CHAPTER 16 A CONDITIONAL MODELING EXAMPLE: REPRESENTING A SUPERSTRUCTURE

To give an example of the application of the conditional modeling tool in ASCEND -the WHEN statement-, we developed a simplified model for the superstructure given in Figure 16-1. The code listed below exists in a file in the ASCEND models subdirectory entitled *when_demo.a4c*. You could run this example by loading this file and using it and its corresponding script *when_demo.a4s*.

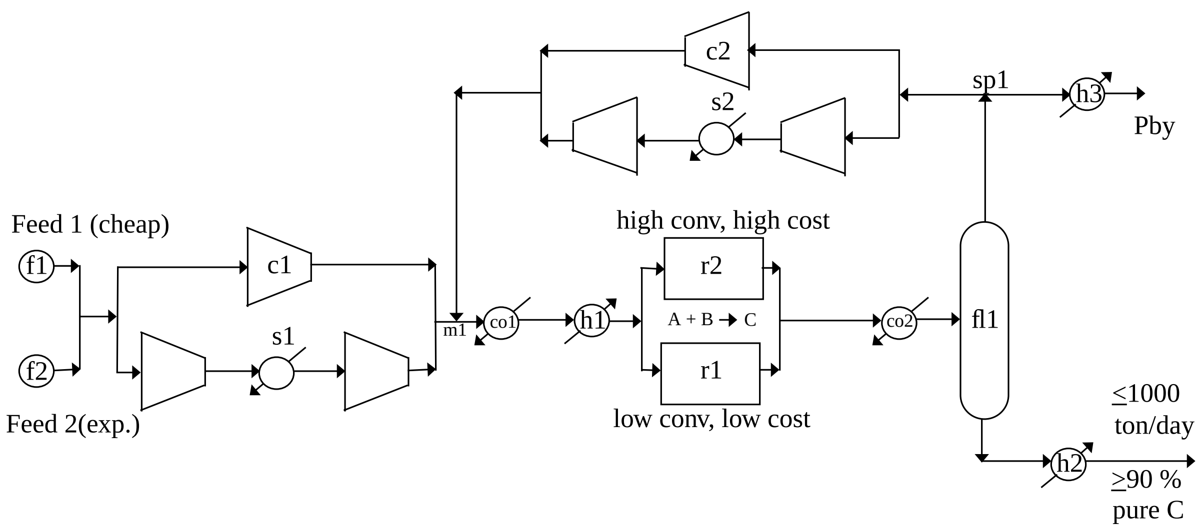

Figure 16-1 Superstructure used in the example of the application of the when statement

16.1 THE WHEN STATEMENT

Before showing the example, we want to start by giving a brief explanation about the semantics of the WHEN statement, a tool which allows ASCEND to represent conditional models efficiently.

In the WHEN statement, we take advantage of the fact that ASCEND is based on object oriented concepts where model definitions can contain parts that contain parts to any level. Furthermore, in ASCEND, a simple relation is treated as an object by itself and can have a name. Based on these ideas, the syntax for the WHEN statement is:

```
WHEN (list_of_variables)
   CASE list_of_values_1:
               USE name_of_equation_1;
               USE name_of_model_1;
   CASE list_of_values_2:
               USE name_of_equation_2;
               USE name_of_model_2;
   CASE list_of_values_nminus1:
               USE name_of_equation_nminu1;
               USE name_of_model_nminus1;
     OTHERWISE:
               USE name_of_equation_n;
               USE name_of_model_n;
END;
```

The following are important observations about the implementation:

1 The WHEN statement does not mean conditional compilation. We create and have available the data structures for all of the variables and equations in each of the models. This is actually a requirement for the solution algorithms of conditional models. All the models and equations whose name is given in each of the cases should be declared inside the model which contains the WHEN statement.

2 The variables in the list of variables can be of any type among boolean, integer or symbol or any combination of them. That is, we are not limited to the use of boolean variables. Obviously, The list of values in each case must be in agreement with the list of variables in the number of elements and type of each element. In other words, order matters in the list of variables of the WHEN statement, and parentheses are enclosing this list to make clear such a feature.

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

The WHEN statement represents an important contribution to modeling: it allows the user to define the domain of validity of both *models* and *equations* inside the cases of a WHEN statement. This feature enormously increases the scope of modeling in an equation based modeling environment.

Mainly, there are two different ways in which the WHEN statement can be used.:

- First, the WHEN statement can be used to select a configuration of a problem among several alternative configurations.
- 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. A typical example of this situation is the laminar-turbulent flow transition. The selection of the equation to calculate the friction factor depends on the value of the Reynolds number, which is an unknown in the problem.

16.2 THE PROBLEM DESCRIPTION

In the example, there are two alternative feedstocks, two possible choices of the reactor and two choices for each of the compression systems. The user has to make 4 decisions (for example, using either the cheap feed or the expensive feed), therefore, there are $2^4 = 16$ feasible configurations of the problem. All these 16 configurations are encapsulated in one ASCEND model containing 4 WHEN statements which depend on the value of 4 boolean variables.

The value of the four boolean variables will determine the structure of the problem to be solved. In this example, those values are defined by the modeler, but they also could be defined by some logic inference algorithm which would allow the automatic change of the structure of the problem.

The following section gives the code for this model. The first models correspond to the different types of unit operations existing in the superstructure. Those model are very simplified. You may want to skip them and analyze only the model *flowsheet*, in which the use and syntax of the WHEN statement as well as the configuration of the superstructure become evident.

16.3 THE CODE

As the code is in our ASCEND examples subdirectory, it has header information that we required of all such files included as one large comment extending over several lines. Comments are in the form (* comment *). The last item in this header information is a list of the files one must load before loading this one, i.e., *system.a4l* and *atoms.a4l*.

```
REQUIRE "atoms.a4l";
(* --> measures,system *)
PROVIDE "when_demo.a4c";
(*********************************************************************\
                        when_demo.a4c
                        by Vicente Rico-Ramirez
                        Part of the Ascend Library

This file is part of the Ascend modeling library.

The Ascend modeling library is free software; you can redistribute
it and/or modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of the
License, or (at your option) any later version.

The Ascend Language Interpreter is distributed in hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License along with
the program; if not, write to the Free Software Foundation, Inc., 675
Mass Ave, Cambridge, MA 02139 USA.  Check the file named COPYING.

Use of this module is demonstrated by the associated script file
when_demo.a4s.
\*********************************************************************)

(*********************************************************************\
  $Date: 1998/05/14 21:39:44 $
  $Revision: 1.5 $
  $Author: rv2a $
 $Source: /afs/cs.cmu.edu/project/ascend/Repository/models/when_demo.a4c,v $
\*********************************************************************)

(*
   This model is intended to demonstrate the degree of flexibility
that the use of conditional statements -when statement- provides
to the representation of superstructures. We hope that this
application will become clear by looking at the MODEL flowsheet,
in which the existence/nonexistence of some of the unit operations
is represented by when statements. A particular combination of
user defined boolean variables -see method values, configuration2,
configuration3- will a define a particular configuration of the
problem.
```

```
This model requires:
         "system.a4l"
          "atoms.a4l"
*)

(* ************************************************* *)

MODEL mixture;

   components                        IS_A set OF symbol_constant;
   Cpi[components]                   IS_A heat_capacity;
   y[components]                     IS_A fraction;
   P                                 IS_A pressure;
   T                                 IS_A temperature;
   Cp                                IS_A heat_capacity;


   SUM[y[i] | i IN components] = 1.0;
   Cp = SUM[Cpi[i] * y[i] | i IN components];

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      Cpi[components].fixed := TRUE;
      P.fixed := TRUE;
      T.fixed := TRUE;
      y[components].fixed := TRUE;
      y[CHOICE[components]].fixed := FALSE;
   END specify;

END mixture;


(* ************************************************* *)

MODEL molar_stream;
   state                IS_A mixture;
   Ftot,f[components]   IS_A molar_rate;
   components           IS_A set OF symbol_constant;
   P                    IS_A pressure;
   T                    IS_A temperature;
   Cp                   IS_A heat_capacity;

```

```
   components, state.components     ARE_THE_SAME;
   P, state.P                       ARE_THE_SAME;
   T, state.T                       ARE_THE_SAME;
   Cp, state.Cp                     ARE_THE_SAME;

   FOR i IN components CREATE
       f_def[i]: f[i] = Ftot*state.y[i];
   END FOR;

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN state.specify;
      state.y[components].fixed := FALSE;
      f[components].fixed := TRUE;
   END specify;

END molar_stream;

(* ************************************************* *)


MODEL cheap_feed;
   stream               IS_A molar_stream;
   cost_factor          IS_A cost_per_mole;
   cost                 IS_A cost_per_time;

   stream.f['A'] = 0.060 {kg_mole/s};
   stream.f['B'] = 0.025 {kg_mole/s};
   stream.f['D'] = 0.015 {kg_mole/s};
   stream.f['C'] = 0.00 {kg_mole/s};
   stream.T = 300 {K};
   stream.P = 5 {bar};

   cost = cost_factor * stream.Ftot;
METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN stream.specify;
      stream.f[stream.components].fixed := FALSE;
```

```
      cost_factor.fixed := TRUE;
      stream.T.fixed := FALSE;
      stream.P.fixed := FALSE;
   END specify;

END cheap_feed;


(* ************************************************* *)

MODEL expensive_feed;
   stream            IS_A molar_stream;
   cost_factor       IS_A cost_per_mole;
   cost              IS_A cost_per_time;

   stream.f['A'] = 0.065 {kg_mole/s};
   stream.f['B'] = 0.030 {kg_mole/s};
   stream.f['D'] = 0.05  {kg_mole/s};
   stream.f['C'] = 0.00  {kg_mole/s};
   stream.T = 320 {K};
   stream.P = 6 {bar};

   cost = 3 * cost_factor * stream.Ftot;

METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN stream.specify;
      stream.f[stream.components].fixed := FALSE;
      cost_factor.fixed := TRUE;
      stream.T.fixed := FALSE;
      stream.P.fixed := FALSE;
    END specify;

END expensive_feed;

(* ************************************************* *)


MODEL heater;
   input,output      IS_A molar_stream;
   heat_supplied     IS_A energy_rate;
   components        IS_A set OF symbol_constant;
```

```
   cost              IS_A cost_per_time;
   cost_factor       IS_A cost_per_energy;

   components,input.components,output.components   ARE_THE_SAME;
   FOR i IN components CREATE
     input.state.Cpi[i], output.state.Cpi[i]       ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
      input.f[i] = output.f[i];
   END FOR;

   input.P = output.P;

   heat_supplied = input.Cp *(output.T - input.T) * input.Ftot;

   cost = cost_factor * heat_supplied;

METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN input.specify;
      cost_factor.fixed := TRUE;
      heat_supplied.fixed := TRUE;
   END specify;

   METHOD seqmod;
      cost_factor.fixed := TRUE;
      heat_supplied.fixed := TRUE;
   END seqmod;

END heater;


(* ************************************************* *)

MODEL cooler;

   input,output         IS_A molar_stream;
   heat_removed         IS_A energy_rate;
   components           IS_A set OF symbol_constant;
   cost                 IS_A cost_per_time;
   cost_factor          IS_A cost_per_energy;
```

```
   components,input.components,output.components      ARE_THE_SAME;
   FOR i IN components CREATE
     input.state.Cpi[i],output.state.Cpi[i]           ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
      input.f[i] = output.f[i];
   END FOR;

   input.P = output.P;
   heat_removed = input.Cp *(input.T - output.T) * input.Ftot;
   cost = cost_factor * heat_removed;

METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN input.specify;
      cost_factor.fixed := TRUE;
      heat_removed.fixed := TRUE;
   END specify;

   METHOD seqmod;
      cost_factor.fixed := TRUE;
      heat_removed.fixed := TRUE;
   END seqmod;

END cooler;

(* ************************************************* *)

MODEL single_compressor; (* Adiabatic Compression *)

   input,output         IS_A molar_stream;
   components           IS_A set OF symbol_constant;
   work_supplied        IS_A energy_rate;
   pressure_rate        IS_A factor;
   R                    IS_A gas_constant;
   cost                 IS_A cost_per_time;
   cost_factor          IS_A cost_per_energy;

   components,input.components,output.components      ARE_THE_SAME;
   FOR i IN components CREATE
```

```
     input.state.Cpi[i],output.state.Cpi[i]           ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
      input.f[i] = output.f[i];
   END FOR;

   pressure_rate = output.P / input.P;

   output.T = input.T * (pressure_rate ^(R/input.Cp) );

   work_supplied = input.Ftot * input.Cp * (output.T - input.T);

   cost = cost_factor * work_supplied;

METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN input.specify;
      cost_factor.fixed := TRUE;
      pressure_rate.fixed := TRUE;
   END specify;

   METHOD seqmod;
      cost_factor.fixed := TRUE;
      pressure_rate.fixed := TRUE;
   END seqmod;

END single_compressor;

(* ************************************************ *)

MODEL staged_compressor;

   input,output            IS_A molar_stream;
   components              IS_A set OF symbol_constant;
   work_supplied           IS_A energy_rate;
   heat_removed            IS_A energy_rate;
   T_middle                IS_A temperature;
   n_stages                IS_A factor;
   pressure_rate           IS_A factor;
   stage_pressure_rate     IS_A factor;
   R                       IS_A gas_constant;
```

```
   cost                    IS_A cost_per_time;
   cost_factor_work        IS_A cost_per_energy;
   cost_factor_heat        IS_A cost_per_energy;

   components,input.components,output.components      ARE_THE_SAME;
   FOR i IN components CREATE
     input.state.Cpi[i],output.state.Cpi[i]           ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
      input.f[i] = output.f[i];
   END FOR;

   output.T = input.T;

   pressure_rate = output.P / input.P;

   stage_pressure_rate =(pressure_rate)^(1.0/n_stages);

   T_middle = input.T * (stage_pressure_rate ^(R/input.Cp));

   work_supplied = input.Ftot * n_stages * input.Cp *
         (T_middle - input.T);

   heat_removed =  input.Ftot * (n_stages - 1.0) *
         input.Cp * (T_middle - input.T);

   cost = cost_factor_work * work_supplied +
               cost_factor_heat * heat_removed;

METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN input.specify;
      n_stages.fixed := TRUE;
      cost_factor_heat.fixed := TRUE;
      cost_factor_work.fixed := TRUE;
      pressure_rate.fixed := TRUE;
   END specify;

   METHOD seqmod;
      n_stages.fixed := TRUE;
      cost_factor_heat.fixed := TRUE;
```

```
      cost_factor_work.fixed := TRUE;
      pressure_rate.fixed := TRUE;
   END seqmod;

END staged_compressor;

(* ************************************************* *)

MODEL mixer;

   components                 IS_A set OF symbol_constant;
   n_inputs                   IS_A integer_constant;
   feed[1..n_inputs], out     IS_A molar_stream;
   To                         IS_A temperature;

   components,feed[1..n_inputs].components,
   out.components             ARE_THE_SAME;
   FOR i IN components CREATE
     feed[1..n_inputs].state.Cpi[i],out.state.Cpi[i]     ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
     cmb[i]: out.f[i] = SUM[feed[1..n_inputs].f[i]];
   END FOR;

   SUM[(feed[i].Cp *feed[i].Ftot * (feed[i].T - To))|i IN [1..n_inputs]]=
                                 out.Cp *out.Ftot * (out.T - To);

   SUM[( feed[i].Ftot * feed[i].T / feed[i].P  )|i IN [1..n_inputs]] =
                                 out.Ftot * out.T / out.P;

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      To.fixed := TRUE;
      RUN feed[1..n_inputs].specify;
   END specify;

   METHOD seqmod;
      To.fixed := TRUE;
   END seqmod;

END mixer;
```

```
(* ************************************************ *)

MODEL splitter;

   components                    IS_A set OF symbol_constant;
   n_outputs                     IS_A integer_constant;
   feed, out[1..n_outputs]       IS_A molar_stream;
   split[1..n_outputs]           IS_A fraction;

   components, feed.components,
        out[1..n_outputs].components         ARE_THE_SAME;
   feed.state,
   out[1..n_outputs].state                   ARE_THE_SAME;

   FOR j IN [1..n_outputs] CREATE
      out[j].Ftot = split[j]*feed.Ftot;
   END FOR;

   SUM[split[1..n_outputs]] = 1.0;

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN feed.specify;
      split[1..n_outputs-1].fixed:=TRUE;
   END specify;

   METHOD seqmod;
      split[1..n_outputs-1].fixed:=TRUE;
   END seqmod;

END splitter;


(* ************************************************ *)

MODEL cheap_reactor;
   components                    IS_A set OF symbol_constant;
   input, output                 IS_A molar_stream;
   low_turnover                  IS_A molar_rate;
   stoich_coef[input.components] IS_A factor;
   cost_factor                   IS_A cost_per_mole;
```

```
   cost                          IS_A cost_per_time;

   components,input.components, output.components     ARE_THE_SAME;
   FOR i IN components CREATE
     input.state.Cpi[i], output.state.Cpi[i]          ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
       output.f[i] = input.f[i] + stoich_coef[i]*low_turnover;
   END FOR;

   input.T = output.T;
   (* ideal gas constant volume *)
   input.Ftot * input.T / input.P = output.Ftot * output.T/output.P;

   cost = cost_factor * low_turnover;

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN input.specify;
      low_turnover.fixed:= TRUE;
      stoich_coef[input.components].fixed:= TRUE;
      cost_factor.fixed := TRUE;
   END specify;

   METHOD seqmod;
      low_turnover.fixed:= TRUE;
      stoich_coef[input.components].fixed:= TRUE;
      cost_factor.fixed := TRUE;
   END seqmod;

END cheap_reactor;


(* ************************************************* *)

MODEL expensive_reactor;

   components                       IS_A set OF symbol_constant;
   input, output                    IS_A molar_stream;
   high_turnover                    IS_A molar_rate;
   stoich_coef[input.components]    IS_A factor;
```

```
   cost_factor                      IS_A cost_per_mole;
   cost                             IS_A cost_per_time;

   components,input.components, output.components     ARE_THE_SAME;
   FOR i IN components CREATE
     input.state.Cpi[i], output.state.Cpi[i]          ARE_THE_SAME;
   END FOR;

   FOR i IN components CREATE
       output.f[i] = input.f[i] + stoich_coef[i]*high_turnover;
   END FOR;

   input.T = output.T;
   (* ideal gas constant volume *)
   input.Ftot * input.T / input.P = output.Ftot * output.T/output.P;

   cost = cost_factor * high_turnover;

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN input.specify;
      high_turnover.fixed:= TRUE;
      stoich_coef[input.components].fixed:= TRUE;
      cost_factor.fixed := TRUE;
   END specify;

   METHOD seqmod;
      high_turnover.fixed:= TRUE;
      stoich_coef[input.components].fixed:= TRUE;
      cost_factor.fixed := TRUE;
   END seqmod;

END expensive_reactor;

(* ************************************************* *)

MODEL flash;

   components                 IS_A set OF symbol_constant;
   feed,vap,liq               IS_A molar_stream;
   alpha[feed.components]     IS_A factor;
   ave_alpha                  IS_A factor;
```

```
   vap_to_feed_ratio          IS_A fraction;

   components,feed.components,
   vap.components,
   liq.components             ARE_THE_SAME;
   FOR i IN components CREATE
     feed.state.Cpi[i],
     vap.state.Cpi[i],
          liq.state.Cpi[i]    ARE_THE_SAME;
   END FOR;

   vap_to_feed_ratio*feed.Ftot = vap.Ftot;

   FOR i IN components CREATE
      cmb[i]: feed.f[i] = vap.f[i] + liq.f[i];
      eq[i]:  vap.state.y[i]*ave_alpha = alpha[i]*liq.state.y[i];
   END FOR;

   feed.T = vap.T;
   feed.T = liq.T;
   feed.P = vap.P;
   feed.P = liq.P;

  METHODS

   METHOD default_self;
   END default_self;

   METHOD specify;
      RUN feed.specify;
      alpha[feed.components].fixed:=TRUE;
      vap_to_feed_ratio.fixed:= TRUE;
   END specify;

   METHOD seqmod;
      alpha[feed.components].fixed:=TRUE;
      vap_to_feed_ratio.fixed:= TRUE;
   END seqmod;

END flash;

(* ************************************************* *)
```

Next, the model *flowsheet* is presented. This model represents one of the applications of the WHEN statement. Namely, selecting among alternative configurations of the problem. Note that in each of the WHEN statements we define the conditional existence of complete ASCEND models. A specific combination for each of the conditional variables -boolean_vars in the example- will define a specific configuration of the problem. Once a configuration has been selected, it will be kept until the user decides to change it. Note that the user does not have to recompile the model to switch among alternative configurations. The reconfiguration of the system can be done automatically by simply changing the values of the conditional variables. An obvious application of this would be the synthesis of process networks. While running the script *when_demo.a4s*, note the changes in the number of active equations, active variables and fixed variables for the different configurations. For example, the configuration defined by one of the feeds, two single-stage compressors and one of the reactors contains 169 active equations.

```
(* ************************************************* *)

MODEL flowsheet;

(* units *)

   f1        IS_A cheap_feed;
   f2        IS_A expensive_feed;

   c1        IS_A single_compressor;
   s1        IS_A staged_compressor;

   c2        IS_A single_compressor;
   s2        IS_A staged_compressor;

   r1        IS_A cheap_reactor;
   r2        IS_A expensive_reactor;

   co1,co2   IS_A cooler;
   h1,h2,h3  IS_A heater;
   fl1       IS_A flash;
   sp1       IS_A splitter;
   m1        IS_A mixer;

(* boolean variables  *)

   select_feed1        IS_A boolean_var;
   select_single1      IS_A boolean_var;
   select_cheapr1      IS_A boolean_var;
   select_single2      IS_A boolean_var;
```

```
(* define sets  *)

   m1.n_inputs :==2;
   sp1.n_outputs :== 2;

(* wire up flowsheet *)

   f1.stream, f2.stream, c1.input, s1.input     ARE_THE_SAME;
   c1.output, s1.output, m1.feed[2]             ARE_THE_SAME;
   m1.out,co1.input                             ARE_THE_SAME;
   co1.output, h1.input                         ARE_THE_SAME;
   h1.output, r1.input, r2.input                ARE_THE_SAME;
   r1.output, r2.output,co2.input               ARE_THE_SAME;
   co2.output, fl1.feed                         ARE_THE_SAME;
   fl1.liq, h2.input                            ARE_THE_SAME;
   fl1.vap, sp1.feed                            ARE_THE_SAME;
   sp1.out[1], h3.input                         ARE_THE_SAME;
   sp1.out[2],c2.input, s2.input                ARE_THE_SAME;
   c2.output, s2.output,m1.feed[1]              ARE_THE_SAME;


(* Conditional statements *)

   WHEN (select_feed1)
     CASE TRUE:
      USE f1;
     CASE FALSE:
      USE f2;
   END WHEN;

   WHEN (select_single1)
     CASE TRUE:
      USE c1;
     CASE FALSE:
      USE s1;
   END WHEN;

   WHEN (select_cheapr1)
     CASE TRUE:
      USE r1;
     CASE FALSE:
      USE r2;
   END WHEN;

   WHEN (select_single2)
```

```
      CASE TRUE:
        USE c2;
      CASE FALSE:
        USE s2;
    END WHEN;


  METHODS

    METHOD default_self;
    END default_self;

    METHOD seqmod;
       RUN c1.seqmod;
       RUN c2.seqmod;
       RUN s1.seqmod;
       RUN s2.seqmod;
       RUN co1.seqmod;
       RUN co2.seqmod;
       RUN h1.seqmod;
       RUN h2.seqmod;
       RUN h3.seqmod;
       RUN r1.seqmod;
       RUN r2.seqmod;
       RUN fl1.seqmod;
       RUN sp1.seqmod;
       RUN m1.seqmod;
    END seqmod;

    METHOD specify;
       RUN seqmod;
       RUN f1.specify;
       RUN f2.specify;
    END specify;

END flowsheet;

(* ************************************************* *)

MODEL test_flowsheet REFINES flowsheet;

   f1.stream.components :== ['A','B','C','D'];

  METHODS

    METHOD default_self;
```

```
   END default_self;

   METHOD values;

        (* Initial Configuration *)
      select_feed1 := TRUE;
      select_single1 := TRUE;
      select_cheapr1 := TRUE;
      select_single2 := TRUE;

        (* Fixed Values *)

(* Physical Properties of Components *)

      f1.stream.state.Cpi['A'] := 0.04 {BTU/mole/K};
      f1.stream.state.Cpi['B'] := 0.05 {BTU/mole/K};
      f1.stream.state.Cpi['C'] := 0.06 {BTU/mole/K};
      f1.stream.state.Cpi['D'] := 0.055 {BTU/mole/K};

(* Feed 1 *)
      f1.cost_factor := 0.026 {dollar/kg_mole};

(* Feed 2 *)
      f2.cost_factor := 0.033 {dollar/kg_mole};

(* Cooler 1 *)
      co1.cost_factor  := 0.7e-06 {dollar/kJ};
      co1.heat_removed := 100 {BTU/s};

(* Cooler 2 *)
      co2.heat_removed := 150 {BTU/s};
      co2.cost_factor  := 0.7e-06 {dollar/kJ};

(* Heater 1 *)
      h1.heat_supplied := 200 {BTU/s};
      h1.cost_factor  := 8e-06 {dollar/kJ};

(* Heater 2 *)
      h2.heat_supplied := 180 {BTU/s};
      h2.cost_factor  := 8e-06 {dollar/kJ};

(* Heater 3 *)
      h3.heat_supplied := 190 {BTU/s};
      h3.cost_factor  := 8e-06 {dollar/kJ};

(* Flash *)
```

```
      fl1.alpha['A'] := 12.0;
      fl1.alpha['B'] := 10.0;
      fl1.alpha['C'] := 1.0;
      fl1.alpha['D'] := 6.0;
      fl1.vap_to_feed_ratio :=0.9;

(* Splitter *)
      sp1.split[1] :=0.05;

(* Mixer *)
      m1.To := 298 {K};

(* Single Compressor 1 *)
      c1.cost_factor := 8.33333e-06 {dollar/kJ};
      c1.pressure_rate := 2.5;

(* Single Compressor 2 *)
      c2.cost_factor := 8.33333e-06 {dollar/kJ};
      c2.pressure_rate := 1.5;

(* Staged Compressor 1 *)
      s1.cost_factor_work := 8.33333e-06 {dollar/kJ};
      s1.cost_factor_heat := 0.7e-06 {dollar/kJ};
      s1.pressure_rate := 2.5;
      s1.n_stages := 2.0;

(* Staged Compressor 2 *)
      s2.cost_factor_work := 8.33333e-06 {dollar/kJ};
      s2.cost_factor_heat := 0.7e-06 {dollar/kJ};
      s2.pressure_rate := 1.5;
      s2.n_stages := 2.0;

(* Reactor 1 *)
      r1.stoich_coef['A']:=-1;
      r1.stoich_coef['B']:=-1;
      r1.stoich_coef['C']:=1;
      r1.stoich_coef['D']:=0;
      r1.low_turnover := 0.0069 {kg_mole/s};

(* Reactor 2 *)
      r2.stoich_coef['A']:=-1;
      r2.stoich_coef['B']:=-1;
      r2.stoich_coef['C']:=1;
      r2.stoich_coef['D']:=0;
      r2.high_turnover := 0.00828 {kg_mole/s};
```

```
        (* Initial Guess *)

(* Flash *)
      fl1.ave_alpha:= 5.0;

   END values;

   METHOD configuration2;
   (* alternative configuration *)
      select_feed1 := FALSE;
      select_single1 := FALSE;
      select_cheapr1 := FALSE;
      select_single2 := FALSE;
   END configuration2;

   METHOD configuration3;
   (* alternative configuration *)
      select_feed1 := FALSE;
      select_single1 := TRUE;
      select_cheapr1 := TRUE;
      select_single2 := FALSE;
   END configuration3;

END test_flowsheet;

(* ************************************************* *)
```