VU Einführung in Wissensbasierte Systeme
ASP Project: Model-based Diagnosis

Sommersemester 2012

May 19, 2012

Revised Edition

This a the revised version of the original task description. Due to errors and typos in the first version, we decided to publish a corrected one, including already announced changes from the official LVA-forum. Note that the grading will be based on this version, all older versions of the task description (denoted by the creation date) are invalid.

General Problem Setting

This exercise deals with the area of energy business and power supply control. The goal of this project is to create an automated tool for error diagnosis in a small power supply system which combines different types of power plants, control elements, and a switching substation. The purpose of the system is to react on over- and under-capacity in the central mains supply by providing additional power or saving energy in a storage power station. A schematic view on the system is shown in Figure 1. It consists of one control station `c`, one pumped-storage hydropower plant `psh` (a storage plant), one windmill `w`, and one fossil-fuel power station `f` (e.g., working with coal). Further, a switching substation `s` is used to connect the different plants and to work as an interface to the regular mains supply. For our system, there are two types of connections between the different stations: control lines and transmission lines. The former are displayed as thin, black lines while the latter are thick lines in gray. The arrow indicates the flow of information or power while the name on the ends of a line denotes the assigned input or output interface of that station. Each control line can be set to one of multiple predefined finite values and on transmission lines in our model, the amount of power can be measured in Kilowatt.

Note that we assume that all power plants work with the same voltage (although this is in practice very unlikely), so no voltage transformation substation is needed.
The system has the following input signals:

- the current load in the mains supply `L_cur`,
- the current demand in the mains supply `L_demand`,
- the state `S_storage` of the pumped-storage hydropower plant `psh`, which can be `full`, `half_full`, or `empty`,
- the weather condition `S_wind` for the windmill `w`, which is measured in km/h and can be an integer between 0 and 60, and
- a transmission line from the mains supply to the switching station `P_in`.

The system has one output signal `P_out`.

The function of the complete system is as follows: The difference between `L_cur` and `L_demand` indicates whether the system should provide additional energy or not. In case that more power is needed, the system uses its different power plants to produce energy in a cost-saving way by preferring renewable energy sources. So, if possible, it uses the windmill and the storage plant to fulfill the additional demand. Unfortunately, these stations can only provide

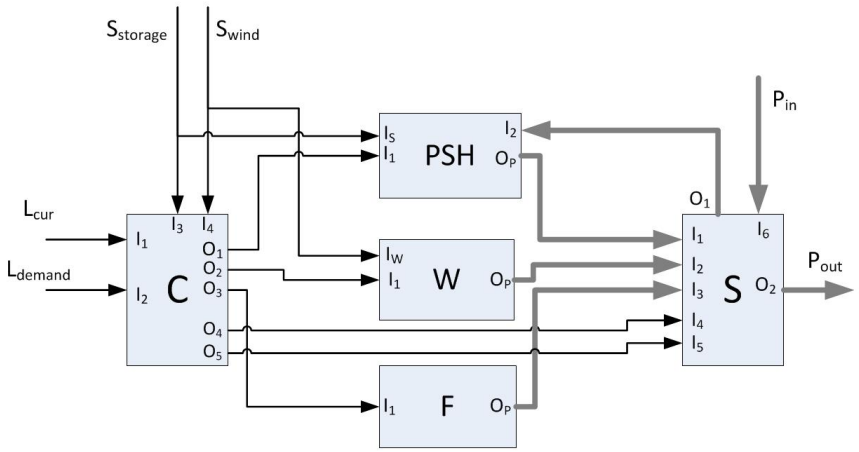

Figure 1: Model of a power plant system.

a certain amount of energy, so in some cases the fossil-fueled power station has to be used to generate the missing amount of energy. In case that the current capacity in the mains supply is too high, the system uses the excess energy to pump water for the pumped-storage hydropower plant so that it can be used later for generating energy.

For our exercise, the complete behavior is modeled as follows: In case that `L_cur` is less than `L_demand`, the control station `c` should calculate the missing amount and activate the different plants. Each power plant has a maximum capacity `C_x` with $x \in \{\texttt{psh}, \texttt{w}, \texttt{f}\}$ of how much power it can produce. The control station `c` then sends a control signal to the corresponding plants with a value equal to the kW is should produce. In addition, the switching substation receives a signal indicating that the power received should be accumulated and send to output transmission line `P_out`.

In case that `L_cur` is greater than `L_demand`, the control station checks if the storage plant `psh` is not full. If this is the case it uses power from the mains supply for charging the storage plant. Therefore, the control station orders the switching station `s` to redirect energy from the mains to the pumped-storage hydropower plant.

In addition, the pumped-storage hydropower plant `psh` has a predefined maximal power demand for charging `W_charge`, so it cannot consume more power at once while charging.

The electrical current flow of the system is modeled with predicate $P(S, I, V)$, where S is the substation, I is the input (or output) of a substation, and V is the power (in kW). For this exercise, the maximal power is 60 kW (so the values of V must be integers between 0 and 60). For the control lines, use the predicate $C(S, I, V)$, where S is again the substation, I is the input (or output) interface, and V is the corresponding value. The single elements of the system behave as follows (in case that they are not defective):

- The control station `c` receives the value of the current load `L_cur` on input `i1`, the current demand `L_demand` on input `i2`, the state of the storage plant `S_storage` on input `i3`, and the weather condition (wind speed) `S_wind` of the windmill on input `i4`. Depending on these values, the control station assigns the following values to its output interfaces:
  - The pumped-storage hydropower plant `psh` can only be used if its storage (indicated via the input `i3`) is not empty.
  - The windmill `w` can only be used under certain circumstances; namely, when the wind speed (which is received on input `i4`) is between 5 and 40.
  - In case that `L_cur` $<$ `L_demand`, the control station uses plants with the following priority with respect to their maximal capacity `C_x` with $x \in \{\texttt{psh}, \texttt{w}, \texttt{f}\}$:
    1. windmill `w`,
    2. pumped-storage hydropower plant `psh`, and

3. fossil-fuel power station `f`.

   This means that if possible, the system uses first the windmill as long as the additional power needed is below or equal the windmills capacity. If the demand exceeds this capacity (so the windmill alone cannot generate enough energy), it further activates the storage plant (if possible) and if the accumulated power of these two plants if still below the additional demand, it uses the fossil-fuel power station to generate the missing amount of energy.

   For using a power plant, the control station assigns the value of Kilowatt needed to the corresponding output variable (`o1` for `psh`, `o2` for `w`, `o3` for `f`). Further, it sends the signal `1` to the switching station `s` by assigning `1` to `o4` and `0` to `o5`.

- In case that $\texttt{L\_cur} > \texttt{L\_demand}$ and the storage plant is not full ($\texttt{i3} \neq \texttt{full}$), the control station orders the system to recharge the pumped-storage hydropower plant `psh` with power from the mains supply. Therefore, the control station informs the storage plant that it should pump water by setting `o1` to `charge` and orders the switching station to redirect $p = \min((\texttt{L\_cur} - \texttt{L\_demand}), \texttt{W\_charge})$ energy from `P_in` by assigning `2` to `o4` and p to `o5`.
- In all other cases, the system does not activate any plants nor the switching station. So, $\texttt{o1} = \texttt{o2} = \texttt{o3} = \texttt{o4} = \texttt{o5} = 0$.

- The pumped-storage hydropower plant `psh` receives control information on input `i1`, the state of the storage on input `is`, and, in case of charging, power for charging on input `i2`.

  Note that we do not model changes in the charging state `is`, so this value always stays the same!

  Depending on these values, the storage plant works as follows:

  - The storage plant can only generate power if it is not empty ($\texttt{is} \neq \texttt{empty}$).
  - In case it holds that $\texttt{i1} = \texttt{charge}$, its storages are not full ($\texttt{is} \neq \texttt{full}$), and it receives energy ($\texttt{W\_charge} \geq \texttt{i2} > 0$), it does not produce energy and therefore $\texttt{op} = 0$.
  - In case $\texttt{i1} = \texttt{charge}$, its storages are not full ($\texttt{is} \neq \texttt{full}$), and it receives energy ($\texttt{i2} > \texttt{W\_charge}$), it redirects the excessive energy back to the switching station, so $\texttt{op} = \texttt{i2} - \texttt{W\_charge}$.
  - In case $\texttt{i1} \neq \texttt{charge}$ but $\texttt{i2} > 0$, or $\texttt{i1} = \texttt{charge}$ and $\texttt{is} = \texttt{full}$, the station just redirects the power of `i2` to `op`. If it generates power, it simply adds this power to the redirected one from `i2`.
  - In case $\texttt{i1} = 0$ and $\texttt{i2} = 0$, it does not produce energy and therefore $\texttt{op} = 0$.
  - In case $\texttt{C\_psh} \geq \texttt{i1} > 0$ and $\texttt{is} \neq \texttt{empty}$, it produces energy, so $\texttt{op} = \texttt{i1}$.
  - In case $\texttt{i1} > \texttt{C\_psh}$ and $\texttt{is} \neq \texttt{empty}$, it generates an amount of energy equal to its maximal capacity `C_psh`, so $\texttt{op} = \texttt{C\_psh}$.

- The windmill plant `w` receives control information on input `i1` and the current wind speed on input `iw`.

  Note that we do not model changes in the wind speed `iw`, so this value always stays the same!

  Depending on these values, the windmill works as follows:

  - The windmill can only generate power if the wind speed `iw` is between 5 and 40 (including these values).
  - In case $\texttt{i1} = 0$, it does not produce energy and therefore $\texttt{op} = 0$.
  - In case $\texttt{C\_w} \geq \texttt{i1} > 0$, it produces energy, so $\texttt{op} = \texttt{i1}$.
  - In case $\texttt{i1} > \texttt{C\_w}$, it generates an amount of energy equal to its maximal capacity `C_w`, so $\texttt{op} = \texttt{C\_w}$.

- The fossil-fuel power station `f` receives control information on input `i1`. Depending on these values, the plant works as follows:

  - In case $\texttt{i1} = 0$, it does not produce energy and therefore $\texttt{op} = 0$.
  - In case $\texttt{C\_f} \geq \texttt{i1} > 0$, it produces energy, so $\texttt{op} = \texttt{i1}$.

  - In case $\texttt{i1} > \texttt{C\_f}$, it generates an amount of energy equal to its maximal capacity $\texttt{C\_f}$, so $\texttt{op} = \texttt{C\_f}$.

- The switching station `s` receives control information on inputs `i4` and `i5` and information about the incoming energy on input `i1` from `psh`, on input `i2` from `w`, on input `i3` from `f`, and on input `i6` from the mains supply `P_in`. Depending on these values, the switching station works as follows:

  - In case $\texttt{i4} = 0$, the system is inactive and the incoming energy `i6` is redirected to `o2`. Furthermore, $\texttt{o1} = 0$ as the pumped-storage hydropower plant `psh` should not receive energy.
  - In case $\texttt{i4} = 1$, the system accumulates all incoming energy from the plants (`i1`, `i2`, and `i3`) and from `i6`, and assigns the sum to `o2`. Again, $\texttt{o1} = 0$ as the pumped-storage hydropower plant `psh` should not receive energy.
  - In case $\texttt{i4} = 2$, the system should store power in the hydropower plant `psh` with the energy mains supply. Therefore, $\texttt{o1} = \texttt{i5}$ and $\texttt{o2} = \texttt{i6} - \texttt{i5}$. This is of course only possible if $\texttt{i6} \geq \texttt{i5}$; if this is not the case, it redirects as much energy as available to the storage plant, so $\texttt{o1} = \texttt{i6}$ and $\texttt{o2} = 0$.

Task 1.1: Define the Components

In this part of the exercise, you should model the behavior of the single components by using answer-set programming. In detail, use DLV to encode the components, using the following predicates:

- Predicate $\texttt{c}(\texttt{C}, \texttt{I}, \texttt{V})$ represents control information, where

  - `C` is the name of the component,
  - `I` is an interface (input or output), and
  - `V` is a value.

  For example, $\texttt{c}(\texttt{w}, \texttt{i1}, 5)$ represents the fact that on interface `i1` of the component `w`, the value 5 is present.

- Predicate $\texttt{p}(\texttt{C}, \texttt{I}, \texttt{V})$ represents information of power transmission interfaces, where

  - `C` is the name of the component,
  - `I` is an interface (input or output), and
  - `V` is a value.

  For example, $\texttt{p}(\texttt{w}, \texttt{op}, 5)$ represents that the fact the component `w` has a power output of 5 on the interface `op`.

- Predicate $\texttt{c\_max}(\texttt{P}, \texttt{V})$, with $P \in \{\texttt{psh}, \texttt{w}, \texttt{f}\}$, defines the maximal capacity V a power plant P can generate.

- Predicate $\texttt{w\_charge}(\texttt{V})$ defines the maximal energy V the storage plant can receive to store at once.

- Predicates $\texttt{control}(\texttt{C})$, $\texttt{storage\_plant}(\texttt{C})$, $\texttt{windmill}(\texttt{C})$, $\texttt{caloric\_plant}(\texttt{C})$, and $\texttt{switching}(\texttt{C})$ define if component C is a control station, a storage plant, a windmill, a fossil-fuel plant, or a switching station, respectively.

Task 1.1.1: Test Cases for the Components

Before you start encoding, design for each component five test cases testing the specification of the components. For example

$\texttt{windmill}(\texttt{w}).\ \texttt{c\_max}(\texttt{w}, 20).$
$\texttt{c}(\texttt{w}, \texttt{i1}, 5).\ \texttt{c}(\texttt{w}, \texttt{iw}, 10).$
$\texttt{expect\_c}(\texttt{w}, \texttt{i1}, 5).\ \texttt{expect\_c}(\texttt{w}, \texttt{iw}, 10).$
$\texttt{expect\_p}(\texttt{w}, \texttt{op}, 5).$

This test case represents a windmill with wind speed of 10 and an incoming value of 5 on `i1`, resulting in a power output of 5 at `o1`.

Name these files X`.test`n`.dl`, where $X \in \{$`c`, `psh`, `w`, `f`, `s`$\}$ (for control, pumped-storage hydropower, windmill, fossil-fuel, and switching), and n is a number ($1 \leq n \leq 5$).

Task 1.1.2: Component Modeling

Write five DLV programs, `c.dl`, `psh.dl`, `w.dl`, `f.dl`, and `s.dl`, which describe the behavior of the components.

Note: DO NOT use constants from the figures for the components (like `w` or `psh`) but the corresponding predicates (`windmill(W)`, `storage_plant(P)`, etc.) Furthermore, use the predicates `ab(C)` as described in the lecture. These predicates should disable the rules if the component `C` does not work correctly.

Hint:

- Use the *built-in* predicates of DLV (e.g.: $X = Y + Z$). For a correct usage of these arithmetic built-in predicates, it is mandatory to define an upper bound for the used integers. For this exercise, it is sufficient to use a range of $[0, 60]$. Therefore, start DLV with the option $-$`N` $=$ `60`.
- Do not forget to use the unary predicate `ab` (for abnormal) for the consistency-based diagnosis in DLV to specify your hypotheses. Furthermore, `ab` must only occur in combination with a default negation (that is, like `not ab(C)`). Moreover, ensure that no other predicate than `ab` occurs with a (default) negation.

Task 1.1.3: Testing the Model

Use your test cases for evaluating the modeling. Therefore, use the following program, which should be denoted by `component.tester.dl`:

```
UNCOMPUTED_c(C, O, X) :- expect_c(C, O, X), not c(C, O, X).
UNCOMPUTED_p(C, O, X) :- expect_p(C, O, X), not p(C, O, X).
UNEXPECTED_c(C, O, X) :- c(C, O, X), not expect_c(C, O, X).
UNEXPECTED_p(C, O, X) :- p(C, O, X), not expect_p(C, O, X).
DUPLICATED_c(C, O, X, Y) :- c(C, O, X), c(C, O, Y), X < Y.
DUPLICATED_p(C, O, X, Y) :- p(C, O, X), p(C, O, Y), X < Y.
```

The program detects for a component `C` on output `O` (either control or transmission output) if a value `X`, which is expected but not calculated. In that case, `UNCOMPUTED_c(C, O, X)` and `UNCOMPUTED_p(C, O, X)` hold. On the other hand, if a different value for `X` is calculated (but not expected), then `UNEXPECTED_c(C, O, X)` and `UNEXPECTED_p(C, O, X)` hold. In case that on output `O` there are two different values `X`, `Y`, the predicate `DUPLICATED_t(C, O, X, Y)` will indicate this.

For testing your model, use the following DLV call:

```
dlv X.dl X.testn.dl component.tester.dl -N=60
```

where $X \in \{$`c`, `psh`, `w`, `f`, `s`$\}$ and n is the number of the test case.

Task 1.2: Defining the Complete System

In the second part of this exercise, we are now constructing the complete system from the single components. To this end, use the following conventions:

- `in_c(l_current, V)` defines the current capacity of the mains supply,
- `in_c(l_demand, V)` defines the current demand of the mains supply,

- in_c(s_storage, V) defines the current state of the storage,
- in_c(s_wind, V) defines the current wind speed,
- in_p(p_in, V) defines the incoming power of the system in Kilowatt, and
- out_p(p_out, V) defines the power leaving the system in Kilowatt,

Task 1.2.1: Test Cases for the Complete System

Similar to the components, define ten test cases for the complete system which should cover as much functionality as possible. For example:

```
in_c(s_storage, full). in_c(s_wind, 0).
in_c(l_demand, 45). in_c(l_current, 20). in_p(p_in, 20).
expect_out_p(p_out, 45).
```

Name your files `connect.testn.dl`, where n is a number ($1 \leq n \leq 10$).

Task 1.2.2: Modeling of the Complete System

Now you can model the complete system. Therefore, define the different components by their names (use the constants `c`, `psh`, `w`, `f`, and `s`), connect the global input and output variables and link the input and output interfaces of the components with each other (according to the system specification).

Furthermore, define the following constant values for the capacity of the plants and the maximal charging capacity:

- c_max(psh, 20) defines the maximal capacity of the storage plant,
- c_max(w, 15) defines the maximal capacity of the windmill,
- c_max(f, 60) defines the maximal capacity of the fossil-fuel plant, and
- w_charge(30) defines the maximal charging capacity of the storage plants.

Save these definitions in the file `connect.dl`.

Task 1.2.3: Testing the Model

Use your test cases for checking your model. For this purpose, consider the following modification of the above introduced DLV program (save this program under the name `connect.tester.dl`):

```
UNCOMPUTED_c(O, X):−expect_out_c(O, X), not out_c(O, X).
UNCOMPUTED_p(O, X):−expect_out_p(O, X), not out_p(O, X).
UNEXPECTED_c(O, X):−out_c(O, X), not expect_out_c(O, X).
UNEXPECTED_p(O, X):−out_p(O, X), not expect_out_p(O, X).
DUPLICATED_c(O, X, Y):−out_c(O, X), out_c(O, Y), X < Y.
DUPLICATED_p(O, X, Y):−out_p(O, X), out_p(O, Y), X < Y.
```

Task 1.3: Diagnosis

So far, we have only worked with the modeling of the system. Now let us focus on the diagnosis part of the exercise. But first we have to define some constraints: Take the rules for DUPLICATED_c and DUPLICATED_p from the programs of Subtasks 1.1.3 and 1.2.3, transform them into constraints, and store them in file `constraints.dl`. The reason for this is that if the system works correctly, DUPLICATED_c and DUPLICATED_p should never be derived, which is modeled via these constraints.

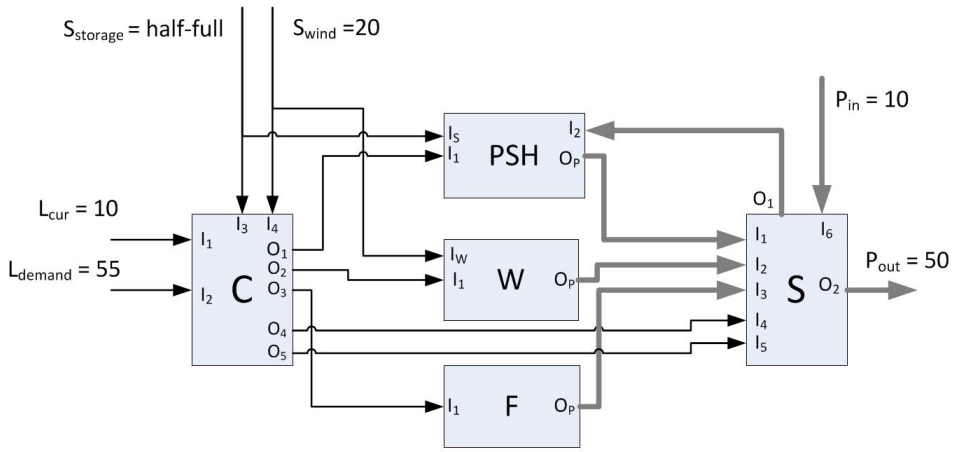

Figure 2: An observed fault.

For making a diagnosis, we check which components `C` seem to work incorrectly, represented by `ab(C)`, such that our model and the given observations are consistent.

Therefore, create appropriate hypotheses (each of the five components can be abnormal) and save them in file `abnormal.hyp`.

Exercise: make a diagnosis about your test cases

Use your test cases for the complete system as observations in a diagnosis problem. Copy the files, replace the suffix `.dl` by `.obs`, and remove the prefix `expect_` from your predicate names.

Then, perform consistency-based diagnosis computing all diagnoses (DLV Option `-FR`), single-fault diagnoses (DLV Option `-FRsingle`), and subset-minimal diagnoses (DLV Option `-FRmin`). Interpret the obtained results!

Note:

- This task does not influence the grading of this exercise. Nevertheless, we recommend that you do it for a better understanding of the modeled system and the diagnosis part.

Task 1.3.1: Make Diagnosis about Faults

Now we will consider concrete observations of faulty systems. Figure 2 represents observations of our system with an incorrect behavior. Represent these observations in the file `fault.obs`.

Again, perform consistency-based diagnosis computing all diagnoses (DLV Option `-FR`), single-fault diagnoses (DLV Option `-FRsingle`), and subset-minimal diagnoses (DLV Option `-FRmin`). Interpret the obtained results!

Task 1.3.2: Measuring Points

Reconsider the situation described in Figure 2 and its minimal diagnoses. Your have the reasonable suspicion that the switching station `s` is not defect. On which point would you do a measurement to confirm your suspicion? Give an example measurement which shows that (in combination with the observations of Figure 2), `s` alone is not defective and write it in the file `fault.s.ok.obs`.

Now, find a further measurement which shows that the (single) solution (in combination with the observations of Figure 2) is

$$\{\texttt{ab(f)}\},$$

and store this measurement in file

`fault.c.psh.w.ok.obs`.

Note: It is not necessary to copy the measurement values of `fault.obs` as these are already modeled in `fault.obs`.

Furthermore, provide a measurement such that the (single) minimal diagnosis (in combination with the observations of Figure 2) is

$$\{\texttt{ab(c)}, \texttt{ab(f)}\}$$

and save them in file `fault.c.f.ab.obs`.

Assuming that the switching station `s` is defective, give a measurement such that the (single) diagnosis (in combination with the observations of Figure 2) is

$$\{\texttt{ab(s)}\}$$

and store them in file `fault.s.ab.obs`.

Submission

Upload a ZIP-file with your solution to the TUWEL system. Name your file `XXXXXXX_project1.zip`, where `XXXXXXX` is replaced by your immatriculation number.

Make sure that the ZIP-file contains the following files:

- X`.test`n`.dl`, where $X \in \{$`c`, `psh`, `w`, `f`, `s`$\}$,
- `c.dl`, `psh.dl`, `w.dl`, `f.dl`, and `s.dl`,
- `connect.test`n`.dl`, where n is a number ($1 \leq n \leq 10$),
- `connect.dl`,
- `component.tester.dl`,
- `abnormal.hyp`,
- `fault.obs`,
- `fault.s.ok.obs`,
- `fault.c.psh.w.ok.obs`,
- `fault.c.f.ab.obs`, and
- `fault.s.ab.obs`

Make sure that your ZIP-file does not contain subdirectories!

Deadline

Deadline for the first turn-in is Tuesday, **June 5, 23:55**. You can submit multiple times, so please use it and do not wait until one minute before the deadline to submit. The deadline for the second turn-in is **June 12, 23:55**.