Non-Causal Equations in μModelica

Andrew Casey

McGill University acasey@cs.mcgill.ca

Abstract. We investigate the system used by the μModelica compiler to solve statically specified sets of non-causal equations by a combination of symbolic manipulation and numerical solving (described in [5]). Equations are simplified and canonicalized, made causal (by matching variables to equations), sorted (with respect to dependencies), and then solved. We hope that this system will extend to dynamic sets of non-causal equations.

1 Introduction

While numerical solvers of non-causal, non-linear equations are very effective, they are at times inefficient. The μModelica compiler (described in [5]) reduces the amount of work that has to be done by a numerical solver by pre-processing the set of non-causal equations and simplifying them, where possible. It accomplishes this in steps.

First, each equation is simplified and rewritten in a canonical representation. Constants are folded, like-terms are gathered, and operands (of associative and commutative operators) are re-ordered. This reduces the amount of work that the numerical solver has to perform and facilitates the following steps.

Second, if possible, μModelica converts each equation to causal form. That is, it attempts to rearrange each equation to isolate a single variable. The problem of determining which variable to isolate in which equation reduces to a matching problem which can be solved using Dinic's algorithm for maximum network flow through a graph.

Third, the equations are re-ordered according to their inter-dependencies. That is, the equations are arranged so that each equation contains only a single unknown when it is evaluated. In some cases, this is impossible because of cyclic dependencies. Such strongly connected components in the dependency graph are marked and treated as single entities in the overall order.

Finally, the equations are solved in order. Those not having cyclic dependencies are evaluated directly by substituting known values. The others are passed to a numeric solver as-is. This may result in multiple calls to the numeric solver.

2 Canonical Representation

Initially, an equation is represented as a tree with constants and variables at the leaves and operators at the internal nodes. This tree is very similar to the

abstract syntax tree produced by a parser. Clearly, a given expression admits many such representations. This presents two problems. First, the representation maybe be unwieldy – it may contain obviously unsimplified subtrees (e.g. with unfolded constants). Second, it is difficult to determine whether two expressions are equal if their representations are different. Canonicalizing all representations resolves the second problem. Furthermore, if the canonical representation requires simplification, then it resolves the first problem as well.

Unfortunately, expression simplification is a difficult problem ([3]). Since there is no "simplest" form for a given equation, we accept that there is no optimal solution and settle for incremental improvements. Besides constant folding, the most important simplifications are:

1. $0 + expr \rightarrow expr$
2. $0 * expr \rightarrow 0$
3. $1 * expr \rightarrow expr$
4. $expr^1 \rightarrow expr$
5. $expr^0 \rightarrow 1$
6. $1^{expr} \rightarrow 1$
7. $a * expr + b * expr \rightarrow (a + b) * expr$
8. $(expr_1^{expr_2})^{expr_3} \rightarrow expr_1^{expr_2 * expr_3}$

These simplifications refer only to addition, multiplication, and exponentiation because it is possible to express subtraction and division in terms of these operations. Furthermore, since addition and multiplication are associative and commutative, they are treated as n-ary operators.

By subtracting the right-hand expression from the left-hand expression, one can reduce the equation to a single expression, which must be equal to zero. It is to this unified equation that the simplifications are applied.

Simplification is performed repeatedly until a fixed point is reached. However, not all simplifications need to be performed on every iteration (see [5] and [4]).

Once an expression is simplified, it can be "canonicalized" by recursively sorting the children of each internal node into a predefined order (e.g. constants in numerical order, followed by variables in alphabetical order, etc). This does not guarantee that two equal expressions have the same canonical representation, but it is sufficient in many cases.

The precise simplification process used by μModelica is described in [5]. It is based on a more elaborate process described in [4].

3 Causality Assignment

To determine the value of a variable symbolically, one must have an equation with the isolated variable on one side and an expression containing only known variables on the other. (Without loss of generality, we will assume that the variable has been isolated on the left-hand side (LHS) of the equation, as opposed to the right-hand side (RHS).) Thus, in order to solve a set of non-causal equations symbolically, one must convert some or all of them to causal form.

However, causal equations are of no use if they are all solved for the same variable. Ideally, we would like to solve each equation for a separate variable. This is a straightforward matching problem which can be recast as a maximum network flow problem in a small bipartite graph. If we weight the edges uniformly, as μModelica does, then we can regard them all as having unit weight, in which case a more-efficient, specialized version of Dinic's algorithm can be applied.

The details of Dinic's algorithm are discussed in [5] and [2]. Broadly, however, the algorithm (as used for causality assignment) works as follows:[1]

1. Create a vertex for each equation and for each variable.
2. Create distinguished source and sink vertices.
3. Add an edge from the source to each equation.
4. Add an edge from each equation to each variable it contains (or, for which it can be solved).
5. Add an edge from each variable to the sink.
6. Use breadth first search to find a path from source to sink. If such a path exists, reverse each edge in the path and repeat 6. Otherwise, proceed to 7.
7. Given an equation u and a variable v, if there is an edge from v to u, then solve u for v.

Unfortunately, in some cases there are more variables than equations (an under-determined system) or more equations than variables (an over-determined system). If a system is under-determined, then the causality assignment will determine how many variables can be solved for, but it will be impossible to solve for all of them. If a system is over-determined, then a perfect causality assignment will allow us to solve for all variables and then we can check that the "extra" equations are satisfied by the solution.

It should be noted that in some cases an equation will contain a variable for which it is impossible to solve. For example, in a system without a logarithmic function, $3^x = 4$ cannot be solved for x. More generally, beyond degree 3, there is no general equation for solving for the variable in a univariate polynomial (e.g. $x^5+x^4+x^3+x^2+x+1=0$). In such cases, it might be advantageous to prevent matching between the equation and the variable (i.e. by removing the edge in the constructed graph).

In other cases, the simplifications used by μModelica may not reduce the number of occurrences of each variable to one. For example, the equation $(x+1)^2+(x-1)^2=10$ will contain two occurrences of x in simplified form because there is no simplification rule for expanding powers of sums. This complicates the solving process; if there is only one occurrence then solving is simply a matter of inverting all of the operations applied to the variable.

4 Equation Sorting

A causal equation is only useful if all the variables that it depends upon (i.e. those appearing on its RHS) have known values. Thus, it is important to ensure

[1] This is a rather liberal interpretation of the algorithm described by [5] and [2] and we accept full responsibility for any errors or omissions.

that a causal equation is processed after the equations which give its dependent variables their values.

μModelica accomplishes this by constructing a dependency graph. A causal equation depends upon any equation solved for a variable appearing on its RHS. Once the graph is constructed, a double depth-first search (DFS) will yield a list of strongly-connected components (SCCs) in topological order ([1]). That is, the equations will be ordered with respect to their dependencies. However, if some of the equations have circular dependencies, then they will be grouped together as SCCs.

Solving the equations is then a matter of processing the equations in order. For each individual equation, μModelica solves for the isolated variable by substituting known values for the variables on the RHS. Each SCC of equations is solved by a numerical solver. Once the solver returns from processing an SCC, the remaining equations are processed.

5 Other Considerations

There remain some points concerning which [5] fails to elaborate.

First, [5] fails to justify its choice of simplification rules and the order in which they are applied. It does not give any data or reasoning to indicate which are the most effective and why others are not included.

Second, [5] fails to include an argument for the existence of a fixed point of the simplification process. Some of the rules (e.g. distribution) actually increase the size of an expression, so it is not obviously a terminating process.

Third, [5] does not include a simplification rule for dropping a constant multiplier applied to all terms on the LHS (when the RHS is zero). This is not important for simplification, but it may prevent some obviously equal expressions from having identical canonical representations. Along these lines, [5] does not mention that the canonical representation is not truly canonical. This might be inferred from a careful reading of the simplification rules, but we feel that it deserves explicit mention.

Fourth, [5] does not seem to address the relative ordering of (for example) addition nodes. That is, a canonical ordering must ensure that a product of sums has a unique representation. In particular, one of the sums must always appear first. This can be resolved by applying sorting addition nodes according to a dictionary order on their operands.

Fifth, [5] does not indicate why, when applying Dinic's algorithm, all edges are given unit weights. It is not obvious to us that there are no cases where we would not want to give preference to certain matches.

Finally, [5] fails to discuss a number of failure cases in which degraded success is still possible. In particular, under- and over-determined systems, equations which cannot be solved for any variable, under-determined cycles in the dependency graph, and equations with multiple solutions seem to require additional comment.

6 Conclusions

By pre-processing a set of non-causal equations, μModelica greatly reduces the amount of work that has to be done by the numeric solver that must eventually be used on some implicit equations. Even if it is impossible to solve for any of the variables symbolically, μModelica may be able to break the system up into smaller systems, thereby reducing the runtime.

7 Future Work

Our goal is to extend this work to dynamic sets of non-causal equations. That is, we would like to provide an interface that allows clients to add and remove non-causal equations at runtime. This raises two important concerns.

First, we would like to be able to perform the steps described above incrementally. It would be unfortunate if the entire process had to be re-performed each time a client added or removed a single equation. If it is not, in general, possible to efficiently update an existing causality assignment or equation ordering, then there may be important special cases where sub-optimal updates yield acceptable results quickly.

Second, μModelica can take time to achieve optimal results because it performs its task only once, at compile-time. However, in a dynamic system simplification must be performed repeatedly, so speed is more important than perfect accuracy. It may be beneficial to reduce the number of simplification passes if the marginal benefit is less than the marginal cost. In particular, it may be that a constant number of passes is more performant than a fixed point computation.

Another possibility for improving the responsiveness of a dynamic system is to parallelize equation solving. Since the equations are actually partially ordered, it should be possible to solve some of them concurrently.

Acknowledgments. We gratefully acknowledge the help of Hans Vangheluwe, who proposed this investigation and provided useful references.

References

1. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford: Introduction to Algorithms, Second Edition. Sect. 22.5. MIT Press, Cambridge, MA (2001)
2. Dinic, E.A.: Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Math. Dokl., 11, 1277-1280 (1970)
3. Indrani, A.V.: Some issues concerning Computer Algebra in AToM^3. Technical Report, School of Computer Science, McGill University, Montreal, QC (2003)
4. Vangheluwe, Hans, Sridharan, Bhama, Indrani, A.V.: An algorithm to implement a canonical representation of algebraic expressions and equations in AToM^3. Technical Report, School of Computer Science, McGill University, Montreal, QC (2003)
5. Xu, Weigao "Steven": The Design and Implementation of the μModelica compiler. MSc. Thesis. School of Computer Science, McGill University, Montreal, QC (2005)