Modular Verification of Synchronous Programs

Manuel Gesell and Klaus Schneider
{gesell,schneider}@cs.uni-kl.de
es.cs.uni-kl.de

Embedded Systems Group
University of Kaiserslautern

13th International Conference on Application of Concurrency to System Design (ACSD), Barcelona, Spain, 8-10th July 2013
# Table of Contents

1. Introduction
   - Motivation
   - Synchronous Model of Computation
   - Example

2. Behavioral Changes Induced by Calling a Module
   - Example 1: Partially Determined Behavior
   - Example 2: Behavior After Substitution
   - The Solution

3. Conclusion
Outline

1 Introduction
   - Motivation
   - Synchronous Model of Computation
   - Example

2 Behavioral Changes Induced by Calling a Module

3 Conclusion
Outline

1 Introduction
   • Motivation
   • Synchronous Model of Computation
   • Example

2 Behavioral Changes Induced by Calling a Module

3 Conclusion
Motivation

interactive proof rule for module calls in synchronous programs
Motivation

interactive proof rule for module calls in synchronous programs
Motivation

Interactive proof rule for module calls in synchronous programs
Outline

1. Introduction
   - Motivation
   - Synchronous Model of Computation
   - Example

2. Behavioral Changes Induced by Calling a Module

3. Conclusion
Synchronous Model of Computation

- execution is divided into a sequence of reactions steps
- behavior in a reaction step
  - all inputs are read
  - all outputs are produced (instantaneously)
  - new internal state is determined
  - each variable has a unique value
Synchronous Languages

- implement the synchronous model
- modeling of hardware and software
- formal semantics
- data-flow oriented languages
  - Lustre
  - Signal
- control-flow oriented languages (imperative)
  - Esterel
  - Quartz
    - developed in our working group
    - Averest toolset (http://www.averest.org)
Definition

A synchronous guarded action \((\gamma \Rightarrow \alpha)\) consists of a boolean guard \(\gamma\) and an atomic assignment \(\alpha\).

Synchronous Guarded Actions

- intermediate format used in Averest
- simplified syntax
- execution of all enabled guarded actions in parallel
- communication over shared variables
Outline

1 Introduction
   - Motivation
   - Synchronous Model of Computation
   - Example

2 Behavioral Changes Induced by Calling a Module

3 Conclusion
Running Example

module EdgeDet(event ?i,!u)

bool p;

loop {
    next(p) = i;
    if(i∧¬p) u=true;
    if(¬i∧p) u=false;
    w: pause;
}

Guarded Actions

- true ⇒ next(r)=true
- (¬r∧w) ⇒ next(w)=true
- (¬r∧w) ⇒ next(p)=i
- (¬r∧w) ∧ i ∧ ¬p ⇒ u=true
- (¬r∧w) ∧ ¬i ∧ p ⇒ u=false
module EdgeDet(event ?i,!u)

bool p;

loop {
  next(p) = i;
  if(i∧¬p) u=true;
  if(¬i∧p) u=false;
  w: pause;
}

Guarded Actions

- true ⇒ next(r)=true
- (¬r∨w) ⇒ next(w)=true
- (¬r∨w) ⇒ next(p)=i
- (¬r∨w) ∧ i ∧ ¬p ⇒ u=true
- (¬r∨w) ∧ ¬i ∧ p ⇒ u=false

Question:
What value has u when i==p and w=true holds?
module EdgeDet(event ?i,!u)

bool p;

loop {
    next(p) = i;
    if(i∧¬p) u=true;
    if(¬i∧p) u=false;
    w: pause;
}

Guarded Actions

- true ⇒ next(r)=true
- (¬r ∨ w) ⇒ next(w)=true
- (¬r ∨ w) ⇒ next(p)=i
- (¬r ∨ w) ∧ i ∧ ¬p ⇒ u=true
- (¬r ∨ w) ∧ ¬i ∧ p ⇒ u=false

Question:
What value has u when i==p and w=true holds?
⇒ (r ∧ ¬w) ∨ (i ↔ p) ⇒ u=false
Transitions diagram (closed system)
Outline

1 Introduction

2 Behavioral Changes Induced by Calling a Module
   - Example 1: Partially Determined Behavior
   - Example 2: Behavior After Substitution
   - The Solution

3 Conclusion
Outline

1. Introduction

2. Behavioral Changes Induced by Calling a Module
   - Example 1: Partially Determined Behavior
   - Example 2: Behavior After Substitution
   - The Solution

3. Conclusion
Example 1: Partially Determined Behavior

```plaintext
module Example (event ?i)

    event u;
    loop{
        if(i) u = true;
        else u = false;
        pause;
    }
    || EdgeDet(i, u);

Guarded Actions for Variable u

1. \((\neg r \lor w) \land i \land \neg p \Rightarrow u=\text{true}\)
2. \((\neg r \lor w) \land \neg i \land p \Rightarrow u=\text{false}\)
3. \((r \land \neg w) \lor (i \leftrightarrow p) \Rightarrow u=\text{false}\)
```
Example 1: Partially Determined Behavior

module Example (event ?i)

event u;
loop{
    if(i) u = true;
    else u = false;
    pause;
}

|| EdgeDet(i, u);

Guarded Actions for Variable u

- \[(\neg r \lor w) \land i \land \neg p \Rightarrow u = \text{true}\]
- \[(\neg r \lor w) \land \neg i \land p \Rightarrow u = \text{false}\]
- \[(r \land \neg w) \lor (i \leftrightarrow p) \Rightarrow u = \text{false}\]
- \[i \Rightarrow u = \text{true}\]
- \[\neg i \Rightarrow u = \text{false}\]
Example 1: Partially Determined Behavior

module Example (event ?i)

event u;
loop{
  if(i) u = true;
  else u = false;
  pause;
}
|| EdgeDet(i, u);

Guarded Actions for Variable u

- $(\neg r \lor w) \land i \land \neg p \Rightarrow u = \text{true}$
- $(\neg r \lor w) \land \neg i \land p \Rightarrow u = \text{false}$
- $(r \land \neg w) \lor (i \leftrightarrow p) \Rightarrow u = \text{false}$
- $i \Rightarrow u = \text{true}$
- $\neg i \Rightarrow u = \text{false}$
Transitions diagram (closed system)
Transitions diagram (open system)
Module Call Influences

- default reaction of interface variables
- set of guarded actions for interface variables
Theorem 1:

\[
\frac{T_{\text{op}} \models \varphi}{T_{\text{cl}} \models \varphi}
\]

- \( T_{\text{op}} \) open system
- \( T_{\text{cl}} \) closed system
- universal specification \( \varphi \)
Outline

1 Introduction

2 Behavioral Changes Induced by Calling a Module
   - Example 1: Partially Determined Behavior
   - Example 2: Behavior After Substitution
   - The Solution

3 Conclusion
Example 2: Behavior After Substitution

module Example (event ?i)

```plaintext
event x, y;
loop{
next (x) = i;
if (i ∧ x) y = true;
pause;
}
||
EdgeDet (i ⊕ x, y);
```

module EdgeDet (event ?i, !u)

```plaintext
bool p;
r: pause;
loop {
next (p) = i;
if (i ∧ ¬p) u = true;
if (¬i ∧ p) u = false;
w: pause;
}
```
Example 2: Behavior After Substitution

module Example (event ?i)

```plaintext
event x, y;
loop{
    next (x) = i;
    if (i ∧ x) y = true;
    pause;
}
||
EdgeDet (i ⊕ x, y);
```

module EdgeDet(event ?i,!u)

```plaintext
bool p;
r: pause;
loop {
    next(p) = i;
    if(i ∧ ¬ p) u=true;
    if(¬ i ∧ p) u=false;
    w: pause;
}
```

Substitution

\[ \varrho := \{ (i \mapsto i \oplus x), (u \mapsto y) \} \]
Example 2: Behavior After Substitution
Example 2: Behavior After Substitution
Module Call Influences

- default reaction of interface variables
- set of guarded actions for interface variables
- body of a module (by substitution)
Module Call Influences

- default reaction of interface variables
- set of guarded actions for interface variables
- body of a module (by substitution)
  - crucial changes in the transition diagram
    - not restricted to synchronous systems
    - more specific in the paper
Theorem 2 - Preservation of Universal Specifications:

- transition system $T_V$
- substitution $\rho$
- universal specification $\varphi$

$$T_V \models \varphi \quad \therefore \quad \rho(T_V) \models \rho(\varphi)$$
Outline

1 Introduction

2 Behavioral Changes Induced by Calling a Module
   - Example 1: Partially Determined Behavior
   - Example 2: Behavior After Substitution
   - The Solution

3 Conclusion
Introduction

Behavioral Changes Induced by Calling a Module

Module A

BlackBox \( \phi \)

Module B

\[ \vdash \varnothing \]

Conclusion
Module A

\[ \Rightarrow \varphi \]
Behavioral Changes Induced by Calling a Module

Module A

\[ \phi \]
Module A

BlackBox \( \varphi \)

Module B

\[ \vdash \varphi \]

\[ \vdash \rho(\varphi) \]
Outline

1. Introduction
2. Behavioral Changes Induced by Calling a Module
3. Conclusion
Summary

The Goal

interactive proof rule for module calls in synchronous programs

Solved Problems Induced by Calling a Module

specific: default reaction
general: substituted behavior
specific/paper only: preemption and delayed start
Questions?
Averest Design Flow

Quartz → Compilation → AIF Module → Linking → AIF System → Transformation

Verification
- AIFProver
- SMV

Simulation
- Trace
- C

SW Synthese
- Java
- SystemC

HW Synthese
- VHDL
- Verilog

http://www.averest.org