# Source-Level Transformations for Improved Formal Verification

by

Brian D. Winters

B.S., California Institute of Technology, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

**Master of Science**

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

**The University of British Columbia**

August 2000

# Abstract

A major obstacle to widespread acceptance of formal verification is the difficulty in using the tools effectively. Although learning the basic syntax and operation of a formal verification tool may be easy, expert users are often able to accomplish a verification task while a novice user encounters time-out or space-out attempting the same task. In this thesis, we assert that often a novice user will model a system in a different manner — semantically equivalent, but less efficient for the verification tool — than an expert user would, that some of these inefficient modeling choices can be easily detected at the source-code level, and that a robust verification tool should identify these inefficiencies and optimize them, thereby helping to close the gap between novice and expert users. To test our hypothesis, we propose some possible optimizations for the Mur$\varphi$ verification system, implement one of these, and compare the results on a variety of examples written by both experts and novices (the Mur$\varphi$ distribution examples, a set of cache coherence protocol models, and a portion of the IEEE 1394 Firewire protocol). The results support our assertion — a nontrivial fraction of the Mur$\varphi$ models written by novice users were significantly accelerated by the single optimization. Our findings strongly support further research in this area.

# Contents

# List of Tables

# List of Figures

# Acknowledgments

This work would not have been possible without extensive support and encouragement from Alan Hu.

I owe many individuals a debt of gratitude for helping me to find this path, and for making the journey more enjoyable. Most important of these were Wendy Belluomini, who provided invaluable advice and guidance, and H. Peter Hofstee, who taught me many valuable lessons and was always generous in his support.

Lastly, I might have missed the boat completely, had Tom Zavisca not twisted my arm at the right time.

Brian D. Winters

*The University of British Columbia*

*August 2000*

To my parents, for their endless patience and support.

# Chapter 1

# Introduction

A major obstacle to widespread acceptance of formal verification is the difficulty in using the tools effectively. Ideally, the tools should be accessible to non-experts, so that formal verification can be used as just another technique to aid design and verification. While some effort learning a tool can be expected, the user must not be expected to have a deep understanding of the specific algorithms and heuristics used by the tool. In reality, the situation is markedly different. Although the syntax and basic operation of formal verification tools might be easy to learn, there is usually considerable subtlety in using the tools effectively on large problems. Expert users with a deep understanding of a tool are able to achieve impressive results; novice users are all too frequently frustrated by exploding memory and CPU time usage.

## 1.1 Motivation

A key difference between novice and expert is that the expert has a detailed understanding of the algorithms and heuristics used by a tool and can use that understanding to choose an efficient way to present the verification problem to the tool.

When applying formal verification, many choices must be made: how to model parts of the system, which parts of a system to abstract, how to perform that abstraction, and so forth. Differing choices can result in an equivalent verification problem, but some choices will be more or less efficient than others for the particular verification tool. When the novice has the chance to consult with the expert, advice often takes the form, "Don't model it like that; do this instead. Otherwise, ...," where the consequence reflects detailed knowledge of the tool: the BDD will have to represent all permutations, the state space will lose symmetry, the reduction heuristic will not work, and so forth. This situation is clearly undesirable; the novice is forced to become an expert on and adapt to the idiosyncrasies of the tool. Even for the expert, being forced to adapt to the tool can be suboptimal, since the description style that most clearly matches the verification problem might not be the style that best suits the verification tool. Also, over time the algorithms used by the tool may change, forcing the expert to rewrite models in order to take advantage of improvements to the tool.

Rather than forcing the user to adapt to the tool, the tool should adapt to the user. A robust, practical verification tool should help close the gap between novice and expert, allowing the novice to get useful results without a lengthy learning curve. For novice and expert alike, the tool should free the user to match the description to the problem being verified, rather than to specific quirks of the verification tool.

We hypothesize that often a novice will model a system differently (less efficiently for the tool) than an expert would, that many of these inefficient modeling choices can be easily detected in the source code of the model description, and that the tool should be able to optimize away these differences.

## 1.2 Project

To test our hypothesis, we consider a small set of source-level transformations with a specific tool. We have chosen the Mur$\varphi$ verification system [6, 5] as the target for our test. The choice of Mur$\varphi$ is fairly arbitrary — our ideas should apply in general, although specific optimizations, obviously, apply only to verification tools with similar features. We will give a brief overview of Mur$\varphi$ in Chapter 2, followed by a sampling of related work.

In Chapter 3, we will propose three possible Mur$\varphi$ source-level transformations which may improve models written by novice users. The first, ruleset rearrangement, eliminates redundant rule firings. These extra rule firings can occur when users group rules without considering the implementation details of how Mur$\varphi$ evaluates rules. The second, scalarset identification, is an automatic means of finding a particular type of symmetry. Exploiting symmetry can significantly reduce both time and space requirements, but novices may not understand how symmetry works or the importance of exploiting it. The last transformation, variable clearing, evaluates data dependencies to find when variables are no longer relevant. Failing to clear variables when they become irrelevant causes creation of redundant states, increasing time and space requirements. As with the other transformations, novices are unlikely to grasp the impact of variable clearing on the verification of their model.

We have implemented the first proposed transformation, ruleset rearrangement, as a modification of Mur$\varphi$ 3.1. We chose not to implement scalarset identification and automatic variable clearing, because of structural problems in the Mur$\varphi$ compiler. In Chapter 4 we will describe our implementation and the details of those problems.

To test the effectiveness of our transformation, we apply it to three sets of examples, including models written by both experts and novices. The results of our experiments, which we will present in Chapter 5, support our hypothesis. Novice users do model problems differently, and some styles are less efficient for the tool than others. In many cases, the application of source-level transformations corrects those inefficiencies.

## 1.3 Contribution

The primary contribution of our work is the identification of a new research area. To date, work on optimizing verification has focused on improving the fundamental algorithms. We assert that optimizations which close the gap between novice and expert are also important, and support our assertion with positive experimental results. Furthermore, we identify several source-level transformations which can help to close this gap, and implement one. In the process of evaluating that transformation, we have developed necessary infrastructure for performing optimizations in Mur$\varphi$ 3.1 and identified impediments to implementation of further source-level optimizations. Finally, we provide advice for follow-on research using the Mur$\varphi$ platform.

# Chapter 2

# Background

In this chapter we provide background on the Mur$\varphi$ verifier, with details on the features which are relevant to our work, and an overview of related work.

## 2.1 Mur$\varphi$

We have chosen the Mur$\varphi$ verification system [6, 5] as the basis for our optimizations. The Mur$\varphi$ verifier is a model checker which performs explicit state space reachability.[1] Mur$\varphi$ has been widely used for a variety of applications (e.g., [7, 25, 18, 10, 19, 11]).

The Mur$\varphi$ verifier has it's own input language for describing the system being verified. This Mur$\varphi$ language is a high-level, guarded-command language [4, 3]. A Mur$\varphi$ program consists of two main parts: declarations and rules. The declaration section is Pascal-like and declares constants, types, global variables, and procedures. The rules define the transitions of the system. At any given time, the system state is determined by the values of the global variables. As the system executes, a rule is

[1] For additional information see `http://verify.stanford.edu/dill/murphi.html`.

chosen nondeterministically and executes atomically, updating the global variables to a new state.

Compiling a Mur$\varphi$ model is a two step process. First Mur$\varphi$ transforms the model into a C++ program which verifies the model. Then a C++ compiler is used to turn that program into an executable. Most elements of a Mur$\varphi$ model translate well into C++, with the exception of rule execution. Since Mur$\varphi$ is verifying the model for all possible executions, all rules are attempted for each visited state.

Mur$\varphi$ tracks visited states in a table, which can become very large. Mur$\varphi$ offers two options, bit compaction and hash compaction, to reduce the memory requirements of the state table. These options trade increased computation time for reduced space. Bit compaction packs states rather than word-aligning them, which may have a dramatic impact on the memory requirements of some models. Hash compaction does not store full state information in the state table, but rather stores a hash of the state. At the end of a run Mur$\varphi$ gives the probability of a hash collision on that run. The hash function is different for each run of a model, allowing multiple runs to be used to reduce the overall probability that states have been missed. Bit compaction and hash compaction may be combined.

Mur$\varphi$ provides several features to simplify writing scalable descriptions of large systems. The normal description style uses numerous subrange types, which can be scaled easily. Of particular interest to this thesis are the special `scalarset` and `multiset` types, which provide automatic symmetry reduction [15]. Scalarsets are like subranges, but without order. Multisets are like arrays, except that the array elements are unordered. Appropriate use of these data types greatly reduces the size of the state space. To simplify writing rules for all values of a subrange or scalarset, Mur$\varphi$ provides a `ruleset` construct, which generates a copy of all enclosed

```
Const
  ProcCount: 2;     -- # of processors
  AddressCount: 2;  -- # of addresses
  ValueCount: 2;    -- # of distinct values
  ...
Type
  Pid: Scalarset(ProcCount);
  Address: Scalarset(AddressCount);
  Value: Scalarset(ValueCount);
  ...
  ProcState: Record
               ...
               cache: Array[Address] of
                        Record
                          ...
                          v: Value;
                        End;
             End;
  ...
Var
  procs: Array[Pid] of ProcState;
  ...
Procedure flushCache(p: Pid);
Begin
  ...
End;
...
```

*(Figure 2.1, continued on next page)*

*(continuation of Figure 2.1)*

```
Ruleset p: Pid Do
  Alias me: procs[p] Do
    Ruleset a: Address Do
      Ruleset v: Value Do
        Rule "Evict shared data"
          (me.cache[a].state = Shared) ==>
            me.cache[a].state := Invalid;
            Undefine me.cache[a].v;
        Endrule;
        Rule "Change exclusive copy"
          (me.cache[a].state = Exclusive) ==>
            me.cache[a].v := v;
        Endrule;
        ...
        Rule "Processor reset"
          true ==>
            flushCache(p);
        Endrule;
      Endruleset;
    Endruleset;
  Endalias;
Endruleset;
...
```

Figure 2.1: Fragments from a Mur$\varphi$ Model of a Cache Coherence Protocol. The syntax is similar to Pascal, extended to support specifying rules as guarded commands. There are two main sections, for declarations (e.g., `Const`, `Type`, `Var` and `Procedure`) and for rules (e.g., `Ruleset` and `Alias`). The scalarset definitions are like subranges, except that they permit automatic symmetry reduction. In this example, the rulesets instantiate the rules for all possible processors, addresses, and values. The rules shown allow any processor to evict any address that it has cached in the shared state, any processor to change to any value any address that it has cached exclusive, and any processor to flush its cache.

```
Ruleset p: Pid Do
  Alias me: procs[p] Do
    Ruleset a: Address Do
      Ruleset v: Value Do
        Rule "Evict shared data"
          (me.cache[a].state = Shared) ==>
            me.cache[a].state := Invalid;
            Undefine me.cache[a].v;
        Endrule;
      Endruleset;
    Endruleset;
  Endalias;
Endruleset;

Ruleset p: Pid Do
  Alias me: procs[p] Do
    Ruleset a: Address Do
      Ruleset v: Value Do
        Rule "Change exclusive copy"
          (me.cache[a].state = Exclusive) ==>
            me.cache[a].v := v;
        Endrule;
      Endruleset;
    Endruleset;
  Endalias;
Endruleset;
```

Figure 2.2: An example of ungrouping two rules from Figure 2.1.

rules for each possible value of its formal parameter. Similarly, the `choose` construct selects an item from a multiset. Figure 2.1 shows some portions of a Mur$\varphi$ model for a cache coherence protocol.

Mur$\varphi$ allows users to group rules within ruleset, choose, and alias statements, which can be nested arbitrarily. Grouping rules together under ruleset, choose or alias statements is logically equivalent[2] to placing each rule under separate identical

[2]In practice each ruleset, choose, and alias statement incurs a small amount of overhead, so grouping rules can improve performance very slightly.

ruleset, choose or alias statements, as in Figure 2.2.

## 2.2 Related Work

The general concept of rewriting something to improve downstream performance is widespread, e.g., database query optimization or high-performance numerical code preprocessing, but this concept has not been applied to formal verification.

Optimizing compilers perform source-level transformations, such as loop unrolling, loop distribution, strength reduction, and induction variable removal [1, 20], some of which are similar to the Mur$\varphi$ transformations we propose. Most programming languages are flexible, allowing the same idea to be expressed in many different ways. This flexibility empowers the user to express themselves in whatever way best suits their problem, rather than the way which best suits the compiler. One of the main goals of compiler optimization is to avoid penalizing users who do not express themselves in the way which directly translates to optimum performance. Mur$\varphi$ takes advantage of compiler optimizations by generating C++ code which is then processed by an optimizing C++ compiler. The primary difference with our work is that compiler optimizations seek to optimize code execution speed, while we seek to improve the verification code which is to be compiled.

Within the formal verification community, a great deal of research has gone into reducing memory usage and runtime (for examples, see some recent surveys [2, 8, 16, 9, 17]). These optimizations are written by experts for experts, optimizing the fundamental verification algorithms in ways that rarely can be expressed in the original language. Obviously, this research is complementary to ours; they optimize models written in a particular style, while we transform models written in a variety of styles into the particular styles which subsequently optimize well. The work most closely related to ours is by Ip [14]. As with our work, he performs high-level analysis of Mur$\varphi$ programs during compilation to accelerate the verification, although again, his optimizations cannot be expressed in the source language, and therefore are not closing the gap between expert and novice.

We have recently become aware of work by Yorav and Grumberg [26], which analyzes the control-flow graph of a Mur$\varphi$ program to reduce the size of the statespace. They attempted two optimizations. One, which they call dead variable reduction, reduces the statespace size by eliminating variables which are no longer used for computation. The optimization is very similar to our variable clearing transformation, described in Section 3.3. The other, called path reduction, shrinks the statespace by shortening computation paths, reducing the number of steps for each computation.

# Chapter 3

# Source-Level Transformations

In this chapter, we identify a few possible source-code transformations which may improve models written by novice users. These are ruleset rearrangement, scalarset identification, and variable clearing.

## 3.1 Ruleset Rearrangement

Mur$\varphi$ allows users to group rules within `ruleset`, `choose`, and `alias` statements. These groupings are primarily for convenience; grouping of rules may be the best match for the problem semantics or may make it easier for the user to understand the model.

If a rule is enclosed within a ruleset or choose statement on which it does not depend, the verifier will needlessly execute the rule for each possible value of the enclosing ruleset or choose parameter. The set of reachable states is unaffected, as each firing of that rule for variations of the independent variable will lead to the same state, but every extra rule firing adds to the runtime of the model.

Obviously, rules should be moved outside the scope of irrelevant ruleset and choose statements. Grouping of rules complicates removal. We will rely on two properties to rewrite the description into an equivalent one that eliminates the needless rule firings:

**Notation** *Let a* SimpleRule *be any single* `rule` *statement. Let a* Ruleset *be any single* `ruleset`*,* `choose`*, or* `alias` *expression. Let a* CompoundRule *be any* SimpleRule*, any* Ruleset *enclosing a* CompoundRule*, or any sequence of* CompoundRules*. If $\phi$ is a* CompoundRule*, let $\langle\phi\rangle$ denote the* CompoundRule *formed by enclosing $\phi$ in a* Ruleset*.*

**Property 1** *A* CompoundRule *$\phi$ is equivalent to $\langle\phi\rangle$ if $\phi$ does not depend on the enclosing* Ruleset*.*

**Property 2** *Any* Ruleset *enclosing a sequence of* CompoundRules *$\langle\phi_1, \phi_2, \ldots, \phi_n\rangle$ is equivalent to $\langle\phi_1\rangle, \langle\phi_2\rangle, \ldots, \langle\phi_n\rangle$.*

A straightforward implemention method starts by ungrouping all rules, so that each rule is alone within its enclosing ruleset, choose, and alias statements. Then, a check is performed to determine whether each rule is dependent on those enclosing statements. (Our ungrouping is similar to loop distribution [23], and our dependency check is similar to induction variable removal [24].) Independent ruleset and choose statements are removed, resulting in an equivalent model which executes faster. Independent alias removal does not directly impact performance, but it greatly simplifies the dependency analysis of parent ruleset and choose statements, so independent alias statements are also removed. For example, in Figure 2.1 the eviction rule does not depend on the ruleset over $v$, and the reset rule does not depend on $v$ or $a$. The transformation would convert the rules as shown in Figure 3.1. Each rule is now enclosed by only the relevant rulesets.

```
Ruleset p: Pid Do
  Alias me: procs[p] Do
    Ruleset a: Address Do
      Rule "Evict shared data"
        (me.cache[a].state = Shared) ==>
          me.cache[a].state := Invalid;
          Undefine me.cache[a].v;
      Endrule;
    Endruleset;
  Endalias;
Endruleset;

Ruleset p: Pid Do
  Alias me: procs[p] Do
    Ruleset a: Address Do
      Ruleset v: Value Do
        Rule "Change exclusive copy"
          (me.cache[a].state = Exclusive) ==>
            me.cache[a].v := v;
        Endrule;
      Endruleset;
    Endruleset;
  Endalias;
Endruleset;

Ruleset p: Pid Do
  Rule "Processor reset"
    true ==>
      flushCache(p);
  Endrule;
Endruleset;
```

Figure 3.1: The result of applying the ruleset rearrangement transformation to the example in Figure 2.1. Note that `"Evict shared data"` and `"Processor reset"` have fewer enclosing rulesets than in Figure 2.2, reducing the number of possible states which must be considered.

The idea of rearranging rulesets is not new, although ours is the first implementation which produces optimal results in all cases. Mur$\varphi$ 3.1 employs an ad hoc heuristic which also attempts to skip some redundant iterations over independent variables. The heuristic shuffles a hierarchy of ruleset and choose statements, leaving rule groupings intact. The rule groupings obscure the dependency structure and defeat most optimizations.

## 3.2 Scalarset Identification

Using scalarsets instead of subranges in cases where ordering within the range does not matter provides a tremendous reduction in state space size. Subranges, however, cannot always be replaced by scalarsets: any operation that relies on order, arithmetic, or distinguishing special elements in the range breaks symmetry and precludes the use of scalarsets.

Novices might not realize all cases when a scalarset can be used. A trivial possible transformation is to check for each subrange type whether it could be redefined as a scalarset.

A more interesting and powerful transformation is possible if one considers common programming practice. Data types are often reused for various purposes if it seems appropriate. An integer type, for example, might be used for a counter, for arithmetic, and for ID numbers. Returning to the example in Figure 2.1, suppose the user had defined an "integer" type as a subrange, and used this type for addresses, processor IDs, and values, as well as for some counters. In that model, one could not apply the scalarset symmetry reduction to addresses, processor IDs, and values, because the same type is also being used for counters, which have order. Even without the counters, having a single scalarset type for addresses, processor IDs, and values would not allow as much symmetry reduction as having a separate scalarset for each.

The data type reuse problem suggests a much more powerful solution to identify possible scalarsets. A syntactic check of the Mur$\varphi$ program can determine which variables interact and must have the same type versus which variables have the same type simply for the user's convenience. A graph can be constructed with a node for each variable and edges indicating which variables must be type-compatible. Each connected component of the graph may be assigned a distinct type. Each of these types may then be checked separately for possible conversion to a scalarset.

## 3.3 Variable Clearing

In most models, not all variables are holding important data at all times. For example, if a cache line is invalid, its contents do not matter, or if a queue is modeled as an array and a tail pointer, the array elements past the tail pointer do not matter.

Although the contents of these variables may not matter to the user or to the accuracy of the model, they do matter to the verification tool. Two states that differ only in the values of don't-matter variables are still considered two distinct states by the verification tool. If the user is not careful to clear any leftover values out of variables whenever the the value no longer matters, the result is a needless explosion in the number of states. For example, in Figure 2.1, if the user omitted the `Undefine` statement, the resulting model would have a much larger set of reachable states, because different values leftover in the `me.cache[a].v` field would generate additional states.

Tracking exactly which variables are no longer needed and making sure to clear out their values is tedious for the expert user and extremely difficult for the novice, who may not even realize the importance of doing so. Fortunately, live variables analysis, a standard program analysis technique (e.g., [22]), can determine automatically which variables are dead (not needed) at each point in a program. Note that the Mur$\varphi$ execution model, in which rules are chosen nondeterministically, results in a very conservative live variables analysis. Some limited data flow analysis to track rule enabling and disabling might strengthen the analysis considerably. Adapting live variables analysis to Mur$\varphi$ programs could determine where `Undefine` statements are needed. The source-code transformation would insert the `Undefine` statements automatically, greatly improving the efficiency of verification. Recent work by Yorav and Grumberg [26] confirms that this is a promising area of research.

# Chapter 4

# Implementation

This chapter discusses our implementation work. We begin with an overview of the relevant portions of Mur$\varphi$, followed by a detailed description of our optimizations. Finally, we will give advice for future optimization projects based on Mur$\varphi$.

Since it was unlikely that our work would become part of the main Mur$\varphi$ distribution, we wished to avoid major changes to any specific version of the compiler. Mur$\varphi$'s original design called for separate parsing and compilation stages, to allow additional processing stages, such as an optimizer, to be easily inserted between parsing and code generation. Our intent was to write routines which analyzed and then modified the parse tree, and to insert those routines between the parsing and code generation stages, saving us from having to develop our own code for parsing and output. Mur$\varphi$ is a big language, and a full Mur$\varphi$ parser would be complex and time consuming to produce. Unfortunately, the implementation of the current Mur$\varphi$ compiler does not permit insertion of a meaningful optimization stage between the parser and the code generator.

## 4.1 Mur$\varphi$ 3.1

In this section we discuss the relevant details of the implementation of Mur$\varphi$ 3.1, the current version of the Mur$\varphi$ compiler. There are numerous little details in Mur$\varphi$ 3.1 that make implementing an optimization stage difficult, which are to be expected when modifying a mature program. The little details did not present significant obstacles. Unfortunately, the underlying structure of the Mur$\varphi$ 3.1 compiler was an obstacle, and greatly hindered our efforts.

Although at a high level it appears that Mur$\varphi$ 3.1 still has separate parsing and compilation stages, a more detailed analysis of the source code shows that many aspects of the implementation violate the original object-oriented design. One aspect which proved nearly impossible to work around is that the parse tree is completely ignored after it is created. Code generation is performed using an independent set of data structures, which are generated at parse time along with the parse tree. Those data structures represent a partial compilation of the program. Our source-level transformations cannot be performed on the partially compiled program.

Fortunately, the rule code generation data structure is simple enough that we can regenerate it after performing modifications on our parse tree. This allows us to perform ruleset rearrangement, but none of our other proposed transformations.

## 4.2 Mur$\varphi$ Optimizer

The ruleset rearrangement transformation is implemented as a modification of Mur$\varphi$ 3.1. We added a small amount of helper code in one class, made necessary adjustments to protections on another class, added the appropriate calls to our code between parsing and code generation, and cleaned up a few problems with

ANSI C++ compliance. Appendix A.3.1 contains a full listing of the changes to Mur$\varphi$ 3.1, in unified diff format.

Our optimizer[1] works in two stages. The first stage traverses Mur$\varphi$'s parse tree, building a new version of that tree. The second stage traverses the new tree, performing any possible optimizations.

The new parse tree is built to overcome deficiencies in the object-orientation of Mur$\varphi$ 3.1. All classes are derived from the same base class, and each class has consistent methods for dependency analysis and optimization. In proper object-oriented fashion, each class knows how to optimize itself. A typical `optimize()` method asks each child of the class to optimize, requests dependency information from those children, and then, based on its own state and the dependencies of its children, will perform transformations on itself. In some cases these transformations require creation of a replacement object based on a different class; such replacement objects are passed to the parent as the return value of `optimize()`.

The ruleset rearrangement transformation requires dependency information from all children of the rule to know whether or not the rule is independent. For the most part, dependency analysis in a Mur$\varphi$ model should be easy. All of a model's state is contained in global variables, and most operations on those global variables are performed in rules, which are easy to analyze. The exception is when rules call functions or procedures. The simplest approach is to assume that all variables passed to a function or procedure are used in that function or procedure, but that may not always be the case. A more thorough approach would be to apply standard interprocedural data-flow analysis [21] to this problem.

Unfortunately, it is very difficult to track variable dependencies in Mur$\varphi$ 3.1.

[1] Full source code for our optimizer may be found in Appendix A, or may be downloaded from `http://www.cs.ubc.ca/labs/isd/Projects/`.

This is because Mur$\varphi$ 3.1 takes advantage of the subsequent C++ compilation step, maintaining as little information as possible about the details of the variable. In the absence of better information, our optimizer currently tracks dependencies by scope.[2] Tracking dependencies by scope is sufficient for simple ruleset rearrangement, but is inadequate for tracing variable use through procedure and function calls.

Ideally our optimizer should not be as large as it is. The extra size comes from recreating many of the structures present in the Mur$\varphi$ compiler. This redundancy is unfortunate, but was necessary given our design goals.

## 4.3 Suggestions for Future Attempts

Our current Mur$\varphi$ optimizer cannot be extended further without a major overhaul of Mur$\varphi$ 3.1. We believe that to be successful, any future work on source-level Mur$\varphi$ optimization should adopt one of two different approaches.

One approach is to create a source-to-source optimizer. All of our proposed transformations are source-level, so an optimizer could ignore code generation entirely. We believe that because Mur$\varphi$ is a flexible language, many other desirable optimizations could also be performed at the source level. This approach requires writing a Mur$\varphi$ parser, but has the advantage that the program output is a trivial dump of the optimized parse tree.

The other approach is to rewrite Mur$\varphi$. Such a rewrite should be based on completely separate parse, optimization, and compilation stages. The class structure of our Mur$\varphi$ optimizer could be used as a basis for such a rewrite, but parse and compilation methods would need to be created, which is a nontrivial task. Without

[2] Each `begin/end` block in a Mur$\varphi$ program constitutes a different scope.

borrowing substantial portions of Mur$\varphi$ 3.1 compilation code, this approach also complicates any assertion that the resulting optimized model is equivalent to an unoptimized model produced through Mur$\varphi$ 3.1.

# Chapter 5

# Experiments

## 5.1 Methodology

Testing was performed on three sets of Mur$\varphi$ programs. The first set is the examples included in the Mur$\varphi$ distribution. These examples are written by expert Mur$\varphi$ users, so our hypothesis predicts little improvement from our optimization. The second set are eleven implementations of a simple, fictitious directory-based cache coherence protocol, each developed independently by students in a formal verification class. The students had no previous experience with Mur$\varphi$. They were given a tabular description of the cache coherence protocol and a partial Mur$\varphi$ model that included declarations, but no rules, so they were free to write the rules in whatever manner was most natural for them. This set of programs represents a large number of novice users independently tackling the same verification task. Our hypothesis predicts that some of these should benefit significantly from our optimization. The third experiment is a model of part of the physical layer of the IEEE 1394 [13] High Performance Serial Bus. The model implements the reset and tree identification portions of the physical layer. The model is significantly larger than the others and was written by the author when he was a Mur$\varphi$ novice. Large, real models written by novices are not widely available, but such models provide the best test of our hypothesis because large real models written by novices are precisely those that we seek to improve.

The Mur$\varphi$ generated C++ code was compiled using egcs version 2.91.66 with "`-O4 -mpentiumpro`" optimization. The Mur$\varphi$ hash table size was set to 96 MB with 30% for the active states. Experiments were run with and without hash compaction. With hash compaction, the default 40-bit hash size was used. The experiments were run on a 400 MHz Pentium II workstation with 128 MB of RAM, running SuSE Linux 6.1.

## 5.2 Results

The results of the experiments support our hypothesis. The size of the model (lines of code and number of states reached), runtime in seconds with and without optimization, and percent improvement are reported for all models. The runtime reported is the median of five runs, in order to eliminate possible outlier effects.[1]

The results for the set of examples distributed with the Mur$\varphi$ verifier are shown in Tables 5.1 and 5.2. As can be seen, the transformation had little effect on these programs, written by expert Mur$\varphi$ users. This result is as expected by our hypothesis — the goal of our optimization is to improve the source code written by novice users, rather than trying to improve the fundamental verification algorithms. Worth noting, however, is that even this extremely simple transformation was able

[1]Few outliers were observed, with most models showing less than 2% variation, fastest to slowest. Some of the smallest models ($< 1s$ running time) had variations as high as 20%, probably due to measurement error and operating system issues.

| Model | Lines | States | Time$_{orig}$ | Time$_{opt}$ | % Imprv |
|---|---|---|---|---|---|
| 2_peterson | 135 | 13 | 2.51 | 2.51 | 0.00 |
| abp | 176 | 80 | 2.51 | 2.50 | 0.40 |
| adash | 2809 | 10466 | 20.99 | 21.18 | -0.91 |
| adashbug | 2809 | 3742 | 6.66 | 6.69 | -0.45 |
| arbiter | 114 | 502 | 1.91 | 1.91 | 0.00 |
| cache3 | 1184 | 31433 | 9.73 | 9.78 | -0.51 |
| dek | 135 | 100 | 2.50 | 2.50 | 0.00 |
| down | 87 | 10957 | 2.66 | 2.67 | -0.38 |
| dp4 | 151 | 112 | 2.49 | 2.49 | 0.00 |
| dpnew | 236 | 121 | 2.51 | 2.51 | 0.00 |
| eadash | 1880 | 133491 | 1453.73 | 1470.83 | -1.18 |
| ldash | 1324 | 254986 | 1254.72 | 1162.85 | 7.32 |
| lin | 62 | 101 | 2.51 | 2.51 | 0.00 |
| list6 | 333 | 23410 | 4.49 | 4.30 | 4.23 |
| list6too | 413 | 1077 | 4.57 | 4.59 | -0.44 |
| mcslock1 | 333 | 23644 | 5.63 | 5.63 | 0.00 |
| mcslock2 | 390 | 540219 | 57.82 | 57.99 | -0.29 |
| n_peterson | 203 | 163298 | 50.95 | 50.95 | 0.00 |
| newcache3 | 1029 | 34781 | 53.11 | 53.20 | -0.17 |
| newlist6 | 309 | 13044 | 6.06 | 6.06 | 0.00 |
| ns | 449 | 467 | 1.25 | 1.28 | -2.40 |
| ns-old | 449 | 17 | 1.23 | 1.23 | 0.00 |
| pingpong | 79 | 4 | 2.50 | 2.51 | -0.40 |
| sci | 4875 | 18193 | 13.58 | 13.06 | 3.83 |
| scierr | 4875 | 64 | 1.24 | 1.24 | 0.00 |
| sets | 137 | 29 | 1.91 | 1.91 | 0.00 |
| sort5 | 101 | 309 | 2.50 | 2.50 | 0.00 |

Table 5.1: Example programs distributed with the Mur$\varphi$ verifier, without hash compaction. Model name, number of lines of source code, number of reachable states, and unoptimized and optimized verification times in seconds are reported. Examples distributed with the Mur$\varphi$ verifier were written by experts. As expected by our hypothesis, they are mostly unaffected by our optimization.

| Model | Lines | States | Time$_{orig}$ | Time$_{opt}$ | % Imprv |
|---|---|---|---|---|---|
| 2_peterson | 135 | 13 | 0.83 | 0.83 | 0.00 |
| abp | 176 | 80 | 0.84 | 0.84 | 0.00 |
| adash | 2809 | 10466 | 21.13 | 21.45 | -1.51 |
| adashbug | 2809 | 3742 | 6.79 | 6.81 | -0.29 |
| arbiter | 114 | 502 | 0.83 | 0.84 | -1.20 |
| cache3 | 1184 | 31433 | 10.22 | 10.26 | -0.39 |
| dek | 135 | 100 | 0.82 | 0.82 | 0.00 |
| down | 87 | 10957 | 1.05 | 1.05 | 0.00 |
| dp4 | 151 | 112 | 0.81 | 0.81 | 0.00 |
| dpnew | 236 | 121 | 0.84 | 0.84 | 0.00 |
| eadash | 1880 | 133491 | 1480.52 | 1479.43 | 0.07 |
| ldash | 1324 | 254986 | 1286.42 | 1179.33 | 8.32 |
| lin | 62 | 101 | 0.77 | 0.78 | -1.30 |
| list6 | 333 | 23410 | 4.60 | 4.56 | 0.87 |
| list6too | 413 | 1077 | 4.09 | 4.09 | 0.00 |
| mcslock1 | 333 | 23644 | 5.28 | 5.28 | 0.00 |
| mcslock2 | 390 | 540219 | 65.13 | 65.08 | 0.08 |
| n_peterson | 203 | 163298 | 53.48 | 53.43 | 0.09 |
| newcache3 | 1029 | 34781 | 54.56 | 54.79 | -0.42 |
| newlist6 | 309 | 13044 | 5.94 | 5.93 | 0.17 |
| ns | 449 | 467 | 0.88 | 0.90 | -2.27 |
| ns-old | 449 | 17 | 0.84 | 0.84 | 0.00 |
| pingpong | 79 | 4 | 0.83 | 0.84 | -1.20 |
| sci | 4875 | 18193 | 13.54 | 13.71 | -1.26 |
| scierr | 4875 | 64 | 0.82 | 0.82 | 0.00 |
| sets | 137 | 29 | 0.84 | 0.84 | 0.00 |
| sort5 | 101 | 309 | 0.82 | 0.83 | -1.22 |

Table 5.2: Example programs distributed with the Mur$\varphi$ verifier, with hash compaction. Examples distributed with the Mur$\varphi$ verifier were written by experts. Again, these are mostly unaffected by our optimization.

| Model | Lines | States | $\text{Time}_{orig}$ | $\text{Time}_{opt}$ | % Imprv |
|---|---|---|---|---|---|
| student1 | 587 | 1828 | 2.88 | 2.88 | 0.00 |
| student2 | 641 | 36984 | 45.31 | 31.50 | 30.48 |
| student3* | 653 | 8204 | 4.38 | 4.39 | -0.23 |
| student4 | 710 | 501446 | 397.91 | 399.20 | -0.32 |
| student5 | 496 | 3522 | 3.17 | 3.17 | 0.00 |
| student6 | 642 | 36995 | 44.21 | 30.94 | 30.02 |
| student7 | 947 | 3647 | 3.52 | 3.48 | 1.14 |
| student8 | 669 | 9071 | 6.87 | 6.83 | 0.58 |
| student9* | 714 | 52771 | 54.62 | 54.35 | 0.49 |
| student10 | 568 | 4403 | 3.97 | 4.00 | -0.76 |
| student11 | 538 | 1828 | 2.22 | 2.22 | 0.00 |

Table 5.3: Cache coherence protocol models, without hash compaction. On models written by novices, we expect that some will happen to be written in a manner that is not ideal for Mur$\varphi$. Those examples show significant improvement. All models were run with 2 processors, 3 addresses, 2 data values, and communication channels scaled as necessary, except those marked with an asterisk, which blew up with 3 addresses, so they were run with only 2 addresses.

| Model | Lines | States | $\text{Time}_{orig}$ | $\text{Time}_{opt}$ | % Imprv |
|---|---|---|---|---|---|
| student1 | 587 | 1828 | 2.58 | 2.59 | -0.39 |
| student2 | 641 | 36984 | 46.28 | 32.48 | 29.82 |
| student3* | 653 | 8204 | 4.14 | 4.18 | -0.97 |
| student4 | 710 | 501446 | 417.02 | 416.93 | 0.02 |
| student5 | 496 | 3522 | 2.92 | 2.92 | 0.00 |
| student6 | 642 | 36995 | 44.96 | 31.94 | 28.96 |
| student7 | 947 | 3647 | 3.32 | 3.23 | 2.71 |
| student8 | 669 | 9071 | 6.80 | 6.78 | 0.29 |
| student9* | 714 | 52771 | 55.87 | 56.18 | -0.55 |
| student10 | 568 | 4403 | 3.81 | 3.82 | -0.26 |
| student11 | 538 | 1828 | 1.95 | 1.96 | -0.51 |

Table 5.4: Cache coherence protocol models, with hash compaction. Again, some examples written by novices show significant improvement. These models were run with the same parameters as in Table 5.3.

to give noticeable speedup on one model written by a Mur$\varphi$ expert.

Tables 5.3 and 5.4 show the results for the set of cache coherence protocol models, each developed independently by a different novice user. These models were tested with 2 processors, 3 addresses, and 2 data values. Communication channels were scaled to the minimum value which would allow verification of the model to complete. In this case, our hypothesis predicts that some of these programs should show significant improvement, corresponding to when a particular user chose a writing style that happens not to be well-suited to the internals of the tool. As expected, our simple optimization significantly speeds up two of the eleven programs. The enormous variations in code size, state count, and run times make it clear that different users naturally express themselves differently. A robust formal verification tool should adapt to this diversity, rather than forcing users to write code specifically tailored to the verification tool's idiosyncrasies.

Results for the third test are shown in Tables 5.5 and 5.6. This program[2] is a larger model of a real protocol, part of the IEEE 1394 "Firewire" standard. The program was written by a novice Mur$\varphi$ user for whom the most natural way to model the system did not give the best arrangement of rules and rulesets for efficient Mur$\varphi$ execution. The transformation significantly accelerated this example. It is again clear that novice users do not understand the subtleties of the verification tool, so their models might not be written in the most efficient manner for the tool. A simple source-code transformation helps rectify the problem.

Compile times (Mur$\varphi$ translation plus C++ compilation) for each model are shown in Tables 5.7, 5.8, 5.9, 5.10, 5.11, and 5.12. Times from only one compilation are reported, because the transformation does not significantly affect compile time.

[2] A listing of the model may be found in Appendix B.1.

In absolute terms, the worst compile time increase was 8.5 seconds on an example that required 255 seconds total. In percentage terms, the worst compile time increase was 18% on an example that required 10.2 seconds total.

| Model | Lines | States | $\text{Time}_{orig}$ | $\text{Time}_{opt}$ | % Imprv |
|---|---|---|---|---|---|
| ieee-1394 | 824 | 2060216 | 372.23 | 271.43 | 27.08 |

Table 5.5: Partial model of the cable physical layer of the IEEE 1394 "Firewire" protocol, without hash compaction. Here we see results on a large example of a real system. The example was written by a novice Mur$\varphi$ user and happened to be written in a style that the user found natural, but was not ideal for Mur$\varphi$. Our optimization improved performance considerably.

| Model | Lines | States | $\text{Time}_{orig}$ | $\text{Time}_{opt}$ | % Imprv |
|---|---|---|---|---|---|
| ieee-1394 | 824 | 2060216 | 372.77 | 278.97 | 25.16 |

Table 5.6: Partial model of the cable physical layer of IEEE 1394, with hash compaction. Again, our optimization improved performance considerably.

| | No Optimization | | With Optimization | |
|---|---|---|---|---|
| Model | Mur$\varphi$ time | C++ time | Mur$\varphi$ time | C++ time |
| 2_peterson | 0.04 | 12.22 | 0.05 | 12.36 |
| abp | 0.05 | 9.35 | 0.06 | 9.35 |
| adash | 0.17 | 112.63 | 3.27 | 112.71 |
| adashbug | 0.16 | 112.81 | 3.27 | 112.90 |
| arbiter | 0.04 | 10.38 | 0.05 | 10.59 |
| cach3multi | 0.02 | 0.02 | 0.02 | 0.02 |
| cache3 | 0.08 | 53.07 | 0.51 | 53.02 |
| dek | 0.04 | 9.53 | 0.06 | 9.62 |
| down | 0.03 | 6.91 | 0.05 | 6.76 |
| dp4 | 0.04 | 9.31 | 0.06 | 8.95 |
| dpnew | 0.04 | 11.04 | 0.07 | 11.05 |
| eadash | 0.13 | 95.37 | 1.55 | 95.65 |
| ldash | 0.09 | 75.90 | 1.21 | 75.57 |
| lin | 0.05 | 6.04 | 0.03 | 5.80 |
| list6 | 0.06 | 20.48 | 0.12 | 20.30 |
| list6too | 0.08 | 29.91 | 0.28 | 29.74 |
| mcslock1 | 0.06 | 19.36 | 0.07 | 19.36 |
| mcslock2 | 0.06 | 27.58 | 0.09 | 27.53 |
| n_peterson | 0.04 | 15.37 | 0.05 | 15.42 |
| newcache3 | 0.09 | 88.43 | 0.50 | 87.68 |
| newlist6 | 0.06 | 37.11 | 0.08 | 37.12 |
| ns | 0.07 | 87.05 | 0.18 | 86.31 |
| ns-old | 0.06 | 86.51 | 0.18 | 86.28 |
| pingpong | 0.04 | 7.95 | 0.05 | 7.71 |
| sci | 0.57 | 244.83 | 7.18 | 246.64 |
| scierr | 0.78 | 241.44 | 7.72 | 240.82 |
| sets | 0.17 | 8.49 | 0.72 | 9.50 |
| sort5 | 0.05 | 6.86 | 0.04 | 6.79 |

Table 5.7: Compilation times (in seconds) for example programs distributed with the Mur$\varphi$ verifier, without hash compaction.

| | No Optimization | | With Optimization | |
|---|---|---|---|---|
| Model | Mur$\varphi$ time | C++ time | Mur$\varphi$ time | C++ time |
| 2_peterson | 0.05 | 13.26 | 0.05 | 13.25 |
| abp | 0.05 | 10.35 | 0.05 | 10.33 |
| adash | 0.17 | 114.18 | 3.28 | 113.84 |
| adashbug | 0.16 | 113.99 | 3.28 | 113.82 |
| arbiter | 0.04 | 11.35 | 0.05 | 11.38 |
| cach3multi | 0.02 | 0.02 | 0.02 | 0.02 |
| cache3 | 0.09 | 54.27 | 0.51 | 54.14 |
| dek | 0.04 | 10.58 | 0.05 | 10.55 |
| down | 0.03 | 7.96 | 0.05 | 7.81 |
| dp4 | 0.04 | 10.03 | 0.08 | 10.03 |
| dpnew | 0.05 | 12.15 | 0.06 | 12.10 |
| eadash | 0.15 | 97.00 | 1.55 | 96.89 |
| ldash | 0.09 | 77.32 | 1.26 | 77.25 |
| lin | 0.03 | 6.95 | 0.04 | 6.83 |
| list6 | 0.04 | 21.63 | 0.10 | 21.40 |
| list6too | 0.05 | 30.93 | 0.27 | 30.81 |
| mcslock1 | 0.05 | 20.38 | 0.08 | 20.40 |
| mcslock2 | 0.06 | 28.61 | 0.09 | 28.67 |
| n_peterson | 0.05 | 16.46 | 0.06 | 16.42 |
| newcache3 | 0.09 | 89.02 | 0.51 | 88.80 |
| newlist6 | 0.05 | 38.61 | 0.07 | 38.45 |
| ns | 0.05 | 87.59 | 0.18 | 88.26 |
| ns-old | 0.06 | 87.36 | 0.19 | 87.47 |
| pingpong | 0.12 | 8.70 | 0.04 | 9.01 |
| sci | 0.77 | 245.46 | 7.98 | 246.74 |
| scierr | 0.77 | 244.36 | 7.69 | 243.30 |
| sets | 0.03 | 9.54 | 0.05 | 9.59 |
| sort5 | 0.05 | 8.10 | 0.05 | 7.96 |

Table 5.8: Compilation times for example programs distributed with the Mur$\varphi$ verifier, with hash compaction.

| | No Optimization | | With Optimization | |
|---|---|---|---|---|
| Model | Mur$\varphi$ time | C++ time | Mur$\varphi$ time | C++ time |
| student1 | 0.06 | 57.53 | 0.34 | 57.24 |
| student2 | 0.08 | 72.63 | 0.30 | 72.03 |
| student3 | 0.07 | 69.44 | 0.29 | 69.49 |
| student4 | 0.08 | 76.04 | 0.28 | 75.98 |
| student5 | 0.06 | 47.54 | 0.20 | 47.70 |
| student6 | 0.08 | 77.08 | 0.31 | 76.50 |
| student7 | 0.11 | 91.22 | 2.48 | 84.84 |
| student8 | 0.09 | 72.77 | 1.50 | 69.65 |
| student9 | 0.09 | 72.72 | 0.39 | 72.80 |
| student10 | 0.07 | 71.56 | 0.64 | 70.88 |
| student11 | 0.07 | 49.33 | 0.22 | 49.38 |

Table 5.9: Compilation times for cache coherence protocol models, without hash compaction.

| | No Optimization | | With Optimization | |
|---|---|---|---|---|
| Model | Mur$\varphi$ time | C++ time | Mur$\varphi$ time | C++ time |
| student1 | 0.06 | 58.57 | 0.28 | 58.64 |
| student2 | 0.08 | 73.82 | 0.30 | 73.18 |
| student3 | 0.09 | 70.58 | 0.30 | 70.61 |
| student4 | 0.08 | 77.19 | 0.29 | 77.29 |
| student5 | 0.06 | 48.92 | 0.21 | 48.71 |
| student6 | 0.08 | 78.36 | 0.31 | 77.56 |
| student7 | 0.11 | 92.61 | 2.48 | 86.42 |
| student8 | 0.08 | 74.02 | 1.51 | 70.90 |
| student9 | 0.09 | 74.07 | 0.40 | 73.96 |
| student10 | 0.08 | 72.75 | 0.64 | 71.98 |
| student11 | 0.07 | 50.77 | 0.22 | 50.55 |

Table 5.10: Compilation times for cache coherence protocol models, with hash compaction.

| Model | No Optimization | | With Optimization | |
|---|---|---|---|---|
| | Mur$\varphi$ time | C++ time | Mur$\varphi$ time | C++ time |
| 1394 | 0.07 | 37.08 | 0.46 | 35.38 |

Table 5.11: Compilation times for a model of part of the IEEE 1394 protocol, without hash compaction.

| Model | No Optimization | | With Optimization | |
|---|---|---|---|---|
| | Mur$\varphi$ time | C++ time | Mur$\varphi$ time | C++ time |
| 1394 | 0.08 | 38.22 | 0.43 | 36.41 |

Table 5.12: Compilation times for a model of part of the IEEE 1394 protocol, with hash compaction.

# Chapter 6

# Conclusions and Future Work

In this thesis, we assert that closing the gap between novice and expert is an important area for verification research. Novices model problems differently than experts, primarily because they understand less about the underlying tool. We have identified several source-level transformations which promise to improve the verification experience for novice users. After creating the necessary infrastructure to work with Mur$\varphi$ 3.1, we implemented one of those transformations to provide a preliminary test of our hypothesis.

As we have seen, even a single source-level transformation was able to significantly improve several models written by novices. This result supports the assertions that novice users are likely to model systems differently and less efficiently for formal verification, that many of these inefficient modeling choices can be easily detected at the source-code level, and that a verification tool can optimize away these inefficient modeling choices, thereby boosting the productivity of novice users. A tool user should not need to understand the inner details of the tool, nor adapt to those details, in order to use it effectively. This work is a step towards closing the gap between novice and expert.

Our goal was to illustrate how verification tools should adapt to novice users, rather than to advocate a specific optimization for a specific tool. Nevertheless, the most obvious direction for future work is to try the other proposed transformations and measure their effectiveness. Doing so with Mur$\varphi$ 3.1 would require a great deal of additional effort.

We made a major planning mistake during this project: choosing to modify the existing Mur$\varphi$ 3.1 compiler rather than writing our own parser. The Mur$\varphi$ language is quite large. We chose to avoid the devil we knew, hoping that the underlying code would be easier to manipulate. Then, as the project progressed and the problems mounted, we chose to press on, believing that the next big obstacle would be the last. In retrospect, writing our own source-to-source compiler would have been easier.

Given the structural problems with Mur$\varphi$ 3.1, future source-level optimization work should follow one of two approaches. Creating a source-to-source optimizer is the simpler of the two, and has the advantage that Mur$\varphi$ 3.1 would still be used for code generation, ensuring compatibility. The more comprehensive approach is to rewrite Mur$\varphi$, maintaining rigid separation between parsing, optimization, and code generation. Although there are greater drawbacks to a full rewrite, including the effort of developing a full compiler and of ensuring compatibility with existing Mur$\varphi$ compilers, the final result would be a more powerful tool.

The more general direction for future work, and the promise of greater impact, is to apply these ideas to other verification tools and languages. Possible questions to investigate include what source-level changes might reduce BDD size, what aspects of Verilog or VHDL might highlight easy-to-implement optimizations, and how might a future hardware description language or verification language be best designed to support robust ease-of-use. Considerable further research needs to be done.

# Bibliography

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[2] Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. Technical Report CMU-CS-92-160, Carnegie Mellon University, July 1992. A version of this report was published in *Computing Surveys*, Vol. 24, No. 3, September 1992, p. 293–318.

[3] K. Mani Chandy and Jayadev Misra. *Parallel Program Design – a foundation.* Addison-Wesley, 1988.

[4] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[5] David L. Dill. The Mur$\varphi$ verification system. In R. Alur and T. A. Henzinger, editors, *Computer-Aided Verification: Eighth International Conference*, pages 390–393. Springer-Verlag, July 1996. Lecture Notes in Computer Science Number 1102.

[6] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *International Conference on Computer Design.* IEEE, October 1992.

[7] David L. Dill, Seungjoon Park, and Andreas G. Nowatzyk. Formal specification of abstract memory models. In *Research on Integrated Systems: Proceedings of the 1993 Symposium*, pages 38–52. MIT Press, 1993. Seattle, Washington, March 14–16.

[8] Aarti Gupta. Formal hardware verification methods: A survey. *Formal Methods in System Design*, 1:151–238, 1992.

[9] Alan J. Hu. Formal hardware verification with BDDs: An introduction. In *Pacific Rim Conference on Communications, Computers, and Signal Processing (PACRIM)*, pages 677–682. IEEE, 1997.

[10] Alan J. Hu, Masahiro Fujita, and Chris Wilson. Formal verification of the HAL S1 system cache coherence protocol. In *International Conference on Computer Design*, pages 438–444. IEEE, 1997.

[11] Alan J. Hu, Rui Li, Xizheng Shi, and Son Vuong. Model checking a secure group communication protocol: A case study. In *Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV)*. IFIP TC6/WG6.1, 1999.

[12] *Cable PHY specification*, pages 49–112. In IEEE Std 1394-1995 [13], August 1996.

[13] *IEEE Standard for a High Performance Serial Bus*. IEEE Std 1394-1995. Insititute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017, USA, August 1996.

[14] C. Norris Ip. Using symbolic analysis to optimize explicit reachability analysis. In *IEEE International High Level Design Validation and Test Workshop*, pages 130–137. IEEE, 1999.

[15] C. Norris Ip and David L. Dill. Efficient verification of symmetric concurrent systems. In *International Conference on Computer Design*, pages 230–234. IEEE, October 1993.

[16] Jawahar Jain, Amit Narayan, Masahiro Fujita, and Alberto Sangiovanni-Vincentelli. Formal verification of combinational circuits. In *International Conference on VLSI Design*, 1997.

[17] Christoph Kern and Mark R. Greenstreet. Formal verification in hardware design: A survey. *ACM Transactions on Design Automation of Electronic Systems*, 4(2):123–193, April 1999.

[18] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Mur$\varphi$. In *Symposium on Security and Privacy*, pages 141–151. IEEE, 1997.

[19] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium*, January 1998.

[20] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997.

[21] Steven S. Muchnick. *Interprocedural Data-Flow Analysis*, pages 619–637. In *Advanced Compiler Design and Implementation* [20], 1997.

[22] Steven S. Muchnick. *Live Variables Analysis*, pages 443–446. In *Advanced Compiler Design and Implementation* [20], 1997.

[23] Steven S. Muchnick. *Loop distribution*, page 694. In *Advanced Compiler Design and Implementation* [20], 1997.

[24] Steven S. Muchnick. *Removal of Induction Variables and Linear-Function Test Replacement*, pages 447–453. In *Advanced Compiler Design and Implementation* [20], 1997.

[25] Lawrence Yang, David Gao, Jamshid Mostoufi, Raju Joshi, and Paul Loewenstein. System design methodology of UltraSPARC-I. In *32nd Design Automation Conference*, pages 7–12. ACM/IEEE, 1995.

[26] Karen Yorav and Orna Grumberg. Static analysis for state-space reductions preserving temporal logics. Technical Report CS-2000-03, The Technion, Haifa, Israel, 2000.

# Appendix A

# Mur$\varphi$ Optimizer Source

This appendix contains full source code for our Mur$\varphi$ optimizer. The source is available electronically,[1] but it is included here for archival purposes. The Internet is a powerful dynamic communication medium, but is not well suited for long term data preservation. The last ten years have seen the demise of several file location protocols (e.g., gopher and archie); it is unlikely that HTTP will fare better over time.

The ruleset rearrangement transformation is implemented as a modification of Mur$\varphi$ 3.1. The modifications to the Mur$\varphi$ 3.1 compiler itself are minimal, and listed in Section A.3.1 in unified diff format. The bulk of the optimizer mirrors the structure of the Mur$\varphi$ parse tree, with methods for manipulating the tree and for updating the necessary Mur$\varphi$ 3.1 data structures for code generation. C++ header files are presented first, followed by the associated C++ code files and finally other support files.

[1]At publication time, from `http://www.cs.ubc.ca/labs/isd/Projects/`.

## A.1 C++ Header Files

### A.1.1 mu_opt.h

```
// mu_opt.h    -*- c++ -*-

#ifndef MU_OPT_H
#define MU_OPT_H

//#include "mu_opt_base.h"
//#include "mu_opt_ste.h"
//#include "mu_opt_stmt.h"
//#include "mu_opt_decl.h"
//#include "mu_opt_rule.h"
#include "mu_opt_root.h"

#endif /* MU_OPT_H */
```

### A.1.2 mu_opt_base.h

```
// mu_opt_base.h    -*- c++ -*-

#ifndef MU_OPT_BASE_H
#define MU_OPT_BASE_H

#include <mu.h>
#include <iostream.h>
#include <list>
#include <set>

//#define MUOPT_DEBUG

typedef unsigned int uint;
typedef set<int> ScopeSet;

class MuOptObject;

typedef struct {
  void        *node;  // new child Murphi object
  MuOptObject *obj;   // new child muopt object
} OptRet;


class MuOptObject {
public:
  MuOptObject(const char *name = NULL);
  virtual ~MuOptObject();

  const char *name() const { return _name; }

  virtual void displayTree(ostream& out, uint indent) const = 0;
  virtual ScopeSet *deps(uint reqNum = 0) const = 0;
  virtual OptRet optimize();

  int depLoop(uint reqNum) const;
  static uint nextReqNum() { return ++_thisReqNum; }
```

```
  static void indentLine(ostream& out, uint indent);

private:
  char *_name;
  mutable uint _lastReqNum;
  static uint _thisReqNum;
};

#endif /* MU_OPT_BASE_H */
```

### A.1.3 mu_opt_decl.h

```
// mu_opt_decl.h    -*- c++ -*-

#ifndef MU_OPT_DECL_H
#define MU_OPT_DECL_H

#include "mu_opt_base.h"
#include "mu_opt_ste.h"
#include "mu_opt_stmt.h"
#include <map>


class MuOptTypeDecl;

class MuOptDecl : public MuOptObject {
public:
  enum Type { TypeDecl, Const, Var, Alias, Quant, Choose, Param, Proc,
              Func, Error, unknown };

protected:
  MuOptDecl(Type t, const char *name) : MuOptObject(name), _type(t) { }
public:
  virtual ~MuOptDecl();

  Type type() const { return _type; }

  static MuOptDecl *newMuOptDecl(decl *);

private:
  Type _type;
  static map<decl*,MuOptDecl*> _existing;
};

class MuOptDeclConst : public MuOptDecl {
public:
  MuOptDeclConst(constdecl *n);
  virtual ~MuOptDeclConst();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  constdecl *_node; // do not own
  MuOptTypeDecl *_typed;
};

class MuOptDeclVar : public MuOptDecl {
```

```
public:
  MuOptDeclVar(vardecl *n);
  virtual ~MuOptDeclVar();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  vardecl *_node; // do not own
  MuOptTypeDecl *_typed;
};

class MuOptDeclAlias : public MuOptDecl {
public:
  MuOptDeclAlias(aliasdecl *n);
  virtual ~MuOptDeclAlias();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  aliasdecl *_node; // do not own
  MuOptExpr _ref;
};

class MuOptDeclQuant : public MuOptDecl {
public:
  MuOptDeclQuant(quantdecl *n);
  virtual ~MuOptDeclQuant();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  quantdecl *_node; // do not own
  MuOptTypeDecl *_typed;
  MuOptExpr _left;
  MuOptExpr _right;
};

class MuOptDeclChoose : public MuOptDecl {
public:
  MuOptDeclChoose(choosedecl *n);
  virtual ~MuOptDeclChoose();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  choosedecl *_node; // do not own
  MuOptTypeDecl *_typed;
};

class MuOptDeclProc : public MuOptDecl {
public:
  MuOptDeclProc(procdecl *n);
  virtual ~MuOptDeclProc();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
```

```
private:
  procdecl *_node; // do not own
  MuOptSTEChain _params;
  MuOptSTEChain _decls;
  MuOptStmtList _body;
};

class MuOptDeclFunc : public MuOptDecl {
public:
  MuOptDeclFunc(funcdecl *n);
  virtual ~MuOptDeclFunc();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  funcdecl *_node; // do not own
  MuOptSTEChain _params;
  MuOptSTEChain _decls;
  MuOptStmtList _body;
  MuOptDecl *_returntype;
};

class MuOptDeclError : public MuOptDecl {
public:
  MuOptDeclError(error_decl *n);
  virtual ~MuOptDeclError();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  error_decl *_node; // do not own
};

#endif /* MU_OPT_DECL_H */
```

### A.1.4 mu_opt_expr.h

```
// mu_opt_expr.h    -*- c++ -*-

#ifndef MU_OPT_EXPR_H
#define MU_OPT_EXPR_H

#include "mu_opt_base.h"
#include "mu_opt_ste.h"

class MuOptExpr : public MuOptObject {
public:
  MuOptExpr(expr *e);
  virtual ~MuOptExpr();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  expr *_node; // do not own
```

```
  MuOptSTEChain *_used;
};

class MuOptDesignator {
public:
  MuOptDesignator() { }
  virtual ~MuOptDesignator();

  virtual void displayTree(ostream& out, uint indent) const = 0;
  virtual ScopeSet *deps(uint reqNum = 0) const = 0;

  virtual bool isList() const = 0;

  static MuOptDesignator *newMuOptDesignator(designator *);
};

class MuOptDesignatorInstance : public MuOptExpr, public MuOptDesignator {
public:
  MuOptDesignatorInstance(designator *d);
  virtual ~MuOptDesignatorInstance();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

  bool isList() const;

private:
  MuOptSTE _origin;
  MuOptExpr _arrayref;
  MuOptSTE _fieldref;
};

class MuOptExprList : public MuOptObject {
public:
  MuOptExprList(exprlist *e);
  virtual ~MuOptExprList();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  list<MuOptExpr*> _list;
};

class MuOptDesignatorList : public MuOptObject, public MuOptDesignator {
public:
  MuOptDesignatorList(designator *d);
  virtual ~MuOptDesignatorList();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

  bool isList() const;

private:
  list<MuOptDesignatorInstance*> _list;
};

#endif /* MU_OPT_EXPR_H */
```

### A.1.5 mu_opt_param.h

```
// mu_opt_param.h    -*- c++ -*-

#ifndef MU_OPT_PARAM_H
#define MU_OPT_PARAM_H

#include "mu_opt_decl.h"
#include <map>

class MuOptParam : public MuOptDecl {
public:
  enum Type { Value, Var, Const, /*Name,*/ Error, unknown };

protected:
  MuOptParam(Type t, const char *name);
public:
  virtual ~MuOptParam();

  Type type() const { return _type; }

  virtual void displayTree(ostream& out, uint indent) const;

  static MuOptParam *newMuOptParam(param *);

private:
  Type _type;
  static map<param*,MuOptParam*> _existing;
};

class MuOptParamValue : public MuOptParam {
public:
  MuOptParamValue(valparam *);
  virtual ~MuOptParamValue();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  valparam *_node;  // do not own
};

class MuOptParamVar : public MuOptParam {
public:
  MuOptParamVar(varparam *);
  virtual ~MuOptParamVar();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  varparam *_node;  // do not own
};

class MuOptParamConst : public MuOptParam {
public:
  MuOptParamConst(constparam *);
  virtual ~MuOptParamConst();

  virtual void displayTree(ostream& out, uint indent) const;
```

```
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  constparam *_node;  // do not own
};

/*
class MuOptParamName : public MuOptParam {
public:
  MuOptParamName(nameparam *);
  virtual ~MuOptParamName();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  nameparam *_node;  // do not own
};
*/

class MuOptParamError : public MuOptParam {
public:
  MuOptParamError(param *);
  virtual ~MuOptParamError();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  param *_node;  // do not own
};

#endif /* MU_OPT_PARAM_H */
```

### A.1.6 mu_opt_root.h

```
// mu_opt_root.h    -*- c++ -*-

#ifndef MU_OPT_ROOT_H
#define MU_OPT_ROOT_H

#include "mu_opt_base.h"
#include "mu_opt_ste.h"
#include "mu_opt_rule.h"

class MuOptRoot {
public:
  MuOptRoot(program *prog);
  virtual ~MuOptRoot();

  void displayTree(ostream& out = cout) const;
  ScopeSet *deps() const;

  void optimize();

private:
  program *_prog;  // do not own
  MuOptSTEChain *_globals;
```

```
  MuOptSTEChain *_procedures;
  MuOptRuleList *_rules;
};

#endif /* MU_OPT_ROOT_H */
```

### A.1.7 mu_opt_rule.h

```
// mu_opt_rule.h    -*- c++ -*-

#ifndef MU_OPT_RULE_H
#define MU_OPT_RULE_H

#include "mu_opt_base.h"
#include "mu_opt_ste.h"
#include "mu_opt_expr.h"
#include "mu_opt_stmt.h"

// bogus class so I can have a single pointer for rules and rule lists
// without going all the way up to MuOptObject
class MuOptRuleBase : public MuOptObject {
public:
  MuOptRuleBase(const char *name = NULL) : MuOptObject(name) { }
  virtual ~MuOptRuleBase();

  virtual bool isList() const = 0;
  virtual void rebuildRuleInfo(const ScopeSet *encl) = 0;
};

class MuOptRule : public MuOptRuleBase {
public:
  enum Type { Simple, Startstate, Invar, Quant, Choose, Alias,
              Fair, Live };

protected:
  MuOptRule(Type type, const char *name, rule *node) :
    MuOptRuleBase(name), _type(type), _node(node) { }
public:
  virtual ~MuOptRule();

  Type type() const { return _type; }
  rule *node() { return _node; }

  bool isList() const;

  static MuOptRule *newMuOptRule(rule*);
  static MuOptRule *newMuOptRule(MuOptRule*);

private:
  Type _type;
  rule *_node;
};

class MuOptRuleList: public MuOptRuleBase {
public:
  MuOptRuleList(rule*);
  MuOptRuleList(const MuOptRuleList* = NULL);
  virtual ~MuOptRuleList();
```

```
  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual OptRet optimize();
  virtual void rebuildRuleInfo(const ScopeSet *encl);

  void split(MuOptRule*& car, MuOptRuleList*& cdr) const;
  MuOptRule *pop_front();
  void append(MuOptRule* r);
  void append(const MuOptRuleList* rl);
  uint size() const { return _list.size(); }

  bool isList() const;

private:
  list<MuOptRule*> _list;
};

class MuOptRuleSimple : public MuOptRule {
public:
  MuOptRuleSimple(simplerule*);
protected:
  MuOptRuleSimple(simplerule*, Type);
public:
  virtual ~MuOptRuleSimple();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual void rebuildRuleInfo(const ScopeSet *encl);

  virtual const char *typeName() const;

private:
  MuOptSTEChain _enclosures;
  MuOptExpr     _condition;
  MuOptSTEChain _locals;
  MuOptStmtList _body;
  bool          _unfair;
};

class MuOptRuleStartstate : public MuOptRuleSimple {
public:
  MuOptRuleStartstate(startstate*);
  virtual ~MuOptRuleStartstate();

  virtual const char *typeName() const;
};

class MuOptRuleInvar : public MuOptRuleSimple {
public:
  MuOptRuleInvar(invariant*);
  virtual ~MuOptRuleInvar();

  virtual const char *typeName() const;
};

class MuOptRuleQuant : public MuOptRule {
public:
  MuOptRuleQuant(quantrule*);
  virtual ~MuOptRuleQuant();
```

```
  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual OptRet optimize();
  virtual void rebuildRuleInfo(const ScopeSet *encl);

  int scope() const { return _quant.scope(); }

private:
  MuOptSTE _quant;
  MuOptRuleBase *_rules;
};

class MuOptRuleChoose : public MuOptRule {
public:
  MuOptRuleChoose(chooserule*);
  virtual ~MuOptRuleChoose();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual OptRet optimize();
  virtual void rebuildRuleInfo(const ScopeSet *encl);

  int scope() const { return _index.scope(); }

private:
  MuOptSTE _index;
  MuOptDesignator *_set;
  MuOptRuleBase *_rules;
};

class MuOptRuleAlias : public MuOptRule {
public:
  MuOptRuleAlias(aliasrule*);
  virtual ~MuOptRuleAlias();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual OptRet optimize();
  virtual void rebuildRuleInfo(const ScopeSet *encl);

  int scope() const { return _aliases.scope(); }

private:
  MuOptSTE _aliases;
  MuOptRuleBase *_rules;
};

class MuOptRuleFair : public MuOptRule {
public:
  MuOptRuleFair(fairness*);
  virtual ~MuOptRuleFair();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual void rebuildRuleInfo(const ScopeSet *encl);
};

class MuOptRuleLive : public MuOptRule {
public:
```

```
  MuOptRuleLive(liveness*);
  virtual ~MuOptRuleLive();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
  virtual void rebuildRuleInfo(const ScopeSet *encl);
};

#endif /* MU_OPT_RULE_H */
```

### A.1.8 mu_opt_ste.h

```
// mu_opt_ste.h    -*- c++ -*-

#ifndef MU_OPT_STE_H
#define MU_OPT_STE_H

#include "mu_opt_base.h"

class MuOptDecl;

class MuOptSTE : public MuOptObject {
public:
  MuOptSTE(ste *s);
  virtual ~MuOptSTE();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

  int scope() const { return _scope; }

  ste *node() { return _node; }

private:
  ste *_node;  // do not own
  int _scope;
  int _lextype;
  MuOptDecl *_value;
};

class MuOptRuleSimple;  // HACK: need to do some funky stuff to Murphi side
                        //       for some of the rule optimizations *+*+*

class MuOptSTEChain : public MuOptObject {
public:
  // run from start up to but not including stop
  MuOptSTEChain(ste *start, ste *stop=NULL);
  MuOptSTEChain(stelist *start, stelist *stop=NULL);

  MuOptSTEChain(stecoll *s);
  virtual ~MuOptSTEChain();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

  ScopeSet *scopes() const;

private:
```

```
  list<MuOptSTE*>& chain() { return _chain; }
  friend MuOptRuleSimple;

private:
  list<MuOptSTE*> _chain;
};

#endif /* MU_OPT_STE_H */
```

### A.1.9 mu_opt_stmt.h

```
// mu_opt_stmt.h    -*- c++ -*-

#ifndef MU_OPT_STMT_H
#define MU_OPT_STMT_H

#include "mu_opt_base.h"
#include "mu_opt_expr.h"
#include <map>

class MuOptStmt : public MuOptObject {
public:
  enum Type { Assignment, While, If, Case, Switch, For, Proc, Clear, Error,
              Assert, Put, Alias, Aliasstmt, Return, Undefine, MultisetAdd,
              MultisetRemove, Null, unknown };

protected:
  MuOptStmt(Type t) : _type(t) { }
public:
  virtual ~MuOptStmt();

  Type type() const { return _type; }

  static MuOptStmt *newMuOptStmt(stmt *);

private:
  Type _type;
  static map<stmt*,MuOptStmt*> _existing;
};

class MuOptStmtList : public MuOptObject {
public:
  MuOptStmtList(stmt *s);
  virtual ~MuOptStmtList();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  list<MuOptStmt*> _list;
};

class MuOptStmtAssignment : public MuOptStmt {
public:
  MuOptStmtAssignment(assignment *a);
  virtual ~MuOptStmtAssignment();

  virtual void displayTree(ostream& out, uint indent) const;
```

```
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  assignment *_node;  // do not own
  MuOptExpr _src;
  MuOptDesignator *_target;
};

class MuOptStmtWhile : public MuOptStmt {
public:
  MuOptStmtWhile(whilestmt *w);
  virtual ~MuOptStmtWhile();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  whilestmt *_node;  // do not own
  MuOptExpr _test;
  MuOptStmtList _body;
};

class MuOptStmtIf : public MuOptStmt {
public:
  MuOptStmtIf(ifstmt *i);
  virtual ~MuOptStmtIf();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  ifstmt *_node;  // do not own
  MuOptExpr _test;
  MuOptStmtList _body;
  MuOptStmtList _else;
};

class MuOptStmtCase : public MuOptStmt {
public:
  MuOptStmtCase(caselist *c);
  virtual ~MuOptStmtCase();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  caselist *_node;  // do not own
  MuOptExprList _values;
  MuOptStmtList _body;
};

class MuOptStmtCaseList : public MuOptObject {
public:
  MuOptStmtCaseList(caselist *c);
  virtual ~MuOptStmtCaseList();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
```

```
  list<MuOptStmtCase*> _list;
};

class MuOptStmtSwitch : public MuOptStmt {
public:
  MuOptStmtSwitch(switchstmt *s);
  virtual ~MuOptStmtSwitch();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  switchstmt *_node;  // do not own
  MuOptExpr _expr;
  MuOptStmtCaseList _cases;
  MuOptStmtList _default;
};

class MuOptStmtFor : public MuOptStmt {
public:
  MuOptStmtFor(forstmt *f);
  virtual ~MuOptStmtFor();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  forstmt *_node;  // do not own
  MuOptSTEChain _index;
  MuOptStmtList _body;
};

class MuOptStmtProc : public MuOptStmt {
public:
  MuOptStmtProc(proccall *p);
  virtual ~MuOptStmtProc();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  proccall *_node;  // do not own
  MuOptSTE _procedure;
  MuOptExprList _actuals;
};

class MuOptStmtClear : public MuOptStmt {
public:
  MuOptStmtClear(clearstmt *c);
  virtual ~MuOptStmtClear();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  clearstmt *_node;  // do not own
  MuOptDesignator *_target;
};

class MuOptStmtError : public MuOptStmt {
```

```
public:
  MuOptStmtError(errorstmt *e);
protected:
  MuOptStmtError(errorstmt *e, bool isAssert);
public:
  virtual ~MuOptStmtError();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

protected:
  const char *errstring() const { return _string; }

private:
  errorstmt *_node;  // do not own
  const char *_string;  // do not own
};

class MuOptStmtAssert : public MuOptStmtError {
public:
  MuOptStmtAssert(assertstmt *a);
  virtual ~MuOptStmtAssert();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  assertstmt *_node;  // do not own
  MuOptExpr _test;
};

class MuOptStmtPut : public MuOptStmt {
public:
  MuOptStmtPut(putstmt *p);
  virtual ~MuOptStmtPut();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  putstmt *_node;  // do not own
};

class MuOptStmtAlias : public MuOptStmt {
public:
  MuOptStmtAlias(alias *a);
  virtual ~MuOptStmtAlias();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  alias *_node;  // do not own
};

class MuOptStmtAliasstmt : public MuOptStmt {
public:
  MuOptStmtAliasstmt(aliasstmt *a);
  virtual ~MuOptStmtAliasstmt();
```

```
  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  aliasstmt *_node;  // do not own
  MuOptSTEChain _aliases;
  MuOptStmtList _body;
};

class MuOptStmtReturn : public MuOptStmt {
public:
  MuOptStmtReturn(returnstmt *r);
  virtual ~MuOptStmtReturn();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  returnstmt *_node;  // do not own
  MuOptExpr _retexpr;
};

class MuOptStmtUndefine : public MuOptStmt {
public:
  MuOptStmtUndefine(undefinestmt *u);
  virtual ~MuOptStmtUndefine();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  undefinestmt *_node;  // do not own
  MuOptDesignator *_target;
};

class MuOptStmtMultisetAdd : public MuOptStmt {
public:
  MuOptStmtMultisetAdd(multisetaddstmt *m);
  virtual ~MuOptStmtMultisetAdd();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  multisetaddstmt *_node;  // do not own
  MuOptDesignator *_element;
  MuOptDesignator *_target;
};

class MuOptStmtMultisetRemove : public MuOptStmt {
public:
  MuOptStmtMultisetRemove(multisetremovestmt *m);
  virtual ~MuOptStmtMultisetRemove();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  multisetremovestmt *_node;  // do not own
  MuOptSTE _index;
```

```
  MuOptDesignator *_target;
  MuOptExpr _criterion;
};

class MuOptStmtNull : public MuOptStmt {
public:
  MuOptStmtNull();
  virtual ~MuOptStmtNull();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;
};

#endif /* MU_OPT_STMT_H */
```

### A.1.10 mu_opt_typedecl.h

```
// mu_opt_typedecl.h    -*- c++ -*-

#ifndef MU_OPT_TYPEDECL_H
#define MU_OPT_TYPEDECL_H

#include "mu_opt_decl.h"
#include <map>


class MuOptTypeDecl : public MuOptDecl {
public:
  enum Type { Enum, Range, Array, MultiSet, MultiSetID, Record, Scalarset,
              Union, Error, unknown };

protected:
  MuOptTypeDecl(Type t, const char *name)
    : MuOptDecl(TypeDecl, name), _type(t) { }
public:
  virtual ~MuOptTypeDecl();

  Type type() const { return _type; }

  static MuOptTypeDecl *newMuOptTypeDecl(typedecl *);

  virtual void displayTree(ostream& out, uint indent) const;

private:
  Type _type;
  static map<typedecl*,MuOptTypeDecl*> _existing;
};

class MuOptTypeDeclEnum : public MuOptTypeDecl {
public:
  MuOptTypeDeclEnum(enumtypedecl *n);
  virtual ~MuOptTypeDeclEnum();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  enumtypedecl *_node; // do not own
```

```
  int _left;
  int _right;
  MuOptSTEChain _idvalues;
};

class MuOptTypeDeclRange : public MuOptTypeDecl {
public:
  MuOptTypeDeclRange(subrangetypedecl *n);
  virtual ~MuOptTypeDeclRange();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  subrangetypedecl *_node; // do not own
  int _left;
  int _right;
};

class MuOptTypeDeclArray : public MuOptTypeDecl {
public:
  MuOptTypeDeclArray(arraytypedecl *n);
  virtual ~MuOptTypeDeclArray();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  arraytypedecl *_node; // do not own
  MuOptTypeDecl *_indexType;
  MuOptTypeDecl *_elementType;
};

class MuOptTypeDeclMultiSet : public MuOptTypeDecl {
public:
  MuOptTypeDeclMultiSet(multisettypedecl *n);
  virtual ~MuOptTypeDeclMultiSet();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  multisettypedecl *_node; // do not own
};

class MuOptTypeDeclMultiSetID : public MuOptTypeDecl {
public:
  MuOptTypeDeclMultiSetID(multisetidtypedecl *n);
  virtual ~MuOptTypeDeclMultiSetID();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  multisetidtypedecl *_node; // do not own
};

class MuOptTypeDeclRecord : public MuOptTypeDecl {
public:
  MuOptTypeDeclRecord(recordtypedecl *n);
```

```
  virtual ~MuOptTypeDeclRecord();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  recordtypedecl *_node; // do not own
  MuOptSTEChain _fields;
};

class MuOptTypeDeclScalarset : public MuOptTypeDecl {
public:
  MuOptTypeDeclScalarset(scalarsettypedecl *n);
  virtual ~MuOptTypeDeclScalarset();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  scalarsettypedecl *_node; // do not own
  int _left;
  int _right;
  MuOptSTE _idvalues; // ***** should this be a chain?
};

class MuOptTypeDeclUnion : public MuOptTypeDecl {
public:
  MuOptTypeDeclUnion(uniontypedecl *n);
  virtual ~MuOptTypeDeclUnion();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  uniontypedecl *_node; // do not own
  MuOptSTEChain _unionmembers;
};

class MuOptTypeDeclError : public MuOptTypeDecl {
public:
  MuOptTypeDeclError(errortypedecl *n);
  virtual ~MuOptTypeDeclError();

  virtual void displayTree(ostream& out, uint indent) const;
  virtual ScopeSet *deps(uint reqNum = 0) const;

private:
  errortypedecl *_node; // do not own
};

#endif /* MU_OPT_TYPEDECL_H */
```

## A.2 C++ Program Files

### A.2.1 mu_opt_base.cc

```
// mu_opt_base.cc    -*- c++ -*-

#include "mu_opt_base.h"

uint MuOptObject::_thisReqNum = 0;

MuOptObject::MuOptObject(const char *name) {
  if(name) {
    _name = new char [strlen(name) + 1];
    strcpy(_name, name);
  } else
    _name = NULL;

  _lastReqNum = 0;
}

MuOptObject::~MuOptObject() {
  delete [] _name;
}

int MuOptObject::depLoop(uint reqNum) const {
  if(reqNum == _lastReqNum)
    return 1;
  _lastReqNum = reqNum;
  return 0;
}

void MuOptObject::indentLine(ostream& out, uint indent) {
  for(uint i = 0; i < indent; i++)
    out << "  ";
}

OptRet MuOptObject::optimize() {
  //assert(0);  // eventually I want to make this purely virtual
  OptRet result;

  result.node = NULL;
  result.obj = NULL;

  return result;
}
```

### A.2.2 mu_opt_decl.cc

```
// mu_opt_decl.cc    -*- c++ -*-

#include "mu_opt_decl.h"
#include "mu_opt_param.h"
#include "mu_opt_typedecl.h"


map<decl*,MuOptDecl*> MuOptDecl::_existing;
```

```
MuOptDecl::~MuOptDecl() {
}

MuOptDecl *MuOptDecl::newMuOptDecl(decl *d) {
  MuOptDecl *result = NULL;
  if(d)
    if(_existing.find(d) == _existing.end()) {
      switch(d->getclass()) {
      case decl::Type:
        result=MuOptTypeDecl::newMuOptTypeDecl(dynamic_cast<typedecl*>(d));
        break;
      case decl::Const:
        result = new MuOptDeclConst(dynamic_cast<constdecl*>(d));
        break;
      case decl::Var:
        result = new MuOptDeclVar(dynamic_cast<vardecl*>(d));
        break;
      case decl::Alias:
        result = new MuOptDeclAlias(dynamic_cast<aliasdecl*>(d));
        break;
      case decl::Quant:
        result = new MuOptDeclQuant(dynamic_cast<quantdecl*>(d));
        break;
      case decl::Choose:
        result = new MuOptDeclChoose(dynamic_cast<choosedecl*>(d));
        break;
      case decl::Param:
        result = MuOptParam::newMuOptParam(dynamic_cast<param*>(d));
        break;
      case decl::Proc:
        result = new MuOptDeclProc(dynamic_cast<procdecl*>(d));
        break;
      case decl::Func:
        result = new MuOptDeclFunc(dynamic_cast<funcdecl*>(d));
        break;
      case decl::Error_decl:
        result = new MuOptDeclError(dynamic_cast<error_decl*>(d));
        break;
      default:
        assert(0);
      }
      _existing[d] = result;
    } else {
      result = _existing[d];
#ifdef MUOPT_DEBUG
      cerr << "reissuing existing decl\n";
#endif
    }
  return result;
}


MuOptDeclConst::MuOptDeclConst(constdecl *n)
  : MuOptDecl(Const, n ? n->name : NULL), _node(n), _typed(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclConst::MuOptDeclConst(constdecl*)\n";
#endif
  //if(n && n->gettype())
  //  _typed = MuOptTypeDecl::newMuOptTypeDecl(n->gettype());
```

```
}

MuOptDeclConst::~MuOptDeclConst() {
  //delete _typed;
}

void MuOptDeclConst::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclConst::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Const \"" << name() << "\"\n";
  if(_typed) {
    indentLine(out, indent + 1);
    out << "[typed]\n";
    _typed->displayTree(out, indent + 2);
  }
}

ScopeSet *MuOptDeclConst::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  if(_typed)
    result = _typed->deps(reqNum);
  else
    result = new ScopeSet;

  //temp = .deps(reqNum);
  //result->insert(temp->begin(), temp->end());
  //delete temp;

  return result;
}


MuOptDeclVar::MuOptDeclVar(vardecl *n)
  : MuOptDecl(Var, n ? n->name : NULL), _node(n), _typed(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclVar::MuOptDeclVar(vardecl*)\n";
#endif
  //if(n && n->gettype())
  //  _typed = MuOptTypeDecl::newMuOptTypeDecl(n->gettype());
}

MuOptDeclVar::~MuOptDeclVar() {
  //delete _typed;
}

void MuOptDeclVar::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclVar::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Var \"" << name() << "\"\n";
```

```
  if(_typed) {
    indentLine(out, indent + 1);
    out << "[typed]\n";
    _typed->displayTree(out, indent + 2);
  }
}

ScopeSet *MuOptDeclVar::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  if(_typed)
    result = _typed->deps(reqNum);
  else
    result = new ScopeSet;

  //temp = .deps(reqNum);
  //result->insert(temp->begin(), temp->end());
  //delete temp;

  return result;
}


MuOptDeclAlias::MuOptDeclAlias(aliasdecl *n)
  : MuOptDecl(Alias, n ? n->name : NULL), _node(n),
    _ref(n ? n->getexpr() : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclAlias::MuOptDeclAlias(aliasdecl*)\n";
#endif
}

MuOptDeclAlias::~MuOptDeclAlias() {
}

void MuOptDeclAlias::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclAlias::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Alias \"" << name() << "\"\n";
  indentLine(out, indent + 1);
  out << "ref\n";
  _ref.displayTree(out, indent + 2);
}

ScopeSet *MuOptDeclAlias::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;
```

```
  result = _ref.deps(reqNum);

  //temp = .deps(reqNum);
  //result->insert(temp->begin(), temp->end());
  //delete temp;

  return result;
}


MuOptDeclQuant::MuOptDeclQuant(quantdecl *n)
  : MuOptDecl(Quant, n ? n->name : NULL), _node(n), _typed(NULL),
    _left(n ? n->left : NULL), _right(n ? n->right : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclQuant::MuOptDeclQuant(quantdecl*)\n";
#endif
  //if(n && n->type)
  //  _typed = MuOptTypeDecl::newMuOptTypeDecl(n->type);
}

MuOptDeclQuant::~MuOptDeclQuant() {
  //delete _typed;
}

void MuOptDeclQuant::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclQuant::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Quant \"" << name() << "\"\n";
  if(_typed) {
    indentLine(out, indent + 1);
    out << "[typed]\n";
    _typed->displayTree(out, indent + 2);
  }
}

ScopeSet *MuOptDeclQuant::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _left.deps(reqNum);

  temp = _right.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  if(_typed) {
    temp = _typed->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}
```

```
MuOptDeclChoose::MuOptDeclChoose(choosedecl *n)
  : MuOptDecl(Choose, n ? n->name : NULL), _node(n), _typed(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclChoose::MuOptDeclChoose(choosedecl*)\n";
#endif
  //if(n && n->type)
  //  _typed = MuOptTypeDecl::newMuOptTypeDecl(n->type);
}

MuOptDeclChoose::~MuOptDeclChoose() {
  //delete _typed;
}

void MuOptDeclChoose::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclChoose::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Choose \"" << name() << "\"\n";
  if(_typed)
    _typed->displayTree(out, indent + 1);
}

ScopeSet *MuOptDeclChoose::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = new ScopeSet;

  if(_typed) {
    temp = _typed->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}


MuOptDeclProc::MuOptDeclProc(procdecl *n)
  : MuOptDecl(Proc, n ? n->name : NULL), _node(n),
    _params(n ? n->params : NULL), _decls(n ? n->decls : NULL),
    _body(n ? n->body : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclProc::MuOptDeclProc(procdecl*)\n";
#endif
}

MuOptDeclProc::~MuOptDeclProc() {
}

void MuOptDeclProc::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
```

```
  cerr << "MuOptDeclProc::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Proc \"" << name() << "\"\n";
  if(_node && 0 /*++++ need to verify that this is all valid */) {
    indentLine(out, indent + 1);
    out << "params\n";
    _params.displayTree(out, indent + 2);
    indentLine(out, indent + 1);
    out << "decls\n";
    _decls.displayTree(out, indent + 2);
    indentLine(out, indent + 1);
    out << "body\n";
    _body.displayTree(out, indent + 2);
  }
}

ScopeSet *MuOptDeclProc::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  // This is sort of a shotgun approach.  In theory nothing above this
  // point should ever have dependencies on things like the scope of
  // params.
  result = _params.deps(reqNum);

  temp = _decls.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}


MuOptDeclFunc::MuOptDeclFunc(funcdecl *n)
  : MuOptDecl(Proc, n ? n->name : NULL), _node(n),
    _params(n ? n->params : NULL), _decls(n ? n->decls : NULL),
    _body(n ? n->body : NULL), _returntype(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclFunc::MuOptDeclFunc(funcdecl*)\n";
#endif
  if(_node) {
    _returntype = MuOptDecl::newMuOptDecl(_node->returntype);
    assert(_returntype);
  }
}

MuOptDeclFunc::~MuOptDeclFunc() {
  //delete _returntype;
}
```

```
void MuOptDeclFunc::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclFunc::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Func \"" << name() << "\"\n";
  if(_node) {
    indentLine(out, indent + 1);
    out << "params\n";
    _params.displayTree(out, indent + 2);
    indentLine(out, indent + 1);
    out << "decls\n";
    _decls.displayTree(out, indent + 2);
    indentLine(out, indent + 1);
    out << "body\n";
    _body.displayTree(out, indent + 2);
    indentLine(out, indent + 1);
    out << "returntype\n";
    _returntype->displayTree(out, indent + 2);
  }
}

ScopeSet *MuOptDeclFunc::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _params.deps(reqNum);

  temp = _decls.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _returntype->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}


MuOptDeclError::MuOptDeclError(error_decl *n)
  : MuOptDecl(Error, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclError::MuOptDeclError(error_decl*)\n";
#endif
}

MuOptDeclError::~MuOptDeclError() {
}

void MuOptDeclError::displayTree(ostream& out, uint indent) const {
```

```
#ifdef MUOPT_DEBUG
  cerr << "MuOptDeclError::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Error_decl \"" << name() << "\"\n";
}

ScopeSet *MuOptDeclError::deps(uint reqNum = 0) const {
  return new ScopeSet;
}
```

### A.2.3 mu_opt_expr.cc

```
// mu_opt_expr.cc    -*- c++ -*-

#include "mu_opt_expr.h"


MuOptExpr::MuOptExpr(expr *e) : _node(e), _used(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptExpr::MuOptExpr(expr*)\n";
#endif
  _used = new MuOptSTEChain(_node ? _node->used_stes() : NULL);
  assert(_used);
}

MuOptExpr::~MuOptExpr() {
  //delete _used;
}

void MuOptExpr::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptExpr::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "[expr]\n";
  _used->displayTree(out, indent + 1);
}

ScopeSet *MuOptExpr::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptExpr::deps(uint=0)\n";
#endif
  ScopeSet *result;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _used->deps(reqNum);

  //cout << "Expr::deps():";
  //for(ScopeSet::const_iterator i=result->begin(); i!=result->end(); i++)
  //  cout << " " << *i;
  //cout << "\n";
```

```
  return result;
}

MuOptDesignator::~MuOptDesignator() {
}

MuOptDesignator *MuOptDesignator::newMuOptDesignator(designator *d) {
    if(!d || d->left)
      // list
      return new MuOptDesignatorList(d);
    else
      // not a list
      return new MuOptDesignatorInstance(d);
}

MuOptDesignatorInstance::MuOptDesignatorInstance(designator *d)
  : MuOptExpr(d), _origin(d->origin), _arrayref(d->arrayref),
  _fieldref(d->fieldref) {
#ifdef MUOPT_DEBUG
  cerr<<"MuOptDesignatorInstance::MuOptDesignatorInstance(designator*)\n";
#endif
}

MuOptDesignatorInstance::~MuOptDesignatorInstance() {
}

bool MuOptDesignatorInstance::isList() const {
  return false;
}

void MuOptDesignatorInstance::displayTree(ostream& out, uint indent) const{
  MuOptExpr::displayTree(out, indent);
}

ScopeSet *MuOptDesignatorInstance::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDesignatorInstance::deps(uint=0)\n";
#endif
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = MuOptExpr::deps(reqNum);

  temp = _origin.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _arrayref.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _fieldref.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;
```

```
  return result;
}


MuOptExprList::MuOptExprList(exprlist *e) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptExprList::MuOptExprList(exprlist*)\n";
#endif
  while(e) {
    if(e->e) {
      MuOptExpr *n = new MuOptExpr(e->e);
      if(n)
        _list.push_back(n);
    }
    e = e->next;
  }
}

MuOptExprList::~MuOptExprList() {
  while(! _list.empty()) {
    //delete _list.front();
    _list.pop_front();
  }
}

void MuOptExprList::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptExprList::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "[exprlist]\n";
  for(list<MuOptExpr*>::const_iterator i=_list.begin();i!=_list.end();i++)
    (*i)->displayTree(out, indent + 1);
}

ScopeSet *MuOptExprList::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = new ScopeSet;
  for(list<MuOptExpr*>::const_iterator i=_list.begin();i!=_list.end();i++){
    temp = (*i)->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}

MuOptDesignatorList::MuOptDesignatorList(designator *d) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDesignatorList::MuOptDesignatorList(designator*)\n";
#endif
  assert(d);
  while(d) {
```

```
    MuOptDesignatorInstance *newdesig = new MuOptDesignatorInstance(d);
    assert(newdesig);
    _list.push_back(newdesig);
    d = d->left;
  }
}

MuOptDesignatorList::~MuOptDesignatorList() {
  while(!_list.empty()) {
    delete _list.front();
    _list.pop_front();
  }
}

bool MuOptDesignatorList::isList() const {
  return true;
}

void MuOptDesignatorList::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDesignatorList::displayTree(ostream&,uint) const\n";
#endif
  //indentLine(indent - 1);
  //out << "[designator list]\n";
  for(list<MuOptDesignatorInstance*>::const_iterator i=_list.begin();
      i!=_list.end(); i++)
    (*i)->displayTree(out, indent);
}

ScopeSet *MuOptDesignatorList::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptDesignatorList::deps(uint=0) const\n";
#endif

  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = new ScopeSet;

  for(list<MuOptDesignatorInstance*>::const_iterator i=_list.begin();
      i!=_list.end(); i++) {
    temp = (*i)->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}
```

### A.2.4 mu_opt_param.cc

```
// mu_opt_param.cc    -*- c++ -*-
```

```
#include "mu_opt_param.h"


map<param*,MuOptParam*> MuOptParam::_existing;


MuOptParam::MuOptParam(Type t, const char *name)
  : MuOptDecl(Param, name), _type(t) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParam::MuOptParam(Type, const char*)\n";
#endif
}

MuOptParam::~MuOptParam() {
}

void MuOptParam::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParam::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Param \"" << name() << "\"\n";
}

MuOptParam *MuOptParam::newMuOptParam(param *p) {
  MuOptParam *result = NULL;
  if(p)
    if(_existing.find(p) == _existing.end()) {
      switch(p->getparamclass()) {
      case param::Value:
        result = new MuOptParamValue(dynamic_cast<valparam*>(p));
        break;
      case param::Var:
        result = new MuOptParamVar(dynamic_cast<varparam*>(p));
        break;
      case param::Const:
        result = new MuOptParamConst(dynamic_cast<constparam*>(p));
        break;
      case param::Name:
        assert(0);  // these don't actually exist in Murphi 3.1!?!
        //result = new MuOptParamName(dynamic_cast<nameparam*>(p));
        break;
      case param::Error_param:
        result = new MuOptParamError(p);
        break;
      default:
        assert(0);
        break;
      }
      _existing[p] = result;
    } else {
      result = _existing[p];
#ifdef MUOPT_DEBUG
      cerr << "reissuing existing param\n";
#endif
    }
  return result;
}
```

```
MuOptParamValue::MuOptParamValue(valparam *n)
  : MuOptParam(Value, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamValue::MuOptParamValue(valparam*)\n";
#endif
}

MuOptParamValue::~MuOptParamValue() {
}

void MuOptParamValue::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamValue::displayTree(int) const\n";
#endif
  MuOptParam::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "param_class is Value\n";
}

ScopeSet *MuOptParamValue::deps(uint reqNum = 0) const {
  return new ScopeSet;
}


MuOptParamVar::MuOptParamVar(varparam *n)
  : MuOptParam(Var, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamVar::MuOptParamVar(varparam*)\n";
#endif
}

MuOptParamVar::~MuOptParamVar() {
}

void MuOptParamVar::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamVar::displayTree(int) const\n";
#endif
  MuOptParam::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "param_class is Var\n";
}

ScopeSet *MuOptParamVar::deps(uint reqNum = 0) const {
  return new ScopeSet;
}


MuOptParamConst::MuOptParamConst(constparam *n)
  : MuOptParam(Const, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamConst::MuOptParamConst(constparam*)\n";
#endif
}

MuOptParamConst::~MuOptParamConst() {
}

void MuOptParamConst::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
```

```
  cerr << "MuOptParamConst::displayTree(int) const\n";
#endif
  MuOptParam::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "param_class is Const\n";
}

ScopeSet *MuOptParamConst::deps(uint reqNum = 0) const {
  return new ScopeSet;
}


/*
MuOptParamName::MuOptParamName(nameparam *n)
  : MuOptParam(Name, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamName::MuOptParamName(nameparam*)\n";
#endif
}

MuOptParamName::~MuOptParamName() {
}

void MuOptParamName::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamName::displayTree(int) const\n";
#endif
  MuOptParam::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "param_class is Name\n";
}

ScopeSet *MuOptParamName::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = .deps(reqNum);

  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}
*/


MuOptParamError::MuOptParamError(param *n)
  : MuOptParam(Error, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamError::MuOptParamError(param*)\n";
#endif
}

MuOptParamError::~MuOptParamError() {
```

```
}

void MuOptParamError::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptParamError::displayTree(int) const\n";
#endif
  MuOptParam::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "param_class is Error\n";
}

ScopeSet *MuOptParamError::deps(uint reqNum = 0) const {
  return new ScopeSet;
}
```

### A.2.5 mu_opt_root.cc

```
// mu_opt_root.cc    -*- c++ -*-

#include "mu_opt_root.h"
#include <fstream.h>


MuOptRoot::MuOptRoot(program *p)
  : _prog(p), _globals(NULL), _procedures(NULL), _rules(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRoot::MuOptRoot(program*)\n";
#endif
  if(_prog) {
    _globals = new MuOptSTEChain(_prog->globals);
    _procedures = new MuOptSTEChain(_prog->procedures, _prog->globals);
    _rules = new MuOptRuleList(_prog->rules);
  }
}

MuOptRoot::~MuOptRoot() {
  delete _globals;
  delete _procedures;
  delete _rules;
}

void MuOptRoot::displayTree(ostream& out = cout) const {
  if(_prog) {
    out << "Globals:\n";
    _globals->displayTree(out, 1);
    out << "\nProcedures:\n";
    _procedures->displayTree(out, 1);
    out << "\nRules:\n";
    _rules->displayTree(out, 1);
  }
}

ScopeSet *MuOptRoot::deps() const {
  ScopeSet *result, *temp;
  uint reqNum = MuOptObject::nextReqNum();

  result = _globals->deps(reqNum);
```

```
  temp = _procedures->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _rules->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

void MuOptRoot::optimize() {
  OptRet or;

  // before snapshot
  /*{
    ofstream before("/tmp/before");
    assert(before.is_open());
    displayTree(before);
  }*/

  // hack some manipulations

  cout << "Optimizing rules\n";

  if(_rules) {
    or = _rules->optimize();
    assert(or.node == NULL);
    //if(or.node)
    //  _prog->rules = static_cast<rule*>(or.node);
    assert(or.obj == NULL);
    //*+++*
    //if(dynamic_cast<MuOptRuleList*>(or.obj))
      //_rules = dynamic_cast<MuOptRuleList>(or.obj);
  }

  // Now, because the parse tree isn't actually used for rule
  // generation, we get to rebuild all sorts of other data structures
  // they are using from our hacked parse tree.  (I guess somewhere
  // along the way someone didn't understand that optimizations were
  // supposed to go in between parsing and code generation, so they
  // just hacked their nifty idea for an optimization into the parsing
  // stage.)
  simplerule::SimpleRuleList = NULL;
  _rules->rebuildRuleInfo(NULL);


  cout << "Optimization complete\n";

  // now that I'm not bothering to update the parse tree (since Murphi
  // ignores it) there isn't any point in doing the rebuild after

  // clean up and rebuild tree
  //delete _globals;
  //delete _procedures;
  //delete _rules;
  //if(_prog) {
  //  _globals = new MuOptSTEChain(_prog->globals);
  //  _procedures = new MuOptSTEChain(_prog->procedures, _prog->globals);
  //  _rules = new MuOptRuleList(_prog->rules);
```

```
  //}

  // after snapshot
  /*{
    ofstream after("/tmp/after");
    assert(after.is_open());
    displayTree(after);
  }*/
}
```

### A.2.6 mu_opt_rule.cc

```
// mu_opt_rule.cc    -*- c++ -*-

#include "mu_opt_rule.h"
#include <typeinfo>

MuOptRuleBase::~MuOptRuleBase() {
}

MuOptRule::~MuOptRule() {
}

bool MuOptRule::isList() const {
  return false;
}

MuOptRule *MuOptRule::newMuOptRule(rule *r) {
  MuOptRule *result = NULL;
  if(r)
    switch(r->getclass()) {
    case rule::Simple:
      result = new MuOptRuleSimple(dynamic_cast<simplerule*>(r));
      break;
    case rule::Startstate:
      result = new MuOptRuleStartstate(dynamic_cast<startstate*>(r));
      break;
    case rule::Invar:
      result = new MuOptRuleInvar(dynamic_cast<invariant*>(r));
      break;
    case rule::Quant:
      result = new MuOptRuleQuant(dynamic_cast<quantrule*>(r));
      break;
    case rule::Choose:
      result = new MuOptRuleChoose(dynamic_cast<chooserule*>(r));
      break;
    case rule::Alias:
      result = new MuOptRuleAlias(dynamic_cast<aliasrule*>(r));
      break;
    case rule::Fair:
      result = new MuOptRuleFair(dynamic_cast<fairness*>(r));
      break;
    case rule::Live:
      result = new MuOptRuleLive(dynamic_cast<liveness*>(r));
      break;
    default:
      assert(0);
    }
```

```
  return result;
}

MuOptRule *MuOptRule::newMuOptRule(MuOptRule *r) {
  MuOptRule *result = NULL;
  if(r)
    if(typeid(*r) == typeid(MuOptRuleSimple))
      result = new MuOptRuleSimple(*dynamic_cast<MuOptRuleSimple*>(r));
    else if(typeid(*r) == typeid(MuOptRuleStartstate))
      result = new MuOptRuleStartstate(*dynamic_cast<MuOptRuleStartstate*>
                                       (r));
    else if(typeid(*r) == typeid(MuOptRuleInvar))
      result = new MuOptRuleInvar(*dynamic_cast<MuOptRuleInvar*>(r));
    else if(typeid(*r) == typeid(MuOptRuleQuant))
      result = new MuOptRuleQuant(*dynamic_cast<MuOptRuleQuant*>(r));
    else if(typeid(*r) == typeid(MuOptRuleChoose))
      result = new MuOptRuleChoose(*dynamic_cast<MuOptRuleChoose*>(r));
    else if(typeid(*r) == typeid(MuOptRuleAlias))
      result = new MuOptRuleAlias(*dynamic_cast<MuOptRuleAlias*>(r));
    else if(typeid(*r) == typeid(MuOptRuleFair))
      result = new MuOptRuleFair(*dynamic_cast<MuOptRuleFair*>(r));
    else if(typeid(*r) == typeid(MuOptRuleLive))
      result = new MuOptRuleLive(*dynamic_cast<MuOptRuleLive*>(r));
    else
      assert(0);
  return result;
}

MuOptRuleSimple::MuOptRuleSimple(simplerule *n)
  : MuOptRule(Simple, n ? n->name : NULL, n),
    _enclosures(n ? n->enclosures : NULL),
    _condition(n ? n->condition : NULL),
    _locals(n ? n->locals : NULL), _body(n ? n->body : NULL),
    _unfair(n ? n->unfair : false) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleSimple::MuOptRuleSimple(simplerule*)\n";
#endif
}

MuOptRuleSimple::MuOptRuleSimple(simplerule*n, Type _type)
  : MuOptRule(_type, n ? n->name : NULL, n),
    _enclosures(n ? n->enclosures : NULL),
    _condition(n && _type!=Startstate ? n->condition : NULL),
    _locals(n && _type!=Invar ? n->locals : NULL),
    _body(n && _type!=Invar ? n->body : NULL),
    _unfair(n ? n->unfair : false) {
}

MuOptRuleSimple::~MuOptRuleSimple() {
}

const char *MuOptRuleSimple::typeName() const {
  return "Simple";
}

void MuOptRuleSimple::rebuildRuleInfo(const ScopeSet *encl) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRule" << typeName()
       << "::rebuildRuleInfo(const ScopeSet*)\n";
#endif
```

```
simplerule *n = dynamic_cast<simplerule*>(node());
assert(n);

// update simplerule::SimpleRuleList
n->NextSimpleRule = simplerule::SimpleRuleList;
simplerule::SimpleRuleList = n;

// update enclosures of this simplerule
list<MuOptSTE*>& chain = _enclosures.chain();  // HACK:  unziiiip...
if(encl) {
  // This is my second attempt (see CVS rev 1.10 for previous
  // version, commented out).  This is a bit ugly, but it should do.
  ste *last = NULL;
  ste *cur = n->enclosures;
  while(cur) {
    // HACK: global scope is hardcoded to 1
    // HACK++: don't throw out scope 0 either
    if(cur->scope > 1 && encl->find(cur->scope) == encl->end()) {
      cout << "cleanup:      rule \"" << name()
           << "\" removing unused enclosing scope "
           << cur->scope
           << " from simple rule deps\n";
      cur = cur->next;
      if(last)
        last->next = cur;
      else
        n->enclosures = cur;
    } else {
      last = cur;
      cur = cur->next;
    }
  }
  MuOptSTEChain temp(n->enclosures);
  chain.clear();
  chain.splice(chain.end(), temp.chain());
} else {
  n->enclosures = NULL;
  chain.clear();
}

// now that the enclosures have been updated, update the hardcoded
// size parameter so that the magic loop constants work
// <rant>Why is this put into a variable????  If they have a
// built-in method to compute it, why not just call the damn method
// in cpp_code.C rather than using the variable???  Public variables
// are evil and should never be used except under very extraordinary
// circumstances, and then only with great care.</rant>
// We also need to update the indep_card parameter.  In theory, if
// I've optimized things correctly, the new indep_card should always
// be 1.  (I have no idea why asking the appropriate Murphi code to
// recompute it doesn't come up with 1, but maybe this will become
// clear to me in time.)
int newsize = n->CountSize(n->enclosures);
if(newsize != n->size || n->indep_card != 1) {
  cout << "cleanup:      updating magic loop constants for rule \""
       << name() << "\"\n";
  n->size = newsize;
  n->indep_card = 1;
}
```

```
}

void MuOptRuleSimple::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRule" << typeName() << "::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "ruleclass is " << typeName();
  if(name())
    out << " (" << name() << ")";
  out << "\n";
  indentLine(out, indent + 1);
  out << "enclosures\n";
  _enclosures.displayTree(out, indent + 2);
  if(type() != Startstate) {
    indentLine(out, indent + 1);
    out << "condition\n";
    _condition.displayTree(out, indent + 2);
  }
  if(type() != Invar) {
    indentLine(out, indent + 1);
    out << "locals\n";
    _locals.displayTree(out, indent + 2);
    indentLine(out, indent + 1);
    out << "body\n";
    _body.displayTree(out, indent + 2);
  }
  if(_unfair) {
    indentLine(out, indent + 1);
    out << "unfair\n";
  }
}

ScopeSet *MuOptRuleSimple::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRule" << typeName() << "::deps(uint=0) const\n";
#endif
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  result = new ScopeSet;

  if(depLoop(reqNum))
    return result;

  // HACK: these conditions on type() are ugly, should polymorph
  if(type() != Startstate) {
    temp = _condition.deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

 // HACK: these conditions on type() are ugly, should polymorph
  if(type() != Invar) {
    temp = _locals.deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
```

```
    temp = _body.deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}

MuOptRuleStartstate::MuOptRuleStartstate(startstate *n)
  : MuOptRuleSimple(dynamic_cast<simplerule*>(n), Startstate) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleStartstate::MuOptRuleStartstate(simplerule*)\n";
#endif
}

MuOptRuleStartstate::~MuOptRuleStartstate() {
}

const char *MuOptRuleStartstate::typeName() const {
  return "Startstate";
}

MuOptRuleInvar::MuOptRuleInvar(invariant *n)
  : MuOptRuleSimple(dynamic_cast<simplerule*>(n), Invar) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleInvar::MuOptRuleInvar(simplerule*)\n";
#endif
}

MuOptRuleInvar::~MuOptRuleInvar() {
}

const char *MuOptRuleInvar::typeName() const {
  return "Invar";
}

MuOptRuleQuant::MuOptRuleQuant(quantrule *n)
  : MuOptRule(Quant, (n && n->quant && n->quant->name
                      ? n->quant->name->getname() : NULL), n),
    _quant(n ? n->quant : NULL), _rules(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleQuant::MuOptRuledQuant(quantrule*)\n";
#endif
  _rules = new MuOptRuleList(n ? n->rules : NULL);
  assert(_rules);
}

MuOptRuleQuant::~MuOptRuleQuant() {
  //delete _rules;
}

void MuOptRuleQuant::rebuildRuleInfo(const ScopeSet *encl) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleQuant::rebuildRuleInfo(const ScopeSet*)\n";
#endif
  ScopeSet newencl;
  if(encl)
    newencl.insert(encl->begin(), encl->end());
  newencl.insert(scope());
  _rules->rebuildRuleInfo(&newencl);
```

```
}

void MuOptRuleQuant::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleQuant::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "ruleclass is Quant\n";
  indentLine(out, indent + 1);
  out << "quant\n";
  _quant.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "rules\n";
  _rules->displayTree(out, indent + 2);
}

ScopeSet *MuOptRuleQuant::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleQuant::deps(uint=0) const\n";
#endif

  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _quant.deps(reqNum);

  temp = _rules->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

OptRet MuOptRuleQuant::optimize() {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleQuant::optimize()\n";
#endif
  OptRet result, rtmp;
  ScopeSet *temp;

  result.node = NULL;
  result.obj = NULL;

  // if _rules is a list, turn me into a list containing multiple
  // quant rules with only a single rule in each of their _rules
  do {
    if(_rules->isList()) {
      int size = dynamic_cast<MuOptRuleList*>(_rules)->size();
      cout << "pseudo-opt:   ruleset \"" << name() << "\" (scope "
           << scope() << ") ";
      if(size > 1)
        cout << "dividing into " << size << " branches\n";
      else
        cout << "rules list has single member, extracting\n";
      MuOptRuleList *newlist = new MuOptRuleList();
```

```
      assert(newlist);
      MuOptRule *car;
      MuOptRuleList *cdr = dynamic_cast<MuOptRuleList*>(_rules);
      assert(cdr);
      while(cdr) {
        cdr->split(car, cdr);
        if(car) { // SLOPPY
          MuOptRuleQuant *twin = new MuOptRuleQuant(*this);
          twin->_rules = car;
          rtmp = twin->optimize();
          if(rtmp.obj != NULL)
            if(dynamic_cast<MuOptRule*>(rtmp.obj))
              newlist->append(dynamic_cast<MuOptRule*>(rtmp.obj));
            else if(dynamic_cast<MuOptRuleList*>(rtmp.obj))
              newlist->append(dynamic_cast<MuOptRuleList*>(rtmp.obj));
            else
              assert(0);
          else
            newlist->append(twin);
        }
      }
      result.node = NULL;
      result.obj = newlist;
      return result;
    } else {
      rtmp = _rules->optimize();
      if(rtmp.obj != NULL)
        if(dynamic_cast<MuOptRuleBase*>(rtmp.obj))
          _rules = dynamic_cast<MuOptRuleBase*>(rtmp.obj);
        else
          assert(0);
    }
  } while(_rules->isList());

  // if remaining rule is not dependent on us, commit suicide
  temp = _rules->deps();
  if(temp->find(scope()) == temp->end()) {
    cout << "optimization: ruleset \"" << name() << "\" (scope "
         << scope() << ") with no dependencies, removing\n";
    // delete myself
    MuOptRule *myrule = dynamic_cast<MuOptRule*>(_rules);
    if(myrule) {
      result.node = myrule->node();
      result.obj = myrule;
    }
  }

  return result;
}


MuOptRuleChoose::MuOptRuleChoose(chooserule *n)
  : MuOptRule(Choose, (n && n->index && n->index->name
                       ? n->index->name->getname() : NULL), n),
    _index(n ? n->index : NULL), _set(NULL), _rules(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleChoose::MuOptRuledChoose(chooserule*)\n";
#endif
  _set = MuOptDesignator::newMuOptDesignator(n ? n->set : NULL);
  assert(_set);
```

```
  _rules = new MuOptRuleList(n ? n->rules : NULL);
  assert(_rules);
}

MuOptRuleChoose::~MuOptRuleChoose() {
  //delete _rules;
}

void MuOptRuleChoose::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleChoose::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "ruleclass is Choose\n";
  indentLine(out, indent + 1);
  out << "index\n";
  _index.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "set\n";
  _set->displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "rules\n";
  _rules->displayTree(out, indent + 2);
}

ScopeSet *MuOptRuleChoose::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleChoose::deps(uint=0) const\n";
#endif
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _index.deps(reqNum);

  temp = _set->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _rules->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

void MuOptRuleChoose::rebuildRuleInfo(const ScopeSet *encl) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleChoose::rebuildRuleInfo(const ScopeSet*)\n";
#endif
  ScopeSet newencl;
  if(encl)
    newencl.insert(encl->begin(), encl->end());
  newencl.insert(scope());
  _rules->rebuildRuleInfo(&newencl);
}
```

```
OptRet MuOptRuleChoose::optimize() {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleChoose::rebuildRuleInfo(const ScopeSet*)\n";
#endif
  OptRet result, rtmp;
  ScopeSet *temp;

  result.node = NULL;
  result.obj = NULL;

  // if _rules is a list, turn me into a list containing multiple
  // choose rules with only a single rule in each of their _rules
  do {
    if(_rules->isList()) {
      int size = dynamic_cast<MuOptRuleList*>(_rules)->size();
      cout << "pseudo-opt:   choose \"" << name() << "\" (scope "
           << scope() << ") ";
      if(size > 1)
        cout << "dividing into " << size << " branches\n";
      else
        cout << "rules list has single member, extracting\n";
      MuOptRuleList *newlist = new MuOptRuleList();
      assert(newlist);
      MuOptRule *car;
      MuOptRuleList *cdr = dynamic_cast<MuOptRuleList*>(_rules);
      assert(cdr);
      while(cdr) {
        cdr->split(car, cdr);
        if(car) { // SLOPPY
          MuOptRuleChoose *twin = new MuOptRuleChoose(*this);
          twin->_rules = car;
          rtmp = twin->optimize();
          if(rtmp.obj != NULL)
            if(dynamic_cast<MuOptRule*>(rtmp.obj))
              newlist->append(dynamic_cast<MuOptRule*>(rtmp.obj));
            else if(dynamic_cast<MuOptRuleList*>(rtmp.obj))
              newlist->append(dynamic_cast<MuOptRuleList*>(rtmp.obj));
            else
              assert(0);
          else
            newlist->append(twin);
        }
      }
      result.node = NULL;
      result.obj = newlist;
      return result;
    } else {
      rtmp = _rules->optimize();
      if(rtmp.obj != NULL)
        if(dynamic_cast<MuOptRuleBase*>(rtmp.obj))
          _rules = dynamic_cast<MuOptRuleBase*>(rtmp.obj);
        else
          assert(0);
    }
  } while(_rules->isList());

  // if remaining rule is not dependent on us, commit suicide
  temp = _rules->deps();
  if(temp->find(scope()) == temp->end()) {
```

```
    cout << "optimization: choose \"" << name() << "\" (scope "
         << scope() << ") with no dependencies, removing\n";
    // delete myself
    MuOptRule *myrule = dynamic_cast<MuOptRule*>(_rules);
    if(myrule) {
      result.node = myrule->node();
      result.obj = myrule;
    }
  }

  return result;
}

MuOptRuleAlias::MuOptRuleAlias(aliasrule *n)
  : MuOptRule(Alias, (n && n->aliases && n->aliases->name
                      ? n->aliases->name->getname() : NULL), n),
    _aliases(n ? n->aliases : NULL), _rules(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleAlias::MuOptRuleAlias(aliasrule*)\n";
#endif
  _rules = new MuOptRuleList(n ? n->rules : NULL);
  assert(_rules);
}

MuOptRuleAlias::~MuOptRuleAlias() {
  //delete _rules;
}

void MuOptRuleAlias::rebuildRuleInfo(const ScopeSet *encl) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleAlias::rebuildRuleInfo(const ScopeSet*)\n";
#endif
  ScopeSet newencl;
  if(encl)
    newencl.insert(encl->begin(), encl->end());
  newencl.insert(_aliases.scope());
  _rules->rebuildRuleInfo(&newencl);
}

void MuOptRuleAlias::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleAlias::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "ruleclass is Alias\n";
  indentLine(out, indent + 1);
  out << "aliases\n";
  _aliases.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "rules\n";
  _rules->displayTree(out, indent + 2);
}

ScopeSet *MuOptRuleAlias::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleAlias::deps(uint=0) const\n";
#endif

  ScopeSet *result, *temp;
```

```
  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _aliases.deps(reqNum);

  temp = _rules->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

OptRet MuOptRuleAlias::optimize() {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleAlias::optimize()\n";
#endif
  OptRet result, rtmp;
  ScopeSet *temp;

  result.node = NULL;
  result.obj = NULL;

  // if _rules is a list, turn me into a list containing multiple
  // alias rules with only a single rule in each of their _rules
  do {
    if(_rules->isList()) {
      int size = dynamic_cast<MuOptRuleList*>(_rules)->size();
      cout << "pseudo-opt:   alias \"" << name() << "\" (scope "
           << scope() << ") ";
      if(size > 1)
        cout << "dividing into " << size << " branches\n";
      else
        cout << "rules list has single member, extracting\n";
      MuOptRuleList *newlist = new MuOptRuleList();
      assert(newlist);
      MuOptRule *car;
      MuOptRuleList *cdr = dynamic_cast<MuOptRuleList*>(_rules);
      assert(cdr);
      while(cdr) {
        cdr->split(car, cdr);
        if(car) {
          MuOptRuleAlias *twin = new MuOptRuleAlias(*this);
          twin->_rules = car;
          rtmp = twin->optimize();
          if(rtmp.obj != NULL)
            if(dynamic_cast<MuOptRule*>(rtmp.obj))
              newlist->append(dynamic_cast<MuOptRule*>(rtmp.obj));
            else if(dynamic_cast<MuOptRuleList*>(rtmp.obj))
              newlist->append(dynamic_cast<MuOptRuleList*>(rtmp.obj));
            else
              assert(0);
          else
            newlist->append(twin);
        }
      }
      result.node = NULL;
      result.obj = newlist;
```

```
      return result;
    } else {
      rtmp = _rules->optimize();
      if(rtmp.obj != NULL)
        if(dynamic_cast<MuOptRuleBase*>(rtmp.obj))
          _rules = dynamic_cast<MuOptRuleBase*>(rtmp.obj);
        else
          assert(0);
    }
  } while(_rules->isList());

  // if remaining rule is not dependent on us, commit suicide
  temp = _rules->deps();
  if(temp->find(scope()) == temp->end()) {
    cout << "optimization: alias \"" << name() << "\" (scope "
         << scope() << ") with no dependencies, removing\n";
    // delete myself
    MuOptRule *myrule = dynamic_cast<MuOptRule*>(_rules);
    if(myrule) {
      result.node = myrule->node();
      result.obj = myrule;
    }
  }

  return result;
}


MuOptRuleFair::MuOptRuleFair(fairness *n)
  : MuOptRule(Fair, n ? n->name : NULL, n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleFair::MuOptRuledFair(fairness*)\n";
#endif
  assert(0);
}

MuOptRuleFair::~MuOptRuleFair() {
}

void MuOptRuleFair::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleFair::displayTree(int) const\n";
#endif
}

ScopeSet *MuOptRuleFair::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleFair::deps(uint=0) const\n";
#endif

  assert(0);
  return NULL;
  /*
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;
```

```
  result = .deps(reqNum);

  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
  */
}

void MuOptRuleFair::rebuildRuleInfo(const ScopeSet *encl) {
  assert(0);
}

MuOptRuleLive::MuOptRuleLive(liveness *n)
  : MuOptRule(Live, n ? n->name : NULL, n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleLive::MuOptRuledLive(liveness*)\n";
#endif
  assert(0);
}

MuOptRuleLive::~MuOptRuleLive() {
}

void MuOptRuleLive::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleLive::displayTree(int) const\n";
#endif
}

ScopeSet *MuOptRuleLive::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleLive::deps(uint=0) const\n";
#endif

  assert(0);
  return NULL;
  /*
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = .deps(reqNum);

  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
  */
}

void MuOptRuleLive::rebuildRuleInfo(const ScopeSet *encl) {
  assert(0);
```

```
}

MuOptRuleList::MuOptRuleList(rule *r) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleList::MuOptRuleList(rule*)\n";
#endif
  assert(r);
  while(r) {
    MuOptRule *newrule = MuOptRule::newMuOptRule(r);
    assert(newrule);
    _list.push_back(newrule);
    r = r->next;
  }
}

MuOptRuleList::MuOptRuleList(const MuOptRuleList *r) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleList::MuOptRuleList(MuOptRuleList*)\n";
#endif
  if(r)
    for(list<MuOptRule*>::const_iterator i = r->_list.begin();
        i != r->_list.end(); i++)
      _list.push_back(*i);
}

MuOptRuleList::~MuOptRuleList() {
  while(!_list.empty()) {
    delete _list.front();
    _list.pop_front();
  }
}

void MuOptRuleList::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleLive::displayTree(int) const\n";
#endif
  //indentLine(indent - 1);
  //out << "[rule list]\n";
  for(list<MuOptRule*>::const_iterator i=_list.begin();i!=_list.end();i++)
    (*i)->displayTree(out, indent);
}

ScopeSet *MuOptRuleList::deps(uint reqNum = 0) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleList::deps(uint=0) const\n";
#endif

  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = new ScopeSet;

  for(list<MuOptRule*>::const_iterator i=_list.begin();i!=_list.end();i++){
    temp = (*i)->deps(reqNum);
    result->insert(temp->begin(), temp->end());
```

```
    delete temp;
  }

  return result;
}

OptRet MuOptRuleList::optimize() {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleList::optimize()\n";
#endif
  OptRet result, temp;
  bool modified = true;

  result.node = NULL;
  result.obj = NULL;

  while(modified) {
    modified = false;
    list<MuOptRule*>::iterator i = _list.begin();
    while(i != _list.end()) {
      temp = (*i)->optimize();
      // ignore temp.node
      if(dynamic_cast<MuOptRule*>(temp.obj)) {
        *i = dynamic_cast<MuOptRule*>(temp.obj);
        i++;
        modified = true;
      } else if(dynamic_cast<MuOptRuleList*>(temp.obj)) {
        cout <<"cleanup:      merging generated list with existing list\n";
        MuOptRuleList *sublist = dynamic_cast<MuOptRuleList*>(temp.obj);
        list<MuOptRule*>::iterator j = i, k = i;
        i++;
        _list.insert(j, sublist->_list.begin(), sublist->_list.end());
        _list.erase(k);
        modified = true;
        i = _list.end();
      } else if(temp.obj != NULL)
        assert(0);
      else
        i++;
    }
  }

  // if we only have one child, return the child and delete ourself
  if(_list.size() == 1) {
    cout << "pseudo-opt:   single member rule list -> rule\n";
    //result.node = NULL;  // no transformation necessary on the
    //                     // Murphi tree
    result.obj = _list.front();
  }

  return result;
}

bool MuOptRuleList::isList() const {
  return true;
}

void MuOptRuleList::rebuildRuleInfo(const ScopeSet *encl) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptRuleList::rebuildRuleInfo(const ScopeSet*)\n";
```

```
#endif
  for(list<MuOptRule*>::const_iterator i=_list.begin();i!=_list.end();i++)
    (*i)->rebuildRuleInfo(encl);
}

void MuOptRuleList::split(MuOptRule*& car, MuOptRuleList*& cdr) const {
  if(_list.size() > 1) {
    cdr = new MuOptRuleList(this);
    car = cdr->pop_front();
  } else {
    cdr = NULL;
    if(_list.size() == 1)
      car = _list.front();
    else
      car = NULL;
  }
}

MuOptRule *MuOptRuleList::pop_front() {
  if(_list.size() > 0) {
    MuOptRule *result = _list.front();
    _list.pop_front();
    return result;
  } else
    return NULL;
}

void MuOptRuleList::append(MuOptRule *r) {
  if(r)
    _list.push_back(r);
}

void MuOptRuleList::append(const MuOptRuleList *rl) {
  if(rl)
    _list.insert(_list.end(), rl->_list.begin(), rl->_list.end());
}
```

### A.2.7 mu_opt_ste.cc

```
// mu_opt_ste.cc    -*- c++ -*-

#include "mu_opt_ste.h"
#include "mu_opt_decl.h"


MuOptSTE::MuOptSTE(ste *s)
  : MuOptObject(s ? (s->getname() ? s->getname()->getname() : NULL) :NULL),
    _node(s), _scope(s ? s->getscope() : -1),
    _lextype(s ? (s->getname() ? s->getname()->getlextype() : -2) : -1),
    _value(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptSTE::MuOptSTE(ste*)\n";
#endif
  //assert(_lextype != -1);
  assert(_lextype != -2);
  if(_node) {
    _value = MuOptDecl::newMuOptDecl(_node->getvalue());
    assert(_value);
```

```
  }
}

MuOptSTE::~MuOptSTE() {
  //delete _value;
}

void MuOptSTE::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptSTE::displayTree(int) const\n";
#endif
  if(_node) {
    indentLine(out, indent);
    out << "[ste]";
    if(_node->getname())
      out << " " << _scope << "::" << name() << "(" << _lextype << ")";
    out << "\n";
    if(_value) {
      indentLine(out, indent + 1);
      out << "value\n";
      _value->displayTree(out, indent + 2);
    }
  }
}

ScopeSet *MuOptSTE::deps(uint reqNum = 0) const {
  ScopeSet *result;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  if(_value)
    result = _value->deps(reqNum);
  else
    result = new ScopeSet;

  result->insert(_scope);

  return result;
}

MuOptSTEChain::MuOptSTEChain(ste *start, ste *stop) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptSTEChain::MuOptSTEChain(ste*, ste*)\n";
#endif
  for(ste *i = start; i && i != stop; i = i->next) {
    MuOptSTE *s = new MuOptSTE(i);
    assert(s);
    if(s)
      _chain.push_back(s);
  }
}

MuOptSTEChain::MuOptSTEChain(stelist *start, stelist *stop) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptSTEChain::MuOptSTEChain(stelist*, stelist*)\n";
#endif
```

```
  for(stelist *i = start; i && i != stop; i = i->next)
    for(ste *j = i->s; j; j = j->next) {
      MuOptSTE *s = new MuOptSTE(j);
      assert(s);
      if(s)
        _chain.push_back(s);
    }
}

MuOptSTEChain::MuOptSTEChain(stecoll *s) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptSTEChain::MuOptSTEChain(stecoll*)\n";
#endif
  // The way this one works doesn't make a ton of sense to me, but then
  // why have redundant data structures?  (Ans: because Murphi is only
  // weakly OO, and not at all OO when it comes to STEs.)
  if(s)
    for(stelist *i = s->first; i; i = i->next) {
      MuOptSTE *n = new MuOptSTE(i->s);
      assert(s);
      if(n)
        _chain.push_back(n);
    }
}

MuOptSTEChain::~MuOptSTEChain() {
  while(! _chain.empty()) {
    //delete _chain.front();
    _chain.pop_front();
  }
}

void MuOptSTEChain::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptSTEChain::displayTree(int) const\n";
#endif
  for(list<MuOptSTE*>::const_iterator i = _chain.begin();
      i != _chain.end(); i++) {
    (*i)->displayTree(out, indent);
  }
}

ScopeSet *MuOptSTEChain::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = new ScopeSet;
  for(list<MuOptSTE*>::const_iterator i = _chain.begin();
      i != _chain.end(); i++) {
    temp = (*i)->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
```

```
}


ScopeSet *MuOptSTEChain::scopes() const {
  ScopeSet *result = new ScopeSet;

  for(list<MuOptSTE*>::const_iterator i = _chain.begin();
      i != _chain.end(); i++)
    result->insert((*i)->scope());

  return result;
}
```

### A.2.8 mu_opt_stmt.cc

```
// mu_opt_stmt.cc    -*- c++ -*-

#include "mu_opt_stmt.h"
#include <typeinfo>


map<stmt*,MuOptStmt*> MuOptStmt::_existing;


MuOptStmt::~MuOptStmt() {
}

MuOptStmt *MuOptStmt::newMuOptStmt(stmt *s) {
  MuOptStmt *result = NULL;
  if(s)
    if(_existing.find(s) == _existing.end()) {
      // no getclass() here, so get to use RTTI (woohoo!)
      if(typeid(*s) == typeid(assignment))
        result = new MuOptStmtAssignment(dynamic_cast<assignment*>(s));
      else if(typeid(*s) == typeid(whilestmt))
        result = new MuOptStmtWhile(dynamic_cast<whilestmt*>(s));
      else if(typeid(*s) == typeid(ifstmt))
        result = new MuOptStmtIf(dynamic_cast<ifstmt*>(s));
      else if(typeid(*s) == typeid(caselist))
        result = new MuOptStmtCase(dynamic_cast<caselist*>(s));
      else if(typeid(*s) == typeid(switchstmt))
        result = new MuOptStmtSwitch(dynamic_cast<switchstmt*>(s));
      else if(typeid(*s) == typeid(forstmt))
        result = new MuOptStmtFor(dynamic_cast<forstmt*>(s));
      else if(typeid(*s) == typeid(proccall))
        result = new MuOptStmtProc(dynamic_cast<proccall*>(s));
      else if(typeid(*s) == typeid(clearstmt))
        result = new MuOptStmtClear(dynamic_cast<clearstmt*>(s));
      else if(typeid(*s) == typeid(errorstmt))
        result = new MuOptStmtError(dynamic_cast<errorstmt*>(s));
      else if(typeid(*s) == typeid(assertstmt))
        result = new MuOptStmtAssert(dynamic_cast<assertstmt*>(s));
      else if(typeid(*s) == typeid(putstmt))
        result = new MuOptStmtPut(dynamic_cast<putstmt*>(s));
      else if(typeid(*s) == typeid(alias))
        result = new MuOptStmtAlias(dynamic_cast<alias*>(s));
      else if(typeid(*s) == typeid(aliasstmt))
        result = new MuOptStmtAliasstmt(dynamic_cast<aliasstmt*>(s));
```

```
      else if(typeid(*s) == typeid(returnstmt))
        result = new MuOptStmtReturn(dynamic_cast<returnstmt*>(s));
      else if(typeid(*s) == typeid(undefinestmt))  // not on "the list"
        result = new MuOptStmtUndefine(dynamic_cast<undefinestmt*>(s));
      else if(typeid(*s) == typeid(multisetaddstmt))  // not on "the list"
        result=new MuOptStmtMultisetAdd(dynamic_cast<multisetaddstmt*>(s));
      else if(typeid(*s) == typeid(multisetremovestmt)) //not on "the list"
        result = new
          MuOptStmtMultisetRemove(dynamic_cast<multisetremovestmt*>(s));
      else if(s == nullstmt)  // have to read carefully to find this one
        result = new MuOptStmtNull();
      else
        assert(0);
      _existing[s] = result;
    } else {
      result = _existing[s];
#ifdef MUOPT_DEBUG
      cerr << "reissuing existing stmt\n";
#endif
    }
  return result;
}

MuOptStmtList::MuOptStmtList(stmt *s) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtList::MuOptStmtList(stmt*)\n";
#endif
  while(s) {
    MuOptStmt *mos = MuOptStmt::newMuOptStmt(s);
    if(mos)
      _list.push_back(mos);
    s = s->next;
  }
}

MuOptStmtList::~MuOptStmtList() {
  while(! _list.empty()) {
    //delete _list.front();
    _list.pop_front();
  }
}

void MuOptStmtList::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtList::displayTree(int) const\n";
#endif
  //indentLine(indent - 1);
  //out << "[stmtlist]\n";
  for(list<MuOptStmt*>::const_iterator i=_list.begin();i!=_list.end();i++)
    (*i)->displayTree(out, indent);
}

ScopeSet *MuOptStmtList::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;
```

```
  result = new ScopeSet;
  for(list<MuOptStmt*>::const_iterator i=_list.begin();i!=_list.end();i++){
    temp = (*i)->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}


MuOptStmtAssignment::MuOptStmtAssignment(assignment *a) :
  MuOptStmt(Assignment), _node(a), _src(a ? a->src : NULL), _target(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAssignment::MuOptStmtAssignment(assignment*)\n";
#endif
  assert(a);
  _target = MuOptDesignator::newMuOptDesignator(a->target);
  assert(_target);
}

MuOptStmtAssignment::~MuOptStmtAssignment() {
}

void MuOptStmtAssignment::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAssignment::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "assignment\n";
  indentLine(out, indent + 1);
  out << "target\n";
  _target->displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "src\n";
  _src.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtAssignment::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _target->deps(reqNum);

  temp = _src.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtWhile::MuOptStmtWhile(whilestmt *w) :
  MuOptStmt(While), _node(w), _test(w->test), _body(_node->body) {
#ifdef MUOPT_DEBUG
```

```
  cerr << "MuOptStmtWhile::MuOptStmtWhile(whilestmt*)\n";
#endif
}

MuOptStmtWhile::~MuOptStmtWhile() {
}

void MuOptStmtWhile::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtWhile::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "whilestmt\n";
  indentLine(out, indent + 1);
  out << "test\n";
  _test.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "body\n";
  _body.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtWhile::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _test.deps(reqNum);

  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtIf::MuOptStmtIf(ifstmt *i) :
  MuOptStmt(If), _node(i), _test(i->test), _body(i->body),
  _else(i->elsecode) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtIf::MuOptStmtIf(ifstmt*)\n";
#endif
}

MuOptStmtIf::~MuOptStmtIf() {
}

void MuOptStmtIf::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtIf::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "ifstmt\n";
  indentLine(out, indent + 1);
  out << "test\n";
  _test.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
```

```
  out << "body\n";
  _body.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "elsecode\n";
  _else.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtIf::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _test.deps(reqNum);

  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _else.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtCase::MuOptStmtCase(caselist *c) :
  MuOptStmt(Case), _node(c), _values(c->values), _body(c->body) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtCase::MuOptStmtCase(caselist*)\n";
#endif
}

MuOptStmtCase::~MuOptStmtCase() {
}

void MuOptStmtCase::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtCase::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "case\n";
  indentLine(out, indent + 1);
  out << "values\n";
  _values.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "body\n";
  _body.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtCase::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
```

```
    return new ScopeSet;

  result = _values.deps(reqNum);

  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtCaseList::MuOptStmtCaseList(caselist *c) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtCaseList::MuOptStmtCaseList(caselist*)\n";
#endif
  while(c) {
    MuOptStmtCase *mosc = new MuOptStmtCase(c);
    if(mosc)
      _list.push_back(mosc);
    c = c->next;
  }
}

MuOptStmtCaseList::~MuOptStmtCaseList() {
  while(! _list.empty()) {
    //delete _list.front();
    _list.pop_front();
  }
}

void MuOptStmtCaseList::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtCaseList::displayTree(int) const\n";
#endif
  for(list<MuOptStmtCase*>::const_iterator i = _list.begin();
      i != _list.end(); i++)
    (*i)->displayTree(out, indent);
}

ScopeSet *MuOptStmtCaseList::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = new ScopeSet;
  for(list<MuOptStmtCase*>::const_iterator i = _list.begin();
      i != _list.end(); i++) {
    temp = (*i)->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}

MuOptStmtSwitch::MuOptStmtSwitch(switchstmt *s) :
```

```
  MuOptStmt(Switch), _node(s), _expr(s->switchexpr), _cases(s->cases),
  _default(s->elsecode) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtSwitch::MuOptStmtSwitch(switchstmt*)\n";
#endif
}

MuOptStmtSwitch::~MuOptStmtSwitch() {
}

void MuOptStmtSwitch::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtSwitch::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "switchstmt\n";
  indentLine(out, indent + 1);
  out << "switchexpr\n";
  _expr.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "cases\n";
  _cases.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "elsecode\n";
  _default.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtSwitch::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _expr.deps(reqNum);

  temp = _cases.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _default.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtFor::MuOptStmtFor(forstmt *f) :
  MuOptStmt(For), _node(f), _index(f->index), _body(f->body) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtFor::MuOptStmtFor(forstmt*)\n";
#endif
}

MuOptStmtFor::~MuOptStmtFor() {
}

void MuOptStmtFor::displayTree(ostream& out, uint indent) const {
```

```
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtFor::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "forstmt\n";
  indentLine(out, indent + 1);
  out << "index\n";
  _index.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "body\n";
  _body.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtFor::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _index.deps(reqNum);

  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtProc::MuOptStmtProc(proccall *p) :
  MuOptStmt(Proc), _node(p), _procedure(p->procedure),
  _actuals(p->actuals) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtProc::MuOptStmtProc(proccall*)\n";
#endif
}

MuOptStmtProc::~MuOptStmtProc() {
}

void MuOptStmtProc::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtProc::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "proccall\n";
  indentLine(out, indent + 1);
  out << "procedure\n";
  _procedure.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "actuals\n";
  _actuals.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtProc::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
```

```
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _procedure.deps(reqNum);

  temp = _actuals.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtClear::MuOptStmtClear(clearstmt *c) :
  MuOptStmt(Clear), _node(c), _target(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtClear::MuOptStmtClear(clearstmt*)\n";
#endif
  assert(c);
  _target = MuOptDesignator::newMuOptDesignator(c->target);
  assert(_target);
}

MuOptStmtClear::~MuOptStmtClear() {
  //delete _target;
}

void MuOptStmtClear::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtClear::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "clearstmt\n";
  indentLine(out, indent + 1);
  out << "target\n";
  _target->displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtClear::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _target->deps(reqNum);

  /*
  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;
  */

  return result;
}

MuOptStmtError::MuOptStmtError(errorstmt *c) :
```

```
  MuOptStmt(Error), _node(c), _string(c->string) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtError::MuOptStmtError(errorstmt*)\n";
#endif
}

MuOptStmtError::MuOptStmtError(errorstmt *c, bool isassert) :
  MuOptStmt(isassert ? Assert : Error), _node(c), _string(c->string) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtError::MuOptStmtError(errorstmt*, bool)\n";
#endif
}

MuOptStmtError::~MuOptStmtError() {
}

void MuOptStmtError::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtError::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "errorstmt \"" << _string << "\"\n";
}

ScopeSet *MuOptStmtError::deps(uint reqNum = 0) const {
  //assert(0);
  return new ScopeSet;
}

MuOptStmtAssert::MuOptStmtAssert(assertstmt *c) :
  MuOptStmtError(c, true), _node(c), _test(c->test){
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAssert::MuOptStmtAssert(assertstmt*)\n";
#endif
}

MuOptStmtAssert::~MuOptStmtAssert() {
}

void MuOptStmtAssert::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAssert::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "assertstmt\n";
  indentLine(out, indent + 1);
  out << "test\n";
  _test.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "string\n";
  indentLine(out, indent + 2);
  out << errstring() << "\n";
}

ScopeSet *MuOptStmtAssert::deps(uint reqNum = 0) const {
  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;
```

```
  return _test.deps(reqNum);
}

MuOptStmtPut::MuOptStmtPut(putstmt *c) :
  MuOptStmt(Put), _node(c) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtPut::MuOptStmtPut(putstmt*)\n";
#endif
    assert(0);
}

MuOptStmtPut::~MuOptStmtPut() {
}

void MuOptStmtPut::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtPut::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "putstmt\n";
}

ScopeSet *MuOptStmtPut::deps(uint reqNum = 0) const {
  assert(0);
  return NULL;
  /*
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = .deps(reqNum);

  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
  */
}

MuOptStmtAlias::MuOptStmtAlias(alias *c) :
  MuOptStmt(Alias), _node(c) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAlias::MuOptStmtAlias(alias*)\n";
#endif
    assert(0);
}

MuOptStmtAlias::~MuOptStmtAlias() {
}

void MuOptStmtAlias::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAlias::displayTree(int) const\n";
#endif
```

```
  indentLine(out, indent);
  out << "alias\n";
}

ScopeSet *MuOptStmtAlias::deps(uint reqNum = 0) const {
  assert(0);
  return NULL;
  /*
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = .deps(reqNum);

  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
  */
}

MuOptStmtAliasstmt::MuOptStmtAliasstmt(aliasstmt *a) :
  MuOptStmt(Aliasstmt), _node(a), _aliases(a->aliases), _body(a->body) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtAliasstmt::MuOptStmtAliasstmt(aliasstmt*)\n";
#endif
}

MuOptStmtAliasstmt::~MuOptStmtAliasstmt() {
}

void MuOptStmtAliasstmt::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptAliasstmt::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "aliasstmt\n";
  indentLine(out, indent + 1);
  out << "aliases\n";
  _aliases.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "body\n";
  _body.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtAliasstmt::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _aliases.deps(reqNum);
```

```
  temp = _body.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtReturn::MuOptStmtReturn(returnstmt *r) :
  MuOptStmt(Return), _node(r), _retexpr(r->retexpr) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtReturn::MuOptStmtReturn(returnstmt*)\n";
#endif
}

MuOptStmtReturn::~MuOptStmtReturn() {
}

void MuOptStmtReturn::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtReturn::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "returnstmt\n";
  indentLine(out, indent + 1);
  out << "retexpr\n";
  _retexpr.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtReturn::deps(uint reqNum = 0) const {
  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  return _retexpr.deps(reqNum);
}

MuOptStmtUndefine::MuOptStmtUndefine(undefinestmt *u) :
  MuOptStmt(Undefine), _node(u), _target(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtUndefine::MuOptStmtUndefine(undefinestmt*)\n";
#endif
  _target = MuOptDesignator::newMuOptDesignator(u ? u->target : NULL);
  assert(_target);
}

MuOptStmtUndefine::~MuOptStmtUndefine() {
}

void MuOptStmtUndefine::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtUndefine::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "undefinestmt\n";
  _target->displayTree(out, indent + 1);
}
```

```
ScopeSet *MuOptStmtUndefine::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _target->deps(reqNum);

  /*
  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;
  */

  return result;
}

MuOptStmtMultisetAdd::MuOptStmtMultisetAdd(multisetaddstmt *u) :
  MuOptStmt(MultisetAdd), _node(u), _element(NULL), _target(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtMultisetAdd::MuOptStmtMultisetAdd(multisetaddstmt*)\n";
#endif
  _element = MuOptDesignator::newMuOptDesignator(u ? u->element : NULL);
  assert(_element);
  _target = MuOptDesignator::newMuOptDesignator(u ? u->target : NULL);
  assert(_target);
}

MuOptStmtMultisetAdd::~MuOptStmtMultisetAdd() {
}

void MuOptStmtMultisetAdd::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptMultisetAdd::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "multisetaddstmt\n";
  indentLine(out, indent + 1);
  out << "element\n";
  _element->displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "target\n";
  _target->displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtMultisetAdd::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _element->deps(reqNum);

  temp = _target->deps(reqNum);
```

```
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}

MuOptStmtMultisetRemove::MuOptStmtMultisetRemove(multisetremovestmt *u) :
  MuOptStmt(MultisetRemove), _node(u), _index(u ? u->index : NULL),
  _target(NULL), _criterion(u ? u->criterion : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtMultisetRemove::"
    "MuOptStmtMultisetRemove(multisetremovestmt*)\n";
#endif
  _target = MuOptDesignator::newMuOptDesignator(u ? u->target : NULL);
  assert(_target);
}

MuOptStmtMultisetRemove::~MuOptStmtMultisetRemove() {
  //delete _target;
}

void MuOptStmtMultisetRemove::displayTree(ostream& out, uint indent) const{
#ifdef MUOPT_DEBUG
  cerr << "MuOptMultisetRemove::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "multisetremovestmt\n";
  indentLine(out, indent + 1);
  out << "index\n";
  _index.displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "target\n";
  _target->displayTree(out, indent + 2);
  indentLine(out, indent + 1);
  out << "criterion\n";
  _criterion.displayTree(out, indent + 2);
}

ScopeSet *MuOptStmtMultisetRemove::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _index.deps(reqNum);

  temp = _target->deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  temp = _criterion.deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
}
```

```
MuOptStmtNull::MuOptStmtNull() : MuOptStmt(Null) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtNull::MuOptStmtNull()\n";
#endif
}

MuOptStmtNull::~MuOptStmtNull() {
}

void MuOptStmtNull::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptStmtNull::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "null statement\n";
}

ScopeSet *MuOptStmtNull::deps(uint reqNum = 0) const {
  return new ScopeSet;
}
```

### A.2.9 mu_opt_typedecl.cc

```
// mu_opt_decl.cc    -*- c++ -*-

#include "mu_opt_decl.h"
#include "mu_opt_param.h"
#include "mu_opt_typedecl.h"


map<typedecl*,MuOptTypeDecl*> MuOptTypeDecl::_existing;


MuOptTypeDecl::~MuOptTypeDecl() {
}

void MuOptTypeDecl::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDecl::displayTree(int) const\n";
#endif
  indentLine(out, indent);
  out << "decl_class is Type \"" << name() << "\"\n";
}

MuOptTypeDecl *MuOptTypeDecl::newMuOptTypeDecl(typedecl *td) {
  MuOptTypeDecl *result = NULL;
  if(td)
    if(_existing.find(td) == _existing.end()) {
      switch(td->gettypeclass()) {
      case typedecl::Enum:
        result = new MuOptTypeDeclEnum(dynamic_cast<enumtypedecl*>(td));
        break;
      case typedecl::Range:
        result=new MuOptTypeDeclRange(dynamic_cast<subrangetypedecl*>(td));
        break;
      case typedecl::Array:
        result = new MuOptTypeDeclArray(dynamic_cast<arraytypedecl*>(td));
        break;
```

```
      case typedecl::MultiSet:
        result = new MuOptTypeDeclMultiSet(dynamic_cast<multisettypedecl*>
                                           (td));
        break;
      case typedecl::MultiSetID:
        result = new
          MuOptTypeDeclMultiSetID(dynamic_cast<multisetidtypedecl*>(td));
        break;
      case typedecl::Record:
        result=new MuOptTypeDeclRecord(dynamic_cast<recordtypedecl*>(td));
        break;
      case typedecl::Scalarset:
        result = new
          MuOptTypeDeclScalarset(dynamic_cast<scalarsettypedecl*>(td));
        break;
      case typedecl::Union:
        result = new MuOptTypeDeclUnion(dynamic_cast<uniontypedecl*>(td));
        break;
      case typedecl::Error_type:
        result = new MuOptTypeDeclError(dynamic_cast<errortypedecl*>(td));
        break;
      default:
        assert(0);
        break;
      }
      _existing[td] = result;
    }else {
      result = _existing[td];
#ifdef MUOPT_DEBUG
      cerr << "reissuing existing typedecl\n";
#endif
    }
  return result;
}


MuOptTypeDeclEnum::MuOptTypeDeclEnum(enumtypedecl *n)
  : MuOptTypeDecl(Enum, n ? n->name : NULL), _node(n),
    _left(n ? n->getleft() : -1), _right(n ? n->getright() : -1),
    _idvalues(n ? n->getidvalues() : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclEnum::MuOptTypeDeclEnum(enumtypedecl*)\n";
#endif
}

MuOptTypeDeclEnum::~MuOptTypeDeclEnum() {
}

void MuOptTypeDeclEnum::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclEnum::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is Enum\n";
  indentLine(out, indent + 2);
  out << "left " << _left << ", right " << _right << "\n";
  indentLine(out, indent + 2);
  out << "idvalues\n";
  _idvalues.displayTree(out, indent + 3);
```

```
}

ScopeSet *MuOptTypeDeclEnum::deps(uint reqNum = 0) const {
  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  return _idvalues.deps(reqNum);
}


MuOptTypeDeclRange::MuOptTypeDeclRange(subrangetypedecl *n)
  : MuOptTypeDecl(Range, n ? n->name : NULL), _node(n),
    _left(n ? n->getleft() : -1), _right(n ? n->getright() : -1) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclRange::MuOptTypeDeclRange(subrangetypedecl*)\n";
#endif
}

MuOptTypeDeclRange::~MuOptTypeDeclRange() {
}

void MuOptTypeDeclRange::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclRange::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is Range\n";
  indentLine(out, indent + 2);
  out << "left " << _left << ", right " << _right << "\n";
}

ScopeSet *MuOptTypeDeclRange::deps(uint reqNum = 0) const {
  return new ScopeSet;
}


MuOptTypeDeclArray::MuOptTypeDeclArray(arraytypedecl *n)
  : MuOptTypeDecl(Array, n ? n->name : NULL), _node(n),
    _indexType(NULL), _elementType(NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclArray::MuOptTypeDeclArray(arraytypedecl*)\n";
#endif
  if(n) {
    if(n->getindextype())
      _indexType = MuOptTypeDecl::newMuOptTypeDecl(n->getindextype());
    if(n->getelementtype())
      _elementType = MuOptTypeDecl::newMuOptTypeDecl(n->getelementtype());
  }
}

MuOptTypeDeclArray::~MuOptTypeDeclArray() {
  //delete _indexType;
  //delete _elementType;
}

void MuOptTypeDeclArray::displayTree(ostream& out, uint indent) const {
```

```
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclArray::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is Array\n";
  if(_indexType) {
    indentLine(out, indent + 1);
    out << "[index type]\n";
    _indexType->displayTree(out, indent + 2);
  }
  if(_elementType) {
    indentLine(out, indent + 1);
    out << "[element type]\n";
    _elementType->displayTree(out, indent + 2);
  }
}

ScopeSet *MuOptTypeDeclArray::deps(uint reqNum = 0) const {
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  if(_indexType)
    result = _indexType->deps(reqNum);
  else
    result = new ScopeSet;

  if(_elementType) {
    temp = _elementType->deps(reqNum);
    result->insert(temp->begin(), temp->end());
    delete temp;
  }

  return result;
}


MuOptTypeDeclMultiSet::MuOptTypeDeclMultiSet(multisettypedecl *n)
  : MuOptTypeDecl(MultiSet, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr <<
    "MuOptTypeDeclMultiSet::MuOptTypeDeclMultiSet(multisettypedecl*)\n";
#endif
  //assert(0);
}

MuOptTypeDeclMultiSet::~MuOptTypeDeclMultiSet() {
}

void MuOptTypeDeclMultiSet::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclMultiSet::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
```

```
  out << "typeclass is MultiSet\n";
}

ScopeSet *MuOptTypeDeclMultiSet::deps(uint reqNum = 0) const {
  //assert(0);
  return new ScopeSet;
  /*
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = .deps(reqNum);

  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
  */
}


MuOptTypeDeclMultiSetID::MuOptTypeDeclMultiSetID(multisetidtypedecl *n)
  : MuOptTypeDecl(MultiSetID, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclMultiSetID::"
    "MuOptTypeDeclMultiSetID(multisetidtypedecl*)\n";
#endif
  assert(0);
}

MuOptTypeDeclMultiSetID::~MuOptTypeDeclMultiSetID() {
}

void MuOptTypeDeclMultiSetID::displayTree(ostream& out, uint indent) const{
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclSetID::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is MultiSetID\n";
}

ScopeSet *MuOptTypeDeclMultiSetID::deps(uint reqNum = 0) const {
  assert(0);
  return NULL;
  /*
  ScopeSet *result, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = .deps(reqNum);
```

```
  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;

  return result;
  */
}


MuOptTypeDeclRecord::MuOptTypeDeclRecord(recordtypedecl *n)
  : MuOptTypeDecl(Record, n ? n->name : NULL), _node(n),
    _fields(n ? n->getfields() : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclRecord::MuOptTypeDeclRecord(recordtypedecl*)\n";
#endif
}

MuOptTypeDeclRecord::~MuOptTypeDeclRecord() {
}

void MuOptTypeDeclRecord::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclRecord::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is Record\n";
  indentLine(out, indent + 2);
  out << "fields\n";
  _fields.displayTree(out, indent + 3);
}

ScopeSet *MuOptTypeDeclRecord::deps(uint reqNum = 0) const {
  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  return _fields.deps(reqNum);
}


MuOptTypeDeclScalarset::MuOptTypeDeclScalarset(scalarsettypedecl *n)
  : MuOptTypeDecl(Scalarset, n ? n->name : NULL), _node(n),
    _left(n ? n->getleft() : -1), _right(n ? n->getright() : -1),
    _idvalues(n ? n->getidvalues() : NULL) {
#ifdef MUOPT_DEBUG
  cerr <<
    "MuOptTypeDeclScalarset::MuOptTypeDeclScalarset(scalarsettypedecl*)\n";
#endif
  //assert(!n->useless);
}

MuOptTypeDeclScalarset::~MuOptTypeDeclScalarset() {
}

void MuOptTypeDeclScalarset::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
```

```
  cerr << "MuOptTypeDeclScalarset::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is Scalarset\n";
  indentLine(out, indent + 2);
  out << "further dependencies ignored\n";
}

ScopeSet *MuOptTypeDeclScalarset::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
    return new ScopeSet;

  result = _idvalues.deps(reqNum);

  /*
  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;
  */

  return result;
}


MuOptTypeDeclUnion::MuOptTypeDeclUnion(uniontypedecl *n)
  : MuOptTypeDecl(Union, n ? n->name : NULL), _node(n),
    _unionmembers(n ? n->getunionmembers() : NULL) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclUnion::MuOptTypeDeclUnion(uniontypedecl*)\n";
#endif

}

MuOptTypeDeclUnion::~MuOptTypeDeclUnion() {
}

void MuOptTypeDeclUnion::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclUnion::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "typeclass is Union\n";
  indentLine(out, indent + 2);
  out << "further dependencies ignored\n";
}

ScopeSet *MuOptTypeDeclUnion::deps(uint reqNum = 0) const {
  ScopeSet *result;//, *temp;

  if(reqNum == 0)
    reqNum = nextReqNum();

  if(depLoop(reqNum))
```

```
    return new ScopeSet;

  result = _unionmembers.deps(reqNum);

  /*
  temp = .deps(reqNum);
  result->insert(temp->begin(), temp->end());
  delete temp;
  */

  return result;
}


MuOptTypeDeclError::MuOptTypeDeclError(errortypedecl *n)
  : MuOptTypeDecl(Error, n ? n->name : NULL), _node(n) {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclError::MuOptTypeDeclError(errortypedecl*)\n";
#endif
}

MuOptTypeDeclError::~MuOptTypeDeclError() {
}

void MuOptTypeDeclError::displayTree(ostream& out, uint indent) const {
#ifdef MUOPT_DEBUG
  cerr << "MuOptTypeDeclError::displayTree(int) const\n";
#endif
  MuOptTypeDecl::displayTree(out, indent);
  indentLine(out, indent + 1);
  out << "Error_type\n";
}

ScopeSet *MuOptTypeDeclError::deps(uint reqNum = 0) const {
  return new ScopeSet;
}
```

## A.3 Other Files

### A.3.1 Murphi3.1-muopt.patch

```
diff -ur Murphi.orig/include/mu_verifier.h Murphi3.1/include/mu_verifier.h
--- Murphi.orig/include/mu_verifier.h   Thu Jan 28 23:49:11 1999
+++ Murphi3.1/include/mu_verifier.h     Mon Aug  2 13:05:26 1999
@@ -95,7 +95,6 @@
 #define BITS(type) ((int)sizeof (type) * CHAR_BIT)
 #endif

-typedef char bool;
 typedef void (*proc) (void);
 typedef bool (*boolfunc) (void);

diff -ur Murphi.orig/src/Makefile Murphi3.1/src/Makefile
--- Murphi.orig/src/Makefile    Thu Jan 28 23:49:11 1999
+++ Murphi3.1/src/Makefile      Thu Aug 19 13:00:03 1999
@@ -38,20 +38,23 @@
```

```
 #CPLUSPLUS = xlC -+ -Q!

 # choice of the options
-CFLAGS = -O4                              # options for g++
+CFLAGS = -g -O4 -I. -I../../muopt              # options for g++
 #CFLAGS = -O4 -pipe -static #
 #CFLAGS = -static #
 #CFLAGS = -w #                              # options for OCC
 #CFLAGS = -w -O3 #                          # options for DCC

+LDFLAGS = -L../../muopt
+LIBS = -lmuopt
+
 OBJS =         \
        cpp_code.o \
        cpp_code_as.o \
        lex.yy.o \
        y.tab.o \
-       cpp_sym.o \
-       cpp_sym_aux.o \
-       cpp_sym_decl.o \
+       cpp_sym.o \
+       cpp_sym_aux.o \
+       cpp_sym_decl.o \
        decl.o \
        error.o \
        expr.o \
@@ -65,8 +68,8 @@
 ########################################
 # Makefile grammar

-mu: $(OBJS)
-       $(CPLUSPLUS) -o mu $(CFLAGS) $(OBJS)
+mu: $(OBJS) ../../muopt/libmuopt.a
+       $(CPLUSPLUS) -o mu $(CFLAGS) $(OBJS) $(LDFLAGS) $(LIBS)

 mu.o: mu.C mu.h
        $(CPLUSPLUS) -c $(CFLAGS) mu.C
diff -ur Murphi.orig/src/expr.h Murphi3.1/src/expr.h
--- Murphi.orig/src/expr.h      Fri Jan 29 00:28:10 1999
+++ Murphi3.1/src/expr.h        Wed Sep 15 17:53:52 1999
@@ -323,6 +323,9 @@
   ********************/
 class designator: public expr
 {
+  friend MuOptDesignator;
+  friend MuOptDesignatorInstance;
+  friend MuOptDesignatorList;
   // variables
   designator * left;
   ste * origin;    /* an identifier; */
diff -ur Murphi.orig/src/mu.C Murphi3.1/src/mu.C
--- Murphi.orig/src/mu.C        Fri Jan 29 00:28:10 1999
+++ Murphi3.1/src/mu.C  Thu Aug  5 13:56:01 1999
@@ -63,6 +63,7 @@
  */

 #include "mu.h"
+#include <mu_opt.h>
 #include <string.h>
```

```
 #include <new.h>

@@ -442,6 +443,8 @@
   error = yyparse();
   if ( !error && Error.getnumerrors() == 0 )
     {
+      MuOptRoot muo(theprog);
+      muo.optimize();
       codefile = setup_outfile( args->filename );
       theprog->generate_code();
       fclose(codefile);
diff -ur Murphi.orig/src/mu.h Murphi3.1/src/mu.h
--- Murphi.orig/src/mu.h        Fri Jan 29 13:56:53 1999
+++ Murphi3.1/src/mu.h  Wed Sep 15 17:54:14 1999
@@ -110,7 +110,6 @@
 /********************
   boolean constants
   ********************/
-typedef char bool; /* just for my sake. */
 #define TRUE 1
 #define FALSE 0

@@ -388,6 +387,7 @@
 {
   stelist *first;
   stelist *last;
+  friend class MuOptSTEChain;
 public:
   stecoll(ste *s) {
     first = last = new stelist (s, NULL); }
@@ -469,7 +471,7 @@
          char * bodyname,
          bool unfair)
   : rulename(rulename), conditionname(conditionname), bodyname(bodyname),
-    next(NULL), unfair(unfair) {};
+    unfair(unfair), next(NULL) {};
   int print_rules(); /* print out a list of rulerecs. */
 };

@@ -491,8 +493,9 @@
          char * rightconditionname,
          live_type livetype)
   : rulename(rulename), preconditionname(preconditionname),
-    livetype(livetype), leftconditionname(leftconditionname),
-    rightconditionname(rightconditionname), next(NULL) {};
+    leftconditionname(leftconditionname),
+    rightconditionname(rightconditionname),
+    livetype(livetype), next(NULL) {};
   int print_liverules();
 };

diff -ur Murphi.orig/src/rule.C Murphi3.1/src/rule.C
--- Murphi.orig/src/rule.C      Mon Mar 22 22:17:50 1999
+++ Murphi3.1/src/rule.C        Thu Sep 16 14:51:03 1999
@@ -96,7 +96,24 @@
   // mentioned in the condition go first
   rearrange_enclosures();

+  copy_enclosures();
 }
```

```
+
+void simplerule::copy_enclosures() {
+  ste *in = enclosures, *last = NULL;
+  enclosures = NULL;
+  while(in) {
+    ste *copy = new ste(*in);
+    copy->next = NULL;
+    if(last)
+      last->next = copy;
+    else
+      enclosures = copy;
+    last = copy;
+    in = in->next;
+  }
+}
+

 simplerule * simplerule::SimpleRuleList = NULL;

diff -ur Murphi.orig/src/rule.h Murphi3.1/src/rule.h
--- Murphi.orig/src/rule.h      Fri Jan 29 00:28:10 1999
+++ Murphi3.1/src/rule.h        Thu Sep 16 14:51:23 1999
@@ -138,9 +138,10 @@
   // code generation
   virtual char *generate_code();

-  CountSize(ste * enclosures);
+  int CountSize(ste * enclosures);

   void rearrange_enclosures();
+  void copy_enclosures();

   int getsize() { return size; }
 };
```

### A.3.2 Makefile

```
CXX = g++
CXXFLAGS = -Wall -I. -I../Murphi3.1/src -I../src -g
LD = g++ -shared -fpic
LDFLAGS =
LIBS =
RANLIB = ranlib

OFILES = mu_opt_base.o mu_opt_decl.o mu_opt_expr.o mu_opt_param.o \
         mu_opt_root.o mu_opt_rule.o mu_opt_ste.o mu_opt_stmt.o \
         mu_opt_typedecl.o
TARGET = libmuopt.a

all: $(TARGET) install

tests: all

libmuopt.a: $(OFILES)
        rm -f $@
        ar cr $@ $^
        $(RANLIB) $@
```

```
libmuopt.so: $(OFILES)
	rm -f $@
	$(LD) $^ $(LDFLAGS) $(LIBS) -o $@

testprog: testprog.o
	$(CXX) $^ $(LDFLAGS) -lmuopt $(LIBS) -o $@


mu_opt_base.o: mu_opt_base.cc mu_opt_base.h
mu_opt_decl.o: mu_opt_decl.cc mu_opt_decl.h mu_opt_param.h \
               mu_opt_typedecl.h
mu_opt_expr.o: mu_opt_expr.cc mu_opt_expr.h mu_opt_base.h
mu_opt_param.o: mu_opt_param.cc mu_opt_param.h
mu_opt_root.o: mu_opt_root.cc mu_opt_root.h
mu_opt_rule.o: mu_opt_rule.cc mu_opt_rule.h
mu_opt_ste.o: mu_opt_ste.cc mu_opt_ste.h mu_opt_decl.h
mu_opt_stmt.o: mu_opt_stmt.cc mu_opt_stmt.h
mu_opt_typedecl.o: mu_opt_typedecl.cc mu_opt_typedecl.h


%.h:
	touch $@

mu_opt.h: mu_opt_base.h mu_opt_decl.h mu_opt_expr.h mu_opt_root.h \
          mu_opt_rule.h mu_opt_ste.h mu_opt_stmt.h
mu_opt_decl.h: mu_opt_base.h mu_opt_ste.h mu_opt_stmt.h
mu_opt_expr.h: mu_opt_base.h mu_opt_ste.h
mu_opt_param.h: mu_opt_decl.h
mu_opt_root.h: mu_opt_base.h mu_opt_ste.h mu_opt_rule.h
mu_opt_rule.h: mu_opt_base.h mu_opt_ste.h mu_opt_expr.h mu_opt_stmt.h
mu_opt_ste.h: mu_opt_base.h
mu_opt_stmt.h: mu_opt_base.h mu_opt_expr.h
mu_opt_typedecl.h: mu_opt_decl.h


install: $(TARGET)
#	cp $(TARGET) ${exec_prefix}/lib
#	cp $(TARGET) ../lib

clean:
	rm -f *~ *.o a.out core

veryclean: clean
	rm -f libmuopt.a libmuopt.so testprog
```

# Appendix B

# Firewire Model Source

## B.1 Novice Firewire Model

This partial model of IEEE 1394 [13] is the one tested in Chapter 5. It was written Fall 1998, by the author, at the beginning of his graduate studies. At that time the author was a Mur$\varphi$ novice. The model is a part of the cable physical layer [12], specifically the bus reset and tree identification state machines.

```
-- 1394.m
--
-- $Id: 1394.m,v 1.10 1998/11/25 17:32:54 bwinters Exp $
--
-- Brian Winters
--
-- Revision history:
--   11/24/98   Worked out startup problems.  Able to reach all states up
--              to S0 (all I've implemented so far).  Statespace blows up
--              doing full evaluation of that statespace though.
--   11/22/98   Have the standard.  Implementing PHY from their code.
--   11/10/98   Started using CVS (I'm sleeping better already).
--              Implementing cables between nodes as a resolution function
--                of the outputs of those two nodes.
--   11/03/98   Initial version.  Linear arrangement of N nodes.

const
  N: 2;             -- number of nodes
  HIGHPORT: 1;      -- highest numbered port possible per node
--HIGHFIFO_DEPTH: 0; -- highest numbered entry in a FIFO
```

```
  TIMER_MAX: 4;

  -- table 4-32
  CONFIG_TIMEOUT:     3; -- 5; -- 14; -- defined as 16384
  FORCE_ROOT_TIMEOUT: 2; -- 4; -- 13; -- defined as >=8192 and
                                      --   <= CONFIG_TIMEOUT
  ROOT_CONTEND_FAST:  1; -- 1; -- 2; -- defined as 24
  ROOT_CONTEND_SLOW:  2; -- 2; -- 3; -- defined as 56
  RESET_TIME:         3; -- 5; -- 14; -- defined as 16384
  RESET_WAIT:         4; -- 6; -- 15; -- defined as 16400
  MAX_ARB_STATE_TIME: RESET_WAIT;

  -- semi-contrived, see table 4-44
  GAP_COUNT_MAX:      2; -- 3; -- 4; -- defined as 63



type
  IDnum: 0..(N-1);
  Portnum: 0..HIGHPORT;
--  FIFOnum: 0..HIGHFIFO_DEPTH;
  irange: 0..(HIGHPORT+1);

  PHY_state: enum { R0, R1,
                    T0, T1, T2, T3,
                    S0, S1, S2, S3, S4,
                    A0, A1, A2, A3, A4, A5, A6 };

  -- bogus types
  dataBit: boolean;
  baserate_time: 0..TIMER_MAX;
  timer: baserate_time;
  gap_counter: 0..GAP_COUNT_MAX;

  -- 4.4 Cable PHY operation, table 4-37
  phyData: enum { L, H, Z };
  speedCode: enum { S100, S200, S400 };
  portData :
    record
      TPA: phyData;
      TPB: phyData;
    end;


  -- 4.3.3: transmitted line states, table 4-27
  ArbLineStateTX: enum { TX_IDLE, TX_REQUEST, TX_GRANT, TX_PARENT_NOTIFY,
                         TX_DATA_PREFIX, TX_CHILD_NOTIFY, TX_IDENT_DONE,
                         TX_DATA_END, TX_BUS_RESET };
  -- 4.3.3: received line states, table 4-28
  ArbLineStateRX: enum { RX_IDLE, RX_PARENT_NOTIFY, RX_REQUEST_CANCEL,
                         RX_IDENT_DONE, RX_SELF_ID_GRANT, RX_REQUEST,
                         RX_ROOT_CONTENTION, RX_GRANT, RX_PARENT_HANDSHAKE,
                         RX_DATA_END, RX_CHILD_HANDSHAKE, RX_DATA_PREFIX,
                         RX_BUS_RESET };



-- PORT definition

  port:
```

```
    record
-- 4.3.9 Port variables
      child: boolean;
      connected: boolean;
      child_ID_complete: boolean;
--    max_peer_speed: MaxSpeed; --NYM
--    speed_OK: boolean; --NYM

      data: portData; -- this one not explicitly defined, guessing here
    end;


-- NODE definition

  node:
    record

-- 4.3.7 Cable PHY node constants
      NPORT: irange;
--    PHY_DELAY_CATEGORY: 0(..3); -- this can only be 0 in current spec
--    PHY_SPEED: MaxSpeed; -- NYM
--    POWER_CLASS: 0..7; -- don't care about physical power characteristics
--    CONTENDER: boolean;

-- 4.3.8 Node variables, table 4-35
      arb_enable: boolean;
--    cable_power_active: boolean;
      force_root: boolean;
      gap_count: gap_counter; -- guessing on variable type here
      initiated_reset: boolean;
      link_active: boolean;
--    more_packets: boolean;
      parent_port: Portnum;
      physical_ID: IDnum;
      receive_port: Portnum;
      root: boolean;

-- 4.4 Cable PHY operation, table 4-37, implied internal variables
--    fifo_rd_ptr: FIFOnum;
--    fifo_wr_ptr: FIFOnum;
--    FIFO_DEPTH: FIFOnum;
--    FIFO: array[FIFOnum] of dataBit;
--    waiting_for_data_start: boolean;
--    rx_speed: speedCode;

      arb_timer: timer;
--    root_test: boolean;
      contend_time: baserate_time;
      child_count: irange;
      lowest_unidentified_child: Portnum;
      all_child_ports_identified: boolean;
      self_ID_complete: boolean;
--    requesting_child: Portnum;
--    force_root: boolean; -- see above
--    end_of_reception: boolean;
--    link_req_active: boolean;
--    arb_enable: boolean; -- see above
      old_connect: array[Portnum] of boolean; -- actually array of NPORT
      bus_initialize_active: boolean;
      gap_count_reset_disable: boolean;
```

```
-- variables to emulate the physical aspects of a node
      ports: array[Portnum] of port;

-- current state of the node
      Pstate: PHY_state;
    end;


var
  nodes: array[IDnum] of node;


-- FUNCTIONS


-- mapping functions, to model the physical wiring

-- assumes N=2
function trans_good(n: IDnum; port_number: Portnum): boolean;
begin
  return ((n=0 & port_number=1) | (n=1 & port_number=0));
end;

-- assumes N=3
--function trans_good(n: IDnum; port_number: Portnum): boolean;
--begin
--  assert (N=3 & HIGHPORT=1) "N and/or HIGHPORT do not match";
--  return ((n=0 & port_number=1) | (n=1) | (n=2 & port_number=0));
--end;

-- assumes N=2
function trans_node(n: IDnum; port_number: Portnum): IDnum;
begin
  if(n=0 & port_number=1) then
    return 1;
  elsif(n=1 & port_number=0) then
    return 0;
  end;
end;

-- assumes N=3, HIGHPORT=1
--function trans_node(n: IDnum; port_number: Portnum): IDnum;
--begin
--  assert (N=3 & HIGHPORT=1) "N and/or HIGHPORT do not match";
--  if(n=0 & port_number=1) then
--    return 1;
--  elsif(n=1 & port_number=0) then
--    return 0;
--  elsif(n=1 & port_number=1) then
--    return 2;
--  elsif(n=2 & port_number=0) then
--    return 1;
--  else
--    error "Invalid node/port combination";
--  end;
--end;

-- assumes N=2
function trans_port(n: IDnum; port_number: Portnum): Portnum;
```

```
begin
  if(n=0 & port_number=1) then
    return 0;
  elsif(n=1 & port_number=0) then
    return 1;
  end;
end;

-- assumes N=3, HIGHPORT=1
--function trans_port(n: IDnum; port_number: Portnum): Portnum;
--begin
--  assert (N=3 & HIGHPORT=1) "N and/or HIGHPORT do not match";
--  if(n=0 & port_number=1) then
--    return 0;
--  elsif(n=1 & port_number=0) then
--    return 1;
--  elsif(n=1 & port_number=1) then
--    return 0;
--  elsif(n=2 & port_number=0) then
--    return 1;
--  else
--    error "Invalid node/port combination";
--  end;
--end;



-- hardware functions, prototyped from table 4-37

--function random_bool(): boolean;
--begin
---- pseudorandom - will protocol work even with bad randomizer?
--  return true;
--  return false;
----  return random_value;
--end;

--procedure portT(n: IDnum; port_number: Portnum; txData: portData);
procedure portT(n: IDnum; port_number: Portnum; txData: ArbLineStateTX);
begin
  alias p: nodes[n].ports[port_number].data do -- from table 4-27
    switch txData
    case TX_IDLE:
      p.TPA := Z;
      p.TPB := Z;
    case TX_REQUEST:
      p.TPA := Z;
      p.TPB := L;
    case TX_GRANT:
      p.TPA := Z;
      p.TPB := L;
    case TX_PARENT_NOTIFY:
      p.TPA := L;
      p.TPB := Z;
    case TX_DATA_PREFIX:
      p.TPA := L;
      p.TPB := H;
    case TX_CHILD_NOTIFY:
      p.TPA := H;
```

```
      p.TPB := Z;
    case TX_IDENT_DONE:
      p.TPA := H;
      p.TPB := Z;
    case TX_DATA_END:
      p.TPA := H;
      p.TPB := L;
    case TX_BUS_RESET:
      p.TPA := H;
      p.TPB := H;
    end;
  end;
end;

function resolve(a, b: phyData): phyData;
begin
  if(a=Z) then return b; end;
  if(b=Z | a=b) then return a; end;
  return Z;
end;

--function portR(n, port_number: IDnum): portData;
function portR(n, port_number: IDnum): ArbLineStateRX;
var Tn, Tp: IDnum;
    otherA, otherB: phyData;
begin
  otherA := Z;
  otherB := Z;
  if(trans_good(n, port_number)) then
    Tn := trans_node(n, port_number);
    Tp := trans_port(n, port_number);
    otherA := nodes[Tn].ports[Tp].data.TPA;
    otherB := nodes[Tn].ports[Tp].data.TPB;
  end;
  alias pt: nodes[n].ports[port_number].data do -- from table 4-28
    if(resolve(pt.TPA, otherB) = Z & resolve(pt.TPB, otherA) = Z) then
      return RX_IDLE;
    elsif(resolve(pt.TPA, otherB) = Z & resolve(pt.TPB, otherA) = L) then
      return RX_PARENT_NOTIFY; -- also RX_REQUEST_CANCEL
    elsif(resolve(pt.TPA, otherB) = Z & resolve(pt.TPB, otherA) = H) then
      return RX_IDENT_DONE;
    elsif(resolve(pt.TPA, otherB) = L & resolve(pt.TPB, otherA) = Z) then
      return RX_SELF_ID_GRANT; -- also RX_REQUEST
    elsif(resolve(pt.TPA, otherB) = L & resolve(pt.TPB, otherA) = L) then
      return RX_ROOT_CONTENTION; -- also RX_GRANT
    elsif(resolve(pt.TPA, otherB) = L & resolve(pt.TPB, otherA) = H) then
      return RX_PARENT_HANDSHAKE; -- also RX_DATA_END
    elsif(resolve(pt.TPA, otherB) = H & resolve(pt.TPB, otherA) = Z) then
      return RX_CHILD_HANDSHAKE;
    elsif(resolve(pt.TPA, otherB) = H & resolve(pt.TPB, otherA) = L) then
      return RX_DATA_PREFIX;
    elsif(resolve(pt.TPA, otherB) = H & resolve(pt.TPB, otherA) = H) then
      return RX_BUS_RESET;
    end;
  end;
end;

function portRcomp(n, port_number: IDnum; c: ArbLineStateRX): boolean;
var r: ArbLineStateRX;
begin
```

```
  r := portR(n, port_number);
  switch r
  case RX_PARENT_NOTIFY:
    if(c = RX_REQUEST_CANCEL) then
      return true;
    end;
  case RX_SELF_ID_GRANT:
    if(c = RX_REQUEST) then
      return true;
    end;
  case RX_ROOT_CONTENTION:
    if(c = RX_GRANT) then
      return true;
    end;
  case RX_PARENT_HANDSHAKE:
    if(c = RX_DATA_END) then
      return true;
    end;
  end;
  return (r = c);
end;



--procedure portTspeed(n, port_number: IDnum; speed: speedCode);
--function portRspeed(n, port_number: IDnum): speedCode;


-- 4.4.2.1.2 Bus reset actions and conditions

function bus_reset_signal_received(n: IDnum): boolean; -- table 4-44
var i: irange;
begin
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].connected & portRcomp(n, i, RX_BUS_RESET)) then
      return true;
    end;
    i:=i+1;
  end;
  return false;
end;

function connection_state_change(n: IDnum): boolean; -- table 4-44
var state_change: boolean; i: irange;
begin
  state_change := false;
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].connected != nodes[n].old_connect[i]) then
      state_change := true;
      nodes[n].old_connect[i] := !nodes[n].old_connect[i];
    end;
    i:=i+1;
  end;
  return state_change;
end;

procedure reset_start_actions(n: IDnum); -- table 4-44
var i: irange;
```

```
begin
  nodes[n].root := false;
  nodes[n].physical_ID := 0;
  nodes[n].child_count := 0;
  nodes[n].bus_initialize_active := true;
  if(nodes[n].gap_count_reset_disable) then
    nodes[n].gap_count_reset_disable := false;
  else
    nodes[n].gap_count := GAP_COUNT_MAX; --hard coded to 0x3F in table 4-44
  end;
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].connected) then
      portT(n, i, TX_BUS_RESET);
    else
      portT(n, i, TX_IDLE);
    end;
    nodes[n].ports[i].child := false;
    nodes[n].ports[i].child_ID_complete := false;
    i:=i+1;
  end;
  nodes[n].arb_timer := 0;
end;

procedure reset_wait_actions(n: IDnum); -- table 4-44
var i: irange;
begin
  i:=0;
  while(i < nodes[n].NPORT) do
    portT(n, i, TX_IDLE);
    i:=i+1;
  end;
  nodes[n].arb_timer := 0;
end;

function reset_complete(n: IDnum): boolean; -- table 4-44
var i: irange;
begin
  i:=0;
  while(i < nodes[n].NPORT) do
    if(!portRcomp(n, i, RX_IDLE) &
       !portRcomp(n, i, RX_PARENT_NOTIFY) &
       nodes[n].ports[i].connected) then
      return false;
    end;
    i:=i+1;
  end;
  return true;
end;

function arb_state_timeout(n: IDnum): boolean; -- support for figure 4-22
begin alias nd: nodes[n] do alias st: nd.Pstate do
  return ((st != R1) & (st != A0) &
          (st != A5) & (st != A6) &
          (nd.arb_timer >= MAX_ARB_STATE_TIME));
end; end; end;


-- 4.4.2.2.2 Tree-ID actions and conditions
```

```
procedure tree_ID_start_actions_interp_1(n: IDnum); -- table 4-45
var i, temp_count: irange;
begin
  nodes[n].arb_timer := 0;
  -- LOOP FOREVER???  EXCUSE ME????

  -- my suspicion here is that they intend that this be going on as some
  -- sort of background process that is always updating child count, but
  -- on second thought that doesn't make sense, because we immediately go
  -- into a reset state if our connections change

--  while(true) do
    temp_count := 0;
    i:=0;
    while(i<nodes[n].NPORT) do
-- formatting of standard is ambiguous here; using C code and not indention
-- my suspicion is that this interpretation is WRONG, but we'll see
      if(!nodes[n].ports[i].connected |
         portRcomp(n, i, RX_PARENT_NOTIFY)) then
        nodes[n].ports[i].child := true;
        temp_count := temp_count + 1;
        nodes[n].child_count := temp_count;
      end;
      i:=i+1;
    end;
--  end;
end;

procedure tree_ID_start_actions_interp_2(n: IDnum); -- table 4-45
var i, temp_count: irange;
begin
  nodes[n].arb_timer := 0;
  temp_count := 0;
  i:=0;
  while(i<nodes[n].NPORT) do
-- formatting of standard is ambiguous here; using indention and not C code
    if(!nodes[n].ports[i].connected |
       portRcomp(n, i, RX_PARENT_NOTIFY)) then
      nodes[n].ports[i].child := true;
      temp_count := temp_count + 1;
    end;
    i:=i+1;
  end;
  nodes[n].child_count := temp_count;
--  end;
end;

procedure tree_ID_start_actions(n: IDnum); -- table 4-45
begin
--  tree_ID_start_actions_interp_1(n);
  tree_ID_start_actions_interp_2(n);
end;

procedure child_handshake_actions(n: IDnum); -- table 4-45
var i: irange;
begin
  nodes[n].root := true;
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].connected) then
```

```
      if(nodes[n].ports[i].child) then
        portT(n, i, TX_CHILD_NOTIFY);
      else
        portT(n, i, TX_PARENT_NOTIFY);
        nodes[n].parent_port := i;
        nodes[n].root := false;
      end;
    end;
    i:=i+1;
  end;
end;

function child_handshake_complete(n: IDnum): boolean; -- table 4-45
var i: irange;
begin
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].child &
       nodes[n].ports[i].connected &
       !portRcomp(n, i, RX_CHILD_HANDSHAKE)) then
      return false;
    end;
    i:=i+1;
  end;
  return true;
end;

procedure root_contend_actions(n: IDnum; r: boolean); -- table 4-45
var i: irange;
begin
--  if(random_bool()) then
  if(r) then
    nodes[n].contend_time := ROOT_CONTEND_SLOW;
  else
    nodes[n].contend_time := ROOT_CONTEND_FAST;
  end;
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].child) then
      portT(n, i, TX_CHILD_NOTIFY);
    else
      portT(n, i, TX_IDLE);
    end;
    i:=i+1;
  end;
  nodes[n].arb_timer := 0;
end;

procedure self_ID_start_actions(n: IDnum); -- table 4-46
var i: irange;
begin
  nodes[n].all_child_ports_identified := true;
  i:=0;
  while(i < nodes[n].NPORT) do
    if(nodes[n].ports[i].child_ID_complete) then
      portT(n, i, TX_DATA_PREFIX);
    else
      portT(n, i, TX_IDLE);
      if(nodes[n].ports[i].child & nodes[n].ports[i].connected) then
        if(nodes[n].all_child_ports_identified) then
```

```
          nodes[n].lowest_unidentified_child := i;
        end;
        nodes[n].all_child_ports_identified := false;
      end;
    end;
    i:=i+1;
  end;
end;

procedure self_ID_grant_actions(n: IDnum); -- table 4-46
var i: irange;
begin
  i:=0;
  while(i<nodes[n].NPORT) do
    if(!nodes[n].all_child_ports_identified &
       (i = nodes[n].lowest_unidentified_child)) then
      portT(n, i, TX_GRANT);
    else
      portT(n, i, TX_DATA_PREFIX);
    end;
    i:=i+1;
  end;
end;

procedure self_ID_receive_actions(n: IDnum); -- table 4-46
var i: irange;
begin
  portT(n, nodes[n].receive_port, TX_IDLE);
--  -- receive_actions(n);
  nodes[n].physical_ID := nodes[n].physical_ID + 1;
  i:=0;
  while(i<nodes[n].NPORT) do
    portT(n, i, TX_IDLE);
    i:=i+1;
  end;
end;

--procedure self_ID_transmit_actions(n: IDnum); -- table 4-46
--var last_SID_packet, SID_pkt_number, port_number: irange;
--begin
--  last_SID_packet := 1; -- (nodes[n].NPORT + 4) / 8;
--  nodes[n].self_ID_complete := false;
--  nodes[n].receive_port := nodes[n].NPORT;
---- -- start_tx_packet(n, S100);
--  SID_pkt_number := 0;
--  while(SID_pkt_number <= last_SID_pkt) begin
--    -- abstracting away SID packet construction
--    while(port_number < ((SID_pkt_number + 1)*8 - 5)) do
--      -- abstracting away more SID packet construction
--      port_number:=port_number+1;
--    end;
--    -- abstracting away even more SID packet construction
--    if(SID_pkt_number = last_SID_pkt) then
--      -- don't send quadlets
---- --      stop_tx_packet(n, TX_DATA_END, S100);
--    else
---- --      stop_tx_packet(n, TX_DATA_PREFIX, S100);
--    end;
--    SID_pkt_number:=SID_pkt_number+1;
--  end;
```

```
--  port_number:=0;
--  while(port_number < nodes[n].NPORT) do
--    if(port_number == parent_port) then
--      portT(n, port_number, TX_IDENT_DONE);
--      -- portTspeed();
--      -- wait_time();
--      -- portTspeed();
--      -- PH_EVENT.ind(SELF_ID_COMPLETE, physical_ID, root);
--    else
--      portT(n, port_number, TX_IDLE);
--    end;
--    port_number:=port_number+1; -- bad bug in standard I think
--  end;
--  self_ID_complete := true;
--end;


-- RULESETS

-- random value hack
ruleset r: boolean do

-- state machine by node

ruleset n: IDnum do
  alias nd: nodes[n] do
  alias st: nd.Pstate do
  rule "increment timer (modeling hack)"
    nd.arb_timer < TIMER_MAX ==>
      begin
        nd.arb_timer := nd.arb_timer + 1;
      end;

  rule "Transition All:R0a (4.4.2.1.1)"
    bus_reset_signal_received(n) ==>
      begin
        st := R0;
        reset_start_actions(n); -- 4.4.2.1
--        -- PH_STATE.ind(BUS_RESET_START) (4.4.2.1)
        nd.initiated_reset := false;
      end;
  rule "Transition All:R0b (4.4.2.1.1)"
    (
--   PH_CONT.req(bus_reset) |
     connection_state_change(n) | arb_state_timeout(n)) ==>
      begin
        st := R0;
--      -- PH_STATE.ind(BUS_RESET_START) (4.4.2.1)
        nd.initiated_reset := true;
      end;
  rule "Transition R0:R1 (4.4.2.1.1)"
    ((st = R0) & (nd.arb_timer >= RESET_TIME)) ==>
      begin
        st := R1;
        reset_wait_actions(n); -- 4.4.2.1
      end;
  rule "Transition R1:R0 (4.4.2.1.1)"
    ((st = R1) & (nd.arb_timer >= RESET_WAIT)) ==>
      begin
        st := R0;
```

```
        reset_start_actions(n); -- 4.4.2.1
      end;
  rule "Transition R1:T0 (4.4.2.1.1)"
    ((st = R1) & reset_complete(n)) ==>
      begin
        st := T0;
        tree_ID_start_actions(n); -- 4.4.2.2
      end;
  rule "Transition T0:T0 (4.4.2.2.1)"
    ((st = T0) & ((nd.arb_timer >= CONFIG_TIMEOUT) &
                  (nd.child_count < (nd.NPORT - 1)))) ==>
      begin
        st := T0;
--        -- signal PH_STATE.ind(CONFIG_TIMEOUT) (4.4.2.2)
        tree_ID_start_actions(n); -- 4.4.2.2
      end;
  rule "Transition T0:T1 (4.4.2.2.1)"
    ((st = T0) & ((nd.child_count = nd.NPORT) |
                  (((nd.force_root & (nd.arb_timer >= FORCE_ROOT_TIMEOUT))|
                    (!nd.force_root & (nd.arb_timer >= 0))) &
                   (nd.child_count = (nd.NPORT - 1))))) ==>
      begin
        st := T1;
        child_handshake_actions(n); -- 4.4.2.2
      end;
  rule "Transition T1:T2 (4.4.2.2.1)"
    ((st = T1) & child_handshake_complete(n)) ==>
      begin
        st := T2;
      end;
  rule "Transition T2:S0 (4.4.2.2.1)"
    ((st = T2) & (nd.root |
                  portRcomp(n, nd.parent_port, RX_PARENT_HANDSHAKE))) ==>
      begin
        st := S0;
        self_ID_start_actions(n);
      end;
  rule "Transition T2:T3 (4.4.2.2.1)"
    ((st = T2) & portRcomp(n, nd.parent_port, RX_ROOT_CONTENTION)) ==>
      begin
        st := T3;
        root_contend_actions(n, r); -- 4.4.2.2
      end;
  rule "Transition T3:T2 (4.4.2.2.1)"
    ((st = T3) & (portRcomp(n, nd.parent_port, RX_IDLE) &
                  (nd.arb_timer > nd.contend_time))) ==>
      begin
        st := T2;
        portT(n, nd.parent_port, TX_PARENT_NOTIFY); -- 4.4.2.2
      end;
  rule "Transition T3:T1 (4.4.2.2.1)"
    ((st = T3) & (portRcomp(n, nd.parent_port, RX_PARENT_NOTIFY) &
                  (nd.arb_timer > nd.contend_time))) ==>
      begin
        st := T1;
        -- 4.4.2.2: "node is now root"
        nd.ports[nd.parent_port].child := true;
      end;

  end;
```

```
  end;
end; -- state machine by node

end; -- random value

-- INITIALIZATION CODE

startstate
  begin
    for n: IDnum do alias nd: nodes[n] do

-- 4.3.7 Cable PHY node constants
      nd.NPORT := HIGHPORT + 1;
--    nd.PHY_DELAY_CATEGORY := 0; -- this can only be 0 in current spec
--    nd.PHY_SPEED := S100; -- NYM
--    nd.POWER_CLASS := 0; -- don't care about phys power characteristics
--    nd.CONTENDER := false; -- don't want to model resource manager role

-- 4.3.8 Node variables, table 4-35
      nd.arb_enable := false;
--    nd.cable_power_active := true;
      nd.force_root := false; -- not going to model this yet
--    nd.gap_count := ; -- not sure what variable type this should be
      nd.initiated_reset := false; -- should add some rules to allow a
                                   -- node to choose to initiate reset at
                                   -- any time
      nd.link_active := false;
--    nd.more_packets := false;
      nd.parent_port := 0;
      nd.physical_ID := 0;
      nd.receive_port := 0;
      nd.root := false;

-- 4.4 Cable PHY operation, table 4-37, implied internal variables
--    nd.fifo_rd_ptr := 0;
--    nd.fifo_wr_ptr := 0;
--    nd.FIFO_DEPTH := HIGHFIFO_DEPTH;
--    for i: FIFOnum do
--      nd.FIFO[i] := false; -- just makin' it up as I go along
--    end;
--    nd.waiting_for_data_start := true;
--    nd.rx_speed := S100;

      nd.arb_timer := 0;
--    nd.root_test := false;
      nd.contend_time := 0;
      nd.child_count := 0;
      nd.lowest_unidentified_child := 0;
      nd.all_child_ports_identified := false;
      nd.self_ID_complete := false;
--    nd.requesting_child := 0;
--    nd.force_root := false; -- see above
--    nd.end_of_reception := false;
--    nd.link_req_active := false;
      nd.arb_enable := false;
      for i: Portnum do
        nd.old_connect[i] := false; -- actually array of NPORT
      end;
      nd.bus_initialize_active := true;
      nd.gap_count_reset_disable := false;
```

```
-- variables to emulate the physical aspects of a node
      for i: Portnum do alias pt: nd.ports[i] do
        pt.child := false;
        pt.connected := false;
        pt.child_ID_complete := false;
--      pt.max_peer_speed := S100;
--      pt.speed_OK := true;
        pt.data.TPA := Z;
        pt.data.TPB := Z;
      end; end;

-- current state of the node
      nd.Pstate := R0;
    end; end;

-- set up connection info
    nodes[0].ports[1].connected := true;
    nodes[1].ports[0].connected := true;

  end;


-- INVARIANTS

invariant "Do I manage to get out of state S0?"
  forall n: IDnum do
    nodes[n].Pstate != S1 & nodes[n].Pstate != S2
  end;
```

## B.2 Later Firewire Model

Our work on modeling firewire ended in March 1999. The final model, covering all of the cable physical layer initialization, leaves little room for source level transformation, reflecting the increased Mur$\varphi$ experience of the author and feedback from other experienced Mur$\varphi$ users. It is included here for archival purposes.

```
-- 1394.m
--
-- $Id: 1394.m,v 1.32 1999/03/26 00:50:22 bwinters Exp $
--
-- Brian Winters
--
-- History:
--   to do: check conditions in All:A0b wrt TIMER_MAX->3?
--   to do: clean up receive_port between states when applicable
--   [nudibranch is a dual UltraSPARC 360MHz, running Solaris]
--   [platypus is a dual UltraSPARC 300MHz, running Solaris]
--   [flake is a dual PII 233MHz, running Linux]
--    3/25/99   Disabled hack on All:R0b for now.  It isn't providing a
```

```
--            huge improvement over unhacked, and does muck with the
--            model.
--              5150047 states, 19071094 rules fired in 852.98s (nudibr)
--   3/25/99  Added back in a state to my All:R0b hack that is necessary
--            for N=3 to avoid deadlock.
--              4584064 states, 15338945 rules fired in 701.40s (nudibr)
--   3/25/99  Put things back together after playing with N=3
--            Tested same model as 2/4 on new machine
--              2907200 states, 9532685 rules fired in 435.33s (nudibr)
--   2/04/99  Made All:R0b only apply to states which deadlock without it
--              2907200 states, 9532685 rules fired in 515.88s (platypus)
--   2/03/99  Removed NPORT from node, made into function
--              5150047 states, 19071094 rules fired in 1121.10s (platy)
--   2/03/99  Commented out a little more dead code (A5:A0a, S0:S2)
--              5150047 states, 19071094 rules fired in 1071.75s (platy)
--   2/02/99  Commented out a bunch of dead code (A[2346] are never
--              reached)
--            Added numerous assert/error statements
--            Commented out the asserts because they were too slow
--              5150047 states, 19071094 rules fired in 1168.31s (pl cnt)
--              5150047 states, 19071094 rules fired in 1156.59s (platy)
--   2/02/99  Removed connected from port, made into function
--            Reordered a couple conditions wrt connect()
--            Commented out one line in one rule (and lost two rule
--            firings)
--                (I think was result of removing old_connect
--                 check/update)
--              5150047 states, 19071094 rules fired
--   2/01/99  Took out dead vars and one assignment in self_ID_trans...
--              5150047 states, 19071096 rules fired in 1206.74s (platy)
--   2/01/99  Commented out some dead code in self_ID_transmit_actions
--              5150047 states, 19071096 rules fired in 1253.60s (platy)
--   1/15/99  Moved random-hack to only affect x:T3 transitions
--              5150047 states, 19071096 rules fired in 1244.00s (platy)
--  12/01/98  All states, two nodes x two ports
--              5150047 states, 37670446 rules fired in 2264.98s (platy)
--  12/01/98  All states, no body on A5 or A6, three nodes, each two
--            ports
--              40-bit hash compression, -m512, "too many active states":
--              74300000 states, 718279094 rules fired in 47145.27s (pla)
--              9940746 states pending
--  11/30/98  All states, no body on A5 or A6
--              5150047 states, 37670446 rules fired in 4549.30s (platy)
--  11/30/98  States R0 through A1 implemented, no :A5 or :A6
--              4343811 states, 31828206 rules fired in 3275.40s (platy)
--              Pr[even one omitted state]    <= 0.000000
--              Pr[even one undetected error] <= 0.000000
--              Diameter of reachability graph: 124
--  11/30/98  States R0 through A1 implemented, A0:(!(A0|A1)) not
--              implemented
--              4343811 states, 31828206 rules fired in 1617.33s (platy)
--  11/27/98  States R0 through S3 implemented, S4 partial, A0 dead ends
--              1141195 states, 8381326 rules fired in 518.08s (flake)
--              Pr[even one omitted state]    <= 0.000000
--              Pr[even one undetected error] <= 0.000000
--              Diameter of reachability graph: 98
--  11/27/98  States R0 through S3 implemented, S4 dead ends
--              358971 states, 2638166 rules fired in 159.42s (flake)
--              Pr[even one omitted state]    <= 0.000000
--              Pr[even one undetected error] <= 0.000000
```

```
--               Diameter of reachability graph: 77
--   11/27/98   States R0 through S2 implemented, S2 and S4 dead end
--               344211 states, 2533862 rules fired in 149.13s (flake)
--                                                     114.31s (platypus)
--               Pr[even one omitted state]    <= 0.000000
--               Pr[even one undetected error] <= 0.000000
--               Diameter of reachability graph: 77
--   11/27/98   States R0 through S2 implemented, no S1:S4, S2 dead ends
--               339171 states, 2498750 rules fired in 147.58s (flake)
--               Pr[even one omitted state]    <= 0.000000
--               Pr[even one undetected error] <= 0.000000
--               Diameter of reachability graph: 77
--   11/27/98   States R0 through S2 implemented, S1 and S2 dead end
--               S0:S2 rule never fires, no effective change in model.
--   11/27/98   States R0 through S1 implemented, S1 dead ends
--               218095 states, 1616350 rules fired in 89.94s (flake)
--               Pr[even one omitted state]    <= 0.000000
--               Pr[even one undetected error] <= 0.000000
--               Diameter of reachability graph: 71
--   11/27/98   States R0 through S0 implemented, S0 dead ends
--               203695 states, 1519390 rules fired in 82.25s (flake)
--               Pr[even one omitted state]    <= 0.000000
--               Pr[even one undetected error] <= 0.000000
--               Diameter of reachability graph: 71
--   11/24/98   States R0 through S0 implemented, S0 dead ends
--               2060216 states, 15204586 rules fired in 787.31s (flake)
--               Pr[even one omitted state]    <= 0.000000
--               Pr[even one undetected error] <= 0.000000
--               Diameter of reachability graph: 121

const
  N: 2;              -- number of nodes
  HIGHPORT: 1;       -- highest numbered port possible per node
--HIGHFIFO_DEPTH: 0; -- highest numbered entry in a FIFO

  TIMER_MAX: 2;

  -- table 4-32
  CONFIG_TIMEOUT:     2; --3; --5; --14; -- defined as 16384
  FORCE_ROOT_TIMEOUT: 1; --2; --4; --13; -- defined as >=8192 and
                                         --   <= CONFIG_TIMEOUT
  ROOT_CONTEND_FAST:  1; --1; --1; -- 2; -- defined as 24
  ROOT_CONTEND_SLOW:  2; --2; --2; -- 3; -- defined as 56
  RESET_TIME:         1; --3; --5; --14; -- defined as 16384
  RESET_WAIT:         2; --4; --6; --15; -- defined as 16400
  MAX_ARB_STATE_TIME: RESET_WAIT;

  -- semi-contrived, see table 4-44
  GAP_COUNT_MAX:      2; --2; --3; -- 4; -- defined as 63


type
  IDnum: 0..(N-1);
  Portnum: 0..HIGHPORT;
--  FIFOnum: 0..HIGHFIFO_DEPTH;
  irange: 0..(HIGHPORT+1);

  PHY_state: enum { R0, R1,
                  T0, T1, T2, T3,
                  S0, S1, S2, S3, S4,
```

```
                  A0, A1, A2, A3, A4, A5, A6 };

  -- bogus types
  dataBit: boolean;
  baserate_time: 0..TIMER_MAX;
  timer: baserate_time;
  gap_counter: 0..GAP_COUNT_MAX;

  -- 4.4 Cable PHY operation, table 4-37
  phyData: enum { L, H, Z };
  speedCode: enum { S100, S200, S400 };
  portData :
    record
      TPA: phyData;
      TPB: phyData;
    end;


  -- 4.3.3: transmitted line states, table 4-27
  ArbLineStateTX: enum { TX_IDLE, TX_REQUEST, TX_GRANT, TX_PARENT_NOTIFY,
                         TX_DATA_PREFIX, TX_CHILD_NOTIFY, TX_IDENT_DONE,
                         TX_DATA_END, TX_BUS_RESET };
  -- 4.3.3: received line states, table 4-28
  ArbLineStateRX: enum { RX_IDLE, RX_PARENT_NOTIFY, RX_REQUEST_CANCEL,
                         RX_IDENT_DONE, RX_SELF_ID_GRANT, RX_REQUEST,
                         RX_ROOT_CONTENTION, RX_GRANT, RX_PARENT_HANDSHAKE,
                         RX_DATA_END, RX_CHILD_HANDSHAKE, RX_DATA_PREFIX,
                         RX_BUS_RESET };



-- PORT definition

  port:
    record
-- 4.3.9 Port variables
      child: boolean;
--    connected: boolean; -- replaced with connected(node,port):boolean
      child_ID_complete: boolean;
--    max_peer_speed: MaxSpeed; --NYM
--    speed_OK: boolean; --NYM

      data: portData; -- this one not explicitly defined, guessing here
    end;


-- NODE definition

  node:
    record

-- 4.3.7 Cable PHY node constants
--    NPORT: irange;
--    PHY_DELAY_CATEGORY: 0(..3); -- this can only be 0 in current spec
--    PHY_SPEED: MaxSpeed; -- NYM
--    POWER_CLASS: 0..7; -- don't care about physical power characteristics
--    CONTENDER: boolean;

-- 4.3.8 Node variables, table 4-35
      arb_enable: boolean;
```

```
--    cable_power_active: boolean;
      force_root: boolean;
      gap_count: gap_counter; -- guessing on variable type here
      initiated_reset: boolean;
      link_active: boolean;
--    more_packets: boolean;
      parent_port: Portnum;
      physical_ID: IDnum;
      receive_port: irange; -- "transmitting" def: receive_port=NPORT
      root: boolean;

-- 4.4 Cable PHY operation, table 4-37, implied internal variables
--    fifo_rd_ptr: FIFOnum;
--    fifo_wr_ptr: FIFOnum;
--    FIFO_DEPTH: FIFOnum;
--    FIFO: array[FIFOnum] of dataBit;
--    waiting_for_data_start: boolean;
--    rx_speed: speedCode;

      arb_timer: timer;
--    root_test: boolean;
      contend_time: baserate_time;
      child_count: irange;
      lowest_unidentified_child: Portnum;
      all_child_ports_identified: boolean;
      self_ID_complete: boolean;
--    requesting_child: Portnum; -- potentially need to clean up this one
--    force_root: boolean; -- see above
      end_of_reception: boolean;
--    link_req_active: boolean;
--    arb_enable: boolean; -- see above
--    old_connect: array[Portnum] of boolean; -- actually array of NPORT
      bus_initialize_active: boolean;
      gap_count_reset_disable: boolean;

-- variables implied by the C code or state machines, but never declared
--    cycle_master_req: boolean;
--    fair_req: boolean;
--    isoch_req: boolean;
--    imm_req: boolean;
--    end_of_transmission: boolean;

-- variables to emulate the physical aspects of a node
      ports: array[Portnum] of port;

-- current state of the node
      Pstate: PHY_state;
    end;


var
  nodes: array[IDnum] of node;


-- FUNCTIONS


-- mapping functions, to model the physical wiring

-- assumes all nodes have same number of ports
```

```
function NPORT(n: IDnum): irange;
begin
  return HIGHPORT+1;
end;

-- assumes N=2, HIGHPORT=1
function connected(n: IDnum; port_number: Portnum): boolean;
begin
--  assert (N=2 & HIGHPORT=1) "N and/or HIGHPORT do not match";
  return ((n=0 & port_number=1) | (n=1 & port_number=0));
end;

-- assumes N=3, HIGHPORT=1
--function connected(n: IDnum; port_number: Portnum): boolean;
--begin
--  assert (N=3 & HIGHPORT=1) "N and/or HIGHPORT do not match";
--  return ((n=0 & port_number=1) | (n=1) | (n=2 & port_number=0));
--end;

-- assumes N=2, HIGHPORT=1
function trans_node(n: IDnum; port_number: Portnum): IDnum;
begin
--  assert (N=2 & HIGHPORT=1) "N and/or HIGHPORT do not match";
  if(n=0 & port_number=1) then
    return 1;
  elsif(n=1 & port_number=0) then
    return 0;
  else
    error "Invalid node/port combination";
  end;
end;

-- assumes N=3, HIGHPORT=1
--function trans_node(n: IDnum; port_number: Portnum): IDnum;
--begin
--  assert (N=3 & HIGHPORT=1) "N and/or HIGHPORT do not match";
--  if(n=0 & port_number=1) then
--    return 1;
--  elsif(n=1 & port_number=0) then
--    return 0;
--  elsif(n=1 & port_number=1) then
--    return 2;
--  elsif(n=2 & port_number=0) then
--    return 1;
--  else
--    error "Invalid node/port combination";
--  end;
--end;

-- assumes N=2, HIGHPORT=1
function trans_port(n: IDnum; port_number: Portnum): Portnum;
begin
--  assert (N=2 & HIGHPORT=1) "N and/or HIGHPORT do not match";
  if(n=0 & port_number=1) then
    return 0;
  elsif(n=1 & port_number=0) then
    return 1;
  else
    error "Invalid node/port combination";
  end;
```

```
end;

-- assumes N=3, HIGHPORT=1
--function trans_port(n: IDnum; port_number: Portnum): Portnum;
--begin
--  assert (N=3 & HIGHPORT=1) "N and/or HIGHPORT do not match";
--  if(n=0 & port_number=1) then
--    return 0;
--  elsif(n=1 & port_number=0) then
--    return 1;
--  elsif(n=1 & port_number=1) then
--    return 0;
--  elsif(n=2 & port_number=0) then
--    return 1;
--  else
--    error "Invalid node/port combination";
--  end;
--end;



-- hardware functions, prototyped from table 4-37

--function random_bool(): boolean;
--begin
---- pseudorandom - will protocol work even with bad randomizer?
--  return true;
--  return false;
----  return random_value;
--end;

--procedure portT(n: IDnum; port_number: Portnum; txData: portData);
procedure portT(n: IDnum; port_number: Portnum; txData: ArbLineStateTX);
begin
  alias p: nodes[n].ports[port_number].data do -- from table 4-27
    switch txData
    case TX_IDLE:
      p.TPA := Z;
      p.TPB := Z;
    case TX_REQUEST:
      p.TPA := Z;
      p.TPB := L;
    case TX_GRANT:
      p.TPA := Z;
      p.TPB := L;
    case TX_PARENT_NOTIFY:
      p.TPA := L;
      p.TPB := Z;
    case TX_DATA_PREFIX:
      p.TPA := L;
      p.TPB := H;
    case TX_CHILD_NOTIFY:
      p.TPA := H;
      p.TPB := Z;
    case TX_IDENT_DONE:
      p.TPA := H;
      p.TPB := Z;
    case TX_DATA_END:
      p.TPA := H;
      p.TPB := L;
```

```
    case TX_BUS_RESET:
      p.TPA := H;
      p.TPB := H;
    end;
  end;
end;

function resolve(a, b: phyData): phyData;
begin
  if(a=Z) then return b; end;
  if(b=Z | a=b) then return a; end;
  return Z;
end;

--function portR(n, port_number: IDnum): portData;
function portR(n, port_number: IDnum): ArbLineStateRX;
var Tn, Tp: IDnum;
    otherA, otherB: phyData;
begin
  otherA := Z;
  otherB := Z;
  if(connected(n, port_number)) then
    Tn := trans_node(n, port_number);
    Tp := trans_port(n, port_number);
    otherA := nodes[Tn].ports[Tp].data.TPA;
    otherB := nodes[Tn].ports[Tp].data.TPB;
  end;
  alias pt: nodes[n].ports[port_number].data do -- from table 4-28
    if(resolve(pt.TPA, otherB) = Z & resolve(pt.TPB, otherA) = Z) then
      return RX_IDLE;
    elsif(resolve(pt.TPA, otherB) = Z & resolve(pt.TPB, otherA) = L) then
      return RX_PARENT_NOTIFY; -- also RX_REQUEST_CANCEL
    elsif(resolve(pt.TPA, otherB) = Z & resolve(pt.TPB, otherA) = H) then
      return RX_IDENT_DONE;
    elsif(resolve(pt.TPA, otherB) = L & resolve(pt.TPB, otherA) = Z) then
      return RX_SELF_ID_GRANT; -- also RX_REQUEST
    elsif(resolve(pt.TPA, otherB) = L & resolve(pt.TPB, otherA) = L) then
      return RX_ROOT_CONTENTION; -- also RX_GRANT
    elsif(resolve(pt.TPA, otherB) = L & resolve(pt.TPB, otherA) = H) then
      return RX_PARENT_HANDSHAKE; -- also RX_DATA_END
    elsif(resolve(pt.TPA, otherB) = H & resolve(pt.TPB, otherA) = Z) then
      return RX_CHILD_HANDSHAKE;
    elsif(resolve(pt.TPA, otherB) = H & resolve(pt.TPB, otherA) = L) then
      return RX_DATA_PREFIX;
    elsif(resolve(pt.TPA, otherB) = H & resolve(pt.TPB, otherA) = H) then
      return RX_BUS_RESET;
    end;
  end;
end;

function portRcomp(n, port_number: IDnum; c: ArbLineStateRX): boolean;
var r: ArbLineStateRX;
begin
  r := portR(n, port_number);
  switch r
  case RX_PARENT_NOTIFY:
    if(c = RX_REQUEST_CANCEL) then
      return true;
    end;
  case RX_SELF_ID_GRANT:
```

```
    if(c = RX_REQUEST) then
      return true;
    end;
  case RX_ROOT_CONTENTION:
    if(c = RX_GRANT) then
      return true;
    end;
  case RX_PARENT_HANDSHAKE:
    if(c = RX_DATA_END) then
      return true;
    end;
  end;
  return (r = c);
end;



--procedure portTspeed(n, port_number: IDnum; speed: speedCode);
--function portRspeed(n, port_number: IDnum): speedCode;


function subaction_gap_detect_time(n: IDnum): gap_counter; -- table 4-33
begin
--defined as:
--  return (28 + (nodes[n].gap_count * 16));
  return nodes[n].gap_count / 2; -- I HAVE NO JUSTIFICATION FOR THIS CHOICE
end;

function arb_reset_gap_detect_time(n: IDnum): gap_counter; -- table 4-33
begin
--defined as:
--  return (28 + (nodes[n].gap_count * 16));
  return nodes[n].gap_count; -- I HAVE NO JUSTIFICATION FOR THIS CHOICE
end;

--function ARB_DELAY(n: IDnum): gap_counter; -- table 4-34
--begin
----defined as:
----  return (nodes[n].gap_count * 4);
--  return nodes[n].gap_count / 2; -- HAVE NO JUSTIFICATION FOR THIS CHOICE
--end;



-- 4.4.2.1.2 Bus reset actions and conditions

function bus_reset_signal_received(n: IDnum): boolean; -- table 4-44
var i: irange;
begin
  i:=0;
  while(i < NPORT(n)) do
    if(connected(n, i) & portRcomp(n, i, RX_BUS_RESET)) then
      return true;
    end;
    i:=i+1;
  end;
  return false;
end;

--function connection_state_change(n: IDnum): boolean; -- table 4-44
```

```
--var state_change: boolean; i: irange;
--begin
--  state_change := false;
--  i:=0;
--  while(i < NPORT(n)) do
--    if(connected(n, i) != nodes[n].old_connect[i]) then
--      state_change := true;
--      nodes[n].old_connect[i] := !nodes[n].old_connect[i];
--    end;
--    i:=i+1;
--  end;
--  return state_change;
--end;

procedure reset_start_actions(n: IDnum); -- table 4-44
var i: irange;
begin
  nodes[n].root := false;
  nodes[n].physical_ID := 0;
  nodes[n].child_count := 0;
  nodes[n].bus_initialize_active := true;
  if(nodes[n].gap_count_reset_disable) then
    nodes[n].gap_count_reset_disable := false;
  else
    nodes[n].gap_count := GAP_COUNT_MAX; --hard coded to 0x3F in table 4-44
  end;
  i:=0;
  while(i < NPORT(n)) do
    if(connected(n, i)) then
      portT(n, i, TX_BUS_RESET);
    else
      portT(n, i, TX_IDLE);
    end;
    nodes[n].ports[i].child := false;
    nodes[n].ports[i].child_ID_complete := false;
    i:=i+1;
  end;
  nodes[n].arb_timer := 0;
end;

procedure reset_wait_actions(n: IDnum); -- table 4-44
var i: irange;
begin
  i:=0;
  while(i < NPORT(n)) do
    portT(n, i, TX_IDLE);
    i:=i+1;
  end;
  nodes[n].arb_timer := 0;
end;

function reset_complete(n: IDnum): boolean; -- table 4-44
var i: irange;
begin
  i:=0;
  while(i < NPORT(n)) do
    if(connected(n, i) &
       !portRcomp(n, i, RX_IDLE) &
       !portRcomp(n, i, RX_PARENT_NOTIFY)) then
      return false;
```

```
    end;
    i:=i+1;
  end;
  return true;
end;

function arb_state_timeout(n: IDnum): boolean; -- support for figure 4-22
begin alias nd: nodes[n] do alias st: nd.Pstate do
  return ((st != R1) & (st != A0) &
          (st != A5) & (st != A6) &
          (nd.arb_timer >= MAX_ARB_STATE_TIME));
end; end; end;


-- 4.4.2.2.2 Tree-ID actions and conditions

procedure tree_ID_start_actions_interp_1(n: IDnum); -- table 4-45
var i, temp_count: irange;
begin
  nodes[n].arb_timer := 0;
  -- LOOP FOREVER???  EXCUSE ME????

  -- my suspicion here is that they intend that this be going on as some
  -- sort of background process that is always updating child count, but
  -- on second thought that doesn't make sense, because we immediately go
  -- into a reset state if our connections change

--  while(true) do
    temp_count := 0;
    i:=0;
    while(i<NPORT(n)) do
-- formatting of standard is ambiguous here; using C code and not indention
-- my suspicion is that this interpretation is WRONG, but we'll see
      if(!connected(n, i) | portRcomp(n, i, RX_PARENT_NOTIFY)) then
        nodes[n].ports[i].child := true;
        temp_count := temp_count + 1;
        nodes[n].child_count := temp_count;
      end;
      i:=i+1;
    end;
--  end;
end;

procedure tree_ID_start_actions_interp_2(n: IDnum); -- table 4-45
var i, temp_count: irange;
begin
  nodes[n].arb_timer := 0;
  temp_count := 0;
  i:=0;
  while(i<NPORT(n)) do
-- formatting of standard is ambiguous here; using indention and not C code
    if(!connected(n, i) | portRcomp(n, i, RX_PARENT_NOTIFY)) then
      nodes[n].ports[i].child := true;
      temp_count := temp_count + 1;
    end;
    i:=i+1;
  end;
  nodes[n].child_count := temp_count;
--  end;
end;
```

```
procedure tree_ID_start_actions(n: IDnum); -- table 4-45
begin
--  tree_ID_start_actions_interp_1(n);
  tree_ID_start_actions_interp_2(n);
end;

procedure child_handshake_actions(n: IDnum); -- table 4-45
var i: irange;
begin
  nodes[n].root := true;
  i:=0;
  while(i < NPORT(n)) do
    if(connected(n, i)) then
      if(nodes[n].ports[i].child) then
        portT(n, i, TX_CHILD_NOTIFY);
      else
        portT(n, i, TX_PARENT_NOTIFY);
        nodes[n].parent_port := i;
        nodes[n].root := false;
      end;
    end;
    i:=i+1;
  end;
end;

function child_handshake_complete(n: IDnum): boolean; -- table 4-45
var i: irange;
begin
  i:=0;
  while(i < NPORT(n)) do
    if(nodes[n].ports[i].child &
       connected(n, i) &
       !portRcomp(n, i, RX_CHILD_HANDSHAKE)) then
      return false;
    end;
    i:=i+1;
  end;
  return true;
end;

procedure root_contend_actions(n: IDnum; r: boolean); -- table 4-45
var i: irange;
begin
--  if(random_bool()) then
  if(r) then
    nodes[n].contend_time := ROOT_CONTEND_SLOW;
  else
    nodes[n].contend_time := ROOT_CONTEND_FAST;
  end;
  i:=0;
  while(i < NPORT(n)) do
    if(nodes[n].ports[i].child) then
      portT(n, i, TX_CHILD_NOTIFY);
    else
      portT(n, i, TX_IDLE);
    end;
    i:=i+1;
  end;
  nodes[n].arb_timer := 0;
```

```
end;

procedure self_ID_start_actions(n: IDnum); -- table 4-46
var i: irange;
begin
  nodes[n].all_child_ports_identified := true;
  i:=0;
  while(i < NPORT(n)) do
    if(nodes[n].ports[i].child_ID_complete) then
      portT(n, i, TX_DATA_PREFIX);
    else
      portT(n, i, TX_IDLE);
      if(nodes[n].ports[i].child & connected(n, i)) then
        if(nodes[n].all_child_ports_identified) then
          nodes[n].lowest_unidentified_child := i;
        end;
        nodes[n].all_child_ports_identified := false;
      end;
    end;
    i:=i+1;
  end;
end;

procedure self_ID_grant_actions(n: IDnum); -- table 4-46
var i: irange;
begin
  i:=0;
  while(i<NPORT(n)) do
    if(!nodes[n].all_child_ports_identified &
       (i = nodes[n].lowest_unidentified_child)) then
      portT(n, i, TX_GRANT);
    else
      portT(n, i, TX_DATA_PREFIX);
    end;
    i:=i+1;
  end;
end;

procedure self_ID_receive_actions(n: IDnum); -- table 4-46
var i: irange;
begin
  portT(n, nodes[n].receive_port, TX_IDLE);
--  -- receive_actions(n);
  nodes[n].physical_ID := nodes[n].physical_ID + 1;
  i:=0;
  while(i<NPORT(n)) do
    portT(n, i, TX_IDLE);
    i:=i+1;
  end;
end;

procedure self_ID_transmit_actions(n: IDnum); -- table 4-46
var port_number: irange;
--  port_number: 0..1024;
--  last_SID_pkt, SID_pkt_number: irange;
begin
--  last_SID_pkt := (NPORT(n) + 4) / 8;
  port_number := 0;
  nodes[n].self_ID_complete := false;
  nodes[n].receive_port := NPORT(n); -- this is def for "transmitting"
```

```
--start_tx_packet(n, S100);
-- I've abstracted away everying about doing the SID packets, which makes
-- this loop pretty meaningless
--  SID_pkt_number := 0;
--  while(SID_pkt_number <= last_SID_pkt) do
--    -- abstracting away SID packet construction
---- THIS SECTION CAUSES PORT NUMBER TO GO OUT OF RANGE. ----
---- AT BEST THIS IS JUST BAD PRACTICE.  AT WORST IT IS VERY BAD. ----
---- Let's give them enough rope to hang themselves. ----
------ Alan and I later looked this over, and the stuff I'm abstracting
------ away handles this properly.  Why they did this was still a mystery.
--    while(port_number < ((SID_pkt_number + 1)*8 - 5)) do
--      -- abstracting away more SID packet construction
--      port_number:=port_number+1;
--    end;
--    -- abstracting away even more SID packet construction
--  if(SID_pkt_number = last_SID_pkt) then
--      -- don't send quadlets
--    stop_tx_packet(n, TX_DATA_END, S100);
--  else
--    stop_tx_packet(n, TX_DATA_PREFIX, S100);
--  end;
--    SID_pkt_number:=SID_pkt_number+1;
--  end;
  port_number:=0;
  while(port_number < NPORT(n)) do
    if(port_number = nodes[n].parent_port) then
      portT(n, port_number, TX_IDENT_DONE);
      -- portTspeed();
      -- wait_time();
      -- portTspeed();
      -- PH_EVENT.ind(SELF_ID_COMPLETE, physical_ID, root);
    else
      portT(n, port_number, TX_IDLE);
    end;
    port_number:=port_number+1; -- bad bug in standard I think
                                --   upon further reflection, I have no
                                --   idea why I thought this
  end;
  nodes[n].self_ID_complete := true;
end;

--function child_request(n: IDnum): boolean; -- table 4-47
--var i: irange;
--begin alias nd: nodes[n] do
--  i:=0;
--  while(i < NPORT(n)) do
--    if(connected(n, i) & nd.ports[i].child &
--       portRcomp(n, i, RX_REQUEST)) then
--      nd.requesting_child := i;
--      return true;
--    end;
--    i:=i+1;
--  end;
--  return false;
--end; end;

function data_coming(n: IDnum): boolean; -- table 4-47
var i: irange;
begin alias nd: nodes[n] do
```

```
  i:=0;
  while(i < NPORT(n)) do
    if(connected(n, i) & portRcomp(n, i, RX_DATA_PREFIX)) then
      nd.receive_port := i;
      return true;
    end;
    i:=i+1;
  end;
  return false;
end; end;

procedure idle_actions(n: IDnum); -- table 4-47
var i: irange;
begin
  i:=0;
  while(i < NPORT(n)) do
    portT(n, i, TX_IDLE);
    i:=i+1;
  end;
end;

procedure subaction_detect_actions(n: IDnum); -- table 4-47
begin
--PH_DATA.ind(SUBACTION_GAP);
  if(nodes[n].bus_initialize_active) then
--  PH_EVENT.ind(BUS_RESET_COMPLETE);
    nodes[n].bus_initialize_active := false;
  end;
end;

--procedure request_delay_actions(n: IDnum); -- table 4-47
--begin
--  nodes[n].arb_timer := 0;
--  nodes[n].link_req_active := true;
--end;

--procedure request_actions(n: IDnum); -- table 4-47
--var i: irange;
--begin alias nd: nodes[n] do
--  i:=0;
--  while(i<NPORT(n)) do
--    if(connected(n, i) & nd.ports[i].child &
--        (nd.link_req_active | (i != nd.requesting_child))) then
--      portT(n, i, TX_DATA_PREFIX);
--    end;
--    i:=i+1;
--  end;
--  portT(n, nd.parent_port, TX_REQUEST);
--end; end;

--procedure grant_actions(n: IDnum); -- table 4-47
--var i: irange;
--begin
--  i:=0;
--  while(i < NPORT(n)) do
--    if(i = nodes[n].requesting_child) then
--      portT(n, i, TX_GRANT);
--    elsif(connected(n, i) & nodes[n].ports[i].child) then
--      portT(n, i, TX_DATA_PREFIX);
--    end;
```

```
--    i:=i+1;
--  end;
--end;

procedure receive_actions(n: IDnum); -- table 4-47
var --i: this one isn't irange;
--  bit_count: ???;
--  phy_pkt: some really big union;
--  test_end, received_data: boolean;
    good_phy_packet: boolean;
begin alias nd: nodes[n] do
--bit_count := 0;
--test_end := false;
--nd.fair_req := false;
--nd.cycle_master_req := false;
--PHY_DATA.ind(DATA_PREFIX);
--nd.rx_speed := start_rx_packet();
--PHY_DATA.ind(DATA_START(rx_speed));
--while(!test_end) do
--  rx_bit(&received_data, &test_end);
--  if(!test_end) then
--    PH_DATA.ind(received_data);
--    if(bit_count < 64) then
--      phy_pkt.bits[bit_count++] = received_data;
--    end;
--  end;
--end;
--if(portRcomp(n, nd.receive_port, RX_DATA_PREFIX)) then
--  PH_DATA.ind(DATA_PREFIX);
--elsif(portRcomp(n, nd.receive_port, RX_DATA_END)) then
--  PH_DATA.ind(DATA_END);
--end;
--stop_rx_packet(ending_data);
  nd.end_of_reception := true;
  if(
--   bit_count = 64
     true
    ) then
    good_phy_packet := true;
--  i:=0;
--  while(i<32) do
--    good_phy_packet := compare bits in 1st half vs the negated 2nd half
--  end;
--  if(good_phy_packet) then
--    phy_addr := received_bits;
--    and a bunch of other stuff I'm not going to model at this point
--  end;
  end;
end; end;

--procedure transmit_actions(n: IDnum); -- table 4-47
--var --i: why did they declare it?
--    test_end: boolean;
----  data_to_transmit: phyData;
--begin alias nd: nodes[n] do
--  test_end := false;
--  nd.imm_req := false;
--  if(nd.fair_req) then
--    nd.arb_enable := false;
--    nd.fair_req := false;
```

```
--  end;
--  nd.cycle_master_req := false;
--  nd.isoch_req := false;
--  nd.receive_port := NPORT(n); -- we are transmitting
----start_tx_packet(req_speed);
----while(!test_end) do
----  a bunch of stuff I'm not going to model yet
----end;
--  nd.end_of_transmission := true;
--end; end;

procedure change_PHY_state_withrand(n: IDnum; r: boolean;
                                    newstate: PHY_state);
begin
--  assert newstate=T3 "Using rand change_PHY on invalid state change";
  nodes[n].Pstate := newstate;
  switch newstate
  case T3:
    root_contend_actions(n, r); -- figure 4-23
  end;
end;

procedure change_PHY_state(n: IDnum; newstate: PHY_state);
begin
--  assert newstate!=T3 "Using nonrand change_PHY on invalid state change";
  nodes[n].Pstate := newstate;
  switch newstate
  case R0:
    reset_start_actions(n); -- figure 4-22
  case R1:
    reset_wait_actions(n); -- figure 4-22
  case T0:
    tree_ID_start_actions(n); -- figure 4-23
  case T1:
    child_handshake_actions(n); -- figure 4-23
  case T2:
    -- this state is just a waiting point (figure 4-23)
--  case T3:
--    root_contend_actions(n, r); -- figure 4-23
  case S0:
    self_ID_start_actions(n); -- figure 4-24
  case S1:
    self_ID_grant_actions(n); -- figure 4-24
  case S2:
    self_ID_receive_actions(n); -- figire 4-24
  case S3:
    -- all output at this state is on transitions -- figure 4-24
  case S4:
    self_ID_transmit_actions(n); -- figure 4-24
  case A0:
    idle_actions(n); -- figure 4-25
  case A1:
    -- this state doesn't actually do anything (figure 4-25, 4.4.2.4.1)
  case A2:
    error "Transition to state A2";
--    request_delay_actions(n); -- figure 4-25
  case A3:
    error "Transition to state A3";
--    request_actions(n); -- figure 4-25
  case A4:
```

```
    error "Transition to state A4";
--    grant_actions(n); -- figure 4-25
  case A5:
    receive_actions(n); -- figure 4-25
  case A6:
    error "Transition to state A6";
--    transmit_actions(n); -- figure 4-25
  end;
end;



-- RULESETS

-- random value hack
--ruleset r: boolean do

-- state machine by node

ruleset n: IDnum do
  alias nd: nodes[n] do
  alias st: nd.Pstate do
  rule "increment timer (modeling hack)"
    ((st != A1) &  -- A1 "takes no time"
     (nd.arb_timer < TIMER_MAX)) ==>
      begin
        nd.arb_timer := nd.arb_timer + 1;
      end;

  rule "Transition All:R0a (4.4.2.1.1)"
    bus_reset_signal_received(n) ==>
      begin
--        -- PH_STATE.ind(BUS_RESET_START) (4.4.2.1)
        nd.initiated_reset := false;
        change_PHY_state(n, R0);
      end;
--- experiment to see how bad deadlock is without the timeout
  rule "Transition All:R0b (4.4.2.1.1)"
    (
     (
--    (st=T2 | st=T3 | st=S4 | st=S0) & -- statespace reduction hack
--                                     -- this may not be valid <======*
      arb_state_timeout(n)) --|
--   PH_CONT.req(bus_reset) |
--   connection_state_change(n)
    ) ==>
      begin
--        -- PH_STATE.ind(BUS_RESET_START) (4.4.2.1)
        nd.initiated_reset := true;
        change_PHY_state(n, R0);
      end;
  rule "Transition R0:R1 (4.4.2.1.1)"
    ((st = R0) & (nd.arb_timer >= RESET_TIME)) ==>
      begin
        change_PHY_state(n, R1);
      end;
  rule "Transition R1:R0 (4.4.2.1.1)"
    ((st = R1) & (nd.arb_timer >= RESET_WAIT)) ==>
      begin
        change_PHY_state(n, R0);
```

```
      end;
  rule "Transition R1:T0 (4.4.2.1.1)"
    ((st = R1) & reset_complete(n)) ==>
      begin
        change_PHY_state(n, T0);
      end;
-- this was never happening, so we can speed things up by removing it
--  rule "Transition T0:T0 (4.4.2.2.1)"
--    ((st = T0) & ((nd.arb_timer >= CONFIG_TIMEOUT) &
--                  (nd.child_count < (NPORT(n) - 1)))) ==>
--      begin
----      -- signal PH_STATE.ind(CONFIG_TIMEOUT) (4.4.2.2)
--        change_PHY_state(n, T0);
--      end;
  rule "Transition T0:T1 (4.4.2.2.1)"
    ((st = T0) & ((nd.child_count = NPORT(n)) |
                 (((nd.force_root & (nd.arb_timer >= FORCE_ROOT_TIMEOUT))|
                   (!nd.force_root & (nd.arb_timer >= 0))) &
                  (nd.child_count = (NPORT(n) - 1))))) ==>
      begin
        change_PHY_state(n, T1);
      end;
  rule "Transition T1:T2 (4.4.2.2.1)"
    ((st = T1) & child_handshake_complete(n)) ==>
      begin
        change_PHY_state(n, T2);
      end;
  rule "Transition T2:S0 (4.4.2.2.1)"
    ((st = T2) & (nd.root |
                 portRcomp(n, nd.parent_port, RX_PARENT_HANDSHAKE))) ==>
      begin
        change_PHY_state(n, S0);
      end;
 ruleset r: boolean do
  rule "Transition T2:T3 (4.4.2.2.1)"
    ((st = T2) & portRcomp(n, nd.parent_port, RX_ROOT_CONTENTION)) ==>
      begin
        change_PHY_state_withrand(n, r, T3);
      end;
 end;
  rule "Transition T3:T2 (4.4.2.2.1)"
    ((st = T3) & (portRcomp(n, nd.parent_port, RX_IDLE) &
                 (nd.arb_timer > nd.contend_time))) ==>
      begin
        portT(n, nd.parent_port, TX_PARENT_NOTIFY); -- 4.4.2.2
        change_PHY_state(n, T2);
      end;
  rule "Transition T3:T1 (4.4.2.2.1)"
    ((st = T3) & (portRcomp(n, nd.parent_port, RX_PARENT_NOTIFY) &
                 (nd.arb_timer > nd.contend_time))) ==>
      begin
        -- 4.4.2.2: "node is now root"
        nd.ports[nd.parent_port].child := true;
        change_PHY_state(n, T1);
      end;
  rule "Transition S0:S1 (4.4.2.3.1)"
    ((st = S0) & (nd.root |
                 portRcomp(n, nd.parent_port, RX_SELF_ID_GRANT))) ==>
      begin
        change_PHY_state(n, S1);
```

```
      end;
-- this was never happening, so speed things up by removing it
--  rule "Transition S0:S2 (4.4.2.3.1)"
--    ((st = S0) & portRcomp(n, nd.parent_port, RX_DATA_PREFIX)) ==>
--      begin
--        nd.receive_port := nd.parent_port;
--        change_PHY_state(n, S2);
--      end;
  rule "Transition S1:S2 (4.4.2.3.1)"
    ((st = S1) &
     portRcomp(n, nd.lowest_unidentified_child, RX_DATA_PREFIX)) ==>
      begin
        nd.receive_port := nd.lowest_unidentified_child;
        change_PHY_state(n, S2);
      end;
  rule "Transition S1:S4 (4.4.2.3.1)"
    ((st = S1) & nd.all_child_ports_identified) ==>
      begin
        -- nd.ports[nd.parent_port].max_peer_speed := S100;
        change_PHY_state(n, S4);
      end;
  rule "Transition S2:S0a (4.4.2.3.1)"
    ((st = S2) & (portRcomp(n, nd.receive_port, RX_IDLE) |
                  portRcomp(n, nd.receive_port, RX_SELF_ID_GRANT))) ==>
      begin
        change_PHY_state(n, S0);
      end;
  rule "Transition S2:S0b (4.4.2.3.1)"
    ((st = S2) & (
--                PHY_SPEED = S100 &
                  portRcomp(n, nd.receive_port, RX_IDENT_DONE))) ==>
      begin
        nd.ports[nd.receive_port].child_ID_complete := true;
        change_PHY_state(n, S0);
      end;
--rule "Transition S2:S3 (4.4.2.3.1)"
--  ((st = S2) & (PHY_SPEED != S100 &
--                portRcomp(n, nd.receive_port, RX_IDENT_DONE)) ==>
--    begin
--      nd.ports[nd.receive_port].child_ID_complete := true;
--      portTspeed(n, nd.receive_port, PHY_SPEED);
--      nd.ports[nd.receive_port].max_peer_speed := S100;
--      nd.arb_timer := 0;
--      change_PHY_state(n, S3);
--    end;
--rule "Transition S3:S0 (4.4.2.3.1)"
--  ((st = S3) & (arb_timer >= SPEED_SIGNAL_LENGTH)) ==>
--    begin
--      portTspeed(n, nd.receive_port, S100);
--      change_PHY_state(n, S0);
--    end;
--rule "Transition S3:S3a (4.4.2.3.1)"
--  ((st = S3) & (portRspeed(n, nd.receive_port) = S200)) ==>
--    begin
--      nd.ports[nd.receive_port].max_peer_speed := S200;
--      change_PHY_state(n, S3);
--    end;
--rule "Transition S3:S3b (4.4.2.3.1)"
--  ((st = S3) & (portRspeed(n, nd.receive_port) = S400)) ==>
--    begin
```

```
--      nd.ports[nd.receive_port].max_peer_speed := S400;
--      change_PHY_state(n, S3);
--    end;
--rule "Transition S4:S4a (4.4.2.3.1)"
--  ((st = S4) & (portRspeed(n, nd.parent_port) = S200)) ==>
--    begin
--      nd.ports[nd.parent_port].max_peer_seed := S200;
--      change_PHY_state(n, S4);
--    end;
--rule "Transition S4:S4b (4.4.2.3.1)"
--  ((st = S4) & (portRspeed(n) = S400)) ==>
--    begin
--      nd.ports[nd.parent_port].max_peer_seed := S400;
--      change_PHY_state(n, S4);
--    end;
  rule "Transition S4:A0 (4.4.2.3.1)"
    ((st = S4) & (nd.self_ID_complete &
                 (nd.root |
                  portRcomp(n, nd.parent_port, RX_DATA_PREFIX)))) ==>
      begin
        change_PHY_state(n, A0);
      end;
--rule "Transition A0:A0a (4.4.2.4.1)"
--  ((st = A0) & (PH_ARB.req(CYCLE_MASTER, req_speed))) ==>
--    begin
--      nd.cycle_master_req := true;
--      change_PHY_state(n, A0);
--    end;
--rule "Transition A0:A0b (4.4.2.4.1)"
--  ((st = A0) & (PH_ARB.req(FAIR, req_speed))) ==>
--    begin
--      nd.fair_req := true;
--      change_PHY_state(n, A0);
--    end;
--rule "Transition A0:A0c (4.4.2.4.1)"
--  ((st = A0) & (PH_ARB.req(ISOCH, req_speed))) ==>
--    begin
--      nd.isoch_req := true;
--      change_PHY_state(n, A0);
--    end;
  rule "Transition A0:A1a (4.4.2.4.1)"
    ((st = A0) & (nd.arb_timer = subaction_gap_detect_time(n))) ==>
      begin
        subaction_detect_actions(n);
        change_PHY_state(n, A1);
      end;
  rule "Transition A0:A1b (4.4.2.4.1)"
    ((st = A0) & (nd.arb_timer = arb_reset_gap_detect_time(n))) ==>
      begin
        nd.arb_enable := true;
--      PH_DATA.ind(ARB_RESET_GAP);
        change_PHY_state(n, A1);
      end;
-- this was never occuring, so speed things up
--  rule "Transition A0:A2 (4.4.2.4.1)"
--    ((st = A0) & ((nd.arb_timer > arb_reset_gap_detect_time(n)) &
--                  (nd.cycle_master_req |
--                   (nd.fair_req & nd.arb_enable)))) ==>
--      begin
--        change_PHY_state(n, A2);
```

```
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A0:A3 (4.4.2.4.1)"
--    ((st = A0) & ((child_request(n) | nd.isoch_req) & !nd.root)) ==>
--      begin
--        change_PHY_state(n, A3);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A0:A4 (4.4.2.4.1)"
--    ((st = A0) & (child_request(n) & nd.root)) ==>
--      begin
--        change_PHY_state(n, A4);
--      end;
  rule "Transition A0:A5 (4.4.2.4.1)"
    ((st = A0) & data_coming(n)) ==>
      begin
--      if(nd.fair_req | nd.cycle_master_req) then
--        PHY_ARB.conf(LOST);
--      end;
        change_PHY_state(n, A5);
      end;
-- this was never occuring, so speed things up
--  rule "Transition A0:A6 (4.4.2.4.1)"
--    ((st = A0) & (nd.imm_req | (nd.root & nd.isoch_req))) ==>
--      begin
----      PHY_ARB.conf(WON);
--        change_PHY_state(n, A6);
--      end;
  rule "Transition A1:A0 (4.4.2.4.1)"
    ((st = A1)
--   & !(nd.cycle_master_req | (nd.fair_req & nd.arb_enable))
    ) ==>
      begin
        change_PHY_state(n, A0);
      end;
-- this was never occuring, so speed things up
--  rule "Transition A1:A2 (4.4.2.4.1)"
--    ((st = A1) & (nd.cycle_master_req |
--                  (nd.fair_req & nd.arb_enable))) ==>
--      begin
--        change_PHY_state(n, A2);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A2:A3 (4.4.2.4.1)"
--    ((st = A2) & ((nd.arb_timer > ARB_DELAY(n)) & !nd.root)) ==>
--      begin
--        change_PHY_state(n, A3);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A2:A5 (4.4.2.4.1)"
--    ((st = A2) & data_coming(n)) ==>
--      begin
----      PHY_ARB.conf(LOST);
--        nd.link_req_active := false;
--        change_PHY_state(n, A5);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A3:A0 (4.4.2.4.1)"
--    ((st = A3) & (portRcomp(n, nd.parent_port, RX_GRANT) &
--                !nd.link_req_active & !child_request(n))) ==>
```

```
--      begin
--        change_PHY_state(n, A0);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A3:A4 (4.4.2.4.1)"
--    ((st = A3) & (portRcomp(n, nd.parent_port, RX_GRANT) &
--                  child_request(n))) ==>
--      begin
--        change_PHY_state(n, A4);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A3:A5 (4.4.2.4.1)"
--    ((st = A3) & (portRcomp(n, nd.parent_port, RX_GRANT) &
--                  !nd.link_req_active & !child_request(n))) ==>
--      begin
--        change_PHY_state(n, A5);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A3:A6 (4.4.2.4.1)"
--    ((st = A3) & (portRcomp(n, nd.parent_port, RX_GRANT) &
--                  (nd.link_req_active | nd.isoch_req))) ==>
--      begin
----      PHY_ARB.conf(WON);
--        change_PHY_state(n, A6);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A4:A0 (4.4.2.4.1)"
--    ((st = A4) &
--     portRcomp(n, nd.requesting_child, RX_REQUEST_CANCEL)) ==>
--      begin
--        nd.arb_timer := 0;
--        change_PHY_state(n, A0);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A4:A5 (4.4.2.4.1)"
--    ((st = A4) & data_coming(n)) ==>
--      begin
--        change_PHY_state(n, A5);
--      end;
-- this was never occuring, so speed things up
--  rule "Transition A5:A0a (4.4.2.4.1)"
--    ((st = A5) & (nd.end_of_reception &
--                  portRcomp(n, nd.receive_port, RX_DATA_END))) ==>
--      begin
--        nd.arb_timer := 0;
--        change_PHY_state(n, A0);
--      end;
  rule "Transition A5:A0b (4.4.2.4.1)"
    ((st = A5) & portRcomp(n, nd.receive_port, RX_IDLE)) ==>
      begin
        nd.arb_timer := 0;
        change_PHY_state(n, A0);
      end;
  rule "Transition A5:A5a (4.4.2.4.1)"
    ((st = A5) & (nd.end_of_reception &
                  portRcomp(n, nd.receive_port, RX_DATA_PREFIX))) ==>
      begin
        change_PHY_state(n, A5);
      end;
--rule "Transition A5:A5b (4.4.2.4.1)"
```

```
--  ((st = A5) & PH_ARB.req(IMMEDIATE, req_speed)) ==>
--    begin
--      nd.imm_req := true;
--      change_PHY_state(n, A5);
--    end;
--rule "Transition A5:A5c (4.4.2.4.1)"
--  ((st = A5) & PH_ARB.req(ISOCH, req_speed)) ==>
--    begin
--      nd.isoch_req := true;
--      change_PHY_state(n, A5);
--    end;
-- this was never occuring, so speed things up
--    rule "Transition A6:A0 (4.4.2.4.1)"
--      ((st = A6) & nd.end_of_transmission) ==>
--        begin
--          nd.arb_timer := 0;
--          change_PHY_state(n, A0);
--        end;

  end;
  end;
end; -- state machine by node

--end; -- random value

-- INITIALIZATION CODE

startstate
  begin
    for n: IDnum do alias nd: nodes[n] do

-- 4.3.7 Cable PHY node constants
--    nd.NPORT := HIGHPORT + 1;
--    nd.PHY_DELAY_CATEGORY := 0; -- this can only be 0 in current spec
--    nd.PHY_SPEED := S100; -- NYM
--    nd.POWER_CLASS := 0; -- don't care about phys power characteristics
--    nd.CONTENDER := false; -- don't want to model resource manager role

-- 4.3.8 Node variables, table 4-35
--    nd.arb_enable := false;
--    nd.cable_power_active := true;
      nd.force_root := false; -- not going to model this yet
      nd.gap_count := GAP_COUNT_MAX;
      nd.initiated_reset := false; -- should add some rules to allow a
                                   -- node to choose to initiate reset at
                                   -- any time
      nd.link_active := false;
--    nd.more_packets := false;
      nd.parent_port := 0;
      nd.physical_ID := 0;
      nd.receive_port := 0;
      nd.root := false;

-- 4.4 Cable PHY operation, table 4-37, implied internal variables
--    nd.fifo_rd_ptr := 0;
--    nd.fifo_wr_ptr := 0;
--    nd.FIFO_DEPTH := HIGHFIFO_DEPTH;
--    for i: FIFOnum do
--      nd.FIFO[i] := false; -- just makin' it up as I go along
--    end;
```

```
--    nd.waiting_for_data_start := true;
--    nd.rx_speed := S100;

      nd.arb_timer := 0;
--    nd.root_test := false;
      nd.contend_time := 0;
      nd.child_count := 0;
      nd.lowest_unidentified_child := 0;
      nd.all_child_ports_identified := false;
      nd.self_ID_complete := false;
--    nd.requesting_child := 0;
--    nd.force_root := false; -- see above
      nd.end_of_reception := true;
--    nd.link_req_active := false;
--    nd.arb_enable := false; -- redundant, see above
--    for i: Portnum do
--      nd.old_connect[i] := false; -- actually array of NPORT
--    end;
      nd.bus_initialize_active := true;
      nd.gap_count_reset_disable := false;

-- variables implied by the C code or state machines, but never declared
--    nd.cycle_master_req := false;
--    nd.fair_req := false;
--    nd.isoch_req := false;
--    nd.imm_req := false;
--    nd.end_of_transmission := true;

-- variables to emulate the physical aspects of a node
      for i: Portnum do alias pt: nd.ports[i] do
        pt.child := false;
--      pt.connected := false;
        pt.child_ID_complete := false;
--      pt.max_peer_speed := S100;
--      pt.speed_OK := true;
        pt.data.TPA := Z;
        pt.data.TPB := Z;
      end; end;

-- current state of the node
      nd.Pstate := R0;
    end; end;

-- set up connection info -- no longer use this method, see connected()

-- assumes N=2, HIGHPORT=1
--    nodes[0].ports[1].connected := true;
--    nodes[1].ports[0].connected := true;

-- assumes N=3, HIGHPORT=1
--    nodes[0].ports[1].connected := true;
--    nodes[1].ports[0].connected := true;
--    nodes[1].ports[1].connected := true;
--    nodes[2].ports[0].connected := true;

  end;


-- INVARIANTS
```

```
invariant "Do I stay within the expected states?"
  forall n: IDnum do
    nodes[n].Pstate != S3 & -- nodes[n].Pstate != A1 &
    nodes[n].Pstate != A2 & nodes[n].Pstate != A3 &
    nodes[n].Pstate != A4 & nodes[n].Pstate != A6
  end;
```