

C μ LOG Project Final Report

An Entity Interaction Simulation Language

John Demme (jdd2127)
Nishant Shah (nrs2127)
Devesh Dedhia (ddd2121)
Cheng Cheng (cc2999)

Columbia University

December 17, 2008

1 Introduction: C μ LOG

C μ LOG is a logic language designed for entity interaction simulation. It uses a brute force method for solution searching similar to Prolog but uses a syntax similar to C, making it easier on the typical programmer's eyes, and is compatible with some code tools, such as code indenters and Emacs's c-mode.

Simulations in C μ LOG involve a set of entities written in C μ LOG which interact in the simulator. The “environment” entity defines the board on which the “agents” play, and defines the game which the entities play. It is a turn-based simulation during which each agent can look at the contents of the environment and decide which direction it should move. During this decision, the agents can modify their own working memory, thus affecting their decision for the next turn.

Additionally, the C μ LOG interpreter may be invoked separately from the simulator. The stand-alone interpreter searches for all the solutions for “main”, but typically the output of these programs will be from “print” directives specified in the program.

1.1 Application & Features

One uses the language to provide a set of facts and rules, and the program is run by asking a question, which the interpreter attempts to answer using inferences based on the fact and rule set. C μ LOG is designed for simulation, so typically a simulator will ask a given agent program what its next action will be. The agent program then uses C μ LOGs entity interaction features to gather information about its environment and decide what to do. Each agent program can communicate with other programs to find out more information about other agents or its own status in the environment. The simulator stores all the contextual information pertaining to the environment and all of the agents present.

As this language is going to be used for simulating real life agents, we strongly emphasize that the program learn and forget data/rules/information at run-time. For this, similar to “assert” and “retract” of Prolog we have introduced two directives called “learn” and “forget.” In C μ LOG there exist no specific data structures like you would see in Java or Python, however rules and facts can be added to the program dynamically, which allows programs to remember data in a much more natural way since the data simply becomes part of the running code.

The simulator discussed could be modified to be used in other with other simulation environments, such as in a three-dimensional grid simulation with several agents—such as a flight simulation. Alternatively, the interpreter could be used in a real environment like the movement of pick and place robots in a warehouse. The language could be used to define the warehouse environment and agent programs for robots, and a replacement for the simulator would feed live information in to the programs in the form of facts, similarly to how the simulation feeds its state information to agents now.

1.2 Goals

The language and simulator presented here attempts to fulfill the following requirements:

Generic Games are defined mostly by the environment application.

Composable Individual behaviors can be written simply and easily, then combined to obtain high-level actions and reasoning

Declarative Programmers can specify what they want entities to do rather than how

Controlled Communication Data in the system is frequently made up of nearly-atomic bits of data many of which can be used both on their own and composed as complex data. This means that subsets and smaller pieces of data can be communicated between entities without losing meaning.

High-level libraries Due to the flexibility and composability of the language, high-level algorithms-such as path finding-can be easily implemented in libraries, allowing further, domain-specific intelligence to be written in the programs.

2 Tutorial

Logic programming is a kind of computer programming using mathematical logic. Specifically, it is based on the idea of applying a theorem-prover to declarative sentences and deriving implications. Compared with procedural languages, logic programming solves the problem by setting rules with which solutions must fit. We can represent logic programming by the formula:

$$Facts + Rules = Solutions$$

Logic programming languages are inherently high level languages, allowing programmers to specify problems in a declarative manner, leaving some or all of the details of solving the problem to the interpreter.

Both the programming and data structures in both prolog and $C\mu$ LOG can be very simple- such as facts. The relationship between code and data is also of note. $C\mu$ LOG uses the Von Neumann style (vs. Harvard architecture) wherein data is code. It is therefore possible (and inherently necessary) for programs to be introspective and self-modifying. In other words, it is easier for programs to learn and adapt.

2.1 Variables

Variables represent a value to be solved for. They don't have a fixed datatype, but match to the referred type. All variables are scoped to the rule, so that variable solutions can be shared between sub-blocks.

Variables are represented by a dollar sign (\$) then the variable name. The name must start with a letter, and is composed of letters, numbers, and underscores. There is a special variable called the anonymous variable which is represented simply by a question mark (?).

Example variable names:

```
\$foo \$bar\_ \$f1o2o3
```

The following are not valid variables:

```
foo \$\_foo $1bar
```

2.2 Statements

These are conditional statements which give output as true or false only and are frequently used to constrain variables. They are of two types, comparison and evaluation statements.

Comparison statements are used to compare variables against constants:

Example comparisons:

```
\$a>1+3-4; //means that variable 'a' is always greater than 0
\boo <= 5; // means that variable 'boo' is less than or equal to 5
```

Evaluation or eval statements are used to query the program for solutions:

```
boofar(\$s,\$d,7); //from all the possible matches in the program's
//graph it returns various possible values for the pair s and d,
//and constrains those values in their scope appropriately, as
//defined by the block in which the statement is contained
```

2.3 Facts

Facts are terminal nodes of the solution search which are always true. Facts help us define constant information in the program like the position of a wall.

Syntax: `id(parameter1, parameter2);`

Examples:

```
wall(2,3); //This means that a wall is defined for 2 and 3

fire(4,a); //Symbols like 'a' can also be a parameter.
// Here, fire of 4 and 'a' evaluates to be true.
```

2.4 Rules

Rules are similar to facts, but are only conditionally true. These conditions are defined inside a block. The definition or declaration of rules suggests that the solution tree is about to branch out to search for new solutions.

```
syntax: id(parameter1, parameter2....) {conditions}
```

The block is "conditions" in the above syntax. Block can be of 2 types, namely 'AND' and 'OR' block. AND blocks evaluate true iff all the conditions inside the block are true. Similarly, the OR block is true if any one of the conditions is true. If no reduction method is specified (i.e. AND or OR is written), by default AND is used.

To define a OR block we use the following construct:

```
{OR:
  foo();
  bar();
}
```

The AND block is written similarly:

```

wall(2,3) {AND:
    foo();
    bar();
    {OR: barfoo(); foobar();}
}

```

Here “OR: barfoo(); foobar();” is a sub-block. wall(2,3) is true if foo() and bar() are true and if either of barfoo() or foobar() are true.

2.5 Directives

Three interpreter directives are supported; print, learn and forget. print is used to output strings and results during runtime. the learn and forget directives are used for database modification. They function similar to assert and retract of prolog.

Syntax: @directive_name(parameters);

Example:

```

//prints "hello world: " then whatever constraints exist on $foo
@print("hello world:", $foo);

//adds a fact to the database that 'fire' is true for 4,5
@learn(fire(4,5));

//erases the fact from the database that tree is true for 3,9.
@forget(tree(3,9));

```

2.6 Simulator

Now for the user to be able to run a simulation or play a game in $C\mu$, they will have to use a simulator which interacts with the logic engine of the language to produce required results. For demonstration we have done so already. This simulator defines a class of games or simulations described as follows:

The environment is grid based and defined by a $C\mu$ LOG program. It potentially includes obstacles and a goals which the agent must reach, however the game is defined mostly by the environment program. Every object (i.e. agents, walls, switches, goals) in the environment is defined by grid positions. The environment specifies the representations of the entities to the simulator. The simulator re-evaluates the various object rules during each turn when it renders the grid, so the contents of the grid can be dynamically defined based on the state of the simulation or the contents of the program (which can be changed by the program.) For example based on the grid position of the agent the environment might remove or insert a wall. The agent program decides the next move based on previous moves and obstacle data.

The simulation of the agent program is also turn based. Each time the agent makes a move it sends its new coordinates to the simulator. The new coordinates become part of the simulation’s state which are exposed to the environment when it is solved to render the scene.

Example 1:

```
Size(5,5); //defines the grid size of 5 by 5
wall(2,3); //a fact where wall is present at coordinates 2,3
wall(4,2);
goal(3,3); //a fact which defines the goal to be achieved by the player

igo("UP"); //move($dir) would be true for all the values of $dir
           // for which igo($dir) is true

move($dir){
    //causes the interpreter to remove igo("UP") from its database.
    @forget(igo($dir));

    //Fetch the next movement
    igo($dir);
}
```

The output of the above program is:

X

```
. . . . .
. . . . .
. | # . .
x . . | .
. . . . .
```

In the above example, 'size', 'goal', 'wall' and 'move' are keywords for the simulator. Size(5,5) defines the grid in which walls (shown by the pipe symbol) are placed at coordinates (2,3) and (4,2). A goal object (shown by #) is placed at (3,3). The game simulation ends when the agent either hits a wall, moves out of the grid or reaches the goal.

In order to run code through the simulator, put your code in a file with a ".ul" extension (this extension is a convention only) then invoke the simulator, passing it the name of your code file:

```
./simulator mySimulation.ul
```

2.7 Program Modification

Now let us look at example using one of our program modification directives.

Example 2:

```
size(5, 5);
wall(2, 3);
wall(4,2);
```

```

goal(3, 3);
imove("UP");
imove("RIGHT");
imove("RIGHT");
imove("UP");

move($dir) {
    @forget1( imove($dir); );
    imove($dir);
}

```

OUTPUT:

```

==== Turn 1 ====
. . . . .
. . . . .
. | # . .
x . . | .
. . . . .

```

x: Moving RIGHT

```

==== Turn 2 ====
. . . . .
. . . . .
. | # . .
. x . | .
. . . . .

```

x: Moving RIGHT

```

==== Turn 3 ====
. . . . .
. . . . .
. | # . .
. . x | .
. . . . .

```

x: Moving UP

Simulation over: x wins!!!Successfully reach the goal at position (3,3)

In the above code we see a carefully drafted route through the grid can make you win the game. Each step of the simulation is displayed. In this example, the 'imove' facts are used as a stack of moves which are queried for each turn, and removed from the stack after using it. The "@forget1" directive shown in this example removes only one fact from the program instead of all the facts which match the pattern.

2.8 Breakout

Although we can define an agent's actions within the environment program, it is typically more desirable to specify a separate agent file so that multiple agents can operate in the same environment. In the next example, we use a separate agent which queries the environment for the movements it should take—sort of like asking for directions.

Environment Program:

```
size(10, 10);
wall(?, 7);
goal(6, 6);

imove("UP");
imove("RIGHT");
imove("RIGHT");
imove("RIGHT");
imove("RIGHT");
imove("RIGHT");

agent("d", "tests/agents/delg_to_env.ul");
```

Agent Program:

```
move($dir) {
    @print($e, " says move ", $dir);
    $e.@forget( imove($dir) );
    env($e);
    $e.imove($dir);
}
```

3 Language Reference Manual

3.1 Lexical

```
[' ' '\t' '\r' '\n'] WS
"/*"      OPENCOMMENT
"*/"      CLOSECOMMENT
"//"      COMMENT
```



```

'('      LPAREN
')'      RPAREN
'{'      LBRACE
'}'      RBRACE
';'      SEMICOLON
','      COMMA
'+'      PLUS
'-'      MINUS
'*'      TIMES
'/'      DIVIDE
"=="     EQ
'<'      LT
"<="     LEQ
'>'      GT
">="     GEQ
'@'      AT
'.'      DOT
'"'      QUOTE
'?'      QUESTION
'!'      NOT
'$'['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_' ]* Variable
['0'-'9']+ Number
['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_' ]* Identifier

```

3.2 Facts

Facts define factual relationships. They have a very similar syntax to rules, except they have no code block to make them conditionally true. Any query which matches a fact is simply true. Another way to think of facts is as terminal nodes in the solution search.

Each fact is composed of a name, and a comma separated list of parameters, each of which may be a constant, or a variable. Using any variable except the anonymous variable doesn't make much sense in a fact, but is allowable.

Example:

```

foo(4, symA); //Foo of 4 and symA is always true
foo(4, symA, ?); //Foo of 4, symA, and anything (wildcard) is always true
wall(4, 5); //In an environment might mean: there is a wall present at (4,5)

```

Grammar:

```

Fact -> Identifier ( ParamList );
ParamList -> Param | ParamList , Param
Param -> Variable | Number | String | Identifier

```

3.3 Rules

Rules define relationships which are conditionally true. They are similar to facts, but instead of ending with a semicolon, they contain have a block, which defines the conditions upon which the rule should be evaluated as true. Another way to think of a rules is as a node in the solution search which may branch, or be a leaf, depending on the contents of the condition block. Each rule is composed of a name, a comma separated list of parameters, and a block.

Example:

```
foo(4) { bar(5); } //Foo of 4 is true if bar(5) is true
foo(4) { bar(6); } //Foo of 4 is true if bar(6) is true
```

The two above rules are together equivalent to:

```
foo(4) {OR: bar(5); bar(6); }
```

Grammar:

```
Fact -> Identifier ( ParamList ) Block
```

3.4 Variables

Variables represent a value to be solved for. During rule matching, they will match any value or type, but can be constrained in an associated block. All variables are scoped to the rule, so that variable solutions can be shared between subblocks. Variables are represented by a dollar sign (\$) then the variable name. The name must start with a letter, and is composed of letters, numbers, and underscores. There is a special variable called the anonymous variable which is represented simply by a question mark (?). It cannot be referenced in the block, and simply matches anything.

Example:

```
foo($X, $y, $foo_bar, $bar9, ?) { }
```

Grammar:

```
Variable -> $[a-zA-Z][a-zA-Z0-9_]* | ?
```

3.5 Blocks

Blocks contain a list of statements (conditions) to determine truth, and specify a reduction method for those statements. Each block will reduce all of its statements using the same reduction method (usually AND or OR), but may contain sub-blocks. If the reduction method is omitted, AND is assumed. The syntax allows for other reduction methods to be allowed (such as xor, or a user-specified method), however the language does not yet support this.

Examples:

```
{
    foo();
    bar();
}
```

```

}
//True if foo and bar are both true.

```

```

{AND:
    foo();
    bar();
}
//True if foo and bar are both true.

```

```

{OR:
    foo();
    bar();
}
//True if foo or bar are true.

```

```

Grammar:
    Block -> { (Identifier:)? StatementList }
    StatementList -> Statement | StatementList Statement

```

3.6 Statements

Statements are boolean qualifiers which are used inside of blocks. They can be any one of three types: comparisons, evaluations, or blocks. Comparisons are used to constrain variables. Only values of the same type can be compared, and certain comparisons only work on certain types, so comparisons can be used to constrain variables by type. Evals are used to query the program, and have a similar syntax as facts. They can be thought of as a branch in the solution search. Blocks are considered a statement to support sub-blocks. They are evaluated and the reduced result is used. Comparisons and evals are both terminated by semicolons.

Examples:

```

1 < $X <= $Y < 10; // A comparison
range($X, $Y, 7); // An eval
!range($X, $Y, 7); // This must not evaluate to true
{OR: $X > 10; $X < 0; } //A sub-block with two binary comparisons

```

```

Grammar:
    Statement -> Block | Eval ; | Comparison ;
    Eval -> (!)? Identifier ( ExprList );
    ExprList -> Expression | ExprList , Expression
    Comparison -> Expression ComparisonOp Expression | Expression ComparisonOp Comparison
    ComparisonOp -> EQ | NEQ | LT | LEQ | GT | GEQ

```

3.7 Comparisons

Expressions are used to constrain variables. One side of the comparison must be a variable, and the other a constant. Depending on the type of the constant, only certain comparisons are allowed.

Examples:

```
$r < 10; // a comparison
```

Grammar:

```
Comparison -> Expression CompOp Expression
CompOp -> EQ | LT | LEQ | GT | GEQ
Expression -> Number | String | Variable | Expression Op Expression | ( Expression )
Op -> PLUS | MINUS | TIMES | DIVIDE
```

3.8 Types

The following types are supported: integers, strings, symbols, and entities. Strings in *CμLOG* are currently atomic, so no string processing such as splitting, joining, or searching is supported. They are primarily used for interaction with the rest of the system (printing, specifying files, ect.). Symbols are simply identifiers and can only be compared with equals. Entities are used to represent other programs (typically agents) and are used for interaction. In addition to equals and not equals comparison operators, they support the dot operator for interaction (discussed later.)

3.9 Directives

CμLOG supports a special syntax for interpreter directives. This allows programs to interact with the interpreter while avoiding symbol collisions. The syntax is similar to that of a fact's, but an at sign (@) is prepended. Three directives are currently supported: print, learn, and forget. Print is used to output strings, and results of searches during runtime. Learn and forget are discussed in the next section.

Examples:

```
@print("Hello, world!");
```

Grammar:

```
Directive -> @ Identifier ( ParamList );
```

3.10 Program Modification

The two directives learn and forget are used to modify a program at runtime. This is the only way in which *CμLOG* supports non-volatile storage. Learn is used to add a fact to a program, and forget is used to remove a fact. The syntax for these two directives is special, consisting of the usual directive syntax, except contained inside the parenthesis is a fact definition. Any non-anonymous variables in this fact definition are filled in with solutions found for those variables, and the learn or forget is “executed” once for each solution. They are similar to Prolog’s assert and retract.

Examples:

```
@learn( wall(4,5); ); //Remember that there is a wall at (4,5)
@forget( agent(8, 10); ); //Forget about the agent at (8, 10)
```

Grammar:

```
Directive -> @ (learn|forget) ( Fact List );
```

3.11 Interaction- The Dot Operator

If a variable or symbol represents another program (entity), then it supports the dot operator. After appending a dot (.) to the reference, one can put an eval, a learn, or a forget, and that action will take place in the other entity's namespace. This can be used to ask for information from another program (such as the environment program or another agent) or to modify the other program—perhaps to teach another agent, to trick a competitor, or to change the operating environment. Future versions of $C\mu$ LOG could likely support some sort of access rules in the destination program, allowing it to control who is allowed to access what data, and who is allowed to change its program, and how. These access rules could potentially modify any queries or changes, perhaps revealing an entirely fake namespace to the other agent. Such access rules are beyond the scope of $C\mu$ LOG initially, however.

Example:

```
$agent.@learn( wall(4,5); ); //Tell agent2 that there is a wall at (4,5)
env.view($X, $Y, $obj); //Query the environment, find out what is at ($X, $Y)
```

Grammar:

```
DotOp -> Directive | Statement
Dot -> Variable . DotOp | Identifier . DotOp
```

4 Project Plan

We're cowboys—we don't have a friggin' plan!

4.1 Responsibilities

It was the responsibility of each team member to complete and help complete the individual parts of the interpreter. Specifically, initially the scanner and parser were developed by Devesh Dedhia and Nishant Shah. The AST file was done by Cheng Cheng. The interpreter and translator were completed by John Demme. Nishant Shah and Cheng Cheng developed the simulator together. Testing each phase and testing the whole system was not assigned to any particular person as it requires as much man power as available. Testing was done by every group member.

4.2 Timeline

The following were the timelines we decided on at the start of the semester:

Table 1: Project Deadlines

Language Features design	Oct 20
LRM	Oct 22
Scanner/Parser	Nov 5
Translator	Nov 15
Interpreter	Nov 22
Simulator	Nov 27
Cold Freeze	Dec 12
Testing	Dec 18

As we started working on the project, it was soon realized that the above deadline are not what our aim should be as, it is not a start-end process. The development process was more like evolution. So every section was up and running by Nov 15th, i.e. by then we were able to print "hello world" in our language. After that we have been adding features and for that support is needed on every level, including the scanner, parser, ast file, translator, interpreter and the simulator. All members have been simultaneously working on the development and also testing the features at the same time.

4.3 Software Development Environment

The project will be developed on Ubuntu using Objective Caml 3.1.0. The scanner was developed using Ocamllex v3.1.0. The parser was developed using Ocamlyacc v3.1.0. We will use Python to run our tests and compare it with the expected output. Version control, managing multiple revisions of files, not only program source files, but any type of files is done using Subversion. We are using Google Code for issue tracking and Subversion hostin, plus Google groups("pltsim") for communicating within ourselves.

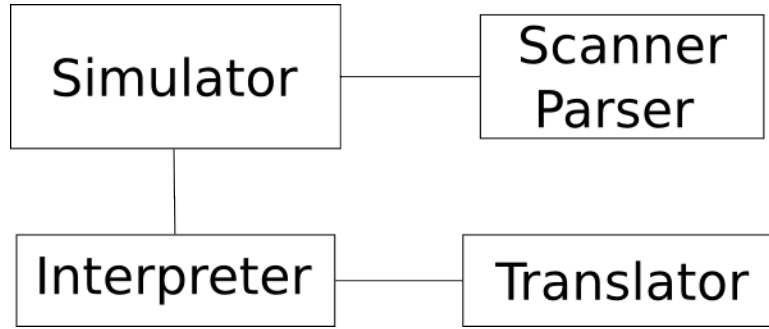
5 Architecture

The language $C\mu$ LOG we have designed will be used for communication between agents and an environment, as well as to determine behavior of said entities. Every agent program communicates with the environment program through a simulator. The simulator runs a $C\mu$ LOG logic solver and interpreter which functions on a set of rules and facts defined and modified by the environment and agents then provides solutions representing the actions to be taken by the agents.

The cmulog interpreter consists of several major blocks: scanner, parser, translator and interpreter. The relationship between these components is demonstrated in figure *. The simulator loads each program by reading in each necessary ".ul" file through the scanner and parser, resulting in an AST. The each AST is then passed into the interpreter, resulting in a database of rules and facts in the interpreter's internal format. The interpreter does not operator directly out of the AST, however: it uses the translator to convert the AST to a "translated syntax tree" (TST) first.

While the AST directly represents the structure of a $C\mu$ LOG, there are a number of static transformations which do not change the meaning of the program, but make interpreting it much easier for the interpreter. The TST represents a simpler version of the program. The translator removes all the variable names and

Figure 1: Architecture Block Diagram



indexes them to a list and then each of them are identified by the number rather than the name. It also partitions all the statements with and without side-effects and runs all the ones without side-effects once for each solution. It performs all possible arithmetic to reduce each statement into its simplest form. It brings the unknown variable to the leftmost side by making all the necessary changes. For $(3 + 4 > \$x - 1)$ will reduce to $(\$x < 6)$. Lastly, all the static semantic checking is also done in the translator.

Using the database reference returned from the interpreter to the simulator, the simulator (or any other program for that matter) can query the database using the interpreter's "query" method, passing in the name to query, and the number of parameters. From this query method, the solution solver lazily evaluates the query by returning either "NoSolution" or "Solution". "NoSolution" indicates the all the results have already been returned. "Solution" is composed of a list of constraints (one per parameter) and a function to generate the next solution. Each solution is computed when the next function is called, so if there are an infinite number of solutions, the query function will not block forever. The caller may iterate through all of the solutions, or use only the first one.

Finally, the simulator uses the interpreter to query various terms for each turn, modifying its state and printing the output. For instance, it queries "wall" with two parameters (for each coordinate) each turn, iterates through all of the solutions, and puts a wall at each solution. For each agent, it queries the "move" term and uses only the first solution to move each agent. Before the first move, the simulator even queries the environment for the "agent" term to get the location of the agent program and its symbol!

Other programs such as the stand-alone "culog.ml" interpreter use the same interpreter interface to parse programs into databases and query these programs using different terms to invoke different behaviors. The stand-alone interpreter queries and "main" term and iterates through the results, and is generally used for testing of the interpreter. Other programs could use their own terms to generate different behaviors.

6 Test Plan

* Show two or three representative source language programs along with the target language program generated for each * Explain why and how these test cases were chosen

6.1 Testing Script

The python script shown in listing 47 is used to run our tests. Each test can be one of three different types: a parsing test, an interpreter test, or a full simulation test.

Listing 1: Testing Script

```
1  #!/usr/bin/python
2
3  import os.path
4  import os
5  import glob
6  import sys
7
8  def run_and_compare(cmd, testOut):
9      pipe = os.popen(cmd)
10     to = open(testOut)
11     for line in pipe.readlines():
12         toLine = to.readline()
13         if toLine != line:
14             to.close()
15             pipe.close()
16             return False
17
18     pipe.close()
19     if to.readline() != '':
20         to.close()
21         return False
22     to.close()
23     return True
24
25 tests = glob.glob("tests/*.ul")
26 tests.sort()
27 for test in tests:
28     testOut = test.replace(".ul", ".out")
29     sys.stdout.write("Running %-35s..." % (test))
30     sys.stdout.flush()
31
32     if not os.path.exists(testOut):
33         print "No Output"
34         continue
35
36     if (test.find("/pr") != -1):
```



```

37         prog = "print <"
38     else :
39         if (test.find("/sim") != -1):
40             prog = "simulator"
41         else :
42             prog = "culog"
43     if run_and_compare("./%s %s 2>&1" % (prog, test), testOut):
44         print "OK"
45     else :
46         print "FAIL!"

```

Listing 1: Testing Script

6.2 Testing Results

As of the writing of this report, the testing results are shown below. Each of the test inputs and outputs can be found in the test cases appendix.

Running tests/andTest.ul	...OK
Running tests/facts.ul	...OK
Running tests/learnForget1.ul	...OK
Running tests/main.ul	...OK
Running tests/main_fall_through.ul	...OK
Running tests/mult-main.ul	...OK
Running tests/neq.ul	...FAIL!
Running tests/not1.ul	...OK
Running tests/plist-twice.ul	...OK
Running tests/printer_test.ul	...OK
Running tests/prsimple.ul	...OK
Running tests/prstrings.ul	...OK
Running tests/range.ul	...OK
Running tests/sim_dot1.ul	...OK
Running tests/sim_dot2.ul	...OK
Running tests/sim_my_loc.ul	...OK
Running tests/sim_ndot2.ul	...OK
Running tests/sim_two_test.ul	...OK
Running tests/simulator_test.ul	...OK
Running tests/sprint1.ul	...OK

7 Lessons Learned

A Appendix: Test Cases

A.1 andTest

Listing 2: andTest Test Case Input

```
1 wall(4,5);
2 wall(6,7);
3
4 wall($x, 5) {AND:
5     $x < 7;
6     $x > 2;
7 }
8
9 wall($x, $y) {AND:
10     $x < 15;
11     $y < 2;
12 }
13
14 main() {
15     @print("(" , $x, ", ", $y, ")");
16     wall($x, $y);
17 }
```

Listing 2: andTest Test Case Input

Listing 3: andTest Test Case Output

```
1 (4,5)
2
3   ^^ Solution   ^^
4
5 (6,7)
6
7   ^^ Solution   ^^
8
9 (2..7,5)
10
11  ^^ Solution   ^^
12
13 (<15,<2)
14
15  ^^ Solution   ^^
16
17 No more solutions
```

Listing 3: andTest Test Case Output

A.2 facts

Listing 4: facts Test Case Input

```
1 foo(4,4);
2
```

```

3  foo(symA);
4
5  bar($name);
6
7  foo(){ wall(3);}
8
9  main() {OR:
10     @print($a, " ", $b);
11     foo($a);
12     foo($a, $b);
13     bar("$a");
14 }

```

Listing 4: facts Test Case Input

Listing 5: facts Test Case Output

```

1  symA Any
2
3  ^^^ Solution ^^^
4
5  4 4
6
7  ^^^ Solution ^^^
8
9  Any Any
10
11 ^^^ Solution ^^^
12
13 No more solutions

```

Listing 5: facts Test Case Output

A.3 learnForget1

Listing 6: learnForget1 Test Case Input

```

1  stack("s1");
2  stack("s2");
3  stack("s3");
4
5  f() {
6     @print("Removing: ", $s);
7     @forget( stack($s); );
8
9     $s == "s1";
10 }
11
12 l() {
13     @print("Learning: ", $s);
14     @learn( stack($s); );
15

```

```

16     $s == 6;
17 }
18
19 main() {OR:
20     @print($s);
21     f();
22     l();
23     stack($s);
24 }

```

Listing 6: learnForget1 Test Case Input

Listing 7: learnForget1 Test Case Output

```

1 Removing: 's1'
2 Any
3
4   ^^^ Solution   ^^^
5
6 Learning: 6
7 6
8
9   ^^^ Solution   ^^^
10
11 's2'
12
13   ^^^ Solution   ^^^
14
15 's3'
16
17   ^^^ Solution   ^^^
18
19 No more solutions

```

Listing 7: learnForget1 Test Case Output

A.4 main

Listing 8: main Test Case Input

```

1 /* test to find the no. solutions one gets under given constraints*/
2
3 wall(3,4);
4 wall(4,8);
5 wall(6,8);
6 wall($X,$Y) {AND:
7     $X>=10;
8     $X<=15;
9     $Y<=8;
10    $Y>=2;
11 }
12 wall(){}

```

```

13
14 main ()
15 {
16     @print(" Wall: ", $x, ", ", $y);
17     wall($x, $y);
18 }

```

Listing 8: main Test Case Input

Listing 9: main Test Case Output

```

1 Wall: 3,4
2
3   ^^^ Solution   ^^^
4
5 Wall: 4,8
6
7   ^^^ Solution   ^^^
8
9 Wall: 6,8
10
11  ^^^ Solution   ^^^
12
13 Wall: 9..16,1..9
14
15  ^^^ Solution   ^^^
16
17 No more solutions

```

Listing 9: main Test Case Output

A.5 main_fall_through

Listing 10: main_fall_through Test Case Input

```

1 main() {
2     @print("ERROR");
3     noexist($y);
4 }
5
6 main() {
7     @print(" Success");
8 }

```

Listing 10: main_fall_through Test Case Input

Listing 11: main_fall_through Test Case Output

```

1 Success
2
3   ^^^ Solution   ^^^
4

```

5 No more solutions

Listing 11: main_fall_through Test Case Output

A.6 mult-main

Listing 12: mult-main Test Case Input

```
1 main(a,b);
2
3 main()
4 {
5     main(a,b);
6 }
7
8 main();
```

Listing 12: mult-main Test Case Input

Listing 13: mult-main Test Case Output

```
1
2   ^^^  Solution  ^^^
3
4
5   ^^^  Solution  ^^^
6
7 No more solutions
```

Listing 13: mult-main Test Case Output

A.7 neq

Listing 14: neq Test Case Input

```
1 main() {
2     @print($x);
3     $x != 8;
4 }
```

Listing 14: neq Test Case Input

Listing 15: neq Test Case Output

```
1 !=8
2
3   ^^^  Solution  ^^^
```

Listing 15: neq Test Case Output

A.8 not1

Listing 16: not1 Test Case Input

```
1 wall(10);
2
3 wall($y) {
4     4 < $y;
5     $y < 6;
6 }
7
8 main() {AND:
9     @print($y);
10    {AND:
11        $y > 1;
12        $y < 15;
13    }
14    !wall($y);
15 }
```

Listing 16: not1 Test Case Input

Listing 17: not1 Test Case Output

```
1 11..15
2
3 ^^^ Solution ^^^
4
5 1..4
6
7 ^^^ Solution ^^^
8
9 6..9
10
11 ^^^ Solution ^^^
12
13 No more solutions
```

Listing 17: not1 Test Case Output

A.9 plist-twice

Listing 18: plist-twice Test Case Input

```
1 foo($y, $u, "hello", $y, 6);
```

Listing 18: plist-twice Test Case Input

Listing 19: plist-twice Test Case Output

```
1 Fatal error: exception Failure("You cannot list the same variable twice in a parameter list")
```

Listing 19: plist-twice Test Case Output

A.10 printer_test

Listing 20: printer_test Test Case Input

```
1  /*This is a test case */
2
3  /*
4    environ1.ul
5    The environment being operated in is the list of the
6    simulator's facts, then the facts and rules below
7  */
8
9  /*This is a sample 15x15 environment*/
10 size(15,15);
11
12 @attach("geometry.ul");
13 /*A wall segment at (5,5)*/
14 wall(5,5);
15 /*A wall segment from (1,10) to (5,10)*/
16 wall($X,$Y) {
17     $X > 0;
18     $X <= 5;
19     $Y == 10;
20 }
21 /*A wall that only appears when an agent is at (1,2) or (1,4)*/
22 wall(1,3) {OR:
23     object(1, 2, agent1);
24     object(1, 4, agent1);
25 }
26
27 /* A wall that only appears when an agent is at (2,2) or (2,4),
28    but stays there after the agent leaves*/
29 wall(2,3) {
30     {OR:
31         object(2, 2, agent1);
32         object(2, 4, agent1);
33     }
34     @learn( wall(2,3); );
35 }
36 /* An invisible switch appears at (3,3) and dissolves the wall
37    at (2,3) when the agent steps on it */
38 object(3, 3, switchObject) {
39     object(3, 3, agent1);
40     @forget( wall(2,3); );
41 }
42 /*The objective is at (15,15)*/
43 object($x, $y, wallObject) {
44     wall($x, $y);
45 }
46
47 /* These are the icons for each object*/
48 repr(wallObject, "pix/wall.png");
```

```

49 repr(switchObject, "pix/switch.png");
50 repr(goalObject, "pix/goal.png");
51
52 /* Agent success if it reaches (15, 15)*/
53 finish(SuccessAgent1) {
54     object(15, 15, agent1);
55 }
56
57 finish(SuccessAgent2) {
58     object(13, 15, agent2);
59 }
60
61 /* Fail the simulation if the agent hits a wall*/
62 finish(Failure) {
63     object($x, $y, agent1);
64     wall($x, $y);
65 }
66 /* Load agent1*/
67 repr(agent1, "agent1.sl");
68
69 /*Place at (1,1) then forget about the agent,
70 so the simulator will take over agent management*/
71 object(1, 1, agent1) {
72     @forget( object(1, 1, agent1); );
73 }
74
75 viewRange($x, $y, $viewer, $obj, $rangeMax) {
76     object($ViewerX, $ViewerY, $viewer);
77     range($x, $y, $ViewerX, $ViewerY, $range);
78     0 <= $range ;
79     $range <= $rangeMax;
80     object($x, $y, $obj);
81 }
82 viewAccessRule(agent1);
83 /*How far can agents see?
84 This is defined in geometry.ul*/
85 view($x, $y, $viewer, $obj) {
86     viewRange($x, $y, $viewer, $obj, 1);
87 }
88
89 repr(agent2, "agent2.ul");
90
91 peers(agent1);
92 peers(agent2);

```

Listing 20: printer_test Test Case Input

Listing 21: printer_test Test Case Output

```

1 size(15,15);
2 @attach("geometry.ul");
3 wall(5,5);

```

```

4  wall($X,$Y) {AND:
5  $X>0;
6  $X<=5;
7  $Y=10;
8  }
9  wall(1,3) {OR:
10 object(1,2,agent1);
11 object(1,4,agent1);
12 }
13 wall(2,3) {AND:
14 {OR:
15 object(2,2,agent1);
16 object(2,4,agent1);
17 }
18 @learn(wall(2,3);)
19 }
20 object(3,3,switchObject) {AND:
21 object(3,3,agent1);
22 @forget(wall(2,3);)
23 }
24 object($x,$y,wallObject) {AND:
25 wall($x,$y);
26 }
27 repr(wallObject,"pix/wall.png");
28 repr(switchObject,"pix/switch.png");
29 repr(goalObject,"pix/goal.png");
30 finish(SuccessAgent1) {AND:
31 object(15,15,agent1);
32 }
33 finish(SuccessAgent2) {AND:
34 object(13,15,agent2);
35 }
36 finish(Failure) {AND:
37 object($x,$y,agent1);
38 wall($x,$y);
39 }
40 repr(agent1,"agent1.sl");
41 object(1,1,agent1) {AND:
42 @forget(object(1,1,agent1);)
43 }
44 viewRange($x,$y,$viewer,$obj,$rangeMax) {AND:
45 object($ViewerX,$ViewerY,$viewer);
46 range($x,$y,$ViewerX,$ViewerY,$range);
47 0<=$range;
48 $range<=$rangeMax;
49 object($x,$y,$obj);
50 }
51 viewAccessRule(agent1);
52 view($x,$y,$viewer,$obj) {AND:
53 viewRange($x,$y,$viewer,$obj,1);
54 }

```

```

55 repr(agent2,"agent2.ul");
56 peers(agent1);
57 peers(agent2);

```

Listing 21: printer_test Test Case Output

A.11 prsimple

Listing 22: prsimple Test Case Input

```

1 foo(4,5);
2 @learn("$x");
3 foo(){OR:
4 @learn(wall(4,5));
5 //@forget(wall(2,3));
6 }

```

Listing 22: prsimple Test Case Input

Listing 23: prsimple Test Case Output

```

1 foo(4,5);
2 @learn("$x");
3 foo(){OR:
4 @learn(wall(4,5));
5 }

```

Listing 23: prsimple Test Case Output

A.12 prstrings

Listing 24: prstrings Test Case Input

```

1 foo(4,"asdf");
2 foo("#@$%");
3 foo(4,"//as oiwer//2356 asdoiulkj ouweoij:::popi%%%^_)+(*^%&$$$^%&(*_^%&$@#^%$&");

```

Listing 24: prstrings Test Case Input

Listing 25: prstrings Test Case Output

```

1 foo(4,"asdf");
2 foo("#@$%");
3 foo(4,"//as oiwer//2356 asdoiulkj ouweoij:::popi%%%^_)+(*^%&$$$^%&(*_^%&$@#^%$&");

```

Listing 25: prstrings Test Case Output

A.13 range

Listing 26: range Test Case Input

```
1  /* test to find how the range cases work*/
2
3  foo($x,$y) {
4      10 >= $x;
5      1 <= $x;
6      $y>9;
7      19 >$y;
8  }
9
10 bar($z) {
11     $z < 50;
12     10 < $z;
13 }
14
15 main() {
16     {OR:
17         foo($x,$y);
18         bar($z);
19         @print("the solutions for $x ",$x,
20             " the solutions for $y ",$y,
21             " the solutions for $z ",$z);
22     }
23
24
25 }
26 /* conclusions: If an infinte range is given the interpreter gives no solution
27                 The values returned are exclusive in a range
28                 so for  10 >= $x;
29                     1 <= $x;
30                 the interpreter returns a range 0..11
31 */
```

Listing 26: range Test Case Input

Listing 27: range Test Case Output

```
1 the solutions for $x 0..11 the solutions for $y 9..19 the solutions for $z Any
2
3   ^^ Solution  ^^
4
5 the solutions for $x Any the solutions for $y Any the solutions for $z 10..50
6
7   ^^ Solution  ^^
8
9 No more solutions
```

Listing 27: range Test Case Output

A.14 sim_dot1

Listing 28: sim_dot1 Test Case Input

```

1 size(10, 10);
2 wall(?, 7);
3 goal(6, 6);
4
5 imove("UP");
6 imove("RIGHT");
7 imove("RIGHT");
8 imove("RIGHT");
9 imove("RIGHT");
10 imove("RIGHT");
11
12 move($dir) {
13     imove($dir);
14 }
15
16 move("UP");
17
18 agent("d", "tests/agents/dot1.ul");

```

Listing 28: sim_dot1 Test Case Input

Listing 29: tests/agents/dot1.ul

```

1 move($dir) {
2     $e.@forget1( imove($dir); );
3     @print($e, " says move ", $dir);
4     env($e);
5     $e.move($dir);
6 }

```

Listing 29: tests/agents/dot1.ul

Listing 30: sim_dot1 Test Case Output

```

1 Agent says move 'UP'
2 d: Moving UP
3
4 == Turn 1 ==
5 . . . . .
6 . . . . .
7 . . . . .
8 | | | | | | | | |
9 . . . . . # . . . .
10 . . . . .
11 . . . . .
12 . . . . .
13 d . . . . .
14 . . . . .
15

```

```

16 Agent says move 'RIGHT'
17 d: Moving RIGHT
18
19 ===== Turn 2 =====
20 . . . . .
21 . . . . .
22 . . . . .
23 | | | | | | | | |
24 . . . . # . . . .
25 . . . . .
26 . . . . .
27 . . . . .
28 . d . . . . .
29 . . . . .
30
31 Agent says move 'RIGHT'
32 d: Moving RIGHT
33
34 ===== Turn 3 =====
35 . . . . .
36 . . . . .
37 . . . . .
38 | | | | | | | | |
39 . . . . # . . . .
40 . . . . .
41 . . . . .
42 . . . . .
43 . . d . . . . .
44 . . . . .
45
46 Agent says move 'RIGHT'
47 d: Moving RIGHT
48
49 ===== Turn 4 =====
50 . . . . .
51 . . . . .
52 . . . . .
53 | | | | | | | | |
54 . . . . # . . . .
55 . . . . .
56 . . . . .
57 . . . . .
58 . . . d . . . . .
59 . . . . .
60
61 Agent says move 'RIGHT'
62 d: Moving RIGHT
63
64 ===== Turn 5 =====
65 . . . . .
66 . . . . .

```

```

67 . . . . . . . . . .
68 | | | | | | | | | |
69 . . . . . # . . . .
70 . . . . . . . . . .
71 . . . . . . . . . .
72 . . . . . . . . . .
73 . . . . d . . . . .
74 . . . . . . . . . .
75
76 Agent says move 'RIGHT'
77 d: Moving RIGHT
78
79 ===== Turn 6 =====
80 . . . . . . . . . .
81 . . . . . . . . . .
82 . . . . . . . . . .
83 | | | | | | | | | |
84 . . . . . # . . . .
85 . . . . . . . . . .
86 . . . . . . . . . .
87 . . . . . . . . . .
88 . . . . . d . . . .
89 . . . . . . . . . .
90
91 Agent says move 'UP'
92 d: Moving UP
93
94 ===== Turn 7 =====
95 . . . . . . . . . .
96 . . . . . . . . . .
97 . . . . . . . . . .
98 | | | | | | | | | |
99 . . . . . # . . . .
100 . . . . . . . . . .
101 . . . . . . . . . .
102 . . . . . d . . . .
103 . . . . . . . . . .
104 . . . . . . . . . .
105
106 Agent says move 'UP'
107 d: Moving UP
108
109 ===== Turn 8 =====
110 . . . . . . . . . .
111 . . . . . . . . . .
112 . . . . . . . . . .
113 | | | | | | | | | |
114 . . . . . # . . . .
115 . . . . . . . . . .
116 . . . . . d . . . .
117 . . . . . . . . . .

```



```

118 . . . . .
119 . . . . .
120
121 Agent says move 'UP'
122 d: Moving UP
123
124 == Turn 9 ==
125 . . . . .
126 . . . . .
127 . . . . .
128 | | | | | | | | |
129 . . . . # . . . .
130 . . . . d . . . .
131 . . . . .
132 . . . . .
133 . . . . .
134 . . . . .
135
136 Agent says move 'UP'
137 d: Moving UP
138
139 Simulation over: d wins!!! Successfully reach the goal at position (6,6)

```

Listing 30: sim_dot1 Test Case Output

A.15 sim_dot2

Listing 31: sim_dot2 Test Case Input

```

1 size(10, 10);
2 wall(?, 7);
3 goal(6, 6);
4
5 imove("UP");
6 imove("RIGHT");
7 imove("RIGHT");
8 imove("RIGHT");
9 imove("RIGHT");
10 imove("RIGHT");
11
12 move($dir) {
13     @forget1( imove($dir); );
14     imove($dir);
15 }
16
17 move("UP");
18
19 agent("d", "tests/agents/delg_to_env.ul");

```

Listing 31: sim_dot2 Test Case Input

Listing 32: tests/agents/delg_to_env.ul

```

1 env(yo);
2
3 move($dir) {
4     @print($e, " says move ", $dir);
5     env($e);
6     $e.move($dir);
7 }

```

Listing 32: tests/agents/delg_to_env.ul

Listing 33: sim_dot2 Test Case Output

```

1 Agent says move 'UP'
2 d: Moving UP
3
4 ===== Turn 1 =====
5 . . . . .
6 . . . . .
7 . . . . .
8 | | | | | | | | |
9 . . . . . # . . . . .
10 . . . . .
11 . . . . .
12 . . . . .
13 d . . . . .
14 . . . . .
15
16 Agent says move 'RIGHT'
17 d: Moving RIGHT
18
19 ===== Turn 2 =====
20 . . . . .
21 . . . . .
22 . . . . .
23 | | | | | | | | |
24 . . . . . # . . . . .
25 . . . . .
26 . . . . .
27 . . . . .
28 . d . . . . .
29 . . . . .
30
31 Agent says move 'RIGHT'
32 d: Moving RIGHT
33
34 ===== Turn 3 =====
35 . . . . .
36 . . . . .
37 . . . . .
38 | | | | | | | | |
39 . . . . . # . . . . .

```

```

40 . . . . .
41 . . . . .
42 . . . . .
43 . . d . . . . .
44 . . . . .
45
46 Agent says move 'RIGHT'
47 d: Moving RIGHT
48
49 ===== Turn 4 =====
50 . . . . .
51 . . . . .
52 . . . . .
53 | | | | | | | | |
54 . . . . # . . . .
55 . . . . .
56 . . . . .
57 . . . . .
58 . . . d . . . . .
59 . . . . .
60
61 Agent says move 'RIGHT'
62 d: Moving RIGHT
63
64 ===== Turn 5 =====
65 . . . . .
66 . . . . .
67 . . . . .
68 | | | | | | | | |
69 . . . . # . . . .
70 . . . . .
71 . . . . .
72 . . . . .
73 . . . . d . . . .
74 . . . . .
75
76 Agent says move 'RIGHT'
77 d: Moving RIGHT
78
79 ===== Turn 6 =====
80 . . . . .
81 . . . . .
82 . . . . .
83 | | | | | | | | |
84 . . . . # . . . .
85 . . . . .
86 . . . . .
87 . . . . .
88 . . . . d . . . .
89 . . . . .
90

```

```

91 Agent says move 'UP'
92 d: Moving UP
93
94 ===== Turn 7 =====
95 . . . . .
96 . . . . .
97 . . . . .
98 | | | | | | | | |
99 . . . . # . . . .
100 . . . . .
101 . . . . .
102 . . . . d . . . .
103 . . . . .
104 . . . . .
105
106 Agent says move 'UP'
107 d: Moving UP
108
109 ===== Turn 8 =====
110 . . . . .
111 . . . . .
112 . . . . .
113 | | | | | | | | |
114 . . . . # . . . .
115 . . . . .
116 . . . . d . . . .
117 . . . . .
118 . . . . .
119 . . . . .
120
121 Agent says move 'UP'
122 d: Moving UP
123
124 ===== Turn 9 =====
125 . . . . .
126 . . . . .
127 . . . . .
128 | | | | | | | | |
129 . . . . # . . . .
130 . . . . d . . . .
131 . . . . .
132 . . . . .
133 . . . . .
134 . . . . .
135
136 Agent says move 'UP'
137 d: Moving UP
138
139 Simulation over: d wins!!! Successfully reach the goal at position (6,6)

```

Listing 33: sim.dot2 Test Case Output

A.16 sim_my_loc

Listing 34: sim_my_loc Test Case Input

```

1 size(10, 10);
2 wall(?, 7);
3 goal(6, 6);
4
5 move("UP") {
6     @print("My location: ", $x, ", ", $y);
7     loc($x, $y);
8     $y < 6;
9 }
10
11 move("RIGHT");
12
13 main() {
14     @print($dir);
15     move($dir);
16 }

```

Listing 34: sim_my_loc Test Case Input

Listing 35: sim_my_loc Test Case Output

```

1 My location: 1,1
2 x: Moving UP
3
4 ===== Turn 1 =====
5 . . . . .
6 . . . . .
7 . . . . .
8 | | | | | | | |
9 . . . . # . . . .
10 . . . . .
11 . . . . .
12 . . . . .
13 x . . . . .
14 . . . . .
15
16 My location: 1,2
17 x: Moving UP
18
19 ===== Turn 2 =====
20 . . . . .
21 . . . . .
22 . . . . .
23 | | | | | | | |
24 . . . . # . . . .
25 . . . . .
26 . . . . .
27 x . . . . .

```

```

28 . . . . .
29 . . . . .
30
31 My location: 1,3
32 x: Moving UP
33
34 ===== Turn 3 =====
35 . . . . .
36 . . . . .
37 . . . . .
38 | | | | | | | | |
39 . . . . . # . . . . .
40 . . . . .
41 x . . . . .
42 . . . . .
43 . . . . .
44 . . . . .
45
46 My location: 1,4
47 x: Moving UP
48
49 ===== Turn 4 =====
50 . . . . .
51 . . . . .
52 . . . . .
53 | | | | | | | | |
54 . . . . . # . . . . .
55 x . . . . .
56 . . . . .
57 . . . . .
58 . . . . .
59 . . . . .
60
61 My location: 1,5
62 x: Moving UP
63
64 ===== Turn 5 =====
65 . . . . .
66 . . . . .
67 . . . . .
68 | | | | | | | | |
69 x . . . . # . . . .
70 . . . . .
71 . . . . .
72 . . . . .
73 . . . . .
74 . . . . .
75
76 x: Moving RIGHT
77
78 ===== Turn 6 =====

```

```

79 . . . . .
80 . . . . .
81 . . . . .
82 | | | | | | | | | |
83 . x . . . # . . . .
84 . . . . .
85 . . . . .
86 . . . . .
87 . . . . .
88 . . . . .
89
90 x: Moving RIGHT
91
92 ===== Turn 7 =====
93 . . . . .
94 . . . . .
95 . . . . .
96 | | | | | | | | | |
97 . . x . . # . . . .
98 . . . . .
99 . . . . .
100 . . . . .
101 . . . . .
102 . . . . .
103
104 x: Moving RIGHT
105
106 ===== Turn 8 =====
107 . . . . .
108 . . . . .
109 . . . . .
110 | | | | | | | | | |
111 . . . x . # . . . .
112 . . . . .
113 . . . . .
114 . . . . .
115 . . . . .
116 . . . . .
117
118 x: Moving RIGHT
119
120 ===== Turn 9 =====
121 . . . . .
122 . . . . .
123 . . . . .
124 | | | | | | | | | |
125 . . . . x # . . . .
126 . . . . .
127 . . . . .
128 . . . . .
129 . . . . .

```

```

130 . . . . .
131
132 x: Moving RIGHT
133
134 Simulation over: x wins!!! Successfully reach the goal at position (6,6)

```

Listing 35: sim_my_loc Test Case Output

A.17 sim_ndot2

Listing 36: sim_ndot2 Test Case Input

```

1 size(3,3);
2
3 wall(3,3);
4
5 goal(3,1);
6
7 disallow("DOWN");
8 disallow("LEFT");
9 disallow("UP");
10
11 agent("x", "tests/agents/ndot2.ul");

```

Listing 36: sim_ndot2 Test Case Input

Listing 37: tests/agents/ndot2.ul

```

1 move($dir) {
2     {OR:
3         $dir == "UP";
4         $dir == "DOWN";
5         $dir == "LEFT";
6         $dir == "RIGHT";
7     }
8
9     @print("Moving: ", $dir);
10    env($e);
11    !$e.disallow($dir);
12 }

```

Listing 37: tests/agents/ndot2.ul

Listing 38: sim_ndot2 Test Case Output

```

1 Moving: 'RIGHT'
2 x: Moving RIGHT
3
4 ===== Turn 1 =====
5 . . |
6 . . .
7 . x #

```



```

8
9 Moving: 'RIGHT'
10 x: Moving RIGHT
11
12 Simulation over: x wins!!! Successfully reach the goal at position (3,1)

```

Listing 38: sim_ndot2 Test Case Output

A.18 sim_two_test

Listing 39: sim_two_test Test Case Input

```

1 /*this is test of simulator , it output the walls and trace of agent into agent1.dat*/
2
3 /* ENV CODE... PLAYER—DON'T CHANGE OR LOOK AT ME!!! */
4 goal(4,4);
5 size(20,20);
6 wall(12,4);
7 wall(4,9);
8 wall(6,8);
9 wall($X,$Y) {AND:
10     $X>=4;
11     $X<=6;
12     $Y<=15;
13     $Y>=10;
14 }
15
16
17 agent("x", "tests/agents/agent1.ul");
18 agent("y", "tests/agents/agent2.ul");

```

Listing 39: sim_two_test Test Case Input

Listing 40: tests/agents/agent1.ul

```

1 imove("UP");
2 imove("UP");
3 imove("RIGHT");
4 imove("RIGHT");
5 imove("RIGHT");
6 imove("RIGHT");
7
8 move($dir) {
9     @forget1( imove($dir); );
10    imove($dir);
11 }
12
13 move("DOWN");
14
15 main() {
16     @print($d);
17     move($d);

```

18 }

Listing 40: tests/agents/agent1.ul

Listing 41: tests/agents/agent2.ul

```
1  imove("RIGHT");
2  imove("RIGHT");
3  imove("LEFT");
4  imove("UP");
5  imove("RIGHT");
6  imove("UP");
7  imove("UP");
8  imove("RIGHT");
9
10 move($dir) {
11     @forget1( imove($dir); );
12     imove($dir);
13 }
14
15 move("DOWN");
```

Listing 41: tests/agents/agent2.ul

Listing 42: sim_two_test Test Case Output

```
1  x: Moving UP
2  y: Moving RIGHT
3
4  ===== Turn 1 =====
5  . . . . .
6  . . . . .
7  . . . . .
8  . . . . .
9  . . . . .
10 . . . | | | . . . . .
11 . . . | | | . . . . .
12 . . . | | | . . . . .
13 . . . | | | . . . . .
14 . . . | | | . . . . .
15 . . . | | | . . . . .
16 . . . | . . . . .
17 . . . . . | . . . . .
18 . . . . .
19 . . . . .
20 . . . . .
21 . . . # . . . . . | . . . . .
22 . . . . .
23 x . . . . .
24 . y . . . . .
25
26 x: Moving UP
```

```

27 y: Moving RIGHT
28
29 ===== Turn 2 =====
30 . . . . .
31 . . . . .
32 . . . . .
33 . . . . .
34 . . . . .
35 . . . | | | . . . . .
36 . . . | | | . . . . .
37 . . . | | | . . . . .
38 . . . | | | . . . . .
39 . . . | | | . . . . .
40 . . . | | | . . . . .
41 . . . | . . . . .
42 . . . . . | . . . . .
43 . . . . .
44 . . . . .
45 . . . . .
46 . . . # . . . . . | . . . . .
47 x . . . . .
48 . . . . .
49 . . y . . . . .
50
51 x: Moving RIGHT
52 y: Moving LEFT
53
54 ===== Turn 3 =====
55 . . . . .
56 . . . . .
57 . . . . .
58 . . . . .
59 . . . . .
60 . . . | | | . . . . .
61 . . . | | | . . . . .
62 . . . | | | . . . . .
63 . . . | | | . . . . .
64 . . . | | | . . . . .
65 . . . | | | . . . . .
66 . . . | . . . . .
67 . . . . . | . . . . .
68 . . . . .
69 . . . . .
70 . . . . .
71 . . . # . . . . . | . . . . .
72 . x . . . . .
73 . . . . .
74 . y . . . . .
75
76 x: Moving RIGHT
77 y: Moving UP

```

```

78
79  ===== Turn 4 =====
80  . . . . .
81  . . . . .
82  . . . . .
83  . . . . .
84  . . . . .
85  . . . | | | . . . . .
86  . . . | | | . . . . .
87  . . . | | | . . . . .
88  . . . | | | . . . . .
89  . . . | | | . . . . .
90  . . . | | | . . . . .
91  . . . | . . . . .
92  . . . . . | . . . . .
93  . . . . .
94  . . . . .
95  . . . . .
96  . . . # . . . . . | . . . . .
97  . . x . . . . .
98  . y . . . . .
99  . . . . .
100
101 x: Moving RIGHT
102 y: Moving RIGHT
103
104  ===== Turn 5 =====
105  . . . . .
106  . . . . .
107  . . . . .
108  . . . . .
109  . . . . .
110  . . . | | | . . . . .
111  . . . | | | . . . . .
112  . . . | | | . . . . .
113  . . . | | | . . . . .
114  . . . | | | . . . . .
115  . . . | | | . . . . .
116  . . . | . . . . .
117  . . . . . | . . . . .
118  . . . . .
119  . . . . .
120  . . . . .
121  . . . # . . . . . | . . . . .
122  . . . x . . . . .
123  . . y . . . . .
124  . . . . .
125
126 x: Moving RIGHT
127 y: Moving UP
128

```

```

129  ===== Turn 6 =====
130  . . . . .
131  . . . . .
132  . . . . .
133  . . . . .
134  . . . . .
135  . . . | | | . . . . .
136  . . . | | | . . . . .
137  . . . | | | . . . . .
138  . . . | | | . . . . .
139  . . . | | | . . . . .
140  . . . | | | . . . . .
141  . . . | . . . . .
142  . . . . . | . . . . .
143  . . . . . . . . . .
144  . . . . . . . . . .
145  . . . . . . . . . .
146  . . . # . . . . . | . . . . .
147  . . y . x . . . . .
148  . . . . . . . . . .
149  . . . . . . . . . .
150
151  x: Moving DOWN
152  y: Moving UP
153
154  ===== Turn 7 =====
155  . . . . .
156  . . . . .
157  . . . . .
158  . . . . .
159  . . . . .
160  . . . | | | . . . . .
161  . . . | | | . . . . .
162  . . . | | | . . . . .
163  . . . | | | . . . . .
164  . . . | | | . . . . .
165  . . . | | | . . . . .
166  . . . | . . . . .
167  . . . . . | . . . . .
168  . . . . . . . . . .
169  . . . . . . . . . .
170  . . . . . . . . . .
171  . . y # . . . . . | . . . . .
172  . . . . . . . . . .
173  . . . . . x . . . . .
174  . . . . . . . . . .
175
176  x: Moving DOWN
177  y: Moving RIGHT
178
179  Simulation over: y wins!!! Successfully reach the goal at position (4,4)

```

Listing 42: sim_two_test Test Case Output

A.19 simulator_test

Listing 43: simulator_test Test Case Input

```
1 /*this is test of simulator , it output the walls and trace of agent into agent1.dat*/
2
3 /* ENV CODE... PLAYER—DON'T CHANGE OR LOOK AT ME!!! */
4 goal(4,4);
5 size(20,20);
6 wall(12,4);
7 wall(4,9);
8 wall(6,8);
9 wall($X,$Y) {AND:
10     $X>=4;
11     $X<=6;
12     $Y<=15;
13     $Y>=10;
14 }
15
16
17 agent("x", "tests/agents/agent1.ul");
```

Listing 43: simulator_test Test Case Input

Listing 44: tests/agents/agent1.ul

```
1 imove("UP");
2 imove("UP");
3 imove("RIGHT");
4 imove("RIGHT");
5 imove("RIGHT");
6 imove("RIGHT");
7
8 move($dir) {
9     @forget1( imove($dir); );
10    imove($dir);
11 }
12
13 move("DOWN");
14
15 main() {
16     @print($d);
17     move($d);
18 }
```

Listing 44: tests/agents/agent1.ul

Listing 45: simulator_test Test Case Output

```

1  x: Moving UP
2
3  ===== Turn 1 =====
4  . . . . .
5  . . . . .
6  . . . . .
7  . . . . .
8  . . . . .
9  . . . | | | . . . . .
10 . . . | | | . . . . .
11 . . . | | | . . . . .
12 . . . | | | . . . . .
13 . . . | | | . . . . .
14 . . . | | | . . . . .
15 . . . | . . . . .
16 . . . . | . . . . .
17 . . . . . . . . .
18 . . . . . . . . .
19 . . . . . . . . .
20 . . . # . . . . | . . . . .
21 . . . . . . . . .
22 x . . . . . . . . .
23 . . . . . . . . .
24
25 x: Moving UP
26
27  ===== Turn 2 =====
28 . . . . .
29 . . . . .
30 . . . . .
31 . . . . .
32 . . . . .
33 . . . | | | . . . . .
34 . . . | | | . . . . .
35 . . . | | | . . . . .
36 . . . | | | . . . . .
37 . . . | | | . . . . .
38 . . . | | | . . . . .
39 . . . | . . . . .
40 . . . . | . . . . .
41 . . . . . . . . .
42 . . . . . . . . .
43 . . . . . . . . .
44 . . . # . . . . | . . . . .
45 x . . . . . . . . .
46 . . . . . . . . .
47 . . . . . . . . .
48
49 x: Moving RIGHT
50

```

```

51  ===== Turn 3 =====
52  . . . . .
53  . . . . .
54  . . . . .
55  . . . . .
56  . . . . .
57  . . . | | | . . . . .
58  . . . | | | . . . . .
59  . . . | | | . . . . .
60  . . . | | | . . . . .
61  . . . | | | . . . . .
62  . . . | | | . . . . .
63  . . . | . . . . .
64  . . . . . | . . . . .
65  . . . . .
66  . . . . .
67  . . . . .
68  . . . # . . . . . | . . . . .
69  . x . . . . .
70  . . . . .
71  . . . . .
72
73  x: Moving RIGHT
74
75  ===== Turn 4 =====
76  . . . . .
77  . . . . .
78  . . . . .
79  . . . . .
80  . . . . .
81  . . . | | | . . . . .
82  . . . | | | . . . . .
83  . . . | | | . . . . .
84  . . . | | | . . . . .
85  . . . | | | . . . . .
86  . . . | | | . . . . .
87  . . . | . . . . .
88  . . . . . | . . . . .
89  . . . . .
90  . . . . .
91  . . . . .
92  . . . # . . . . . | . . . . .
93  . . x . . . . .
94  . . . . .
95  . . . . .
96
97  x: Moving RIGHT
98
99  ===== Turn 5 =====
100 . . . . .
101 . . . . .

```


48

```

153 . . . | | | . . . . . . . . . . . . . . .
154 . . . | | | . . . . . . . . . . . . . . .
155 . . . | | | . . . . . . . . . . . . . . .
156 . . . | | | . . . . . . . . . . . . . . .
157 . . . | | | . . . . . . . . . . . . . . .
158 . . . | | | . . . . . . . . . . . . . . .
159 . . . | . . . . . . . . . . . . . . .
160 . . . . . | . . . . . . . . . . . . . . .
161 . . . . . . . . . . . . . . . . . . .
162 . . . . . . . . . . . . . . . . . . .
163 . . . . . . . . . . . . . . . . . . .
164 . . . # . . . . . . | . . . . . . . . .
165 . . . . . . . . . . . . . . . . . . .
166 . . . . x . . . . . . . . . . . . . . .
167 . . . . . . . . . . . . . . . . . . .
168
169 x: Moving DOWN
170
171 ===== Turn 8 =====
172 . . . . . . . . . . . . . . . . . . .
173 . . . . . . . . . . . . . . . . . . .
174 . . . . . . . . . . . . . . . . . . .
175 . . . . . . . . . . . . . . . . . . .
176 . . . . . . . . . . . . . . . . . . .
177 . . . | | | . . . . . . . . . . . . . . .
178 . . . | | | . . . . . . . . . . . . . . .
179 . . . | | | . . . . . . . . . . . . . . .
180 . . . | | | . . . . . . . . . . . . . . .
181 . . . | | | . . . . . . . . . . . . . . .
182 . . . | | | . . . . . . . . . . . . . . .
183 . . . | . . . . . . . . . . . . . . .
184 . . . . . | . . . . . . . . . . . . . . .
185 . . . . . . . . . . . . . . . . . . .
186 . . . . . . . . . . . . . . . . . . .
187 . . . . . . . . . . . . . . . . . . .
188 . . . # . . . . . . | . . . . . . . . .
189 . . . . . . . . . . . . . . . . . . .
190 . . . . . . . . . . . . . . . . . . .
191 . . . . x . . . . . . . . . . . . . . .
192
193 x: Moving DOWN
194
195 Simulation over: x hits the x margin and Game over!!!

```

Listing 45: simulator_test Test Case Output

A.20 sprint1

Listing 46: sprint1 Test Case Input

```

1 wall(4,4);

```

```

2  wall(6,3);
3  move("UP");
4  move("UP");
5  move("UP");
6  move("RIGHT");
7  move("RIGHT");
8  move("RIGHT");
9  face("foo");
10
11 face(symA);
12
13 main() {OR:
14     @print("Wall: ", $x, ", ", $y);
15     wall($x, $y);
16     face($y);
17     move($y);
18 }

```

Listing 46: sprint1 Test Case Input

Listing 47: sprint1 Test Case Output

```

1 Wall: 4, 4
2
3   ^^^ Solution   ^^^
4
5 Wall: 6, 3
6
7   ^^^ Solution   ^^^
8
9 Wall: Any, 'foo'
10
11  ^^^ Solution   ^^^
12
13 Wall: Any, symA
14
15  ^^^ Solution   ^^^
16
17 Wall: Any, 'UP'
18
19  ^^^ Solution   ^^^
20
21 Wall: Any, 'RIGHT'
22
23  ^^^ Solution   ^^^
24
25 No more solutions

```

Listing 47: sprint1 Test Case Output

B Appendix: Code Listings

B.1 parser.mly

Listing 48: CμLOG Parser

```
1  /* Original author:Devesh Dedhia*/
2
3  %{ open Ast %}
4
5  %token PLUS MINUS TIMES DIVIDE NOT ASSIGN EOF COMMENT
6  %token LBRACE RBRACE LPAREN RPAREN
7  %token ARROPEN ARRCLOSE AT DOT
8  %token SEMICOLON OR AND COMMA COLON QUOTE QUESTION
9  %token <string> ID VARIABLE STRING
10 %token <int> DIGIT
11
12 /* Comparison tokens */
13 %token EQ GT LT GEQ LEQ NEQ
14
15 %nonassoc EQ
16 %left NEQ
17 %left LT GT LEQ GEQ
18 %left PLUS MINUS
19 %left TIMES DIVIDE
20
21
22 %start program
23 %type < Ast.program> program
24
25 %%
26
27 program :
28     main { Program($1) }
29
30 main :
31     EOF { [] }
32 | top main { $1 :: $2 }
33
34 top :
35     culogRule          { $1 } /* the program consists of facts rules and directives*/
36 | culogFact            { $1 }
37 | culogDirective       { $1 }
38
39 culogFact :                /* wall(2,2);*/
40     ID LPAREN param_list RPAREN SEMICOLON      { Fact($1, Params(List.rev $3) ) }
41
42
43 culogRule :                /* wall(){}*/
44     ID LPAREN param_list RPAREN block          { Rule($1, Params(List.rev $3), $5 ) }
45
46 culogDirective :           /*@print(" these are global directives");*/
```

```

47  AT ID LPAREN param_list RPAREN SEMICOLON          { GlobalDirective($2, Params(List.rev $4)) }
48
49
50  param_list:
51    { [] }
52  | param                                           { [$1] }
53  | param_list COMMA param                         { $3::$1 } /* Params seprated by Commas*/
54
55  param:
56    VARIABLE          { Var($1) }          /* $x,$agent*/
57  | ID                { Sym($1) }          /* symbA */
58  | DIGIT             { Lit($1) }          /* 0...9*/
59  | PLUS DIGIT        { Lit($2) }          /* +0...+9*/
60  | MINUS DIGIT        { Lit(-1*$2) }       /* -0...-9*/
61  | STRING            { Str($1) }          /* "STRINGS"*/
62  | array             { Arr($1) }          /* [$x,$y]*/
63  | QUESTION          { Ques }             /* ?- to indicate Anonymous variables */
64
65  array:
66    ARROPEN param_list ARRCLOSE { Array( List.rev $2 ) }
67
68  block:
69    LBRACE stmt_list RBRACE { Block("AND", Stmts( $2 ) ) } /* Default reduction operator is AND */
70  | LBRACE ID COLON stmt_list RBRACE{ Block($2, Stmts( $4 )) } /* any operator can be used*/
71
72  stmt_list:
73    /*nothing*/ { [] }
74  | statement stmt_list { $1 :: $2 }
75
76  statement:          /* statements can be sub-blocks , facts ,comparison statements */
77                      /* directives statements or dot operator statements*/
78    block { $1 }
79  | ID LPAREN param_list RPAREN SEMICOLON          { Eval($1, Params(List.rev $3)) }
80  | NOT ID LPAREN param_list RPAREN SEMICOLON      { NEval($2, Params(List.rev $4)) }
81  | VARIABLE DOT ID LPAREN param_list RPAREN SEMICOLON { Dot2($1,$3,Params(List.rev $5)) }
82  | NOT VARIABLE DOT ID LPAREN param_list RPAREN SEMICOLON { NDot2($2,$4,Params(List.rev $6)) }
83  | VARIABLE DOT AT ID LPAREN direc_list RPAREN SEMICOLON { Dot1($1,$4,(List.rev $6)) }
84  | expr EQ expr SEMICOLON                        { Comp($1,Eq,$3) }
85  | expr NEQ expr SEMICOLON                       { Comp($1,Neq,$3) }
86  | expr GT expr SEMICOLON                        { Comp($1,Gt,$3) }
87  | expr LT expr SEMICOLON                        { Comp($1,Lt,$3) }
88  | expr GEQ expr SEMICOLON                       { Comp($1,Geq,$3) }
89  | expr LEQ expr SEMICOLON                       { Comp($1,Leq,$3) }
90  | AT ID LPAREN param_list RPAREN SEMICOLON      { Directive($2, Params(List.rev $4)) }
91  | AT ID LPAREN direc_list_first RPAREN SEMICOLON { DirectiveStudy($2,(List.rev $4)) }
92
93  direc_list_first:
94    directive SEMICOLON direc_list                { $1 :: $3 }
95
96  direc_list:
97    { [] }

```

```

98 | directive SEMICOLON direc_list { $1 :: $3 }
99
100 directive:
101 ID LPAREN param_list RPAREN { ($1, Params(List.rev $3)) }
102
103
104 expr:
105     expr PLUS expr { Binop($1, Plus, $3) } /* $X+4*/
106 | expr MINUS expr { Binop($1, Minus, $3) } /*3-4*/
107 | expr TIMES expr { Binop($1, Mult, $3) } /* $x*4 */
108 | expr DIVIDE expr { Binop($1, Divide, $3) } /* $x/$y */
109 | DIGIT { ELit($1) }
110 | MINUS DIGIT { ELit(-1*$2) }
111 | PLUS DIGIT { ELit($2) }
112 | VARIABLE { EVar($1) }
113 | STRING { EStr($1) }
114 | ID { EId($1) }

```

Listing 48: CμLOG Parser

B.2 scanner.mll

Listing 49: CμLOG Scanner

```

1 { open Parser }
2 rule token = parse
3   [ ' ' '\t' '\r' '\n' ] { token lexbuf }
4   | "/" { comment lexbuf }
5   | "//" { linecomment lexbuf }
6   | '(' { LPAREN }
7   | ')' { RPAREN }
8   | '{' { LBRACE }
9   | '}' { RBRACE }
10  | ';' { SEMICOLON }
11  | ',' { COMMA }
12  | '+' { PLUS }
13  | '-' { MINUS }
14  | '*' { TIMES }
15  | '/' { DIVIDE }
16  | "==" | "=" { EQ }
17  | "!=" { NEQ }
18  | '<' { LT }
19  | "<=" { LEQ }
20  | ">" { GT }
21  | ">=" { GEQ }
22  | '@' { AT }
23  | '.' { DOT }
24  | ':' { COLON }
25  | '[' { ARROPEN }
26  | ']' { ARRCLOSE }
27  | '"' { QUOTE }

```

```

28 | '?' { QUESTION }
29 | '!' { NOT }
30 | '$' [ 'a'-'z' 'A'-'Z' ] [ 'a'-'z' 'A'-'Z' '0'-'9' '_' ]* as var { VARIABLE(var) } (* variables
31 | [ '0'-'9' ]+ as lxm { DIGIT(int_of_string lxm) }
32 | [ 'a'-'z' 'A'-'Z' ] [ 'a'-'z' 'A'-'Z' '0'-'9' '_' ]* as lxm { ID(lxm) } (* An ID must start
33 | '"' ( [ '^' '"' '\t' '\r' '\n' ]+ as lxm ) '"' { STRING(lxm) } (* any thing declared
34 |                                     quotes is a string
35 | eof { EOF }
36
37 and comment = parse
38     "*/" { token lexbuf }
39 | - { comment lexbuf }
40
41 and linecomment = parse
42     [ '\r' '\n' ] { token lexbuf }
43 | - { linecomment lexbuf }

```

Listing 49: CμLOG Scanner

B.3 ast.mli

Listing 50: CμLOG AST

```

1  (* Original author: Cheng Cheng
2    Edited          : Devesh Dedhia
3    support added to include directives *)
4
5  type operator = Plus | Minus | Mult | Divide
6  type compoperator = Lt | Leq | Gt | Geq | Eq | Neq
7
8  (* type study = learn | forget *)
9
10 type param =
11     Lit of int (* 0...9 *)
12 | Sym of string (* sym1 *)
13 | Var of string (* $X *)
14 | Str of string (* "asdf" *)
15 | Arr of params (* [2, $x, symb1] *)
16 | Ques
17
18 and params =
19     Params of param list
20 | Array of param list
21
22 type expr =
23     Binop of expr*operator*expr (* 0>$X>=5 $X==$Y 5!=4 *)
24 | ELit of int (* 0...9 *)
25 | EVar of string (* $X *)
26 | EStr of string (* "asdf" *)
27 | EIId of string (* sym1 *)
28

```

```

29 type eval = string*params
30
31 type stmt =
32   Block of string*stmts                                (* {.....} *)
33   | Comp of expr*compoperator*expr                     (* $5+5<$4 $a=5,$b=6; *)
34   | NEval of string*params                             (* !wall(4,5) *)
35   | Eval of eval                                       (* wall(4,5) *)
36   | DirectiveStudy of string*(eval list)              (* @learn(wall(4,5);) *)
37   | Directive of string*params                        (* @print("dfdsf"); *)
38   | Dot1 of string*string*(eval list)                  (* $agent.@learn(wall(4,5);) *)
39   | Dot2 of string*string*params                       (* $env.view($X,$Y,$Obj) *)
40   | NDot2 of string*string*params                     (* !env.view($X,$Y,$Obj) *)
41
42 and stmts=Stmts of stmt list    (* statement1;statement2;statement3; *)
43
44
45 type ruleFact =
46   Rule of string * params * stmt    (* wall(3,4){AND: ....} *)
47   | Fact of string * params          (* wall(2,2); *)
48   | GlobalDirective of string*params (* @attach("dfsfsa") *) (* @print("ddafafa") *)
49
50
51 type program = Program of ruleFact list

```

Listing 50: CμLOG AST

B.4 printer.ml

Listing 51: CμLOG AST Printer

```

1  (*
2  *   printer.ml
3  *
4  *
5  *   Started on   Wed Nov  5 15:18:34 2008 John Demme
6  *   Last update  Wed Nov  5 16:05:31 2008 John Demme
7  *)
8
9  open Ast
10
11 let string_of_compoperator = function
12   Lt   -> "<"
13   | Leq -> "<="
14   | Gt   -> ">"
15   | Geq  -> ">="
16   | Eq   -> "="
17   | Neq  -> "!="
18
19 let string_of_operator = function
20   Plus  -> "+"
21   | Minus -> "-"

```



```

22 | Mult    -> "*"
23 | Divide -> "/"
24
25
26 let rec string_of_expr = function
27   Binop(e1, o, e2) -> (string_of_expr e1) ^ (string_of_operator o) ^ (string_of_expr e2)
28 | ELit(i) -> string_of_int i
29 | EVar(s) -> s
30 | EStr(s) -> s
31 | EId(s)  -> s
32
33 let rec string_of_param = function
34   Lit(i) -> string_of_int i
35 | Sym(s) -> s
36 | Var(s) -> s
37 | Str(s) -> "\"" ^ s ^ "\""
38 | Arr(a) -> "[" ^ string_of_params a ^ "]"
39 | Ques  -> "?"
40
41
42 and string_of_params = function
43   Params(pList) -> String.concat "," (List.map string_of_param pList)
44 | Array(pList) -> String.concat "," (List.map string_of_param pList)
45
46 let rec string_of_stmts = function
47   Stmts(sList) -> String.concat "\n" (List.map string_of_stmt sList)
48 and string_of_stmt = function
49   Block(red, stmts) -> "{" ^ red ^ ":\n" ^ (string_of_stmts stmts) ^ "\n}"
50 | Comp(e1, c, e2) -> (string_of_expr e1) ^ (string_of_compoperator c)
51   ^ (string_of_expr e2) ^ ";";
52 | Eval(name, ps) -> name ^ "(" ^ (string_of_params ps) ^ ");";
53 | NEval(name1, ps1) -> "!" ^ name1 ^ "(" ^ (string_of_params ps1) ^ ");";
54 | DirectiveStudy(name, stmts) -> "@" ^ name ^ "(" ^
55   (string_of_stmts (Stmts (List.map (fun a -> Eval(a)) stmts))) ^ ")"
56 | Directive(name, params) -> "@" ^ name ^ "(" ^ (string_of_params params) ^ ")"
57 | Dot1(str1, str2, stmts) -> str1 ^ "." ^ "@" ^ str2 ^ "(" ^
58   (string_of_stmts (Stmts (List.map (fun a -> Eval(a)) stmts))) ^ ")"
59 | Dot2(str1, str2, ps) -> str1 ^ "." ^ str2 ^ "(" ^ (string_of_params ps) ^ ");";
60 | NDot2(str1, str2, ps) -> "!" ^ str1 ^ "." ^ str2 ^ "(" ^ (string_of_params ps) ^ ");";
61
62
63 let string_of_ruleFact = function
64   Rule(name, params, stmt) -> name ^ "(" ^ (string_of_params params) ^ "),"
65   ^ (string_of_stmt stmt)
66 | Fact(name, params) -> name ^ "(" ^ (string_of_params params) ^ ");";
67 | GlobalDirective(name, ps) -> "@" ^ name ^ "(" ^ (string_of_params ps) ^ ");";
68
69 let string_of_program = function
70   Program(ruleList) -> String.concat "\n" (List.map string_of_ruleFact ruleList) ^ "\n"

```

Listing 51: C_μLOG AST Printer

B.5 tst.mli

Listing 52: CμLOG Translated Syntax Tree

```

1  (*
2    This is a simpler, much more restrictive version of the AST.
3    It is much easier for the interpreter to deal with, and is relatively
4    easy to obtain given an AST. The trans.ml module translates from the AST
5    to this TST.
6  *)
7  type param =
8      Lit   of int
9      | Sym of string
10     | Var of int
11     | Anon
12     | Str of string
13     | Arr of param list
14
15  and params = param list
16
17  type expr =
18      | ELit of int
19
20  type eval = string*params
21  type var  = int
22
23  type stmt =
24      Block of string*stmts                                (* {.....} *)
25      | Comp of var*Ast.compoperator*expr                 (* $5+5<$4 $a=5,$b=6; *)
26      | StrComp of var*string
27      | SymComp of var*string
28      | NEval of eval                                       (* !wall(4,5) *)
29      | Eval of eval                                         (* wall(4,5) *)
30      | DirectiveStudy of string*(eval list)               (* @learn(wall(4,5);) *)
31      | Directive of string*params                         (* @print("dfdsf"); *)
32      | Dot1 of int*string*(eval list)                     (* $agent.@learn(wall(4,5);) *)
33      | Dot2 of int*string*params                          (* env.view($X,$Y,$Obj) *)
34      | NDot2 of int*string*params                        (* !env.view($X,$Y,$Obj) *)
35
36  and stmts=stmt list  (* statment1;statment2;statement3; *)
37
38  type ruleFact =
39      | Rule of string * params * int * stmt * stmt list
40      | Fact of string * params
41
42
43  type program = ruleFact list

```

Listing 52: CμLOG Translated Syntax Tree

B.6 trans.ml

```

1  (* Functions to modify the AST slightly
2     to make parsing it easier for the interpreter.
3
4     Static checking could happen here as well.
5
6     John Demme
7  *)
8
9  open Ast
10
11  module StringMap = Map.Make(String);;
12
13  (* Give me the number of items in a StringMap *)
14  let map_length sMap =
15      let fLength k a b =
16          b + 1
17      in
18      StringMap.fold fLength sMap 0
19  ;;
20
21  (* Use me with List.fold to get a maximum index *)
22  let max_index s i l =
23      if i > l
24      then i
25      else l
26  ;;
27
28  (* Print all items in a StringMap *)
29  let smPrint key a =
30      Printf.printf "%s: %d\n" key a; ()
31  ;;
32
33  (* Get a variable name to variable number binding from a rule *)
34  let getBindings mRule =
35      (* TODO: Many of these functions could be made nicer using stuff like List.fold *)
36      let add_binding var bindings =
37          if (StringMap.mem var bindings) then
38              bindings
39          else
40              (StringMap.add var (map_length bindings) bindings)
41      in
42      let rec get_params_var_mapping params bindings =
43          let len = map_length bindings in
44          match params with
45          | [] -> bindings
46          | Var(name) :: tail ->
47              if (StringMap.mem name bindings)
48              then failwith "You cannot list the same variable twice in a parameter list"
49              else get_params_var_mapping tail (StringMap.add name len bindings)
50          | i :: tail ->

```

```

51         get_params_var_mapping tail (StringMap.add (string_of_int len) len bindings)
52     in
53     let rec get_eval_var_mapping params bindings =
54         match params with
55         [] -> bindings
56         | Var(name) :: tail ->
57             get_eval_var_mapping tail
58             (add_binding name bindings)
59         | _ :: tail -> get_eval_var_mapping tail bindings
60     in
61     let rec get_expr_var_mapping e bindings =
62         match e with
63         EVar(name) -> add_binding name bindings
64         | Binop(a, op, b) -> get_expr_var_mapping a (get_expr_var_mapping b bindings)
65         | _ -> bindings
66     in
67     let rec get_stmts_var_mapping stmts bindings =
68         match stmts with
69         [] -> bindings
70         | Block(redOp, Stmt(stmts)) :: tail ->
71             get_stmts_var_mapping tail (get_stmts_var_mapping stmts bindings)
72         | Comp(expr1, compOp, expr2) :: tail ->
73             get_stmts_var_mapping tail
74             (get_expr_var_mapping expr1 (get_expr_var_mapping expr2 bindings))
75         | Eval(name, Params(params)) :: tail ->
76             get_stmts_var_mapping tail (get_eval_var_mapping params bindings)
77         | Directive(name, Params(params)) :: tail ->
78             get_stmts_var_mapping tail (get_eval_var_mapping params bindings)
79         | _ :: tail ->
80             get_stmts_var_mapping tail bindings
81     in
82     match mRule with
83     Rule(name, Params(params), stmt) ->
84         get_stmts_var_mapping [stmt] (get_params_var_mapping params StringMap.empty)
85     | Fact(name, Params(params)) ->
86         (get_params_var_mapping params StringMap.empty)
87     | _ -> StringMap.empty
88 ;;
89
90 (* Translate a rule or fact from AST to TST *)
91 let translate_rule mRule =
92     let bindings = getBindings mRule in
93     let bget name =
94         StringMap.find name bindings
95     in
96     (* Translate parameters using these bindings *)
97     let translate_params params =
98         let translate_param param =
99             match param with
100             Var(name) -> Tst.Var(bget name)
101             | Lit(i) -> Tst.Lit(i)

```

```

102         | Sym(s)      -> Tst.Sym(s)
103         | Str(s)      -> Tst.Str(s)
104         | Arr(prms) -> failwith "Sorry, _arrays_are_unsupported"
105         | Ques        -> Tst.Anon
106     in
107         List.map translate_param params
108 in
109 let rec translate_stmts stmts =
110     (* Move the variable to one side, and simplify to a constant on the other *)
111     let translate_comp expr1 op expr2 =
112         (* Can this expression be numerically reduced? *)
113         let rec can_reduce expr =
114             match expr with
115             | ELit(i) -> true
116             | Binop(e1, op, e2) -> (can_reduce e1) && (can_reduce e2)
117             | _ -> false
118         in
119         (* Give me the reverse of an operator *)
120         let rev_op op =
121             match op with
122             | Lt -> Gt
123             | Gt -> Lt
124             | Eq -> Eq
125             | Neq -> Neq
126             | Geq -> Leq
127             | Leq -> Geq
128         in
129         (* Translate a comparison where the variable is on the LHS *)
130         let translate_comp_sv var_expr op expr =
131             (* Reduce a constant expression to a literal *)
132             let reduce expr =
133                 let rec num_reduce expr =
134                     match expr with
135                     | ELit(i) -> i
136                     | Binop(e1, op, e2) ->
137                         (let re1 = num_reduce e1 in
138                          let re2 = num_reduce e2 in
139                          match op with
140                          | Plus -> re1 + re2
141                          | Minus -> re1 - re2
142                          | Mult -> re1 * re2
143                          | Divide -> re1 / re2)
144                     | _ -> failwith "Internal_error_8"
145                 in
146                 Tst.ELit(num_reduce expr)
147             in
148             match var_expr with
149             | EVar(name) ->
150                 (* Can we numerically reduce the RHS? *)
151                 if not (can_reduce expr)
152                 then

```

```

153             (* If not, if better be a simple string of symbol comparison *)
154             match (op, expr) with
155                 Eq, EStr(s) -> Tst.StrComp(bget name, s)
156                 | Eq, EId (s) -> Tst.SymComp(bget name, s)
157                 | _ -> failwith "Unsupported_comparison"
158             else
159                 Tst.Comp(bget name, op, reduce expr)
160                 | _ -> failwith "Comparison_unsupported"
161         in
162             (* Does this expression have a variable *)
163             let rec has_var expr =
164                 match expr with
165                     EVar(i) -> true
166                     | Binop(e1, op, e2) -> (has_var e1) || (has_var e2)
167                     | _ -> false
168             in
169                 (* Check each expression for variables *)
170                 let ev1 = has_var expr1 in
171                 let ev2 = has_var expr2 in
172                     if ev1 && ev2
173                     then failwith "Comparisons_with_multiple_variables_are_unsupported."
174                     else if (not ev1) && (not ev2)
175                     then failwith "Error:_Comparison_is_constant"
176                     else if ev1
177                     then translate_comp_sv expr1 op expr2
178                     else translate_comp_sv expr2 (rev_op op) expr1
179         in
180             (* translate a list of evals *)
181             let mapEvList evList =
182                 List.map
183                     (fun ev ->
184                         match ev with
185                             (name, Params(plist)) ->
186                                 (name,
187                                     translate_params plist)
188                             | (name, Array(alist)) ->
189                                 failwith "Syntax_error,_arrays_not_permitted_as_params")
190                     evList
191         in
192             (* Translate a single statement *)
193             let rec replace_stmt stmt =
194                 match stmt with
195                     Block(redOp, Stmts(stmts)) ->
196                         Tst.Block(redOp, translate_stmts stmts)
197                     | Comp(expr1, compOp, expr2) ->
198                         translate_comp expr1 compOp expr2
199                     | Eval(name, Params(params)) ->
200                         Tst.Eval(name, translate_params params)
201                     | NEval(name, Params(params)) ->
202                         Tst.NEval(name, translate_params params)
203                     | Dot2(vname, pred, Params(params)) ->

```

```

204         Tst.Dot2(bget vname, pred, translate_params params)
205     | NDot2(vname, pred, Params(params)) ->
206         Tst.NDot2(bget vname, pred, translate_params params)
207     | Directive(n, Params(params)) ->
208         Tst.Directive(n, translate_params params)
209     | DirectiveStudy(n, evList) ->
210         Tst.DirectiveStudy(n, mapEvList evList)
211     | Dot1(vname, n, evList) ->
212         Tst.Dot1(bget vname, n, mapEvList evList)
213     | _ -> failwith "Unsupported_statement"
214 in
215     List.map replace_stmt stmts
216 in
217     (* Given a list of TST statements, prune out the ones
218        which have no effect on the solutions *)
219 let rec filterSE stmts =
220     match stmts with
221     [] -> []
222     | Tst.Block(redOp, stmts) :: tail ->
223         Tst.Block(redOp, filterSE stmts) :: filterSE tail
224     | Tst.Directive(_, _) :: tail ->
225         filterSE tail
226     | Tst.DirectiveStudy(_, _) :: tail ->
227         filterSE tail
228     | Tst.Dot1(_, _, _) :: tail ->
229         filterSE tail
230     | head :: tail ->
231         head :: filterSE tail
232 in
233     (* Given a list of TST statements, prune out the ones
234        which have some effect on the solutions *)
235 let rec filterNSE stmts =
236     match stmts with
237     [] -> []
238     | Tst.Block(redOp, stmts) :: tail ->
239         List.append (filterNSE stmts) (filterNSE tail)
240     | Tst.Directive(n, p) :: tail ->
241         Tst.Directive(n, p) :: filterNSE tail
242     | Tst.DirectiveStudy(n, p) :: tail ->
243         Tst.DirectiveStudy(n, p) :: filterNSE tail
244     | Tst.Dot1(v, n, p) :: tail ->
245         Tst.Dot1(v, n, p) :: filterNSE tail
246     | head :: tail ->
247         head :: filterNSE tail
248 in
249     (* This is the entry point for translate_rule
250        ... It's been awhile, so I figured you might need a reminder *)
251 match mRule with
252 Rule(name, Params(params), stmt) ->
253     let replacedStmts = translate_stmts [stmt] in
254     Tst.Rule(name,

```

```

255             (translate_params params),
256             1 + (StringMap.fold max_index bindings (-1)),
257             List.hd (filterSE replacedStmts),
258             filterNSE replacedStmts)
259         | Fact(name, Params(params)) ->
260           Tst.Fact(name, (translate_params params))
261         | _ -> failwith "Unsupported_global_directive"
262 ;;
263
264
265 let translate prog =
266   match prog with
267   | Program (rfList) ->
268     let newProgram = List.map translate_rule rfList in
269     (* print_string (Printer.string_of_program newProgram); *)
270     newProgram
271 ;;

```

Listing 53: C μ LOG AST to TST Translator

B.7 culog.ml

Listing 54: C μ LOG “General Purpose” Interpreter

```

1  (*
2  *   culog.ml
3  *
4  *   Made by (John Demme)
5  *   Login   <teqdruid@teqBook>
6  *
7  *   Started on   Mon Nov 24 16:03:20 2008 John Demme
8  *   Last update  Mon Nov 24 16:03:27 2008 John Demme
9  *)
10
11 open Interp
12
13 let rec iter_sols nxt =
14   match nxt with
15   | NoSolution -> print_string "No_more_solutions\n"
16   | Solution(c,n) ->
17     (print_string "\n^^^^_Solution_^^^^\n\n");
18     iter_sols (n ())
19 ;;
20
21 let myDBD db =
22   print_string "Database_dump:\n";
23   dump_db !db;
24   print_string "\n";
25
26 let _ =
27   let lexbuf = Lexing.from_channel (open_in Sys.argv.(1)) in

```



```

28  let program = Parser.program Scanner.token lexbuf in
29  let pDB = parseDB(program) in
30    (* myDBD pDB; *)
31    (let sGen = query pDB (ref []) "main" 0 in
32      iter_sols sGen);
33    (* myDBD pDB; *)
34  ;;

```

Listing 54: C μ LOG “General Purpose” Interpreter

B.8 simulator.ml

Listing 55: C μ LOG Simulator

```

1  (* simulator.ml
2  * This is a simulator for entities interaction.
3  * Specifically, simulator comprises of two parts.
4  * Simulator is divided into 2 parts, one for obtaining information about agents and environ
5  * their behaviors and the other being output driver.
6  * Original author: Cheng Cheng ( worked on communicating with interpreter and simulating
7  * Nishant Shah( worked on integrating the output driver with the simulator
8  * Support for loading multi-agents is added by John Demme
9  *)
10
11 open Interp
12 open Ast
13
14 (* define global references to the parameters of environment*)
15 let grid_size_ref=ref 1;;
16 let grid_x_size_ref=ref 1;;
17 let grid_y_size_ref=ref 1;;
18 let goal_x_ref=ref 1;;
19 let goal_y_ref=ref 1;;
20
21 (* define data structure of agent*)
22 type sim_agent = {
23   x    : int;
24   y    : int;
25   sym  : char;
26   db   : database
27 }
28
29 (*define a global array to restore information of wall and positions of agents *)
30 (* maximum environment size is 100*100 *)
31 (* '.' represents empty grid, '|' represents wall*)
32 let record=
33   let f index='.' in
34     Array.init 10000 f ;;
35
36 let clear_array a=
37   for index=0 to (Array.length a)-1 do

```

```

38     if index= (!grid_y_size_ref- !goal_y_ref)* !grid_x_size_ref+ !goal_x_ref-1 then
39         begin
40             a.(index)<-'#'
41         end
42     else a.(index)<-'.'
43 done
44 ;;
45
46 let sim_exit s =
47     Printf.printf "\nSimulation_over:_%s\n\n" s;
48     exit(1)
49 ;;
50
51 (*set the size of environment *)
52 let rec set_size nxt=
53     match nxt with
54     NoSolution-> ()
55     | Solution(c,n)->
56         (match c with
57             [CEqInt(x);CEqInt(y)]-> if x<1||x>100 then failwith "the_length_of_grid_is_
58             else if y<1||y>100 then failwith "the_width_of_grid_is_not_illegal!!!"
59             else
60                 begin
61                     grid_x_size_ref:=x;
62                     grid_y_size_ref:=y;
63                     grid_size_ref:=x*y
64                 end
65             |_-> ())
66 ;;
67
68 (* set the goal agents try to reach*)
69 let rec set_goal nxt=
70     match nxt with
71     NoSolution-> ()
72     | Solution(c,n)->
73         (match c with
74             [CEqInt(x);CEqInt(y)]-> if x<1||x>!grid_x_size_ref then failwith "illegal_
75             else if y<1||y>!grid_x_size_ref then failwith "illegal_goal_y_position"
76             else
77                 begin
78                     goal_x_ref:=x;
79                     goal_y_ref:=y;
80                 end
81             |_->())
82 ;;
83
84 (* display and output the results after interactions*)
85 let print_grid oc arr =
86     for a=0 to !grid_y_size_ref-1
87     do
88         for j= !grid_x_size_ref*(a) to !grid_x_size_ref*(a+1)-1

```

```

89     do
90         Printf.fprintf oc "%c_" arr.(j)
91     done;
92     Printf.fprintf oc "\n"
93 done
94 ;;
95
96 let print_file j arr =
97     let file = "Agent"^string_of_int(j)^".dat" in (* Write message to file *)
98     let oc = open_out file in (* create or truncate file, return channel *)
99     (print_grid oc arr;
100      close_out oc)
101 ;;
102
103 let print_stdout j arr =
104     Printf.printf "\n=====Turn_%d=====\\n" j;
105     print_grid stdout arr;
106     print_string "\n"
107 ;;
108
109 (* create wall in environment*)
110 let create_wall x_start x_end y_start y_end =
111     if x_start < 1 || x_end > !grid_x_size_ref then
112         failwith "Creating_Wall_: x_position_of_wall_exceeds_the_grids"
113     else if y_start < 1 || y_end > !grid_y_size_ref then
114         failwith "Creating_Wall_: y_position_of_wall_exceeds_the_grids"
115     else if x_start > x_end || y_start > y_end then failwith "Creating_Wall: wrong_range!!!"
116     else for i=x_start to x_end do
117         for j=y_start to y_end do
118             record.((!grid_y_size_ref-j)* !grid_x_size_ref+i-1)<-'|'
119         done
120     done
121 ;;
122
123
124 (* obtain wall information from interpreter and create walls*)
125 let rec iter_wall nxt=
126     match nxt with
127     | NoSolution -> ()
128     | Solution(c,n) ->
129         (match c with
130          | [Any; Any]-> create_wall 1 !grid_x_size_ref 1 !grid_y_size_ref
131          | [CEqInt(x); Any]-> create_wall x x 1 !grid_y_size_ref
132          | [CLT(x); Any]-> create_wall 1 (x-1) 1 !grid_y_size_ref
133          | [CGT(x); Any]-> create_wall (x+1) !grid_x_size_ref 1 !grid_y_size_ref
134          | [CRange(x1,x2); Any]-> create_wall (x1+1) (x2-1) 1 !grid_y_size_ref
135          | [Any; CEqInt(y)]-> create_wall 1 !grid_x_size_ref y y
136          | [Any; CLT(y)]-> create_wall 1 !grid_y_size_ref 1 (y-1)
137          | [Any; CGT(y)]-> create_wall 1 !grid_x_size_ref (y+1) !grid_y_size_ref
138          | [Any; CRange(y1,y2)]-> create_wall 1 !grid_x_size_ref (y1+1) (y2-1)
139          | [CEqInt(x); CEqInt(y)]-> create_wall x x y y

```

```

140         | [CEqInt(x);CLT(y)]-> create_wall x x 1 (y-1)
141         | [CEqInt(x);CGT(y)]-> create_wall x x (y+1) !grid_y_size_ref
142         | [CEqInt(x);CRange(y1,y2)]-> create_wall x x (y1+1) (y2-1)
143         | [CLT(x);CEqInt(y)]-> create_wall 1 (x-1) y y
144         | [CLT(x);CLT(y)]-> create_wall 1 (x-1) 1 (y-1)
145         | [CLT(x);CGT(y)]-> create_wall 1 (x-1) (y+1) !grid_y_size_ref
146         | [CLT(x);CRange(y1,y2)]-> create_wall 1 (x-1) (y1+1) (y2-1)
147         | [CGT(x);CEqInt(y)]-> create_wall (x+1) !grid_x_size_ref y y
148         | [CGT(x);CLT(y)]-> create_wall (x+1) !grid_x_size_ref 1 (y-1)
149         | [CGT(x);CGT(y)]-> create_wall (x+1) !grid_x_size_ref (y+1) !grid_y_size_ref
150         | [CGT(x);CRange(y1,y2)]-> create_wall (x+1) !grid_x_size_ref (y1+1) (y2-1)
151         | [CRange(x1,x2);CEqInt(y)]-> create_wall (x1+1) (x2-1) y y
152         | [CRange(x1,x2);CLT(y)]-> create_wall (x1+1) (x2-1) 1 (y-1)
153         | [CRange(x1,x2);CGT(y)]-> create_wall (x1+1) (x2-1) (y+1) !grid_y_size_ref
154         | [CRange(x1,x2);CRange(y1,y2)]-> create_wall (x1+1) (x2-1) (y1+1) (y2-1)
155         | _ -> ();
156     iter_wall (n ())
157 ;;
158
159 (* agent moves towards to a direction*)
160 let agent_move a direction =
161     Printf.printf "%c: Moving %s\n" a.sym direction;
162     match direction with
163     | "UP" -> {x = a.x; y = a.y + 1; db = a.db; sym = a.sym}
164     | "DOWN" -> {x = a.x; y = a.y - 1; db = a.db; sym = a.sym}
165     | "LEFT" -> {x = a.x - 1; y = a.y; db = a.db; sym = a.sym}
166     | "RIGHT" -> {x = a.x + 1; y = a.y; db = a.db; sym = a.sym}
167     | _ -> failwith "No such a direction!"
168 ;;
169
170 (* simulator stores the information of agent's move *)
171 (* if agent reaches the goal or hits wall, simulator terminates *)
172 let do_agent_move a =
173     let array_index = (!grid_y_size_ref - a.y) * !grid_x_size_ref + a.x - 1 in
174     if a.x < 1 || a.x > !grid_x_size_ref then (* x position is beyond range *)
175         begin
176             let str = (Char.escaped a.sym) ^ "_hits_the_y_margin_and_Game_over!!!" in
177             sim_exit str
178         end
179     else if a.y < 1 || a.y > !grid_y_size_ref then (* y position is beyond range *)
180         begin
181             let str = (Char.escaped a.sym) ^ "_hits_the_x_margin_and_Game_over!!!" in
182             sim_exit str
183         end
184     else if Array.get record array_index = '|' then
185         begin
186             let str = (Char.escaped a.sym) ^ "_hits_the_wall_and_Game_over!!!" in
187             sim_exit str
188         end
189     else if Array.get record array_index = '#' then
190         begin

```

```

191         let str=(Char.escaped a.sym)^"_wins!!! Successfully _reach _the _goal _at _position _(" ^ s
192         ^", " ^ string_of_int (!goal_y_ref)^")"
193         in
194         sim_exit str
195     else if (Array.get record array_index) != '.' then
196         begin
197             let str="Game_over!!! Agents_crash!!! _at _position _(" ^ string_of_int(a.x)^", " ^ string_of_int(a.y)^")"
198             sim_exit str
199         end
200     else record.(array_index)<- a.sym
201 ;;
202
203 (* obtain current position of agent from interpretor and make it move*)
204 let iter_move agent nxt =
205     match nxt with
206     | NoSolution -> failwith "No_Solution"
207     | Solution([CEqlStr(dir)], -) ->
208         let new_agent = agent_move agent dir in
209         ignore (do_agent_move new_agent);
210         new_agent
211     | - -> failwith "Invalid_(or_no)_move"
212 ;;
213
214 (* load the databases of all agents in environment*)
215 let my_loc_db agent all env =
216     ref ([Interp.Fact({name = "loc"; params = [CEqlInt(agent.x); CEqlInt(agent.y)]});
217         Interp.Fact({name = "env"; params = [CEqlAgent(env)]});
218         @
219         (List.map
220          (fun other ->
221              Interp.Fact({name = "agent"; params = [CEqlAgent(other.db)]})
222              (List.filter (fun a -> a != agent) all)))
223     );
224 (* simulation function*)
225 let simulation envDB agents =
226     let rec loop i agents =
227         let sGen_size=query envDB (ref []) "size" 2 in
228         set_size sGen_size;
229         let sGen_goal=query envDB (ref []) "goal" 2 in
230         set_goal sGen_goal;
231         clear_array record;
232         let sGen_wall=query envDB (ref []) "wall" 2 in
233         iter_wall sGen_wall;
234         let new_agents =
235             List.map
236             (fun agent ->
237                 let sGen_move = query agent.db (my_loc_db agent agents envDB) "move" 1 in
238                 iter_move agent sGen_move)
239             agents
240         in

```

```

241         print_stdout i record;
242         if i>100 then sim_exit "You_lose!_Can_not_reach_the_goal_with_in_100_steps"
243         else loop (i+1) new_agents
244     in loop 1 agents
245 ;;
246
247
248 let load_agent db_loc =
249     match db_loc with
250     (c, s) ->
251         let lexbuf1 = Lexing.from_channel (open_in s) in
252         let program = Parser.program Scanner.token lexbuf1 in
253         {x=1; y=1; sym = (String.get c 0); db = Interp.parseDB(program)}
254 ;;
255
256 (*load database of rules and facts for a single agent*)
257 let load_db db_loc =
258     let lexbuf1 = Lexing.from_channel (open_in db_loc) in
259     let program = Parser.program Scanner.token lexbuf1 in
260     {x=1; y=1; sym = 'x'; db = Interp.parseDB(program)}
261 ;;
262
263
264 let get_agent_locs db =
265     let rec gal_int res =
266         match res with
267         NoSolution -> []
268         | Solution([CEqlStr(c); CEqlStr(s)], nxt) -> (c,s) :: (gal_int (nxt()))
269         | _ -> failwith "Failed_to_load_agent"
270     in
271     gal_int (query db (ref [])) "agent" 2)
272 ;;
273
274
275 let _ =
276     let envDB = load_db Sys.argv.(1) in
277     let agent_locs = get_agent_locs envDB.db in
278     if 0 == (List.length agent_locs)
279     then simulation envDB.db [envDB]
280     else
281         let agentDBs = List.map load_agent agent_locs in
282         simulation envDB.db agentDBs
283 ;;

```

Listing 55: C μ LOG Simulator

B.9 interp.ml

Listing 56: C μ LOG Interpreter

1 (*

```

2  *  interp.ml
3  *
4  *  This guy is the interpreter... It "compiles" the TST to a bunch of OCaml
5  *    functions to be run during a query.
6  *
7  *  You'll quickly be able to tell that this whole method is _begging_ for co-routines.
8  *    Lazy evaluation could be beneficial here as well.
9  *
10 *
11 *  John Demme
12 *
13 *)
14
15 type var_cnst =
16   Any
17   | FalseSol
18   | CEqlSymbol of string
19   | CEqlInt    of int
20   | CEqlStr    of string
21   | CLT        of int
22   | CGT        of int
23   | CRange     of int*int
24   | CEqlAgent  of database
25
26 and cnst = var_cnst list
27
28 and signature = {
29   name   : string;
30   params : cnst
31 }
32
33 and next =
34   NoSolution
35   | Solution of cnst * (unit -> next)
36
37 and rule_fact =
38   Fact of signature
39   | Rule of signature * (database -> database -> cnst -> next)
40
41 and database = rule_fact list ref
42 ;;
43
44
45 let string_of_cnst = function
46   Any -> "Any"
47   | FalseSol -> "False"
48   | CEqlSymbol(s) -> s
49   | CEqlStr(s) -> "\"" ^ s ^ "\""
50   | CEqlInt(i) -> string_of_int i
51   | CLT(i) -> "<" ^ (string_of_int i)
52   | CGT(i) -> ">" ^ (string_of_int i)

```

```

53 | CRange(a,b)-> (string_of_int a) ^ ".." ^ (string_of_int b)
54 | CEqlAgent(a) -> "Agent"
55 ;;
56
57 let string_of_eval name vars =
58   name ^ "(" ^ String.concat "," (List.map string_of_cnst vars) ^ ")\n"
59 ;;
60
61 let cAnd a b =
62   let rec and_int a b t =
63     match (a, b) with
64     | (Any, _) -> b
65     | (_, Any) -> a
66     | (CEqlAgent(a1), CEqlAgent(a2)) when (a1 == a2) -> a
67     | (CEqlSymbol(s1), CEqlSymbol(s2)) when (0 == String.compare s1 s2) -> a
68     | (CEqlStr(s1), CEqlStr(s2)) when (0 == String.compare s1 s2) -> a
69     | (CEqlInt(i1), CEqlInt(i2)) when (i2 == i2) -> a
70     | (CEqlInt(i1), CGT(i2)) when (i1 > i2) -> a
71     | (CEqlInt(i1), CLT(i2)) when (i1 < i2) -> a
72     | (CLT(i1), CLT(i2)) -> CLT(min i1 i2)
73     | (CGT(i1), CGT(i2)) -> CLT(max i1 i2)
74     | (CGT(i1), CLT(i2)) when (i1 < i2) -> CRange(i1, i2)
75     | (CRange(l, u), CEqlInt(i)) when (i > l && i < u) -> b
76     | (CRange(l, u), CGT(i)) when (i < u - 1) -> CRange((max l i), u)
77     | (CRange(l, u), CLT(i)) when (i > l + 1) -> CRange(l, (min u i))
78     | (CRange(l1, u1), CRange(l2, u2)) when (u1 > l2 && u2 > l1) -> CRange((max l1 l2),
79     | (_, _) when t -> and_int b a false
80     | (_, _) -> FalseSol
81   in
82     and_int a b true
83 ;;
84
85 let range_to_int c =
86   match c with
87   | CRange(l, u) when (l + 2 == u) -> CEqlInt(l+1)
88   | _ -> c
89 ;;
90
91 let int_to_range c =
92   match c with
93   | CEqlInt(i) -> CRange(i-1, i+1)
94   | _ -> c
95 ;;
96
97 (* For each constraint, subtract the second from the first *)
98 (* TODO: There are off-by-one errors in here... Fix when you have a clearer head *)
99 let cMinus b s =
100   let cmi b s =
101     (* Too many combinations and no play makes Johnny go something something *)
102     match (b, s) with
103     | (_, Any) ->

```



```

104     []
105     | (Any, _) ->
106         failwith "Unsupported_subtraction_need_constraint"
107     | (CEqlSymbol(s1), CEqlSymbol(s2)) when (0 != String.compare s1 s2) ->
108         [b; s]
109     | (CEqlStr(s1), CEqlStr(s2)) when (0 != String.compare s1 s2) ->
110         [b; s]
111     | (CEqlInt(i1), CEqlInt(i2)) when (i1 != i2) ->
112         [b; s]
113     | (CEqlAgent(a1), CEqlAgent(a2)) when (a1 != a2) ->
114         [b; s]
115     | (CRange(bl, bu), CRange(sl, su)) when (bl < sl && bu > su) ->
116         [CRange(bl, sl); CRange(su, bu)]
117     | (CRange(bl, bu), CRange(sl, su)) when (bl < sl && bu < su) ->
118         [CRange(bl, min sl bu)]
119     | (CRange(bl, bu), CRange(sl, su)) when (bl < sl && bu > su) ->
120         [CRange(max bl su, bu)]
121     | (CRange(bl, bu), CRange(sl, su)) when (bu > sl || bl > su) ->
122         [b]
123     | (CGT(bi), CLT(si)) -> [CGT(max bi si)]
124     | (CLT(bi), CGT(si)) -> [CLT(min bi si)]
125     | (CGT(bi), CGT(si)) when (bi < si) -> [CRange(bi, si)]
126     | (CLT(bi), CLT(si)) when (bi < si) -> [CRange(si, bi)]
127     | _ -> []
128   in
129     List.map range_to_int (cmi (int_to_range b) (int_to_range s))
130 ;;
131
132 let list_acc mapper list =
133   let rec acc list ret =
134     match list with
135     | [] -> ret
136     | hd :: tl -> acc tl ((mapper hd) @ ret)
137   in
138     acc list []
139 ;;
140
141 (* return the first n elements of list *)
142 let rec list_first n list =
143   match list with
144   | [] -> []
145   | hd :: tl when n > 0 -> hd :: list_first (n - 1) tl
146   | _ -> []
147 ;;
148
149 let rec list_fill item number =
150   if number <= 0
151   then []
152   else item :: (list_fill item (number - 1));;
153
154 let cnst_extend a b =

```

```

155   let delta = (List.length a) - (List.length b) in
156   if delta > 0
157   then (a, List.append b (list_fill Any delta))
158   else if delta < 0
159   then (List.append a (list_fill Any (delta * -1)), b)
160   else (a,b)
161 ;;
162
163 let cnst_extend_to a l =
164   let delta = l - (List.length a) in
165   if delta > 0
166   then List.append a (list_fill Any delta)
167   else a
168 ;;
169
170 let cnstAndAll aC bC =
171   let (aC, bC) = cnst_extend aC bC in
172   List.map2 cAnd aC bC
173 ;;
174
175 let match_signature signature name vars =
176   let match_params param vars =
177     let anded = cnstAndAll param vars in
178     List.for_all (fun a -> a != FalseSol) anded
179   in
180   (String.compare signature.name name == 0) &&
181   ((List.length signature.params) == (List.length vars)) &&
182   (match_params signature.params vars)
183 ;;
184
185 let remove_fact_all db pred cnsts =
186   (* print_string ("Removing: " ^ (string-of-eval pred cnsts) ^ "\n"); *)
187   List.filter
188     (fun curr ->
189       match curr with
190         Fact(signature) when
191           match_signature signature pred cnsts -> false
192       | _ -> true)
193   db
194 ;;
195
196 let rec remove_fact1 db pred cnsts =
197   (* print_string ("Removing: " ^ (string-of-eval pred cnsts) ^ "\n"); *)
198   match db with
199     [] -> []
200   | Fact(sign) :: tl when match_signature sign pred cnsts -> tl
201   | hd :: tl -> hd :: (remove_fact1 tl pred cnsts)
202 ;;
203
204 let cnst_of_params params env =
205   let param_to_cnst = function

```

```

206     Tst.Lit(i) -> CEqlInt      (i)
207   | Tst.Sym(s) -> CEqlSymbol  (s)
208   | Tst.Var(i) -> List.nth env i
209   | Tst.Str(s) -> CEqlStr     (s)
210   | Tst.Anon   -> Any
211   | Tst.Arr(a) -> failwith "Arrays_are_not_supported_yet"
212   in
213     (* print_string (string_of_eval "cop-env" env);
214     print_string ("cop: " ^ (Printer.string_of_params (Tst.Params(params))) ^ "\n");*)
215     List.map param_to_cnst params
216 ;;
217
218 let sig_to_cnst signature =
219   let param_to_cnst = function
220     Tst.Lit(i) -> CEqlInt      (i)
221   | Tst.Sym(s) -> CEqlSymbol  (s)
222   | Tst.Var(i) -> Any
223   | Tst.Anon   -> Any
224   | Tst.Str(s) -> CEqlStr     (s)
225   | Tst.Arr(a) -> failwith "Arrays_are_not_supported_yet"
226   in
227     List.map param_to_cnst signature
228 ;;
229
230 (* TODO: Need constraints list mapping *)
231 let rec run_eval db addDB name vars =
232   let rec run_gen tail nextGen =
233     let sols = (nextGen ()) in
234     match sols with
235     | NoSolution -> eval_loop tail
236     | Solution (cnst, gen) ->
237       Solution(
238         (list_first (List.length vars) cnst),
239         (fun unit -> run_gen tail gen))
240   and eval_loop e =
241     match e with
242     | [] -> NoSolution
243     | Fact (signature) :: tail
244       when match_signature signature name vars ->
245         Solution (cnstAndAll vars signature.params,
246           (fun unit -> eval_loop tail))
247     | Rule (signature, exec) :: tail
248       when match_signature signature name vars ->
249         let matchedVars = cnstAndAll vars signature.params in
250         run_gen tail (fun unit -> exec db addDB matchedVars)
251     | head :: tail -> eval_loop tail
252   in
253     (* print_string ("In: " ^ (string_of_eval name vars));*)
254     eval_loop (!addDB @ !db)
255 ;;
256

```

```

257 let rec list_replace i e list =
258   match list with
259     [] -> []
260   | hd :: tl ->
261     if i = 0
262     then e :: tl
263     else hd :: (list_replace (i - 1) e tl)
264 ;;
265
266 let rec range i j = if i >= j then [] else i :: (range (i+1) j)
267
268 let parseDB (prog) =
269
270   (* Our only compiler directive is print, for now.
271      learn/forget have a special syntax *)
272   let parseCompilerDirective name params =
273     let nc = String.compare name in
274     (* Print something... probably just "Hello World" *)
275     if (nc "print") = 0
276     then
277       fun db addDB cnst ->
278         let print_param param =
279           match param with
280             Tst.Lit(i) -> print_int i
281           | Tst.Str(s) -> print_string s
282           | Tst.Sym(s) -> print_string s
283           | Tst.Var(i) -> print_string (string_of_cnst (List.nth cnst i))
284           | _ -> ()
285         in
286         (List.iter print_param params;
287          print_string "\n";
288          NoSolution)
289     else
290       (print_string "Unknown_compiler_directive";
291        fun db addDB cnst ->
292          NoSolution)
293   in
294
295   (* Compute AND blocks by cANDing all the solutions in each row
296      * of the cross product of all the possible solutions
297      *)
298   let rec parseAndBlock stmts =
299     match stmts with
300     [] -> (fun db addDB cnst -> Solution (cnst, fun unit -> NoSolution))
301   | stmt :: tail ->
302     let nextStatement = (parseAndBlock tail) in
303     let thisStatement = (parseStatement stmt) in
304     fun db addDB cnst ->
305       let nextGenMain = (nextStatement db addDB) in
306       let rec runThisGens thisGen =

```

```

308         match (thisGen ()) with
309             NoSolution -> NoSolution
310         | Solution (thisCnsts, thisGenNxt) ->
311             let rec runNextGens nextGen =
312                 match (nextGen ()) with
313                     NoSolution ->
314                         runThisGens thisGenNxt
315                     | Solution(nextCnsts, nextGenNxt) ->
316                         Solution(nextCnsts, fun unit -> runNextGens nextGenNxt)
317             in
318                 runNextGens (fun unit -> nextGenMain thisCnsts)
319     in
320         runThisGens (fun unit -> thisStatement db addDB cnst)
321
322
323     (* Return all the solutions from one, then go to the next *)
324 and parseOrBlock stmts =
325     match stmts with
326     [] -> (fun db addDB cnst -> NoSolution)
327     | stmt :: tail ->
328         let nextStmt = (parseOrBlock tail) in
329         let currStmt = (parseStatement stmt) in
330         fun db addDB cnst ->
331             let rec runOr nxt =
332                 match nxt with
333                     NoSolution -> nextStmt db addDB cnst
334                     | Solution(vars, nxt) -> Solution(vars,
335                                                         (fun unit -> runOr (nxt ())))
336             in
337                 runOr (currStmt db addDB cnst)
338
339
340
341     (* Return the results from a query *)
342 and parseEval name params =
343     let param_var_index var_idx =
344         let rec pvi_iter plist idx =
345             match plist with
346             [] -> -1
347             | Tst.Var(i) :: tl when i == var_idx -> idx
348             | _ :: tl -> pvi_iter tl (idx + 1)
349         in
350             pvi_iter params 0
351     in
352     fun db addDB cnst ->
353         let cnsts = cnst_of_params params cnst in
354         (* Map the slots returned from the eval into our slot-space *)
355         let revMap rCnsts =
356             List.map2
357                 (fun cnst idx ->
358                     let pIdx = param_var_index idx in

```

```

359         if pIdx == -1
360         then cnst
361         else cAnd cnst (List.nth rCnsts pIdx))
362     cnst
363     (range 0 (List.length cnst))
364 in
365     (* Run the eval, then send back the results, reverse mapping the slots as we go *)
366 let nxt = run_eval db addDB name cnsts in
367 let rec doNxt nxt =
368     match nxt with
369     | NoSolution -> NoSolution
370     | Solution(rCnsts, nxt) ->
371         (* print_string (string_of_eval name rCnsts); *)
372         let rCnsts = revMap (list_first (List.length params) rCnsts) in
373         (* print_string (string_of_eval name rCnsts); *)
374         Solution(rCnsts, (fun unit -> doNxt (nxt ())))
375 in
376     doNxt nxt
377
378
379 (* ***** BEHOLD — The bane of my existence!!!!!! *)
380
381 (* A dumber man could not have written this function...)
382 * ... A smarter man would have known not to.
383 *)
384 and parseNotEval name params =
385 let eval = parseEval name params in
386 fun db addDB cnsts ->
387     (* It probably will help to think of this function as a binary blob...)
388     * I blacked out while I was writing it, but I remember it having
389     * something to do with lazily-generated cross products. ~John
390     *)
391 let rec iter_outs bigList =
392     (* Printf.printf "%s\n" (string_of_eval "Level:" (List.hd bigList)); *)
393     match bigList with
394     | [] -> failwith "Internal_error_23"
395     | myRow :: [] ->
396         let rec linearGen myList =
397             match myList with
398             | [] -> NoSolution
399             | hd :: tl -> Solution([hd], fun unit -> linearGen tl)
400         in
401             linearGen myRow
402     | myRow :: tl ->
403         let tlGenMain = iter_outs tl in
404         let rec twoGen myList nxtGen =
405             match myList with
406             | [] -> NoSolution
407             | myHd :: myTl ->
408                 match nxtGen with
409                 | NoSolution ->

```

```

410             twoGen myTl tlGenMain
411             | Solution(sol, nxtGen) ->
412               Solution(myHd :: sol, fun unit -> twoGen myList (nxtGen()))
413         in
414           twoGen myRow tlGenMain
415     in
416
417     (* Iterate through all the solutions, subtracting all the new solutions
418      * from the existing ones being stored in 'outs'
419      *)
420     let rec minus nxt outs =
421       match nxt with
422         NoSolution ->
423           iter_outs outs
424       | Solution(evCnsts, nxt) ->
425         minus
426           (nxt())
427           (* (list_acc (fun out -> List.map2 cMinus out evCnsts) outs) *)
428           (List.map2
429             (fun out evCnst ->
430               list_acc (fun o -> cMinus o evCnst) out)
431             outs
432             evCnsts)
433     in
434       (* Start with the input solution, and subtract all the results *
435       minus (eval db addDB cnsts) (List.map (fun c -> [c]) cnsts)
436
437
438       (* Run an eval in somebody else's database *)
439     and parseDot2 v pred params =
440       let eval = parseEval pred params in
441       (fun db addDB cnst ->
442         match (List.nth cnst v) with
443           CEqlAgent(adb) ->
444             eval adb (ref []) cnst
445         | a -> (Printf.printf
446               "Warning: attempted_dot_( '.')_on_a_non-agent:_%s\n"
447               (string_of_cnst a);
448               NoSolution))
449     and parseNDDot2 v pred params =
450       let eval = parseNotEval pred params in
451       (fun db addDB cnst ->
452         match (List.nth cnst v) with
453           CEqlAgent(adb) ->
454             eval adb (ref []) cnst
455         | a -> (Printf.printf
456               "Warning: attempted_dot_( '.')_on_a_non-agent:_%s\n"
457               (string_of_cnst a);
458               NoSolution))
459
460

```

```

461 and doAnd myCnsts db addDB cnst =
462   let sol = cnstAndAll myCnsts cnst in
463     (* (print_string (string_of_eval "" myCnsts));
464       (print_string (string_of_eval "" cnst)); *)
465     if List.for_all (fun a -> a != FalseSol) sol
466     then Solution(sol, fun () -> NoSolution)
467     else NoSolution
468 and parseCompOp op v e2 =
469   let compOp i =
470     match op with
471     | Ast.Lt -> CLT(i)
472     | Ast.Gt -> CGT(i)
473     | Ast.Leq -> CLT(i + 1)
474     | Ast.Geq -> CGT(i - 1)
475     | Ast.Eq -> CEqlInt(i)
476     | _ -> failwith "Unsupported_comparison_operator"
477   in
478     match e2 with
479     | Tst.ELit(i) ->
480       doAnd ((list_fill Any v) @ [(compOp i)])
481 and parseStrComp v s =
482   doAnd ((list_fill Any v) @ [CEqlStr(s)])
483 and parseSymComp v s =
484   doAnd ((list_fill Any v) @ [CEqlSymbol(s)])
485 and parseLearnForget name statements =
486   let remove_facts db addDB cnsts =
487     let remove_fact (name, params) =
488       db := remove_fact_all !db name (cnst_of_params params cnsts)
489     in
490       List.iter remove_fact statements
491   in
492     let remove_fact1 db addDB cnsts =
493       let remove_fact (name, params) =
494         db := remove_fact1 !db name (cnst_of_params params cnsts)
495       in
496         List.iter remove_fact statements
497   in
498     let add_facts db addDB cnsts =
499       let add_fact (name, params) =
500         db := Fact({name = name; params = (cnst_of_params params cnsts)}) :: !db
501       in
502         List.iter add_fact statements
503   in
504     let nm = String.compare name in
505       if (nm "learn") = 0
506       then (fun db addDB cnsts -> add_facts db addDB cnsts; NoSolution)
507       else if (nm "forget") = 0
508       then (fun db addDB cnsts -> remove_facts db addDB cnsts; NoSolution)
509       else if (nm "forget1") = 0
510       then (fun db addDB cnsts -> remove_fact1 db addDB cnsts; NoSolution)
511       else failwith ("Invalid_directive:_" ^ name)

```



```

512 and parseDot1 v dname statements =
513   let study = parseLearnForget dname statements in
514   (fun db addDB cnst ->
515     match (List.nth cnst v) with
516     | CEqlAgent(adb) ->
517       study adb (ref []) cnst
518     | a -> (Printf.printf
519       "Warning: attempted to use a non-agent: %s\n"
520       (string_of_cnst a);
521       NoSolution))
522 and parseStatement statement =
523   match statement with
524   | Tst.Block (redOp, statements)
525     when 0 == (String.compare redOp "AND") ->
526     parseAndBlock statements
527   | Tst.Block (redOp, statements)
528     when 0 == (String.compare redOp "OR") ->
529     parseOrBlock statements
530   | Tst.Block (redOp, statements) ->
531     (Printf.printf "Invalid reduction operator %s\n" redOp;
532     (fun db addDB cnst -> NoSolution))
533   | Tst.Eval (name, params) ->
534     parseEval name params
535   | Tst.NEval (name, params) ->
536     parseNotEval name params
537   | Tst.Directive (name, params) ->
538     parseCompilerDirective name params
539   | Tst.Comp(e1, compOp, e2) ->
540     parseCompOp compOp e1 e2
541   | Tst.DirectiveStudy(name, statements) ->
542     parseLearnForget name statements
543   | Tst.StrComp(v, s) ->
544     parseStrComp v s
545   | Tst.SymComp(v, s) ->
546     parseSymComp v s
547   | Tst.Dot1(v, dname, statements) ->
548     parseDot1 v dname statements
549   | Tst.Dot2(v, pred, params) ->
550     parseDot2 v pred params
551   | Tst.NDot2(v, pred, params) ->
552     parseNDot2 v pred params
553 in
554 let parseRule stmt slots actions =
555   fun db addDB inCnsts ->
556   let rec runPer sols nxt =
557     match nxt with
558     | NoSolution -> NoSolution
559     | Solution(outCnsts, nxt) ->
560       (* Have we already given this solution? *)
561       if (List.mem outCnsts sols)
562       then runPer sols (nxt())

```

```

563         else
564             (List.iter
565              (fun action ->
566               (ignore (action db addDB outCnsts)))
567              actions;
568              Solution(outCnsts, fun () -> runPer (outCnsts :: sols)(nxt())))
569     in
570     (* print_string ("Num slots: " ^ (string_of_int slots) ^ "\n"); *)
571     runPer [] (stmt db addDB (cnst_extend_to inCnsts slots))
572 in
573 let parseRF = function
574   Tst.Rule (name, parms, numVars, statement, nseStmt) ->
575     Rule ({ name = name; params = (sig_to_cnst parms)} ,
576           (parseRule (parseStatement statement) numVars
577                    (List.map parseStatement nseStmt)))
578   | Tst.Fact (name, parms) ->
579     Fact ({ name = name; params = (sig_to_cnst parms)})
580 in
581 let tProg = Trans.translate(prog) in
582   ref (List.map parseRF tProg)
583 ;;
584
585 let query db addDB pred numVars =
586   run_eval db addDB pred (list_fill Any numVars)
587 ;;
588
589 let rec dump_db db =
590   let print_sig s =
591     Printf.printf "%s(%s)"
592       s.name
593       (String.concat "," (List.map string_of_cnst s.params))
594   in
595   let dump_rf rf =
596     match rf with
597     | Fact(s) ->
598       print_sig s;
599       print_string ";\n"
600     | Rule(s, f) ->
601       print_sig s;
602       print_string "_{ }\n"
603   in
604     List.iter dump_rf db
605 ;;

```

Listing 56: C μ LOG Interpreter