Project 22703: SYRF

Deliverable 3.5.2

Automatic Testing

---

1 Introduction

This task was quite early during the first year, since the LURETTE prototype was already available. During this second year, the results were published [HNRW98, RWNH98] (the second reference is given in Annex A.3.5.2), and the main progress has been the experiment of the tool on Schneider-Electric case study, SRIC. In this deliverable, we essentially present this experiment.

The SRIC consists of a reactive program, written in LUSTRE — or, more precisely, using the industrial tool SCADE. For automatic testing with LURETTE, we only need the C code produced from this program. To test such a program, we have to write observers describing

- the assumptions about the environment, and, more generally, the restrictions about the input scenarii that must be generated;
- the properties one wants to check during the testing process.

2 Observers Interface

All observers have the same inputs, which are all the input/output parameters of the SRIC program. Knowing the general structure of the C code generated by the SCADE compiler, LURETTE is able to analyze the code, and to produce a flat list of parameters of basic types (Boolean integer, real), in suitable order (inputs first), since the parameters in LUSTRE are passed by position. The only problem — solved by hand — was to retrieve the meaning of these parameters, since many of them were records fields, which have no name in SCADE. The only output of each observer is a Boolean, usually named `ok`.

3 Modelling the Environment

3.1 The "ictometer" inputs

The pair (`ccc`, `cctt`) follows a staircase-like curve (see Fig. 1), given in the specification. Its validity is defined by the folowing LUSTRE equation:

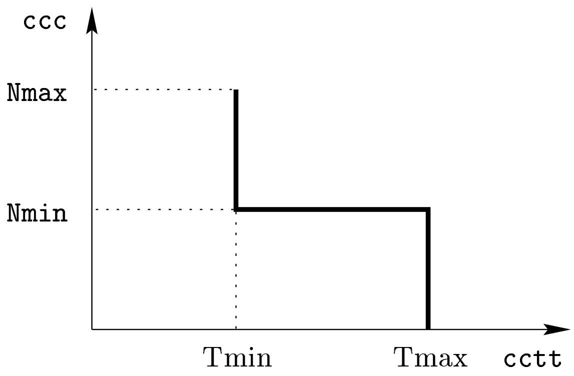

Figure 1: Domain of (ccc, cctt)

```
curve_ok = (    ((cctt = Tmin) and (ccc > Nmin) and (ccc < Nmax))
             or ((cctt > Tmin) and (cctt < Tmax) and (ccc = Nmin))
             or ((cctt = Tmax) and (ccc > 0) and (ccc < Nmin))
             );
```

Time is counted in 1/1600sec. the minimum time is 20ms (i.e., $32 \times 1/1600$) and the maximum time is 5.12sec ($8192 \times 1/1600$). The min and max numbers of hits are, respectively, 4096 and 65535 ($2^{16} - 1$):

```
const  Tmin = 32;
       Tmax = 8192;
       Nmin = 4096;
       Nmax = 65535;
```

When ccc = Nmax the "icto" board is always invalidated (i.e., the variable RAF_ICTO_val is true). In cases of "null counts",the board is also invalidated:

```
raf_icto_val_ok =
    (ccc = Nmax => RAF_ICTO_val) and
    (((ccc = 0) and (cctt = Tmax)) => RAF_ICTO_val)
```

3.2 Given Constants

Some parameters come from an *eprom*, and are constant for each execution. Default values are fixed by:

```
default_params_ok =
    (maxaz3tco = 1000) and
    (maxpifco = 1000000) and
    (minazipe = 20) and
    (minaz3pe = 10);
```

4 Restrictions on Scenarii

Several general situations have been defined and tested.

4.1 Quasi-random Testing: the Observer assert000

This first observer only takes into account the assumptions on the environment, without further restriction:

```
ok = default_params_ok and raf_icto_val_ok and curve_ok;
```

4.2 Testing Nominal Functionning

4.2.1 Nominal Functionning

In nominal mode, the following properties hold:

- The “audio” board is in linear mode:

  ```
  audio_lin and not audio_log
  ```

- The power supply is working normally:

  ```
  bf_ch_alim
  ```

- Concerning the high voltage supervision, no reset occurs, and the low and high thresholds are never reached:

  ```
  not rearm_ht and seuil_h_ht_nf and seuil_b_ht_nf
  ```

- The running mode is “normal”

  ```
  not ch_en_tst and not ch_en_regl
  ```

- Concerning the reading back of threshold passing, the corresponding inputs obey, with a unit delay, the values computed by the system:

  ```
  true -> (   rel_az3tco_nf = pre(az3tco_nf)
              and rel_piftco_nf = pre(piftco_nf)
              and rel_azipe_nf = pre(azipe_nf)
              and rel_az3pe_nf = pre(az3pe_nf)
          )
  ```

- Electronics, frame and fan work correctly:

  ```
  bf_elect and porte_fermee and bf_ventil
  ```

- Representative voltages are counted in point numbers ($[0, 4095]$); the high voltage value is not used and its initial value may be randomly chosen; the discrimination threshold must be less than the SHDI (high bound of discrimination threshold), which is 3071.

```
((val_ht >= 0 ) and (val_ht <= 4095)) -> (val_ht = pre(val_ht))
and ((seuil_disc >= 0) and ((seuil_disc <= 3071)) -> (seuil_disc = pre(seuil_disc))
```

- Constant parameters have their default values

```
default_params_ok
```

- All invalidity bits are false:

```
not VAL_HT_val and not SEUIL_DISC_val
    and not BF_ELECT_val and not RAF_ICTO_val
```

All these properties are summarized in a variable nominal:

```
nominal = audio_lin and not audio_log
          and bf_ch_alim
          and not rearm_ht and seuil_h_ht_nf and seuil_b_ht_nf
          and not ch_en_tst and not ch_en_regl
          and true -> (   rel_az3tco_nf = pre(az3tco_nf)
                          and rel_piftco_nf = pre(piftco_nf)
                          and rel_azipe_nf = pre(azipe_nf)
                          and rel_az3pe_nf = pre(az3pe_nf)
                      )
          and bf_elect and porte_fermee and bf_ventil
          and (((val_ht >= 0 ) and (val_ht <= 4095))
                      -> (val_ht = pre(val_ht)))
          and (((seuil_disc >= 0) and ((seuil_disc <= 3071))
                      -> (seuil_disc = pre(seuil_disc)))
          and default_params_ok
          and not VAL_HT_val and not SEUIL_DISC_val
          and not BF_ELECT_val and not RAF_ICTO_val;
```

4.2.2 The Observer assert001

It tests the case where the ratio ccc/cctt increases, with cctt = Tmax. From an initial value close to (but strictly greater then) 0, ccc increases with a slope between 10 and 15, but doesn't bypass Nmin. This behaviour is specified by the auxiliary variable increase_ccc. An integer variable delta_ccc is introduced to represent the slope.

```
delta_ccc = ccc - (0 -> pre(ccc));
```

```
increase_ccc =  (cctt = Tmax) and
                ((ccc >= 1 and ccc <= 10) ->
                        if raf_icto then
                            if pre(ccc) < Nmin - 50 then
                                (delta_ccc >= 10) and (delta_ccc <= 50)
                            else ccc = pre(ccc)
                        else ccc = pre(ccc)
                );
```

Now, the output of the observer is:

```
ok = nominal and increase_ccc;
```

4.2.3 The Observer assert002

We consider now the case where the ratio ccc/cctt increases, with ccc = Nmin. cctt decreases from Tmax, with a slope between -50 and -10, without passing Tmin. New values are available at each instant (i.e., raf_icto is true).

```
delta_cctt = cctt - (0 -> pre(cctt));
decrease_cctt =     (ccc = Nmin) and raf_icto and
                    ((cctt = Tmax ->
                            if pre(cctt) > Tmin + 50 then
                                (delta_cctt >= -50) and (delta_cctt <= -10)
                            else cctt = pre(cctt)
                    );
ok = nominal and decrease_cctt;
```

4.2.4 The Observer assert003

The last observer concerns the case where ccct = Tmin and ccc increases from Nmin according to a slope between 100 and 500, without passing Nmax. It is similar to the observer assert001.

5 Modelling the Properties

The "oracle" observer is used to check the expected properties during the test. Two properties, called P_1 and P_{11} are required:

```
ok = P1 and P11;
```

P_1 expresses relations between ccc_ccarte and ccc when new values are available:

```
P1 = raf_icto =>    if ccc >= 2 then (OUT_ccc = ccc-1)
                    else (OUT_ccc = ccc);
```

P_{11} states that, when the counting rate is high, the variables tcor and tcol must be (almost) equal:

```
high_rate = 1000.0;
epsilon = 1.0;
P11 = (tcor > high_rate) +> abs(tcor - tcol) <= epsilon;
```

6 Running the Tester

Once the observers have been defined, the tool LURETTE can be called. It displays a graphical interface window (see Fig. 2) which proposes the following steps:

- The program under test and its main node are selected (possibly through a browser). For the time being, the program can either come from a SCADE (formerly SAO+) description, or be written in the academic version of LUSTRE. The tool must know the compiler used, to be able to retrieve the interface in the generated C code. Once this selection has been made, the button "Build Test Program" compiles the program and links it with LURETTE libraries.

- The second step can be performed several times on the same program, without recompilation: the assertion and oracle observer are selected, together with the number and length of the test sequences. Then the button "Run Test Program" performs the test. Each test sequence is produced up to the specified length, or stopped as soon as the oracle observer complains (property violation). Complete test sequences (with program inputs and outputs) are stored for further examination (see below) or replay.

Once the sequences have been produced, they can be examined (see Fig. 3). Notice that, in general, each member of a sequence is a tuple of values, one for each input or output of the program. These tuple have generally many values, and it is interesting to see only some of them. A first displaying tool allows these interesting variables to be selected, and shows the reduced sequence in column. Moreover, a timing diagram view of such a sequence can be displayed.

7 Conclusion and Future Work

Several remarks can be made about this case study:

- Unfortunately, the model of the environment was rather poor, and did not permit to use the whole power of observers, in particular concerning the modelling of dynamic behavior. We should get more interesting examples from the hybrid model of the SAAB case study (deliverable 7.1).

- However, Section 4 showed that assertion observer can be very useful in driving the testcase generation towards "interesting" scenarii.

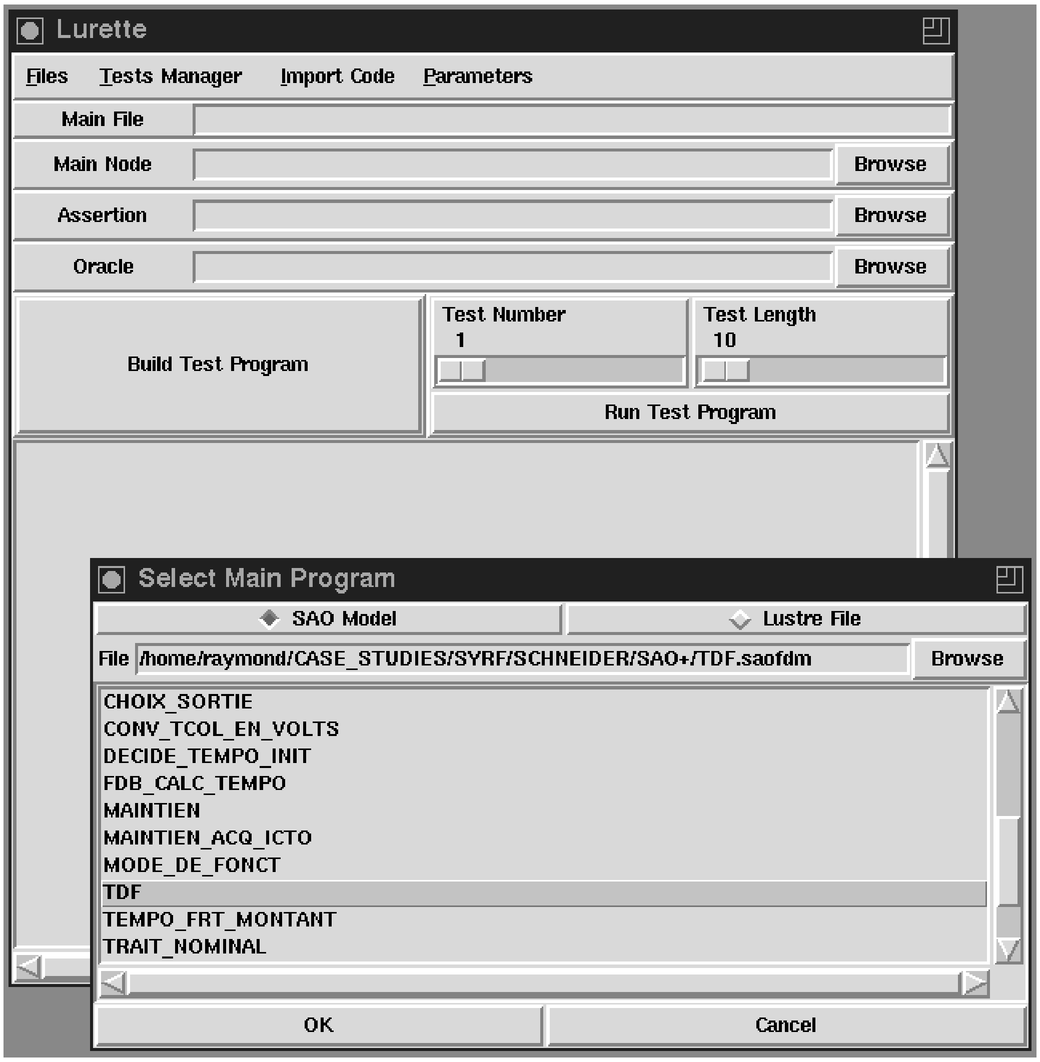

Figure 2: Graphical Interface

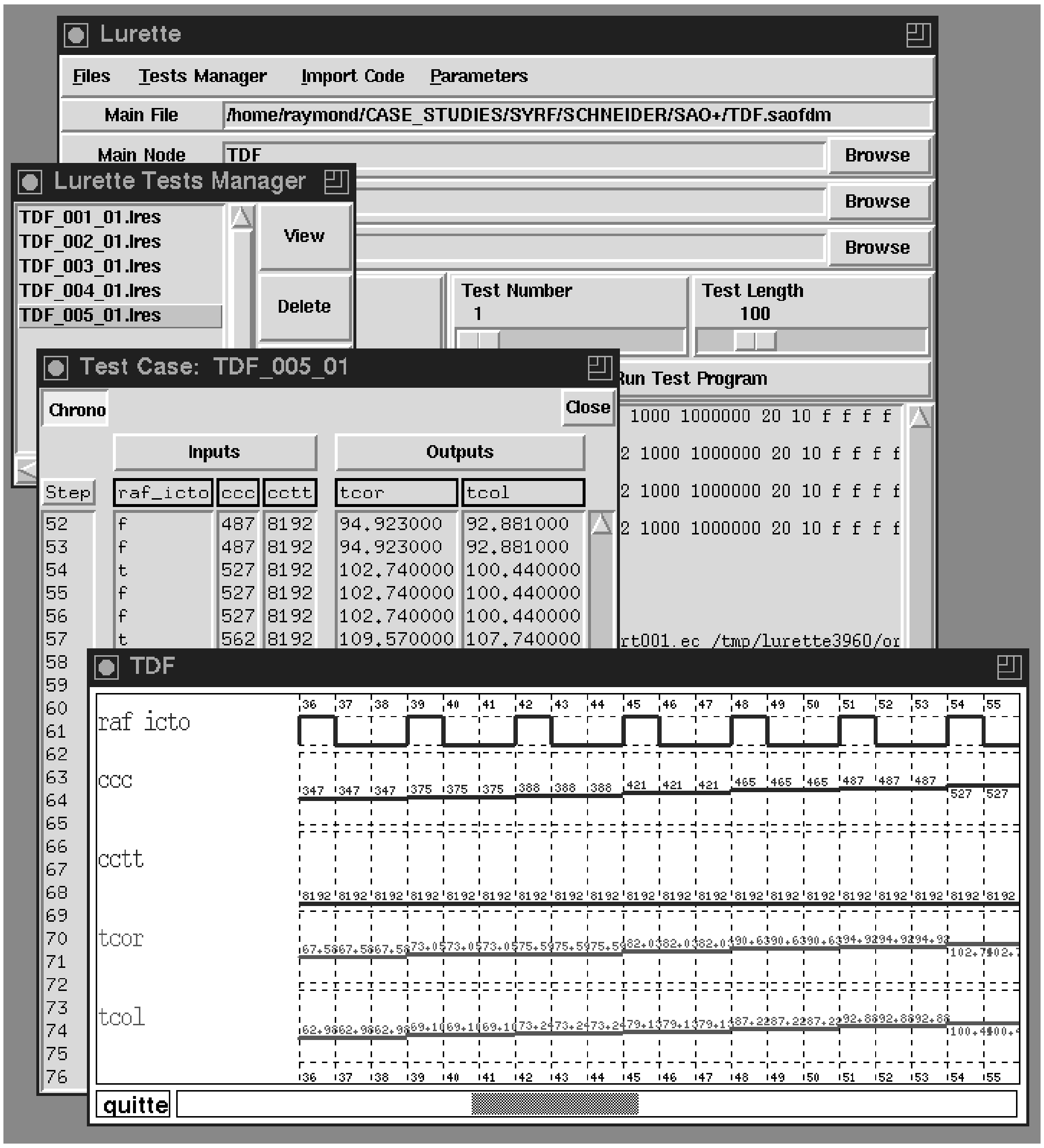

Figure 3: Displaying Testcases

- The tools for diplaying the results appear also very useful, even if, in the considered system, interesting behaviors can only be observed on very long sequences (typically, several hundreds of tuples; this is why we cannot show here a complete interesting sequence).
- Of course, the experiment did not find any errors in the program (the case study is extracted from a real software, actually running in a nuclear plant!). A "mutation analysis" [DeM80] remains to be done, to evaluate the quality of test case selection.

The next step will be to deal with the SAAB case study, using (a simplified version of) the hybrid model of its environment. Unfortunately, the present version of LURETTE does not allow such a model to be used, because it requires all the relations between present and past values of numerical variables to be *linear*. This restriction will be weakenned, since, at a given step of the generation, past values are known, and can be considered as coefficients. It should not be difficult to deal with assertions like

```
true -> (pre(x)*x <= 5)
```

which are likely to appear from the discretization of reasonably complex differential equations.

We would like also to improve the testcase selection procedure, which appears presently not to perform a "uniform" selection of points within a polyhedron. A point x is selected as a convex combination of polyhedron vertices $(v_i, i = 1 \ldots n)$

$$x = \sum_{i=1}^{n} \lambda_i.v_i, \quad 0 \leq \lambda_i \leq 1, \quad \Sigma_i \lambda_i = 1$$

so the selection consists in the selection of the λ_i. Thus, we only have to study the uniform selection of a point in a hypercube.

These tasks will be performed during the last year of the project.

References

[DeM80] R. A. DeMillo. Mutation analysis as a tool for software quality assurance. In *COMPSAC'80*, Chicago, October 1980.

[HNRW98] N. Halbwachs, X. Nicollin, P. Raymond, and D. Weber. Test automatique de systèmes réactifs. In *Ecole d'été "Modélisation et Vérification des Processus Paralleèles, MOVEP'98*, Nantes, July 1998.

[RWNH98] P. Raymond, D. Weber, X. Nicollin, and N. Halbwachs. Automatic testing of reactive systems. In *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.