Lifting Verification Results for Preemption Statements

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

Embedded Systems Group
University of Kaiserslautern

11th International Conference on Software Engineering and Formal Methods, Madrid, Spain, 25-27th September 2013
Outline

1. Introduction
   - Motivation
   - Synchronous Model of Computation
   - Preemption Behavior

2. Lifting Verification Results for Preemption Statements
   - Abort Transformation
   - Suspend Transformation

3. Conclusion
Outline

1. Introduction
   - Motivation
   - Synchronous Model of Computation
   - Preemption Behavior

2. Lifting Verification Results for Preemption Statements

3. Conclusion
Outline

1. Introduction
   - Motivation
   - Synchronous Model of Computation
   - Preemption Behavior

2. Lifting Verification Results for Preemption Statements

3. Conclusion
Motivation

interactive proof rules for module calls in synchronous programs
Motivation

interactive proof rules for module calls in synchronous programs

Module A

\[ \vdash \phi \]

Module B

\[ \vdash ? \]
What is this talk about?

Preemption

Module A

\[ \models \varphi \]

\[ \models ? \]
What is this talk about?

Motivation
interactive proof rules for module calls in synchronous programs

Module A
BlackBox $\varphi$
Preemption $\Theta(\varphi)$
Influence of Preemption on Modules
Influence of Preemption on Modules
Influence of Preemption on Modules

The diagram shows a sequence of modules represented by circles with arrows pointing from one module to the next, indicating the flow of execution. The module labeled 'a' is highlighted, suggesting a point of interest or an area to focus on. The 'Fa' outside the diagram indicates a condition or a rule that affects the modules, possibly related to preemption.
What is this talk about?

**Approach**

- restriction to preemption specific behavior
- step wise application possible
- preemption-specific $\Theta$
- specification should preserved ’as much as possible’
- correct by construction
Outline

1. Introduction
   • Motivation
   • Synchronous Model of Computation
   • Preemption Behavior

2. Lifting Verification Results for Preemption Statements

3. Conclusion
Synchronous Model of Computation

- execution is divided into a sequence of reactions steps
- behavior in a reaction step
  - starts/ends in a **pause**
  - all inputs are read
  - all outputs are produced (instantaneously)
  - new internal state is determined
  - each variable has a **unique** value
1 Introduction
   • Motivation
   • Synchronous Model of Computation
   • Preemption Behavior

2 Lifting Verification Results for Preemption Statements

3 Conclusion
Example

**abort behavior**

- execution of nested program part is stopped
Example

```c
abort{
    ...
    l_0: pause;
    ...
    l_1: pause;
    ...
    l_2: pause;
    ...
} when (σ)

σ

} when (σ)
```
Example

\[
\text{weak abort}\{
\text{l}_0: \text{pause} ; \\
\text{l}_1: \text{pause} ; \\
\text{l}_2: \text{pause} ; \\
\text{when (σ)}
\}
\]

\[
\text{when (σ)}
\]

\[
\text{when (σ)}
\]
Example

**suspend** behavior

- execution of nested program part is (infinitely) postponed
Example

```c
suspend{
    .
    .
    .
    l0: pause;
    .
    .
    .
    l1: pause;
    .
    .
    .
    l2: pause;
    .
    .
    .
} when (σ)
```
Example

weak suspend{
    l₀: pause;
    l₁: pause;
    l₂: pause;
}

}when (σ)
Outline

1 Introduction

2 Lifting Verification Results for Preemption Statements
   • Abort Transformation
   • Suspend Transformation

3 Conclusion
Outline

1 Introduction

2 Lifting Verification Results for Preemption Statements
   - Abort Transformation
   - Suspend Transformation

3 Conclusion
Idea

- **abort** stops execution
- preemptive specification preserves ’as much as possible’
- preemptive specification $\Theta_{ab}^{st}(\varphi)$ for $\varphi$ is satisfied
  - $\varphi$ already satisfied
  - abortion in a state not violating $\varphi$
Naive Transformation #1 (too weak)

\[ \varphi | F \text{ abort} \]
Naive Transformation #1 (too weak)
Naive Transformation #1 (too weak)

\[ \varphi \mid F \text{ abort} \]
Naive Transformation #2 (too strong)
Naive Transformation #2 (too strong)
Naive Transformation #2 (too strong)
Correct Transformation for Abort

**Transformation** $\Theta_{ab}^{st}(\varphi, \sigma)$

The transformation $\Theta_{ab}^{st}(\varphi, \sigma)$ that generates an **abort**-sensitive specification for $A\varphi$ given in NNF is defined recursively as

$$\Theta_{ab}^{st}(\varphi, \sigma) := \begin{cases} 
\sigma \lor \varphi, & \text{if } \varphi \text{ is propositional} \\
\sigma \lor X(\Theta_{ab}^{st}(\psi, \sigma)), & \text{if } \varphi = X\psi \\
\Theta_{ab}^{st}(\psi, \sigma) \otimes \Theta_{ab}^{st}(\gamma, \sigma), & \text{if } \varphi = \psi \otimes \gamma \text{ with } \\
\otimes \in \{\lor, \land, \lor\}. & 
\end{cases}$$

**Examples**

- $\Theta_{ab}^{st}(\text{X}a, \sigma) \equiv \sigma \lor \Theta_{ab}^{st}(\text{X}a, \sigma) \equiv \sigma \lor X(\sigma \lor a)$
- $\Theta_{ab}^{st}(\text{Ga}, \sigma) \equiv \Theta_{ab}^{st}([a \lor \sigma], \sigma) \equiv [(\sigma \lor a) \lor \sigma] \equiv [a \lor \sigma]$  
- $\Theta_{ab}^{st}(\text{Fa}, \sigma) \equiv \Theta_{ab}^{st}([1 \lor \sigma], \sigma) \equiv [1 \lor (a \lor \sigma)] \equiv F(a \lor \sigma)$
Interactive Proof Rule

\[ M \models \varphi \]

\[
\text{abort}\{M\}\text{when}(\sigma) \models \Theta^\text{st}_{ab}(\varphi, \sigma)
\]
Outline

1. Introduction

2. Lifting Verification Results for Preemption Statements
   - Abort Transformation
   - Suspend Transformation

3. Conclusion
Idea

- **suspend** postpones execution
- preemptive specification $\Theta_{sp}^{st}(\varphi)$ for $\varphi$ is satisfied
  - ignoring suspended states satisfies $\varphi$
  - possibility of infinite suspension
Idea for Suspend

φ
Idea for Suspend

\[ \Theta(\phi) \]
Idea for Suspend

\[ \Theta(\varphi) \]
Correct Transformation for Suspend

**Transformation** $\Theta^{st}_{sp}(\varphi, \sigma)$

For a given specification $A\varphi$, the transformation $\Theta^{st}_{sp}(\varphi, \sigma)$ is defined as

$$\Theta^{st}_{sp}(\varphi, \sigma) := \begin{cases} 
[\varphi \lor \neg \sigma], & \text{if } \varphi \text{ is propositional} \\
[(X\Theta^{st}_{sp}(\psi, \sigma)) \lor \neg \sigma], & \text{if } \varphi = X\psi \\
\Theta^{st}_{sp}(\psi, \sigma) \otimes \Theta^{st}_{sp}(\gamma, \sigma), & \text{if } \varphi = \psi \otimes \gamma \text{ with } \\
\otimes \in \{U, U, \land, \lor\}. 
\end{cases}$$

**Examples**

$$\Theta^{st}_{sp}(Xa, \sigma) \equiv [(X\Theta^{st}_{sp}(a, \sigma)) \lor \neg \sigma] \equiv [(X[a \lor \neg \sigma]) \lor \neg \sigma]$$

$$\Theta^{st}_{sp}(Ga, \sigma) \equiv \Theta^{st}_{sp}([a \lor 0], \sigma) \equiv [[a \lor \neg \sigma] \lor 0] \equiv G[a \lor \neg \sigma]$$

$$\Theta^{st}_{sp}(Fa, \sigma) \equiv \Theta^{st}_{sp}([1 \lor a], \sigma) \equiv [1 \lor [a \lor \neg \sigma]] \equiv F[a \lor \neg \sigma]$$
Interactive Proof Rule

\[
M \models \varphi \\
\text{suspend}\{M\}\text{when}(\sigma) \models \Theta_{sp}^{st}(\varphi, \sigma)
\]
Outline

1. Introduction

2. Lifting Verification Results for Preemption Statements

3. Conclusion
Summary

- synchronous model of computation
- preemption behavior
- reuse of verification results in a preemption context
- automatic and correct-by-construction transformation
Application of Suspend Transformation

- Clock refinement
- Translation from synchronous to interleaved guarded actions
Questions?
Already Published Approach

[GeSc13a]
M. Gesell and K. Schneider
Modular Verification of Synchronous Programs, ACSD 2013
Already Published Approach

\[ \text{Module A} \quad \boxed{\text{closed system}} = \varphi \]

\[ \text{Module A} \quad \boxed{\text{open system}} \]

[GeSc13a]
M. Gesell and K. Schneider
Modular Verification of Synchronous Programs, ACSD 2013
Already Published Approach

[GeSc13a]
M. Gesell and K. Schneider
Modular Verification of Synchronous Programs, ACSD 2013
Already Published Approach

[GeSc13a]
M. Gesell and K. Schneider
Modular Verification of Synchronous Programs, ACSD 2013
Already Published Approach

\[ \Psi \]

**Module A**
Already Published Approach

Manuel Gesell
Already Published Approach

Advantages and Disadvantages

+ reuse of verification results

− manual transition step

− additional verification task

○ no necessary relation to the original specification
Averest Design Flow

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

Compilation → AIF Module

Transformation → Verification

AIFProver → SMV

Simulation → Trace

C → Java

SW Synthese → SystemC

Java → VHDL

HW Synthese → Verilog

http://www.averest.org