New Width Inference Algorithm for Ptolemy II

Ben Lickly
Bert Rodiers

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-13
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-13.html

January 26, 2009

New Width Inference Algorithm for Ptolemy II

Ben Lickly, Bert Rodiers
Center for Hybrid and Embedded Software Systems
University of California, Berkeley
{blickly,rodiers}@eecs.berkeley.edu

January 26, 2009

Abstract

In Ptolemy II, the widths of the relations between actors have an important role in specifying the semantics of a model. In some situations, however, explicit relation widths break the modularity of a model. In these cases, we would like to infer the widths of relations from the widths of neighboring relations. There is an existing implementation in Ptolemy, but it has some severe limitations.

In this paper, we present the general needs for width inference in Ptolemy II, as well as the weaknesses of the current algorithm. We argue that a better algorithm is necessary, present a possible new algorithm, and analyze its performance over variety of metrics.

1 Introduction

A model of computation is a system for describing how computations can be performed. Common models of computation include Turing machines, finite state machines, Kahn process networks, and discrete event systems. In the domain of embedded systems, models of computation less powerful than Turing machines are a useful object of study because they may be more analyzable, and properties such as termination may be provable. Other models of computation may be useful to study because of the way they deal with concurrency, composition, or scheduling.

1.1 Ptolemy II

Ptolemy II[1] is a framework in which heterogeneous models of computation can be combined and simulated.

Many of these models of computation are actor-based models. Here, the primitive components of computation are called *actors* and they are able to communicate between each other along explicitly defined channels. These actors are able to execute in a conceptually

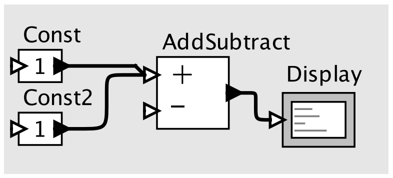

Figure 1: Simple example of addition in the actor model, with two relations connected to the add multiport. Notice that multiports are white and single width ports are black.

concurrent manner subject to certain constraints given by the model of computation. Another important feature of Ptolemy II is hierarchical composition of models of computation, in which the specification of one actor may be a separate Ptolemy model operating under a different model of computation. This allows for heterogeneous composition of models of computation.

Unlike in object-oriented design, in which components interact by method calls, actors interact through channels called *relations*. Data, encapsulated in *tokens*, can be sent between actors along relations, from the input and output *ports* of an actor.

Some ports, called *multiports* may be connected to more than one relation, enabling them to send or receive multiple tokens at a time. A simple example of a multiport is in the AddSubtract actor, as shown in Figure 1. If multiple relations are connected to the Add multiport, it will read tokens from all of the relations and add them together.

When hierarchical models are allowed, the existence of ports with a width greater than 1 also means that relations of width greater than 1 must be supported. For example, in Figure 2 the relation between the port (which is a multiport) and the AddSubtract actor should be 2 in order to carry the data from both constant actors to the AddSubtract actor. The default width for relations is 1, so in order to create a width 2 relation the user must manually change the width.

In order to send data from one actor to another, data encapsulated in *tokens* can be sent along relations. Many actors are only able to produce one token at a time (single execution), but in some cases an actor may produce multiple tokens conceptually concurrently. Relations between actors may also specify their *width*, or the number of concurrent tokens that they can accept. The width has no effect on the total number of tokens that a relation may hold, but for a relation of width n, only n tokens may be concurrently placed on that relation.

1.2 Width Inference

One desirable feature would to be to have models that can determine the widths of their own relations. For example, the creator of the AddSubtractCompositeActor in Figure 2 may not know how many relations will be connected to its input port. In this case she may want to have the width of the inside relation to be inferred by looking at the combined widths of the relations connected to the outside of the port. The current release of Ptolemy II supports just this type of width inference, in which the user is able to specify certain relations to be

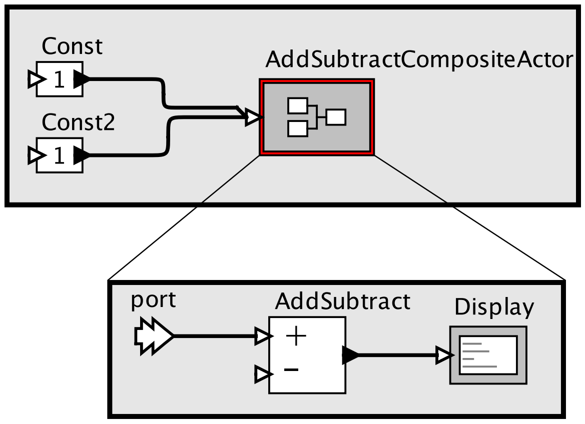

Figure 2: Example in which hierarchy necessitates a width 2 relation between the port and the AddSubtract actor.

automatically inferred. The current algorithm is somewhat simple, however, in that it does not try to deal with cases where more than relation connected to the same port tries to be inferred. Unfortunately, the are many reasonable cases in which multiple relations connected to the same port must be inferred, such as inferring across multiple levels of hierarchy, or inferring multiple branches at a junction.

For example, while the current implementation works for the example shown in Figure 2, it fails on the only slightly more complicated example in Figure 3. In this example, another layer of hierarchy has been added, and the algorithm is asked to infer the lengths of the two inner relations. This discrepancy arises because the first example requires only a single relation to be inferred, whereas in the second example there are two relations both connected to *port*2 that need to be inferred.

2 Problem Statement

This project concerned a more powerful algorithm for width inference. This will likely need to balance the desire for a more capable functionality against the requirements of performance and backward compatibility.

2.1 Constraints

The following constraints must be addressed in design of the new width inference algorithm.

Ports of non-matching width In Ptolemy II it is considered valid for a model to have ports in which the width of the input relations is not equal to the width of the output relations. In this case, the number of tokens that get written on a firing is the smaller of the two relations. This makes width inference much more difficult because this

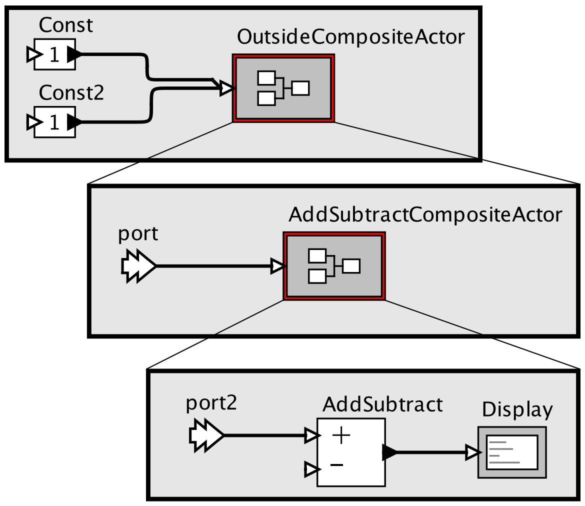

Figure 3: Example in which current inference algorithm fails.

means that a relation may have multiple different widths that would both produce a valid model. An inference algorithm will need to allow these types of relations, and may have to determine which width assignments are preferred over others.

Performance Constraints Ptolemy II is used to create and run complex models with thousands of actors, and it is very important that the performance does not degrade significantly even on these very large models. The current width inference algorithm, while lacks ability to make complicated inferences, does have the property that it is able to infer quickly, partially because it runs incrementally as the model is being built. One undesirable result of this approach is that it may introduce errors that are false positives while the model is being built. A width inference algorithm that runs at run time, rather than incrementally at model run time has the potential to handle more cases by taking a global view, but it also means that running the model would incur additional overhead.

Infer by Default While the current default width for newly-created relations is 1, it may be preferable to have the default be to infer the widths. In order for this to be possible, the new algorithm must be powerful enough to infer widths for most reasonable graphs and give useful errors in the cases where it is not able to infer widths. It will also be important that the performance of a model in which most relations are inferred does not significantly degrade over a model with explicitly declared relations.

3 Resulting Algorithm

There are many ways to clearly describe a desired inference behavior, such as an integer linear programming or a system of linear equations. Some formulations are even able to handle ambiguities in which more than one assignment (for example by favoring smaller widths in the objective function). Unfortunately, our performance constraints force us to choose a faster algorithm over a slower, more full-feature one. Rather than choosing a possibly slower algorithm that is able to make global decisions, we chose an approach that can make all its decisions based on the local state of adjacent relations. The pseudocode of this algorithm is as follows: [1]

```
**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** $|workingSet| > 0 \land |unspecifiedSet| > 0$ **do**
    $relation \leftarrow workingSet.pop()$
    $unspecifiedSet \leftarrow unspecifiedSet \setminus \{relation\}$
    **for** $port \in$ multiports connected to $relation$ **do**
      $updatedRelations \leftarrow$ **updateRelationsAt**$(port)$
      $workingSet \leftarrow workingSet \cup updatedRelations$
    **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
    **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. Due to the local nature of the algorithm, it is not able to deal with ambiguous

[1] The final Ptolemy implementation is slightly more complicated than this. In particular, to allow more flexible inference, there is a special case when $difference$ is equal to the number of remaining relations. Since this causes the width inference to be non-deterministic for certain pathological cases, that special case is not included in the algorithm used here.

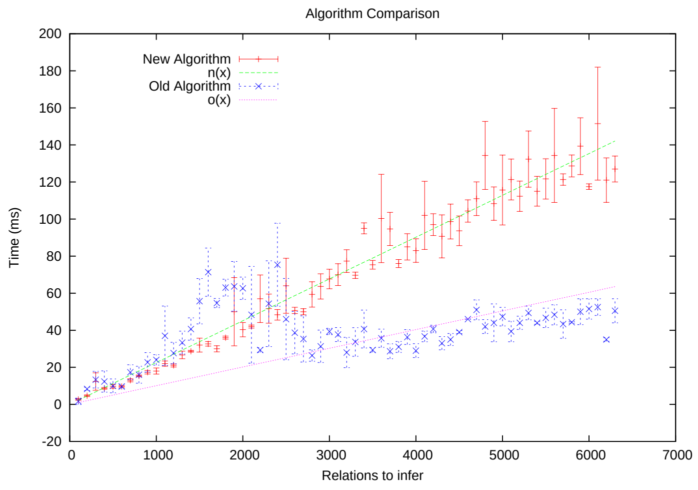

Figure 4: Run times of the new and old width inference algorithm.

inference situations. This means that it can only solve inference graphs for which the shape of inferred relations is a tree. This is because any (non-directed) cycle introduces ambiguities.

We have preserved backwards compatibility for ports to have non-matching width by ignoring relations that are specified explicitly. The inferred ports are required to have matching width, but most existing models with non-matching widths have the widths set explicitly, so this requirement should not break them.

3.1 Results

Since each multiport can only be considered once for each relation, and each relation is only considered once, this algorithm should be linear in the number of inferred relations. To test this hypothesis, we use the model transformation domain [2] to generate models with large numbers of inferred relations, and examine how long it takes to infer the widths of these models.

First we constructed a simple type of model that has no two inferred relation sharing a port. This configuration is simple enough that the old inference algorithm would work, and allows us to compare the performance of the old and new algorithms. We should expect that the new algorithm will be slower, since it is more powerful, and comparing with the old algorithm gives us a baseline to compare with. Figure 4 shows the inference time for both width inference algorithms on these simple models. Even though the new algorithm is slower for large models, it still appears to be proportional to the number of relations to be inferred.

In addition to the simple example that could be inferred by both algorithms, we also

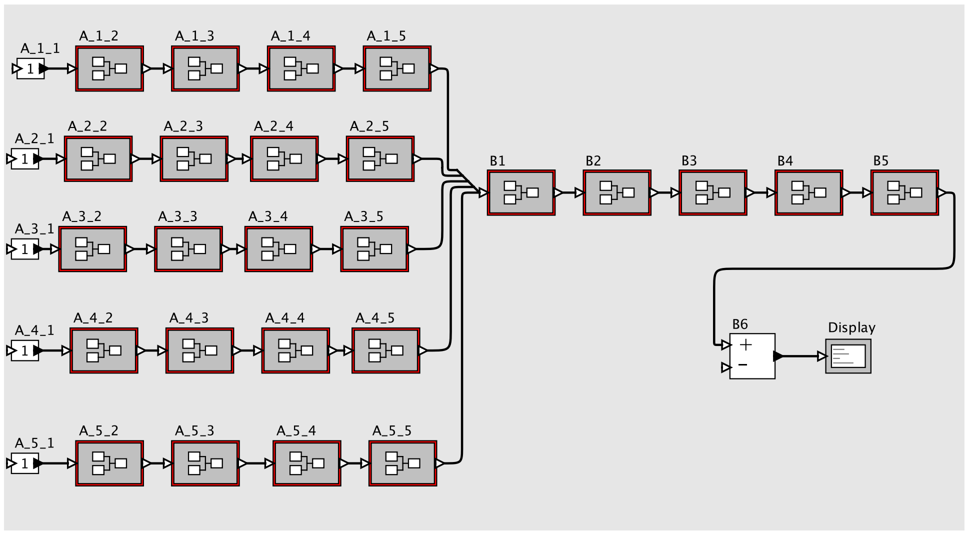

Figure 5: Example of Spider pattern with the number of iterations set to five. In larger graphs, more relations are connected to the input port of $B1$.

looked at the performance of more complicated examples designed to be more difficult to infer. The two types of examples are the Spider pattern, as seen in Figure 5, and the Ladder pattern, as seen in Figure 6. At the center of the Spider pattern, there is one port that has many inferred relations connected to it, and the Ladder pattern is specifically constructed so that there is only one port at a given time which can successfully be inferred from. In Figure 7 we can see how the runtime of the width inference algorithm grows as the size of the Spider and Ladder graphs grow. Since the algorithm simply looks through relations in the working set until it finds a port that it can infer from, the Ladder models should be a pathological case for the new algorithm. In effect, the Ladder graphs have many false leads in their working sets, which explains why they take longer to run width inference.

One caveat is that although the algorithm itself has a running time that is linear in the number of relations to be inferred, it also requires proper initialization in order to know which relations are to be inferred. In order to properly initialize $unspecifiedSet$ before the algorithm begins, the algorithm searches through all the relations in the graph, which is linear in the total number of relations in the graph. This means that models with few inferred relations suffer a performance penalty over the old algorithm. This problem can be visualized easily in Figure 8, which shows the run time of both the initialization and the width inference for a model with no inferred relations. Even though the actual algorithm run time does not increase as uninferred relations are added, the initialization time does.

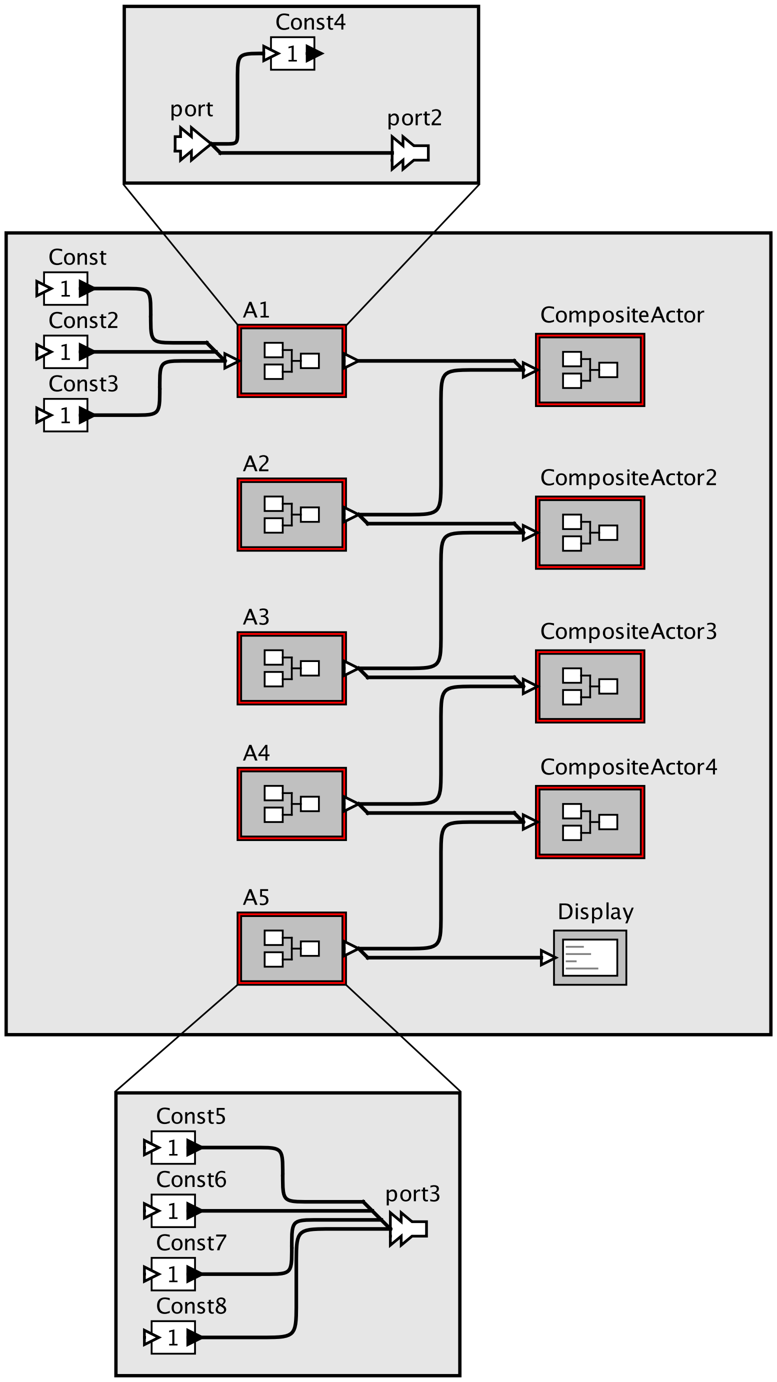

Figure 6: Example of Spider pattern with the number of iterations set to five. Note that the relation connected to the display is specified, allowing width inference to proceed from the bottom up.

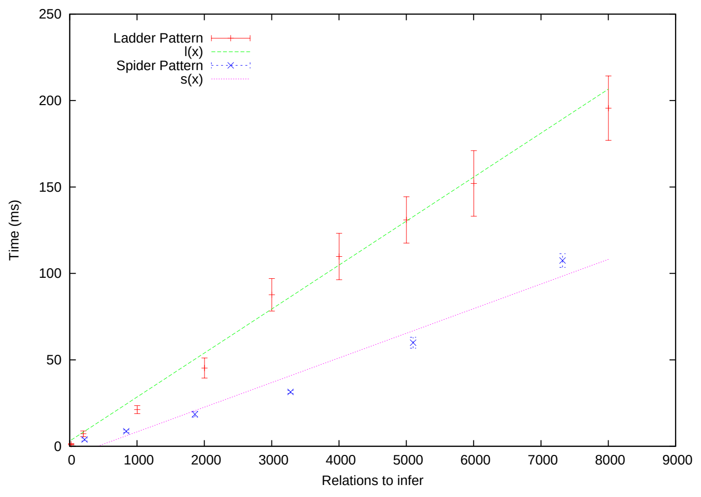

Figure 7: Run times of the Spider models and the Ladder models with respect to the number of relations to infer.

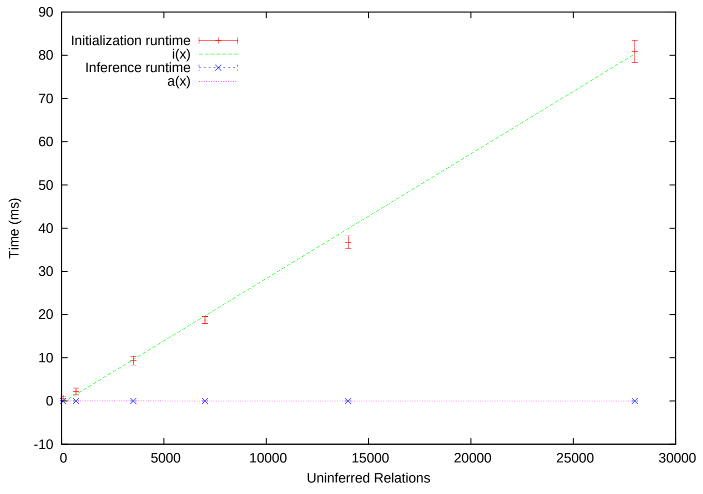

Figure 8: Comparison of the initialization time and run time for a model in which no relations need to be inferred.

4 Conclusion

The new algorithm has the main advantage over the old algorithm that it is able to successfully infer widths in many more cases. Compared to other possible choices of replacement algorithms, it also has the advantage that its performance is still linear in the number of relations to infer. The new algorithm does run slower than the old algorithm, but this is to be expected given its greater expressiveness. Since the new algorithm is unable to deal correctly with cycles, there are still models for which it can not infer all widths. It is probably not feasible to infer by default with the new algorithm, although it would work much more effectively than the previous algorithm.

The main disadvantage over the old algorithm is the initialization time penalty, which causes unnecessary slowdown for models with few inferred relations. This could be addressed by removing the initialization phase and only only inferring widths on demand. When a width is inferred, the algorithm could check all of the connected inferred relations and infer them together. This is further work.

One other area to investigate further would be the behavior of the algorithm for even larger models. The model transformation domain allows large parametric models to be expressed very succinctly, but the performance limitations made generating models a bottleneck. This ruled out testing with models of more than a few thousand actors. It is likely that another method of generating models would be able to produce larger models. This may be able to produce graphs which give us more confidence that the algorithm is linear.

Acknowledgments

This paper would not have been possible with out the help of Thomas Feng, who gave advice, quick fixes, and general pointing in the right for everything in the model transformation domain.

References

[1] J. Eker, J.W. Janneck, E.A. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, Jan 2003.

[2] Thomas Huining Feng and Edward A. Lee. Scalable models using model transformation. Technical Report UCB/EECS-2008-85, EECS Department, University of California, Berkeley, Jul 2008.