AGREE tutorial

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

November 6, 2014

1 Introduction

This document is a tutorial to learn to use the AGREE language and its associated toolset. This is not a user-manual that covers all aspect of the language features, all these aspects are described in the AGREE user-manual[1]. This document is a way to learn how to use the language and its associated tools through several case studies.

1.1 Examples location

All the examples used in this tutorial are available online on the public OSATE github repository[2]. You can import the model into your workspace directly to reproduce the examples presented through this tutorial.

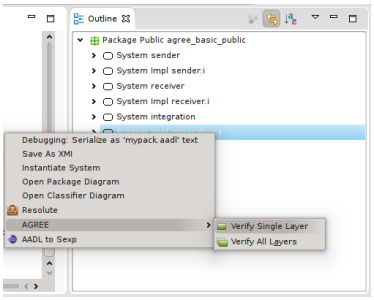

Figure 1: AGREE menu in OSATE outline

1.2 Use Analysis Tools

To use AGREE, model components must be annotated with AGREE annex subclauses. Then, invoking AGREE can be done by selecting the top-level system instance and

[1] https://github.com/smaccm/smaccm/tree/master/documentation/agree
[2] https://github.com/osate/examples/

make a right-click and select two options:

1. **Verify Single Layer**: analyze and verify only one depth of the component hierarchy.
2. **Verify All Layers**: analyze the complete components hierarchy.

1.3 Limitations

When using AGREE, your models must enforce some constraints. There is the list of the constraints your model has to enforce:

- **Execution Order**: the execution order of the model is done in the order of the declaration of the subcomponents.
- **Multiple fanin** are **not** supported. In other words, an incoming feature can have only one incoming connection.
- **Top-level component** must have an AGREE subclause, even if you do not want to verify anything and want to validate the subcomponent. Hopefully, you can insert a dummy subclause like the one shown below.

```
system mysystem
annex agree {**
   guarantee "dummy" : true;
**};
end mysystem;
```

1.4 Understanding Analysis Results

For each component, AGREE provides the following analysis:

1. **Contract Guarantees**
2. **Contract Assumptions**
3. **Contract Consistency**

2 First AGREE model

To understand AGREE basics, we will design a first model with some basics validation contracts. In this example, we will design a system with a sender and a receiver. The sender sends an integer through its outgoing interface.

The integration contract will then guarantee that the value sent through the interfaces is bound (with a range between 0 and 100) and the one received by the receiver has the same bound as well.

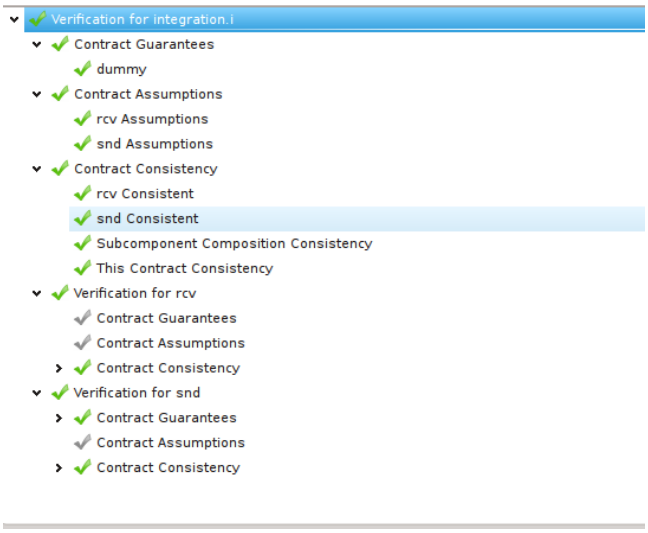

Figure 2: AGREE Results View in OSATE

2.1 Defining guarantees

In the following listing, we show the guarantee contract in the component type (the section defined with the `guarantee` keyword). The component implementation defines the behavior: the integer starts at 1, is incremented at each *tick* and eventually set to 10 when the value reached 90.

```
system sender
features
    dataout : out data port base_types::integer;
annex agree {**
    guarantee "data sent is between 0 and 100": dataout < 100 and dataout > 0;
**};
end sender;

system implementation sender.i
annex agree {**
    eq k : int = 1 -> if (pre(k) > 90) then 10 else pre(k) + 1;
    assert (dataout = k);
**};
end sender.i;
```

2.2 Defining assumptions

We also need to define components assumptions so that the analysis tool can check that they are consistent with the guarantees. In the following listing, we define the assumption for the receiver, specifying that the data received is bound within a range of 0 and 100.

```
system receiver
features
   datain : in data port base_types::integer;
annex agree {**
   assume "data produced between 0 and 100": (datain < 100) and (datain > 0);
```

```
**};
end receiver;
```

2.3 Running the tool

Once both `assume` and `guarantee` are defined, we integrate the components and can start the analysis tool. The results are shown in figure 2.

To show how AGREE can help you to investigate errors within your system, we will introduce an error in the contract. Let's change the `guarantee` of the sender component and specify that the data sent be within a range of 0 to 150. The component type specification should then looks like the following.

```
system sender
features
    dataout : out data port base_types::integer;
annex agree {**
    guarantee "data sent is between 0 and 150": dataout < 150 and dataout > 0;
**};
end sender;
```

When invoking the analysis tool again, it reports that the assumptions of the `rcv` component are not met, as shown in figure 3. When a contract is not validated, AGREE can then provide a counter example that details the different execution paths that lead to the validation error. To get the trace, right click on the assumption/guarantee/consistency contract not met and select the option to show a counter example. Counter examples can be shown in different format: text-based, spreadsheet (with Excel or OpenOffice) or in Eclipse, as shown in figure 4.

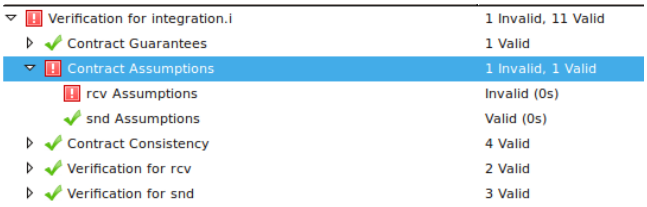

Figure 3: AGREE Results View in OSATE - Contracts not validated

Name	Step 1
integration_i_Instance	
CLOCK_rcv	true
CLOCK_snd	true
rcv Assumptions	false
rcv	
▽ rcv	
datain	100
snd	
▽ snd	
dataout	100

Figure 4: Counter Example in Eclipse