CHAPTER 13 CREATING CONDITIONAL MODELS IN ASCEND

In this chapter, we describe how one can create conditional models in the ASCEND environment.

what is a conditional model ?

Formally, we consider as a conditional model any problem in which the domain of validity of alternatives sets of equations depends on one or more discrete conditions; conditions can be expressed in terms of any logical, integer, or binary variables or constants. For instance, think of a case in which you need to solve a system of equations including some sort of numerical correlation (correlation data for physical properties, for instance). You realize that the coefficients of your correlation change with the value of some other variables of the problem (temperature, pressure, etc.). You have a conditional model.

ASCEND support three modeling capabilities for the efficient development of conditional models:

- Conditional configuration of the model structure.
- Conditional compilation.
- Conditional execution of the procedural code of methods.

In the following sections we describe the modeling tools for the performance of each of these tasks: the WHEN statements for the conditional configuration of a model structure, the SELECT statement for conditional compilation, and the SWITCH statement for conditional execution of procedural statements.

13.1 THE WHEN STATEMENT: CONDITIONAL CONFIGURATION OF THE MODEL STRUCTURE

We start by defining the syntax for the WHEN statement:

```
eq1_identifier: definition_of_equation_1;
model1_identifier: definition_of_model_1;
```

```
      ⋮
   modeln_identifier: definition_of_model_n;

   WHEN (list_of_variables)
      CASE list_of_values_1:
               USE eq1_identifier;
      CASE list_of_values_2:
               USE model1_identifier;
      ⋮
      OTHERWISE:
               USE modeln_identifier;
   END WHEN;
```

observations about the WHEN statement

The following are important observations about the WHEN statement:

1 A list of variables is used to define the applicability of each of the alternative configuration. The variables in this list can be of any type among boolean, integer or symbol or any combination of them. Note that the list is surrounded by rounded parentheses: (). We do that to emphasize that order matter in such a list — consistent with the use of rounded parentheses throughout ASCEND.

2 The values in this list for each of the cases are in one to one correspondence with the variables in the list.

3 Names of arrays of models or equations are allowed inside the scope of each CASE.

4 All the objects and equations used in the different CASEs of a WHEN statement are compiled. However, the objects (the variables and relations defined in it) of a particular CASE will only become part of our mathematical problem if the values in the list of values of that CASE match the current values of the variables in the list of variables. Practically speaking, to "USE" an object (model) means that the variables and equations contained in that object will become an active part of the system of nonlinear equations representing the current configuration of the problem.

There are two different ways in which the WHEN statement can be used.:

select among alternatives

- First, the WHEN statement can be used to select a configuration of a problem among several alternative configurations. This chapter is mainly concerned with this type of simpler and more common application.

conditional program

- Second, in combination with logical relations, the WHEN statement can be used for conditional programming — that is, a problem in which the system of equations to be solved depends on the solution of the problem.

13.1.1 THE SIMPLEST EXAMPLE

Assume that you want to solve a system of equations in which two correlations are possible for the calculation of a variable. Of course, you could create two simple models, each of them including one of the alternative equations. You could also use the WHEN statement to create only one model, in which you could include both alternatives. In this latter case you will be able to switch readily from one alternative to the other without recompiling. Look at the following simple case:

```
laminar  IS_A boolean;
Re,f     IS_A factor;

invariant: sqrt(f) * Re = 0.00034576;
low_flow: Re = 64/f;
high_flow: Re = (0.206307/f)^4;

WHEN (laminar)
   CASE TRUE:
      USE low_flow;
   CASE FALSE:
      USE high_flow;
END WHEN;
```

The model contains three equations, all of which are compiled. There is one equation (named `invariant`) which is not used in any of the CASEs of the WHEN statement. Such an equation is always part of the mathematical problem that we are trying to represent. On the other hand, the equations `low_flow` and `high_flow` are conditional equations because they are used in a CASE of the WHEN statement. The equations `low_flow` and `high_flow` are part of the mathematical problem only when the value of the boolean variable `laminar` matches the value of the list of values of the CASE in which they are defined. If we decide that we need to use the equation `low_flow`, then we have to give the value of TRUE to the boolean variable `laminar`. if we decide to use the equation `high_flow`, then we have to give the value of FALSE to the boolean variable `laminar`. Note that the value of the variable `laminar` can be modified as many times as the user wishes. In this way, the user may readily switch from one configuration to the other. In either of the CASEs, the resulting system of equations contains two equations (`invariant` and either `low_flow` or `high_flow`) in two variables (Re and f).

We could have used another kind of variable in the list of variables with exactly the same result. In the following example, an integer is used instead of a boolean. With an integer variable, we can have as many distinct CASEs as we wish inside a WHEN statement.

```
laminar  IS_A integer;
Re,f     IS_A factor;

invariant: sqrt(f) * Re = 0.00034576;
low_flow: Re = 64/f;
high_flow: Re = (0.206307/f)^4;

WHEN (laminar)
   CASE 1:
      USE low_flow;
   CASE 2:
      USE high_flow;
END WHEN;
```

13.1.2 A SECOND EXAMPLE

```
method   IS_A symbol;
simplified_flash  IS_A VLE_flash;
rigorous_flash    IS_A td_VLE_flash;

WHEN (method)
   CASE ‘rigorous’:
      USE rigorous_flash;
   CASE ‘simplified’:
      USE simplified_flash;
END WHEN;
```

For this example, we have exactly the same capability as in our previous simplest example; however, here the objects named inside the WHEN statement are models and not relations. Also, the decision is based in the value of a symbol variable: `method`. As mentioned before, practically speaking, to “USE” an object (model) means that the variables and equations contained in that object will become an active part of the system of nonlinear equations representing the current configuration of the problem.

13.2 THE SELECT STATEMENT: CONDITIONAL

COMPILATION

Aside from the flexibility that conditional statements (such as the WHEN statement) gives to the configuration of a model structure, another application of conditional tools is the economy of programming. An example commonly occurring in engineering is the selection of the thermodynamic model to be used for equilibrium calculations. In general, it is convenient to code all of the alternative methods so that, depending on the species appearing in the equilibrium system, we can select the most appropriate method.

In this kind of problem, the decision as to which configuration we are going to use can be made before we compile the model. We would like to compile only the configuration appropriate for the problem rather than compiling all available configurations.

The SELECT statement incorporates conditional compilation into the ASCEND system. While this conditional tool is flexible enough to represent all of the alternatives, its presence will indicate that only those alternatives consistent with the model data will be available after compilation.

Even though the syntax for the SELECT statement is similar to that described for the WHEN statement, we nned to highlight some important differences:

- In the WHEN statement the declaration of the object is external to the conditional statement since of all the alternatives are going to be created anyway. In the SELECT statement, the actual declaration of an object (or any other declarative statement affecting objects) is done within each CASE of the conditional statement, explicitly discriminating among the alternative statements. Thus parts of a particular kind can exist in only one case within a select statement.
- The selection among alternatives in the SELECT statement depends on constant boolean variables, constant integer variables or constant symbols. Since these values imply a one time structural decision, they must not be modified during the solution of the problem. That is why they have to be constants.

The following is the syntax used for the conditional compilation tool:

```
defintion_of_constants;
assignment_of_constant_values;
```

```
SELECT (list_of_constants)
   CASE list_of_values_1:
            list1_of_declarative_statements;
   CASE list_of_values_2:
            list2_of_declarative_statements;
   ⋮
   OTHERWISE:
            listn_of_declarative_statements;
END SELECT;
```

Summarizing, the SELECT statement provides the capability of conditional compilation. It allows the representation of structural alternatives pursuing economy in programming, but, since only the desired data structure is created, it does not affect the computational requirements of the model.

13.2.1 A SIMPLE EXAMPLE

The following example shows an ASCEND model which is similar to that shown in the previous section. The difference is that we use the SELECT statement rather than the WHEN statement. This time, the symbol `method` is a constant, and, once it is defined, its value will not change. That value will always be a user decision. Also, note that the definition of the objects is done inside the SELECT statement. For this example, since the value of the symbol `method` is ‘rigorous’, the system will compile only the list of statements in the first CASE.

```
method   IS_A symbol_constant;
method :== ‘rigorous’;

SELECT (method)
   CASE ‘rigorous’:
      rigorous_flash IS_A td_VLE_flash;
   CASE ‘simplified’:
      simplified_flashIS_A VLE_flash;
END SELECT;
```

13.3 THE SWITCH STATEMENT: CONDITIONAL EXECUTION OF PROCEDURAL CODE

Because of the use of conditional statements in the declarative description of a model, a similar feature must also exist to give the user the ability to program the conditional execution of methods. For instance, each alternative configuration of a model may require different initialization and a different selection of the independent variables for the solution process. Hence, ASCEND has the following conditional procedural SWITCH statement:

```
SWITCH (list_of_variables)
   CASE list_of_values_1:
            list1_of_procedural_statements;
   CASE list_of_values_2:
            list2_of_procedural_statements;
    ⋮
   OTHERWISE:
            listn_of_procedural_statements;
END SWITCH;
```

This statement has the same meaning as conditional statements that exist in procedural modeling languages such as C and FORTRAN. The procedural statements in each of these cases do not involve new object definitions, they are only useful for the numerical processing of objects already created.

13.3.1 A SIMPLE EXAMPLE

The use of the SWITCH statement for the conditional execution of procedural code is illustrated below. In this example, the value of the variable `ave_alpha` is set to 1.5 only if the value of the symbol `method` is ‘simplified’. If the value of the symbol `method` is ‘rigorous’, then a procedure called `adiabatic` is executed instead.

```
METHODS
   METHOD values;
      RUN reset;
      SWITCH (method)
         CASE ‘rigorous’:
            RUN adiabatic;
         CASE ‘simplified’:
            ave_alpha := 1.5;
      END SWITCH;
   END values;
```