OpenModelica Compiler Phases

www.OpenModelica.org

by

Adrian Pop
{adrpo@ida.liu.se}

Open Source Modelica Consortium
2008-05-26

Table of Contents

Chapter 1 The Use Case Example **5**

1.1 The ExampleModel.mo **5**
1.2 The plotting of ExampleModel.mo **6**

Chapter 2 The Main Package **7**

2.1 Function `Main.translateFile` **7**
2.2 Function `Main.optimizeDae` **7**
2.3 Function `Main.simcodegen` **8**

Chapter 3 The Parse package **9**

3.1 The Abysn.Program tree **9**
3.2 The Absyn.ElementItem sub-tree **10**
3.3 The Absyn.Modification sub-tree **11**
3.4 The Absyn.EquationItem sub-tree **12**
3.4.1 The first equation: `der(x) = x;` 12
3.4.2 The last equation: `when x > 2.5 then …;` 13

Chapter 4 The SCode package **14**

4.1 The SCode.elaborate function **14**
4.2 The SCode.Program tree **14**
4.3 The SCode.Element sub-tree **15**
4.4 The SCode.EquationI list **16**
4.4.1 The first equation as SCode.Equation sub-tree: `der(x) = x;` 16
4.4.2 The last equation as SCode.Equation sub-tree: `when x > 2.5 then …;` 17

Chapter 5 The Inst Package **18**

5.1 The Inst.instantiate function **18**
5.2 The DAE.DAEList tree **18**
5.3 The first component (Real x) as a DAE.Element sub-tree **19**
5.4 The first equation (der(x) = x) as a DAE.Element sub-tree **21**
5.5 The last equation (when x > 2.5 then …) as a DAE.Element sub-tree **21**
5.5.1 The equations present in the when equation 22
5.6 The equation (v = y > 1.5;) as an DAE.EQUATION record **23**

Chapter 6 The DAE Package **24**

6.1 The DAE.DAEList structure **24**
6.2 The DAE.Element structure **24**
6.3 The function DAE.transformIfEqToExpr(DAE.DAEList) **24**

Chapter 7 The DAELow Package **26**

7.1 The DAELow.DAELow structure **26**
7.2 The DAELow.lower function **26**
7.3 The DAELow.Variables and DAELow.Var structures **27**
7.4 DAELow.Variables: orderedVars, knownVars, externalObjects **27**
7.4.1 The DAELow.Var tree describing Real x; component in the orderedVars structure. 28
7.5 The DAELow structures for equation, algorithm and event **28**
7.6 The equation, algorithm and event representation in DAELow **29**
7.6.1 The DAE when equation is translated into two DAELow.Equations 30
7.7 The DAELow.incidenceMatrix and DAELow.transposeMatrix functions **32**
7.8 The DAELow.matchingAlgorithm and DAELow.strongComponents functions **33**
7.9 The output of omc +d=bltdump **35**
7.9.1 The output before the matching algorithm 35
7.9.2 The output after the matching algorithm which also performs index reduction 36
7.10 The DAELow.translateDAE and DAELow.calculateValues functions **38**

Chapter 8 The SimCodegen Package **40**

8.1 The preparation for generation of simulation code **40**
8.2 Function SimCodegen.generateFunctions **40**

8.3 Function SimCodegen.generateSimulationCode 40
8.3.1 Code for SimCodegen.generateSimulationCode 40
8.4 Function SimCodegen.generateInitData 42
8.4.1 Code for SimCodegen.generateInitData 42
8.5 Function SimCodegen.generateMakefile 43
8.5.1 Code for SimCodegen.generateMakefile 43

Chapter 9 The generated code 45

9.1 The ExampleModel.cpp file 45
9.2 The ExampleModel.makefile 51

List of Figures

Figure 1-1. The plot of ExampleModel.mo variables. ... 6
Figure 3-1. The parsed AST with Absyn.ElementItem list and Absyn.EquationItem list collapsed ... 9
Figure 3-2. The first `Absyn.ElementItem` that contains component `Real x;` The `Absyn.Modification` sub-tree is collapsed. ... 10
Figure 3-3. The list of modifications for component `Real x;` ... 11
Figure 3-4. The first equation: `der(x) = x;` ... 12
Figure 3-5. The when equation. The then and else part of the if equation are collapsed. ... 13
Figure 4-1. The result: `SCode.Program p_1 = SCode.elaborate(Absyn.Program ast).` The tree has the list of elements and the list of equations collapsed. ... 14
Figure 4-2. The SCode.Element sub-tree for the first component: `Real x;` ... 15
Figure 4-3. The first equation `der(x) = x;` as an SCode.Equation sub-tree. ... 16
Figure 4-4. The `when x > 2.5 then …;` equation as SCode.Equation sub-tree. ... 17
Figure 5-1. The result of instantiation via the `Inst.instantiate` function is a `DAE.DAEList`. ... 18
Figure 5-2. The first component (`Real x`) as a `DAE.Element` containing a `DAE.VAR` record. The first component of the fullType tuple is collapsed. ... 19
Figure 5-3. The type of `DAE.Element` for component `Real x`. The type of stateSelect attribute is collapsed. ... 20
Figure 5-4. The first equation (`der(x) = x;`) as `DAE.EQUATION` record within a `DAE.Element`. ... 21
Figure 5-5. The last equation (`when x > 5 then ...`) as `DAE.EQUATION` record within a `DAE.Element`. ... 21
Figure 5-6. The equation list present in the when equation. The else part of the if equation (equations3) is collapsed. ... 22
Figure 5-7. The equation (`v = y > 1.5;`) as a `DAE.EQUATION` record of type `DAE.Element`. ... 23
Figure 6-1. The if equation in the when equation is now transformed into an if expression. ... 25
Figure 7-1. The `DAELow.DAELow` tree. The tree is collapsed. ... 26
Figure 7-2. DAELow.Variables: orderedVars, knownVars, externalObjects. ... 27
Figure 7-3. The component (`Real x;`) in the `DAELow.Var` representation. ... 28
Figure 7-4. The representation of equation, algorithm and event in DAELow package. ... 30
Figure 7-5. The first DAELow.Equation generated from DAE when equation. ... 31
Figure 7-6. The second DAELow.Equation generated from DAE when equation. ... 31
Figure 7-7. The incidence matrix m. The rows are equations indexes and the columns are variable indexes. ... 32
Figure 7-8. The transposed incidence matrix mT. The rows are variable indexes and the columns are equation indexes. ... 32
Figure 7-9. The v1 is an array of indexes of variables that tells which variable is solved in which equation. The v2 is an array of indexes of equations that tells which variable is solved in which equation. ... 33
Figure 7-10. The new incidence matrix m and its transpose mT after the matching algorithm. ... 34
Figure 7-11. The strong connected components of the system of equations. ... 35
Figure 7-12. Variable name replacement in DAELow.Var. ... 38
Figure 8-1. Partition of blt into states and no states ... 41
Figure 8-2. The created initialization data for the simulation ... 43

Chapter 1

The Use Case Example

1.1 The ExampleModel.mo

The Modelica code that is used as a testcase is presented below ExampleModel.mo:

```
model ExampleModel
  Real x(start=2, stateSelect=StateSelect.always)
       "state with constraint equation with y (choosen)";
  Real y(start=1)
       "state with constraint equation with x (not choosen)";
  // y gets converted to algebraic variable, der(y) is introduced as algebraic variable
  // and the equation "2*der(y) = der(x)" is introduced (derivative of constraint).

  Real z(fixed = true, start=1) "state";
  Real a "output algebraic variable";
  Real b "algebraic variable";
  discrete Real w(start=1) "discrete variable";
  discrete Real u(start=0) "discrete output variable";
  Boolean v;
equation
  der(x) = x;
  der(y) = y + a;
  der(z) = y + b;
  b = der(x) + w;
  x = 2*y "constraint between states";
  v = y > 1.5;
  when x > 2.5 then
    w = time;
    if (v) then  // gets converted to u = if (v) then pre(w) else pre(u) + 1;
      u = pre(w);
    else
      u = pre(u) + 1;
    end if;
  end when;
end ExampleModel;
```

1.2 The plotting of ExampleModel.mo

The `ExampleModel.mos` script to run the simulation and plot the variables.

```
loadFile("ExampleModel.mo");
simulate(ExampleModel);
plot({x, y, z, a, b, w, u, v});
```

The Plot:

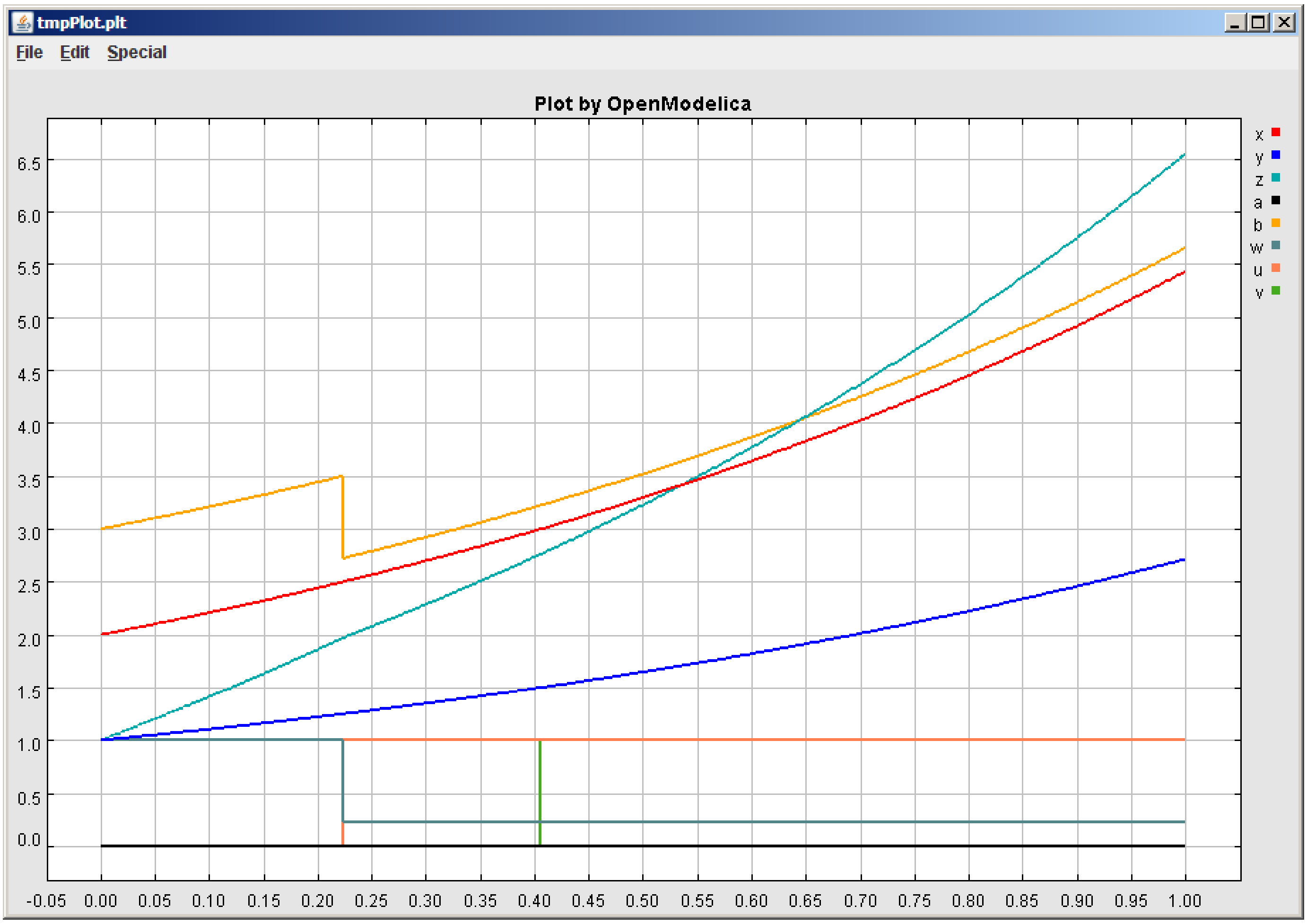


Figure 1-1. The plot of ExampleModel.mo variables.

Chapter 2

The Main Package

2.1 Function `Main.translateFile`

The Main package calls the function `Main.translateFile({modelicaFile})` which performs this sequence of calls:

```
// Parse the file and get an AST back
ast = Parse.parse(modelicaFile);
// Elaborate the file
scode = SCode.elaborate(ast);
// instantiate the simplified code
(cache, dae1) = Inst.instantiate(Env.emptyCache, scode);
// Transform all if equations to if expressions
dae2 = DAE.transformIfEqToExpr(dae1);
// Transform dots in component names into underscores ("." => "_")
dae = fixModelicaOutput(dae2);
// Retrieve the last class name from the AST. This class will be instantiated.
lastClassName = Absyn.lastClassname(ast);
// Call the function that optimizes the DAE
optimizeDae(scode, ast, dae, dae, lastClassName);
```

2.2 Function `Main.optimizeDae`

The function `Main.optimizeDae(scode, ast, dae, daeImpl, lastClassName)` performs this sequence of calls:

```
// Transform the DAE representation into the DAELow representation.
// The DAELow representation splits the DAE into equations and variables
// and further divides variables into known and unknown variables and the
// equations into simple and non-simple equations.
daeLow = DAELow.lower(dae, true, true);
// Calculate the incidence matrix from daeLow. In the incidence matrix the
// rows are equations and the columns are variables. A value of m[row, column] = 1
// is present if the variable represented by “column” appears in the equation
// represented by “row”. A value of m[row, column] = 0 appears if the variable
// represented by “column” is NOT present in the equation represented by “row”.
m = DAELow.incidenceMatrix(daeLow);
// Calculate the transpose of the incidence matrix
mT = DAELow.transposeMatrix(m);
// Apply the matching algorithm to the DAELow representation. This function performs
// the matching algorithm, which is the first part of sorting the equations into
// BLT (Block Lower Triangular) form. The matching algorithm finds a variable that
// is solved in each equation.  To find out which equations forms a block of equations,
// the second algorithm of the BLT sorting: strong components algorithm is run.
// The function returns the updated DAE in case of index reduction has added equations
// and variables, and the incidence matrix. The variable assignments is returned as a
// vector of variable indices, as well as its inverse, i.e. which equation a variable
// is solved in as a vector of equation indices.
// MatchingOptions contains options given to the algorithm.
//    - if index reduction should be used or not.
//    - if the equation system is allowed to be under constrained or not
//      which is used when generating code for initial equations.
(v1,v2,daeLowNew,m,mT) = DAELow.matchingAlgorithm(daeLow, m, mT,
  (DAELow.INDEX_REDUCTION(),DAELow.EXACT(),DAELow.REMOVE_SIMPLE_EQN()));

// Calculate the strong connected components of the incidence matrix.
```

```
// The strong connected components represent subsystems of equations
strongConnComp = DAELow.strongComponents(m, mT, v1, v2);
// Call the simulation code generation function
simcodegen(lastClassName, scode, ast, daeImpl, daeLowNew, v1, v2, m, mT, strongConnComp);
```

2.3 Function Main.simcodegen

The function `simcodegen(lastClassName, scode, ast, dae, daeLowNew, v1, v2, m, mT, strongConnComp)` performs this sequence of calls:

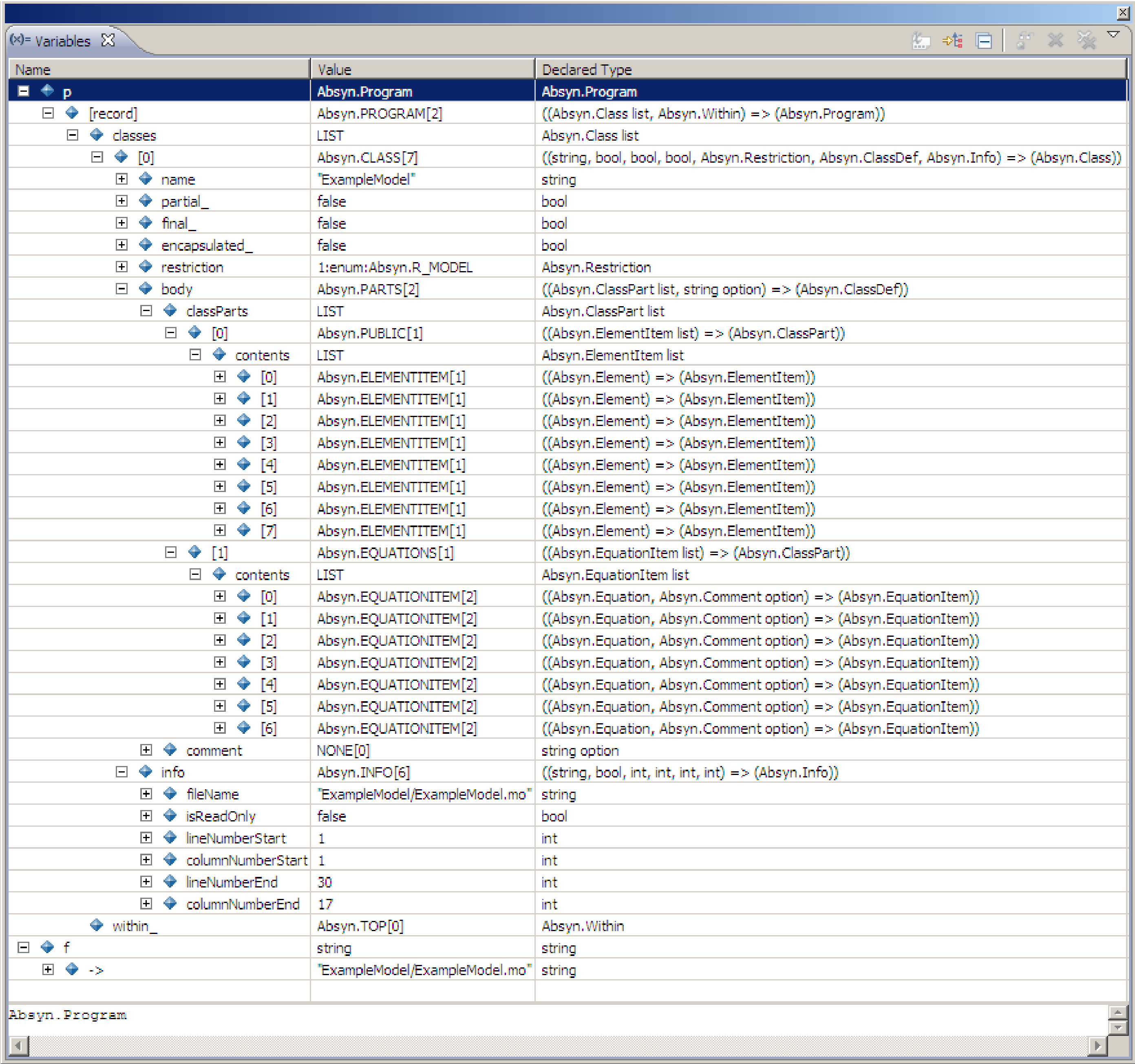
```
// Translates the dae so variables are indexed into different arrays:
//  - xd for derivatives, x for states
//  - dummy_der for dummy derivatives, dummy for dummy states
//  - y for algebraic variables, p for parameters
// This is done by creating defines for each variable.
//  - dots and subscripts in variable names are replaced: $P=".", $lB="[", $rB="]", etc.
//  - For instance, #define a$Pb$Pc xd[3]
// The equations are also updated with the new variable names.
indexedDaeLow1 = DAELow.translateDae(daeLow);

// This function calculates the values from the parameter binding expressions.
// This is performed by building an environment and adding all the parameters
// and constants to it and then calling Ceval.ceval function to retrieve the
// constant values of each parameter or constant.
indexedDaeLow = DAELow.calculateValues(indexedDaeLow1);

// Transform the lastClassName path into a string
classNameStr = Absyn.pathString(lastClassName);

// Generate files to hold the simulation code, functions, init values and a makefile
cppFileName  = stringAppend(classNameStr, ".cpp");
funcFileName = stringAppend(classNameStr, "_functions.cpp");
initFileName = stringAppend(classNameStr, "_init.txt");
makeFileName = stringAppend(classNameStr, ".makefile"});

// Obtain the directory where to output the generated file
componentReference = Absyn.pathToCref(lastClassName);
directoryOfFile = Ceval.getFileDir(componentReference, ast);

// Generate the code for the functions and write it into funcFileName
libs = SimCodegen.generateFunctions(scode, dae, indexedDaeLow,
                                    lastClassName, funcFileName);
// Generate the simulation code and write it into cppFileName.
// Also, include funcFileName in the generated simulation code.
SimCodegen.generateSimulationCode(dae, indexedDaeLow, v1, v2, m, mT,
   strongConnComp, lastClassName, cppFileName, funcFileName, directoryOfFile);

// Generate the initialization assignments that contains values for parameters and
// the values for start attribute of variables and write them into initFileName.
SimCodegen.generateInitData(indexedDaeLow, lastClassName, classNameStr,
   initFileName, 0.0, 1.0, 500.0, 1e-10, "dassl");

// Generate the makefile to help build the executable for the code generation
SimCodegen.generateMakefile(makeFileName, classNameStr, libs, directoryOfFile);
```

Chapter 3

The Parse package

The Parse.parse function parses the file given as argument and builds an AST. When called with “`ExampleModel.mo`” the following AST is generated. Please check the package `Absyn.mo` to understand the presented structures.

3.1 The Abysn.Program tree



Name	Value	Declared Type
p	Absyn.Program	Absyn.Program
[record]	Absyn.PROGRAM[2]	((Absyn.Class list, Absyn.Within) => (Absyn.Program))
classes	LIST	Absyn.Class list
[0]	Absyn.CLASS[7]	((string, bool, bool, bool, Absyn.Restriction, Absyn.ClassDef, Absyn.Info) => (Absyn.Class))
name	"ExampleModel"	string
partial_	false	bool
final_	false	bool
encapsulated_	false	bool
restriction	1:enum:Absyn.R_MODEL	Absyn.Restriction
body	Absyn.PARTS[2]	((Absyn.ClassPart list, string option) => (Absyn.ClassDef))
classParts	LIST	Absyn.ClassPart list
[0]	Absyn.PUBLIC[1]	((Absyn.ElementItem list) => (Absyn.ClassPart))
contents	LIST	Absyn.ElementItem list
[0]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[1]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[2]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[3]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[4]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[5]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[6]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[7]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
[1]	Absyn.EQUATIONS[1]	((Absyn.EquationItem list) => (Absyn.ClassPart))
contents	LIST	Absyn.EquationItem list
[0]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
[1]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
[2]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
[3]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
[4]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
[5]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
[6]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
comment	NONE[0]	string option
info	Absyn.INFO[6]	((string, bool, int, int, int, int) => (Absyn.Info))
fileName	"ExampleModel/ExampleModel.mo"	string
isReadOnly	false	bool
lineNumberStart	1	int
columnNumberStart	1	int
lineNumberEnd	30	int
columnNumberEnd	17	int
within_	Absyn.TOP[0]	Absyn.Within
f	string	string
->	"ExampleModel/ExampleModel.mo"	string

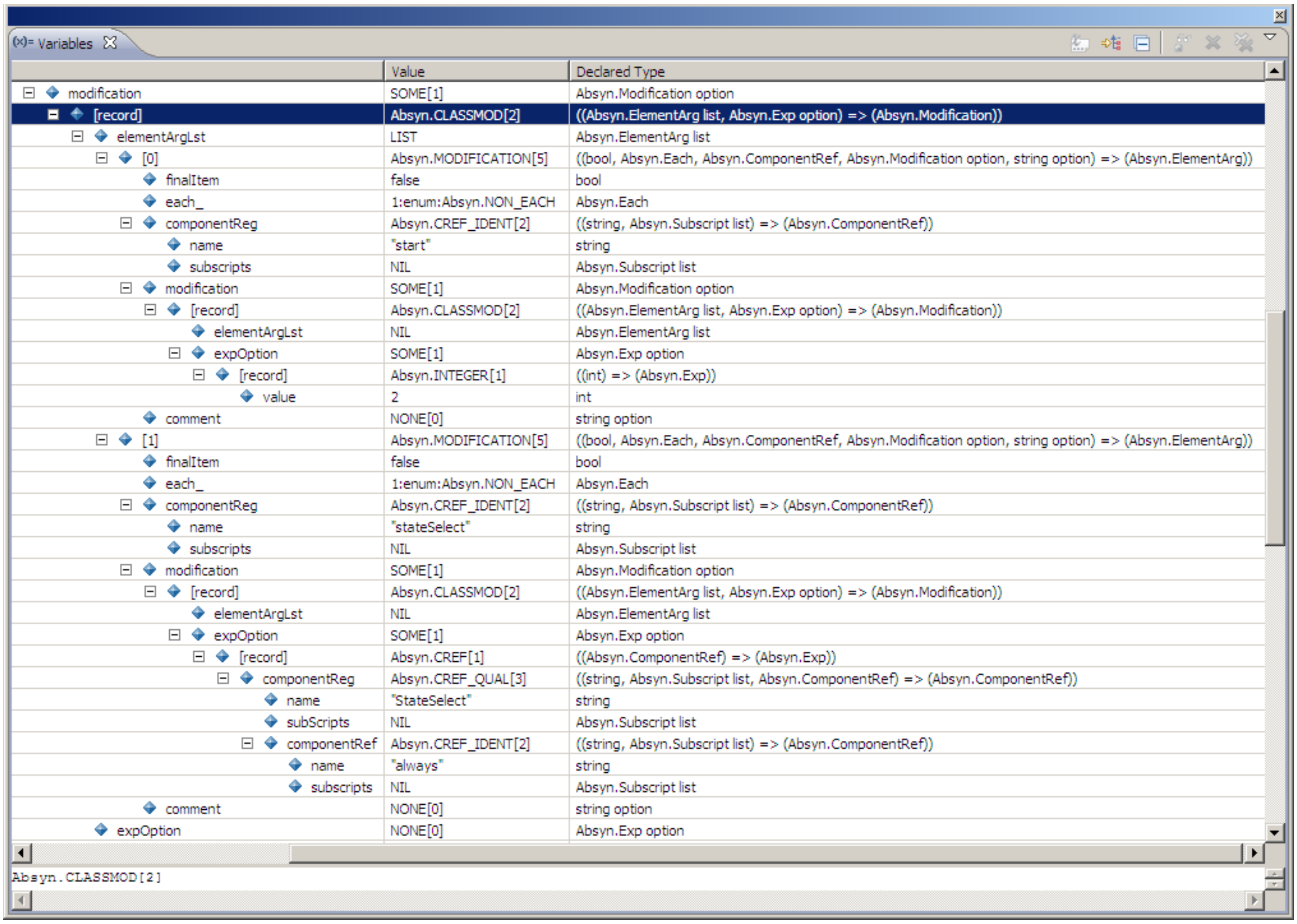
Figure 3-1. The parsed AST with Absyn.ElementItem list and Absyn.EquationItem list collapsed

3.2 The Absyn.ElementItem sub-tree

	Value	Declared Type
contents	LIST	Absyn.ElementItem list
[0]	Absyn.ELEMENTITEM[1]	((Absyn.Element) => (Absyn.ElementItem))
element	Absyn.ELEMENT[7]	((bool, Absyn.RedeclareKeywords option, Absyn.InnerOuter, string, Absyn.ElementSpec, Absyn.Info, Absyn.
final_	false	bool
redeclareKeywords	NONE[0]	Absyn.RedeclareKeywords option
innerOuter	3:enum:Absyn.UNSPECIF…	Absyn.InnerOuter
name	"component"	string
specification	Absyn.COMPONENTS[3]	((Absyn.ElementAttributes, Absyn.TypeSpec, Absyn.ComponentItem list) => (Absyn.ElementSpec))
attributes	Absyn.ATTR[4]	((bool, Absyn.Variability, Absyn.Direction, Absyn.Subscript list) => (Absyn.ElementAttributes))
flow_	false	bool
variability	0:enum:Absyn.VAR	Absyn.Variability
direction	2:enum:Absyn.BIDIR	Absyn.Direction
arrayDim	NIL	Absyn.Subscript list
typeSpec	Absyn.TPATH[2]	((Absyn.Path, Absyn.Subscript list option) => (Absyn.TypeSpec))
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"Real"	string
arrayDim	NONE[0]	Absyn.Subscript list option
components	LIST	Absyn.ComponentItem list
[0]	Absyn.COMPONENTITEM[3]	((Absyn.Component, Absyn.Exp option, Absyn.Comment option) => (Absyn.ComponentItem))
component	Absyn.COMPONENT[3]	((string, Absyn.Subscript list, Absyn.Modification option) => (Absyn.Component))
name	"x"	string
arrayDim	NIL	Absyn.Subscript list
modification	SOME[1]	Absyn.Modification option
[record]	Absyn.CLASSMOD[2]	((Absyn.ElementArg list, Absyn.Exp option) => (Absyn.Modification))
elementArgLst	LIST	Absyn.ElementArg list
[0]	Absyn.MODIFICATION[5]	((bool, Absyn.Each, Absyn.ComponentRef, Absyn.Modification option, string option) => (Absyn.ElementArg))
[1]	Absyn.MODIFICATION[5]	((bool, Absyn.Each, Absyn.ComponentRef, Absyn.Modification option, string option) => (Absyn.ElementArg))
expOption	NONE[0]	Absyn.Exp option
condition	NONE[0]	Absyn.Exp option
comment	SOME[1]	Absyn.Comment option
[record]	Absyn.COMMENT[2]	((Absyn.Annotation option, string option) => (Absyn.Comment))
annotation_	SOME[1]	Absyn.Annotation option
comment	SOME[1]	string option
->	"state with constraint equ…	string
info	Absyn.INFO[6]	((string, bool, int, int, int, int) => (Absyn.Info))
fileName	"ExampleModel/ExampleM…	string
isReadOnly	false	bool
lineNumberStart	2	int
columnNumberStart	3	int
lineNumberEnd	3	int
columnNumberEnd	57	int
constrainClass	NONE[0]	Absyn.ConstrainClass option

Figure 3-2. The first **`Absyn.ElementItem`** that contains component **`Real x`**;
The **`Absyn.Modification`** sub-tree is collapsed.

3.3 The Absyn.Modification sub-tree



(x)= Variables

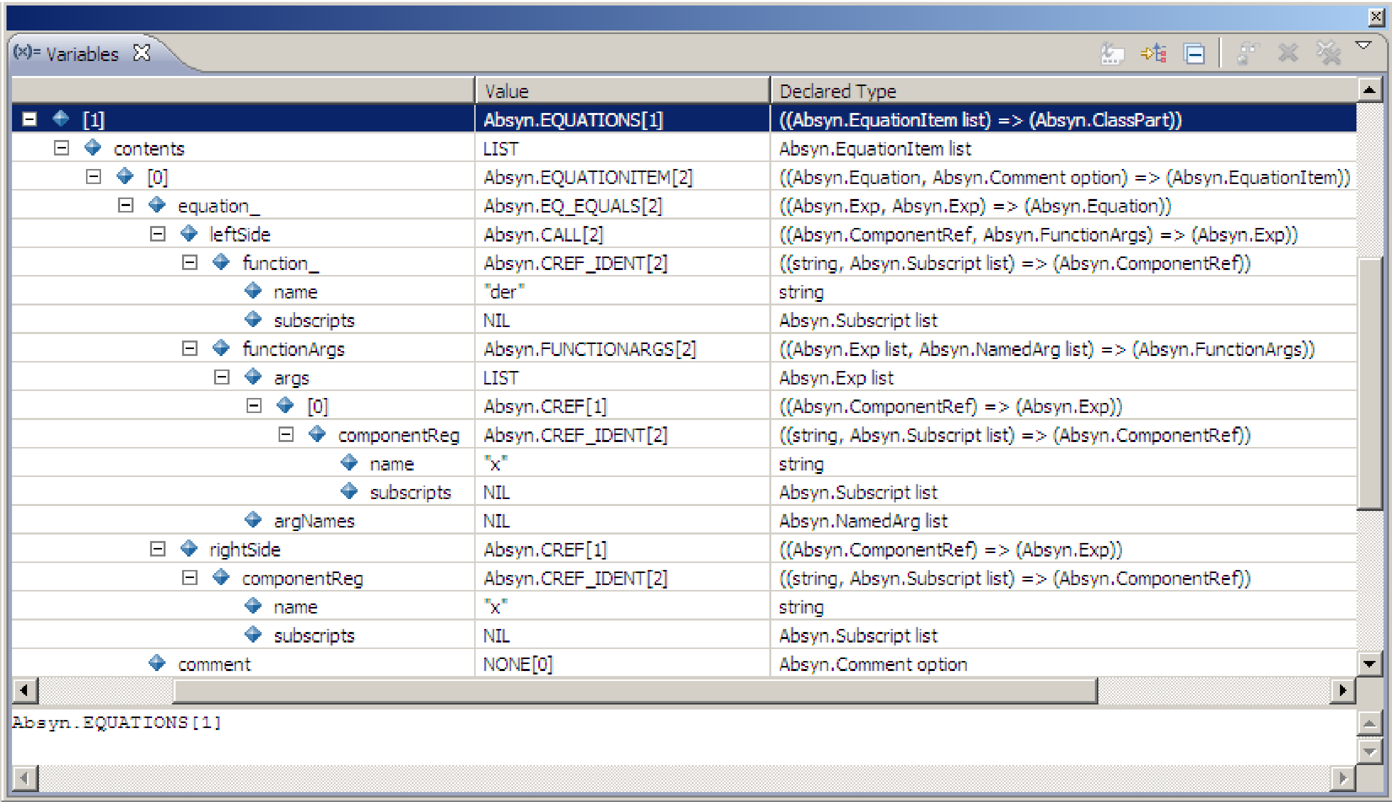
	Value	Declared Type
modification	SOME[1]	Absyn.Modification option
[record]	Absyn.CLASSMOD[2]	((Absyn.ElementArg list, Absyn.Exp option) => (Absyn.Modification))
elementArgLst	LIST	Absyn.ElementArg list
[0]	Absyn.MODIFICATION[5]	((bool, Absyn.Each, Absyn.ComponentRef, Absyn.Modification option, string option) => (Absyn.ElementArg))
finalItem	false	bool
each_	1:enum:Absyn.NON_EACH	Absyn.Each
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"start"	string
subscripts	NIL	Absyn.Subscript list
modification	SOME[1]	Absyn.Modification option
[record]	Absyn.CLASSMOD[2]	((Absyn.ElementArg list, Absyn.Exp option) => (Absyn.Modification))
elementArgLst	NIL	Absyn.ElementArg list
expOption	SOME[1]	Absyn.Exp option
[record]	Absyn.INTEGER[1]	((int) => (Absyn.Exp))
value	2	int
comment	NONE[0]	string option
[1]	Absyn.MODIFICATION[5]	((bool, Absyn.Each, Absyn.ComponentRef, Absyn.Modification option, string option) => (Absyn.ElementArg))
finalItem	false	bool
each_	1:enum:Absyn.NON_EACH	Absyn.Each
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"stateSelect"	string
subscripts	NIL	Absyn.Subscript list
modification	SOME[1]	Absyn.Modification option
[record]	Absyn.CLASSMOD[2]	((Absyn.ElementArg list, Absyn.Exp option) => (Absyn.Modification))
elementArgLst	NIL	Absyn.ElementArg list
expOption	SOME[1]	Absyn.Exp option
[record]	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_QUAL[3]	((string, Absyn.Subscript list, Absyn.ComponentRef) => (Absyn.ComponentRef))
name	"StateSelect"	string
subScripts	NIL	Absyn.Subscript list
componentRef	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"always"	string
subscripts	NIL	Absyn.Subscript list
comment	NONE[0]	string option
expOption	NONE[0]	Absyn.Exp option

Absyn.CLASSMOD[2]

Figure 3-3. The list of modifications for component `Real x;`

3.4 The Absyn.EquationItem sub-tree

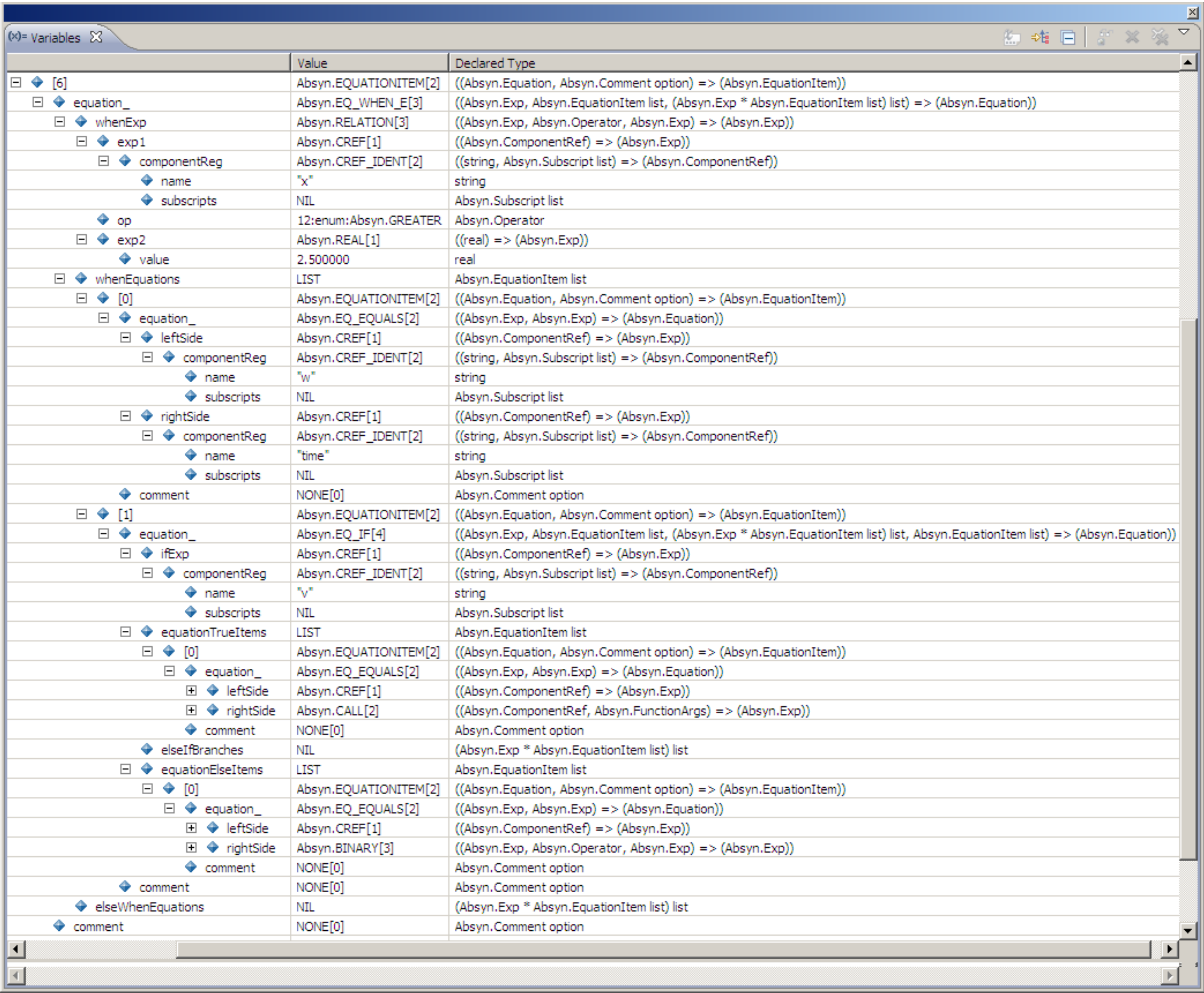
3.4.1 The first equation: `der(x) = x;`



	Value	Declared Type
[1]	Absyn.EQUATIONS[1]	((Absyn.EquationItem list) => (Absyn.ClassPart))
contents	LIST	Absyn.EquationItem list
[0]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
equation_	Absyn.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (Absyn.Equation))
leftSide	Absyn.CALL[2]	((Absyn.ComponentRef, Absyn.FunctionArgs) => (Absyn.Exp))
function_	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"der"	string
subscripts	NIL	Absyn.Subscript list
functionArgs	Absyn.FUNCTIONARGS[2]	((Absyn.Exp list, Absyn.NamedArg list) => (Absyn.FunctionArgs))
args	LIST	Absyn.Exp list
[0]	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"x"	string
subscripts	NIL	Absyn.Subscript list
argNames	NIL	Absyn.NamedArg list
rightSide	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"x"	string
subscripts	NIL	Absyn.Subscript list
comment	NONE[0]	Absyn.Comment option

Figure 3-4. The first equation: `der(x) = x;`

3.4.2 The last equation: `when x > 2.5 then …;`



	Value	Declared Type
[6]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
equation_	Absyn.EQ_WHEN_E[3]	((Absyn.Exp, Absyn.EquationItem list, (Absyn.Exp * Absyn.EquationItem list) list) => (Absyn.Equation))
whenExp	Absyn.RELATION[3]	((Absyn.Exp, Absyn.Operator, Absyn.Exp) => (Absyn.Exp))
exp1	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"x"	string
subscripts	NIL	Absyn.Subscript list
op	12:enum:Absyn.GREATER	Absyn.Operator
exp2	Absyn.REAL[1]	((real) => (Absyn.Exp))
value	2.500000	real
whenEquations	LIST	Absyn.EquationItem list
[0]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
equation_	Absyn.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (Absyn.Equation))
leftSide	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"w"	string
subscripts	NIL	Absyn.Subscript list
rightSide	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"time"	string
subscripts	NIL	Absyn.Subscript list
comment	NONE[0]	Absyn.Comment option
[1]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
equation_	Absyn.EQ_IF[4]	((Absyn.Exp, Absyn.EquationItem list, (Absyn.Exp * Absyn.EquationItem list) list, Absyn.EquationItem list) => (Absyn.Equation))
ifExp	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"v"	string
subscripts	NIL	Absyn.Subscript list
equationTrueItems	LIST	Absyn.EquationItem list
[0]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
equation_	Absyn.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (Absyn.Equation))
leftSide	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
rightSide	Absyn.CALL[2]	((Absyn.ComponentRef, Absyn.FunctionArgs) => (Absyn.Exp))
comment	NONE[0]	Absyn.Comment option
elseIfBranches	NIL	(Absyn.Exp * Absyn.EquationItem list) list
equationElseItems	LIST	Absyn.EquationItem list
[0]	Absyn.EQUATIONITEM[2]	((Absyn.Equation, Absyn.Comment option) => (Absyn.EquationItem))
equation_	Absyn.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (Absyn.Equation))
leftSide	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
rightSide	Absyn.BINARY[3]	((Absyn.Exp, Absyn.Operator, Absyn.Exp) => (Absyn.Exp))
comment	NONE[0]	Absyn.Comment option
comment	NONE[0]	Absyn.Comment option
elseWhenEquations	NIL	(Absyn.Exp * Absyn.EquationItem list) list
comment	NONE[0]	Absyn.Comment option

Figure 3-5. The when equation. The then and else part of the if equation are collapsed.

Chapter 4

The SCode package

4.1 The SCode.elaborate function

This function transfoms an Abyn.Program structure into an SCode.Program structure which is a list of SCode.Class. The translation of the Absyn.Program tree generated by the parser is transformed into the following SCode.Program.

4.2 The SCode.Program tree

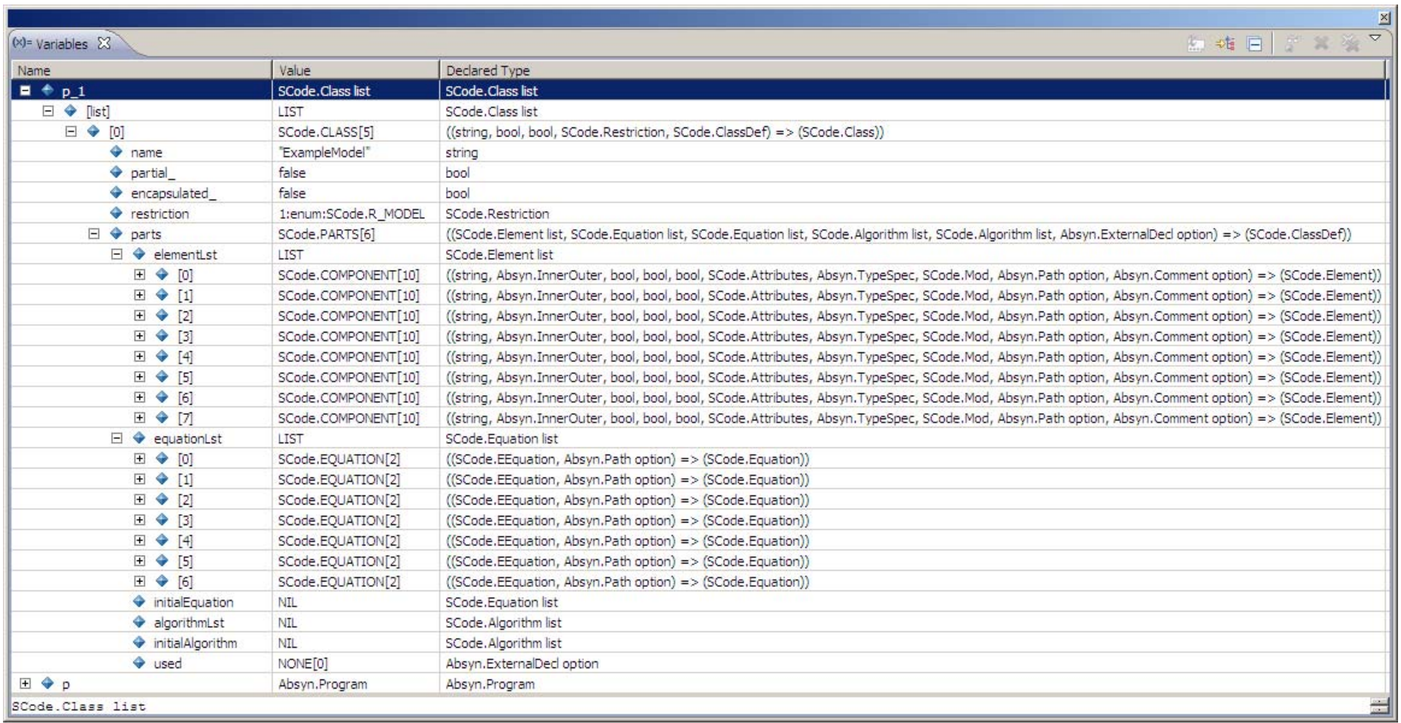


Figure 4-1. The result: **`SCode.Program p_1 = SCode.elaborate(Absyn.Program ast).`**
The tree has the list of elements and the list of equations collapsed.

4.3 The SCode.Element sub-tree

	Value	Declared Type
elementLst	LIST	SCode.Element list
[0]	SCode.COMPONENT[10]	((string, Absyn.InnerOuter, bool, bool, bool, SCode.Attributes, Absyn.TypeSpec, SCod
component	"x"	string
innerOuter	3:enum:Absyn.UNSPECIFIED	Absyn.InnerOuter
final_	false	bool
replaceable_	false	bool
protected_	false	bool
attributes	SCode.ATTR[5]	((Absyn.Subscript list, bool, SCode.Accessibility, SCode.Variability, Absyn.Direction) =>
arrayDim	NIL	Absyn.Subscript list
flow_	false	bool
RW	0:enum:SCode.RW	SCode.Accessibility
parameter_	0:enum:SCode.VAR	SCode.Variability
input_	2:enum:Absyn.BIDIR	Absyn.Direction
typeSpec	Absyn.TPATH[2]	((Absyn.Path, Absyn.Subscript list option) => (Absyn.TypeSpec))
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"Real"	string
arrayDim	NONE[0]	Absyn.Subscript list option
mod	SCode.MOD[4]	((bool, Absyn.Each, SCode.SubMod list, (Absyn.Exp * bool) option) => (SCode.Mod))
final_	false	bool
each_	1:enum:Absyn.NON_EACH	Absyn.Each
subModLst	LIST	SCode.SubMod list
[0]	SCode.NAMEMOD[2]	((string, SCode.Mod) => (SCode.SubMod))
ident	"start"	string
A	SCode.MOD[4]	((bool, Absyn.Each, SCode.SubMod list, (Absyn.Exp * bool) option) => (SCode.Mod))
final_	false	bool
each_	1:enum:Absyn.NON_EACH	Absyn.Each
subModLst	NIL	SCode.SubMod list
absynExpOption	SOME[1]	(Absyn.Exp * bool) option
[tuple]	TUPLE[2](4)	(Absyn.Exp * bool)
[0]	Absyn.INTEGER[1]	((int) => (Absyn.Exp))
value	2	int
[1]	false	bool
[1]	SCode.NAMEMOD[2]	((string, SCode.Mod) => (SCode.SubMod))
ident	"stateSelect"	string
A	SCode.MOD[4]	((bool, Absyn.Each, SCode.SubMod list, (Absyn.Exp * bool) option) => (SCode.Mod))
final_	false	bool
each_	1:enum:Absyn.NON_EACH	Absyn.Each
subModLst	NIL	SCode.SubMod list
absynExpOption	SOME[1]	(Absyn.Exp * bool) option
absynExpOption	NONE[0]	(Absyn.Exp * bool) option
baseclass	NONE[0]	Absyn.Path option
this	SOME[1]	Absyn.Comment option
[record]	Absyn.COMMENT[2]	((Absyn.Annotation option, string option) => (Absyn.Comment))
annotation_	SOME[1]	Absyn.Annotation option
comment	SOME[1]	string option
->	"state with constraint equation with y (choosen)"	string

Figure 4-2. The SCode.Element sub-tree for the first component: `Real x;`

4.4 The SCode.EquationI list

4.4.1 The first equation as SCode.Equation sub-tree: `der(x) = x;`

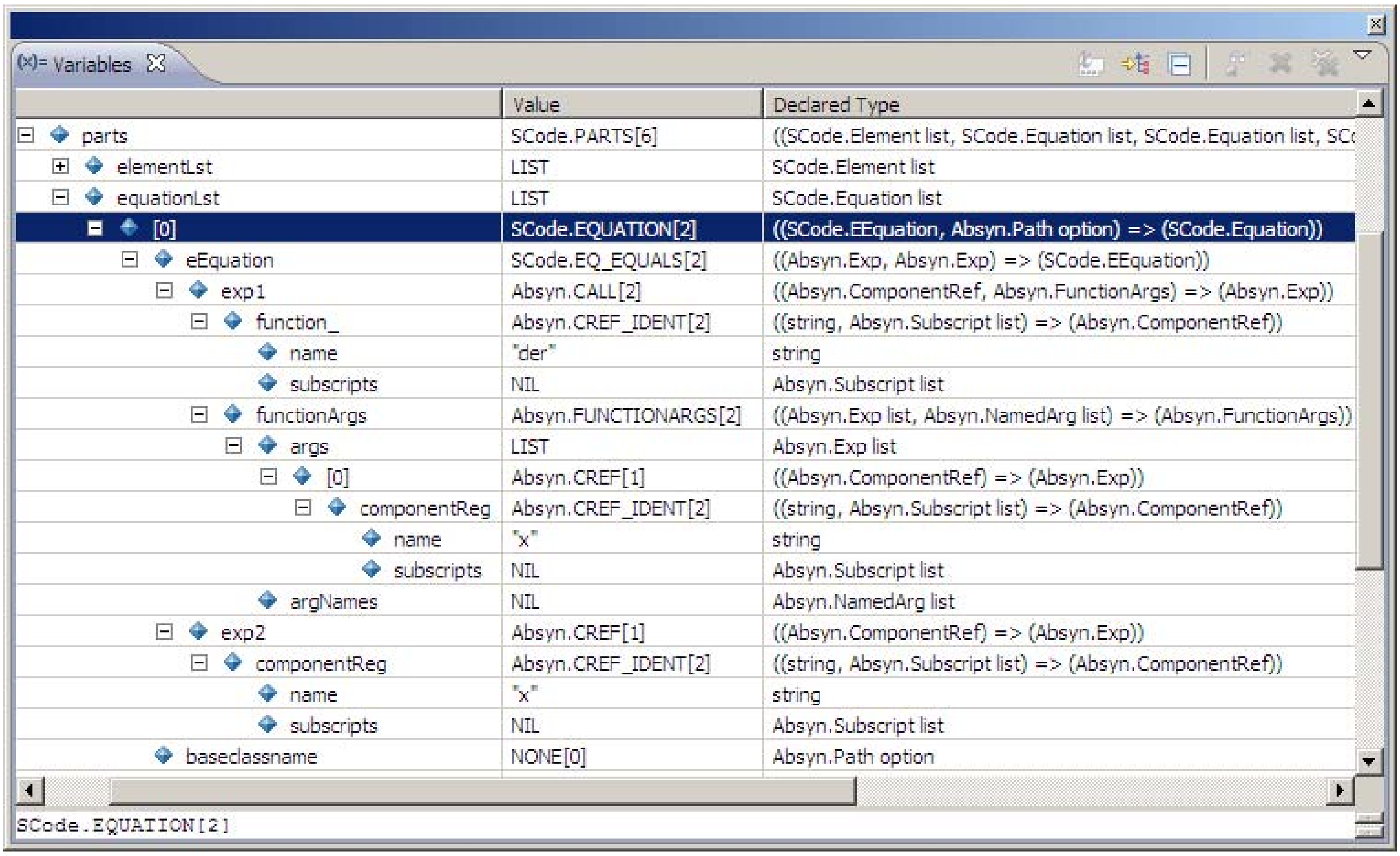
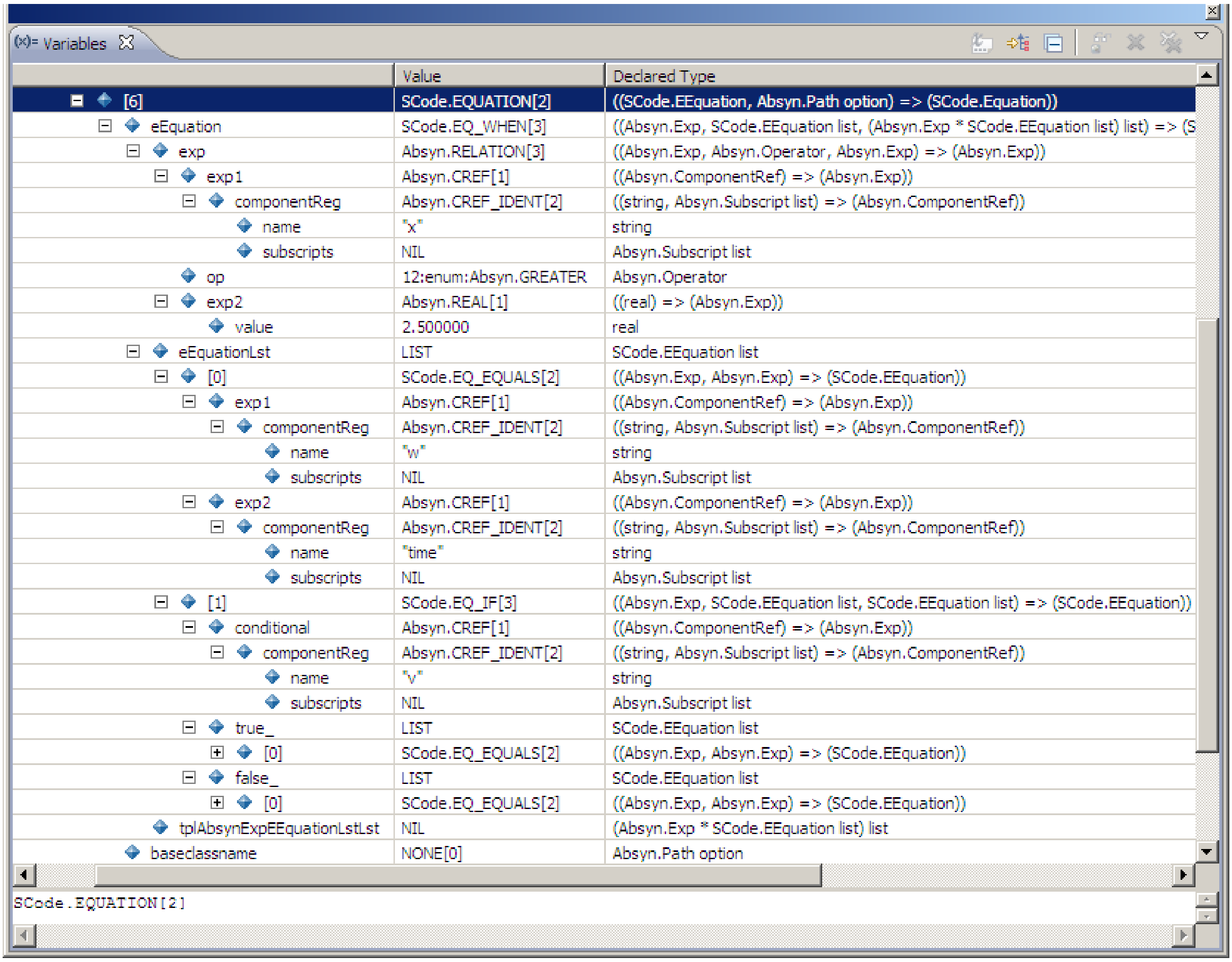


Figure 4-3. The first equation `der(x) = x;` as an SCode.Equation sub-tree.

4.4.2 The last equation as SCode.Equation sub-tree: `when x > 2.5 then …;`



	Value	Declared Type
[6]	SCode.EQUATION[2]	((SCode.EEquation, Absyn.Path option) => (SCode.Equation))
eEquation	SCode.EQ_WHEN[3]	((Absyn.Exp, SCode.EEquation list, (Absyn.Exp * SCode.EEquation list) list) => (S
exp	Absyn.RELATION[3]	((Absyn.Exp, Absyn.Operator, Absyn.Exp) => (Absyn.Exp))
exp1	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"x"	string
subscripts	NIL	Absyn.Subscript list
op	12:enum:Absyn.GREATER	Absyn.Operator
exp2	Absyn.REAL[1]	((real) => (Absyn.Exp))
value	2.500000	real
eEquationLst	LIST	SCode.EEquation list
[0]	SCode.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (SCode.EEquation))
exp1	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"w"	string
subscripts	NIL	Absyn.Subscript list
exp2	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"time"	string
subscripts	NIL	Absyn.Subscript list
[1]	SCode.EQ_IF[3]	((Absyn.Exp, SCode.EEquation list, SCode.EEquation list) => (SCode.EEquation))
conditional	Absyn.CREF[1]	((Absyn.ComponentRef) => (Absyn.Exp))
componentReg	Absyn.CREF_IDENT[2]	((string, Absyn.Subscript list) => (Absyn.ComponentRef))
name	"v"	string
subscripts	NIL	Absyn.Subscript list
true_	LIST	SCode.EEquation list
[0]	SCode.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (SCode.EEquation))
false_	LIST	SCode.EEquation list
[0]	SCode.EQ_EQUALS[2]	((Absyn.Exp, Absyn.Exp) => (SCode.EEquation))
tplAbsynExpEEquationLstLst	NIL	(Absyn.Exp * SCode.EEquation list) list
baseclassname	NONE[0]	Absyn.Path option

Figure 4-4. The `when x > 2.5 then …;` equation as SCode.Equation sub-tree.

Chapter 5

The Inst Package

5.1 The Inst.instantiate function

The Inst.instantiate function instantiates the Modelica code by traversing the SCode.Program tree. It transforms a SCode.Program tree into a DAE.DAElist tree.

The function is called like:

```
(cache, dae) = Inst.instantiate(Env.emptyCache, scode);
```

To instantiate a Modelica program, an initial environment is built, containing the predefined types. Then the program is instantiated by the function instProgram.

The function splits an SCode.Program into two sub-trees, one containing only functions and one with all the other classes. Then the two sub-trees are instantiated separately and the resulted DAE.DAElist lists are appended.

5.2 The DAE.DAEList tree

The `SCode.Program` tree presented in section 4.2 is transformed by applying the function `Inst.instantiate` into the following `DAE.DAEList` list:

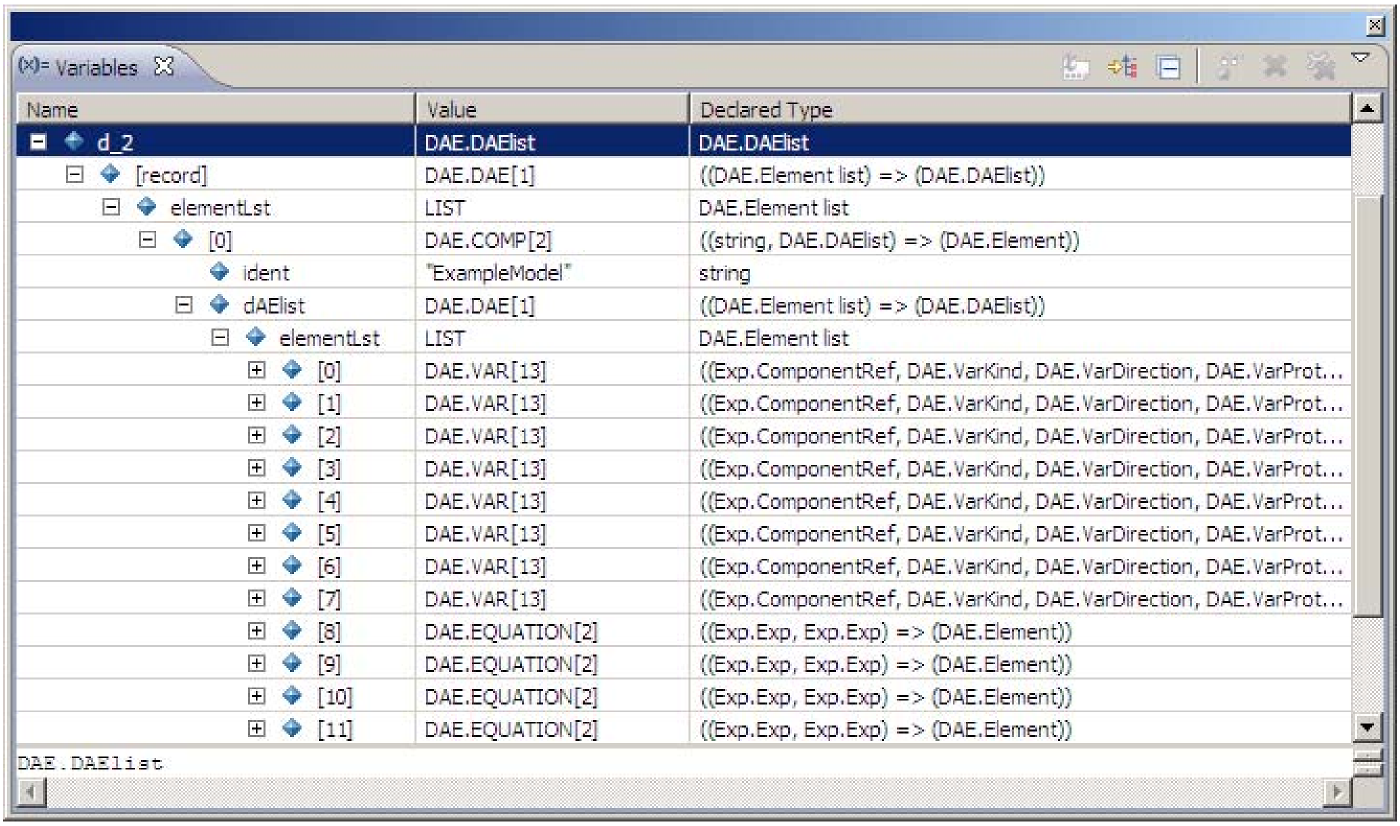
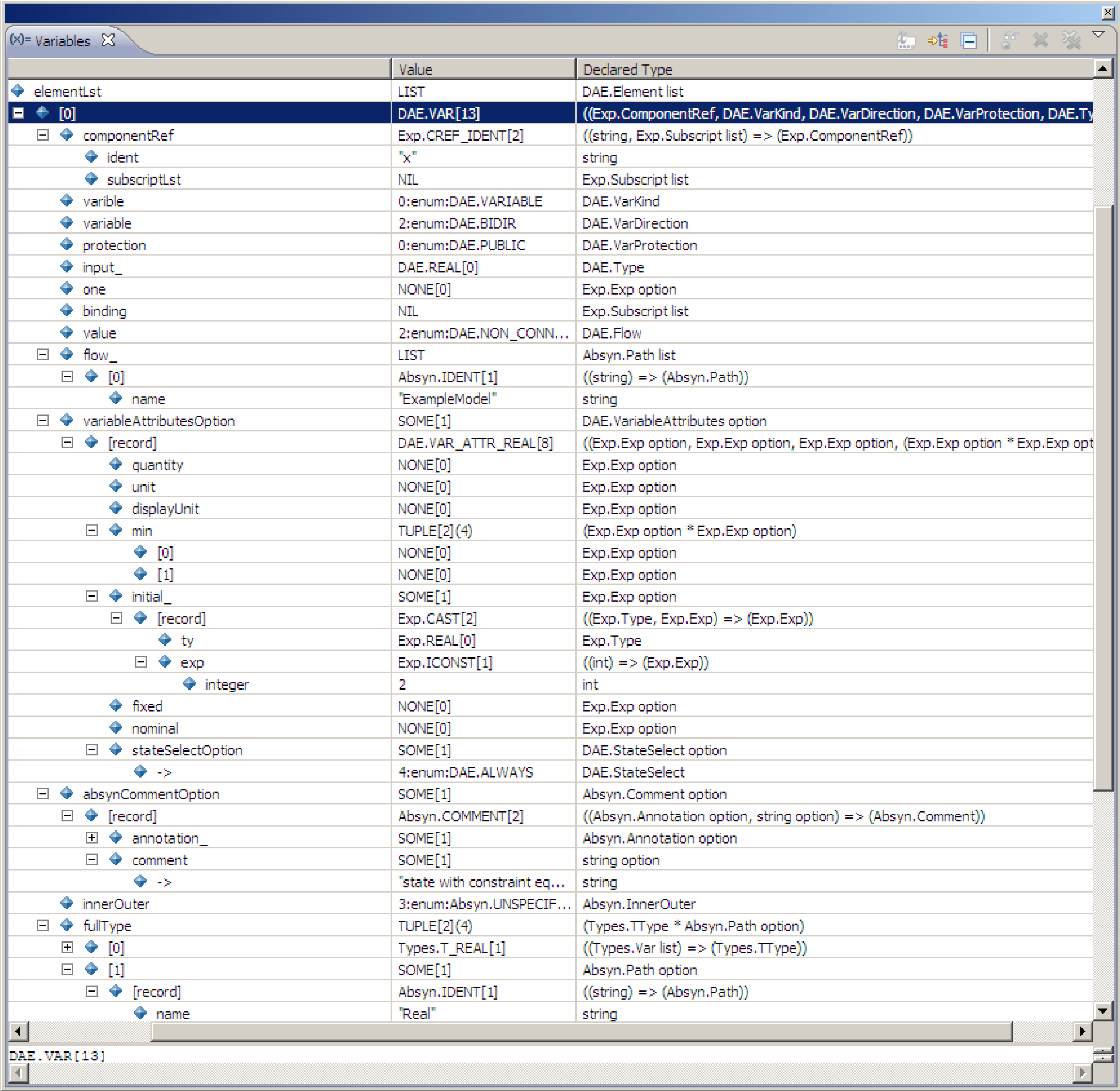


Figure 5-1. The result of instantiation via the **`Inst.instantiate`** function is a **`DAE.DAEList`**.

As one can see inside the tree we have a list of DAE.Element which contains eight (8) DAE.VAR records and four (4) DAE.EQUATION records.

5.3 The first component (Real x) as a DAE.Element sub-tree



	Value	Declared Type
elementLst	LIST	DAE.Element list
[0]	DAE.VAR[13]	((Exp.ComponentRef, DAE.VarKind, DAE.VarDirection, DAE.VarProtection, DAE.Ty
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"x"	string
subscriptLst	NIL	Exp.Subscript list
varible	0:enum:DAE.VARIABLE	DAE.VarKind
variable	2:enum:DAE.BIDIR	DAE.VarDirection
protection	0:enum:DAE.PUBLIC	DAE.VarProtection
input_	DAE.REAL[0]	DAE.Type
one	NONE[0]	Exp.Exp option
binding	NIL	Exp.Subscript list
value	2:enum:DAE.NON_CONN...	DAE.Flow
flow_	LIST	Absyn.Path list
[0]	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"ExampleModel"	string
variableAttributesOption	SOME[1]	DAE.VariableAttributes option
[record]	DAE.VAR_ATTR_REAL[8]	((Exp.Exp option, Exp.Exp option, Exp.Exp option, (Exp.Exp option * Exp.Exp opt
quantity	NONE[0]	Exp.Exp option
unit	NONE[0]	Exp.Exp option
displayUnit	NONE[0]	Exp.Exp option
min	TUPLE[2](4)	(Exp.Exp option * Exp.Exp option)
[0]	NONE[0]	Exp.Exp option
[1]	NONE[0]	Exp.Exp option
initial_	SOME[1]	Exp.Exp option
[record]	Exp.CAST[2]	((Exp.Type, Exp.Exp) => (Exp.Exp))
ty	Exp.REAL[0]	Exp.Type
exp	Exp.ICONST[1]	((int) => (Exp.Exp))
integer	2	int
fixed	NONE[0]	Exp.Exp option
nominal	NONE[0]	Exp.Exp option
stateSelectOption	SOME[1]	DAE.StateSelect option
->	4:enum:DAE.ALWAYS	DAE.StateSelect
absynCommentOption	SOME[1]	Absyn.Comment option
[record]	Absyn.COMMENT[2]	((Absyn.Annotation option, string option) => (Absyn.Comment))
annotation_	SOME[1]	Absyn.Annotation option
comment	SOME[1]	string option
->	"state with constraint eq...	string
innerOuter	3:enum:Absyn.UNSPECIF...	Absyn.InnerOuter
fullType	TUPLE[2](4)	(Types.TType * Absyn.Path option)
[0]	Types.T_REAL[1]	((Types.Var list) => (Types.TType))
[1]	SOME[1]	Absyn.Path option
[record]	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"Real"	string

Figure 5-2. The first component (**`Real x`**) as a **`DAE.Element`** containing a **`DAE.VAR`** record.
The first component of the fullType tuple is collapsed.

The first component of the fullType tuple is presented below as is quite large. The component is a Types.Var structure which contains a Types.T_REAL record.

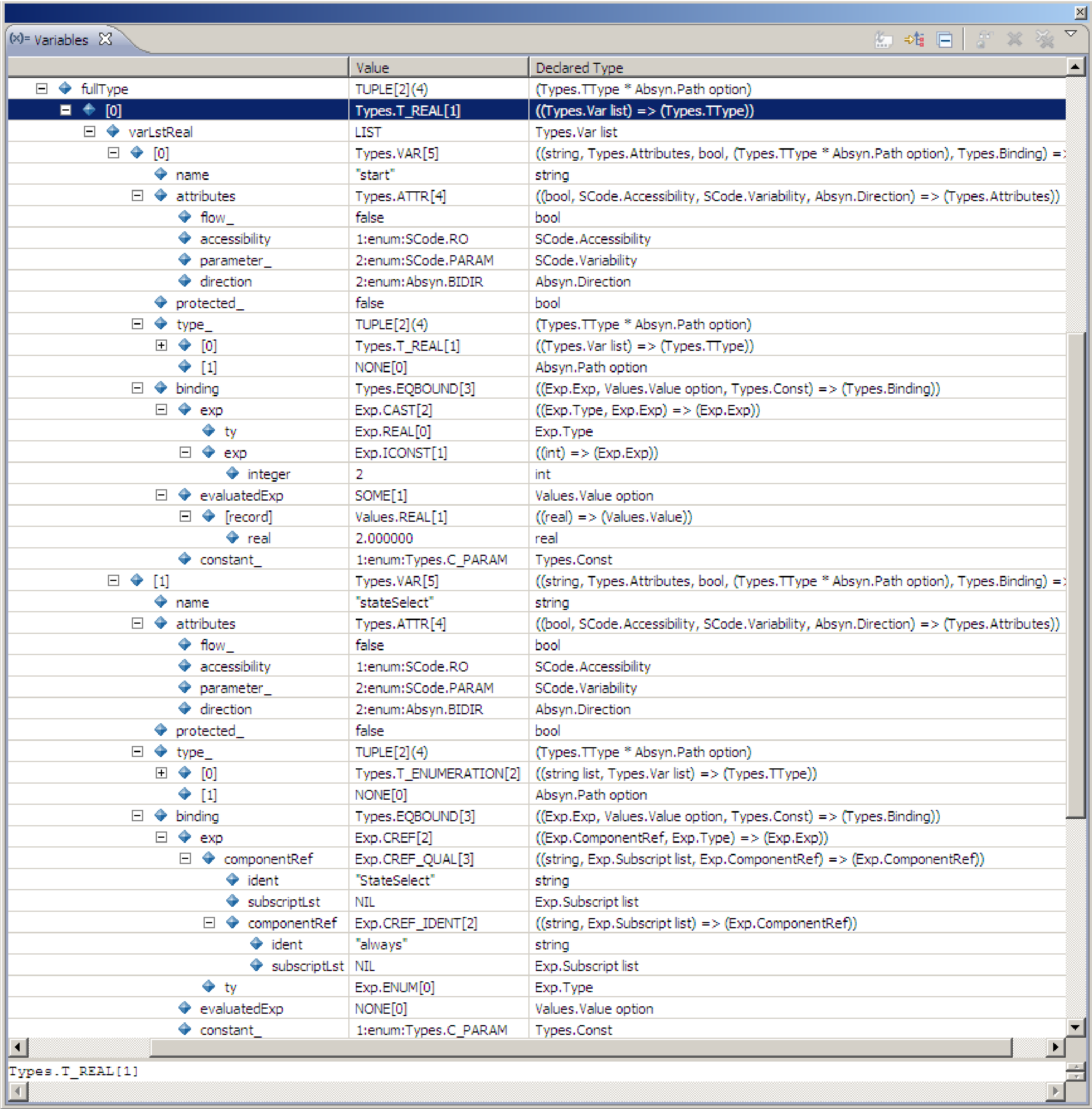
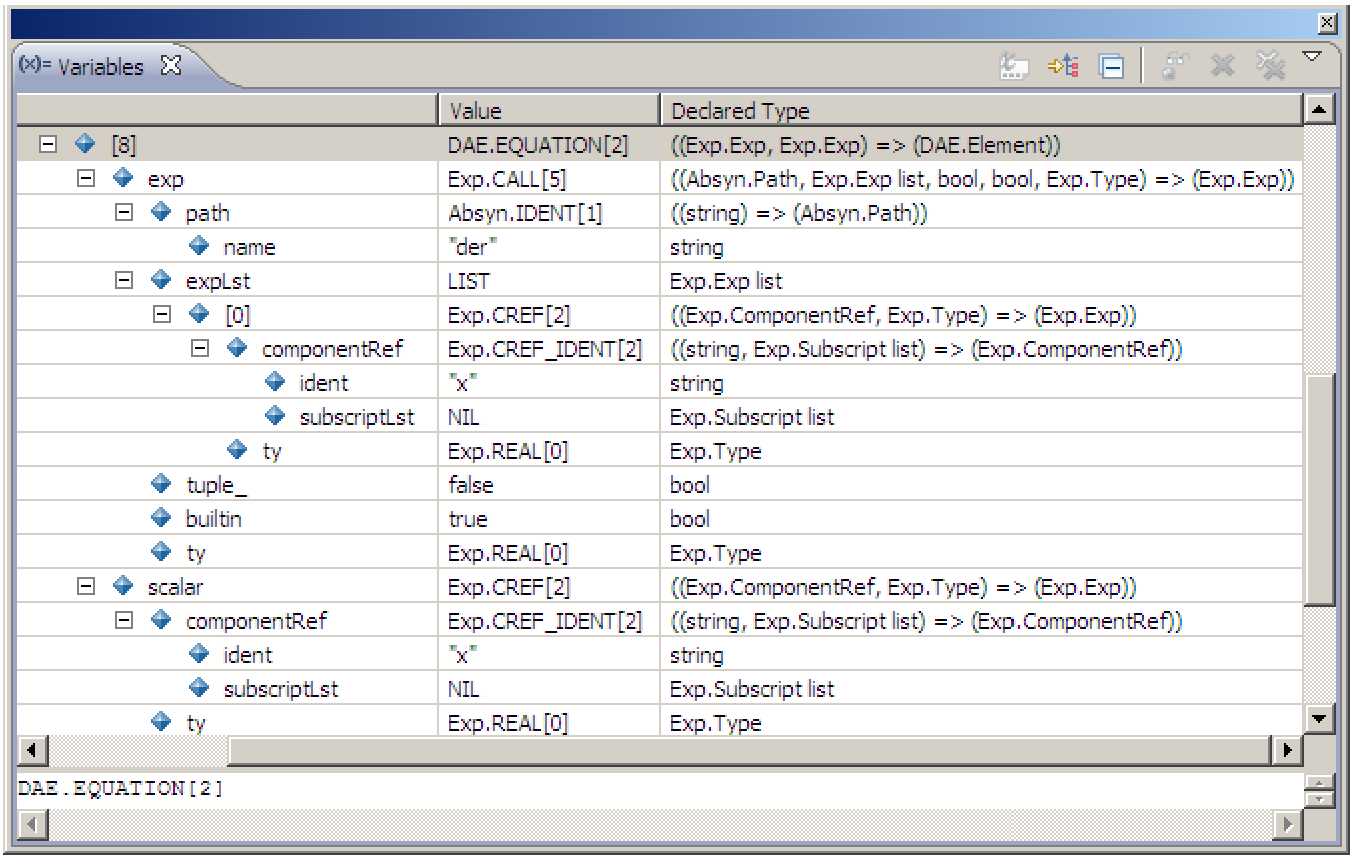


Figure 5-3. The type of **`DAE.Element`** for component **`Real x`**.
The type of stateSelect attribute is collapsed.

5.4 The first equation (der(x) = x) as a DAE.Element sub-tree



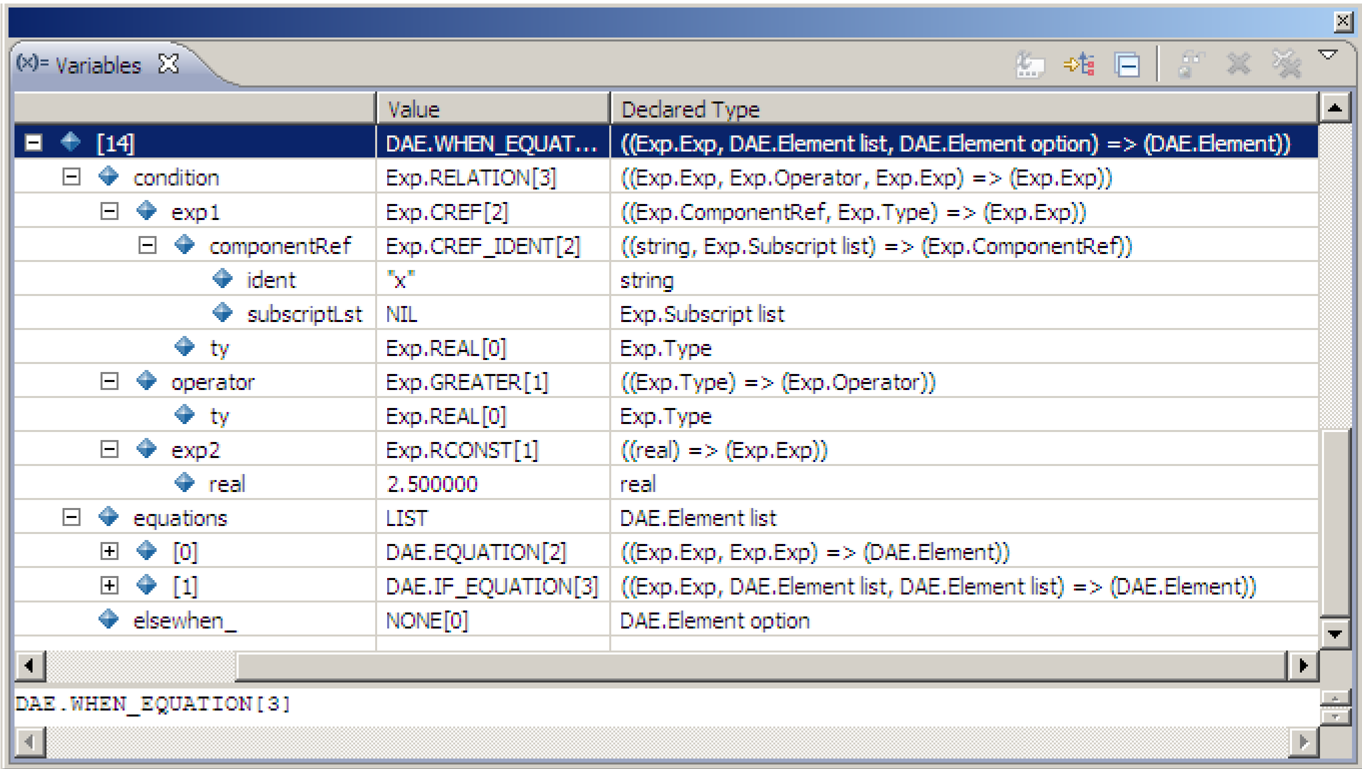
(x)= Variables

	Value	Declared Type
[8]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
exp	Exp.CALL[5]	((Absyn.Path, Exp.Exp list, bool, bool, Exp.Type) => (Exp.Exp))
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"der"	string
expLst	LIST	Exp.Exp list
[0]	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"x"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
tuple_	false	bool
builtin	true	bool
ty	Exp.REAL[0]	Exp.Type
scalar	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"x"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type

DAE.EQUATION[2]

Figure 5-4. The first equation (**`der(x) = x;`**) as **`DAE.EQUATION`** record within a **`DAE.Element`**.

5.5 The last equation (when x > 2.5 then …) as a DAE.Element sub-tree



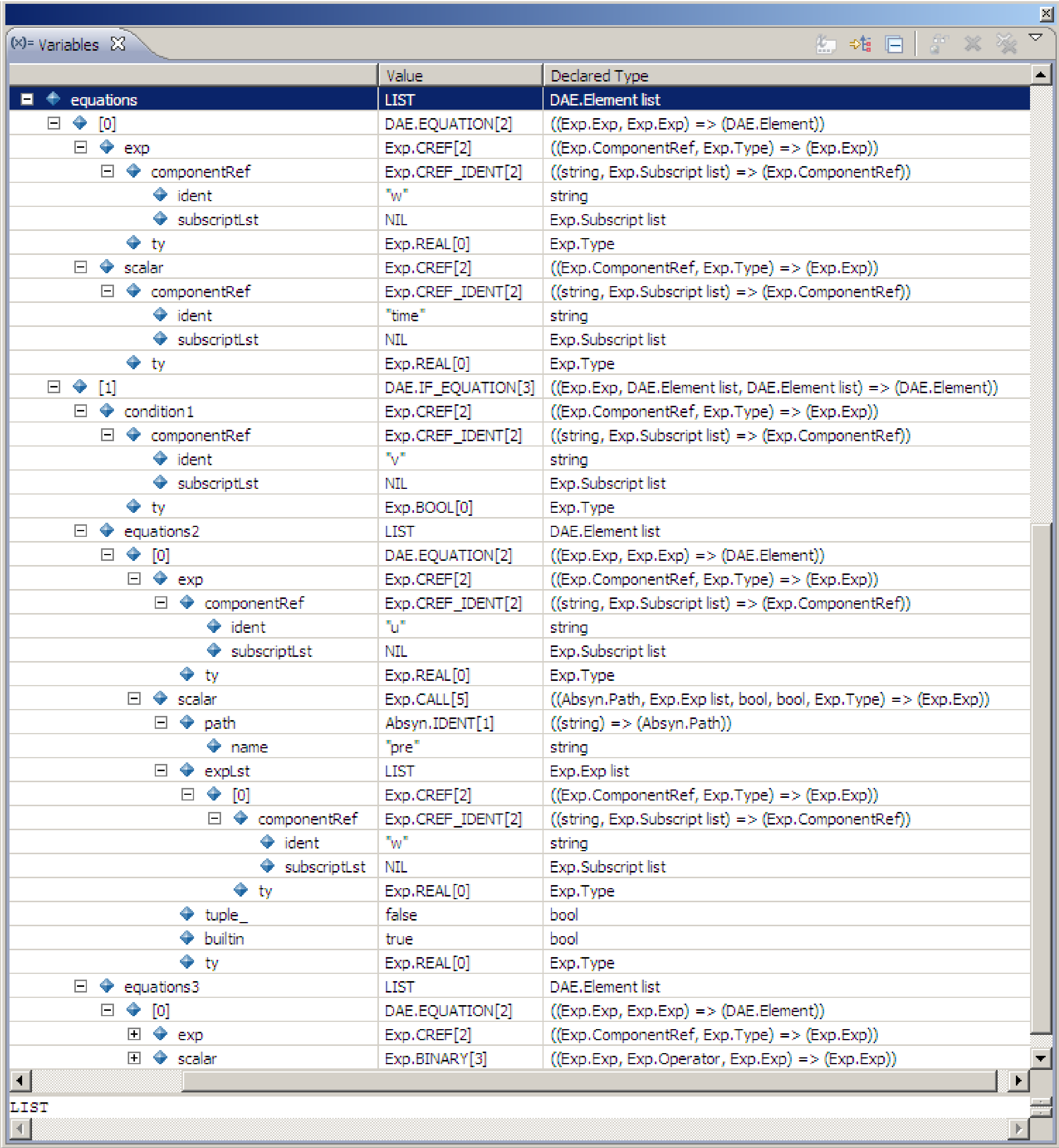
(x)= Variables

	Value	Declared Type
[14]	DAE.WHEN_EQUAT...	((Exp.Exp, DAE.Element list, DAE.Element option) => (DAE.Element))
condition	Exp.RELATION[3]	((Exp.Exp, Exp.Operator, Exp.Exp) => (Exp.Exp))
exp1	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"x"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
operator	Exp.GREATER[1]	((Exp.Type) => (Exp.Operator))
ty	Exp.REAL[0]	Exp.Type
exp2	Exp.RCONST[1]	((real) => (Exp.Exp))
real	2.500000	real
equations	LIST	DAE.Element list
[0]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
[1]	DAE.IF_EQUATION[3]	((Exp.Exp, DAE.Element list, DAE.Element list) => (DAE.Element))
elsewhen_	NONE[0]	DAE.Element option

DAE.WHEN_EQUATION[3]

Figure 5-5. The last equation (**`when x > 5 then ...`**) as **`DAE.EQUATION`** record within a **`DAE.Element`**.

5.5.1 The equations present in the when equation



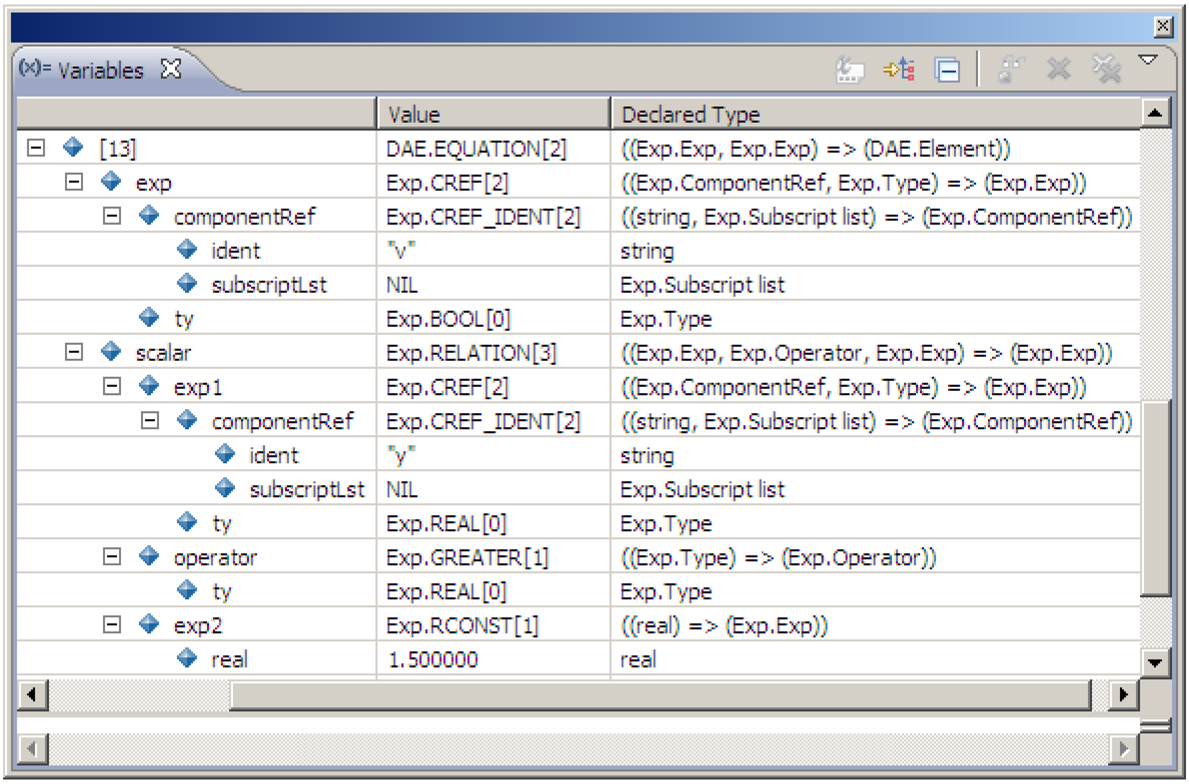
(x)= Variables

	Value	Declared Type
equations	LIST	DAE.Element list
[0]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
exp	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"w"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
scalar	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"time"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
[1]	DAE.IF_EQUATION[3]	((Exp.Exp, DAE.Element list, DAE.Element list) => (DAE.Element))
condition1	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"v"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.BOOL[0]	Exp.Type
equations2	LIST	DAE.Element list
[0]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
exp	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"u"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
scalar	Exp.CALL[5]	((Absyn.Path, Exp.Exp list, bool, bool, Exp.Type) => (Exp.Exp))
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"pre"	string
expLst	LIST	Exp.Exp list
[0]	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"w"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
tuple_	false	bool
builtin	true	bool
ty	Exp.REAL[0]	Exp.Type
equations3	LIST	DAE.Element list
[0]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
exp	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
scalar	Exp.BINARY[3]	((Exp.Exp, Exp.Operator, Exp.Exp) => (Exp.Exp))

LIST

Figure 5-6. The equation list present in the when equation.
The else part of the if equation (equations3) is collapsed.

5.6 The equation (v = y > 1.5;) as an DAE.EQUATION record



	Value	Declared Type
[13]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
exp	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"v"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.BOOL[0]	Exp.Type
scalar	Exp.RELATION[3]	((Exp.Exp, Exp.Operator, Exp.Exp) => (Exp.Exp))
exp1	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"y"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
operator	Exp.GREATER[1]	((Exp.Type) => (Exp.Operator))
ty	Exp.REAL[0]	Exp.Type
exp2	Exp.RCONST[1]	((real) => (Exp.Exp))
real	1.500000	real

Figure 5-7. The equation (**`v = y > 1.5;`**) as a **`DAE.EQUATION`** record of type **`DAE.Element`**.

Chapter 6

The DAE Package

6.1 The DAE.DAEList structure

```
uniontype DAElist
"A list of Elements. Variables, equations, functions, algorithms, etc. are all found here."
  record DAE
    list<Element> elementLst;
  end DAE;
end DAElist;
```

6.2 The DAE.Element structure

Parts of the DAE.Element uniontype is presented below. Check DAE.mo for the entire structure.

```
uniontype Element

  record VAR
    Exp.ComponentRef componentRef " The variable name";
    VarKind varible "varible kind" ;
    VarDirection variable "variable, constant, parameter, etc." ;
    VarProtection protection "if protected or public";
    Type input_ "input, output or bidir" ;
    Option<Exp.Exp> one "one of the builtin types" ;
    InstDims binding "Binding expression e.g. for parameters" ;
    Flow value "value of start attribute" ;
    list<Absyn.Path> flow_ "Flow of connector variable. Needed for unconnected flow variables";
    Option<VariableAttributes> variableAttributesOption;
    Option<Absyn.Comment> absynCommentOption;
    Absyn.InnerOuter innerOuter "inner/outer required to 'change' outer references";
    Types.Type fullType "Full type information required to analyze inner/outer elements";
  end VAR;

  record EQUATION "Scalar equation"
    Exp.Exp exp;
    Exp.Exp scalar ;
  end EQUATION;

  record WHEN_EQUATION " a when equation"
    Exp.Exp condition "Condition" ;
    list<Element> equations "Equations" ;
    Option<Element> elsewhen_ "Elsewhen should be of type WHEN_EQUATION" ;
  end WHEN_EQUATION;

  record IF_EQUATION " an if-equation"
    Exp.Exp condition1 "Condition" ;
    list<Element> equations2 "Equations of true branch" ;
    list<Element> equations3 "Equations of false branch" ;
  end IF_EQUATION;

...

end Element;
```

6.3 The function DAE.transformIfEqToExpr(DAE.DAEList)

This function transform if equations to if expressions. It searches for `DAE.IF_EQUATION` records inside the list of `DAE.Element` and transform them into `DAE.EQUATION` records containing `Exp.IF_EXP(expCond, expThen, expElse)`.

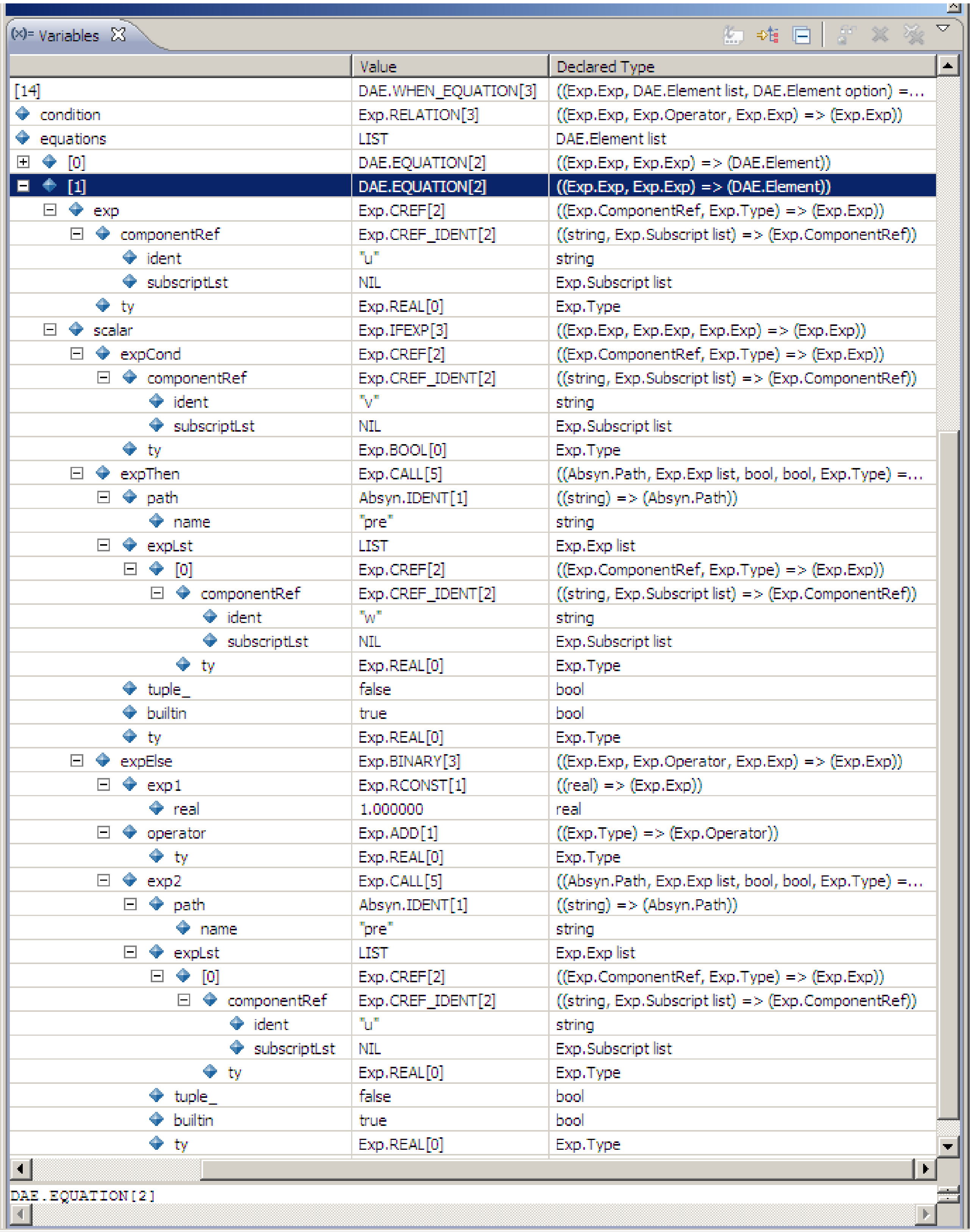
As an example, the if equation below (Figure 5-6):

```
if (v) then  // gets converted to u = if (v) then pre(w) else pre(u) + 1;
  u = pre(w);
else
  u = pre(u) + 1;
end if;
```

is transformed into the following equation (Figure 6-1):

```
u = if v then pre(w) else pre(u) + 1;
```

The resulting equation containing the if expression is presented below.



Variables

	Value	Declared Type
[14]	DAE.WHEN_EQUATION[3]	((Exp.Exp, DAE.Element list, DAE.Element option) =...
condition	Exp.RELATION[3]	((Exp.Exp, Exp.Operator, Exp.Exp) => (Exp.Exp))
equations	LIST	DAE.Element list
[0]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
[1]	DAE.EQUATION[2]	((Exp.Exp, Exp.Exp) => (DAE.Element))
exp	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"u"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
scalar	Exp.IFEXP[3]	((Exp.Exp, Exp.Exp, Exp.Exp) => (Exp.Exp))
expCond	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"v"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.BOOL[0]	Exp.Type
expThen	Exp.CALL[5]	((Absyn.Path, Exp.Exp list, bool, bool, Exp.Type) =...
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"pre"	string
expLst	LIST	Exp.Exp list
[0]	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"w"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
tuple_	false	bool
builtin	true	bool
ty	Exp.REAL[0]	Exp.Type
expElse	Exp.BINARY[3]	((Exp.Exp, Exp.Operator, Exp.Exp) => (Exp.Exp))
exp1	Exp.RCONST[1]	((real) => (Exp.Exp))
real	1.000000	real
operator	Exp.ADD[1]	((Exp.Type) => (Exp.Operator))
ty	Exp.REAL[0]	Exp.Type
exp2	Exp.CALL[5]	((Absyn.Path, Exp.Exp list, bool, bool, Exp.Type) =...
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"pre"	string
expLst	LIST	Exp.Exp list
[0]	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"u"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
tuple_	false	bool
builtin	true	bool
ty	Exp.REAL[0]	Exp.Type

DAE.EQUATION[2]

Figure 6-1. The if equation in the when equation is now transformed into an if expression.

Chapter 7

The DAELow Package

7.1 The DAELow.DAELow structure

```
uniontype DAELow
"THE LOWERED DAE consists of variables and equations. The variables are split into two lists,
 one for unknown variables states and algebraic and one for known variables constants and parameters.
 The equations are also split into two lists, one with simple equations, a=b, a-b=0, etc., that are
 removed from  the set of equations to speed up calculations."
  record DAELOW
    Variables orderedVars "orderedVars ; ordered Variables, only states and alg. vars" ;
    Variables knownVars "knownVars ; Known variables, i.e. constants and parameters" ;
    Variables externalObjects "External object variables";
    EquationArray orderedEqs "orderedEqs ; ordered Equations" ;
    EquationArray removedEqs "removedEqs ; Removed equations a=b" ;
    EquationArray initialEqs "initialEqs ; Initial equations" ;
    MultiDimEquation[:] arrayEqs "arrayEqs ; Array equations" ;
    Algorithm.Algorithm[:] algorithms "algorithms ; Algorithms" ;
    EventInfo eventInfo "eventInfo" ;
    ExternalObjectClasses extObjClasses "classes of external objects, contains constructor & destructor";
  end DAELOW;

end DAELow;
```

7.2 The DAELow.lower function

This function transforms a DAE.DAEList into a DAELow.DAELow representation.

```
// Transform the DAE representation into the DAELow representation.
// The DAELow representation splits the DAE into equations and variables
// and further divides variables into known and unknown variables and the
// equations into simple and non-simple equations.
daeLow = DAELow.lower(DAE.DAEList dae, Boolean addDummyState, Boolean simplify);
```

The result of application of function DAELow.lower(DAE.DAEList) is a DAELow.DAELow structure presented below.

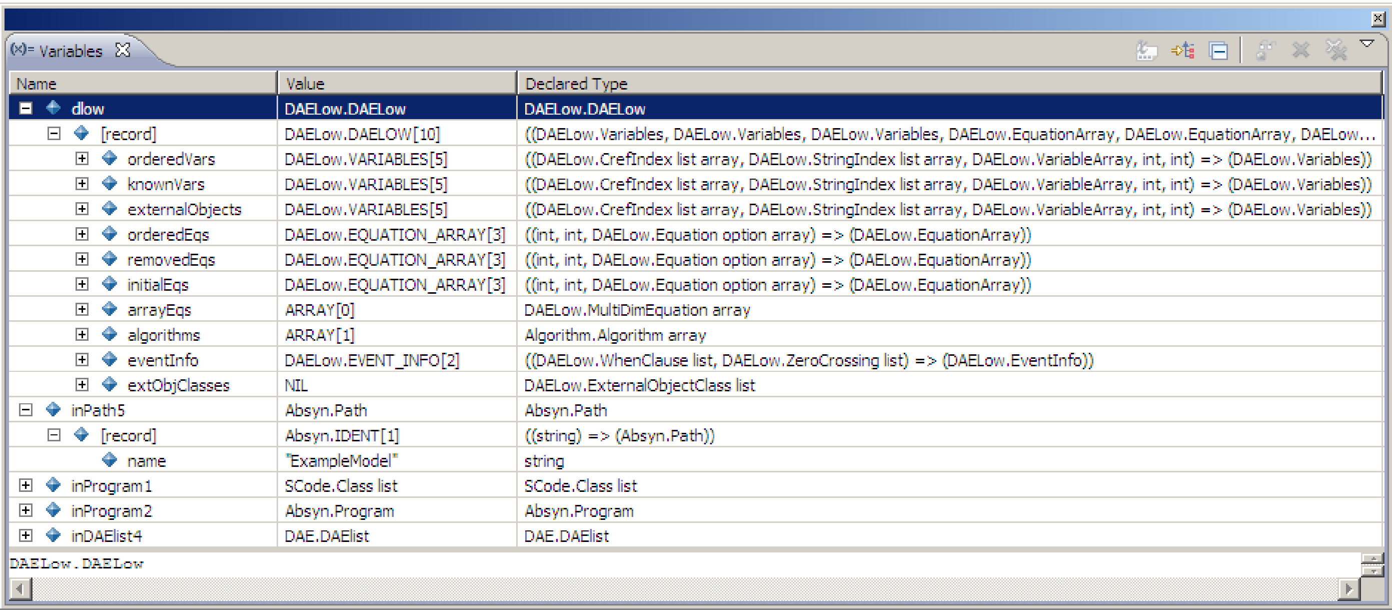
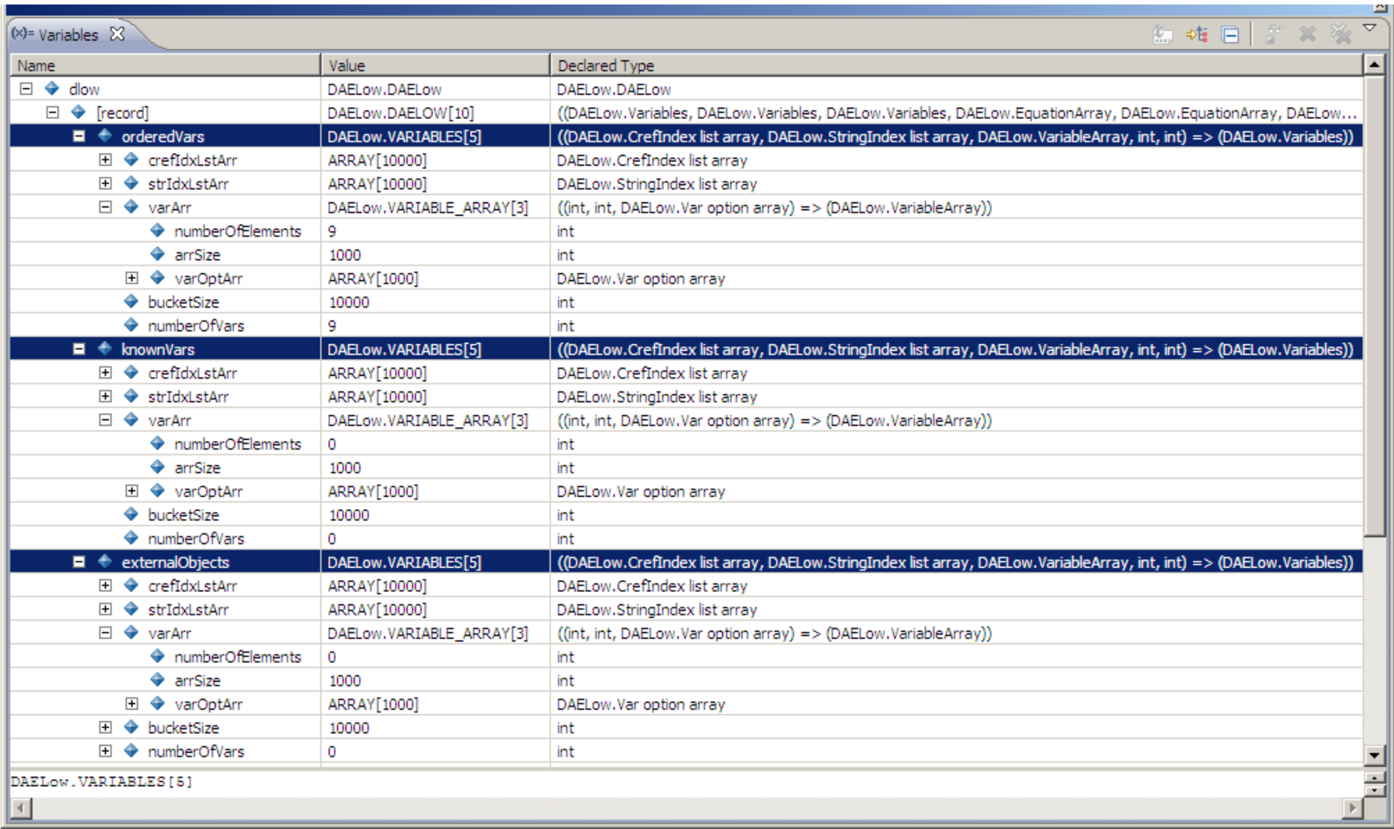


Figure 7-1. The **`DAELow.DAELow`** tree. The tree is collapsed.

7.3 The DAELow.Variables and DAELow.Var structures

```
uniontype Variables "- Variables"
  record VARIABLES
    list<CrefIndex>[:]   crefIdxLstArr "crefIdxLstArr ; HashTB, cref->indx" ;
    list<StringIndex>[:] strIdxLstArr  "strIdxLstArr ; HashTB, cref->indx for old names" ;
    VariableArray        varArr        "varArr ; Array of variables" ;
    Integer              bucketSize    "bucketSize ; bucket size" ;
    Integer              numberOfVars  "numberOfVars ; no. of vars" ;
  end VARIABLES;
end Variables;

uniontype Var "- Variables"
  record VAR
    Exp.ComponentRef varName "varName ; variable name" ;
    VarKind varKind "varKind ; Kind of variable" ;
    DAE.VarDirection varDirection "varDirection ; input, output or bidirectional" ;
    DAE.Type varType "varType ; builtin type or enumeration" ;
    Option<Exp.Exp> bindExp "bindExp ; Binding expression e.g. for parameters" ;
    Option<Values.Value> bindValue "bindValue ; binding value for parameters" ;
    DAE.InstDims arryDim "arryDim ; array dimensions on nonexpanded var" ;
    Integer index "index ; index in impl. vector" ;
    Exp.ComponentRef origVarName "origVarName ; original variable name" ;
    list<Absyn.Path> className "className ; classname variable belongs to" ;
    Option<DAE.VariableAttributes> values "values ; values on builtin attributes" ;
    Option<Absyn.Comment> comment "comment ; this contains the comment and annotation from Absyn" ;
    DAE.Flow flow_ "flow ; if the var is a flow" ;
  end VAR;
end Var;
```

7.4 DAELow.Variables: orderedVars, knownVars, externalObjects



Name	Value	Declared Type
dlow	DAELow.DAELow	DAELow.DAELow
[record]	DAELow.DAELOW[10]	((DAELow.Variables, DAELow.Variables, DAELow.Variables, DAELow.EquationArray, DAELow.EquationArray, DAELow...
orderedVars	DAELow.VARIABLES[5]	((DAELow.CrefIndex list array, DAELow.StringIndex list array, DAELow.VariableArray, int, int) => (DAELow.Variables))
crefIdxLstArr	ARRAY[10000]	DAELow.CrefIndex list array
strIdxLstArr	ARRAY[10000]	DAELow.StringIndex list array
varArr	DAELow.VARIABLE_ARRAY[3]	((int, int, DAELow.Var option array) => (DAELow.VariableArray))
numberOfElements	9	int
arrSize	1000	int
varOptArr	ARRAY[1000]	DAELow.Var option array
bucketSize	10000	int
numberOfVars	9	int
knownVars	DAELow.VARIABLES[5]	((DAELow.CrefIndex list array, DAELow.StringIndex list array, DAELow.VariableArray, int, int) => (DAELow.Variables))
crefIdxLstArr	ARRAY[10000]	DAELow.CrefIndex list array
strIdxLstArr	ARRAY[10000]	DAELow.StringIndex list array
varArr	DAELow.VARIABLE_ARRAY[3]	((int, int, DAELow.Var option array) => (DAELow.VariableArray))
numberOfElements	0	int
arrSize	1000	int
varOptArr	ARRAY[1000]	DAELow.Var option array
bucketSize	10000	int
numberOfVars	0	int
externalObjects	DAELow.VARIABLES[5]	((DAELow.CrefIndex list array, DAELow.StringIndex list array, DAELow.VariableArray, int, int) => (DAELow.Variables))
crefIdxLstArr	ARRAY[10000]	DAELow.CrefIndex list array
strIdxLstArr	ARRAY[10000]	DAELow.StringIndex list array
varArr	DAELow.VARIABLE_ARRAY[3]	((int, int, DAELow.Var option array) => (DAELow.VariableArray))
numberOfElements	0	int
arrSize	1000	int
varOptArr	ARRAY[1000]	DAELow.Var option array
bucketSize	10000	int
numberOfVars	0	int

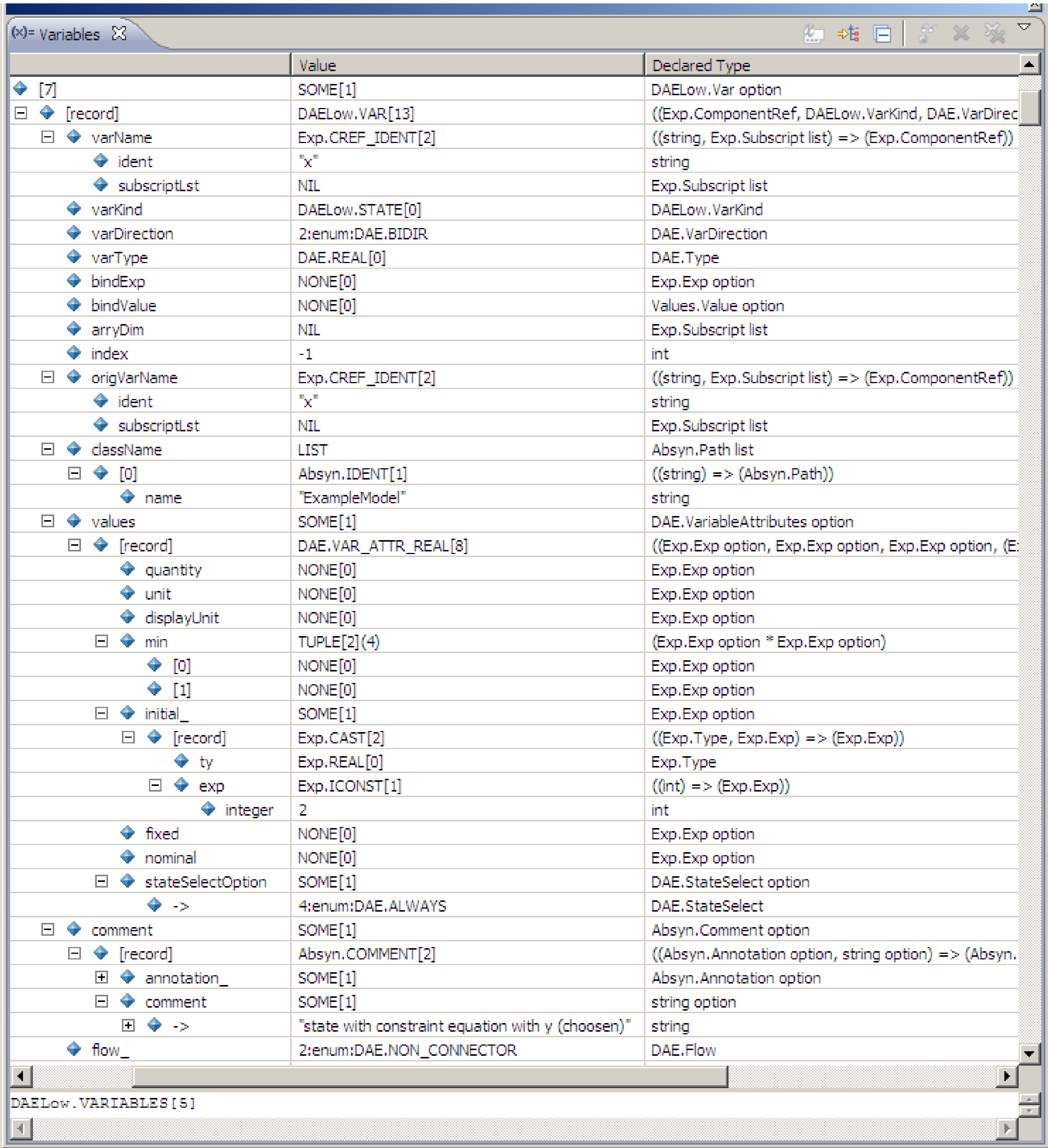
DAELow.VARIABLES[5]

Figure 7-2. DAELow.Variables: orderedVars, knownVars, externalObjects.

As one can see, the variables are split into several classes:

- orderedVars - Ordered variables, only states and algebraic variables.
- knownVars - Known variables, i.e. constants and parameters.
- externalObjects - External object variables.

7.4.1 The DAELow.Var tree describing Real x; component in the orderedVars structure.



	Value	Declared Type
[7]	SOME[1]	DAELow.Var option
[record]	DAELow.VAR[13]	((Exp.ComponentRef, DAELow.VarKind, DAE.VarDirec
varName	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"x"	string
subscriptLst	NIL	Exp.Subscript list
varKind	DAELow.STATE[0]	DAELow.VarKind
varDirection	2:enum:DAE.BIDIR	DAE.VarDirection
varType	DAE.REAL[0]	DAE.Type
bindExp	NONE[0]	Exp.Exp option
bindValue	NONE[0]	Values.Value option
arryDim	NIL	Exp.Subscript list
index	-1	int
origVarName	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"x"	string
subscriptLst	NIL	Exp.Subscript list
className	LIST	Absyn.Path list
[0]	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"ExampleModel"	string
values	SOME[1]	DAE.VariableAttributes option
[record]	DAE.VAR_ATTR_REAL[8]	((Exp.Exp option, Exp.Exp option, Exp.Exp option, (E:
quantity	NONE[0]	Exp.Exp option
unit	NONE[0]	Exp.Exp option
displayUnit	NONE[0]	Exp.Exp option
min	TUPLE[2](4)	(Exp.Exp option * Exp.Exp option)
[0]	NONE[0]	Exp.Exp option
[1]	NONE[0]	Exp.Exp option
initial_	SOME[1]	Exp.Exp option
[record]	Exp.CAST[2]	((Exp.Type, Exp.Exp) => (Exp.Exp))
ty	Exp.REAL[0]	Exp.Type
exp	Exp.ICONST[1]	((int) => (Exp.Exp))
integer	2	int
fixed	NONE[0]	Exp.Exp option
nominal	NONE[0]	Exp.Exp option
stateSelectOption	SOME[1]	DAE.StateSelect option
->	4:enum:DAE.ALWAYS	DAE.StateSelect
comment	SOME[1]	Absyn.Comment option
[record]	Absyn.COMMENT[2]	((Absyn.Annotation option, string option) => (Absyn.
annotation_	SOME[1]	Absyn.Annotation option
comment	SOME[1]	string option
->	"state with constraint equation with y (choosen)"	string
flow_	2:enum:DAE.NON_CONNECTOR	DAE.Flow

DAELow.VARIABLES[5]

Figure 7-3. The component (**`Real x;`**) in the **`DAELow.Var`** representation.

7.5 The DAELow structures for equation, algorithm and event

```
uniontype EquationArray "- Equation Array"

  record EQUATION_ARRAY
    Integer numberOfElement "numberOfElement ; no. elements" ;
    Integer arrSize "arrSize ; array size" ;
    Option<Equation>[:] equOptArr "equOptArr" ;
  end EQUATION_ARRAY;

end EquationArray;
```

```
uniontype MultiDimEquation "- Multi Dimensional Equation"

  record MULTIDIM_EQUATION
    list<Integer> dimSize "dimSize ; dimension sizes" ;
    Exp.Exp left "left ; lhs" ;
    Exp.Exp right "right ; rhs" ;
  end MULTIDIM_EQUATION;

end MultiDimEquation;


uniontype WhenClause "- When Clause"

  record WHEN_CLAUSE
    Exp.Exp condition                   "The when-condition";
    list<ReinitStatement> reinitStmtLst "List of reinit statements associated to the when clause.";
    Option<Integer> elseClause          "index of elsewhen clause";
    // HL only needs to know if it is an elsewhen the equations take care of which clauses are related.
    // The equations associated to the clause are linked to this when clause by the index in the
    // when clause list where this when clause is stored.
  end WHEN_CLAUSE;

end WhenClause;

uniontype ZeroCrossing "- Zero Crossing"

  record ZERO_CROSSING
    Exp.Exp relation_          "The zero crossing relation";
    list<Integer> occurEquLst  "List of equations where the function occurs";
    list<Integer> occurWhenLst "List of when clauses where the function occurs";
  end ZERO_CROSSING;

end ZeroCrossing;

uniontype EventInfo "- EventInfo"

  record EVENT_INFO
    list<WhenClause> whenClauseLst
    "List of when clauses. The WhenEquation datatype refer to this list by position";
    list<ZeroCrossing> zeroCrossingLst
    "List of zero crossings";
  end EVENT_INFO;

end EventInfo;
```

7.6 The equation, algorithm and event representation in DAELow

The equations are split into several classes:

- orderedEqs - Ordered equations
- removedEqs - Removed simple equations of the form a=b
- initialEqs - Initial equations
- arrayEqs - Array equations

As one can see for our example, there are:

- 9 equations in orderedEqs,
- 0 equations in removedEqs,
- no arrayEqs,
- no algorithms,
- 2 when clauses and 2 zero crossings in the eventInfo

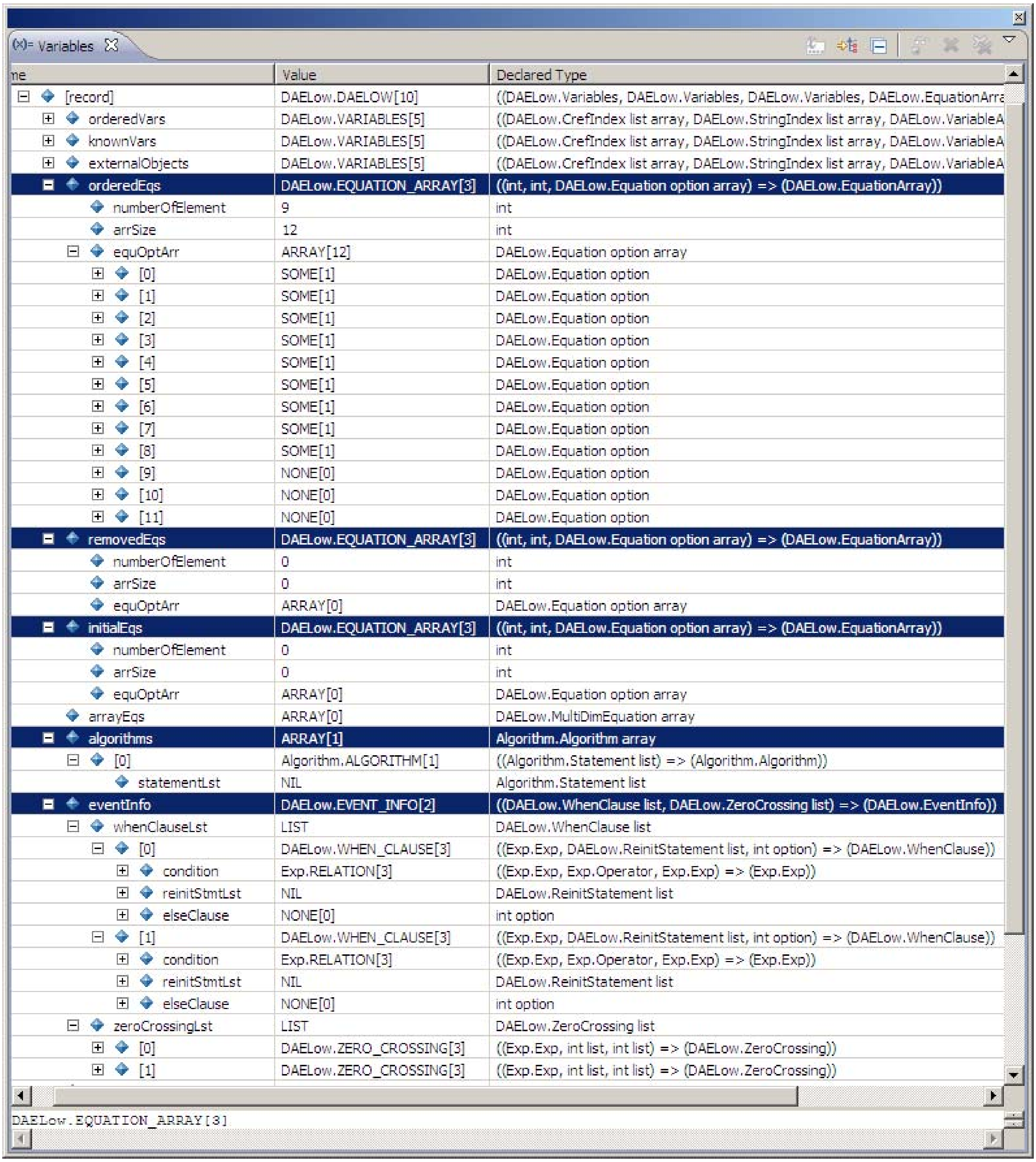


Figure 7-4. The representation of equation, algorithm and event in DAELow package.

7.6.1 The DAE when equation is translated into two DAELow.Equations

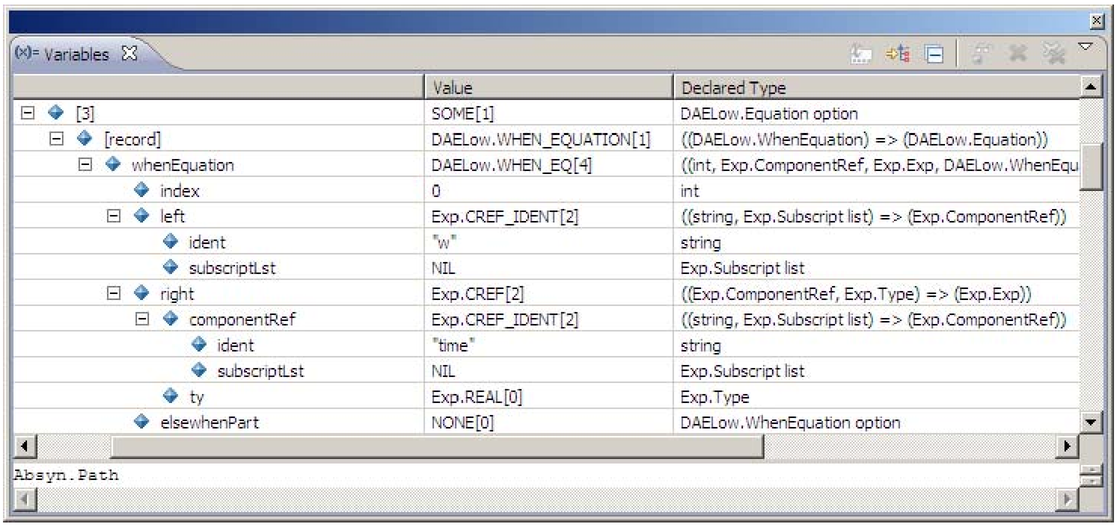
The DAE when equation:

```
when x > 2.5 then
  w = time;
  u = if v then pre(w) else pre(u) + 1;
end when;
```

is translated to two equations.

First DAELow.Equation (Figure 7-5):

```
when x > 2.5 then w = time; end when;
```



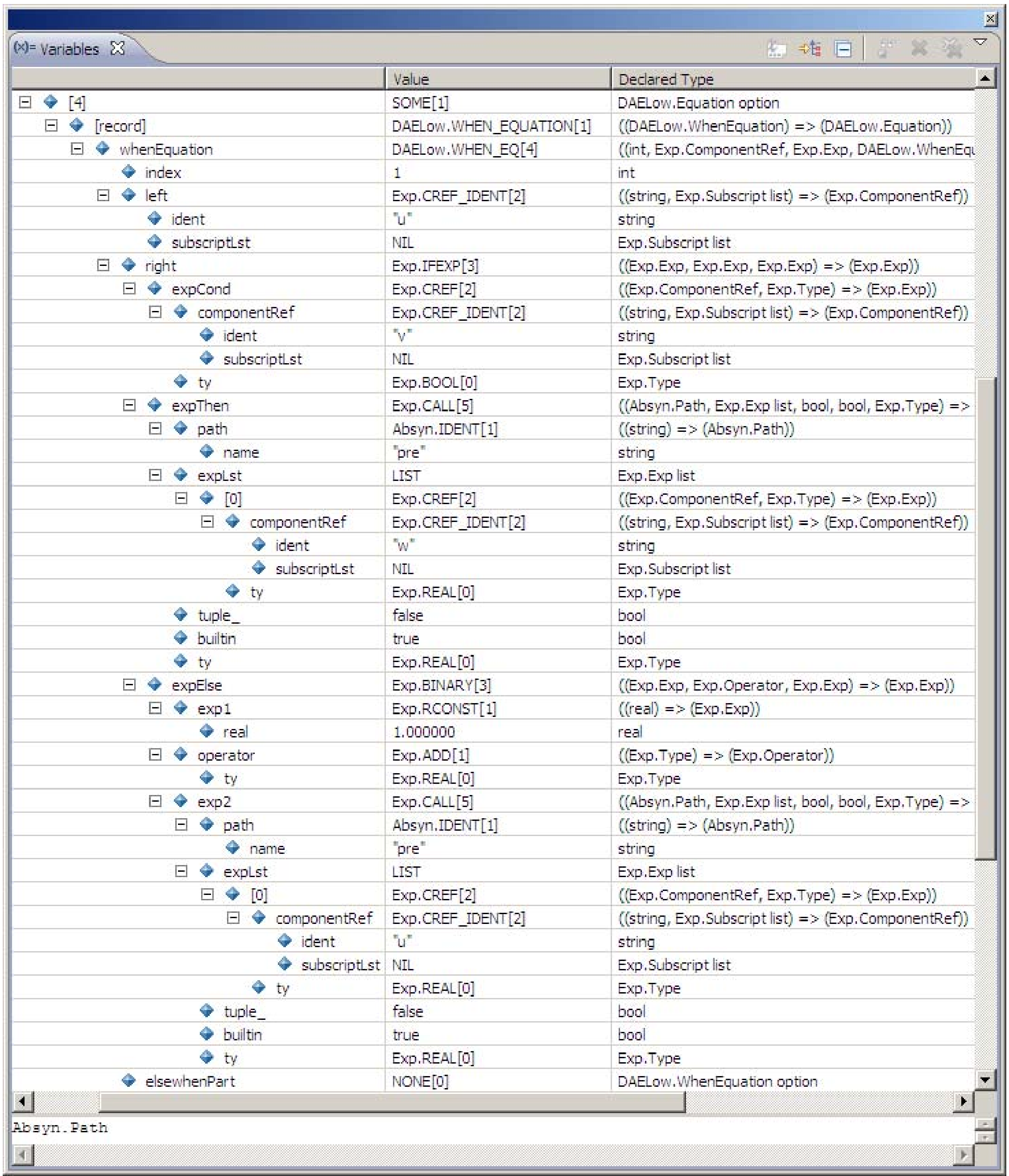
	Value	Declared Type
[3]	SOME[1]	DAELow.Equation option
[record]	DAELow.WHEN_EQUATION[1]	((DAELow.WhenEquation) => (DAELow.Equation))
whenEquation	DAELow.WHEN_EQ[4]	((int, Exp.ComponentRef, Exp.Exp, DAELow.WhenEqu
index	0	int
left	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"w"	string
subscriptLst	NIL	Exp.Subscript list
right	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"time"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
elsewhenPart	NONE[0]	DAELow.WhenEquation option

Absyn.Path

Figure 7-5. The first DAELow.Equation generated from DAE when equation.

Second DAELow.Equation (Figure 7-6):

```
when x > 2.5 then u = if v then pre(w) else pre(u) + 1; end when;
```



	Value	Declared Type
[4]	SOME[1]	DAELow.Equation option
[record]	DAELow.WHEN_EQUATION[1]	((DAELow.WhenEquation) => (DAELow.Equation))
whenEquation	DAELow.WHEN_EQ[4]	((int, Exp.ComponentRef, Exp.Exp, DAELow.WhenEq
index	1	int
left	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"u"	string
subscriptLst	NIL	Exp.Subscript list
right	Exp.IFEXP[3]	((Exp.Exp, Exp.Exp, Exp.Exp) => (Exp.Exp))
expCond	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"v"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.BOOL[0]	Exp.Type
expThen	Exp.CALL[5]	((Absyn.Path, Exp.Exp list, bool, bool, Exp.Type) =>
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"pre"	string
expLst	LIST	Exp.Exp list
[0]	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"w"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
tuple_	false	bool
builtin	true	bool
ty	Exp.REAL[0]	Exp.Type
expElse	Exp.BINARY[3]	((Exp.Exp, Exp.Operator, Exp.Exp) => (Exp.Exp))
exp1	Exp.RCONST[1]	((real) => (Exp.Exp))
real	1.000000	real
operator	Exp.ADD[1]	((Exp.Type) => (Exp.Operator))
ty	Exp.REAL[0]	Exp.Type
exp2	Exp.CALL[5]	((Absyn.Path, Exp.Exp list, bool, bool, Exp.Type) =>
path	Absyn.IDENT[1]	((string) => (Absyn.Path))
name	"pre"	string
expLst	LIST	Exp.Exp list
[0]	Exp.CREF[2]	((Exp.ComponentRef, Exp.Type) => (Exp.Exp))
componentRef	Exp.CREF_IDENT[2]	((string, Exp.Subscript list) => (Exp.ComponentRef))
ident	"u"	string
subscriptLst	NIL	Exp.Subscript list
ty	Exp.REAL[0]	Exp.Type
tuple_	false	bool
builtin	true	bool
ty	Exp.REAL[0]	Exp.Type
elsewhenPart	NONE[0]	DAELow.WhenEquation option

Absyn.Path

Figure 7-6. The second DAELow.Equation generated from DAE when equation.

7.7 The DAELow.incidenceMatrix and DAELow.transposeMatrix functions

```
// Calculate the incidence matrix from daeLow. In the incidence matrix the
// rows are equations and the columns are variables. A value of m[row, column] = 1
// is present if the variable represented by “column” appears in the equation
// represented by “row”. A value of m[row, column] = 0 appears if the variable
// represented by “column” is NOT present in the equation represented by “row”.
m = DAELow.incidenceMatrix(daeLow);
// Calculate the transpose of the incidence matrix
mT = DAELow.transposeMatrix(m);
```

The incidence matrix m is presented below:

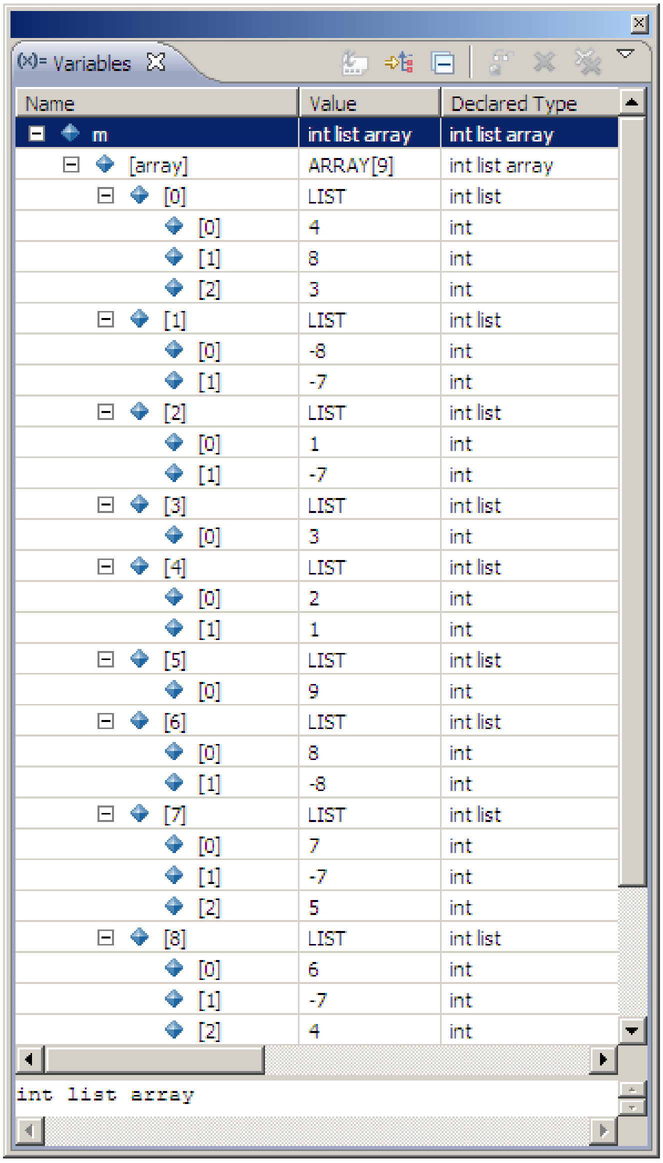


Figure 7-7. The incidence matrix m. The rows are equations indexes and the columns are variable indexes.

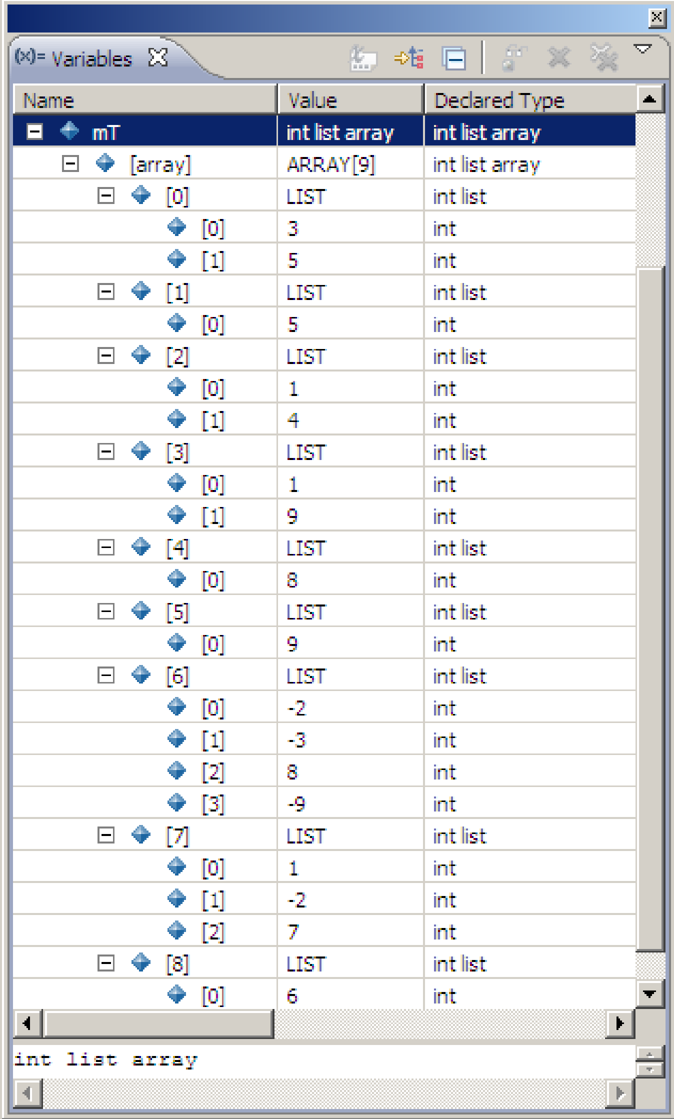


Figure 7-8. The transposed incidence matrix mT. The rows are variable indexes and the columns are equation indexes.

7.8 The DAELow.matchingAlgorithm and DAELow.strongComponents functions

```
// Apply the matching algorithm to the DAELow representation. This function performs
// the matching algorithm, which is the first part of sorting the equations into
// BLT (Block Lower Triangular) form. The matching algorithm finds a variable that
// is solved in each equation.  To find out which equations forms a block of equations,
// the second algorithm of the BLT sorting: strong components algorithm is run.
// The function returns the updated DAE in case of index reduction has added equations
// and variables, and the incidence matrix. The variable assignments is returned as a
// vector of variable indices, as well as its inverse, i.e. which equation a variable
// is solved in as a vector of equation indices.
// MatchingOptions contains options given to the algorithm.
//    - if index reduction should be used or not.
//    - if the equation system is allowed to be under constrained or not
//      which is used when generating code for initial equations.
(v1,v2,daeLowNew,m,mT) = DAELow.matchingAlgorithm(daeLow, m, mT,
  (DAELow.INDEX_REDUCTION(),DAELow.EXACT(),DAELow.REMOVE_SIMPLE_EQN()));
```

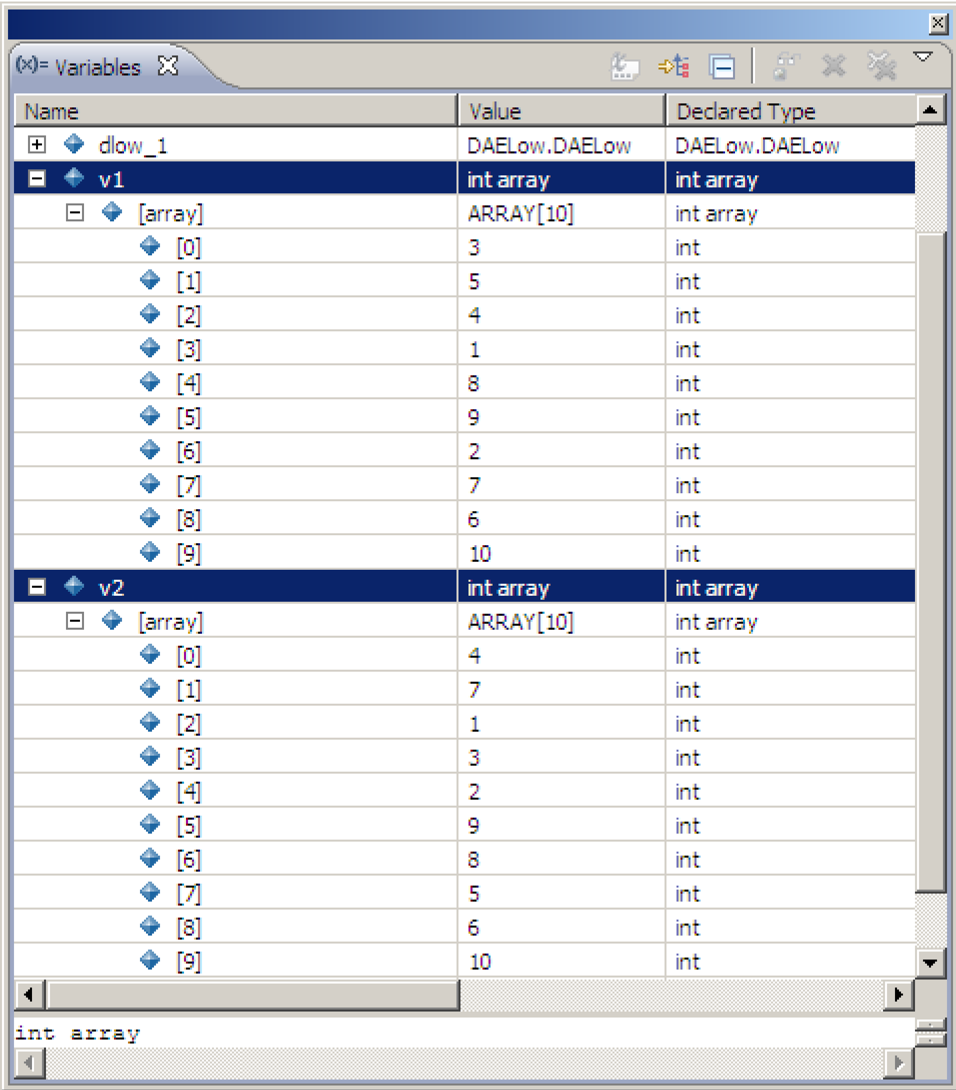


Figure 7-9. The v1 is an array of indexes of variables that tells which variable is solved in which equation. The v2 is an array of indexes of equations that tells which variable is solved in which equation.

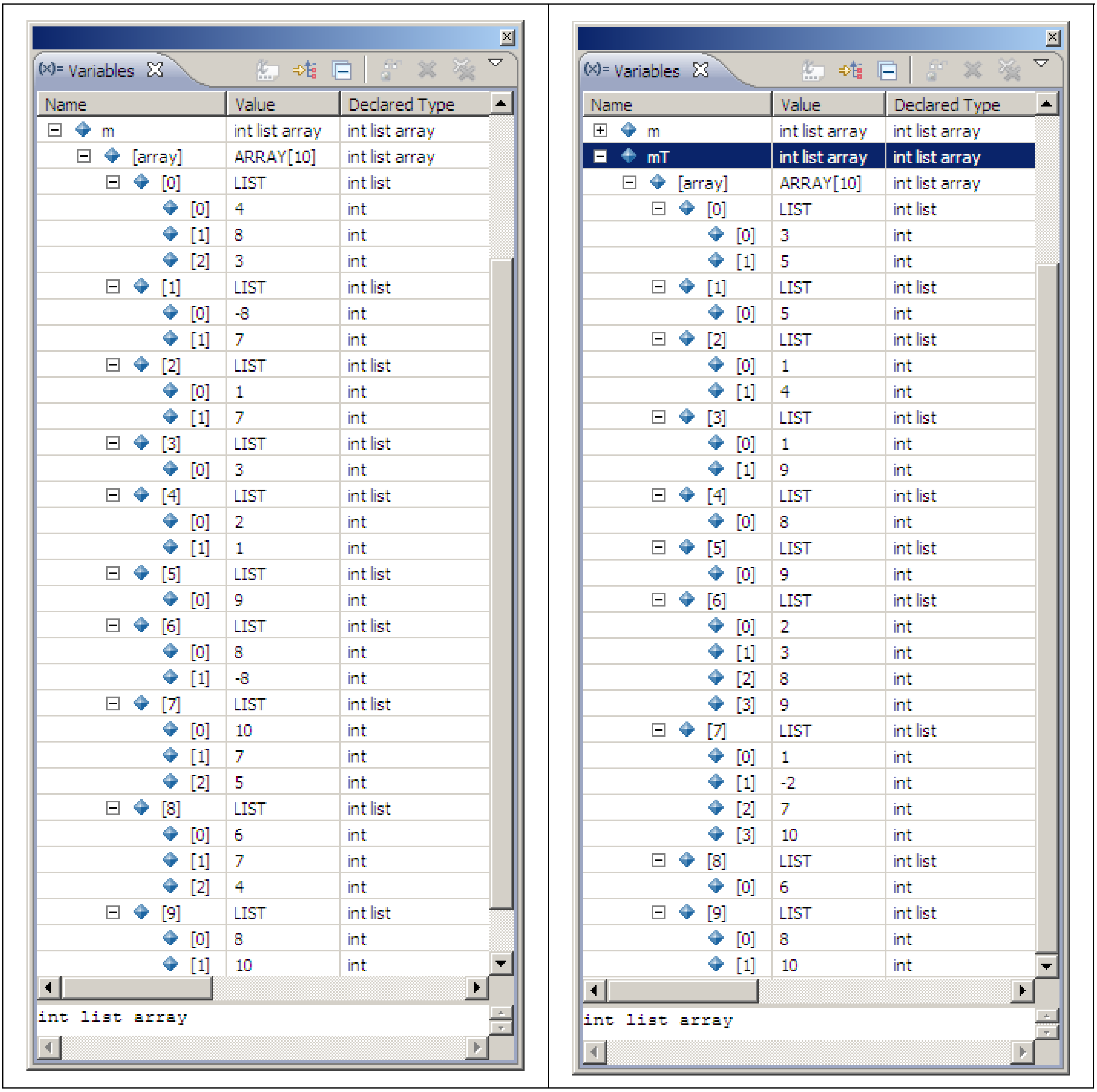


Figure 7-10. The new incidence matrix m and its transpose mT after the matching algorithm.

```
// Calculate the strong connected components of the incidence matrix.
// The strong connected components represent subsystems of equations
strongConnComp = DAELow.strongComponents(m, mT, v1, v2);
```

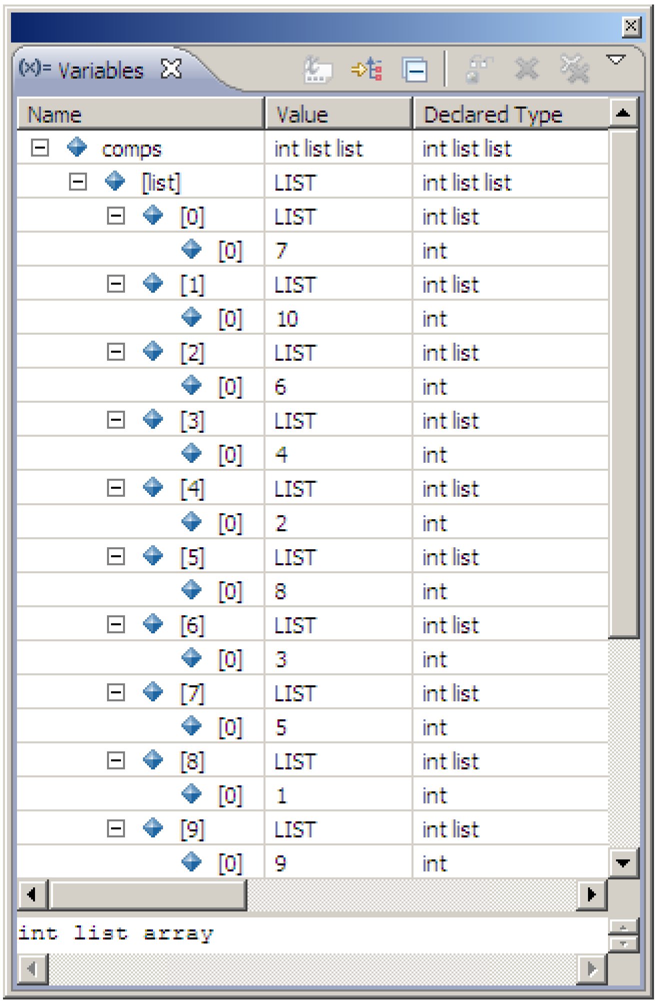


Figure 7-11. The strong connected components of the system of equations.

7.9 The output of omc +d=bltdump

Calling omc with +d=bltdump:

```
./omc +d=bltdump ExampleModel.mos
```

will generate the following output.

7.9.1 The output before the matching algorithm

```
Variables (9)
=========
1:  v:DISCRETE ExampleModel type: Boolean  indx = -1 fixed:false
2:  u:DISCRETE ExampleModel type: Real  indx = -1 fixed:false
3:  w:DISCRETE ExampleModel type: Real  indx = -1 fixed:false
4:  b:VARIABLE ExampleModel type: Real  indx = -1 fixed:false
5:  a:VARIABLE ExampleModel type: Real  indx = -1 fixed:false
6:  z:STATE ExampleModel type: Real  indx = -1 fixed:true
7:  y:STATE ExampleModel type: Real  indx = -1 fixed:true
8:  x:STATE ExampleModel type: Real  indx = -1 fixed:true
9:  $dummy:STATE  type: Real  indx = -1 fixed:true

Known Variables (constants) (0)
=============================

External Objects (0)
=============================

Classes of External Objects (0)
=============================
```

```
Equations (9)
=========
1 : b = der(x) + w
2 : x = 2.0 * y
3 : v = y > 1.5
4 : w := time when clause no:0
5 : u := if v then pre(w) else 1.0 + pre(u) when clause no:1
6 : der($dummy) = sin(time * 628.318530717)
7 : der(x) = x
8 : der(y) = y + a
9 : der(z) = y + b

Simple Equations (0)
=========
Initial Equations (0)
=========

Zero Crossings :
===============
y > 1.5 in equations [3] and when conditions []
x > 2.5 in equations [] and when conditions [1,2]

Array Equations :
===============

Algorithms:
===============

Incidence Matrix (row == equation)
====================================
number of rows: 10
01: 4 8 3
02: -8 7
03: 1 7
04: 3
05: 2 1
06: 9
07: 8 -8
08: 10 7 5
09: 6 7 4
10: 8 10

Transpose Incidence Matrix (row == var)
=====================================
number of rows: 10
01: 3 5
02: 5
03: 1 4
04: 1 9
05: 8
06: 9
07: 2 3 8 9
08: 1 -2 7 10
09: 6
10: 8 10
```

7.9.2 The output after the matching algorithm which also performs index reduction

```
Variables (10)
=========
01:  $v:DISCRETE ExampleModel type: Boolean  indx = 6 fixed:false
02:  $u:DISCRETE ExampleModel type: Real  indx = 5 fixed:false
03:  $w:DISCRETE ExampleModel type: Real  indx = 4 fixed:false
04:  $b:VARIABLE ExampleModel type: Real  indx = 3 fixed:false
05:  $a:VARIABLE ExampleModel type: Real  indx = 2 fixed:false
06:  $z:STATE ExampleModel type: Real  indx = 2 fixed:true
07:  $y:DUMMY_STATE ExampleModel type: Real  indx = 1 fixed:false
08:  $x:STATE ExampleModel type: Real  indx = 1 fixed:true
09:  $$dummy:STATE  type: Real  indx = 0 fixed:true
10:  $der$lPy$rP:DUMMY_DER ExampleModel type: Real indx = 0 fixed:false
```

```
Known Variables (constants) (0)
=============================
External Objects (0)
=============================
Classes of External Objects (0)
=============================

Equations (10)
=========
01 : $b = $derivative$x + $w
02 : $x = 2.0 * $y
03 : $v = $y > 1.5
04 : $w := time when clause no:0
05 : $u := if $v then pre($w) else 1.0 + pre($u) when clause no:1
06 : $derivative$$dummy = sin(time * 628.318530717)
07 : $derivative$x = $x
08 : $der$lPy$rP = $y + $a
09 : $derivative$z = $y + $b
10 : $derivative$x = 2.0 * $der$lPy$rP

Simple Equations (0)
=========
Initial Equations (0)
=========

Zero Crossings :
===============
$y > 1.5 in equations [3] and when conditions []
$x > 2.5 in equations [] and when conditions [1,2]

Array Equations :
===============
Algorithms:
===============

Matching
========
10 variables and equations
var 1 is solved in eqn 3
var 2 is solved in eqn 5
var 3 is solved in eqn 4
var 4 is solved in eqn 1
var 5 is solved in eqn 8
var 6 is solved in eqn 9
var 7 is solved in eqn 2
var 8 is solved in eqn 7
var 9 is solved in eqn 6
var 10 is solved in eqn 10

State blocks (dynamic section): Blocks
=======
{7}
{6}
{4}
{2}
{1}
{9}

Algebraic blocks (accepted section): Blocks
=======
{10}
{8}
{3}
{5}
```

7.10 The DAELow.translateDAE and DAELow.calculateValues functions

```
// Translates the dae so variables are indexed into different arrays:
//  - xd for derivatives, x for states
//  - dummy_der for dummy derivatives, dummy for dummy states
//  - y for algebraic variables, p for parameters
// This is done by creating defines for each variable.
//  - dots and subscripts in variable names are replaced: $P=".", $lB="[", $rB="]", etc.
//  - For instance, #define a$Pb$Pc xd[3]
// The equations are also updated with the new variable names.
indexedDaeLow1 = DAELow.translateDae(daeLow);
```

The result of the DAELow.translateDae(daeLow) will result the same DAELow structure but with the variable names changed. For example, below the DAELow.Var presented in Figure 7-3 will be transformed into:

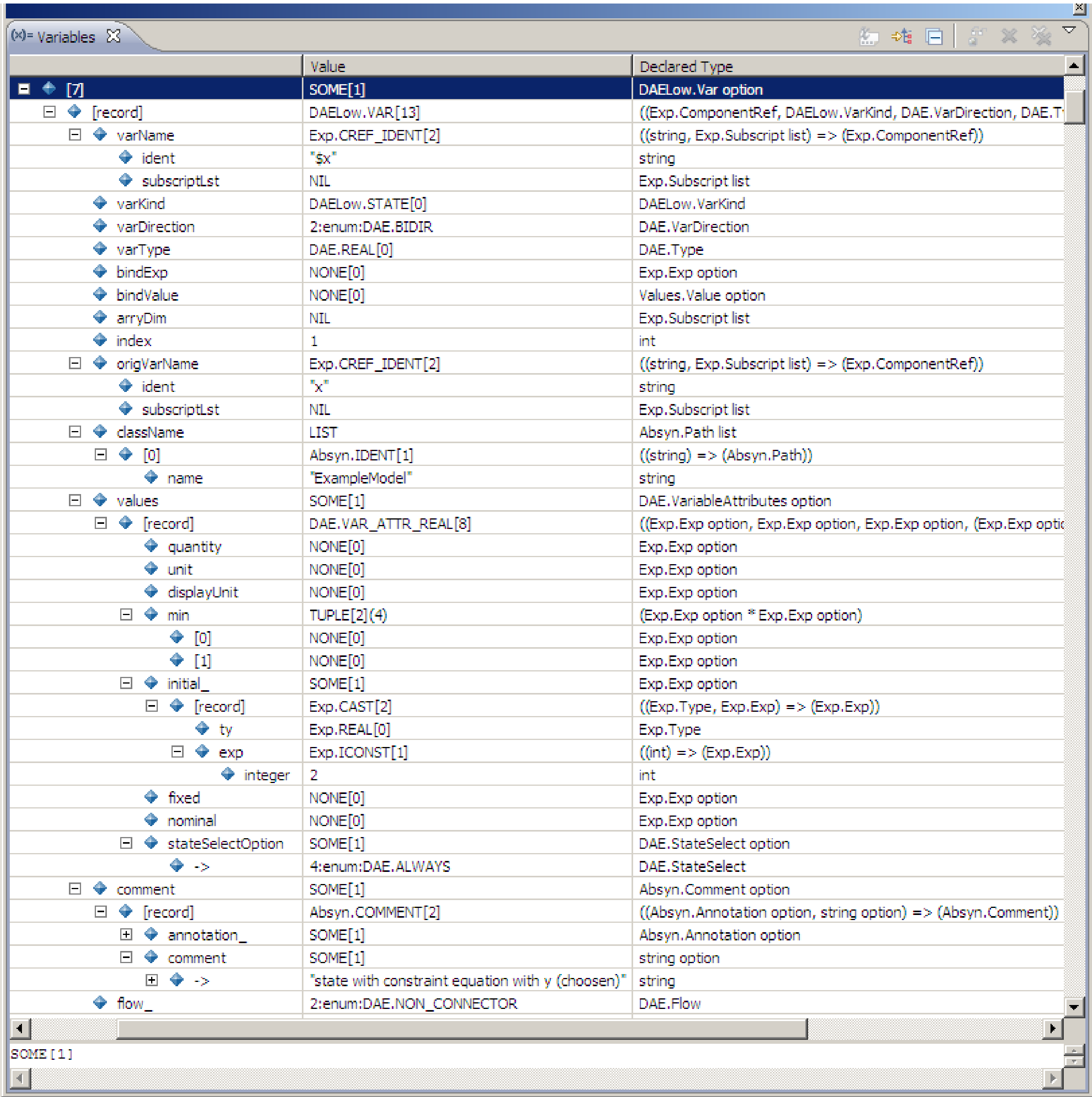


Figure 7-12. Variable name replacement in DAELow.Var.

```
// This function calculates the values from the parameter binding expressions.
// This is performed by building an environment and adding all the parameters
// and constants to it and then calling Ceval.ceval function to retrieve the
// constant values of each parameter or constant.
indexedDaeLow = DAELow.calculateValues(indexedDaeLow1);
```

As there are no parameters in this model, the indexedDaeLow variable will be equal to indexedDaeLow1.

Chapter 8

The SimCodegen Package

8.1 The preparation for generation of simulation code

In the preparation to generate the simulation code in function `Main.simcodegen` the following are generated:

```
// Transform the lastClassName path into a string
classNameStr = Absyn.pathString(lastClassName); // ExampleModel

// Generate files to hold the simulation code, functions, init values and a makefile
cppFileName  = stringAppend(classNameStr, ".cpp"); // ExampleModel.cpp
funcFileName = stringAppend(classNameStr, "_functions.cpp"); // ExampleModel_function.cpp
initFileName = stringAppend(classNameStr, "_init.txt"); // ExampleModel_init.txt
makeFileName = stringAppend(classNameStr, ".makefile"}); // ExampleModel.makefile

// Obtain the directory where to output the generated file
componentReference = Absyn.pathToCref(lastClassName); // Absyn.CREF(ExampleModel)
directoryOfFile = Ceval.getFileDir(componentReference, ast); // current directory
```

8.2 Function SimCodegen.generateFunctions

In our case this function generates an empty file ExampleModel_functions.cpp as there are no functions in our example.

```
// Generate the code for the functions and write it into funcFileName
libs = SimCodegen.generateFunctions(scode, dae, indexedDaeLow,
                                    lastClassName, funcFileName);
```

8.3 Function SimCodegen.generateSimulationCode

This particular function does not return any output, it will generate files on disk.

```
// Generate the simulation code and write it into cppFileName.
// Also, include funcFileName in the generated simulation code.
SimCodegen.generateSimulationCode(dae, indexedDaeLow, v1, v2, m, mT,
   strongConnComp, lastClassName, cppFileName, funcFileName, directoryOfFile);
```

8.3.1 Code for SimCodegen.generateSimulationCode

The following functions are called in this function:

```
cname = Absyn.pathString(lastClassName);
// This function traverses the equations to find out which blocks needs to
// be solved by the numerical solver (Dynamic Section) and which blocks only
// needs to be solved for output to file (Accepted Section).
// This is done by traversing the graph of strong components, where
// equations/variable pairs correspond to nodes of the graph. The edges of
// this graph are the dependencies between blocks or components.
// The traversal is made in the backward direction of this graph.
// The result is a split of the blocks into two lists.
(blt_states,blt_no_states) =
  DAELow.generateStatePartition(comps, dlow, ass1, ass2, m, mt);
```

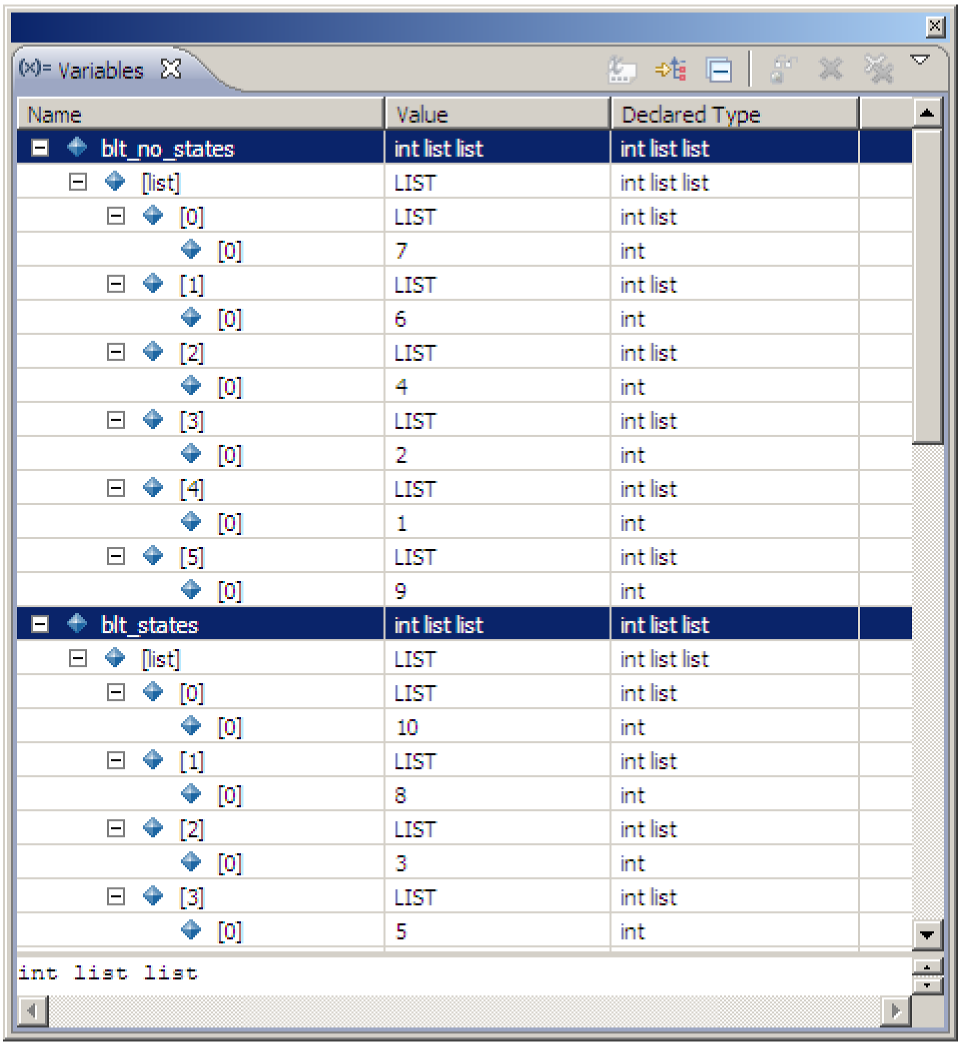


Figure 8-1. Partition of blt into states and no states.

```
(c_eventchecking,helpVarInfo1)                                                 =
   generateEventCheckingCode(dlow, comps, ass1, ass2, m, mt, class_);
(helpVarInfo,dlow2) = generateHelpVarsForWhenStatements(helpVarInfo1,dlow);
(out_str,n_o) = generateOutputFunctionCode(dlow2);
(in_str,n_i) = generateInputFunctionCode(dlow2);
n_h = listLength(helpVarInfo);
(s_code2,nres) = generateInitialValueCode2(dlow2,ass1,ass2);
(s_code3) = generateInitialBoundParameterCode(dlow2);
cglobal = generateGlobalData(class_, dlow2, n_o, n_i, n_h, nres,fileDir);

coutput = generateComputeOutput(cname, dae, dlow2, ass1, ass2,m,mt, blt_no_states);
cstate = generateComputeResidualState(cname, dae, dlow2, ass1, ass2, blt_states);

c_ode = generateOdeCode(dlow2, blt_states, ass1, ass2, m, mt, class_);
s_code = generateInitialValueCode(dlow2);
cwhen = generateWhenClauses(cname, dae, dlow2, ass1, ass2, comps);
czerocross = generateZeroCrossing(cname, dae, dlow2, ass1, ass2, comps, helpVarInfo);
(extObjIncludes,_) = generateExternalObjectIncludes(dlow2);
extObjInclude = Util.stringDelimitList(extObjIncludes,"\n");

res = Util.stringAppendList(
  {"//Simulation code for ",cname,
   "\n//Generated by OpenModelica.\n","\n#include \"modelica.h\"\n",
   "\n#include \"assert.h\"\n",
   "\n#include \"string.h\"\n",
   "\n#include \"simulation_runtime.h\"\n","\n#include \"",funcfilename,"\"\n",
   "extern \"C\" {\n",extObjInclude,"\n}\n",
   cglobal,coutput,in_str,out_str,
   cstate,czerocross,cwhen,c_ode,s_code,s_code2,s_code3,c_eventchecking});

System.writeFile(filename, res);
```

8.4 Function SimCodegen.generateInitData

```
// Generate the initialization assignments that contains values for parameters and
// the values for start attribute of variables and write them into initFileName.
SimCodegen.generateInitData(indexedDaeLow, lastClassName, classNameStr,
   initFileName,
   0.0 /* start */,
   1.0 /* stop */,
   500.0 /* intervals */,
   1e-10 /* tolerance*/,
   "dassl" /* method */);
```

8.4.1 Code for SimCodegen.generateInitData

```
// This function generates initial values for the
// simulation by investigating values of variables.
delta_time = stop -. start;             // start - stop
step = delta_time/.intervals;           // step = (start-stop)/intervals
start_str = realString(start);          // convert start into a string
stop_str = realString(stop);            // convert stop into a string
step_str = realString(step);            // convert step into a string
tolerance_str = realString(tolerance); // convert tolerance into a string
// Calculate the number of state variables, nx,
// the number of algebraic variables, ny
// and the number of parameters/constants, np.
// Outputs:
//    - Integer nx          "number of states";
//    - Integer ny          "number of alg. vars";
//    - Integer np          "number of parameters";
//    - Integer ng          "number of zerocrossings";
//    - Integer next        "number of external objects";
//    - Integer ny_string   "number of alg.vars which are strings";
//    - Integer np_string   "number of parameters which are strings";
(nx,ny,np,_,_,nystring,npstring) = DAELow.calculateSizes(dlow);
nx_str = intString(nx);
ny_str = intString(ny);
np_str = intString(np);
npstring_str = intString(npstring);
nystring_str = intString(nystring);
init_str = generateInitData2(dlow, nx, ny, np, nystring, npstring);
str = Util.stringAppendList({
   start_str," // start value\n",
   stop_str," // stop value\n",
   step_str," // step value\n",
   tolerance_str, " // tolerance\n",
   "\"",method,"\" // method\n",
   nx_str," // n states\n",
   ny_str," // n alg vars\n",
   np_str," //n parameters\n",
   npstring_str," // n string-parameters\n",
   nystring_str," // n string variables\n",
   init_str});
// write the created string into the initFileName
System.writeFile(initFileName, str);
```

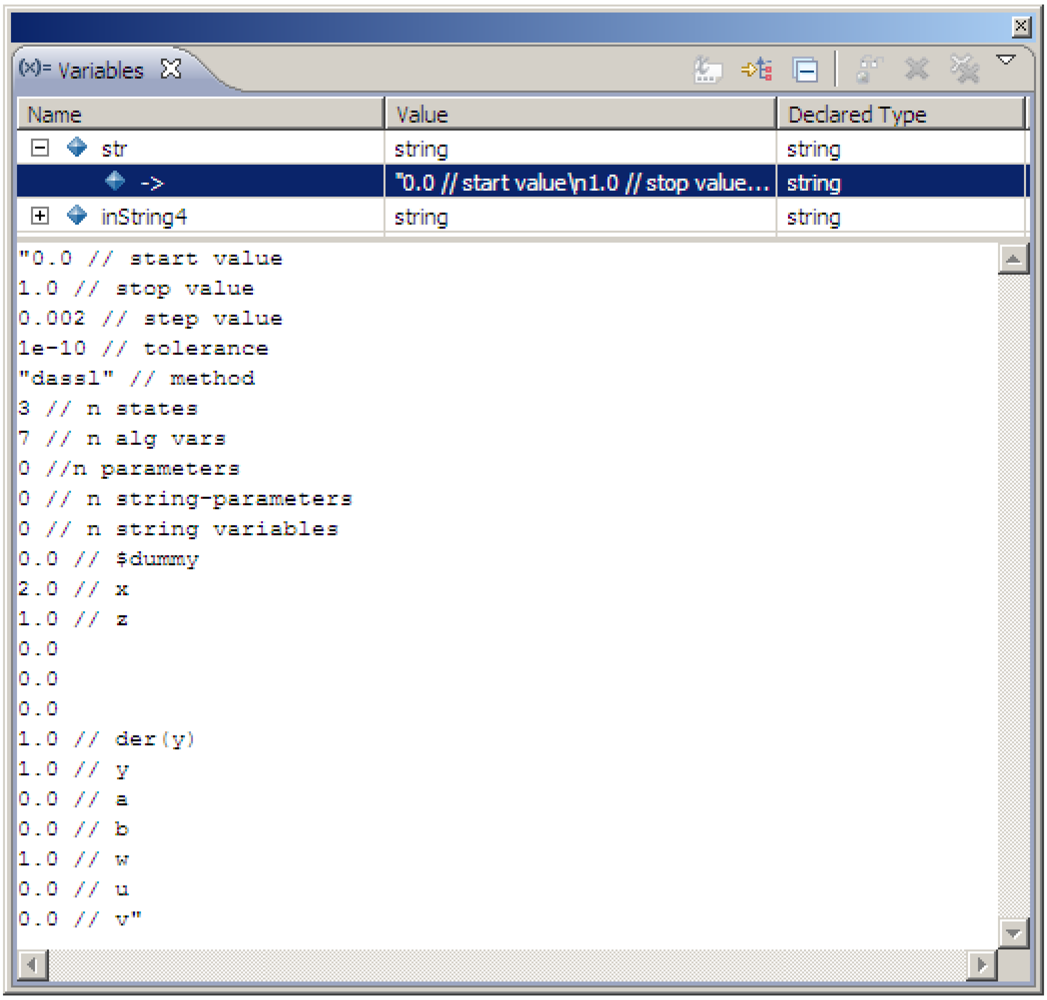


Figure 8-2. The created initialization data for the simulation.

8.5 Function SimCodegen.generateMakefile

```
// Generate the makefile to help build the executable for the code generation
SimCodegen.generateMakefile(makeFileName, classNameStr, libs, directoryOfFile);
```

8.5.1 Code for SimCodegen.generateMakefile

```
// The function Ceval.generateMakefile retrieves the compiler, linker, etc.
// settings for a certain platform and generates a makefile header:
//    ccompiler=System.getCCompiler();
//    cxxcompiler=System.getCXXCompiler();
//    linker=System.getLinker();
//    exeext=System.getExeExt();
//    dllext=System.getDllExt();
//    omhome=Settings.getInstallationDirectoryPath();
//    omhome=System.trim(omhome, "\"");
//    cflags=System.getCFlags();
//    ldflags=System.getLDFlags();
//    header=Util.stringAppendList(
//              {"#Makefile generated by OpenModelica\n\n",
//               "CC=",ccompiler,"\n", "CXX=",cxxcompiler,"\n",
//               "LINK=",linker,"\n", "EXEEXT=",exeext,"\n",
//               "DLLEXT=",dllext,"\n",
//               "CFLAGS= -I\"",omhome,"/include\" ", cflags,"\n",
//               "LDFLAGS= -L\"",omhome,"/lib\" ",ldflags,"\n"});
header = Ceval.generateMakefileHeader();

cpp_file = Util.stringAppendList({cname,".cpp"});
libs_1 = Util.stringDelimitList(libs, " ");
```

```
str = Util.stringAppendList(
   {header,
    "\n.PHONY: ", classNameStr, "\n",
    classNameStr,": ",cpp_file,"\n","\t $(CXX)",
    " $(CFLAGS)",
    " -I.",
    " -I\"",directoryOfFile,"\"",
    " -o ",classNameStr,"$(EXEEXT) ",cpp_file,
    " -L\"",directoryOfFile,"\"",
    " -lsim",
    " $(LDFLAGS)",
    " -lf2c",
    " ${SENDDATALIBS} ",
    libs_1,
    "\n"});

// write the created string into the file: Makefile
System.writeFile(makeFileName, str);
```

Chapter 9

The generated code

9.1 The ExampleModel.cpp file

```
//Simulation code for ExampleModel
//Generated by OpenModelica.

#include "modelica.h"
#include "assert.h"
#include "string.h"
#include "simulation_runtime.h"
#include "ExampleModel_functions.cpp"
extern "C" {
}

#define NHELP 2
#define NG 2//number of zero crossing
#define NX 3
#define NY 7
#define NP 0 // number of parameters
#define NO 0 // number of outputvar on topmodel
#define NI 0 // number of inputvar on topmodel
#define NR 9 // number of residuals for initialization function
#define NEXT 0 // number of external objects
#define MAXORD 5
#define NYSTR 0 // number of alg. string variables
#define NPSTR 0 // number of alg. string variables

static DATA* localData = 0;
#define time localData->timeValue
char *model_name="ExampleModel";
char *model_dir="";
char* state_names[3]={"$dummy", "x", "z"};
char* derivative_names[3]={"der($dummy)", "der(x)", "der(z)"};
char* algvars_names[7]={"der(y)", "y", "a", "b", "w", "u", "v"};
char* input_names[1] = {""};
char* output_names[1] = {""};
char* param_names[1] = {""};
char* string_alg_names[1] = {""};
char* string_param_names[1] = {""};

char* state_comments[3]={"",  "state with constraint equation with y (choosen)",  "state"};
char* derivative_comments[3]={"",  "state with constraint equation with y (choosen)",  "state"};
char* algvars_comments[7]={ "state with constraint equation with x (not choosen)",  "state with constraint
equation with x (not choosen)",  "output algebraic variable",  "algebraic variable",  "discrete variable",
"discrete output variable", ""};
char* input_comments[1] = {""};
char* output_comments[1] = {""};
char* param_comments[1] = {""};
char* string_param_comments[1] = {""};
char* string_alg_comments[1] = {""};

#define $v localData->algebraics[6]
#define $u localData->algebraics[5]
#define $w localData->algebraics[4]
#define $b localData->algebraics[3]
#define $a localData->algebraics[2]
#define $z localData->states[2]
#define $derivative$z localData->statesDerivatives[2]
#define $y localData->algebraics[1]
#define $x localData->states[1]
#define $derivative$x localData->statesDerivatives[1]
#define $$dummy localData->states[0]
#define $derivative$$dummy localData->statesDerivatives[0]
#define $der$lPy$rP localData->algebraics[0]
```

```
char* getName( double* ptr)
{
  if( &$v == ptr ) return algvars_names[6];
  if( &$u == ptr ) return algvars_names[5];
  if( &$w == ptr ) return algvars_names[4];
  if( &$b == ptr ) return algvars_names[3];
  if( &$a == ptr ) return algvars_names[2];
  if( &$derivative$z == ptr ) return derivative_names[2];
  if( &$z == ptr ) return state_names[2];
  if( &$y == ptr ) return algvars_names[1];
  if( &$derivative$x == ptr ) return derivative_names[1];
  if( &$x == ptr ) return state_names[1];
  if( &$derivative$$dummy == ptr ) return derivative_names[0];
  if( &$$dummy == ptr ) return state_names[0];
  if( &$der$lPy$rP == ptr ) return algvars_names[0];
  return "";
}

static char init_fixed[NX+NX+NY+NP]={1/*$dummy*/, 1/*x*/, 1/*z*/, 1/*default*/,
1/*default*/, 1/*default*/, 0/*der(y)*/,
0/*y*/, 0/*a*/, 0/*b*/,
0/*w*/, 0/*u*/, 0/*v*/};
char var_attr[NX+NY+NP]={/*$dummy:*/1+0, /*x:*/1+0, /*z:*/1+0, /*der(y):*/1+0,
/*y:*/1+0, /*a:*/1+0, /*b:*/1+0,
/*w:*/1+16, /*u:*/1+16, /*v:*/8+16};
#define DIVISION(a,b,c) ((b != 0) ? a / b : a / division_error(b,c))

int encounteredDivisionByZero = 0;
double division_error(double b,const char* division_str)
{
  if(!encounteredDivisionByZero){
    fprintf(stderr,"ERROR: Division by zero in partial equation: %s.\n",division_str);
    encounteredDivisionByZero = 1;
   }   return b;
}

void setLocalData(DATA* data)
{
   localData = data;
}

DATA* initializeDataStruc(DATA_FLAGS flags)
{
  DATA* returnData = (DATA*)malloc(sizeof(DATA));
  if(!returnData) //error check
    return 0;
  memset(returnData,0,sizeof(DATA));
  returnData->nStates = NX;
  returnData->nAlgebraic = NY;
  returnData->nParameters = NP;
  returnData->nInputVars = NI;
  returnData->nOutputVars = NO;
  returnData->nZeroCrossing = NG;
  returnData->nInitialResiduals = NR;
  returnData->nHelpVars = NHELP;
  returnData->stringVariables.nParameters = NPSTR;
  returnData->stringVariables.nAlgebraic = NYSTR;
  if(flags & STATES && returnData->nStates){
    returnData->states = (double*) malloc(sizeof(double)*returnData->nStates);
    returnData->oldStates = (double*) malloc(sizeof(double)*returnData->nStates);
    returnData->oldStates2 = (double*) malloc(sizeof(double)*returnData->nStates);
    assert(returnData->states&&returnData->oldStates&&returnData->oldStates2);
    memset(returnData->states,0,sizeof(double)*returnData->nStates);
    memset(returnData->oldStates,0,sizeof(double)*returnData->nStates);
    memset(returnData->oldStates2,0,sizeof(double)*returnData->nStates);
  }else{
    returnData->states = 0;
    returnData->oldStates = 0;
    returnData->oldStates2 = 0;
  }
  if(flags & STATESDERIVATIVES && returnData->nStates){
    returnData->statesDerivatives = (double*) malloc(sizeof(double)*returnData->nStates);
    returnData->oldStatesDerivatives = (double*) malloc(sizeof(double)*returnData->nStates);
    returnData->oldStatesDerivatives2 = (double*) malloc(sizeof(double)*returnData->nStates);
    assert(returnData->statesDerivatives&&returnData->oldStatesDerivatives &&
           returnData->oldStatesDerivatives2);
    memset(returnData->statesDerivatives,0,sizeof(double)*returnData->nStates);
    memset(returnData->oldStatesDerivatives,0,sizeof(double)*returnData->nStates);
    memset(returnData->oldStatesDerivatives2,0,sizeof(double)*returnData->nStates);
  }else{
    returnData->statesDerivatives = 0;
    returnData->oldStatesDerivatives = 0;
    returnData->oldStatesDerivatives2 = 0;
  }
```

```
if(flags & HELPVARS && returnData->nHelpVars){
  returnData->helpVars = (double*) malloc(sizeof(double)*returnData->nHelpVars);
  assert(returnData->helpVars);
  memset(returnData->helpVars,0,sizeof(double)*returnData->nHelpVars);
}else{
  returnData->helpVars = 0;
}
if(flags & ALGEBRAICS && returnData->nAlgebraic){
  returnData->algebraics = (double*) malloc(sizeof(double)*returnData->nAlgebraic);
  returnData->oldAlgebraics = (double*) malloc(sizeof(double)*returnData->nAlgebraic);
  returnData->oldAlgebraics2 = (double*) malloc(sizeof(double)*returnData->nAlgebraic);
  assert(returnData->algebraics&&returnData->oldAlgebraics&&returnData->oldAlgebraics2);
  memset(returnData->algebraics,0,sizeof(double)*returnData->nAlgebraic);
  memset(returnData->oldAlgebraics,0,sizeof(double)*returnData->nAlgebraic);
  memset(returnData->oldAlgebraics2,0,sizeof(double)*returnData->nAlgebraic);
}else{
  returnData->algebraics = 0;
  returnData->oldAlgebraics = 0;
  returnData->oldAlgebraics2 = 0;
  returnData->stringVariables.algebraics = 0;
}
if (flags & ALGEBRAICS && returnData->stringVariables.nAlgebraic) {
  returnData->stringVariables.algebraics =
    (char**)malloc(sizeof(char*)*returnData->stringVariables.nAlgebraic);
  assert(returnData->stringVariables.algebraics);
  memset(returnData->stringVariables.algebraics,0,sizeof(char*)*returnData->stringVariables.nAlgebraic);
} else {
  returnData->stringVariables.algebraics=0;
}  if(flags & PARAMETERS && returnData->nParameters){
  returnData->parameters = (double*) malloc(sizeof(double)*returnData->nParameters);
  assert(returnData->parameters);
  memset(returnData->parameters,0,sizeof(double)*returnData->nParameters);
}else{
  returnData->parameters = 0;
}
if (flags & PARAMETERS && returnData->stringVariables.nParameters) {
  returnData->stringVariables.parameters =
    (char**)malloc(sizeof(char*)*returnData->stringVariables.nParameters);
  assert(returnData->stringVariables.parameters);
  memset(returnData->stringVariables.parameters,0,sizeof(char*)*returnData->stringVariables.nParameters);
} else {
  returnData->stringVariables.parameters=0;
} if(flags & OUTPUTVARS && returnData->nOutputVars){
  returnData->outputVars = (double*) malloc(sizeof(double)*returnData->nOutputVars);
  assert(returnData->outputVars);
  memset(returnData->outputVars,0,sizeof(double)*returnData->nOutputVars);
}else{
  returnData->outputVars = 0;
}
if(flags & INPUTVARS && returnData->nInputVars){
  returnData->inputVars = (double*) malloc(sizeof(double)*returnData->nInputVars);
  assert(returnData->inputVars);
  memset(returnData->inputVars,0,sizeof(double)*returnData->nInputVars);
}else{
  returnData->inputVars = 0;
}
if(flags & INITIALRESIDUALS && returnData->nInitialResiduals){
  returnData->initialResiduals = (double*) malloc(sizeof(double)*returnData->nInitialResiduals);
  assert(returnData->initialResiduals);
  memset(returnData->initialResiduals,0,sizeof(double)*returnData->nInitialResiduals);
}else{
  returnData->initialResiduals = 0;
}
if(flags & INITFIXED){
  returnData->initFixed = init_fixed;
}else{
  returnData->initFixed = 0;
}

/*   names   */
if(flags & MODELNAME){
  returnData->modelName = model_name;
}else{
  returnData->modelName = 0;
}

if(flags & STATESNAMES){
  returnData->statesNames = state_names;
}else{
  returnData->statesNames = 0;
}
if(flags & STATESDERIVATIVESNAMES){
  returnData->stateDerivativesNames = derivative_names;
}else{
```

```
    returnData->stateDerivativesNames = 0;
  }
  if(flags & ALGEBRAICSNAMES){
    returnData->algebraicsNames = algvars_names;
  }else{
    returnData->algebraicsNames = 0;
  }
  if(flags & PARAMETERSNAMES){
    returnData->parametersNames = param_names;
  }else{
    returnData->parametersNames = 0;
  }
  if(flags & INPUTNAMES){
    returnData->inputNames = input_names;
  }else{
    returnData->inputNames = 0;
  }
  if(flags & OUTPUTNAMES){
    returnData->outputNames = output_names;
  }else{
    returnData->outputNames = 0;
  }

  /*   comments  */
  if(flags & STATESCOMMENTS){
    returnData->statesComments = state_comments;
  }else{
    returnData->statesComments = 0;
  }
  if(flags & STATESDERIVATIVESCOMMENTS){
    returnData->stateDerivativesComments = derivative_comments;
  }else{
    returnData->stateDerivativesComments = 0;
  }
  if(flags & ALGEBRAICSCOMMENTS){
    returnData->algebraicsComments = algvars_comments;
  }else{
    returnData->algebraicsComments = 0;
  }
  if(flags & PARAMETERSCOMMENTS){
    returnData->parametersComments = param_comments;
  }else{
    returnData->parametersComments = 0;
  }
  if(flags & INPUTCOMMENTS){
    returnData->inputComments = input_comments;
  }else{
    returnData->inputComments = 0;
  }
  if(flags & OUTPUTCOMMENTS){
    returnData->outputComments = output_comments;
  }else{
    returnData->outputComments = 0;
  }
  if (flags & EXTERNALVARS) {
    returnData->extObjs = (void**)malloc(sizeof(void*)*NEXT);
  if (!returnData->extObjs) {
     printf("error allocating external objects\n");
     exit(-2);
  }
  memset(returnData->extObjs,0,sizeof(void*)*NEXT);
  setLocalData(returnData); /* must be set since used by constructors*/
  }

  return returnData;

}

void deInitializeDataStruc(DATA* data, DATA_FLAGS flags)
{
  if(!data)
    return;
  if(flags & STATES && data->states){
    free(data->states);
    data->states = 0;
  }
  if(flags & STATESDERIVATIVES && data->statesDerivatives){
    free(data->statesDerivatives);
    data->statesDerivatives = 0;
  }
  if(flags & ALGEBRAICS && data->algebraics){
    free(data->algebraics);
    data->algebraics = 0;
  }
```

```c
  if(flags & PARAMETERS && data->parameters){
    free(data->parameters);
    data->parameters = 0;
  }
  if(flags & OUTPUTVARS && data->inputVars){
    free(data->inputVars);
    data->inputVars = 0;
  }
  if(flags & INPUTVARS && data->outputVars){
    free(data->outputVars);
    data->outputVars = 0;
  }
  if(flags & INITIALRESIDUALS && data->initialResiduals){
    free(data->initialResiduals);
    data->initialResiduals = 0;
  }
  if (flags & EXTERNALVARS && data->extObjs) {
    free(data->extObjs);
    data->extObjs = 0;
  }
}

int functionDAE_output()
{
  state mem_state;
  mem_state = get_memory_state();
  $der$lPy$rP = ((-$derivative$x) / -2.0);
  $a = ($der$lPy$rP - $y);
  restore_memory_state(mem_state);
  return 0;
}

int functionDAE_output2()
{
  state mem_state;
  mem_state = get_memory_state();
  restore_memory_state(mem_state);
  return 0;
}

/*
*/
int input_function()
{
  return 0;
}

int output_function()
{
  return 0;
}

int functionDAE_res(double *t, double *x, double *xd, double *delta,
                    long int *ires, double *rpar, long int* ipar)
{
  int i;
  double temp_xd[NX];
  double* statesBackup;
  double* statesDerivativesBackup;
  double timeBackup;

  statesBackup = localData->states;
  statesDerivativesBackup = localData->statesDerivatives;
  timeBackup = localData->timeValue;
  localData->states = x;
  for (i=0; i<localData->nStates; i++)
    temp_xd[i]=localData->statesDerivatives[i];

  localData->statesDerivatives = temp_xd;
  localData->timeValue = *t;

  functionODE();
  /* get the difference between the temp_xd(=localData->statesDerivatives) and xd(=statesDerivativesBackup)*/
  for (i=0; i < localData->nStates; i++)
    delta[i]=localData->statesDerivatives[i]-statesDerivativesBackup[i];

  localData->states = statesBackup;
  localData->statesDerivatives = statesDerivativesBackup;
  localData->timeValue = timeBackup;
  if (modelErrorCode) {
    if (ires) *ires = -1;  modelErrorCode =0;
  }
  return 0;
}
```

```
int function_zeroCrossing(long *neqm, double *t, double *x, long *ng, double *gout, double *rpar, long* ipar)
{
  double timeBackup;
  state mem_state;

  mem_state = get_memory_state();
  timeBackup = localData->timeValue;
  localData->timeValue = *t;
  functionODE();
  functionDAE_output();

  ZEROCROSSING(0,Greater($y,1.5));
  ZEROCROSSING(1,Greater($x,2.5));

  restore_memory_state(mem_state);
  localData->timeValue = timeBackup;
  return 0;
}

int handleZeroCrossing(long index)
{
  state mem_state;

  mem_state = get_memory_state();
  switch(index) {
    case 0:
      save($v);
      break;
    case 1:
      break;
    default: break;
  }
  restore_memory_state(mem_state);
  return 0;
}

int function_updateDependents()
{
  state mem_state;
  sin_rettype tmp0;
  modelica_boolean tmp1;
  modelica_boolean tmp2;
  modelica_boolean tmp3;

  inUpdate=initial()?0:1;

  mem_state = get_memory_state();
  $derivative$x = $x;
  $der$lPy$rP = ((-$derivative$x) / -2.0);
  tmp0 = sin((time * 628.318530717));
  $derivative$$dummy = tmp0;
  $y = ((-$x) / -2.0);
  $a = ($der$lPy$rP - $y);
  RELATIONGREATER(tmp1,$y,1.5);
  $v = tmp1;
  $b = ($derivative$x + $w);
  $derivative$z = ($y + $b);
  RELATIONGREATER(tmp2,$x,2.5);
  localData->helpVars[1] = tmp2;
  RELATIONGREATER(tmp3,$x,2.5);
  localData->helpVars[0] = tmp3;

  restore_memory_state(mem_state);
  inUpdate=0;
  return 0;
}

int function_when(int i)
{
  modelica_boolean tmp0;
  modelica_real tmp1;
  modelica_real tmp2;
  state mem_state;

  mem_state = get_memory_state();
  switch(i) {
     case 0: //when $x > 2.5
       save($w);
       $w = time;
       break;
     case 1: //when $x > 2.5
       save($u);
       tmp0 = $v;
       if (tmp0) { tmp1 = pre($w); }
```

```
        else { tmp2 = pre($u); }
        $u = ((tmp0)?tmp1:(1.0 + tmp2));
        break;
     default: break;
  }

  restore_memory_state(mem_state);
  return 0;
}

int functionODE()
{
  state mem_state;
  sin_rettype tmp0;

  mem_state = get_memory_state();
  $derivative$x = $x;
  tmp0 = sin((time * 628.318530717));
  $derivative$$dummy = tmp0;
  $y = ((-$x) / -2.0);
  $b = ($derivative$x + $w);
  $derivative$z = ($y + $b);

  restore_memory_state(mem_state);
  return 0;

}

int initial_function()
{
  return 0;
}

int initial_residual()
{
  sin_rettype tmp0;
  int i=0;
  state mem_state;


  mem_state = get_memory_state();
  localData->initialResiduals[i++] = ($b - ($derivative$x + $w));
  localData->initialResiduals[i++] = ($x - (2.0 * $y));
  tmp0 = sin((time * 628.318530717));
  localData->initialResiduals[i++] = ($derivative$$dummy - tmp0);
  localData->initialResiduals[i++] = ($derivative$x - $x);
  localData->initialResiduals[i++] = ($der$lPy$rP - ($y + $a));
  localData->initialResiduals[i++] = ($derivative$z - ($y + $b));
  localData->initialResiduals[i++] = ($derivative$x - (2.0 * $der$lPy$rP));
  localData->initialResiduals[i++] = ($z - 1.0);
  localData->initialResiduals[i++] = ($x - 2.0);

  restore_memory_state(mem_state);
  return 0;
}

int bound_parameters()
{
  state mem_state;
  mem_state = get_memory_state();
  restore_memory_state(mem_state);
  return 0;
}

int checkForDiscreteVarChanges()
{
  int needToIterate=0;
  if (edge(localData->helpVars[1])) AddEvent(0 + localData->nZeroCrossing);
  if (edge(localData->helpVars[0])) AddEvent(1 + localData->nZeroCrossing);
  if (change($v)) { needToIterate=1; }
  if (change($u)) { needToIterate=1; }
  if (change($w)) { needToIterate=1; }
  for (long i = 0; i < localData->nHelpVars; i++) {
    if (change(localData->helpVars[i])) { needToIterate=1; }
  }
  return needToIterate;
}
```

9.2 The ExampleModel.makefile

```
#Makefile generated by OpenModelica
```

```
CC=gcc
CXX=g++
LINK=gcc -shared -export-dynamic
EXEEXT=.exe
DLLEXT=.dll
CFLAGS= -I"C:/OpenModelica/build/include" -Wall -msse2 -mfpmath=sse ${MODELICAUSERCFLAGS}
LDFLAGS= -L"C:/OpenModelica/build/lib" -lc_runtime

.PHONY: ExampleModel
ExampleModel: ExampleModel.cpp
	$(CXX) $(CFLAGS) -I. -o ExampleModel$(EXEEXT) ExampleModel.cpp -lsim $(LDFLAGS) -lf2c ${SENDDATALIBS}
```