**Planning and design for non-expanded-arrays**

**1. Planning**

**1.1. Plan**

***Week 43, OpenModelica Developers Week (Linköping):***

*Meeting 27/10*: Adrian Pop, Per Östlund, Kristian Stavåker, and Alexey Lebedev had a meeting (~1 hour) where the outline of the work and which branch to work on were discussed.

Merge the branch with non-expanded arrays by Alexey:

<https://openmodelica.ida.liu.se/svn/OpenModelica/branches/NonExpandedArrays>

into:

<https://openmodelica.ida.liu.se/svn/OpenModelica/branches/sjoelund-functiontree>

Per and Alexey work together to do the merging.

Kristian Stavåker will look into how to not expand for-equations and how to do the matching without expanding arrays (KS has a meeting with Jens about the matching).

The general idea is to generate a DAE that is has arrays/equations not expanded (needed by us and Equa) and then have a phase to translate it into a DAE that is expanded (because MostForWater needs it).

*Meeting 27/10*: Kristian Stavåker and Jens Frenkel had a meeting (2-3 hours) where the modifications of the matching and sorting algorithms were discussed. Hand compilation of examples on paper.

*Meeting 29/10*: Kristian Stavåker and Jens Frenkel had a meeting (2-3 hours) where it was discussed exactly which functions in the compiler are to be modified. (***More information to be added to document shortly***)

***Week 44:***

Contact between Kristian Stavåker and Jens Frenkel via e-mail and Skype regarding matching and sorting of non-expanded array equations.

**2. Design principles**

**2.1. Backward compatibility**

The compiler must have arrays non-expanded if and only if it has been launched with +a flag. If this flag has not been set, the compiler must work as before. This leads to the following guidelines:

The cases in matchcontinue which process non-expanded arrays should be guarded by checks of the function RTOpts.splitArays(). This function returns **false** if +a flag has been set and **true** otherwise.

It is preferable to preserve compiler’s data types whenever possible. Changing types makes it harder to guarantee backward compatibility since functions processing such types have to be changed as well. However, sometimes it is unavoidable.

It is also preferable to preserve types of input/output variables of functions, though this is less important since easier to control.

**3. COMPILER FRONT-END: Current OMC implementation**

(This is some information for those who are not very familiar with the way OMC works.)

OMC front-end produces what is essentially flat Modelica in the form of elements of the type DAE.DAElist . Even though DAE stands for “differential algebraic equations”, elements of these lists (of the type DAE.Element) can also represent variables as records DAE.VAR. This record contains variable’s subscripts as a list of DAE.Subscript.

The uniontype DAE.Subscript (which is used not only in DAE.VAR) can be one of three records: INDEX (which is supposed to represent an index), SLICE (which is supposed to represent a range of indices) and WHOLEDIM (which is supposed to represent the whole index range available for a given variable and a given index position, like the first position in the expression a[:,1]).

All the DAE.VAR produced by the front-end as variables within a model (not within a function) represent scalar variables (in particular, scalar elements of arrays) and have subscripts of the form INDEX with index expression being an integer constant.

When instantiating a variable, the uniontype Prefix.ComponentPrefix is used to represent the variable whose subcomponent the variable being instantiated is. For example, in the model

model M

class A

Real b;

end A;

A[2] a;

end M;

the variable a is instantiated with an empty prefix, the variable b is first instantiated with prefix a[1], then with prefix a[2].

Subscripts of Prefix.ComponentPrefix are represented by a list of integers.

**4. COMPILER FRONT-END: Implemented (at least, partially) design changes**

**4.1. Representation of subscripts of non-expanded arrays.** Subscripts of non-expanded array variables are represented by

DAE.SLICE(exp=DAE.RANGE(ty=DAE.ET\_INT, exp=DAE.ICONST(1), expOption=NONE(), range=dim)),

where dim is the dimension (of type DAE.Exp). This requires change of the type of the 14th input variable in the functions Inst.instVar, instVar\_dispatch, instVar2 and 15th input variable in the function instArray from list<Integer> to list<DAE.Subscript> (=Inst.InstDims). Changes in these functions (mostly in instVar2) are also needed of course.

Comments:

Alexey: 1)Maybe, SLICE(RANGE(1:dim)) is an overkill, and SLICE(dim) would be enough. (It will be easy to change the program this way.) At least, it would be enough not to confuse non-expanded array variables with scalar elements of arrays. On the other hand, it could be useful for the future development of Modelica – for example, if one day Modelica allows array ranges not to start with 1 or to have step different from 1.

2) How should enumeration ranges be represented? I have not worked on this yet.

**4.2. Evaluation of dimensions.** The dimension expressions are evaluated if it is possible. They are left unevaluated only if they cannot be evaluated. It means that, for example, the model

model M

parameter Integer p=4;

Real r[p];

end M;

produces variable r with subscript SLICE(RANGE(1:4)), while the model

model M

parameter Integer p;

Real r[p];

end M;

produces variable r with subscript SLICE(RANGE(1:p)).

Comments:

Alexey: It may be preferable to keep leave dimensions unevaluated even when they can be evaluated. For example, it would make it easier to check that the model is balanced for all values of parameters.

**4.3. Prefix subscripts.** The type of subscripts variable in Prefix.ComponetPrefix is changed from list<Integer> to list<DAE.Exp>. Functions in PrefixUtil package are modified accordingly.

**5. COMPILER FRONT-END: Further changes**

**5.1. Subscripts/dimensions type changes.** It will probably be needed to change types of variables representing subscripts or array dimensions from Integers to more complex types in the following places: Values.ARRAY, DAE.ARRAY\_EQUATION, DAE.INITIAL\_ARRAY\_EQUATION.

**6. COMPILER MIDDLE: Implementation design**

**7. COMPILER BACK-END (code generator): Implementation design**