RESOLUTE tutorial

JULIEN DELANGE
Software Engineering Institute
jdelange@sei.cmu.edu

November 6, 2014

1 Introduction

This tutorial gives a tour of the functionality of RESOLUTE. It does not provide a complete description of the language, which is rather provided by the user-manual of RESOLUTE[1]. This tutorial assumes that you have a working installation of RESOLUTE and that you are able to use the validation tool.

This tutorial uses several examples hosted on OSATE github repository. You can get the examples on the OSATE example repository (see `https://github.com/osate/examples/`) under the directory `core-examples/resolute`.

2 Verifying property definition

This first example checks the definition of a property on a component. The related file is `property_verification.aadl`. The instance model is shown in figure 1.

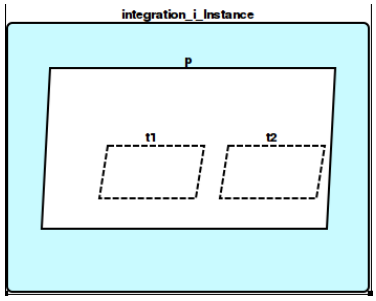

Figure 1: System instance in the `property_verification.aadl`

In this model, each `thread` has the property `resoluteps::foobar` defined:

- **t1** has a value of 10

[1]Documentation avaialble on `https://github.com/juli1/smaccm-improvements/tree/sei-improvements/documentation/resolute`

- **t1** has a value of 20

We define a theorem (`have_good_foobar`, see file `theorems.aadl` in the textual model) to check that all threads have a value for the `resoluteps::foobar` greater than 15.

The theorem works as follow:

1. It retrieves all the `thread` components associated to the process
2. It calls the theorem `check_thread` on each component.
3. The `check_thread` checks that the value of the `resoluteps::foobar` is greater than 15.

```
have_good_foobar(p : component) <=
   ** " Check threads in component " p **
    forall(t : thread). contained(t, p) => check_thread (t)

check_thread(t : thread) <=
   ** "The thread " t " as a foobar bigger than 15"  **
      (has_property (t, resoluteps::foobar)) and
      (property (t, resoluteps::foobar) > 15)
```

When running resolute on the initial model, the model is not validated because **t1** has a value of 10. To be able to validate the model, change the definition of t1 and associate a value greater than 15. For example, changing the definition of the task as follow will be sufficient to validate the model.

```
t1 : thread t.i {resoluteps::foobar => 20;};
```

After changing the model, run the analysis again, the model would be validated.
In this part, we introduce the following RESOLUTE concepts:

1. Call to another RESOLUTE theorem
2. Use the `forall` keyword
3. Use the `has_property` built-in function to check that a property is defined on a component.
4. Use the `property` built-in function to get the value of a property.

3 Analyzing Connections

Now that we have discussed the verification of property values in the components, we will present how you can analyze connections. In this example, we will check that each incoming port has only one incoming connection. This is a validation oen might want to validate in the model in order to ensure that there is only one sender for each communication port. This type of modeling restriction is required by some tools such as AGREE[2].

[2]AGREE actually does not support multiple fan-in, such a validation tool can then help designers to check compliance of their model against AGREE constraints

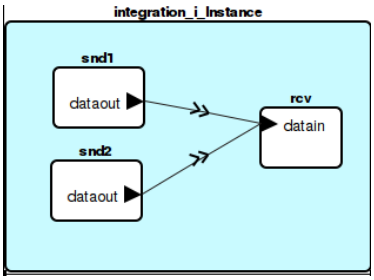

Figure 2: System Instance

The file related to this example is `check_fanin.aadl`. The related instance model is shown in figure 2. The theorem `no_double_fanin` retrieves all components within the model and validate them using the `has_single_fanin` theorem. This theorem gets all the incoming component features in a set and checks that they only have one connection.

```
no_double_fanin() <=
   ** " All incoming feature have only one connection" **
    forall (c : component) . true => has_single_fanin (c)


has_single_fanin (comp : component) <=
   ** " All incoming feature have only one connection on " comp **
   forall (f : features (comp)) . (direction(f) = "in") => (length (connections (f)) = 1)
```

The model is not validated (result shown in figure 3) because components `snd1` and `snd2` are connected to `recv` through the same incoming feature. In order to be able to validate the model, one solution is to add a new feature on `recv` and connect `snd1` and `snd2` to a single and distinct port.

Figure 3: Analysis result for the double fanin system

In this part, we introduce the following RESOLUTE concepts:

1. Call to another RESOLUTE theorem
2. Use the `forall` keyword
3. Use the `direction` built-in function (returns either `"in"`, `"out"` or `"inout"`).
4. Use the `length` built-in function that returns the size of a set

4 Analyzing Connections Consistency

5 Checking Compliance of an Architecture