# Contract Scoping
# With AADL and AGREE

Danielle Stewart

April 2019

## 1 The Point of it All

While working on the contracts and overall behavior of the large scale WBS, a problem rose regarding port names at various levels of the system (see Tech Report for more information about the WBS: `https://www.cs.umn.edu/sites/cs.umn.edu/files/tech_reports/18-007_0.pdf`). For instance, one of the top level inputs to the system is *power*. This gets passed to lower level components and used in the contracts and is still called *power*. To refer to this port value, the dot construction must be used (in both AADL and AGREE). One of the contracts in the BSCU refers to a few of these inputs that come from the top level and flow down. A direct child component of the BSCU (*Channel*) has the same contract over its outputs/inputs as the BSCU, but the scope is different.
While testing these contracts, it was seen that the lower level contract proves, but going up to the BSCU, the same contract does not prove. It was assumed that it is due to the scope of those ports - just because they have the same name does not mean they are the same value. This small system was created to illustrate the behavior of scoping in AADL and AGREE. It is located in the following repository: `https://github.com/dkstewart/University-Research/tree/master/Models` and is titled *ContractScope*.

## 2 Contract Scope Model Description

The first version of the model has a top level component (*Oberst*) and one child (*Hauptmann*). The in and out ports are purposefully named the same thing in both levels.
Top level component, *Oberst*, has one boolean input (*in_1*) and one boolean output (*out_1*) .
Subcomponents:

- Hauptmann : One boolean input: *in_1* and one boolean output: *out_1*

The logic is very basic. In the child component, input equals output. There are no assumptions in the first version of the boolean model.

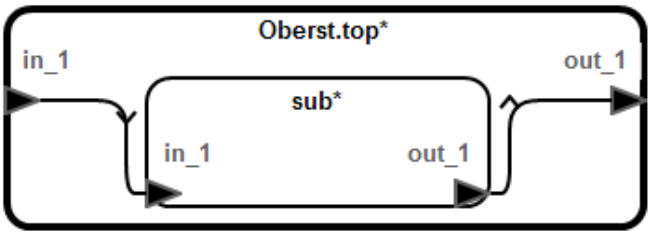



Figure 1: Structure of the Contract Scope system with connection

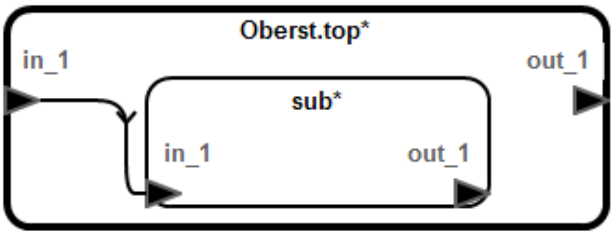



Figure 2: Structure of the Contract Scope system without connection

# 3 Top Level Contracts and Testing

There are a couple of main questions that we want to answer.

**1. If we lift up a property to a higher level and the ports are named the same, are we talking about different values/variables?**

To address this issue, I first just wrote a contract at the top level stating that output at the top equals output of the subcomponent.

*"Top level output equals sub level output."*
$out\_1 = sub.out\_1$

We expect this to be true AS LONG AS the connection port is defined between the child component output and the top level output (see Figure 1. When running the analysis with the connection in place, this proves. If the connection between these ports is commented out (Figure 2), this lemma fails to prove. If the input connection is severed, this lemma still proves. The output is all that matters.

Likewise, I wrote a contract stating that the top level input equals the top level output.
$in\_1 = out\_1$

The only contract that guarantees that this should be true is in the child component. Therefore if all connections are sound, this proves. Of course, if not, it does not prove. This is true for both the input and output connections.

```
annex agree {**
    lemma "Top level input equals sub level input." :
        in_1 = sub.in_1;

    lemma "Top level output equals sub level output." :
        out_1 = sub.out_1;

    lemma "Input equals output (top level)" :
        in_1 = out_1;

    lemma "Sub level input equals sub level output." :
        sub.out_1 = sub.in_1;
**};
```

Figure 3: Top level contracts for boolean model

All top level contracts are shown in Figure 3.

**2. What is the initial state of the system? Do we need to wait a step before the top level output port reflects the contract of the child component?**

To answer this question and really see initial behavior as well as test the contracts using the "→" operator, a second version of the system was made. The components and connections are the same, but the ports are integers and there are new assumptions. These integers are bounded between 0 and 10 and the initial value of the input is 1. See Figure 4.

The first round, I left in all the lemmas as shown in Figure 3 and added the "→' operator. All prove with connections declared. No "prev" operators are required anywhere in the model.

```
annex agree {**

    assume "Initial value of input is 1." :
        (in_1 = 1) -> true;

    assume "Input bounded between 0 and 10." :
        true -> ((in_1 >= 0) and (in_1 <= 10));

    guarantee "Output bounded between 0 and 10." :
        true -> ((out_1 >= 0) and (out_1 <= 10));

**};
```

Figure 4: Top level assumptions for integer model

When running the analysis with the connection in place, this proves. If the connection between these ports is commented out (Figure 2), this lemma fails to prove. All top level contracts are shown in Figure 3.
As a further test (mostly out of curiousity), I took out the "→" operator from the top level lemmas. Partly this is to check behavior when it is removed and partly to see whether we need consistency regarding this operator and if so, why.
The results of verification are shown in Figure 5.

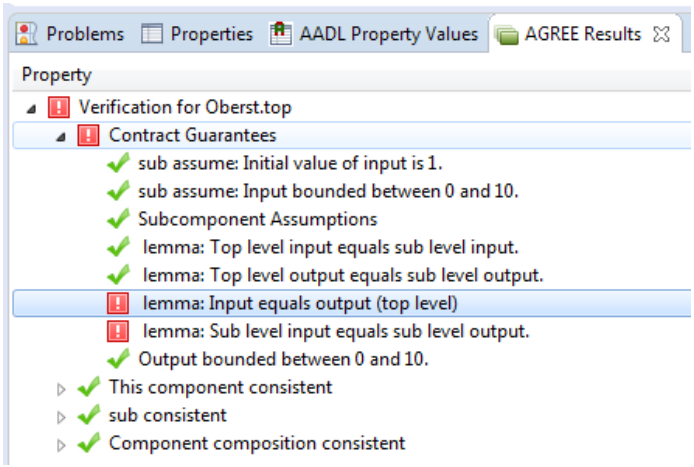
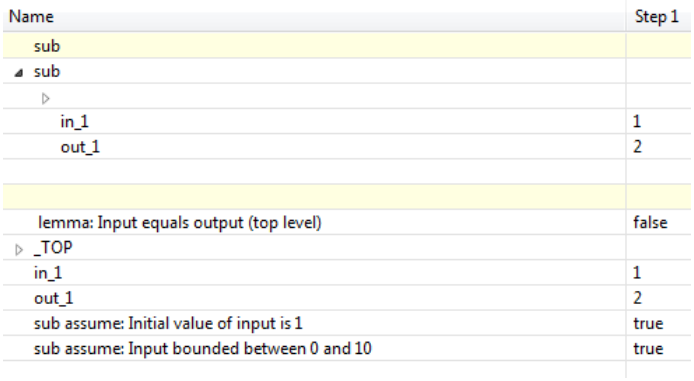


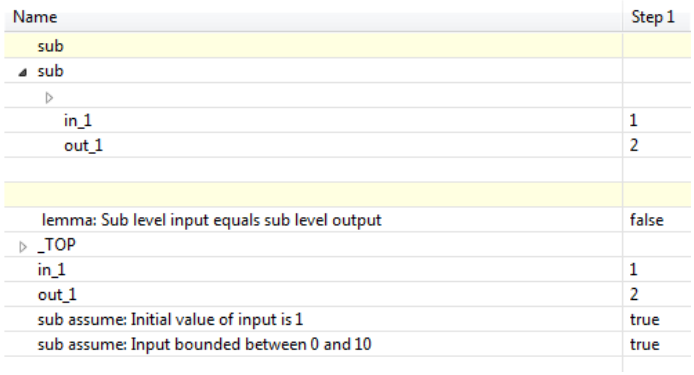

Figure 5: Verification for integer model without use of arrow operator

The counterexamples existed in the initial state of the system and showed for both of these failed contracts that when we do not specify initial state in the top level properties, the inputs follow the assumptions (equal to 1), but the output is unconstrained. See Figures 6 and 7.



| Name | Step 1 |
|---|---|
| sub | |
| ◢ sub | |
| ▷ | |
| in_1 | 1 |
| out_1 | 2 |
| | |
| | |
| lemma: Input equals output (top level) | false |
| ▷ _TOP | |
| in_1 | 1 |
| out_1 | 2 |
| sub assume: Initial value of input is 1 | true |
| sub assume: Input bounded between 0 and 10 | true |

Figure 6: Counterexample for top level input equals top level output



| Name | Step 1 |
|---|---|
| sub | |
| ◢ sub | |
| ▷ | |
| in_1 | 1 |
| out_1 | 2 |
| | |
| | |
| lemma: Sub level input equals sub level output | false |
| ▷ _TOP | |
| in_1 | 1 |
| out_1 | 2 |
| sub assume: Initial value of input is 1 | true |
| sub assume: Input bounded between 0 and 10 | true |

Figure 7: Counterexample for sub level input equals sub level output

I then changed the subcomponent contract regarding the output to *exclude* the arrow operator. Thus the guarantee was then simply:

With this in place, all lemmas prove at the top level.

Thus when using the arrow operator and defining a particular initial state, any behavior that should hold in ALL states should not have the arrow operator. It is not required to be consistent through the model, just correctly define the

```
guarantee "Input equals output." :
    (in_1 = out_1);
```

Figure 8: Countract on output without arrow operator

inital state. If it is not important what the behavior of a component is in the initial state, then we can use "true →". If it is important, it is sufficient to eliminate that operator. This causes the guarantee to apply in all states.

# 4 Thoughts

Scoping is pretty clear. As long as the connections are in place, the scope is what we thought it was. If the connections are nonexistant in the model, then the scope and behavior of component in scope determines proof of contract in question.

Of course initial behavior adds some complexity to the behavior model, but one thing that seems important is this: Our top level lemmas should prove in every state including initial state. If we add initial state behavior to a model through the arrow operator, it might be prudent to keep the arrow operator out of the top level contracts. This forces us to implement correct initial behavior in all of the pertinant subcomponents whose contracts are used in the proof of the top level properties.