A Modelica Compiler

Steven Xu

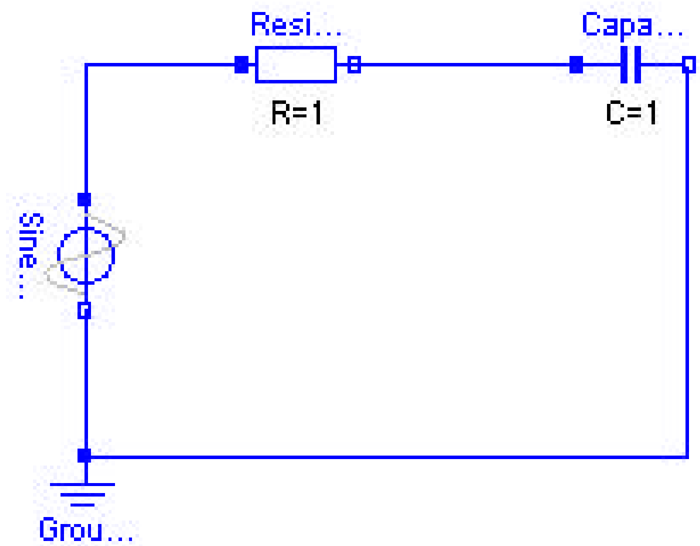
School Of Computer Science
McGill University

March 31, 2003

From last time

- My last presentation has given a brief introduction to Modelica, an object-oriented language for physical system modeling
- Before presenting today’s topic, let’s have a very brief review of modeling in Modelica with a simple circuit

A simple circuit



A simple circuit in Modelica

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

- A connection **connect**(`Pin1, Pin2`), connects the two pins such that they form one node
- This implies two equations:

```
    Pin1.v = Pin2.v
    Pin1.i + Pin2.i = 0
```

A simple circuit in Modelica

- An electrical port

```
partial model OnePort “Superclass of Components
with two electrical pins p and n"
     Voltage v "Voltage drop between p and n";
     Current i "Current flowing from p to n";
     Pin p;
     Pin n;
equation
     v = p.v - n.v;
     0 = p.i + n.i;
     i = p.i;
end OnePort;
```

A simple circuit in Modelica

- Resistor

```
model Resistor "Ideal linear
   electrical resistor"
   extends OnePort;
   parameter Real R(unit=“O”)

equation
   R*i = v;    “Ohm’s Law”
end Resistor;
```

- Capacitor

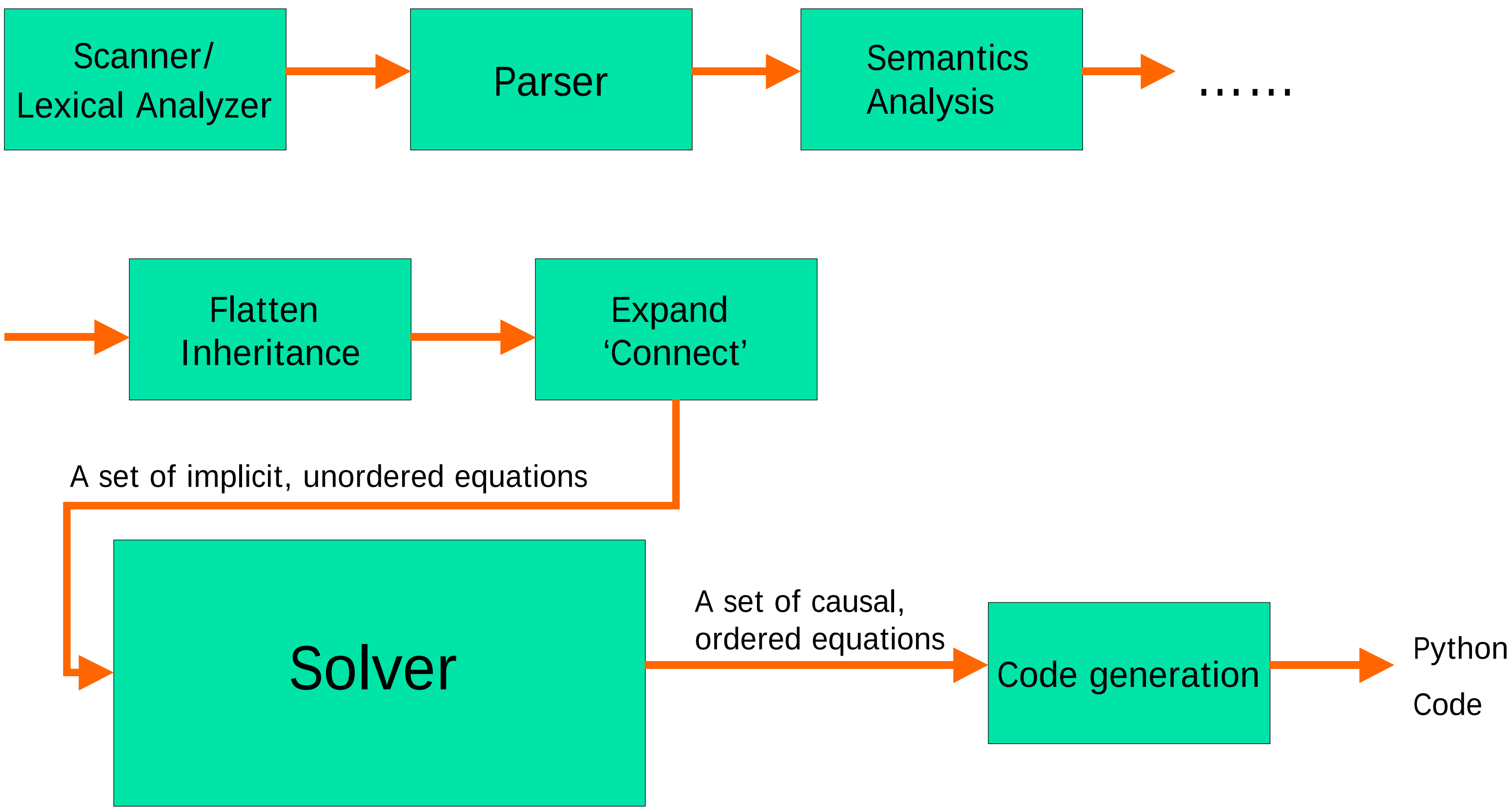
```
model Capacitor "Ideal
   electrical Capacitor"
   extends OnePort;
   parameter real C(unit=“F”)

equation
   C*der(v) = i;
end Capacitor;
```

A simple circuit in Modelica

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

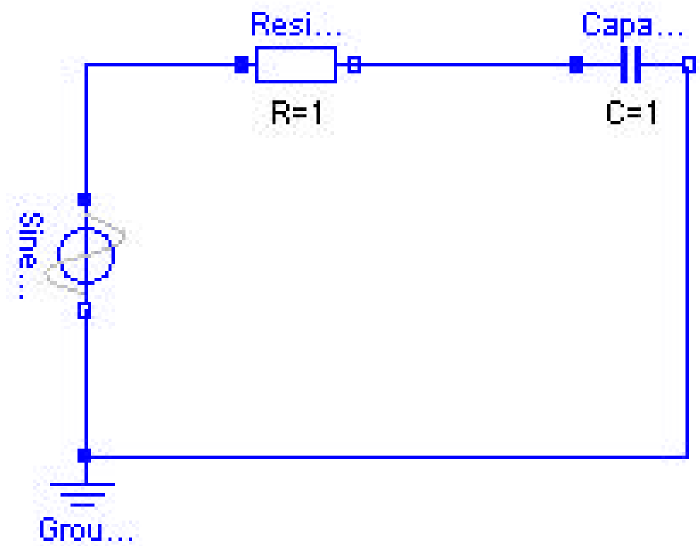
Overview of a Modelica Compiler



Possible Steps of the Solver

- Isolate redundant equations
- Symbolic solution of constant coefficient linear equations
- Detect and correct high index problems
- Causality assignment: Maximum Flow Algorithm
- Sort equations
- Solve linear and non-linear algebraic loops
- Numerical resolution of differential algebraic systems

The Previous Example



Number of equations

- AC Source: 3 + 1
- Resistor: 3 + 1
- Capacitor: 3 + 1
- Ground: 1
- First ‘connect’: 2
- Second ‘connect’: 2
- Third ‘connect’: 2
- Last ‘connect’: 1
- Total number of equations: 20

Equations

- E1: AC.v = AC.p.v – AC.n.v
- E2: AC.p.i + AC.n.i = 0
- E3: AC.i = AC.p.i
- E4: AC.v = sin(t)
- E5: R.v = R.p.v – R.n.v
- E6: R.p.i + R.n.i = 0
- E7: R.i = R.p.i
- E8: R.v = r * R.i
- E9: AC.p.v = R.p.v
- E10: AC.p.i + R.p.i = 0
- E11: C.v = C.p.v –C.n.v
- E12: C.p.i + C.n.i = 0
- E13: C.i = C.p.i
- E14: C.i = C * dev(C.v)
- E15: R.n.v = C.p.v
- E16: R.n.i + C.p.i = 0
- E17: AC.n.v = C.n.v
- E18: AC.n.v = G.p.v
- E19: AC.n.i + C.n.i = 0
- E20: G.p.v = 0

Variables

- V1: AC.v
- V2: AC.p.v
- V3: AC.n.v
- V4: AC.p.i
- V5: AC.n.i
- V6: AC.i
- V7: R.v
- V8: R.p.v
- V9: R.n.v
- V10: R.p.i
- V11: R.n.i
- V12: R.i
- V13: C.v
- V14: C.p.v
- V15: C.n.v
- V16: C.p.i
- V17: C.n.i
- V18: C.i
- V19: dev(C.v)
- V20: G.p.v

Isolate Redundant Equations

- The ‘connect’ statement in Modelica generates many equations in the form of `‘a = b’`, or `‘a + b = 0’`
- If these equations can be isolated before going to the stage causality assignment and sorting, the whole solver will be much more efficient, since causality assignment and sorting equations are time-consuming

Isolate Redundant Equations

- E1: AC.v = AC.p.v – AC.n.v
- E2: AC.p.i + AC.n.i = 0
- E3: AC.i = AC.p.i
- E4: AC.v = sin(t)
- E5: R.v = R.p.v – R.n.v
- E6: R.p.i + R.n.i = 0
- E7: R.i = R.p.i
- E8: R.v = r * R.i
- E9: **AC.p.v = R.p.v**
- E10: AC.p.i + R.p.i = 0
- E11: C.v = C.p.v –C.n.v
- E12: C.p.i + C.n.i = 0
- E13: C.i = C.p.i
- E14: C.i = C * dev(C.v)
- E15: **R.n.v = C.p.v**
- E16: R.n.i + C.p.i = 0
- E17: **AC.n.v = C.n.v**
- E18: **AC.n.v = G.p.v**
- E19: AC.n.i + C.n.i = 0
- E20: G.p.v = 0

Isolate Redundant Equations

- E9: **AC.p.v = R.p.v**
- E15: **R.n.v = C.p.v**
- E17: **AC.n.v = C.n.v**
- E18: **AC.n.v = G.p.v**

- **AC.p.v = R.p.v**
- **R.n.v = C.p.v**
- **AC.n.v = C.n.v = G.p.v**

The LHS can be substituted by the RHS in the remaining equation set, e.g. replace AC.p.v with R.p.v, equation

AC.v = AC.p.v – AC.n.v

becomes

AC.v = R.p.v – G.p.v

Isolate Redundant Equations

- E1: AC.v = AC.p.v – AC.n.v
- E2: **AC.p.i + AC.n.i = 0**
- E3: **AC.i = AC.p.i**
- E4: AC.v = sin(t)
- E5: R.v = R.p.v – R.n.v
- E6: **R.p.i + R.n.i = 0**
- E7: **R.i = R.p.i**
- E8: R.v = r * R.i
- E9: AC.p.v = R.p.v
- E10: **AC.p.i + R.p.i = 0**
- E11: C.v = C.p.v –C.n.v
- E12: **C.p.i + C.n.i = 0**
- E13: **C.i = C.p.i**
- E14: C.i = C * dev(C.v)
- E15: R.n.v = C.p.v
- E16: **R.n.i + C.p.i = 0**
- E17: AC.n.v = C.n.v
- E18: AC.n.v = G.p.v
- E19: **AC.n.i + C.n.i = 0**
- E20: G.p.v = 0

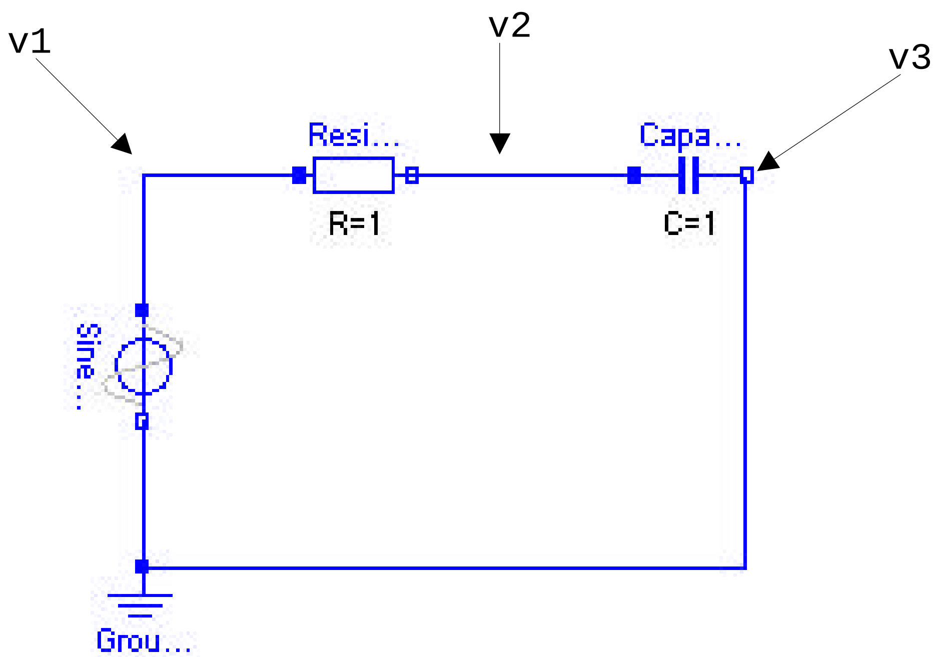
Isolate Redundant Equations

- E2: **AC.p.i + AC.n.i = 0**
- E3: **AC.i = AC.p.I**
- E6: **R.p.i + R.n.i = 0**
- E7: **R.i = R.p.I**
- E10: **AC.p.i + R.p.i = 0**
- E12: **C.p.i + C.n.i = 0**
- E13: **C.i = C.p.I**
- E16: **R.n.i + C.p.i = 0**
- E19: **AC.n.i + C.n.i = 0**

→

R.i = R.p.i
= - R.n.i
= C.i
= C.p.i
= - C.n.i
= - AC.i
= - AC.p.i
= AC.n.i

Isolate Redundant Equations



Reduced Equation Set

Equations

- `E1: Ac.v = v1 – v3`
- `E2: AC.v = sin(t)`
- `E3: R.v = v1 – v2`
- `E4: R.v = r * R.i`
- `E5: C.v = v2 – v3`
- `E6:` **`R.i = C * dev(C.v)`**
- `E7: v3 = 0`

Variables

- `V1: v1 (AC.p.v)`
- `V2: v2 (C.p.v)`
- `V3: v3 (C.n.v)`
- `V4: AC.v`
- `V5: R.v`
- `V6:` **`C.v`**
- `V7: R.i`

Differential Causality

- `E6: R.i = C * dev(C.v)`
- `dev(C.v) = R.i/C`
- `dev(C.v) = d(C.v)/dt`
- `C.v(t) = C.v(t-△t)] + △ t * R.i(t-△t)/C`
- Since new values of the LHS (C.v at time step t) are calculated based on old values (R.i at time step t-1), the order of evaluation of derivative equations does not matter

Reduced Equation Set

Equations

- E1: Ac.v = v1 – v3
- E2: AC.v = sin(t)
- E3: R.v = v1 – v2
- E4: R.v = r * R.i
- E5: C.v = v2 – v3
- E6: v3 = 0

Variables

- V1: v1
- V2: v2
- V3: v3
- V4: AC.v
- V5: R.v
- V6: R.i

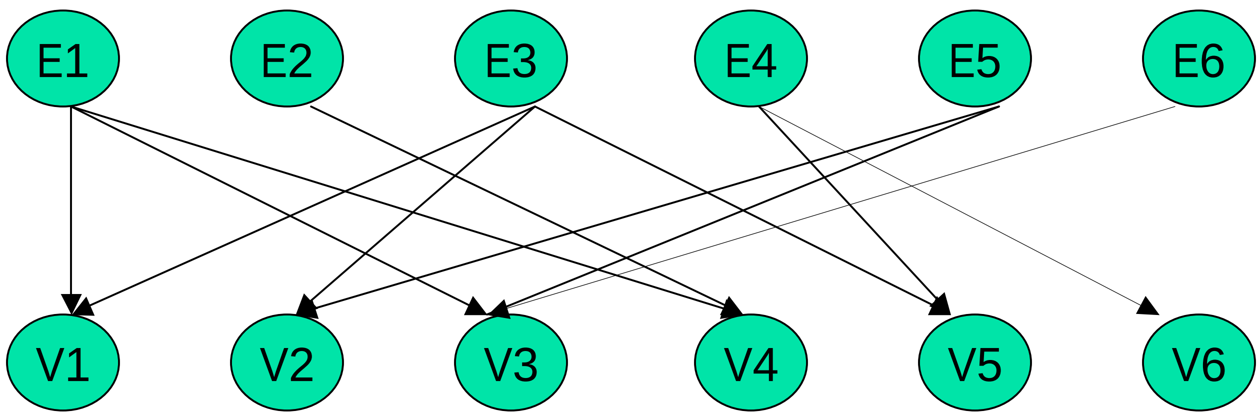
Causality Assignment

- To be able to solve for the various unknowns in the equation set, we need to have a causal representation
- A matching of equations and variables is required, i.e. identify which equation can be used to solved which variable
- This can be accomplished by turning equations and variables into nodes, and dependencies into edges in a bipartite graph
- The problem of matching equations and variables is thus reduced to a *maximum cardinality matching* problem in the bipartite graph

Causality Assignment

- The problem of causality assignment can then be solved by turning it into the *max flow problem* in a one-source, one-sink network
- To form the network, a source and a sink need to be added to the bipartite graph

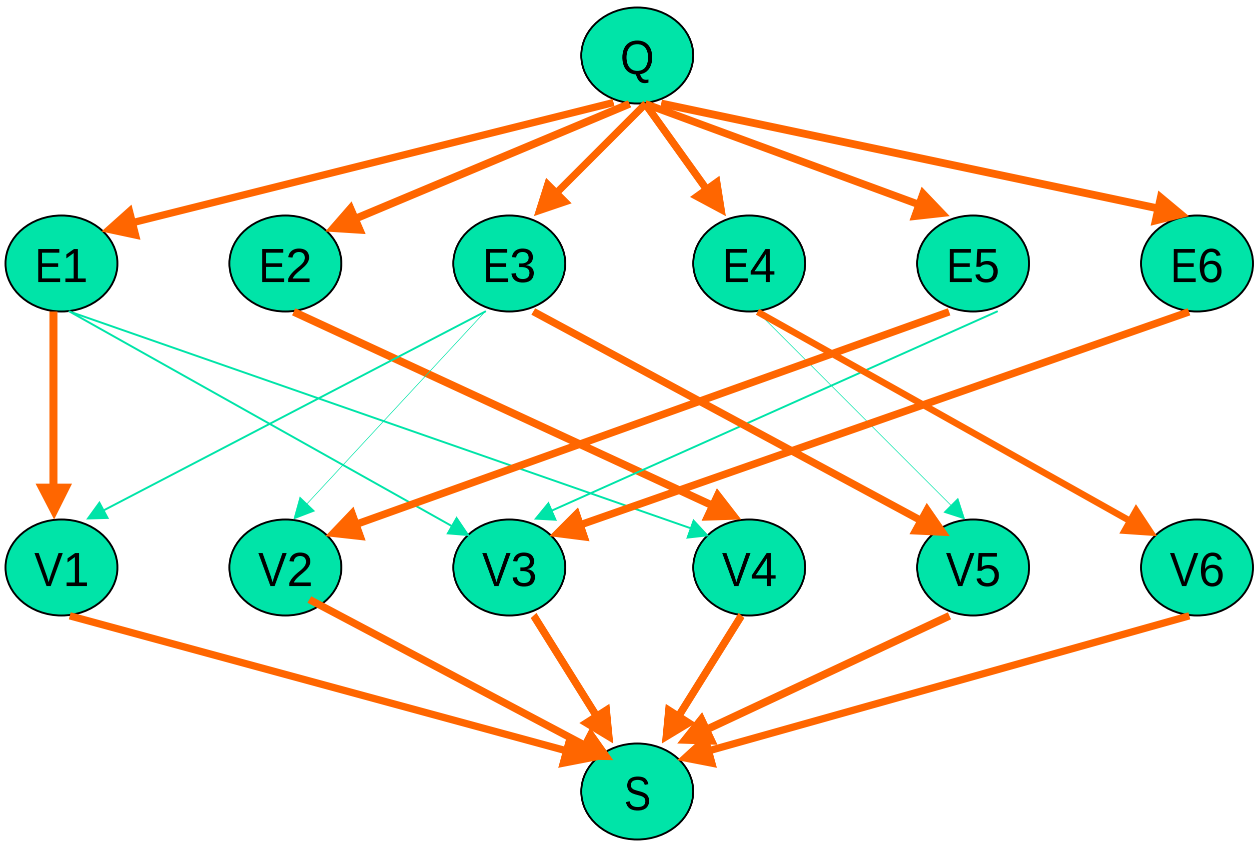
Causality Assignment



Causality Assignment

- Augmenting Path Method (Ford and Fulkerson)
  - Begin with zero flow on all edges
  - Find an augmenting path p for the current flow
  - Increase the value of the flow by pushing res(p) units of flow along p, where res(p) = cap(p) – f(p)
  - Repeat until a flow without an augmenting path is found

Causality Assignment



Causality Assignment

Result: the correspondence between a variable and the equation used to solve it

- `E1: v1 = Ac.v + v3`
- `E2: AC.v = sin(t)`
- `E3: R.v = v1 – v2`
- `E4: i = R.v/r`
- `E5: v2 = C.v + v3`
- `E6: v3 = 0`

Sorting Equations

- The causality assignment gives pairing between equations and variables, but the equations are still in their original sequence
- The objective is that equations must be sorted in the reversed order of their dependencies, i.e. if to calculate a variable is necessary to know the value of another, then the other has to be calculated first
- Algorithm for sorting equations: build a dependency graph and perform a Depth First Search with post-order numbering on this graph

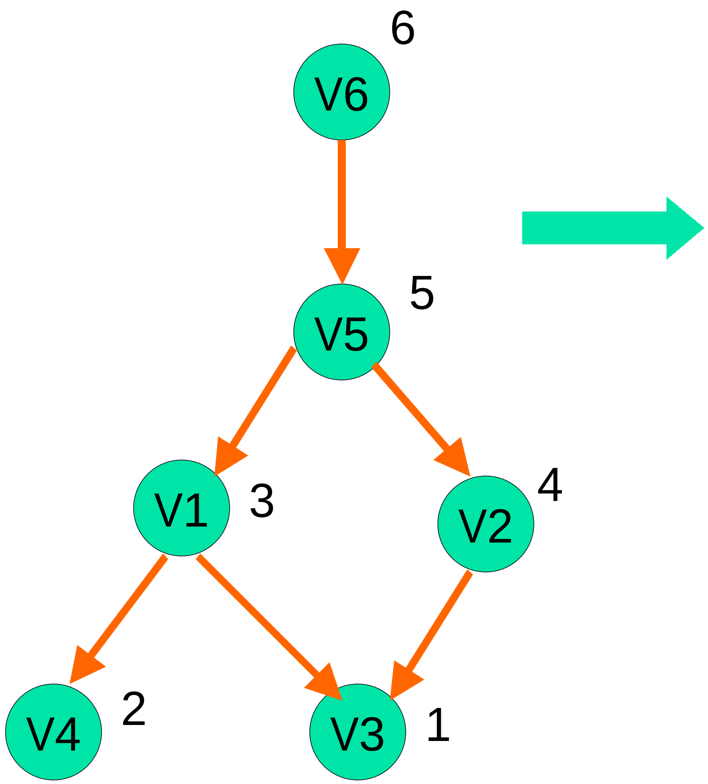
Sorting Equations

Algorithm: DFS sorting

```
dfsCounter=0
for all v in V do
   visited[v]=false
   dfsNr[v]=0
end for
for all v in V
   if not visited[v] then
        DFS(v)
   end if
end for
```

```
DFS(v):
if not visited[v] then
   visited[v]=true
   for all children w of v
        DFS(w)
   end for
   dfsCounter++
   dfsNr[v]=dfsCounter
end if
```

Sorting Equations



- E6: v3 = 0
- E2: AC.v = sin(t)
- E1: v1 = Ac.v + v3
- E5: v2 = C.v + v3
- E3: R.v = v1 – v2
- E4: i = R.v / r

Detecting Algebraic Loops

- In some cases, sorting is not possible due to dependency cycle, or algebraic loop, e.g

```
z = 5
x – y = -6
x + y = -z
```

- Tarjan's O(n+m) (n is the number of vertices, m is the number of graph edges) loop detection algorithm provides an efficient solution

Detecting Algebraic Loops

Tarjan's loop detection algorithm

- Complete DFS on G
- Reverse edges in the annotated G yielding G'
- DFS on G' starting with the highest numbered v. The set of vertices in each DFS tree is a strong component. Remove the strong component form G' and repeat until G' has been removed completely. If there exists no loops, the sets of vertices found will be singletons

Solving Algebraic Loops

- Linear Algebraic Loops
  - An implicit, linear set of n equations in n unknowns may be solved using Cramer's Rule
  - e.g, the equation set

    z = 5

    x – y = -6

    x + y = -z

    has the solution

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

Solving Algebraic Loops

- EcosimPro employs the following numerical procedure to calculate the values of the variables in a non-linear algebraic loop
  - Assuming that the values of one or more variables are known
  - Obtain the value of other variables
  - Obtain the differences between the calculated variables and the expected values
  - Iterate until the differences cancel, i.e. convergence

Solving Algebraic Loops

- For example, to solve this non-linear algebraic loop

  `x + y = 2`

  `x * y = 1`

  - Assuming x is known
  - `y = 2 – x`
  - `Res = x * y – 1`
  - **`Iterate until convergence, ie. res is less than the tolerance`**

Code Generation

- Up to this point, all equations are now in explicit form and ordered in a way such that unknowns can be computed sequentially
- With this set explicit equations, we can now generate code for simulation

References

[1] Overview article of Modelica. Available at: http://www.modelica.org/

[2] Modelica Tutorial, version 1.4. Available at: http://www.modelica.org/documents.shtml

[3] EcosimPro Mathematical Algorithms

[4] Dymola User Manual.

[5] Introduction to Physical Modeling with Modelica. Michael Tiller. 2001

[6] Some Issues Concerning Computer Algebra in AToM3