Width Inference Documentation

Bert Rodiers
Ben Lickly

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-120
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-120.html

August 24, 2010

Acknowledgement

This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #CCR-0225610 (ITR), #0720882 (CSR-EHS: PRET) and #0931843 (ActionWebs)), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312 and AF-TRUST #FA9550-06-1-0244), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC) and the following companies: Bosch, National Instruments, Thales, and Toyota.

Width Inference Documentation

Bert Rodiers, Ben Lickly

August 24, 2010

1 Introduction

In Ptolemy II actors are connected to each other with relations. A relation can be a bus with multiple channels, and we call the number of channels the *width* of the relation. The width of a relation determines the number of tokens that can be transported in parallel through that relation.

In Figure 1, a Ptolemy model is shown that has a hierarchy of three levels. In this model the tokens coming from the three source actors at the toplevel are added to each other and finally the end result is displayed in the Display actor. The toplevel has a CompositeActor. The port called *port* of this actor has three relations connected to it on the outside and one relation on the inside. If the user does not want to lose any data or create channels that don't transport any data, the only sensible width for the relation on the inside is 3. If a model builder added an extra source actor in the model, however, this value would need to be updated.

Having to explicitly specify these widths is a tedious job. To make matters worse, the addition of one relation to a multiport might force the model builder to go through the entire model again to adapt widths.

To cope with this the user can set the width parameter of relations to *Auto*. This causes the width of the relation to be automatically inferred from the other relations and other ports.

2 The algorithm

There are many possible algorithms to determine the widths of relations. On every multiport you have relations between the sum of the widths of the

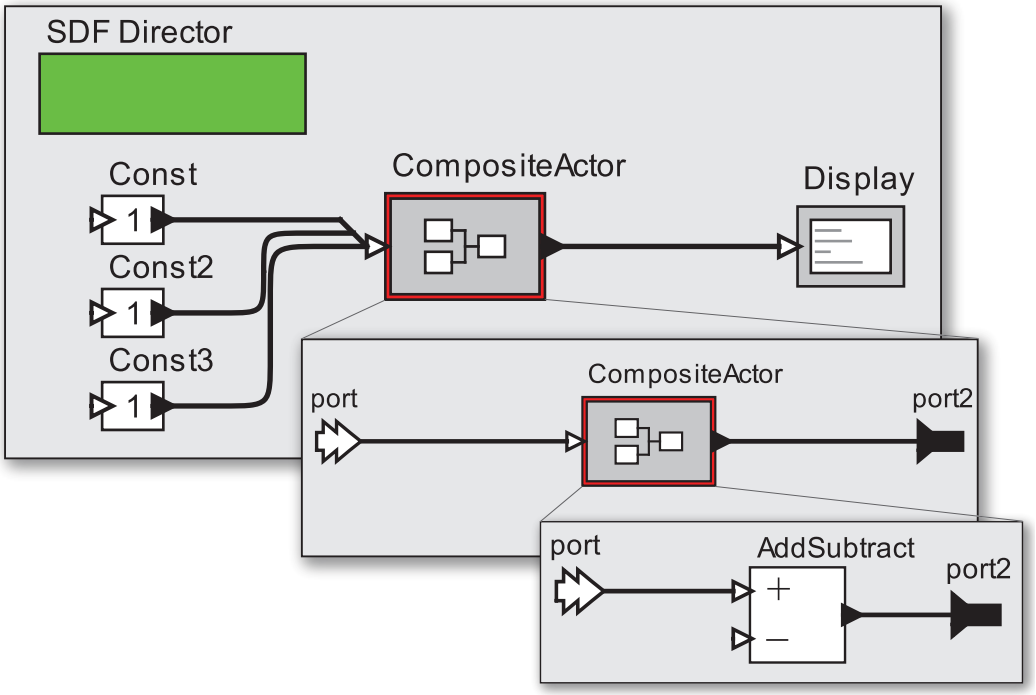

Figure 1: A hierarchical Ptolemy II model to illustrate width inference.

incoming and the sum of the widths of outgoing relations. This could be written as a system of constraints that form an Integer Linear Programming problem, or formulated as a problem for a General Propagation-based constraint solver [1]. Solving this problem with either of these two methods, however, would require solving an NP-complete problem. Another possibility would be to use a fixed point algorithm over the lattice of the ordered natural numbers, trying to find a minimum width without losing any tokens. Using this method we would have no guarantees as to how fast the algorithm would converge or even whether it would converge at all.

We opt instead for a graph algorithm where the width of the relation is inferred from other relations or other ports. The algorithm propagates the widths it knows and at multiports tries to infer the widths of other relations from these propagated widths.

A simplified version of the algorithm can be found below:

inferWidths()
  $unspecifiedSet \leftarrow$ all relations that need to be inferred
  $workingSet \leftarrow$ all known relations connected to a relation that needs to be inferred
  while $|workingSet| > 0 \wedge |unspecifiedSet| > 0$ **do**
    $relation \leftarrow workingSet.pop()$
    for $port \in$ multiports connected to $relation$ **do**
      $updatedRelations \leftarrow$ **updateRelationsAt**($port$)
      $workingSet \leftarrow workingSet \cup updatedRelations$
      $unspecifiedSet \leftarrow unspecifiedSet \setminus updatedRelations$
    end for
  end while
  if $|unspecifiedSet| > 0$ **then**
    Error: Could not infer relations remaining in $unspecifiedSet$
  end if

updateRelationsAt($port$)
  if all relations connected to one side of $port$ have known width **then**
    $s \leftarrow$ that side
    $s' \leftarrow$ the other side (with unknown width relations)
    $difference \leftarrow$ widths of known relations at s − widths of known relations at s'
    if $difference < 0$ **then**
      Error: This model would infer negative widths

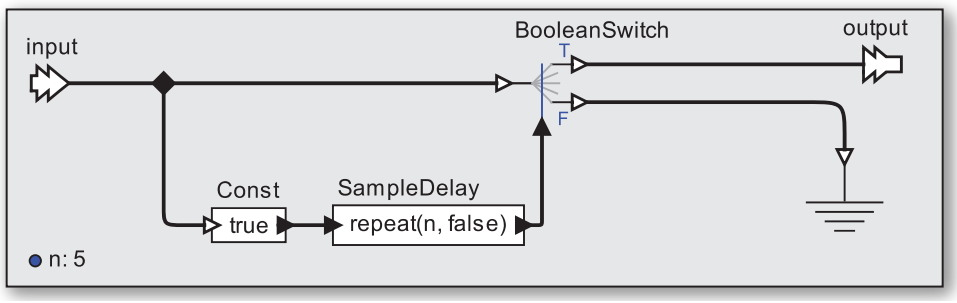

Figure 2: A model to illustrate the width constraints for the BooleanSwitch actor.

else if $difference = 0$ **then**
the widths of all relations at $s' \leftarrow 0$
else if there is only one unspecified relation at s' **then**
That relation has a width equal to $difference$
end if
end if
return Set of inferred relations

The algorithm starts with the known relations that border relations that need to be inferred, and works its way inward by locally inferring the widths at multiports that it encounters.

3 Width constraints on ports

If we define the width of a port of an Atomic Actor as the sum of the widths of the connected relations, then for certain actors there exists a relation between the widths of their input ports and the widths of their output ports.

Let's take a look at Figure 2. In the figure you can see that the False output port (denoted with F in the figure) of BooleanSwitch is connected to a Discard actor. The Discard actor will discard all incoming tokens. Since the input port of the Discard actor and the output port of BooleanSelect are both multiports, the width inference algorithm can't directly infer the width of the relation between both actors. To resolve this problem, *width constraints on ports* have been introduced. Currently there exist two types of width constraints. The first one imposes an equality constraint between

two different ports. For example you can write

```
outputPort.setWidthEquals(inputPort, true);
```

This says that the width of the port *outputPort* has to be equal to the width of the port *inputPort*.

The boolean value determines whether the constraint should be applied in two directions are not. In normal occasions the value should be true. With the second type of constraint you can set the width of a port equal to a certain *Parameter*. For example

```
Parameter parameter = ...;
outputPort.setWidthEquals(parameter);
```

An example of this type of width constraints is the Ptalon actor *MapFileStorage*, a DE actor that stores data from a MapWorker and distributes it to a ReduceWorker upon request. This component has a parameter called *numberOfOutputs* which is the number of output actors to write to. The ports *outputKey* and *outputValue* should have a width equal to this parameter and hence this actor will add the following constraints in its constructor:

```
outputKey.setWidthEquals(numberOfOutputs);
outputValue.setWidthEquals(numberOfOutputs);
```

In order to preserve these constraints when cloned, it must also add constraints to the colne method:

```
public Object clone(Workspace workspace) throws
    CloneNotSupportedException {
     MapFileStorage newObject = (MapFileStorage) super.clone(
         workspace);
     newObject.outputKey.setWidthEquals(newObject.
         numberOfOutputs);
     newObject.outputValue.setWidthEquals(newObject.
         numberOfOutputs);
     return newObject;
}
```

In the implementation of an actor these constraints should be made explicit if they exist. This should be done both in the constructor and in the clone method. For example for the BooleanSelect the following lines have been added to the constructor

```
output.setWidthEquals(trueInput, true);
output.setWidthEquals(falseInput, true);
```

and the following lines have been added to the clone method

```
newObject.output.setWidthEquals(newObject.trueInput, true);
newObject.output.setWidthEquals(newObject.falseInput, true);
```

The width inference algorithm will first infer the width of relations at multiports using the algorithm described in Section 2. Only when no valid width can be inferred with that algorithm will it use the width constraints. Then these newly inferred relation widths are propagated again using the algorithm in Section 2. This is important in case there are inconsistencies. See Section 5 for more details about inconsistencies and the inconsistency checks and Section 7 for information about how disable the inconsistency checks.

Other examples of actors with width constraints are the Publisher and Subscriber.

4 Default widths for ports

In Ptolemy II a number of actors have been written that have a multiport output for which all channels carry the same data (the values are broadcasted on the port). Using multiports this way is unnecessary and is now considered to be bad design practice. To avoid breaking existing models, however, these actors have remained unchanged. One example of such an actor is the Minimum actor.

In Figure 3 you can see a model with such a Minimum actor. The width inference algorithm is not able to infer the width of the relation between the Minimum actor and the Display actor. To cope with this *default widths* have been introduced. In its constructor the actor can let the width of its output port default to 1, so that in case the width inference algorithm can't infer the width of a connected relation it will use the default width of the port to infer the widths of the connected relations. Unlike width constraints, default widths do not need to be set again in the *clone* method.

This code fragment illustrates how to specify default constraints:

```
minimumValue = new TypedIOPort(this, "minimumValue",
                                false, true);
...
minimumValue.setDefaultWidth(1);
```

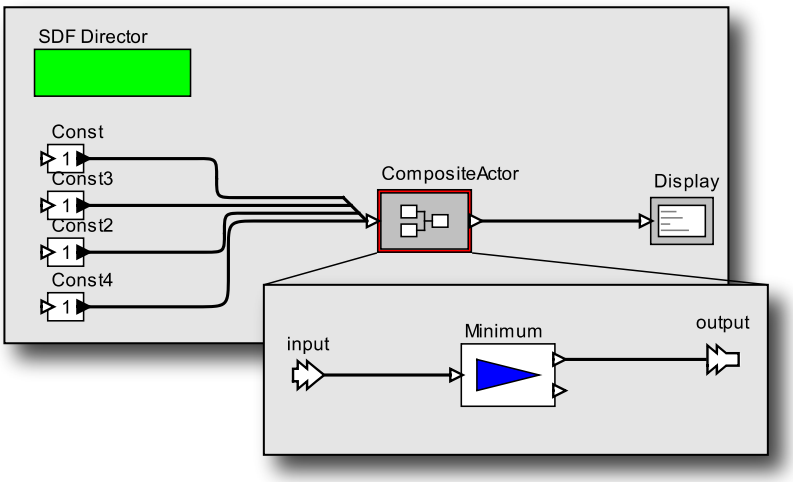

Figure 3: A model to illustrate the default widths.

In case the relations connected to such a port can be inferred the default value will not be used.

5 Consistency Check

In the previous section we discussed how to deal with the case when our algorithm can not infer the width of all of the relations in the model. Another case we address is when our algorithm produces a potential solution that is actually erroneous. There are two ways that this can happen.

One is that after the end of the algorithm there are relations remaining in the *workingSet* that do not agree with the inferred value of a relation adjacent on the multiport of a CompositeActor. This can be constructed in Ptolemy II by, for example, creating a relation to be inferred between two multiports of different widths. Note that some composite actors don't have any visible inside relations but are still are composite actors. In this case the outside width does not need to be equal to the inside one (which is zero).

The other way that a potential solution can be erroneous is if it violates the width constraints for a given actor. In this case we call the model consistent if the (outside) width of the port is equal to the value obtained by the width constraint (if the width constraint would have been used).

By default, if either of these two checks fails, then an error is returned. In some cases, a user may want to force models that fail these consistency checks to simply use the inferred widths. Section 7 explains how to do this. Be aware, however, that doing so can cause inferred widths to be resolved non-deterministically, since they may depend on the internal ordering of relations in the inference algorithm.

6 Backward compatibility

In Ptolemy II release 8.0 a number of things have been changed related to the widths of relations. In a previous implementation of the width inference algorithm, the value to infer the widths was *0*. This has been changed to *Auto* or *-1* to add the possibility to disable part of the graph by using a width equal to zero. Another change is related to the default width. This used to be *1*, but has been changed to *Auto*. Hence for new models the widths of relations are by default inferred. The default width is not stored in the model (to reduce the size of the MoML file).

To not break existing models, when opening an older model all widths equal to *0* are changed to *Auto*. Furthermore if the width is not specified in the MoML file (the default width was used), we change the width explicitly to 1 to not break the model (for some patterns the algorithm can't uniquely infer the width of a relations and hence we don't want to always infer the widths of all relations).

7 Width inference options

Let's us take a look at Figure 4. In this figure you can see *Publisher* and a *Subscriber*. A Publisher "publishes" under a certain name all the tokens that arrive at its input port and a subscriber can use the same name to get those tokens and send them to its output port. The link between Publisher and Subscriber can be seen as a hidden relation (as if the Publisher and Subscriber were not there). Since there are 2 sources connected to the Publisher port, this port has width 2. You would hence expect that the output port and the connected relation also have width 2. However this relation connected to the CompositeActor, which is connected with an actor with a non multiport. Hence you would expect that the width to be one. This is what we call an

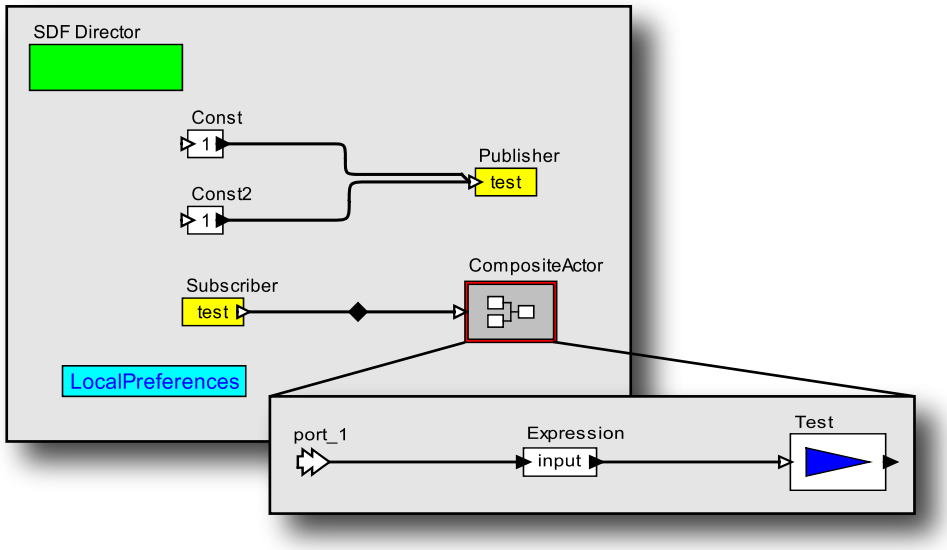

Figure 4: This model has inconsistent widths at *port_1* of *CompositeActor*.

inconsistent model from a width inference point of view. This will be flagged as an error by the width inference algorithm since it can't know whether the model might have unexpected side-effects or not. If as a user, you want to live with this inconsistency you can disable the consistency check. This can be done in the LocalPreferences dialog in Figure 5. In this example the option to check width constraints has been switched off. This is because Publishers and Subscribers use width constraints to infer the widths of the connected relations.

There are also checks on the inside and outside widths of multiports (if there is at least one relation connected for which the width needs to be inferred). For some models users might want to disable this option and cope with the inconsistency. This can be done by toggling the first checkbox in the LocalPreferences dialog.

A third width-related option allows the use of width 1 as default width for all relations. This value will only be used in case the width of a relation cannot be inferred. This option is added since a width of one used to be the default case. Users who are used to this behavior can still use this option, although it may hide inconsistencies in the model.

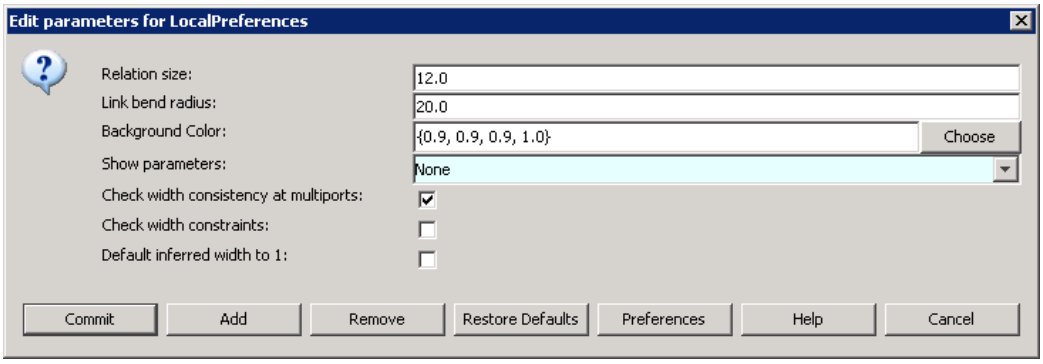

Figure 5: LocalPreferences dialog.

8 Dealing with old models

With a previous version of the width inference algorithm, widths were not inferred when there was more than one relation connected to a multiport for which the width needed to be inferred. If the width needs to be inferred for relations i and j in Figure 6, the previous algorithm would bail out since both relations are connected to the same multiport. To cope with this users could set the width one of the relations to a sufficiently large number that it would not need to change when the model was changed (when for example a source was added in our example). In the current algorithm this will be flagged as an error if the width of relation j has a fixed value (different from 3) since the width of relation i is not uniquely defined.

Solution: Change the width to *Auto* so it will be inferred.

References

[1] Guido Tack. *Constraint Propagation – Models, Techniques, Implementation*. Doctoral dissertation, Saarland University, January 2009.

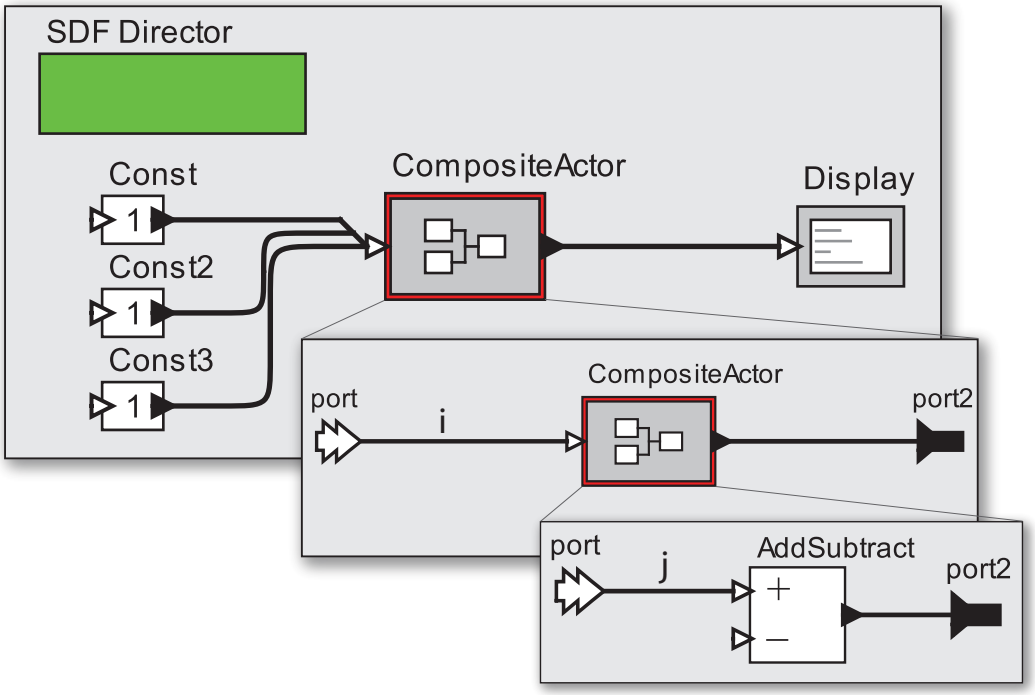

Figure 6: A hierarchical Ptolemy II model to illustrate width inference.