Translating Synchronous Guarded Actions to Interleaved Guarded Actions

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 Formal Methods and Models for Codeign, October 18-20, 2013 - Portland, Oregon, USA
**Definition: Synchronous Guarded Actions (SGAs)**

A synchronous guarded action \((\gamma \Rightarrow \alpha)\) consists of

- a Boolean guard \(\gamma\) and
- a single atomic immediate/delayed assignment \(\alpha\).

**Behavior of SGAs**

- execution of all enabled guarded actions in parallel
**Definition: Synchronous Guarded Actions (SGAs)**

A synchronous guarded action \((\gamma \Rightarrow \alpha)\) consists of
- a Boolean guard \(\gamma\) and
- a single atomic immediate/delayed assignment \(\alpha\).

**Behavior of SGAs**
- execution of all enabled guarded actions in parallel

**Definition: Interleaved Guarded Actions (IGAs)**

A interleaved guarded action \((\gamma \Rightarrow \alpha)\) consists of
- a Boolean guard \(\gamma\) and
- a set of atomic assignments \(\alpha\).

**Behavior of IGAs (subset of Dijkstra’s Guarded Commands)**
- execution of a single enabled guarded actions
Outline

1. Motivation
2. Problems
3. The Solution
4. Conclusion
Outline

1 Motivation
2 Problems
3 The Solution
4 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 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

Quartz

- imperative synchronous language
- C-like syntax
- **pause** defines start/end of reaction step
- input language for Averest
- compiler generates synchronous guarded actions (SGAs)
What is this talk about?

Quartz  Averest  SGAs  SAL
Quartz  Averest  SGAs  CAOS
Quartz  Averest  SGAs  Murphi
Quartz  Averest  SGAs  Rodin
What is this talk about?

Quartz \rightarrow \text{Averest} \rightarrow \text{SGAs} \rightarrow \text{IGAs} \rightarrow \text{SAL} \rightarrow \text{CAOS} \rightarrow \text{Murphi} \rightarrow \text{Rodin}
What is this talk about?

Quartz → Averest → SGAs → This Talk → IGAs

- SAL
- CAOS
- Murphi
- Rodin
Outline

1. Motivation
2. Problems
3. The Solution
4. Conclusion
Problem #1

\[
\{ 
\text{true } \Rightarrow z = y \\
\text{true } \Rightarrow y = x
\}
\]

\[
\begin{align*}
x &= 1 \\
y &= 0 \\
z &= 0 \\
x &= 1 \\
y &= 1 \\
z &= 1
\end{align*}
\]
Problem #1

\[
\begin{align*}
\text{true } & \Rightarrow \ z=y \\
\text{true } & \Rightarrow \ y=x
\end{align*}
\]
Problem #1

\[
\{ \text{true } \Rightarrow z=y \\
\text{true } \Rightarrow y=x \}
\]

\[
\begin{align*}
x &= 1 \\
y &= 0 \\
z &= 0
\end{align*}
\]

\[
\begin{align*}
x &= 1 \\
y &= 1 \\
z &= 0
\end{align*}
\]

\[
\begin{align*}
x &= 1 \\
y &= 1 \\
z &= 1
\end{align*}
\]

\[
\begin{align*}
x &= 1 \\
y &= 0 \\
z &= 0
\end{align*}
\]
Problem #2

\[
\begin{align*}
\text{true} & \Rightarrow \text{next}(x) = y \\
\text{true} & \Rightarrow \text{next}(y) = x
\end{align*}
\]

\[
\begin{array}{c}
x = 1 \\
y = 0
\end{array}
\quad \Rightarrow \quad
\begin{array}{c}
x = 0 \\
y = 1
\end{array}
\]

next(x) = y
next(y) = x
Problem #2

\[
\begin{align*}
\text{true} & \Rightarrow \text{next}(x) = y \\
\text{true} & \Rightarrow \text{next}(y) = x
\end{align*}
\]
Problem #3

\[
\begin{align*}
\text{true } & \Rightarrow x = z \\
\text{true } & \Rightarrow y = \neg z \\
\text{true } & \Rightarrow \text{next}(z) = \neg z
\end{align*}
\]

\[
\begin{array}{c}
x=1 \\
y=0 \\
z=1
\end{array}
\quad
\begin{array}{c}
x=0 \\
y=1 \\
z=0
\end{array}
\]

Manuel Gesell

Translating SGA to IGA
Problem #3

\[
\begin{align*}
\text{true} & \Rightarrow x = z \\
\text{true} & \Rightarrow y = \neg z \\
\text{true} & \Rightarrow \text{next}(z) = \neg z \\
\end{align*}
\]

Diagram:

- State #1: \(x=1, y=0, z=1\)
- State #2: \(x=0, y=1, z=0\)
- State #3: \(x=1, y=0, z=0\)

Transitions:

- From #1 to #2: \#1
- From #1 to #3: \#2
- From #2 to #1: \#3
- From #2 to #1: \#3
- From #3 to #1: \#2
- From #3 to #2: \#3
Problem #3

\[
\begin{aligned}
&\text{true} \Rightarrow x = z \\
&\text{true} \Rightarrow y = \neg z \\
&\text{true} \Rightarrow \text{next}(z) = \neg z
\end{aligned}
\]
Problem #3

\[
\begin{cases}
\text{true} \Rightarrow x=z \\
\text{true} \Rightarrow y=\neg z \\
\text{true} \Rightarrow \text{next}(z)=\neg z
\end{cases}
\]
Problem #3

\[
\begin{align*}
\text{true} & \Rightarrow x = z \\
\text{true} & \Rightarrow y = \neg z \\
\text{true} & \Rightarrow \text{next}(z) = \neg z
\end{align*}
\]
Problem #3

\[
\begin{array}{l}
\text{true } \Rightarrow x = z \\
\text{true } \Rightarrow y = \neg z \\
\text{true } \Rightarrow \text{next}(z) = \neg z
\end{array}
\]

\[\models G(x \lor y)\]

\[
\begin{array}{c}
x=1 \\
y=0 \\
z=1
\end{array}
\quad
\begin{array}{c}
x=0 \\
y=1 \\
z=0
\end{array}
\]
Problem #3

\[
\begin{align*}
\text{true} & \Rightarrow x = z \\
\text{true} & \Rightarrow y = \neg z \\
\text{true} & \Rightarrow \text{next}(z) = \neg z \\
\end{align*}
\]

\( \not\vDash G(x \lor y) \)
## Summary of Identified Problems

<table>
<thead>
<tr>
<th>Problems to Solve</th>
</tr>
</thead>
<tbody>
<tr>
<td>- assignment behavior</td>
</tr>
<tr>
<td>- execution order</td>
</tr>
<tr>
<td>- reaction step behavior</td>
</tr>
<tr>
<td>- temporal behavior</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Problems
3. The Solution
4. Conclusion
### Problems and Solutions

- **assignment behavior**
  - an immediate assignment may influence all other assignments
  - a delayed assignment does not influence other assignments
  \rightarrow \text{two phase approach}

- **execution order**
  - data-dependency between immediate/delayed assignments
    \rightarrow \text{two phase approach}
  - data-dependency between immediate assignments
    - read access only to already determined values
    - no write after write access (e.g. no multiple execution)
    \rightarrow \text{solved inside first phase}

- **reaction step behavior**
  \rightarrow \text{two phase approach + correct execution order}

- **temporal behavior**
### Problems and Solutions

- **Assignment behavior**
  - An immediate assignment may influence all other assignments.
  - A delayed assignment does not influence other assignments.
    - Two phase approach

- **Execution order**
  - Data dependency between immediate/delayed assignments
    - Two phase approach
  - Data dependency between immediate assignments
    - Read access only to already determined values
    - No write after write access (e.g., no multiple execution)
    - Solved inside first phase

- **Reaction step behavior**
  - Two phase approach + correct execution order

- **Temporal behavior**
### Problems and Solutions

- **Assignment behavior**
  - An immediate assignment may influence all other assignments.
  - A delayed assignment does not influence other assignments.
  \[ \Rightarrow \text{two phase approach} \]

- **Execution order**
  - Data-dependency between immediate/delayed assignments.
  \[ \Rightarrow \text{two phase approach} \]
  - Data-dependency between immediate assignments.
    - Read access only to already determined values.
    - No write after write access (e.g., no multiple execution).
    \[ \Rightarrow \text{solved inside first phase} \]

- **Reaction step behavior**
  \[ \Rightarrow \text{two phase approach + correct execution order} \]

- **Temporal behavior**
Problems and Solutions

- **assignment behavior**
  - an immediate assignment may influence all other assignments
  - a delayed assignment does not influence other assignments
    ⇒ two phase approach

- **execution order**
  - data-dependency between immediate/delayed assignments
    ⇒ two phase approach
  - data-dependency between immediate assignments
    - read access only to already determined values
    - no write after write access (e.g. no multiple execution)
    ⇒ solved inside first phase

- **reaction step behavior**
  ⇒ two phase approach + correct execution order

- **temporal behavior**
**Problems and Solutions**

- **assignment behavior**
  - an immediate assignment may influence all other assignments
  - a delayed assignment does not influence other assignments
    \(\Rightarrow\) two phase approach

- **execution order**
  - data-dependency between immediate/delayed assignments
    \(\Rightarrow\) two phase approach
  - data-dependency between immediate assignments
    - read access only to already determined values
    - no write after write access (e.g. no multiple execution)
    \(\Rightarrow\) solved inside first phase

- **reaction step behavior**
  \(\Rightarrow\) two phase approach + correct execution order

- **temporal behavior**
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  - use valid flag $x_v$ to deactivate guarded actions writing $x$
  - each SGA is represented by an IGA
  - complete behavior of the current step ($\forall x \in \mathcal{V}.x_v = \text{true}$)

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is $\bigwedge_{x \in \mathcal{V}} x_v$
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
    - no complete serialization
      → introduce a valid flag \( x_v \) for each Variable \( x \)
    - prevent write after write access
      → use valid flag \( x_v \) to deactivate guarded actions writing \( x \)
    - each SGA is represented by an IGA
    - complete behavior of the current step (\( \forall x \in V. x_v = \text{true} \))

- **Phase 2: evaluation of delayed assignments**
  - no execution order
    → simultaneous/parallel execution
    → only a single IGA (the conclusion) is required
    ⇒ conclusion’s guard is \( \bigwedge_{x \in V} x_v \)
Two Phase Approach:

- **Phase 1:** evaluation of immediate assignments
  - define execution order
  - respect data dependencies
  - no complete serialization
  
  → introduce a valid flag $x_v$ for each Variable $x$
  
  - prevent write after write access
  
  → use valid flag $x_v$ to deactivate guarded actions writing $x$
  
  ⇒ each SGA is represented by an IGA
  
  ⇒ complete behavior of the current step ($\forall x \in V. x_v = \text{true}$)

- **Phase 2:** evaluation of delayed assignments
  
  - no execution order
  
  → simultaneous/parallel execution
  
  ⇒ only a single IGA (the conclusion) is required
  
  ⇒ conclusion’s guard is $\bigwedge_{x \in V} x_v$
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag $x_v$ for each Variable $x$
    - prevent write after write access
    - use valid flag $x_v$ to deactivate guarded actions writing $x$
    - each SGA is represented by an IGA
    - complete behavior of the current step ($\forall x \in V. x_v = \text{true}$)

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is $\bigwedge_{x \in V} x_v$
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag \( x_v \) for each Variable \( x \)
  - prevent write after write access
  - use valid flag \( x_v \) to deactivate guarded actions writing \( x \)
  - each SGA is represented by an IGA
  - complete behavior of the current step (\( \forall x \in \mathcal{V}. x_v = \text{true} \))

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is \( \bigwedge_{x \in \mathcal{V}} x_v \)
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  
  → introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  
  → use valid flag $x_v$ to deactivate guarded actions writing $x$
  
  ⇒ each SGA is represented by an IGA
  ⇒ complete behavior of the current step ($\forall x \in V. x_v = true$)

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  
  → simultaneous/parallel execution
  
  ⇒ only a single IGA (the conclusion) is required
  
  ⇒ conclusion’s guard is $\bigwedge_{x \in V} x_v$
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  - use valid flag $x_v$ to deactivate guarded actions writing $x$
  - each SGA is represented by an IGA
  - complete behavior of the current step ($\forall x \in \mathcal{V}. x_v = \text{true}$)

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is $\bigwedge_{x \in \mathcal{V}} x_v$
Two Phase Approach:

- **Phase 1**: evaluation of immediate assignments
  - define execution order
  - respect data dependencies
  - no complete serialization
  \[ \rightarrow \] introduce a valid flag \( x_v \) for each Variable \( x \)
  - prevent write after write access
  \[ \rightarrow \] use valid flag \( x_v \) to deactivate guarded actions writing \( x \)
  \[ \Rightarrow \] each SGA is represented by an IGA
  \[ \Rightarrow \] complete behavior of the current step (\( \forall x \in \mathcal{V}. x_v = \text{true} \))

- **Phase 2**: evaluation of delayed assignments
  - no execution order
  \[ \rightarrow \] simultaneous/parallel execution
  \[ \Rightarrow \] only a single IGA (the conclusion) is required
  \[ \Rightarrow \] conclusion’s guard is \( \bigwedge_{x \in \mathcal{V}} x_v \)
Two Phase Approach:

- **Phase 1:** evaluation of immediate assignments
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  - use valid flag $x_v$ to deactivate guarded actions writing $x$
  - each SGA is represented by an IGA
  - complete behavior of the current step ($\forall x \in V. x_v = \text{true}$)

- **Phase 2:** evaluation of delayed assignments
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is $\bigwedge_{x \in V} x_v$
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  → introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  → use valid flag $x_v$ to deactivate guarded actions writing $x$
  ⇒ each SGA is represented by an IGA
  ⇒ complete behavior of the current step ($\forall x \in V. x_v = \text{true}$)

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  → simultaneous/parallel execution
  ⇒ only a single IGA (the conclusion) is required
  ⇒ conclusion’s guard is $\bigwedge_{x \in V} x_v$
Two Phase Approach:

- **Phase 1: evaluation of immediate assignments**
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  - use valid flag $x_v$ to deactivate guarded actions writing $x$
  - each SGA is represented by an IGA
  - complete behavior of the current step ($\forall x \in V. x_v = true$)

- **Phase 2: evaluation of delayed assignments**
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is $\wedge x \in V. x_v$
Two Phase Approach:

- **Phase 1**: evaluation of immediate assignments
  - define execution order
  - respect data dependencies
  - no complete serialization
  - introduce a valid flag $x_v$ for each Variable $x$
  - prevent write after write access
  - use valid flag $x_v$ to deactivate guarded actions writing $x$
  - each SGA is represented by an IGA
  - complete behavior of the current step ($\forall x \in \mathcal{V}.x_v = \text{true}$)

- **Phase 2**: evaluation of delayed assignments
  - no execution order
  - simultaneous/parallel execution
  - only a single IGA (the conclusion) is required
  - conclusion’s guard is $\bigwedge_{x \in \mathcal{V}} x_v$
### Synchronous Guarded Actions for $x$

<table>
<thead>
<tr>
<th>$\gamma_1$</th>
<th>$\Rightarrow$</th>
<th>$x = \tau_1$</th>
<th>$\delta_1$</th>
<th>$\Rightarrow$</th>
<th>$\text{next}(x) = \nu_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\gamma_n$</td>
<td>$\Rightarrow$</td>
<td>$x = \tau_n$</td>
<td>$\delta_m$</td>
<td>$\Rightarrow$</td>
<td>$\text{next}(x) = \nu_m$</td>
</tr>
</tbody>
</table>


Synchronous Guarded Actions for $x$

\[
\begin{align*}
\gamma_1 & \Rightarrow x = \tau_1 \\
\vdots \ \\
\gamma_n & \Rightarrow x = \tau_n \\
\delta_1 & \Rightarrow \text{next}(x) = \upsilon_1 \\
\vdots \ \\
\delta_m & \Rightarrow \text{next}(x) = \upsilon_m
\end{align*}
\]

Interleaved Guarded Actions for $x$ in Phase 1

\[
\begin{align*}
\gamma_1 \land \neg x_v \land \left( \bigwedge_{v \in \text{read}(\gamma_1 \Rightarrow x = \tau_1)} v_v \right) & \Rightarrow \left\{ \begin{array}{l}
x = \tau_1 \\
x_v = \text{true}
\end{array} \right. \\
\vdots \\
\gamma_n \land \neg x_v \land \left( \bigwedge_{v \in \text{read}(\gamma_n \Rightarrow x = \tau_n)} v_v \right) & \Rightarrow \left\{ \begin{array}{l}
x = \tau_n \\
x_v = \text{true}
\end{array} \right.
\end{align*}
\]
Synchronous Guarded Actions for $x$

\[
\begin{align*}
\gamma_1 & \Rightarrow x = \tau_1 \\
\vdots \\
\gamma_n & \Rightarrow x = \tau_n \\
\delta_1 & \Rightarrow \text{next}(x) = v_1 \\
\vdots \\
\delta_m & \Rightarrow \text{next}(x) = v_m
\end{align*}
\]

Interleaved Guarded Actions for $x$ in Phase 1

\[
\begin{align*}
\gamma_1 \land \neg x_v \land \left( \bigwedge_{v \in \text{read}(\gamma_1 \Rightarrow x = \tau_1)} v_v \right) & \Rightarrow \left\{ \begin{array}{l}
x = \tau_1 \\
x_v = \text{true}
\end{array} \right. \\
\vdots \\
\gamma_n \land \neg x_v \land \left( \bigwedge_{v \in \text{read}(\gamma_n \Rightarrow x = \tau_n)} v_v \right) & \Rightarrow \left\{ \begin{array}{l}
x = \tau_n \\
x_v = \text{true}
\end{array} \right. \\
\left( \bigwedge_{i=1\ldots n} \neg \gamma_i \right) \land \neg x_v \land \left( \bigwedge_{v \in \text{read}(\gamma_i)} v_v \right) & \Rightarrow \left\{ x_v = \text{true} \right\}
\end{align*}
\]
### Synchronous Guarded Actions for $x$

$$\gamma_1 \Rightarrow x = \tau_1 \quad \delta_1 \Rightarrow \text{next}(x) = \nu_1$$
$$\vdots$$
$$\gamma_n \Rightarrow x = \tau_n \quad \delta_m \Rightarrow \text{next}(x) = \nu_m$$

### Interleaved Guarded Actions for $x$ in Phase 2

$$\bigwedge_{v \in V} v_v \Rightarrow x = \begin{cases} \nu_1 & : \text{if } \delta_1 \\ \vdots \\ \nu_m & : \text{if } \delta_m \\ \text{defaultVal}(x) & : \text{else} \end{cases}$$
$$x_v = \bigvee_{i=1\ldots m} \delta_i$$
Execution Behavior

SGA

IGA
## Problem #3

### SGAs

\[
\begin{align*}
\text{true } & \Rightarrow x = z \\
\text{true } & \Rightarrow y = \neg z \\
\text{true } & \Rightarrow \text{next}(z) = \neg z
\end{align*}
\]

### IGAs

\[
\begin{align*}
\neg x_v \land z_v & \Rightarrow \begin{cases} 
x = z \\
x_v = \text{true}
\end{cases} \\
\neg y_v \land z_v & \Rightarrow \begin{cases} 
y = \neg z \\
y_v = \text{true}
\end{cases} \\
x_v \land y_v \land z_v & \Rightarrow \begin{cases} 
z = \neg z \\
z_v = \text{true} \\
x_v = \text{false} \\
y_v = \text{false}
\end{cases}
\end{align*}
\]
Problem #3

IGAs

\[
\begin{align*}
\neg x_v \land z_v & \Rightarrow \begin{cases} 
  x = z \\
  x_v = true \\
  \end{cases} \\
\neg y_v \land z_v & \Rightarrow \begin{cases} 
  y = \neg z \\
  y_v = true \\
  \end{cases} \\
x_v \land y_v \land z_v & \Rightarrow \\
& \begin{cases} 
  z = \neg z \\
  z_v = true \\
  x_v = false \\
  y_v = false \\
  \end{cases}
\end{align*}
\]

\[
\begin{align*}
& x=1 \\
& y=0 \\
& z=1
\end{align*}
\]

\[
\begin{align*}
& x=0 \\
& y=1 \\
& z=0
\end{align*}
\]

\[
\begin{align*}
& x=1 \\
& y=0 \\
& z=0
\end{align*}
\]

\[
\begin{align*}
& x=0 \\
& y=0 \\
& z=0
\end{align*}
\]

\[
\begin{align*}
& x=1 \\
& y=1 \\
& z=0
\end{align*}
\]
Problems and Solutions

- **assignment behavior**
  - an immediate assignment may influence all other assignments
  - a delayed assignment does not influence other assignments
    \[ \Rightarrow \text{two phase approach} \]

- **execution order**
  - data-dependency between immediate/delayed assignments
    \[ \Rightarrow \text{two phase approach} \]
  - data-dependency between immediate assignments
    - read access only to already determined values
    - no write after write access (e.g. no multiple execution)
    \[ \Rightarrow \text{solved inside first phase} \]

- **reaction step behavior**
  \[ \Rightarrow \text{two phase approach + correct execution order} \]

- **temporal behavior**
Reuse of an Existing Method

M. Gesell, A. Morgenstern, and K. Schneider
Lifting Verification Results for Preemption Statements
Software Engineering and Formal Methods (SEFM) 2013

Summary

- reuse of verification results in a preemption context
- generating refined temporal logic specifications
- preserving as 'much as possible'
- automatic and correct-by-construction transformation
  ⇒ reuse of suspend-sensitive transformation Θ
Idea for Suspend

\[ \varphi \]
Idea for Suspend

Θ(φ)
Execution Behavior

SGA

IGA
Application of Suspend Transformation

$\varphi$

$\Theta(\varphi)$

SGA

IGA

Translating SGA to IGA
Problem #3

SGAs
\[
\begin{align*}
\text{true} & \Rightarrow x = z \\
\text{true} & \Rightarrow y = \neg z \\
\text{true} & \Rightarrow \text{next}(z) = \neg z \\
\end{align*}
\]
\[\models G (x \lor y)\]

IGAs
\[
\begin{align*}
\neg x_v \land z_v & \Rightarrow \begin{cases} 
    x = z \\
    x_v = \text{true} \\
    y = \neg z \\
    y_v = \text{true} \\
    z = \neg z
\end{cases} \\
\neg y_v \land z_v & \Rightarrow \begin{cases} 
    z_v = \text{true} \\
    x_v = \text{false} \\
    y_v = \text{false}
\end{cases} \\
x_v \land y_v \land z_v & \Rightarrow \begin{cases} 
    z_v = \text{true} \\
    x_v = \text{false} \\
    y_v = \text{false}
\end{cases}
\end{align*}
\]
\[\models G [\neg (x_v \land y_v \land z_v) \cup (x \lor y)]\]
Problem #3

IGAs

\[
\begin{align*}
\neg x_v \land z_v &\Rightarrow \begin{cases} 
 x = z \\
 x_v = \text{true} \\
 y = \neg z
\end{cases} \\
\neg y_v \land z_v &\Rightarrow \begin{cases} 
 y = \neg z \\
 y_v = \text{true}
\end{cases} \\
x_v \land y_v \land z_v &\Rightarrow \begin{cases} 
 z = \neg z \\
 z_v = \text{true} \\
 x_v = \text{false} \\
 y_v = \text{false}
\end{cases}
\end{align*}
\]

\[\models G \left[ \neg (x_v \land y_v \land z_v) \cup (x \lor y) \right]\]
Outline

1 Motivation
2 Problems
3 The Solution
4 Conclusion
Summary

- synchronous model of computation
- identified problems for the translation of SGAs to IGAs
- solution: 2 phase approach and valid flags
- reuse a method that lifts verification results for preemption
The End

Questions?
Averest Design Flow

Quartz → Compilation → AIF Module

Transformation

AIF System

Linking

Verification → AIFProver
SMV

Simulation
Trace
C
Java
SystemC
VHDL
Verilog

SW Synthesis
HW Synthesis

http://www.averest.org