# Analyzing Reconfigurable Component-Based Systems Using Attribute Grammars

Thomas Noll

Software Modeling and Verification Group
RWTH Aachen University

noll@cs.rwth-aachen.de

FACS 2011; September 14, 2011; Oslo, Norway

# Overview

# The ESA COMPASS Project

(http://compass.informatik.rwth-aachen.de/)

## Overall objective

Develop a model-based approach to system-software co-engineering while focusing on a coherent set of modeling and analysis techniques for evaluating system-level correctness, safety, dependability, and performance of on-board computer-based aerospace systems.


## Derived objectives

1. Modeling formalism: System-Level Integrated Modeling Language (SLIM; "extended subset" of AADL)
2. Verification methodology based on state-of-the-art formal methods
3. Toolset supporting the analysis of SLIM models
4. Evaluation on industrial-size case studies from aerospace domain

# The ESA COMPASS Project

(http://compass.informatik.rwth-aachen.de/)

## Overall objective

Develop a model-based approach to system-software co-engineering while focusing on a coherent set of modeling and analysis techniques for evaluating system-level correctness, safety, dependability, and performance of on-board computer-based aerospace systems.

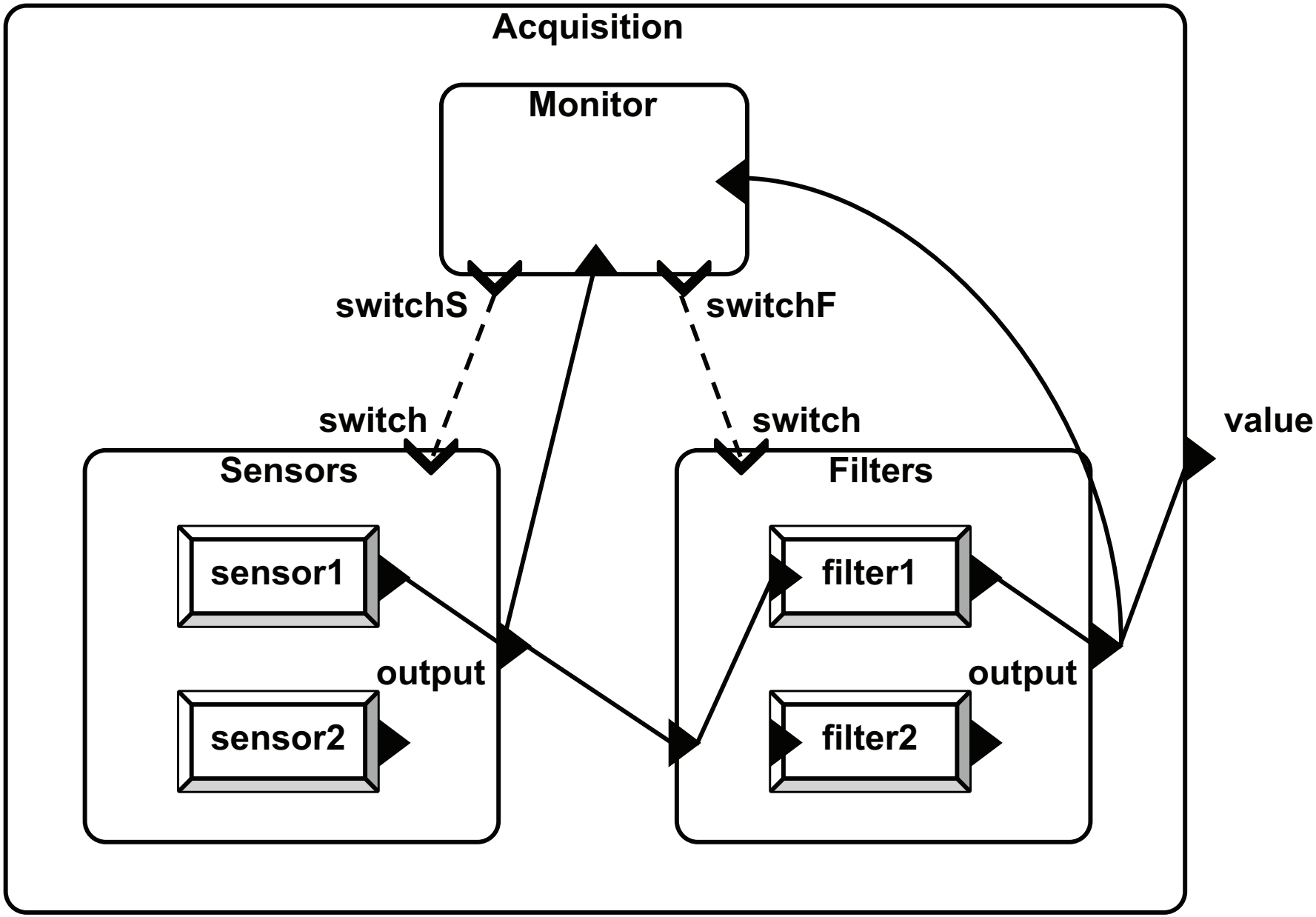

## Derived objectives

1. Modeling formalism: System-Level Integrated Modeling Language (SLIM; "extended subset" of AADL)
2. Verification methodology based on state-of-the-art formal methods
3. Toolset supporting the analysis of SLIM models
4. Evaluation on industrial-size case studies from aerospace domain

# The Architecture Analysis and Design Language

- System = hierarchy of interacting (HW/SW) components
- Ports provide interaction interfaces
  - event ports: for (multi-way) hand-shaking communication
  - data ports: for continuous exchange of values
- Connections define interaction topology
  - event port connections
  - (inter-component) data port connections
  - (intra-component) data flows
- Mode transition system specifies component nominal behavior ($\approx$ timed automaton)
- Error occurrence and handling
- Dynamic reconfiguration: mode-dependent (de-)activation of components and connections

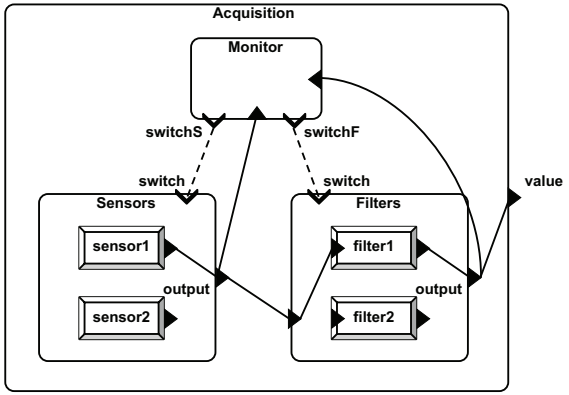
# A Reconfigurable Data Acquisition System



# The Main Component

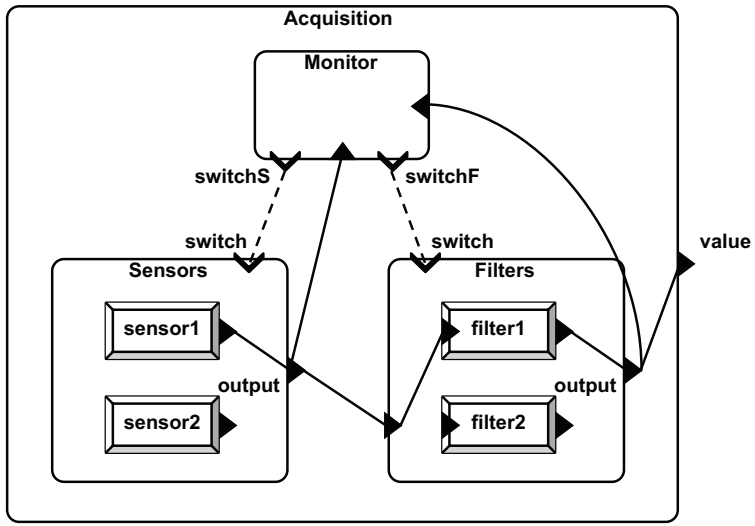
```
system Acquisition
  features
    value: out data port real;
end Acquisition;

system implementation Acquisition.Impl
  subcomponents
    sensors: system Sensors;
    filters: system Filters;
    monitor: system Monitor;
  connections
    data port sensors.output -> filters.input;
    data port sensors.output -> monitor.valueS;
    data port filters.output -> value;
    data port filters.output -> monitor.valueF;
    event port monitor.switchS -> sensors.switch;
    event port monitor.switchF -> filters.switch;
end Acquisition.Impl;
```



# The Sensor Part

```
system Sensors
  features
    output: out data port real;
    switch:  in event port;
end Sensors;
system implementation Sensors.Impl
  subcomponents
    sensor1: device Sensor in modes (Primary);
    sensor2: device Sensor in modes (Backup);
  connections
    data port sensor1.output -> output in modes (Primary);
    data port sensor2.output -> output in modes (Backup);
  modes
    Primary: initial mode;
    Backup: mode;
  transitions
    Primary -[switch]-> Backup;
end Sensors.Impl;
device Sensor
  features
    output: out data port real;
end Sensor;
device implementation Sensor.Impl
  ...
end Sensor.Impl;
```



# Cyclic Data Port Dependencies

- Mode transitions and event port connections yield configuration transition system
  - configuration = current mode of each active component + data values
- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration

$\Rightarrow$ Data port dependencies (as imposed by data port connections/flows) must always be acyclic

**Definition (Circularity of AADL specifications)**

An AADL specification is called circular if there exists a mode configuration such that the corresponding data port dependency graph has a (directed) cycle. Otherwise it is called noncircular.

**Note:** restriction to reachable mode configuration undecidable

$\Longrightarrow$ Approximation by considering all combinations of modes

# Cyclic Data Port Dependencies

- Mode transitions and event port connections yield configuration transition system
  - configuration = current mode of each active component + data values
- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration

$\Rightarrow$ Data port dependencies (as imposed by data port connections/flows) must always be acyclic

**Definition (Circularity of AADL specifications)**

An AADL specification is called circular if there exists a mode configuration such that the corresponding data port dependency graph has a (directed) cycle. Otherwise it is called noncircular.

**Note:** restriction to reachable mode configuration undecidable

$\Longrightarrow$ Approximation by considering all combinations of modes

# Cyclic Data Port Dependencies

- Mode transitions and event port connections yield configuration transition system
  - configuration = current mode of each active component + data values
- Data port connections/flows induce equation system over data port values
- Required: unique solution in each system configuration

$\Rightarrow$ Data port dependencies (as imposed by data port connections/flows) must always be acyclic

**Definition (Circularity of AADL specifications)**

An AADL specification is called circular if there exists a mode configuration such that the corresponding data port dependency graph has a (directed) cycle. Otherwise it is called noncircular.

**Note:** restriction to reachable mode configuration undecidable

$\Longrightarrow$ Approximation by considering all combinations of modes

# A Simple Circular Specification

```
system Cyclic
end Cyclic;
system implementation Cyclic.Impl
  subcomponents
    inc1: system Inc;
    inc2: system Inc;
  connections
    data port inc1.output -> inc2.input;
    data port inc2.output -> inc1.input;
end Cyclic.Impl;

system Inc
  features
    input: in data port int;
    output: out data port int;
end Inc;
system implementation Inc.Impl
  flows
    output := input + 1;
end Inc.Impl;
```

# Attribute Grammars (AGs)

- Origininally devised by D. Knuth to define semantics of context-free languages
- Idea: enrich context-free grammar by semantic rules which annotate syntax tree with attribute values
- Attributes attached to nonterminal symbols

  synthesized: bottom-up computation (from the leaves to the root)

  inherited: top-down computation (from the root to the leaves)
- With every production a set of semantic rules is associated
  - define values of inner attributes
    (= synthesized/inherited of LHS/RHS) ...
  - in dependence of outer attributes
    (= synthesized/inherited of RHS/LHS)

# Example: Knuth's Binary Numbers

## Example (Knuth's binary numbers)

| | | |
|---|---|---|
| Numbers | $S \to L$ | $v.0 = v.1$<br>$p.1 = 0$ |
| | $S \to L.L$ | $v.0 = v.1 + v.3$<br>$p.1 = 0$<br>$p.3 = -l.3$ |
| Lists | $L \to B$ | $v.0 = v.1$<br>$l.0 = 1$<br>$p.1 = p.0$ |
| | $L \to LB$ | $v.0 = v.1 + v.2$<br>$l.0 = l.1 + 1$<br>$p.1 = p.0 + 1$<br>$p.2 = p.0$ |
| Bits | $B \to 0$ | $v.0 = 0$ |
| Bits | $B \to 1$ | $v.0 = 2^{p.0}$ |

Synthesized attributes of $S, L, B$: $v$ (value; domain $V_v := \mathbb{Q}$)
of $L$: $l$ (length; domain $V_l := \mathbb{N}$)
Inherited attribute of $L, B$: $p$ (position; domain $V_p := \mathbb{Z}$)

# Example: Knuth's Binary Numbers

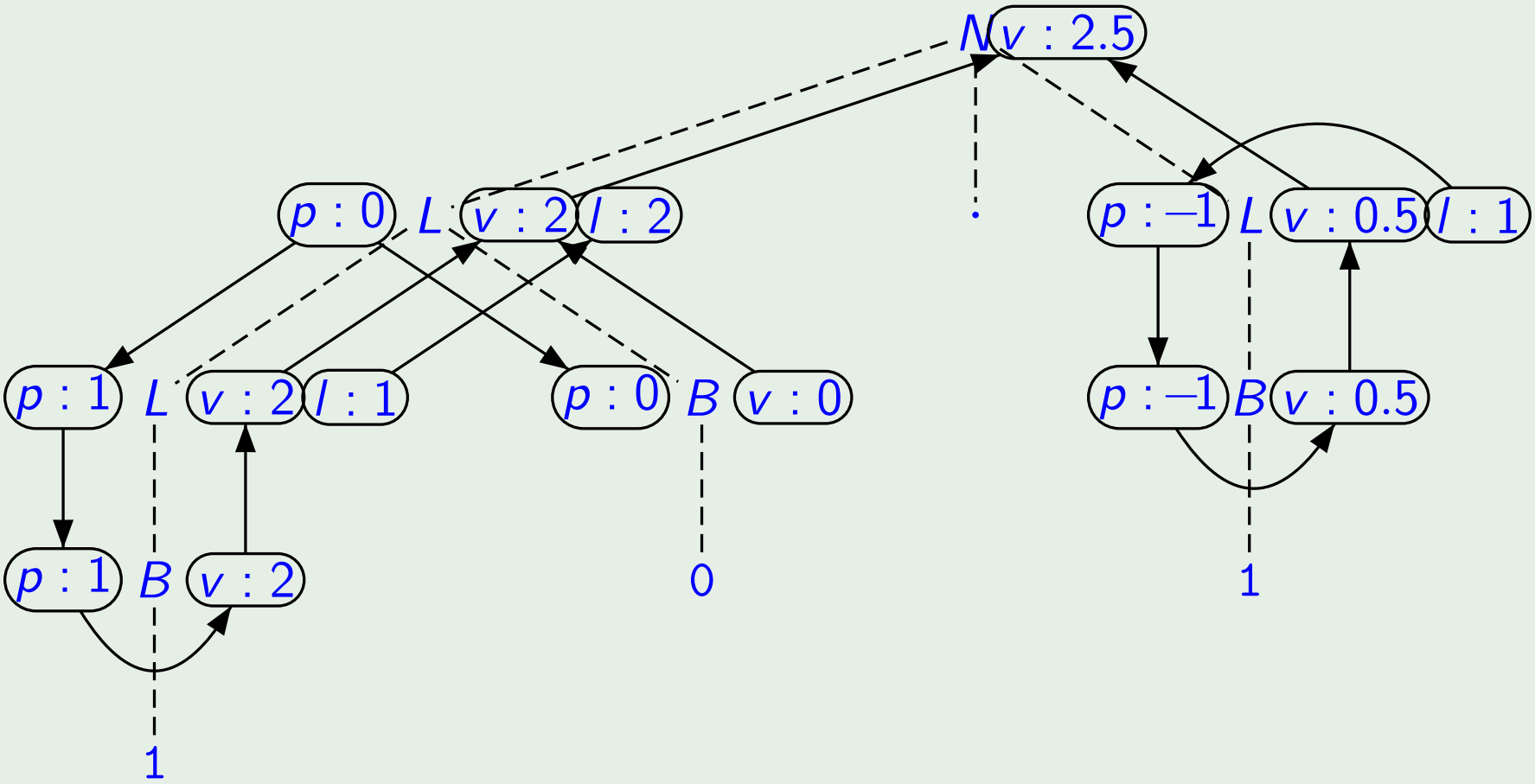
## Example (Knuth's binary numbers)

| | | |
|---|---|---|
| Numbers | $S \to L$ | $v.0 = v.1$ |
| | | $p.1 = 0$ |
| | $S \to L.L$ | $v.0 = v.1 + v.3$ |
| | | $p.1 = 0$ |
| | | $p.3 = -l.3$ |
| Lists | $L \to B$ | $v.0 = v.1$ |
| | | $l.0 = 1$ |
| | | $p.1 = p.0$ |
| | $L \to LB$ | $v.0 = v.1 + v.2$ |
| | | $l.0 = l.1 + 1$ |
| | | $p.1 = p.0 + 1$ |
| | | $p.2 = p.0$ |
| Bits | $B \to 0$ | $v.0 = 0$ |
| Bits | $B \to 1$ | $v.0 = 2^{p.0}$ |

Synthesized attributes of $S, L, B$: $v$ (value; domain $V_v := \mathbb{Q}$)
of $L$: $l$ (length; domain $V_l := \mathbb{N}$)
Inherited attribute of $L, B$: $p$ (position; domain $V_p := \mathbb{Z}$)

# An Attributed Derivation Tree

## Example (Knuth's binary numbers)



# Circularity of Attribute Grammars

**Again:** unique solvability of equation system required

$\Longrightarrow$ avoid cyclic dependencies

**Definition (Circularity of AGs)**

An attribute grammar is called circular if there exists a syntax tree $t$ such that the attribute equation system of $t$ is recursive (i.e., some attribute variable of $t$ depends on itself). Otherwise it is called noncircular.

# Circularity of Attribute Grammars

**Again:** unique solvability of equation system required

$\Longrightarrow$ avoid cyclic dependencies

## Definition (Circularity of AGs)

An attribute grammar is called circular if there exists a syntax tree $t$ such that the attribute equation system of $t$ is recursive (i.e., some attribute variable of $t$ depends on itself). Otherwise it is called noncircular.

# Decidability of Circularity

## Definition (Attribute dependence)

Given: AG with underlying CFG $G = \langle N, \Sigma, P, S \rangle$.

- If $t$ is a syntax tree with root label $A \in N$ and root node $k$, $\alpha \in \mathrm{syn}(A)$, and $\beta \in \mathrm{inh}(A)$ such that $\beta.k \longrightarrow_t^+ \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \overset{A}{\hookrightarrow} \alpha$)
- For every syntax tree $t$ with root label $A \in N$,
$$\mathrm{is}(A, t) := \{(\beta, \alpha) \in \mathrm{inh}(A) \times \mathrm{syn}(A) \mid \beta \overset{A}{\hookrightarrow} \alpha \text{ in } t\}$$
- For every $A \in N$,
$$IS(A) := \{\mathrm{is}(A, t) \mid t \text{ syntax tree with root label A}\} \subseteq 2^{Inh \times Syn}$$

## Example (Knuth's binary numbers)

- $\mathrm{is}(L, L \Rightarrow B \Rightarrow 0) = \emptyset$
- $\mathrm{is}(L, L \Rightarrow B \Rightarrow 1) = \{(p, v)\}$
- $I$ never dependent on any inherited attribute

$\Rightarrow IS(L) = \{\emptyset, \{(p, v)\}\}$

# Decidability of Circularity

## Definition (Attribute dependence)

Given: AG with underlying CFG $G = \langle N, \Sigma, P, S \rangle$.

- If $t$ is a syntax tree with root label $A \in N$ and root node $k$, $\alpha \in \mathrm{syn}(A)$, and $\beta \in \mathrm{inh}(A)$ such that $\beta.k \longrightarrow_t^+ \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \overset{A}{\hookrightarrow} \alpha$)
- For every syntax tree $t$ with root label $A \in N$,
$$\mathrm{is}(A, t) := \{(\beta, \alpha) \in \mathrm{inh}(A) \times \mathrm{syn}(A) \mid \beta \overset{A}{\hookrightarrow} \alpha \text{ in } t\}$$
- For every $A \in N$,
$$IS(A) := \{\mathrm{is}(A, t) \mid t \text{ syntax tree with root label A}\} \subseteq 2^{Inh \times Syn}$$

## Example (Knuth's binary numbers)

- $\mathrm{is}(L, L \Rightarrow B \Rightarrow 0) = \emptyset$
- $\mathrm{is}(L, L \Rightarrow B \Rightarrow 1) = \{(p, v)\}$
- $I$ never dependent on any inherited attribute

$\Rightarrow$ $IS(L) = \{\emptyset, \{(p, v)\}\}$

# The Circularity Test

## Algorithm (Circularity test)

1. *Iterative computation of IS sets*
2. *AG circular iff exists* $\pi = A_0 \to A_1 \ldots A_n \in P$ *and* $D_i \in IS(A_i)$ $(\forall i \in [n])$ *such that* $D(\pi) \cup \bigcup_{i=1}^{n} D_i$ *has a cycle*

## Theorem

*The time complexity of the circularity test is exponential in the size of the attribute grammar (= maximal length of right-hand sides of productions).*

## Proof.

by reduction of the word problem of alternating Turing machines (see M. Jazayeri: *A Simpler Construction for Showing the Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars*, Comm. of the ACM 28(4), 1981, pp. 715–720) □

# The Circularity Test

## Algorithm (Circularity test)

1. *Iterative computation of IS sets*
2. *AG circular iff exists* $\pi = A_0 \to A_1 \ldots A_n \in P$ *and* $D_i \in IS(A_i)$ $(\forall i \in [n])$ *such that* $D(\pi) \cup \bigcup_{i=1}^{n} D_i$ *has a cycle*

## Theorem

*The time complexity of the circularity test is exponential in the size of the attribute grammar (= maximal length of right-hand sides of productions).*

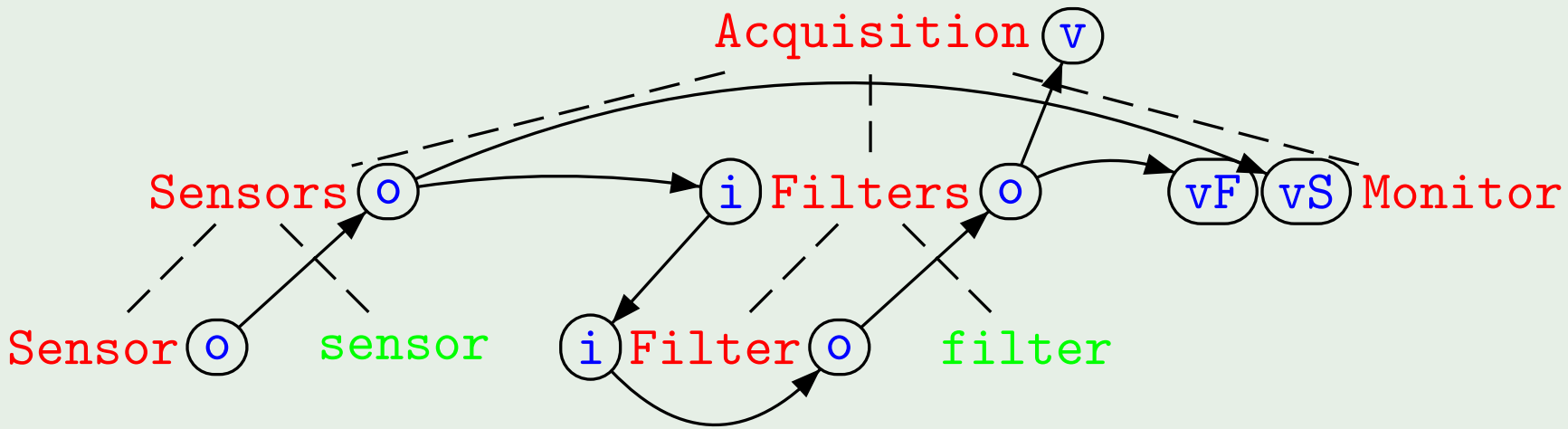
## Proof.

by reduction of the word problem of alternating Turing machines (see M. Jazayeri: *A Simpler Construction for Showing the Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars*, Comm. of the ACM 28(4), 1981, pp. 715–720) □

# AADL Specifications vs. Attribute Grammars

| AADL | Attribute grammars |
|---|---|
| System configuration | Derivation tree |
| Mode | Production |
| Active component | Nonterminal symbol |
| Inactive component | Terminal symbol |
| Incoming data port | Inherited attribute |
| Outgoing data port | Synthesized attribute |
| Flow/data port connection | Semantic rule |

## Example (Data acquisition system)



# From AADL to AGs – Context-Free Part

## Definition (Context-free grammar of AADL specification)

$G = \langle N, \Sigma, P, S \rangle$ is given as follows:

- $N := Cmp$
- $\Sigma := Cmp^{\dagger}$ ($c^{\dagger}$ denotes inactive component)
- $P := \{\pi_{c,m} \mid c \in Cmp, m \in Mod(c)\}$ where
  - $\pi_{c,m} := c \to c'_1 \dots c'_n$ for
  - $\bigcup_{m \in Mod(c)} Cmp(c,m) = \{c_1, \dots, c_n\}$ and
  - $c'_i := \begin{cases} c_i & \text{if } c_i \in Cmp(c,m) \\ c_i^{\dagger} & \text{otherwise} \end{cases}$
- $S := \texttt{main}$

# Data Acquisition System – Context-Free Part

## Example

$$
\begin{aligned}
\texttt{Acquisition} &\rightarrow \texttt{Sensors Filters Monitor} \\
\texttt{Sensors} &\rightarrow \texttt{Sensor sensor} \\
\texttt{Sensors} &\rightarrow \texttt{sensor Sensor} \\
\texttt{Sensor} &\rightarrow \varepsilon \\
\texttt{Filters} &\rightarrow \texttt{Filter filter} \\
\texttt{Filters} &\rightarrow \texttt{filter Filter} \\
\texttt{Filter} &\rightarrow \varepsilon \\
\texttt{Monitor} &\rightarrow \varepsilon
\end{aligned}
$$

# From AADL to AGs – Attribution Part

## Definition (Attribution scheme of AADL specification)

- $Inh := \bigcup_{c \in Cmp} IPrt(c)$
- $Syn := \bigcup_{c \in Cmp} OPrt(c)$
- For every $c \in Cmp$:
  - $\mathrm{inh}(c) := IPrt(c)$
  - $\mathrm{syn}(c) := OPrt(c)$
  - For every $m \in Mod(c)$:

$$E_\pi := \{q.j = p.i \mid (p.i, q.j) \in Con(c, m)\} \cup \{q.0 = e[p \mapsto p.0; p \in IPrt(c)] \mid (e, q) \in Flw(c, m)\}.$$

# Data Acquisition System – Attribution Part

## Example

$$
\begin{array}{lll}
\texttt{Acquisition} \rightarrow \texttt{Sensors Filters Monitor}: & \texttt{input.2} = \texttt{output.1} \\
 & \texttt{valueS.3} = \texttt{output.1} \\
 & \texttt{valueF.3} = \texttt{output.2} \\
 & \texttt{value.0} = \texttt{output.2} \\
\texttt{Sensors} \rightarrow \texttt{Sensor sensor}: & \texttt{output.0} = \texttt{output.1} \\
\texttt{Sensors} \rightarrow \texttt{sensor Sensor}: & \texttt{output.0} = \texttt{output.2} \\
\texttt{Sensor} \rightarrow \varepsilon & \\
\texttt{Filters} \rightarrow \texttt{Filter filter}: & \texttt{input.1} = \texttt{input.0} \\
 & \texttt{output.0} = \texttt{output.1} \\
\texttt{Filters} \rightarrow \texttt{filter Filter}: & \texttt{input.2} = \texttt{input.0} \\
 & \texttt{output.0} = \texttt{output.2} \\
\texttt{Filter} \rightarrow \varepsilon: & \texttt{output.0} = 2.0 * \texttt{input.0} \\
\texttt{Monitor} \rightarrow \varepsilon &
\end{array}
$$

# Analyzing AADL Specifications Using AGs

**Theorem**

*An AADL specification is circular iff the corresponding attribute grammar is circular.*

# Implementation in the COMPASS Toolset

- Problem: exponential complexity of circularity test
- Solution: switch to strong noncircularity

## Definition (Attribute dependence (modified))

Given: AG with underlying CFG $G = \langle N, \Sigma, P, S \rangle$.

- Reminder: if $t$ is a syntax tree with root label $A \in N$ and root node $k$, $\alpha \in \mathrm{syn}(A)$, and $\beta \in \mathrm{inh}(A)$ such that $\beta.k \longrightarrow_t^+ \alpha.k$, then $\alpha$ is dependent on $\beta$ below $A$ in $t$ (notation: $\beta \overset{A}{\hookrightarrow} \alpha$).
- For every $A \in N$,

$$IS'(A) := \{(\beta, \alpha) \mid \beta \overset{A}{\hookrightarrow} \alpha \text{ in some syntax tree with root label A}\} \subseteq Inh \times Syn$$

- Implemented in COMPASS Toolset

# Strong vs. Weak Noncircularity

A (weakly) noncircular, but not strongly nonircular AADL specification:

```
system Super
  features
    value: out data port real;
end Super;
system implementation Super.Impl
  subcomponents
    sub1, sub2: system Sub;
  connections
    data port sub1.out1 -> sub2.in2, sub1.out2 -> sub2.in1;
    data port sub2.out1 -> sub1.in1, sub2.out2 -> sub1.in2;
end Super.Impl;
system Sub
  features
    in1, in2: in data port int;
    out1, out2: out data port int;
end Sub;
system implementation Sub.Impl
  flows
    out1 := in2 in modes (m0); out1 := 1 in modes (m1);
    out2 := 2 in modes (m0); out2 := in1 in modes (m1);
  modes
    m0: initial mode;
    m1: mode;
  transitions ...
end Sub.Impl;
```

# A Strong Cycle