mathcal{A}_R$ (called co-executions) and $\mathcal{A}_R$ using pre- and postconditions.
- Note: Program $p$ violates specification iff there exists computation of $p$ and corresponding run of $\mathcal{A}_R$ such that computation starts with initial variable valuation and run of $\mathcal{A}_R$ is accepting.
Co-Executions

Example

\[ p: \begin{bmatrix} b_1=0 \\ b_2=0 \end{bmatrix} \xrightarrow{(1, \epsilon)} \begin{bmatrix} b_1=1 \\ b_2=0 \end{bmatrix} \xrightarrow{(\epsilon, 1)} \begin{bmatrix} b_1=1 \\ b_2=0 \end{bmatrix} \xrightarrow{(0, \epsilon)} \begin{bmatrix} b_1=0 \\ b_2=0 \end{bmatrix} \xrightarrow{(1, \epsilon)} \begin{bmatrix} b_1=0 \\ b_2=1 \end{bmatrix} \]

\[ \mathcal{A}_R: \quad s_1 \xrightarrow{(1, 1)} s_2 \]

Input sequence: 1 0 1
Output sequence: 1
Pre-/Postconditions for Co-Executions

Co-Configuration

A co-configuration $\gamma$ is a tuple $(\sigma, s, u, v)$, where

- $\sigma$: current valuation of variables in $B$,
- $s$: current state of $A_R^-$,
- $u$: input symbols still to be consumed by $A_R^-$,
- $v$: output symbols still to be consumed by $A_R^-$.

Either $u = \varepsilon$ or $v = \varepsilon$ (or both).

$|u|, |v| \leq k$ (sufficient for programs with delay $\leq k$).
Co-Execution Signatures (1)

“Description” of finite computations of $p$:

<table>
<thead>
<tr>
<th>Finite Co-Execution Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cosig^{\text{fin}}(p, \mathcal{A}_R, k)$ is a set of tuples $(\gamma, f, \gamma')$.</td>
</tr>
</tbody>
</table>

$(\gamma, f, \gamma') \in cosig^{\text{fin}}(p, \mathcal{A}_R, k)$ iff there exists a finite co-execution (of $p$ and $\mathcal{A}_R$) from $\gamma$ to $\gamma'$ with delay $\leq k$ such that

$$f = \begin{cases} 
1 & \text{if } \mathcal{A}_R \text{ visits an accepting state} \\
0 & \text{otherwise}
\end{cases}$$
Co-Execution Signatures (2)

“Description” of infinite computations of $p$:

**Infinite Co-Execution Signature**

$\text{cosig}^\infty(p, \mathcal{A}_R, k)$ is a set of co-configurations.

$\gamma \in \text{cosig}^\infty(p, \mathcal{A}_R, k)$ iff there exists an infinite co-execution (of $p$ and $\mathcal{A}_R$) with delay $\leq k$ starting at $\gamma$ such that $\mathcal{A}_R$ infinitely often visits an accepting state.
States of DTA = pairs of finite and infinite co-execution signatures.

If $p$ has delay $\leq k$:

\[ \langle \langle p \rangle \rangle \subseteq R \iff (\sigma_0, s_0, \varepsilon, \varepsilon) \notin \text{cosig}^\infty(p, \mathcal{A}_R, k) \]

$\Rightarrow$ Final states of DTA: signatures that satisfy this condition.

Co-execution signatures of a program can be computed inductively from signatures of its subprograms (see next slide).

$\Rightarrow$ DTA can compute signatures bottom-up.
Inductive Construction of Signatures

Finite Co-Execution Signature for $p = 'p_1 ; p_2'$

$(\gamma, f, \gamma') \in \text{cosig}^{\text{fin}}(p, \mathcal{A}_R, k)$ iff there exist $\gamma''$, $f_1$, $f_2$ such that

1. $(\gamma, f_1, \gamma'') \in \text{cosig}^{\text{fin}}(p_1, \mathcal{A}_R, k)$, and
2. $(\gamma'', f_2, \gamma') \in \text{cosig}^{\text{fin}}(p_2, \mathcal{A}_R, k)$, and
3. $f = \max \{f_1, f_2\}$.

Infinite Co-Execution Signature for $p = 'p_1 ; p_2'$

$\gamma \in \text{cosig}^{\infty}(p, \mathcal{A}_R, k)$ iff

1. $\gamma \in \text{cosig}^{\infty}(p_1, \mathcal{A}_R, k)$, or
2. there exist $\gamma'$, $f$ such that
   - $(\gamma, f, \gamma') \in \text{cosig}^{\text{fin}}(p_1, \mathcal{A}_R, k)$, and
   - $\gamma' \in \text{cosig}^{\infty}(p_2, \mathcal{A}_R, k)$. 

Resolving Some Remaining Issues

- DTA might accept some programs with delay $> k$.
  $\Rightarrow$ Intersection with another DTA recognizing programs with delay $\leq k$.

- DTA accepts some non-reactive programs.
  $\Rightarrow$ Intersection with another DTA recognizing reactive programs.
Emptiness Check

- Last step of synthesis procedure: Emptiness check for DTA.
- Standard algorithm yields smallest tree = shortest program.
Part 4: Complexity Considerations

1. Introduction

2. Structured Reactive Programs

3. Synthesizing Structured Reactive Programs

4. **Complexity Considerations**

5. Conclusion
Complexity of the Synthesis Procedures

Size of the DTA obtained by DTA-based synthesis procedure:
- exponential in size of $\mathcal{A}_R^{-}$,
- doubly exponential in $|B|$, 
- doubly exponential in $k$. 
Optimality of the Tree Automaton

**Theorem**

- Let $B$ be a finite set of Boolean variables,
- let $k \geq 1$ be a delay bound,
- let $R$ be an $\omega$-regular specification that is realizable by a program over $B$ with delay $\leq k$.

Any NTAC $C$ that accepts a program $p$ over $B$ iff $p$ has delay $\leq k$ and $\langle \langle p \rangle \rangle \subseteq R$ has at least $2^{(2^{|B|-1})}$ states.
Question: How many program variables are necessary to satisfy a given specification?

- “Classical” LTL synthesis:
  Size of smallest Mealy automaton realizing an LTL specification $\varphi$ is at most doubly exponential in $|\varphi|$.

- $\Rightarrow O(2^{|\varphi|})$ Boolean variables suffice to build a structured program for $\varphi$.

- But what about a lower bound?
Part 5: Conclusion

1. Introduction
2. Structured Reactive Programs
3. Synthesizing Structured Reactive Programs
4. Complexity Considerations
5. Conclusion
Main results:

- DTA-based procedure for synthesis of structured reactive programs.
- Size of DTA: exponential in size of specification automaton, doubly exponential in number of variables and delay bound.
- Size of DTA is optimal with respect to number of variables.

Open question:

- Lower bound for number of variables?