Advanced model checking for verification and safety assessment
Lab 2

2016-05-XX

1 Introduction

This laboratory divided into:

1. Redundant Sensor (part 2)
2. Bonus Exercise: Mutual Exclusion

and assumes that you have the models from Lab 1.

2 Redundant Sensor

In Lab 1 we developed an architecture for the Redundant Sensor and nominal implementations for the leaf components.

2.1 Contracts

In contract-based design, we specify the top-level property and decompose it in the sub components. The following contract should be similar to the property that you verified at the end of Lab 1. Add it now to the top-level component `RedundantSensors`:

```
-- assuming: a bounded variance
-- guarantee: the output error bounded at max_sys_error
CONTRACT system_error
 assume: always
  (abs_diff(reading, next(reading)) <= max_variance);

 guarantee: always
        (abs_diff(reading, next(out)) <= max_sys_error);
```

In the refinement section of `RedundantSensors`, add the following:

```
CONTRACT system_error
 REFINEDBY sensor1.nominal,
           sensor2.nominal,
           voter.nominal;
```

This is stating that the contract is refined by the contracts named `nominal` in the three subcomponents.

Task

1. Create and define the contracts `nominal` for `Sensor` and `Voter`;
2. Run the script `ocra_nominal.cmd`. What is the output of `ocra_check_refinement` ? What is the output of `ocra_check_implementation`?

To write the contracts, you need to consider the properties that each component satisfies, as described in Lab 1. The file `hint_selector.oss` contains the contract for the selector component, that is the most complex contract. Look at it only if you get stuck.

2.2 Sensor Faults

The objective of the Redundant Sensor system is to be resilient to the occurrence of a single fault of a sensor. To check whether the system works, we extend the Sensor in order to add the bounded fault dynamic described in `bounded_fault.smv`.

1. Copy `Sensor.smv` into `Sensor_extended.smv`;
2. Copy the fault dynamic from `bounded_fault.smv` within `Sensor_extended.smv`;
3. Instantiate the fault module within `Sensor` as `VAR random_fault: BoundedFault(0);`
4. Modify sensor, so that whenever the fault is active, the output of the sensor is any possible value from the domain;
5. Copy `nominal.map` to `extended.map`, and change the mapping of the `Sensor` component accordingly;
6. Try to answer the following questions before running `ocra_extended.cmd`:
   - Is the contract refinement still valid?
   - Are the contracts satisfied by the implementation?
   - Is the top-level property/contract satisfied by the overall architecture?
7. Run `ocra_extended.cmd`. This will generate a file `System_extended.smv` that includes the extended Sensors.

Extended Analysis For the top-level property to be satisfied, we need to restrict the behavior of the system a bit further. In particular, we want to assume that initially there is no fault (neither for sensor1 nor sensor2), and that sensor1 can never fail.

Recall that given an LTL property φ, we can add a preconditions ψ by writing $\psi -> \varphi$.

- Modify `nuxmv.cmd` in order to include a stronger version of the property *RedundantSensor.system_error_norm_guarantee*, using the assumptions on the faults described above.

To show that the failure of both sensors is a cause of problems for the system, we compute the fault tree associated with the top-level contract:

`RedundantSensors_inst.system_error_norm_guarantee.`

To do so, run the script `fta.cmd`. You should now have two files (`events.txt` and `gates.txt`) that can be opened with the viewer:

```
$ ./ftv.sh events.txt gates.txt
```

In the viewer you should see that there is a Minimal Cutset of Cardinality 2 (Figure 1), meaning that the top-level property is violated if we allow both sensors to fail.

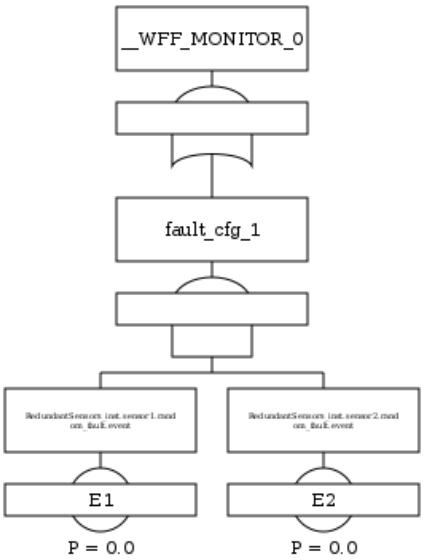

Figure 1: Fault Tree

3 Bonus Exercise: Mutual Exclusion

A typical example of formal methods is their application in multi-threaded programs, and in particular in mutual exclusion protocols. In the file `semaphore.smv` you will find an example of mutual exclusion between two systems.

- What features are usually expected of a mutual exclusion protocol?
- Write (at least) two LTL properties to verify those properties. Are they satisfied?
- If they are not satisfied, can you identify why and suggest a fix?