

**Computational Science and Engineering Seminar**

McGill, Montréal

21 September, 2001

# **Object-Oriented Modelling and Simulation of Physical Systems**

**Hans Vangheluwe**



School of Computer Science

McGill University

Montréal, Canada

# Presentation Overview

1. WEST and Modelica
2. A simple Modelica model
3. Object-Orientation (Software vs. Dynamical systems)
  - Encapsulation, objects, classes, . . .
  - Types (Software vs. Dynamical systems)
    - Subtypes
    - Contravariance
    - Semantics of composition
  - Inheritance
  - Different levels of abstraction: state automata

4. Model Base development procedure
5. Non-causal modelling
  - Why ?
  - Causality assignment
  - Sorting algebraic equations
  - Target: numerical simulator
6. Demo with Dynasim's Dymola
7. The future: multi-formalism, multi-abstraction, meta-modelling

## Based on . . .

- WEST (bioactivated sludge waste water treatment)

`www.hemmiswest.com`

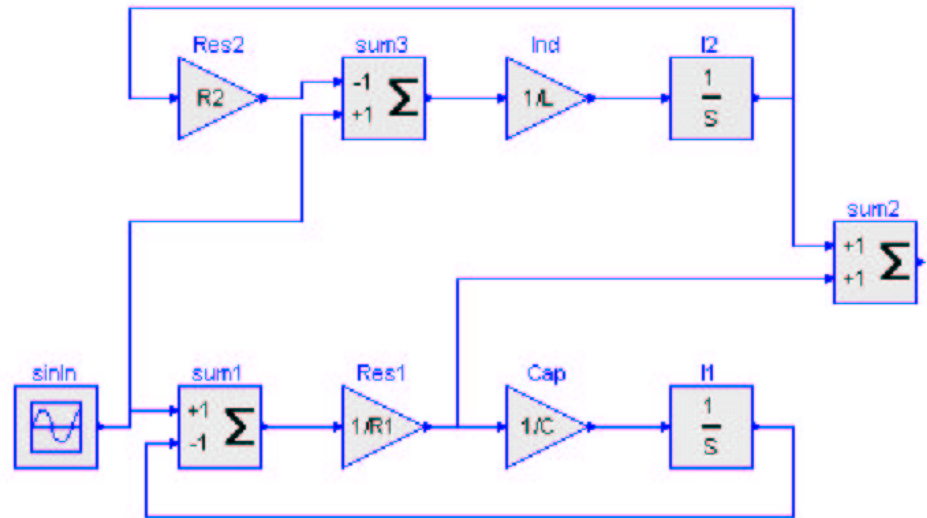
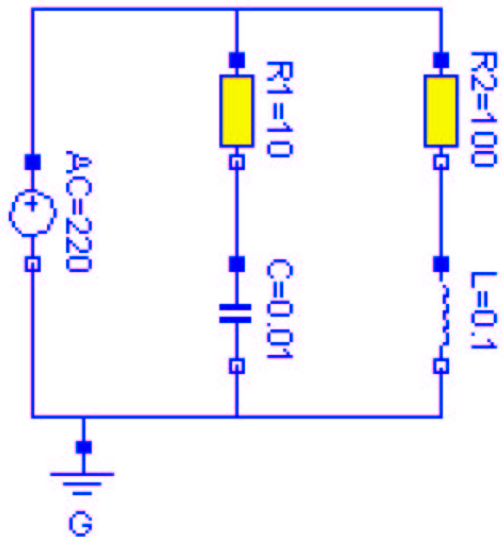
- Modelica (ESPRIT Basic Research, now Association)

`www.modelica.org`

### Aims:

- standard language for model exchange and re-use
- Support non-causal, hybrid, hierarchical modelling
- Semantics based on Hybrid DAEs
- Separate model from its numerical solution
- Library of basic models

# Electrical example: Modelica vs. Matlab/Simulink



# Electrical Types

```
type Time = Real (final quantity="Time", final unit="s");
type ElectricPotential = Real (final quantity="ElectricPotential",
                               final unit="V");
type Voltage = ElectricPotential;
type ElectricCurrent = Real (final quantity="ElectricCurrent",
                             final unit="A");
type Current = ElectricCurrent;
```

# Electrical Pin Interface

```
connector PositivePin "Positive pin of an electric component"  
    Voltage v "Potential at the pin";  
    flow Current i "Current flowing into the pin";  
end PositivePin;
```

# Electrical Port

```
partial model OnePort
  "Component with two electrical pins p and n
  and current i from p to n"
  Voltage v "Voltage drop between the two pins (= p.v - n.v)";
  Current i "Current flowing from pin p to pin n";
  PositivePin p;
  NegativePin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

# Electrical Resistor

```
model Resistor "Ideal linear electrical resistor"  
  extends OnePort;  
  parameter Resistance R=1 "Resistance";  
  equation  
    R*i = v;  
end Resistor;
```

# The circuit

```
model circuit
  Resistor R1(R=10);
  Capacitor C(C=0.01);
  Resistor R2(R=100);
  Inductor L(L=0.1);
  VsourceAC AC;
  Ground G;
equation
  connect(AC.p, R1.p);
  connect(R1.n, C.p);
  connect(C.n, AC.n);
  connect(R1.p, R2.p);
  connect(R2.n, L.p);
  connect(L.n, C.n);
  connect(AC.n, G.p);
end circuit;
```

# Object-Oriented Software

- Started in simulation (Simula).  
Beware: Discrete Event Simulation !
- Encapsulation
- Interfaces

- **Classes/Objects**

- attributes (data)
- methods (behaviour)

```
Class Value:
```

```
double value = 10  
double get_value()  
void set_value(double)
```

```
value1 = Value()  
value2 = Value()  
value1.set_value(value2.get_value())
```

- **Inheritance: extending and specializing**
- **Composition**

# OO for Modelling and Simulation

- Dynamic behaviour is a function of time !
- Variables are functions: `real v` means

$$v : T \rightarrow \mathbb{R}$$

- Assume “lumped models” (for composition)
- Assume formalism used: Hybrid DAEs

# Signals vs. Values

# Types in Software

`typeof(a) = A`

`typeof(b) = B`

- Type = *set* of values a variable may take
- Check type compatibility:  
 $a = b$  is allowed if  $B <: A$
- Determine type of operators:  
 $a + b \Rightarrow + : A \times B \rightarrow C$
- Classification

# Types in Physical Systems

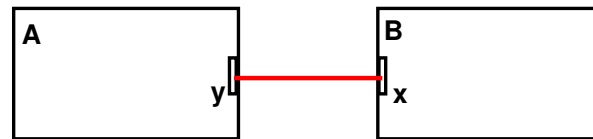
- Quantities (Length, Temperature, ...)  
⇒ dimensional analysis
- Units (m, V, ...)  
⇒ re-write equations:  
 $l1 [m] + l2 [km] \rightarrow l1 [m] + 1000 * l2m [m]$
- Variables are not values but functions/signals !

# Subtypes

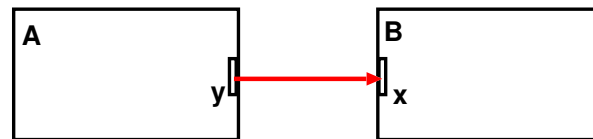
- $a + b \Rightarrow a' + b$  is allowed  
Condition:  $A' <: A$
- $b = f(a)$  and  $\text{typeof}(f') <: \text{typeof}(f) \Rightarrow b = f'(a)$  is allowed. Conditions:
  - $f: A \rightarrow B$
  - $f': A' \rightarrow B'$
  - $B' <: B$
  - $A <: A'$  : contravariance

# Semantics of Coupling/Composition: Block Diagram

non-causal



causal



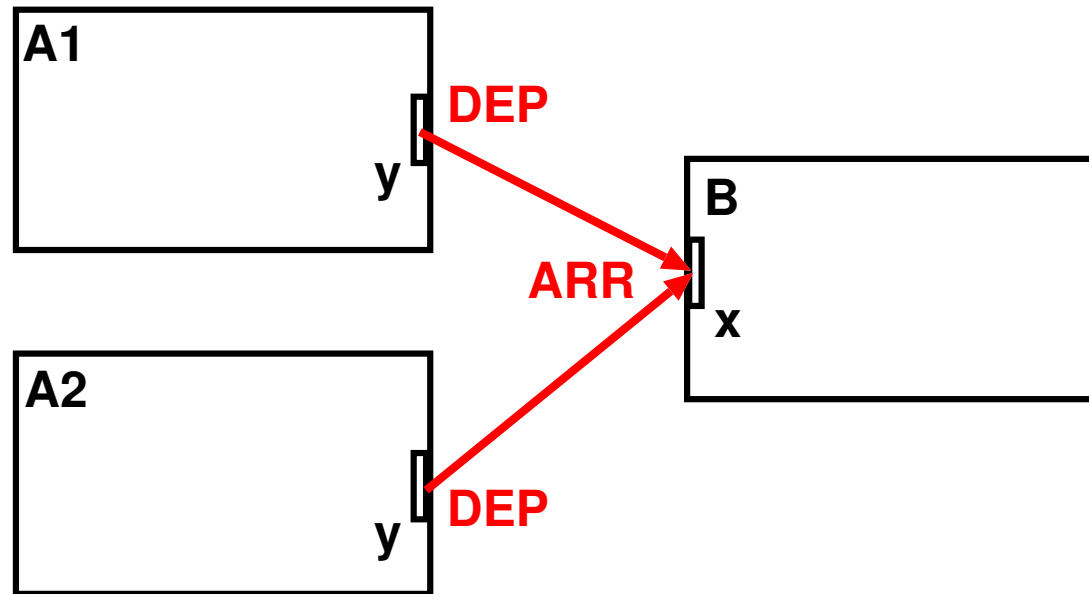
Non-Causal:

$$A.y = B.x$$

Causal:

$$B.x := A.y$$

# Semantics Coupling/Composition: Discrete Event

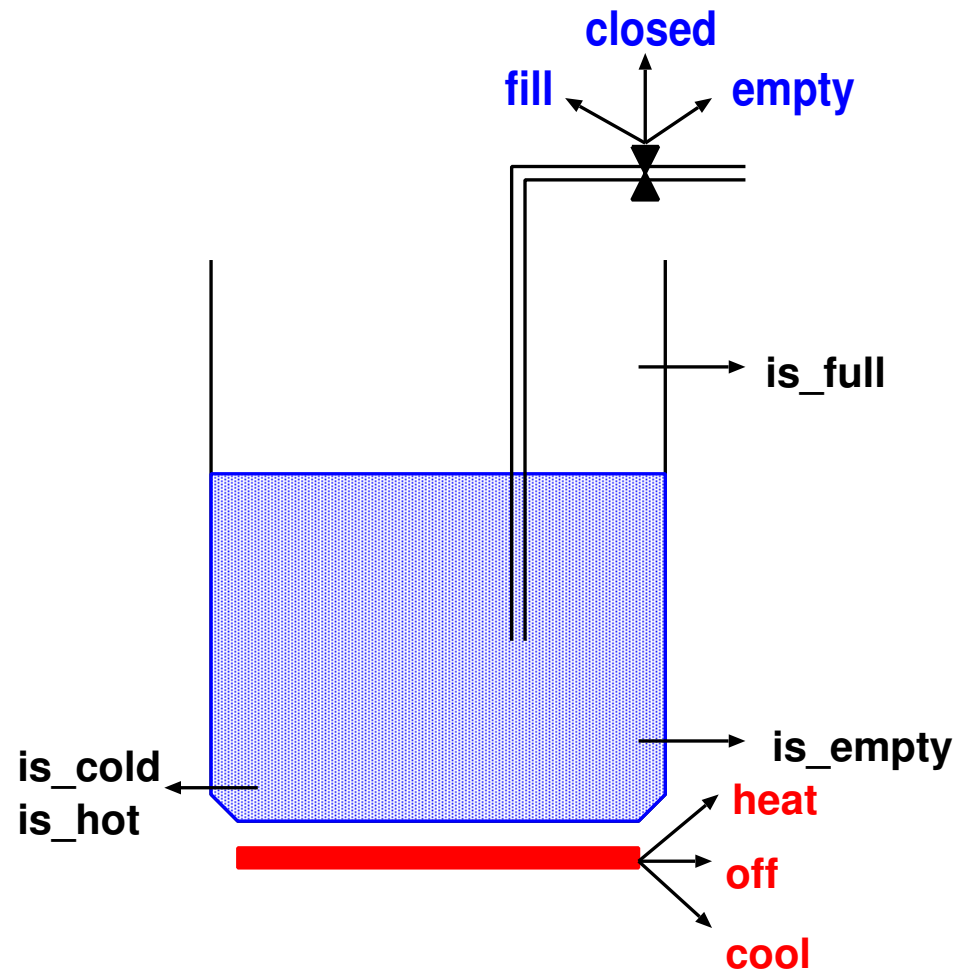


- *schedule* ARRivals
- *resolve* collisions

# Subtypes in Models of Dynamical Systems

- Don't forget time-base and contravariant relationship !
- More general: sub-type = behaviour sub-space !
- How to reason about behaviour sub-spaces ?

# System under study: $T, l$ controlled liquid



# Detailed (continuous) view, ALG + ODE formalism

Inputs (discontinuous  $\rightarrow$  hybrid model):

- Emptying, filling flow rate  $\phi$
- Rate of adding/removing heat  $W$

Parameters:

- Cross-section surface of vessel  $A$
- Specific heat of liquid  $c$
- Density of liquid  $\rho$

State variables:

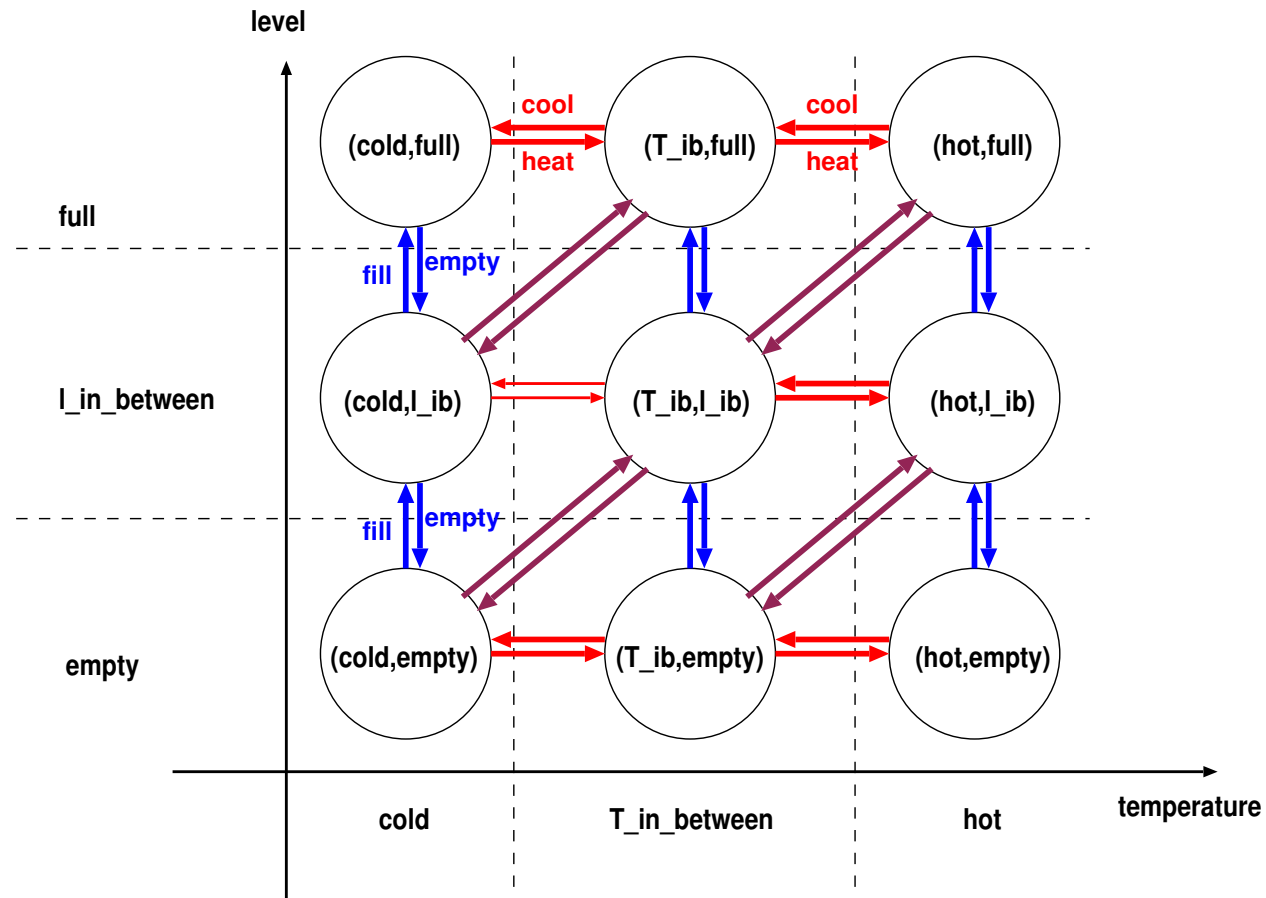
- Temperature  $T$
- Level of liquid  $l$

Outputs (sensors):

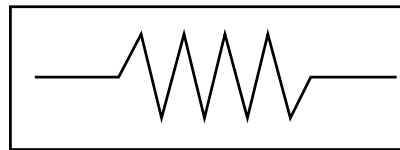
- $is\_low, is\_high, is\_cold, is\_hot$

$$\left\{ \begin{array}{l} \frac{dT}{dt} = \frac{1}{l} \left[ \frac{W}{c\rho A} - \phi T \right] \\ \frac{dl}{dt} = \phi \\ is\_low = (l < l_{low}) \\ is\_high = (l > l_{high}) \\ is\_cold = (T < T_{cold}) \\ is\_hot = (T > T_{hot}) \end{array} \right.$$

# High-level (discrete) view, FSA formalism

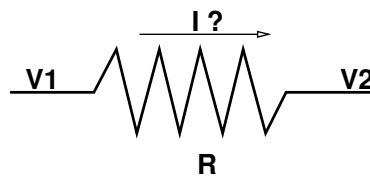


# Object-oriented re-use and causality

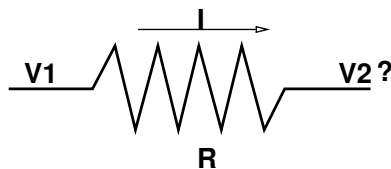


Object "resistor"

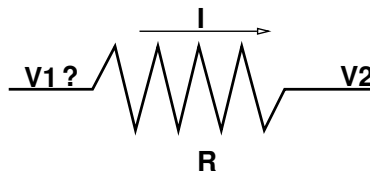
$$V1 - V2 = R * I$$



$$I = (V1 - V2) / R$$



$$V2 = V1 - R * I$$

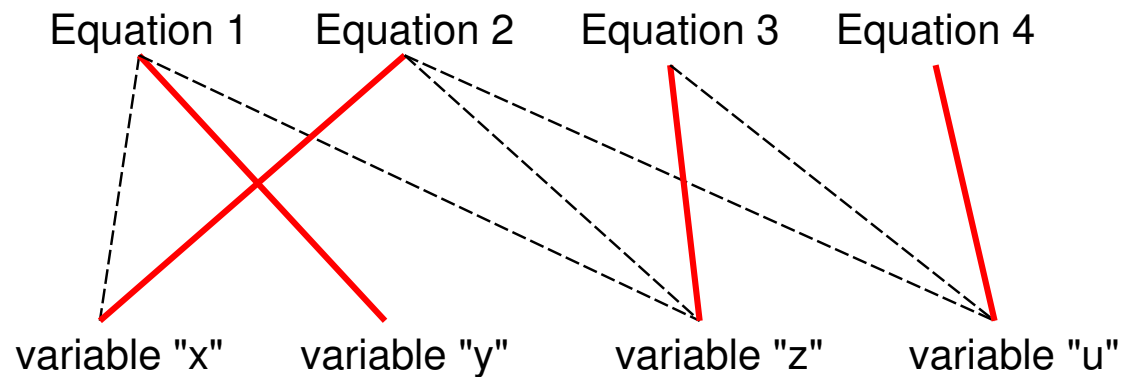
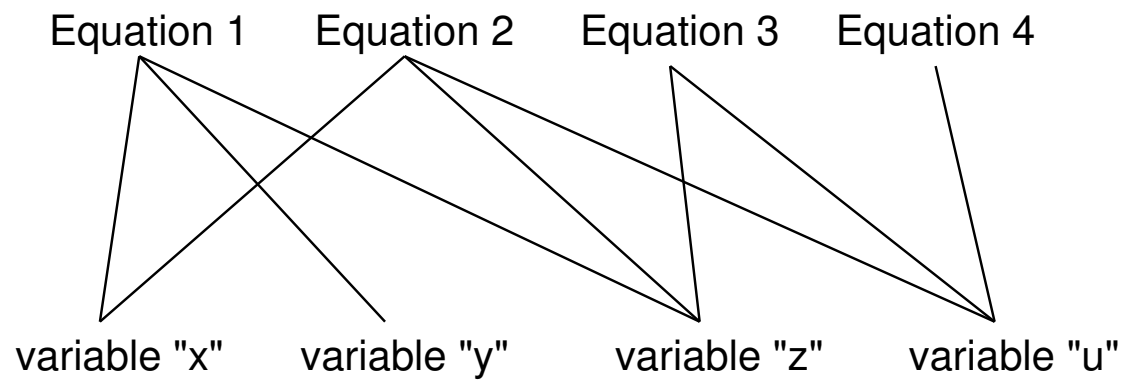


$$V1 = V2 + R * I$$

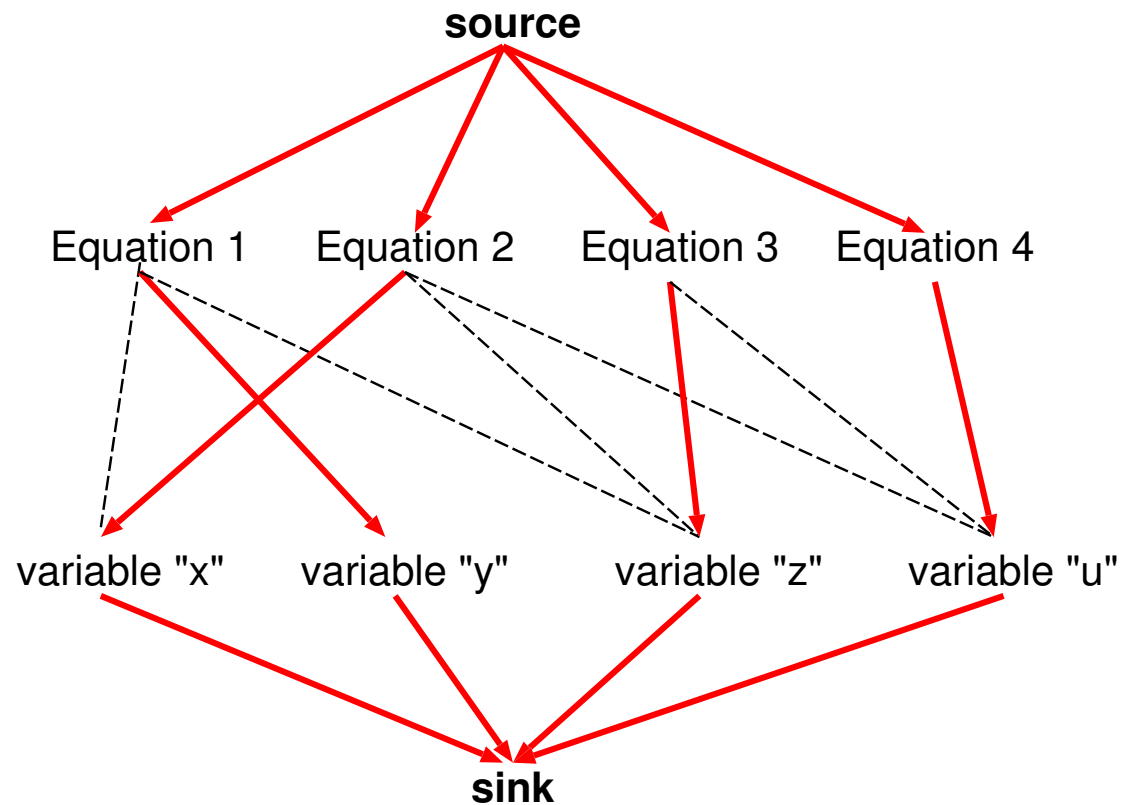
# Non-causal model (*e.g.*, from physical conservation laws)

$$\left\{ \begin{array}{l} x + y + z = 0 \quad \text{Equation 1} \\ x + 3z + u^2 = 0 \quad \text{Equation 2} \\ z - u - 16 = 0 \quad \text{Equation 3} \\ u - 5 = 0 \quad \text{Equation 4} \end{array} \right.$$

# Causality assignment: bipartite graph, maximum cardinality matching



# Causality assignment: network flow



+ weights for "bad inverses"

## Causality assigned

$$\left\{ \begin{array}{l} \underline{x} + \underline{y} + z = 0 \quad \text{Equation 1} \\ \underline{x} + 3z + u^2 = 0 \quad \text{Equation 2} \\ \underline{z} - u - 16 = 0 \quad \text{Equation 3} \\ \underline{u} - 5 = 0 \quad \text{Equation 4} \end{array} \right.$$

re-write in causal form

$$\left\{ \begin{array}{l} \underline{y} = -x - z \\ \underline{x} = -3z - u^2 \\ \underline{z} = u + 16 \\ \underline{u} = 5 \end{array} \right.$$

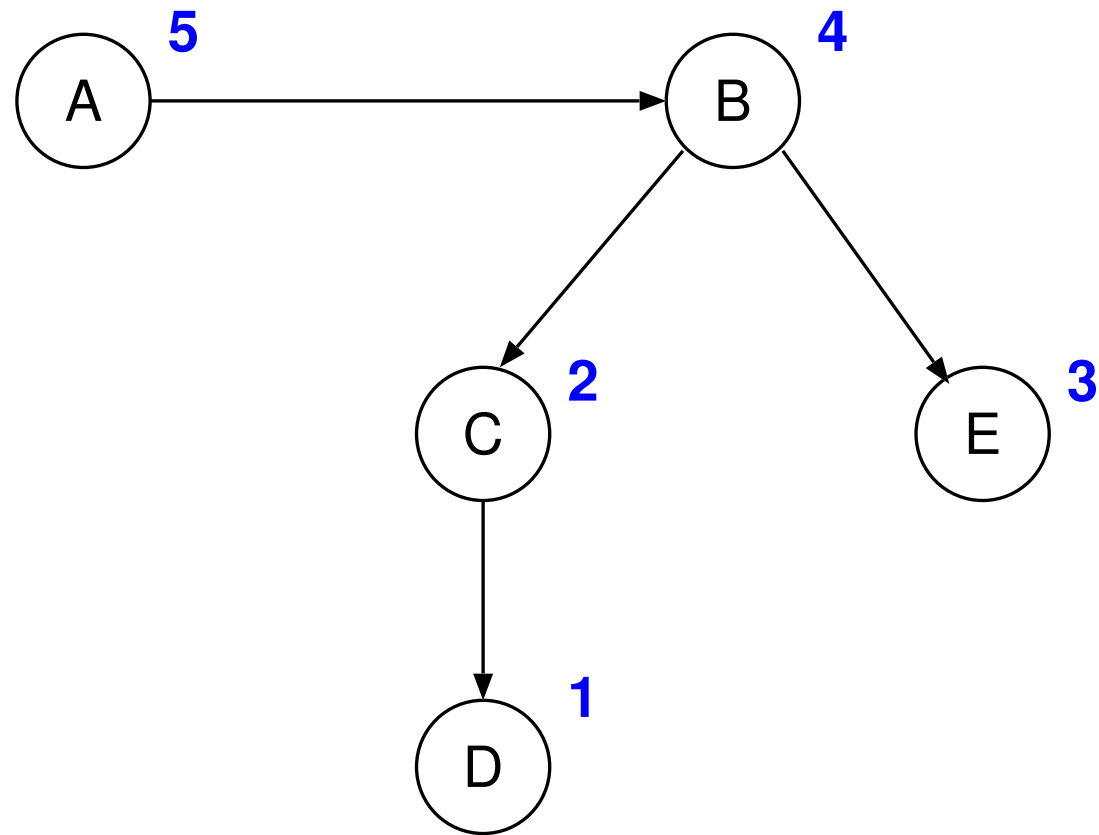
## Set of Algebraic Eqns (no cyclic dependencies)

$$\left\{ \begin{array}{l} a = b^2 + 3 \\ b = \sin(c \times e) \\ c = \sqrt{d - 4.5} \\ d = \pi/2 \\ e = u() \end{array} \right.$$

**WRONG:**

$$\left[ \begin{array}{l} a = b^2 + 3 = 3 \\ b = \sin(c \times e) = 0 \\ c = \sqrt{d - 4.5} = \text{error} \\ d = \pi/2 \\ e = u() \end{array} \right.$$

# Sorting (no cyclic dependencies) DFS, postorder numbering of dependency graph



# Sorting Result

$$\left[ \begin{array}{l} d = \pi/2 \\ e = u() \\ c = \sqrt{d - 4.5} \\ b = \sin(c \times e) \\ a = b^2 + 3 \end{array} \right.$$

## Dependency Cycle (aka Algebraic Loop)

$$\begin{cases} x = y + 16 \\ y = -x - z \\ z = 5 \end{cases}$$

Can *never* be sorted

due to a dependency *cycle* aka *strong component*

(every vertex in the component is reachable from every other)

$$x \rightarrow y \rightarrow x$$

May be solved implicitly

$$\begin{cases} z = 5 \\ x - y = -6 \\ x + y = -z \end{cases}$$

Implicit set of  $n$  equations in  $n$  unknowns.

- non-linear  $\rightarrow$  non-linear solver.
- linear  $\rightarrow$  numerical or symbolic solution.

Linear: may be solved symbolically (Cramer)

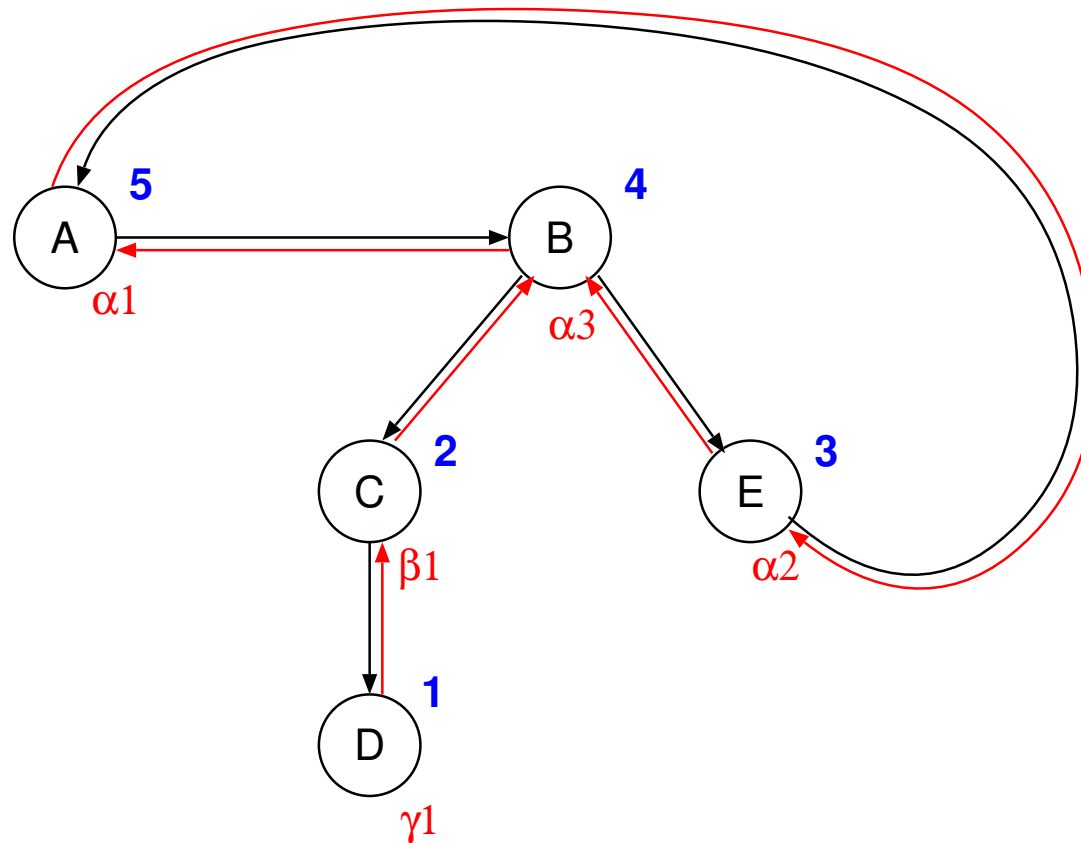
$$x = \frac{\begin{vmatrix} -6 & -1 \\ -z & 1 \end{vmatrix}}{\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}} = \frac{-6-z}{2}; \quad y = \frac{\begin{vmatrix} 1 & -6 \\ 1 & -z \end{vmatrix}}{\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}} = \frac{6-z}{2}$$

$$\begin{cases} z = 5 \\ x = \frac{-6-z}{2} \\ y = \frac{6-z}{2} \end{cases}$$

# Tarjan's algorithm for Cycle Detection

$$\left\{ \begin{array}{l} a = b^2 + 3 \\ b = \sin(c \times e) \\ c = \sqrt{d - 4.5} \\ d = \pi/2 \\ e = a^2 + u() \end{array} \right.$$

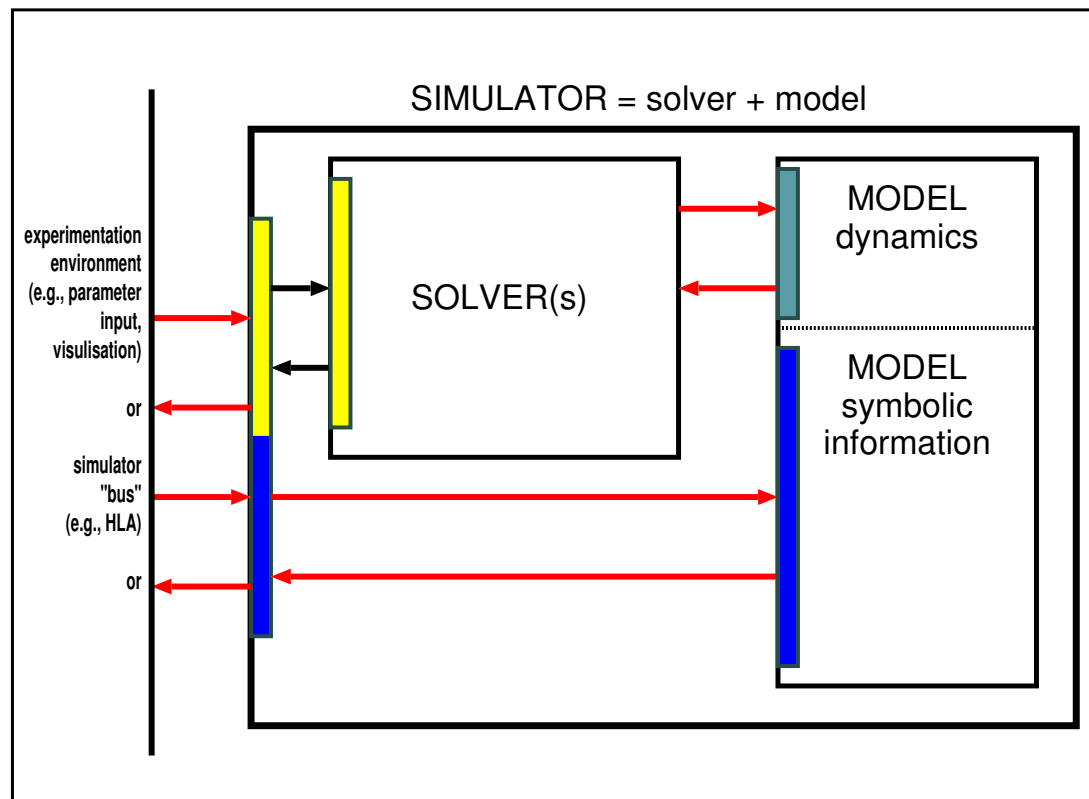
# Algebraic Loop (Cycle) Detection



## Algebraic Loop (Cycle) Detection Result

$$\left[ \begin{array}{l} d = \pi/2 \\ c = \sqrt{d-4.5} \\ \left\{ \begin{array}{l} b = \sin(c \times e) \\ a = b^2 + 3 \\ e = a^2 + u() \end{array} \right. \end{array} \right. ; \left[ \begin{array}{l} d = \pi/2 \\ c = \sqrt{d-4.5} \\ \left\{ \begin{array}{l} b - \sin(c \times e) = 0 \\ a - b^2 - 3 = 0 \\ a^2 - e + u() = 0 \end{array} \right. \end{array} \right.$$

# Target: Model-solver architecture (Modelica - DSblock)



## DSblock structure: readable vectors

```
#define _t_ IndepVarValues[0]
#define _x_out_ OutputVarValues[0]
#define _y_out_ OutputVarValues[1]
#define _x_ DerStateVarValues[0]
#define _y_ DerStateVarValues[1]
#define _D_x_ Derivatives[0]
#define _D_y_ Derivatives[1]
```

## DSblock structure: symbolic information

```
CircleClass :: CircleClass(StringType name_arg)
{
  set_name(name_arg);
  set_description("Circle test.");
  set_class_name("CircleClass");

  set_no_indep_vars(1);
  set_indep_var(0, new MSLEIndepVarClass("t", "s"));

  set_no_output_vars(2);
  set_output_var(0, new MSLEOutputVarClass("x_out", "", 0));
  set_output_var(1, new MSLEOutputVarClass("y_out", "", 0));

  set_no_der_state_vars(2);
  set_der_state_var(0, new MSLEDerStateVarClass("x", "", 0.1));
  set_der_state_var(1, new MSLEDerStateVarClass("y", "", 0.1));
}
```

## DSblock structure: the computation part

```
void CircleClass :: ComputeInitial(void)
{
}
void CircleClass :: ComputeState(void)
{
    _D_x_ = _y_;
    _D_y_ = -_x_;
}
void CircleClass :: ComputeTerminal(void)
{
}
void CircleClass :: ComputeOutput(void)
{
    _x_out_ = _x_;
    _y_out_ = _y_;
}
```

# DSblock advantages

- Readable, without performance compromise
- Contains non-numerical information
- In compiled form: pluggable, difficult to reverse-engineer

# Internal model representation

- Abstract Syntax Tree + Symbol Table

$$b + 2 - (a + 3) = x$$

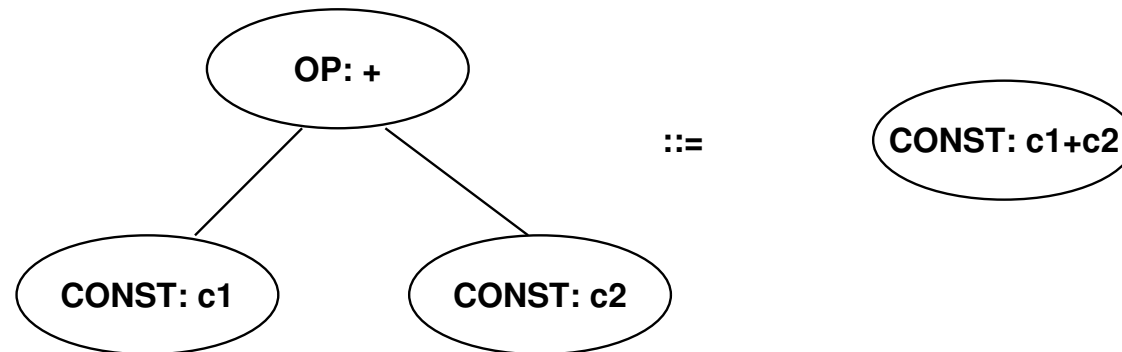
$$= [ + [ b, + [ 2, - [ a, 3 ] ], x ]$$

- From the AST + ST, a dependency graph can be built.
  1. for causality assignment,
  2. for equation re-write,
  3. for loop detection and sorting,
  4. for constant folding,
  5. for parameter expression lifting ( $-K/g$ ),
  6. for output equation selection

# Constant Folding Graph Grammar

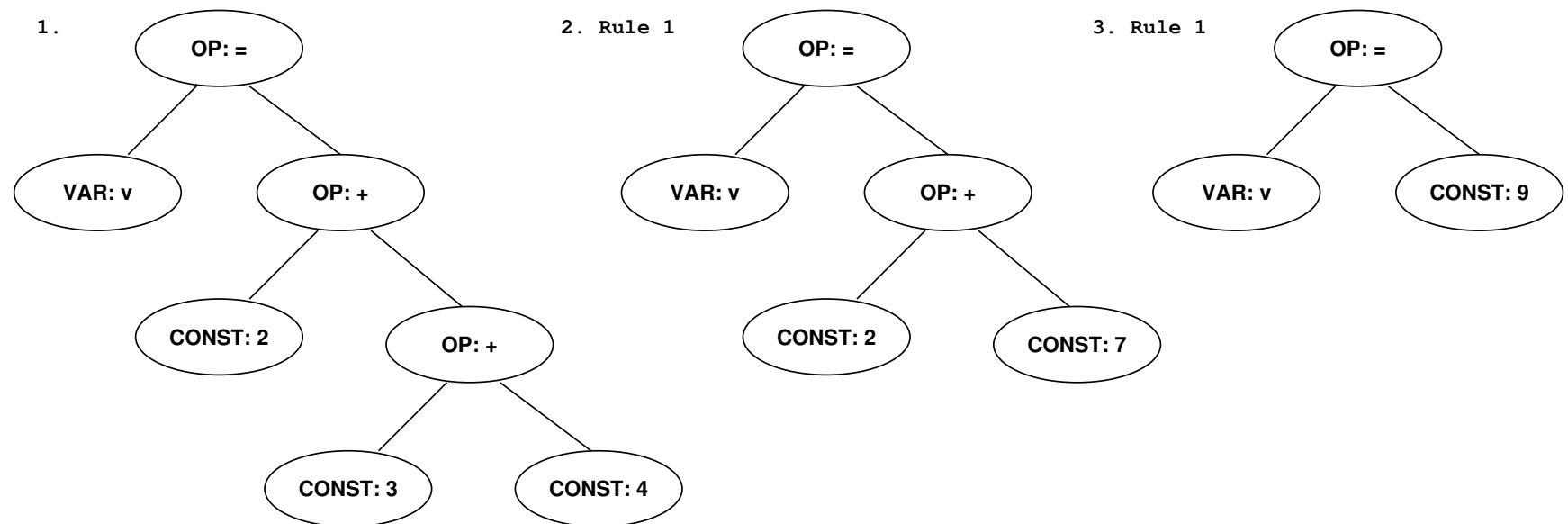
$c1 + c2$

Rule 1.



# Constant Folding Transformation

$$v = 2 + (3 + 4)$$



# Canonical Representation

To encode *associativity* and *commutativity* of operators.

A representation of a mathematical object is *canonical* if two different representations always correspond to two different objects.

1.  $n$ -ary operators
2. *inv* for each operator
3. lexicographic ordering

$$\begin{aligned} &+[2, \text{inv}[3], a, \text{inv}[b]] \\ &+[\text{inv}[1], a, \text{inv}[b]] \end{aligned}$$

$$2 - 3 + a - b \Rightarrow -1 + a - b$$

—→ **symbolic operations** (simplify, analyze, ...)

—→ **re-use AND performance !**

# Polynomials in multiple variables

canonical, natural representation

Different types of ordering:

- *lexicographic*: alphabetically ordered. Within one variable name, ordered by powers. If the powers of that variable are the same, look at the next (lexicographic) variable.  $x^2 + 2xy + x + y^2 + y + 1$
- *total degree, then lexicographic*: lexicographic distinction between same total degree, ordered by total degree.  $x^2 + 2xy + y^2 + x + y + 1$
- *total degree, then inverse lexicographic*:  $y^2 + 2xy + x^2 + y + x + 1$

# Future

- multi-formalism
- multi-abstraction
- meta-modelling