INITIALIZATION OF MODELS IN DYMOLA

1. Introduction

Dymola supports initialization of simulation problems according to the specification of Modelica 2.0. This Appendix of the Dymola User's Manual describes the means of Dymola to define initial conditions for a simulation problem.

2. How to define initial conditions

Modelica provides two ways of specifying initial conditions:

1. Start values for variables
2. Initial equations and initial algorithms

Dymola gives guidance on how to make the initialization problem well posed and supports interactive setting of start values.

2.1 Start values for variables

Variables being subtypes of Real, Integer, Boolean and String have an attribute start allowing specification of a start value for the variable

```
Real v(start = 2.0);
parameter Real initx = 0.5;
Real x(start = initx);
```

The value for start should be a parameter expression.

There is also another Boolean attribute *fixed* to indicate whether the value of start is a guess value (fixed = false) to be used in possible iterations or whether the variable is required to have this value at start (fixed = true). For a continuous time variable,

```
Real x(start = initx, fixed = true);
```

implies the additional initialization equation

```
x = initx;
```

while for discrete variables the declarations

```
Boolean b(start = false, fixed = true);
Integer i(start = 1, fixed = true);
```

imply the additional initialization equations

```
pre(b) = false;
pre(i) = 1;
```

For constants and parameters, the attribute fixed is by default true, otherwise fixed is by default false.

2.2 Initial equations and algorithms

A model may have the sections **initial equation** and **initial algorithm** with additional equations and assignments that are used solely in the initialization phase. The equations and assignments in these initial sections are viewed as pure algebraic constraints between the initial values of variables and possibly there derivatives. It is not allowed to use when clauses in the initial sections.

For example, to specify that a variable x shall start in stationarity, we can write

```
initial equation
  der(x) = 0;
```

A more advanced example is

```
parameter Real initx;
parameter Boolean steadyState;
parameter Boolean fixed;
Real x;

initial equation
  if steadyState then
    der(x) = 0;
  else if fixed then
    x = initx;
  end if;
```

If the parameter steadyState is true, then x will be initialized at steady state, because the model specifies the initialization equation

```
initial equation
    der(x) = 0;
```

If the parameter steadyState is false, but fixed is true then there is an initialization equation

```
initial equation
  x = initx;
```

If both staedyState and fixed are false, then there is no initial equation.

The approach as outlined above, allows initx to be any time varying expression. When initx is a parameter expression, the specification above can also be given shorter as

```
parameter Real initx;
parameter Boolean fixed, steadyState;
Real x(fixed = fixed and not steadyState, start = initx);

initial equation
  if steadyState then
    der(x) = 0;
end if;
```

2.3 When clauses and discrete variables at initialization

For the initialization problem there are special semantics rules for when clauses appearing in the model. During simulation a when clause is only active when its condition becomes true. During initialization the equations of a when clause are only active during initialization, if the **initial**() operator explicitly enables it.

```
when {initial(), condition1, …} then
  v = …
end when;
```

Otherwise a when clause is in the initialization problem replaced by v = **pre**(v) for all its left hand side variables.

A reinit statement being active at initialization

```
when initial() then
  reinit(x, expr);
end when;
```

is treated as

```
initial equation
  x = expr;
```

2.4 Interactive setting of start values

The x0 dialogue of the Dymola main window has been redesigned. Previously, it included all continuous time states. Now it includes the continuous time variables having active literal start values. Interactive setting of start values for discrete variables is not yet supported. Setting parameters may of course influence an active start value bound to a parameter expression.

When setting variables from scripts Dymola generates a warning if setting the variable has no effect what-so-ever, e.g. if it is a structural parameter.

2.5 Automatic default selection of initial conditions

The initialization problem is obtained by adding the initial equations as defined above to the simulation problem. We need as many initial equations as there are continuous time states and discrete states. We will not go into this in more technical detail here. Dymola gives guidance.

If initial conditions are missing, Dymola makes automatic default selection of initial conditions. The approach is to select continuous time states or discrete states with inactive start values and make their start values active by virtually turning their fixed to true to get a structurally well posed initialization problem. A log message informing about the result of such a selection is obtained by enabling the option "Logging of default connections". Use the model window menu Preferences/Option or enter the command LogDefaultInitialConditions = true in Dymola main window.

2.6 Over specified initialization problems

At translation Dymola analyzes the initialization problem to check if it is well posed. Dymola splits the problem into four parts with respect to the basic scalar type Real, Integer, Boolean and String and decides whether each of them are structurally well-posed. If such a problem is over specified, Dymola outputs an error message indicating a set of initial equations or fixed start values from which initial equations must be removed or start values inactivated by setting fixed = false.

3. An Example: Initialization of discrete controllers

Below four variants of initializing a simple plant controlled by a discrete PI controller are discussed.

Variant 1: Initial values are given explicitly

```
  parameter Real k=10   "gain of PI controller";
  parameter Real T=1 "Time constant of PI controller";
  parameter Real sampleTime = 0.01;
  input     Real xref  "reference input";
  Real           x (fixed = true, start=2);  // continuous state
  discrete  Real xd(fixed = true, start=0);  // discrete state
  discrete  Real u (fixed = true, start=0);
equation
  // Plant model
  der(x) = -x + u;

  // Discrete PI controller
  when sample(0, sampleTime) then
     xd = pre(xd) + sampleTime/T*(xref - x);
     u  = k*(xd + xref - x);
  end when;
```

The model specifies all the initial values for the states explicitly. The when clause is not enabled at initialization but it is replaced by

```
xd      := pre(xd)
u       := pre(u)
```

The initialization problem is thus

```
x       := x.start  (= 2)
pre(xd) := xd.start (= 0)
pre(u)  := u.start  (= 0)
xd      := pre(xd)
u       := pre(u)
der(x)  := -x + u;
```

Variant 2: Initial values are given explicitly and active controller

The next variant is as Variant 2, but the when clauses is enabled

```
  // Same declaration as variant 1
equation
   der(x) = -x + u;

   when {initial(), sample(0,sampleTime)} then
     xd = pre(xd) + sampleTime/T*(xref - x);
     u  = k*(xd + xref - x);
   end when;
```

It means that the when clause appears as

```
xd = pre(xd) + sampleTime/T*(xref - x);
u  = k*(xd + xref - x);
```

in the initialization problem, which becomes

```
x       := x.start  (= 2)
pre(xd) := xd.start (= 0)
pre(u)  := u.start  (= 0)
xd      := pre(xd) + sampleTime/T*(xref - x);
u       := k*(xd + xref - x);
der(x)  := -x + u;
```

Variant 3: As Variant 2 but initial conditions defined by initial equations

```
  discrete  Real xd;
  discrete  Real u;
  // Remaining declarations as in variant 1
equation
  der(x) = -x + u;
  when {initial(), sample(0, sampleTime)} then
     xd = pre(xd) + sampleTime/T*(xref - x);
     u  = k*(xd + xref - x);
  end when;
initial equation
  pre(xd) = 0;
  pre(u)  = 0;
```

leads to the following equations during initialization

```
x       := x.start  (= 2)
pre(xd) := 0
pre(u)  := 0
xd      := pre(xd) + sampleTime/T*(xref - x)
u       := k*(xd + xref - x)
der(x)  := -x + u;
```

Variant 4: Steady state initialization

Assume that the system is to start in steady state. For continuous time state, x, it means that its derivative shall be zero; **der**(x) = 0; While it for the discrete state, xd, means **pre**(xd) = x; and the when clause shall be active during initialization

```
  Real           x (start=2);
  discrete  Real xd;
  discrete  Real u;
  // Remaining declaration as in Variant 1
equation
  // Plant model
  der(x) = -x + u;

  // Discrete PID controller
  when {initial(), sample(0, sampleTime)} then
     xd = pre(xd) + sampleTime/T*(x - xref);
     u  = k*(xd + x - xref);
  end when;
initial equation
  der(x)  = 0;
  pre(xd) = x;
```

The initialization problem becomes

```
der(x) := 0
// Linear system of equations
    pre(xd) = xd
    xd      = pre(xd) + sampleTime/T*(x - xref)
    u       = k*(xd + xref - x)
    der(x)  = -x + u;
```

Solving the system of equations leads to

```
der(x)  := 0
x       := xref
u       := xref
xd      := xref/k
pre(xd) := xd
```