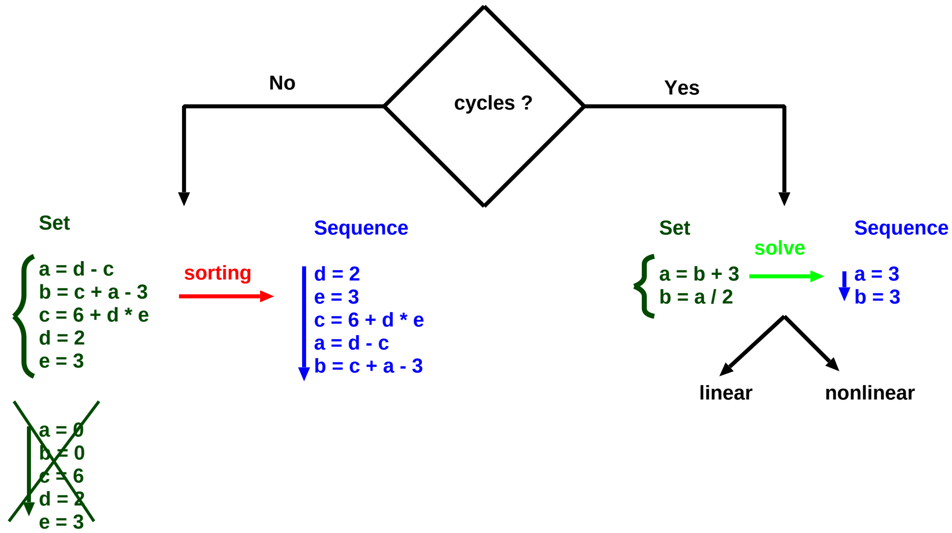
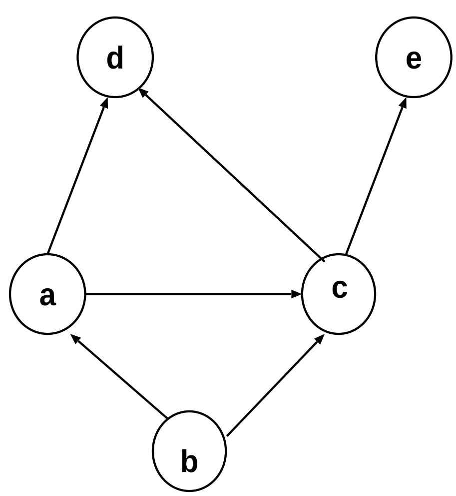
Problems with model re-use: set vs. sequence

DAE-set != DAE-sequence



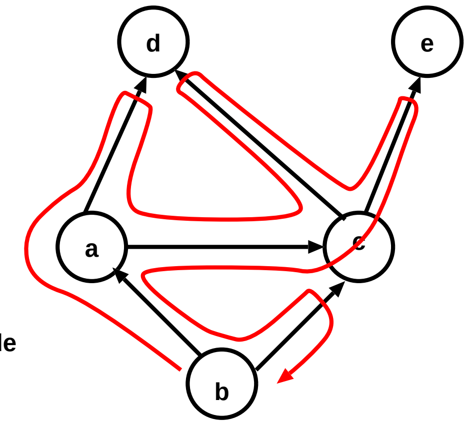
Dependency Graph

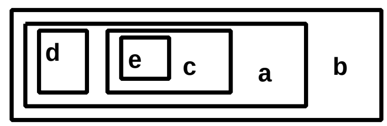
a = d - c
b = c + a - 3
c = 6 + d * e
d = 2
e = 3



Problems with model re-use: sorting post-order traversal depth-first search

```
1. find_root(s)
2. DFS(root)

DFS(node)
{
    if (not_visited(node))
    {
       mark_visited(node)
       foreach child_node of node
       {
          DFS(child_node)
       }
       print(node)
     }
}
```





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

$$\left[\begin{array}{l} z = 5 \\ \left\{\begin{array}{lcl} x - y & = & -6 \\ x + y & = & -z \end{array}\right. \end{array}\right.$$

Implicit set of n equations in n unknowns.

- non-linear $\rightarrow$ non-linear residual solver.
- linear $\rightarrow$ numeric or symbolic solution.

May be solved symbolically (if linear and not too large)

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

$$\left[\begin{array}{lcl} z & = & 5 \\ x & = & \frac{-6-z}{2} \\ y & = & \frac{6-z}{2} \end{array}\right.$$

Simple Loop Detection

1. Build dependency matrix D
2. Calculate transitive closure D^*
3. If $True$ on diagonal of D^*, a loop exists

Even with Warshall's algorithm, still $O(n^3)$ and don't know immediately which nodes involved in the loop(s).

Tarjan's $O(n+m)$ Loop Detection (1972)

1. Complete Depth First Search (DFS) on G
   (possibly multiple DFS trees), postorder numbering

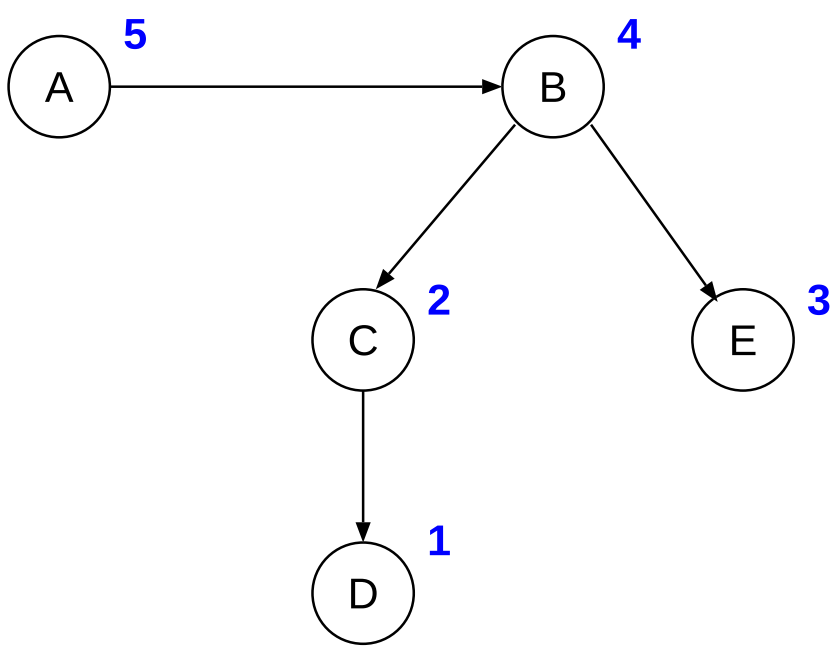
```
FOREACH v IN V
 dfsNr[v] <- 0
FOREACH v IN V
 IF dfsNr[v] == 0
  DFS(v)
```

2. Reverse edges in the annotated $G \to G_R$
3. DFS on G_R starting with highest numbered v
   set of vertices in each DFS tree $=$ strong component

Set of Algebraic Eqns, no Loops

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

Sorting, no Loops



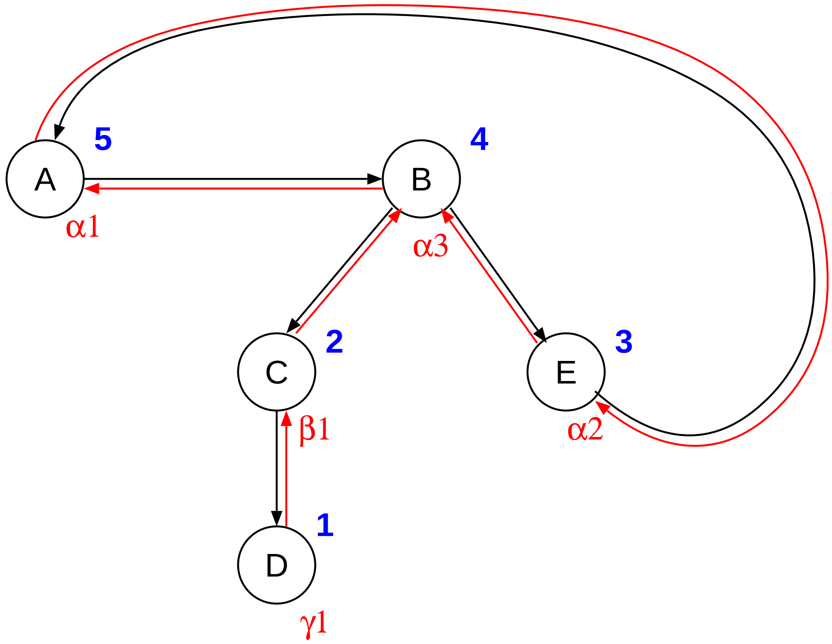
Sorting Result

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

Algebraic Loop (Cycle) Detection

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

Algebraic Loop (Cycle) Detection



Algebraic Loop (Cycle) Detection Result

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

Continuous System Simulation Languages (CSSLs)

- block oriented vs. equation based
- the CSi 1968 CSSL standard
- CSSL-IV, ACSL, Simulink, ADSIM/RT, . . .

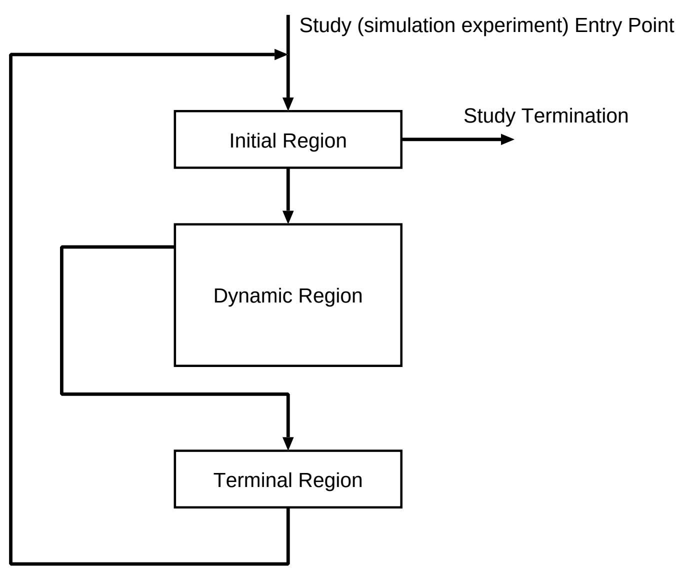
CSSL Requirements

- Easy Model Description (equation based, block oriented)
- Integrator control:
  - select integrator
  - (initial) step size
  - error control
  - variable initialisation
  - parameter setting
- Documentation of model and experiments
- Structured: model vs. experiments (re-use)

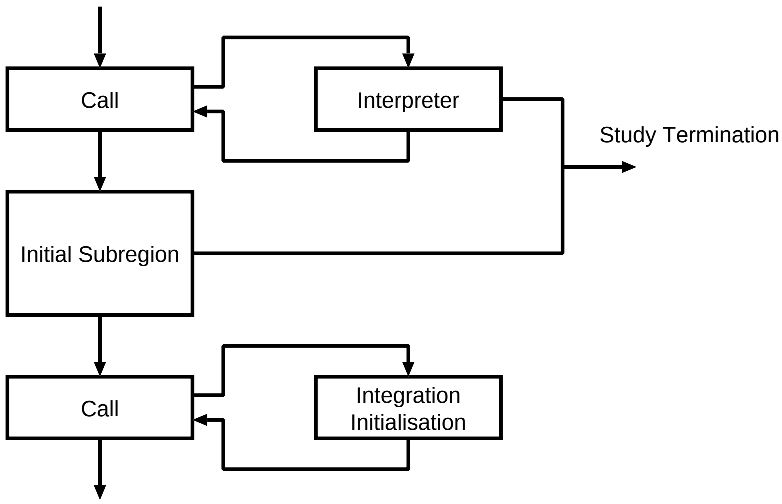
Model Description

```
 DX = INTEG(F-B*X-A*DX, DX0)
  X = INTEG(DX, X0)

DX′ = F-B*X-A*DX
 X′ = DX

at t = 0:

   X = X0
  DX = DX0
```

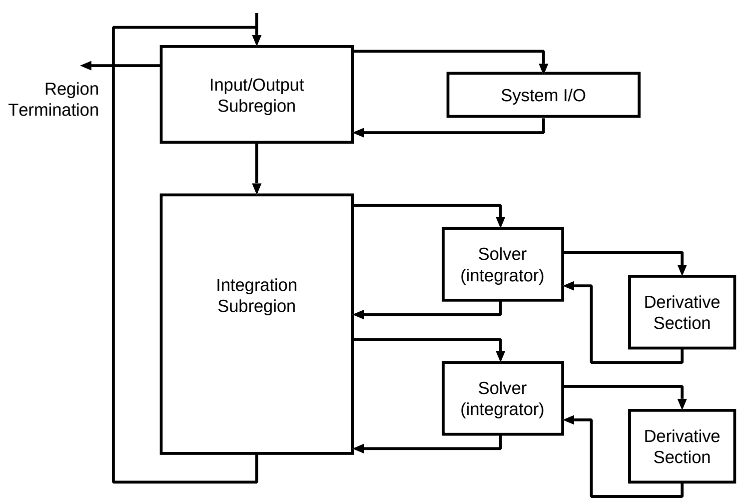

“CSSL study” structure



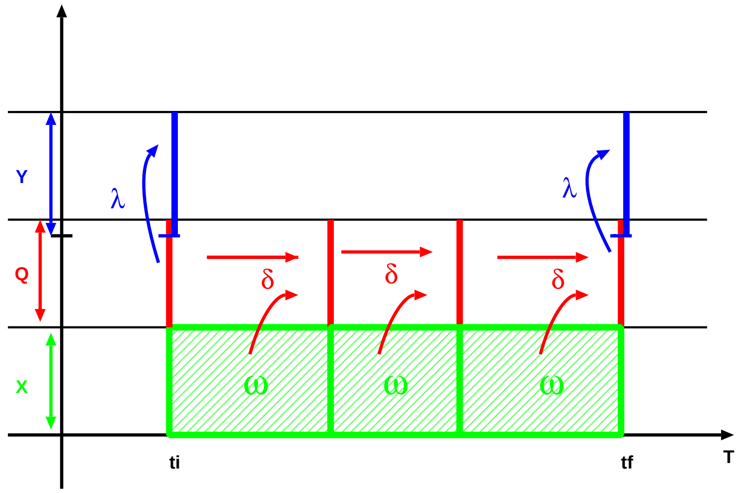
“CSSL initial region” structure



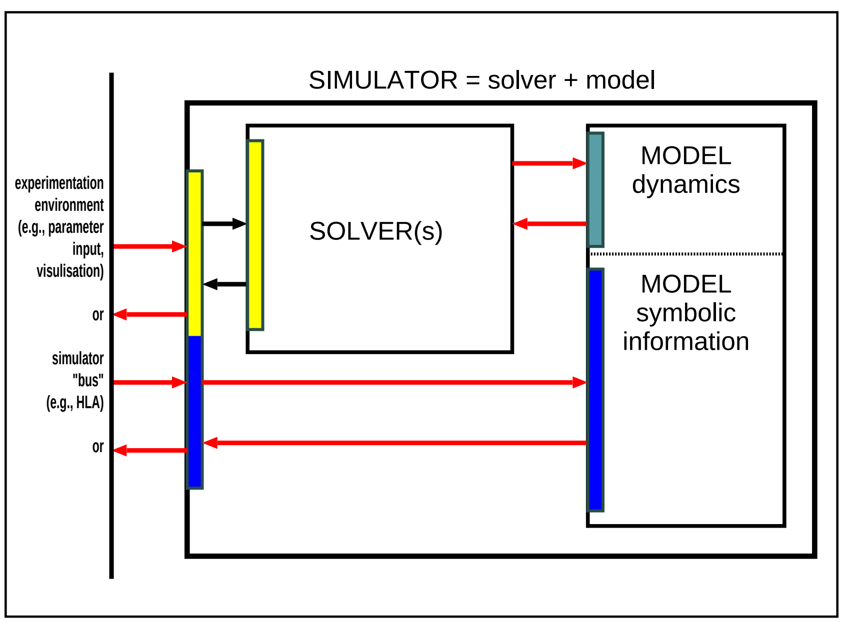
“CSSL dynamic region” structure



General, state-based simulation kernel



Model-solver Architecture



MSL-EXEC Model Representation


```
#include <math.h>
#include <assert.h>
#include "MSLE.h"
#include "MSLExternal.h"
#include "MSLU.h"
#include "Circle.h"

#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]

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));
```

```
  set_no_indep_var_values(1);
  GetIndepVar(0)->LinkValue(this, MSLE_INDEP_VAR, 0);

  set_no_output_var_values(2);
  GetOutputVar(0)->LinkValue(this, MSLE_OUTPUT_VAR, 0);
  GetOutputVar(1)->LinkValue(this, MSLE_OUTPUT_VAR, 1);

  set_no_der_state_var_values(2);
  GetDerStateVar(0)->LinkValue(this, MSLE_DER_STATE_VAR, 0);
  GetDerStateVar(1)->LinkValue(this, MSLE_DER_STATE_VAR, 1);
  GetDerStateVar(0)->LinkInitialValue(this, 0);
  GetDerStateVar(1)->LinkInitialValue(this, 1);
  GetDerStateVar(0)->LinkDerivative(this, 0);
  GetDerStateVar(1)->LinkDerivative(this, 1);

  Reset();
}
```



```
void CircleClass :: ComputeOutput(void)
{
  _x_out_ = _x_;
  _y_out_ = _y_;
}

void CircleClass :: ComputeInitial(void)
{
}

void CircleClass :: ComputeState(void)
{
  _D_x_ = _y_;
  _D_y_ = -_x_;
}

void CircleClass :: ComputeTerminal(void)
{
}

#undef _t_
#undef _x_out_
#undef _y_out_
#undef _x_
#undef _y_
```