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 Codesign, October 18-20, 2013 - Portland, Oregon, USA

Definition: Synchronous Guarded Actions (SGAs)

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

- a Boolean guard γ and
- a single atomic immediate/delayed assignment α.

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 γ and
- a single atomic immediate/delayed assignment α.

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 γ and
- a set of atomic assignments α.

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

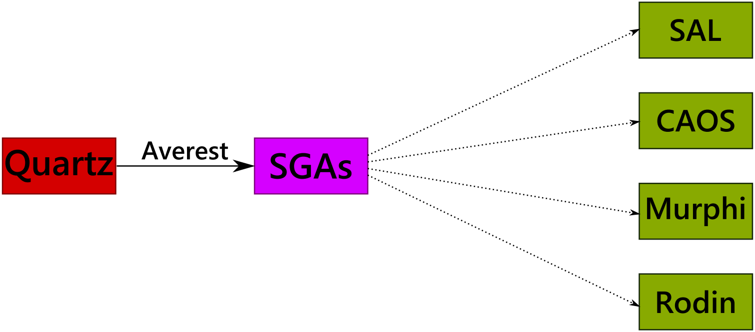
Synchronous Model of Computation

- 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

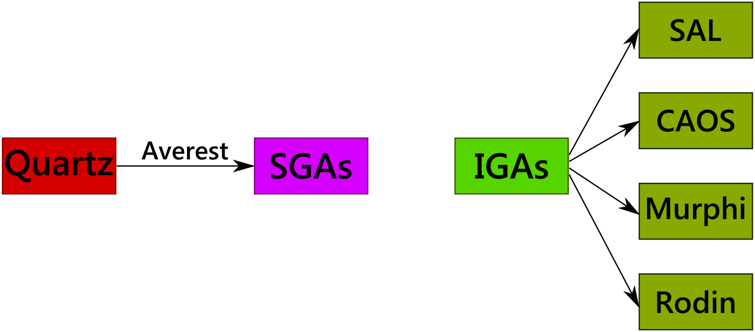
Quartz

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

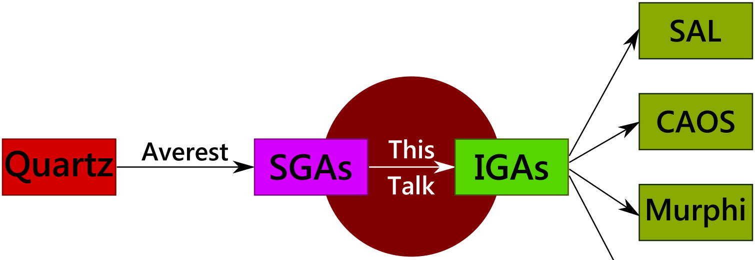
What is this talk about?



What is this talk about?



What is this talk about?



Outline

1 Motivation

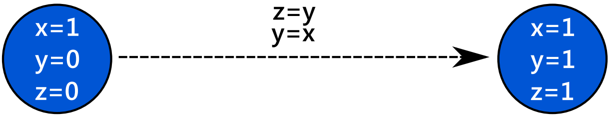
2 Problems

3 The Solution

4 Conclusion

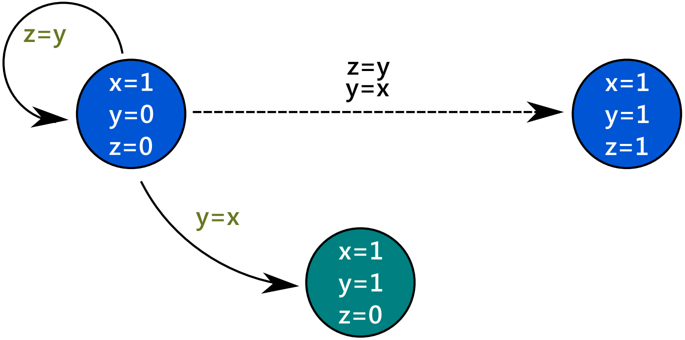
Problem #1

$$\left\{ \begin{array}{l} \text{true} \Rightarrow z{=}y \\ \text{true} \Rightarrow y{=}x \end{array} \right\}$$



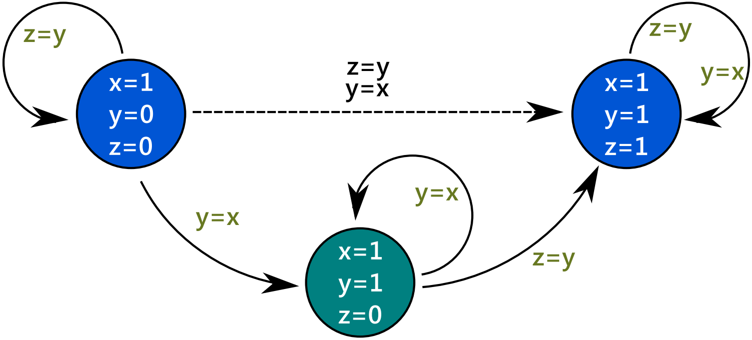
Problem #1

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \text{z=y} \\ \text{true} \Rightarrow \text{y=x} \end{array} \right\}$$



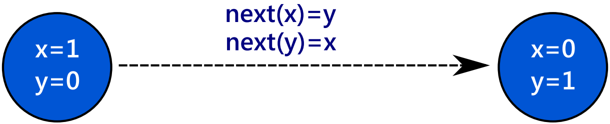
Problem #1

$$\left\{ \begin{array}{l} \text{true} \Rightarrow z{=}y \\ \text{true} \Rightarrow y{=}x \end{array} \right\}$$



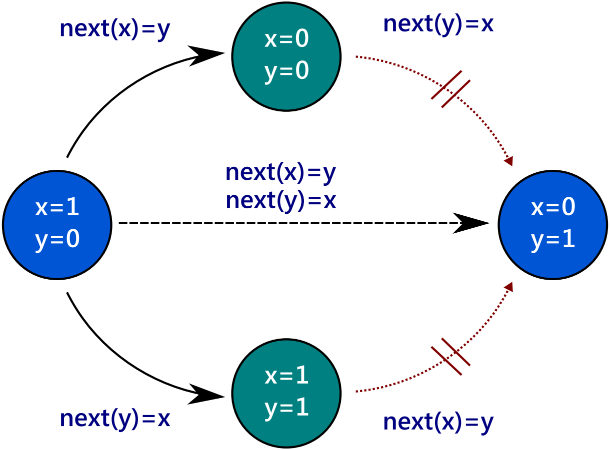
Problem #2

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \textbf{next}(x)=y \\ \text{true} \Rightarrow \textbf{next}(y)=x \end{array} \right\}$$



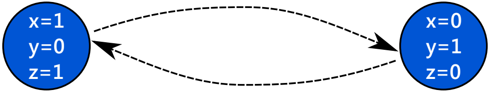
Problem #2

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \mathbf{next}(x)=y \\ \text{true} \Rightarrow \mathbf{next}(y)=x \end{array} \right\}$$



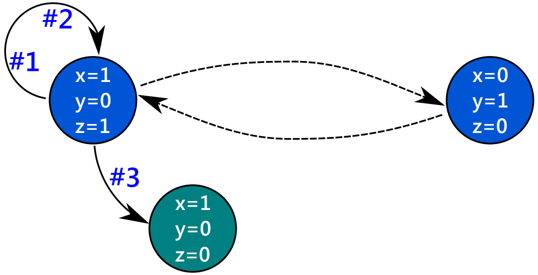
Problem #3

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \texttt{x=z} \\ \text{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \text{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\}$$



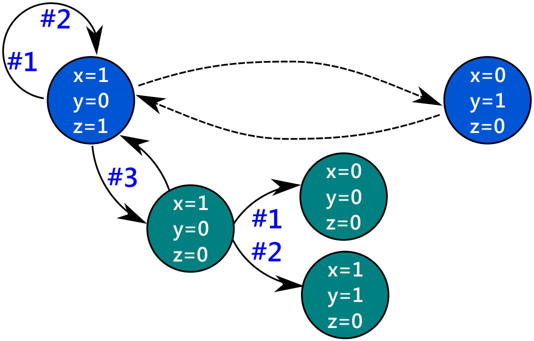
Problem #3

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \texttt{x=z} \\ \text{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \text{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\}$$



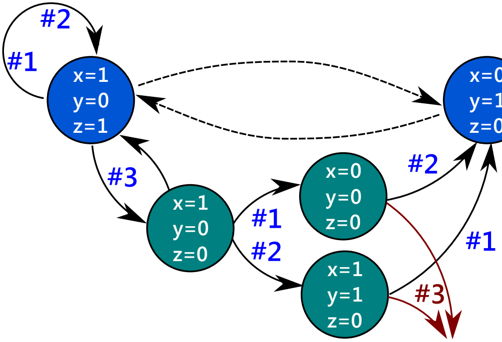
Problem #3

$$\left\{\begin{array}{l} \text{true} \Rightarrow \texttt{x=z} \\ \text{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \text{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array}\right\}$$



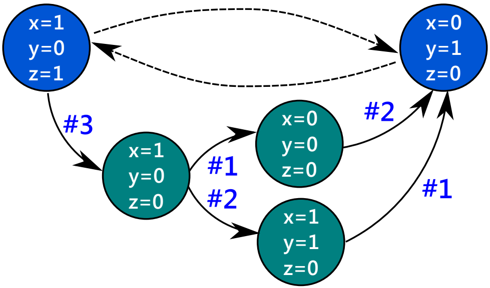
Problem #3

$$\left\{ \begin{array}{l} \texttt{true} \Rightarrow \texttt{x=z} \\ \texttt{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \texttt{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\}$$



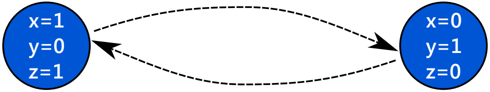
Problem #3

$$\left\{ \begin{array}{l} \texttt{true} \Rightarrow \texttt{x=z} \\ \texttt{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \texttt{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\}$$



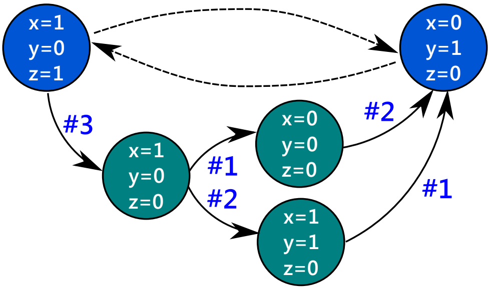
Problem #3

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \texttt{x=z} \\ \text{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \text{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\} \models \mathbf{G}\ (\texttt{x}\vee\texttt{y})$$



Problem #3

$$\left\{ \begin{array}{l} \mathtt{true} \Rightarrow \mathtt{x=z} \\ \mathtt{true} \Rightarrow \mathtt{y=\neg z} \\ \mathtt{true} \Rightarrow \mathbf{next}\,\mathtt{(z)=\neg z} \end{array} \right\} \not\models \mathbf{G}\ \mathtt{(x \vee y)}$$



Summary of Identified Problems

Problems to Solve

- assignment behavior
- execution order
- reaction step behavior
- temporal behavior

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

    $\Rightarrow$ 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

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

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

    ⇒ 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 ($\forall x \in \mathcal{V}.x_v = \text{true}$)
- 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 ($\forall x \in \mathcal{V}.x_v = \text{true}$)
- 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 ($\forall x \in \mathcal{V}.x_v = \text{true}$)
- 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 $\mathtt{x}_v$ for each Variable $\mathtt{x}$
  - prevent write after write access
  - → use valid flag $\mathtt{x}_v$ to deactivate guarded actions writing $\mathtt{x}$
  - ⇒ each SGA is represented by an IGA
  - ⇒ complete behavior of the current step ($\forall \mathtt{x} \in \mathcal{V}.\mathtt{x}_v = \text{true}$)
- Phase 2: evaluation of delayed assignments
  - no execution order
  - → simultaneous/parallel execution
  - ⇒ only a single IGA (the conclusion) is required
  - ⇒ conclusion's guard is $\bigwedge_{\mathtt{x} \in \mathcal{V}} \mathtt{x}_v$

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 $\mathtt{x}_v$ for each Variable $\mathtt{x}$
  - prevent write after write access

  $\rightarrow$ use valid flag $\mathtt{x}_v$ to deactivate guarded actions writing $\mathtt{x}$

  $\Rightarrow$ each SGA is represented by an IGA

  $\Rightarrow$ complete behavior of the current step ($\forall \mathtt{x} \in \mathcal{V}.\mathtt{x}_\mathsf{v} = \mathsf{true}$)
- 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 $\bigwedge_{\mathtt{x} \in \mathcal{V}} \mathtt{x}_\mathsf{v}$

Two Phase Approach:

- Phase 1: evaluation of immediate assignments
  - define execution order
  - respect data dependencies
  - no complete serialization

  → introduce a valid flag $\mathtt{x}_v$ for each Variable $\mathtt{x}$
  - prevent write after write access

  → use valid flag $\mathtt{x}_v$ to deactivate guarded actions writing $\mathtt{x}$

  ⇒ each SGA is represented by an IGA

  ⇒ complete behavior of the current step ($\forall \mathtt{x} \in \mathcal{V}.\mathtt{x}_\mathsf{v} = \mathsf{true}$)
- Phase 2: evaluation of delayed assignments
  - no execution order

  → simultaneous/parallel execution

  ⇒ only a single IGA (the conclusion) is required

  ⇒ conclusion's guard is $\bigwedge_{\mathtt{x} \in \mathcal{V}} \mathtt{x}_\mathsf{v}$

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 $\mathtt{x}_v$ for each Variable $\mathtt{x}$
  - prevent write after write access
  - → use valid flag $\mathtt{x}_v$ to deactivate guarded actions writing $\mathtt{x}$
  - ⇒ each SGA is represented by an IGA
  - ⇒ complete behavior of the current step $(\forall \mathtt{x} \in \mathcal{V}.\mathtt{x}_\mathsf{v} = \mathsf{true})$
- Phase 2: evaluation of delayed assignments
  - no execution order
  - → simultaneous/parallel execution
  - ⇒ only a single IGA (the conclusion) is required
  - ⇒ conclusion's guard is $\bigwedge_{\mathtt{x} \in \mathcal{V}} \mathtt{x}_\mathsf{v}$

Two Phase Approach:

- Phase 1: evaluation of immediate assignments
  - define execution order
  - respect data dependencies
  - no complete serialization
  - → introduce a valid flag $\mathtt{x}_v$ for each Variable $\mathtt{x}$
  - prevent write after write access
  - → use valid flag $\mathtt{x}_v$ to deactivate guarded actions writing $\mathtt{x}$
  - ⇒ each SGA is represented by an IGA
  - ⇒ complete behavior of the current step ($\forall \mathtt{x} \in \mathcal{V}.\mathtt{x}_\mathsf{v} = \mathsf{true}$)
- Phase 2: evaluation of delayed assignments
  - no execution order
  - → simultaneous/parallel execution
  - ⇒ only a single IGA (the conclusion) is required
  - ⇒ conclusion's guard is $\bigwedge_{\mathtt{x} \in \mathcal{V}} \mathtt{x}_\mathsf{v}$

Synchronous Guarded Actions for x

$$
\begin{array}{llll|llll}
\gamma_1 & \Rightarrow & \mathtt{x} & = \tau_1 & \delta_1 & \Rightarrow & \mathbf{next}(\mathtt{x}) & = \upsilon_1 \\
 & \vdots & & & & \vdots & & \\
\gamma_n & \Rightarrow & \mathtt{x} & = \tau_n & \delta_m & \Rightarrow & \mathbf{next}(\mathtt{x}) & = \upsilon_m
\end{array}
$$

Synchronous Guarded Actions for x

$$\begin{array}{llll|llll} \gamma_1 & \Rightarrow & \texttt{x} & = \tau_1 & \delta_1 & \Rightarrow & \textbf{next}(\texttt{x}) & = \upsilon_1 \\ & \vdots & & & & \vdots & & \\ \gamma_n & \Rightarrow & \texttt{x} & = \tau_n & \delta_m & \Rightarrow & \textbf{next}(\texttt{x}) & = \upsilon_m \end{array}$$

Interleaved Guarded Actions for x in Phase 1

$$\gamma_1 \wedge \neg \texttt{x}_\mathsf{v} \wedge \left(\bigwedge_{\texttt{v} \in \mathsf{read}(\gamma_1 \Rightarrow \texttt{x} = \tau_1)} \texttt{v}_\mathsf{v} \right) \Rightarrow \left\{ \begin{array}{ll} \texttt{x} & = \tau_1 \\ \texttt{x}_\mathsf{v} & = \mathsf{true} \end{array} \right\}$$

$$\vdots$$

$$\gamma_n \wedge \neg \texttt{x}_\mathsf{v} \wedge \left(\bigwedge_{\texttt{v} \in \mathsf{read}(\gamma_n \Rightarrow \texttt{x} = \tau_n)} \texttt{v}_\mathsf{v} \right) \Rightarrow \left\{ \begin{array}{ll} \texttt{x} & = \tau_n \\ \texttt{x}_\mathsf{v} & = \mathsf{true} \end{array} \right\}$$

Synchronous Guarded Actions for x

$$
\begin{array}{llll|llll}
\gamma_1 & \Rightarrow & \mathtt{x} & = \tau_1 & \delta_1 & \Rightarrow & \mathbf{next}(\mathtt{x}) & = \upsilon_1 \\
 & \vdots & & & & \vdots & & \\
\gamma_n & \Rightarrow & \mathtt{x} & = \tau_n & \delta_m & \Rightarrow & \mathbf{next}(\mathtt{x}) & = \upsilon_m
\end{array}
$$

Interleaved Guarded Actions for x in Phase 1

$$
\gamma_1 \wedge \neg \mathtt{x}_\mathsf{v} \wedge \left(\bigwedge_{\mathtt{v} \in \mathsf{read}(\gamma_1 \Rightarrow \mathtt{x} = \tau_1)} \mathtt{v}_\mathsf{v} \right) \Rightarrow \left\{ \begin{array}{ll} \mathtt{x} & = \tau_1 \\ \mathtt{x}_\mathsf{v} & = \mathsf{true} \end{array} \right\}
$$

$$
\vdots
$$

$$
\gamma_n \wedge \neg \mathtt{x}_\mathsf{v} \wedge \left(\bigwedge_{\mathtt{v} \in \mathsf{read}(\gamma_n \Rightarrow \mathtt{x} = \tau_n)} \mathtt{v}_\mathsf{v} \right) \Rightarrow \left\{ \begin{array}{ll} \mathtt{x} & = \tau_n \\ \mathtt{x}_\mathsf{v} & = \mathsf{true} \end{array} \right\}
$$

$$
\left(\bigwedge_{i=1\ldots n} \neg \gamma_i \right) \wedge \neg \mathtt{x}_\mathsf{v} \wedge \left(\bigwedge_{\mathtt{v} \in \mathsf{read}(\gamma_i)} \mathtt{v}_\mathsf{v} \right) \Rightarrow \left\{ \mathtt{x}_\mathsf{v} = \mathsf{true} \right\}
$$

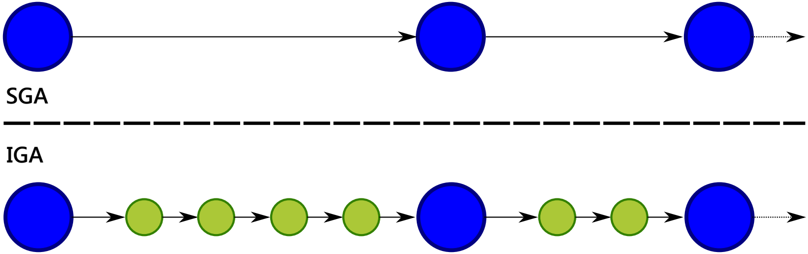
Synchronous Guarded Actions for x

$$\begin{array}{llll|llll} \gamma_1 & \Rightarrow & \mathtt{x} & = \tau_1 & \delta_1 & \Rightarrow & \mathbf{next}(\mathtt{x}) & = \upsilon_1 \\ & \vdots & & & & \vdots & & \\ \gamma_n & \Rightarrow & \mathtt{x} & = \tau_n & \delta_m & \Rightarrow & \mathbf{next}(\mathtt{x}) & = \upsilon_m \end{array}$$

Interleaved Guarded Actions for x in Phase 2

$$\bigwedge_{v \in \mathcal{V}} v_{\mathsf{v}} \Rightarrow \left\{ \begin{array}{l} \vdots \\ x = \begin{cases} \upsilon_1 & : \text{ if } \delta_1 \\ \vdots & \\ \upsilon_m & : \text{ if } \delta_m \\ \mathsf{defaultVal}(x) & : \text{ else} \end{cases} \\ x_{\mathsf{v}} = \bigvee_{i=1\ldots m} \delta_i \\ \vdots \end{array} \right\}$$

Execution Behavior



Problem #3

SGAs

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \texttt{x=z} \\ \text{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \text{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\}$$

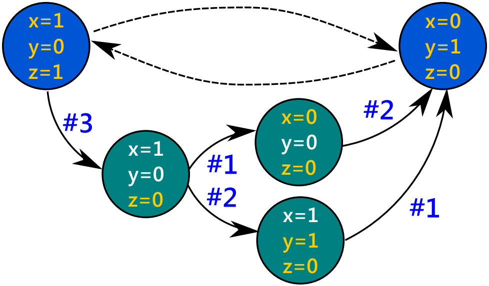
IGAs

$$\left\{ \begin{array}{l} \neg \mathtt{x}_v \wedge \mathtt{z}_v \Rightarrow \left\{ \begin{array}{l} \mathtt{x} = \mathtt{z} \\ \mathtt{x}_v = \textbf{true} \end{array} \right\} \\ \neg \mathtt{y}_v \wedge \mathtt{z}_v \Rightarrow \left\{ \begin{array}{l} \mathtt{y} = \neg \mathtt{z} \\ \mathtt{y}_v = \textbf{true} \end{array} \right\} \\ \mathtt{x}_v \wedge \mathtt{y}_v \wedge \mathtt{z}_v \Rightarrow \left\{ \begin{array}{l} \mathtt{z} = \neg \mathtt{z} \\ \mathtt{z}_v = \textbf{true} \\ \mathtt{x}_v = \textbf{false} \\ \mathtt{y}_v = \textbf{false} \end{array} \right\} \end{array} \right\}$$

Problem #3

IGAs

$$\left\{ \begin{array}{l} \neg x_v \wedge z_v \Rightarrow \left\{ \begin{array}{l} x = z \\ x_v = \mathbf{true} \end{array} \right\} \\ \neg y_v \wedge z_v \Rightarrow \left\{ \begin{array}{l} y = \neg z \\ y_v = \mathbf{true} \end{array} \right\} \\ x_v \wedge y_v \wedge z_v \Rightarrow \left\{ \begin{array}{l} z = \neg z \\ z_v = \mathbf{true} \\ x_v = \mathbf{false} \\ y_v = \mathbf{false} \end{array} \right\} \end{array} \right\}$$



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

Reuse of an Existing Method

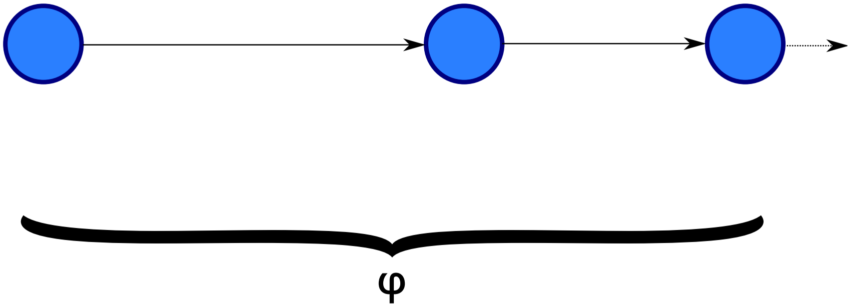
M. Gesell, A. Morgenstern, and K. Schneider
Lifting Verification Results for Preemption Statements
Software Engineering and Formal Methods (SEFM) 2013

Summary

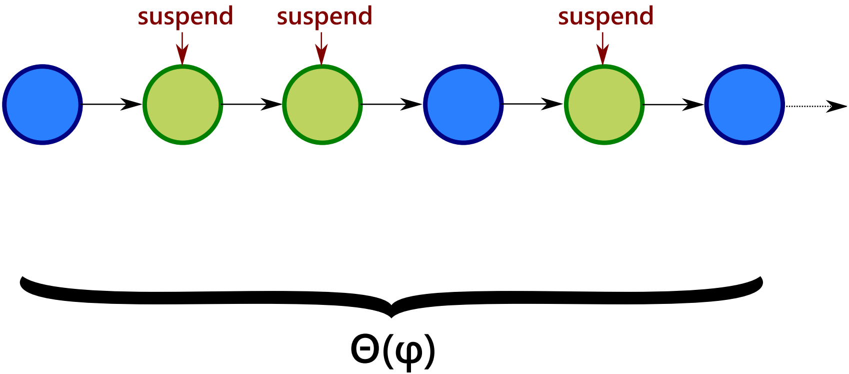
- reuse of verification results in a preemption context
- generating refined temporal logic specifications
- preserving as 'much as possible'
- automatic and correct-by-construction transformation

$\Rightarrow$ reuse of suspend-sensitive transformation Θ

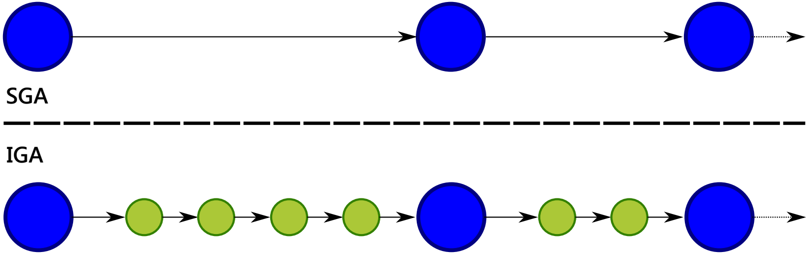
Idea for Suspend



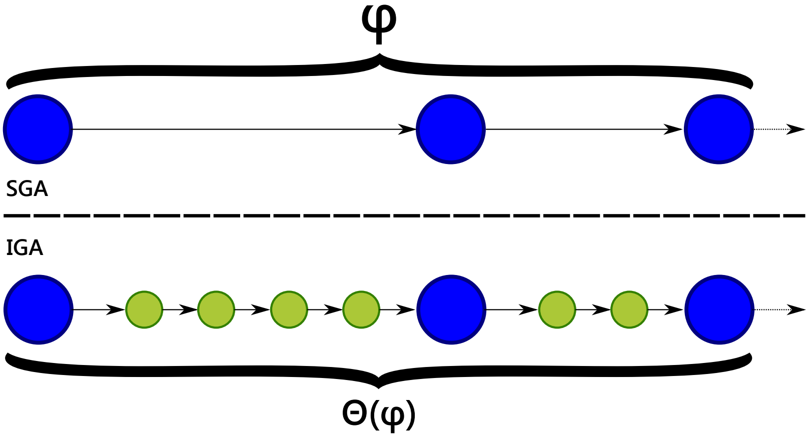
Idea for Suspend



Execution Behavior



Application of Suspend Transformation



Problem #3

SGAs

$$\left\{ \begin{array}{l} \text{true} \Rightarrow \texttt{x=z} \\ \text{true} \Rightarrow \texttt{y=}\neg\texttt{z} \\ \text{true} \Rightarrow \textbf{next}\texttt{(z)=}\neg\texttt{z} \end{array} \right\} \models \mathbf{G}\ (\texttt{x}\vee\texttt{y})$$

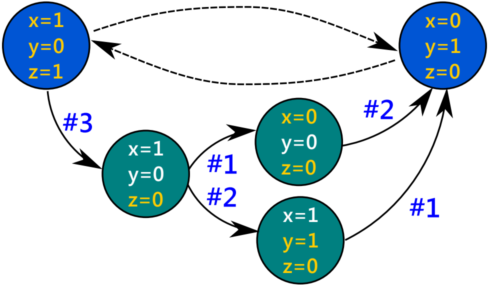
IGAs

$$\left\{ \begin{array}{l} \neg x_v \wedge z_v \Rightarrow \left\{ \begin{array}{l} x = z \\ x_v = \textbf{true} \end{array} \right\} \\ \neg y_v \wedge z_v \Rightarrow \left\{ \begin{array}{l} y = \neg z \\ y_v = \textbf{true} \end{array} \right\} \\ x_v \wedge y_v \wedge z_v \Rightarrow \left\{ \begin{array}{l} z = \neg z \\ z_v = \textbf{true} \\ x_v = \textbf{false} \\ y_v = \textbf{false} \end{array} \right\} \end{array} \right\} \models \mathbf{G}\left[\neg\left(x_v \wedge y_v \wedge z_v\right) \mathsf{U} \left(x \vee y\right)\right]$$

Problem #3

IGAs

$$\left\{\begin{array}{l} \neg x_v \wedge \mathbf{z}_v \Rightarrow \left\{\begin{array}{l} x = \mathbf{z} \\ x_v = \mathbf{true} \end{array}\right\} \\ \neg y_v \wedge \mathbf{z}_v \Rightarrow \left\{\begin{array}{l} y = \neg\mathbf{z} \\ y_v = \mathbf{true} \end{array}\right\} \\ x_v \wedge y_v \wedge \mathbf{z}_v \Rightarrow \left\{\begin{array}{l} \mathbf{z} = \neg\mathbf{z} \\ \mathbf{z}_v = \mathbf{true} \\ x_v = \mathbf{false} \\ y_v = \mathbf{false} \end{array}\right\} \end{array}\right\} \models \mathbf{G}\left[\neg(x_v \wedge y_v \wedge \mathbf{z}_v) \,\mathbf{U}\, (x \vee y)\right]$$



Outline

1 Motivation

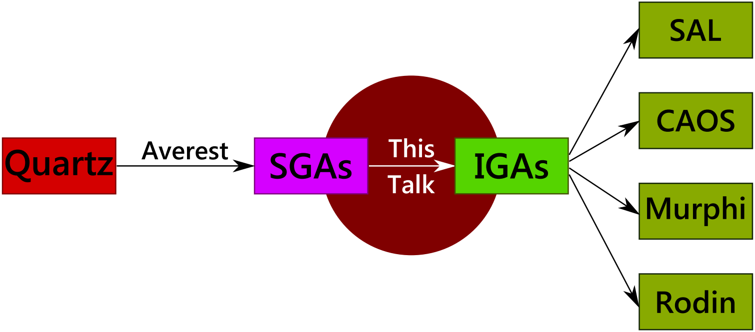
2 Problems

3 The Solution

4 Conclusion

Summary

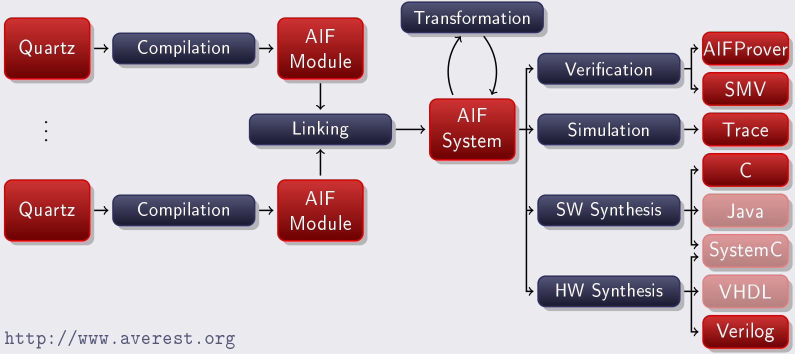
- synchronous model of computation
- identified problems for the translation of SGAs to IGAs
- solution: 2 phase approach and valid flags
- reuse a method that lifts verification results for preemption



The End

Questions?

Averest Design Flow



http://www.averest.org