Equations

Usage of Equations

In Modelica equations are used for many tasks

- The main usage of equations is to represent relations in mathematical models.
- *Assignment statements* in conventional languages are usually represented as equations in Modelica
- *Attribute assignments* are represented as equations
- Connections between objects generate equations
Equation Categories

Equations in Modelica can informally be classified into three different categories

• *Normal equations* (e.g., \( expr1 = expr2 \)) occurring in equation sections, including *connect equations* and other equation types of special syntactic form

• *Declaration equations*, (e.g., Real \( x = 2.0 \)) which are part of variable, parameter, or constant declarations

• *Modifier equations*, (e.g. \( x(\text{unit}="V") \)) which are commonly used to modify attributes of classes.

Constraining Rules for Equations

**Single Assignment Rule**

The total number of "equations" is identical to the total number of "unknown" variables to be solved for

**Synchronous Data Flow Principle**

• All variables keep their actual values until these values are explicitly changed

• At every point in time, during "continuous integration" and at event instants, the *active* equations express relations between variables which have to be fulfilled *concurrently*

  Equations are not active if the corresponding *if*-branch or *when*-equation in which the equation is present is not active because the corresponding branch condition currently evaluates to *false*

• Computation and communication at an event instant does not take time
Declaration Equations

Declaration equations:

\[ \text{constant Integer one} = 1; \]
\[ \text{parameter Real mass} = 22.5; \]

It is also possible to specify a declaration equation for a normal non-constant variable:

\[ \text{Real speed} = 72.4; \]

```
model MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
  Rocket apollo(name="apollo13", mass(start=1038.358) );
  CelestialBody moon(mass=7.382e22, radius=1.738e6, name="moon");
  equation
    apollo.thrust = if (time<thrustDecreaseTime) then force1
                  else if (time<thrustEndTime) then force2
                  else 0;
    apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;
```

Modifier Equations

Modifier equations occur for example in a variable declaration when there is a need to modify the default value of an attribute of the variable. A common usage is modifier equations for the start attribute of variables.

```
modeller
  Real speed(start=72.4);
```

Modifier equations also occur in type definitions:

```
type Voltage = Real(unit="V", min=-220.0, max=220.0);
```

```
model MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
  Rocket apollo(name="apollo13", mass(start=1038.358) );
  CelestialBody moon(mass=7.382e22, radius=1.738e6, name="moon");
  equation
    apollo.thrust = if (time<thrustDecreaseTime) then force1
                  else if (time<thrustEndTime) then force2
                  else 0;
    apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;
```
Kinds of Normal Equations in Equation Sections

Kinds of equations that can be present in equation sections:

- equality equations
- connect equations
- assert and terminate
- reinit

- repetitive equation structures with for-equations
- conditional equations with if-equations
- conditional equations with when-equations

MoonLanding
parameter Real force1 = 36350;
parameter Real force2 = 1308;
parameter Real thrustEndTime = 210;
parameter Real thrustDecreaseTime = 43.2;
Rocket apollo(name="apollo13", mass(start=1038.358) );
CelestialBody moon(mass=7.382e22, radius=1.738e6, name="moon");
equation
if (time<thrustDecreaseTime) then
  apollo.thrust = force1;
elseif (time<thrustEndTime) then
  apollo.thrust = force2;
else
  apollo.thrust = 0;
end if;
apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;

Equality Equations

expr1 = expr2;
(out1, out2, out3,...) = function_name(in_expr1, in_expr2, ...);

class EqualityEquations
Real x,y,z;
equation
(x, y, z) = f(1.0, 2.0); // Correct!
(x+1, 3.0, z/y) = f(1.0, 2.0); // Illegal!
// Not a list of variables
// on the left-hand side
end EqualityEquations;
Repetitive Equations

The syntactic form of a for-equation is as follows:

```
for <iteration-variable> in <iteration-set> loop
  <equation1>
  <equation2>
end for;
```

Consider the following simple example with a for-equation:

```
class FiveEquations
  Real[5]  x;
equation
  for i in 1:5 loop
    x[i] = i+1;
  end for;
eend FiveEquations;
```

```
class FiveEquationsUnrolled
  Real[5]  x;
equation
  x[1] = 2;
  x[2] = 3;
  x[3] = 4;
  x[4] = 5;
  x[5] = 6;
eend FiveEquationsUnrolled;
```

In the class on the right the for-equation has been unrolled into five simple equations:

In Modelica connect-equations are used to establish connections between components via connectors.

```
connect(connector1, connector2)
```

Repetitive connect-equations

```
class RegComponent
  Component components[n];
equation
  for i in 1:n-1 loop
    connect(components[i].outlet, components[i+1].inlet);
  end for;
eend RegComponent;
```
Conditional Equations: if-equations

If the conditions have higher variability than constant or parameter must include an else-part. Each then-, elseif-, and else-branch must have the same number of equations.

```model MoonLanding
  parameter Real force1 = 36350;
  ... Rocket apollo(name="apollo13", mass(start=1038.358) );
  CelestialBody moon(mass=7.382e22, radius=1.738e6, name="moon");
  equation
    if (time<thrustDecreaseTime) then
      apollo.thrust = force1;
    elseif (time<thrustEndTime) then
      apollo.thrust = force2;
    else
      apollo.thrust = 0;
    end if;
    apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;
```

Conditional Equations: when-equations

In when-equations are instantaneous equations that are active at events when conditions become true.

Events are ordered in time and form an event history:

- An event is a point in time that is instantaneous, i.e., has zero duration.
- An event condition switches from false to true in order for the event to take place.
Conditional Equations: when-equations cont'

when <conditions> then
  <equations>
end when;

when x > 2 then
  y1 = sin(x);
  y3 = 2*x + y1+y2;
end when;

when |x > 2, sample(0,2), x < 5| then
  y1 = sin(x);
  y3 = 2*x + y1+y2;
end when;

when initial() then
  ...
  // Equations to be activated at the beginning of a simulation
end when;

when terminal() then
  ...
  // Equations to be activated at the end of a simulation
end when;

when <conditions> then
  <equations>
end when;

when x > 2
  y1 = sin(x);
  y3 = 2*x + y1+y2;
end when;

when |x > 2, sample(0,2), x < 5|
  y1 = sin(x);
  y3 = 2*x + y1+y2;
end when;

when initial() then
  ...
  // Equations to be activated at the beginning of a simulation
end when;

when terminal() then
  ...
  // Equations to be activated at the end of a simulation
end when;

Restrictions on when-equations

Form restriction

Modelica restricts the allowed equations within a when-equation to: variable = expression, if-equations, for-equations,...

In the WhenNotValid model when the equations within the when-equation are not active it is not clear which variable, either x or y, that is a “result” from the when-equation to keep constant outside the when-equation.

A corrected version appears in the class WhenValidResult below

model WhenValidResult
  Real x,y;
equation
  x + y = 5; // Equation to be used to compute y.
  y = ?; // Correct! y is a result variable from the when!
end when;
end WhenValidResult;

model WhenNotValid
  Real x,y;
equation
  x + y = 5; // Equation to be used to compute x.
  y = ?; // Error: not valid Modelica
end when;
end WhenNotValid;
Restrictions on when-equations cont’

Restriction on nested when-equations

```model ErrorNestedWhen
Real x,y1,y2;
equation
  when x > 2 then
    when y1 > 3 then // Error!
      y2 = sin(x);    // when-equations
    end when;
  end when;
end ErrorNestedWhen;
```

when-equations cannot be nested!

Restrictions on when-equations cont’

Single assignment rule: same variable may not be defined in several when-equations.

A conflict between the equations will occur if both conditions would become true at the same time instant

```model DoubleWhenConflict
  Boolean close;   // Error: close defined by two equations!
equation
  ...
  when condition1 then
    close = true;  // First equation
  end when;
  ...
  when condition2 then
    close = false; // Second equation
  end when;
end DoubleWhenConflict
```
Restrictions on when-equations cont’

Solution to assignment conflict between equations in independent when-equations:

• Use `elseif` to give higher priority to the first when-equation

```model DoubleWhenConflictResolved
  ... 
  when condition1 then 
    close = true;  // First equation has higher priority!
  elseif condition2 then 
    close = false;  // Second equation
  end when;
end DoubleWhenConflictResolved;
```
assert-equations

assert is a predefined function for giving error messages taking a Boolean condition and a string as an argument.

The intention behind assert is to provide a convenient means for specifying checks on model validity within a model.

class AssertTest
    parameter Real lowlimit = -5;
    parameter Real highlimit = 5;
    Real x;
    equation
        assert(x >= lowlimit and x <= highlimit, "Variable x out of limit");
end AssertTest;

terminate-equations

The terminate-equation successfully terminates the current simulation, i.e. no error condition is indicated.

model MoonLanding
    parameter Real force1 = 36350;
    parameter Real force2 = 1308;
    parameter Real thrustEndTime = 210;
    parameter Real thrustDecreaseTime = 43.2;
    Rocket apollo(name="apollo13", mass(start=1038.358));
    CelestialBody moon(mass=7.382e22,radius=1.738e6,name="moon");
    equation
        apollo.thrust = if (time<thrustDecreaseTime) then force1
            else if (time<thrustEndTime) then force2
                else 0;
        apollo.gravity = moon.g * moon.mass / (apollo.height + moon.radius)^2;
    when apollo.height < 0 then // termination condition
        terminate("The moon lander touches the ground of the moon");
end when;
end MoonLanding;