

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

Contents

1	Introduction: CμLOG	3
1.1	Application & Features	3
1.2	Goals	3
2	Tutorial	4
2.1	Variables	4
2.2	Statements	4
2.3	Facts	5
2.4	Rules	5
2.5	Directives	6
2.6	Simulator	6
2.7	Program Modification	7
2.8	Breakout	9
3	Language Reference Manual	9
3.1	Lexical	9
3.2	Facts	10
3.3	Rules	11
3.4	Variables	11
3.5	Blocks	11
3.6	Statements	12
3.7	Comparisons	13
3.8	Types	13
3.9	Directives	13
3.10	Program Modification	13
3.11	Interaction- The Dot Operator	14
4	Project Plan	14
4.1	Responsibilities	14
4.2	Timeline	14
4.3	Software Development Environment	15
5	Architecture	15
6	Test Plan	16
6.1	Testing Script	16
6.2	Test Case Rationale	18
6.3	Testing Results	18

7	Lessons Learned	18
7.1	From Devesh	18
7.2	From John	19
7.3	From Nishant	20
7.4	From Cheng	20
A	Appendix: Test Cases	20
A.1	andTest	21
A.2	facts	21
A.3	learnForget1	22
A.4	main	23
A.5	main fall through	24
A.6	mult-main	25
A.7	neq	25
A.8	not1	26
A.9	plist-twice	26
A.10	printer test	27
A.11	prsimple	30
A.12	prstrings	30
A.13	range	31
A.14	sim dot1	32
A.15	sim dot2	35
A.16	sim my loc	39
A.17	sim ndot2	42
A.18	sim two test	43
A.19	simulator test	48
A.20	sprint1	52
B	Appendix: Code Listings	54
B.1	parser.mly	54
B.2	scanner.mll	56
B.3	ast.mli	57
B.4	printer.ml	58
B.5	tst.mli	60
B.6	trans.ml	61
B.7	culog.ml	66
B.8	simulator.ml	67
B.9	interp.ml	73

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 μ LOG can be very simple- such as facts. The relationship between code and data is also of note. C μ 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");
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:

```

$X < 10; // A comparison
range($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 $C\mu$ LOG 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

$C\mu$ LOG 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 $C\mu$ LOG 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

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:

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

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

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 Ocamllyacc 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.

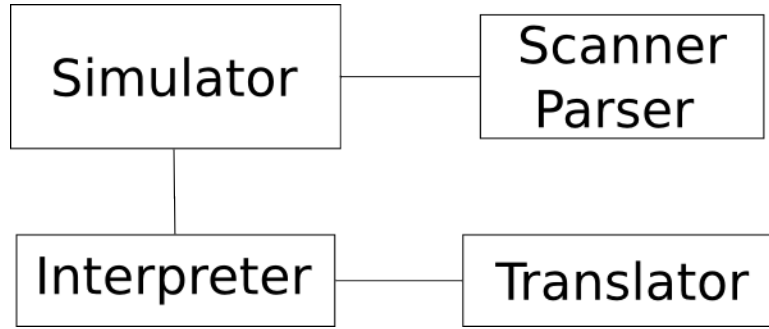
5 Architecture

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 synatax 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 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

Figure 1: Architecture Block Diagram



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

6.1 Testing Script

The python script shown in listing 1 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 Test Case Rationale

Most of our test cases are written to test a specific feature. For instance, “andTest.ul” is designed to test AND blocks. To whatever extent possible, these tests avoid testing other features. This makes it easier to determine what feature has been broken when tests start failing. Other tests are designed to fail to make sure that various parts of the system fail properly. Still other tests are composite tests and are designed to test the system as a whole- they test multiple features at once to ensure that there are not bizarre interactions between various parts of the system.

6.3 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

7.1 From Devesh

Programming languages and translators project introduced me to a whole new world of programming both logic and functional. In the first project meeting we envisioned a programming language to simulate agent movement in a grid based environment. The most obvious choice was having a logic level programming language. The learning course started with choosing a convenient and accurate grammar. After that working

on the parser and scanner made me realize the power of type checking in ocaml. I also learnt how to logically reason and resolve the shift reduce and reduce reduce conflicts.

Implementation of the interpreter introduced me the power of recursive programming. Writing tests help me find bugs in the language, also it taught me that the testing is an ongoing process. Working in a team and meeting self made deadlines was also part of my learning.

After the working on this project I found it much easier to understand the syntax and program in "Murphy" a formal verification language. Along with programming languages it also exposed me to shell scripts, Makefiles, Svn repositories.

7.2 From John

A few rules to live by:

Tools Don't use the wrong tools for the right job or the right tools for the wrong job. Even parity is required! The interpreter is written to lazily evaluate queries both to reduce memory usage and avoid infinite loops in cases of infinite solutions. This lazy evaluation strategy would have been much easier to implement either with co-routines or lazy evaluation (a la Haskell.) Since OCaml offers neither of these features, I implemented lazy evaluation by hand, and it made everything harder by an order of magnitude. Since I didn't have the proper tools available, I shouldn't have written lazy evaluation. I should have created a logic solver which could only operate on a smaller class of programs, and would return all the results at once.

Testing Everyone tests while they write code. You have to. Frequently, you write tests in a temporary file and discard when the feature is "working." Don't do this. It's almost always worth the extra time to set up a test bed and put your tests in it. Then keep them. Run them often. Passing tests gives you a warm, fuzzy feeling which grows with the number of tests. So, keep all your "temporary" tests and you'll not only feel better about yourself, but you'll be ensuring long-term quality.

Refactoring At first, you don't know the features of the library and language. You'll write an AST with unnecessary boxing and unboxing. You'll re-write List.filter, and use lambdas when currying would have done the job. Realizing it later on is the mark of a good programmer. Refactoring this code is the mark of a diligent one.

BYOT Bring your own tools! Tools are what separate the chaff from the wheat. If you need something done and a tool can do the job, write the tool. Scripting languages are great for this, so one of the best time savers is intimate knowledge of a scripting language. Any language will do, but you can't be afraid of using it. Python is my pocket knife of choice, and as far as I'm concerned, there's no such thing as abuse!

Recycling Does the code you're writing right now look a lot like some of the code you wrote yesterday? Don't write it again, refactor the old code into a more generic function. I won't claim to be an angel, but I'm sure that "copy and paste" are tools of the devil.

7.3 From Nishant

Programming Languages and Translators, is my first ever core CS course. Being an EE student it was difficult but seemed interesting enough a course to be taken. The motivation behind taking this course was to learn about programming languages and the working of a translator and the various components a language interpreter/translator/compiler is made of. Another reason was to get involved in a programming project to get a feel of programming and think as a system programmer, designing stuff for the end user.

After taking the class and brain storming with the group members, the thought was to create this language and looked unsurmountable to me. As a niche programmer I learnt a few valuable lessons in regards of thinking as a system programmer. This class and the project has introduced me to many different types of languages, like Prolog (logical) and obviously OCaml(functional). Learning how to program in OCaml took some doing. Handling the return types and recursion was not easy. But after a full course and the project it is possible to write in OCaml. Adapting to a new language, was a very useful thing learnt as well. While using a system modelling language called Promela for another class, I found it extremely easy to adapt to it. The other most important thing I learnt was errors, their types and their origin. To conclude, this project has taught me the tricks of the trade to “program” and given me the tool, “OCaml”.

7.4 From Cheng

From this project, i have learned not only how to construct an interpreter step by step but also how to collaborate with my teammates.

Basically, there are three most important things i have learned from this project. First, I have acquired a deep understanding of basic concepts covered in the class. This is definitely helpful when i am trying to study a new programming language. Specifically, I can quickly learn how to program using this language and solve the compiling errors as soon as i know about the general features of its compiler (such as naming and scoping rules). Moreover, the success of the project is largely determined by its design. With a good design, coding part will be more easily. However, with a bad design, it will be a painful experience. So it is wise to spend most time on the software design. Finally, teamwork plays a key role in software development. In this project, I have improved my skills of communication. With better communication, the whole team can work more efficiently.

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)

```


A.19 simulator test

```
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");
```

```
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 }
```

```

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 . . . . .

```

51

```

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 | EId 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  *   Originally written in group, typed by John Demme
5  *   Updated as necessary by everyone
6  *)
7
8  open Ast
9
10 let string_of_compoperator = function
11   Lt   -> "<"
12   | Leq -> "<="
13   | Gt   -> ">"
14   | Geq  -> ">="
15   | Eq   -> "="
16   | Neq  -> "!="
17
18 let string_of_operator = function
19   Plus  -> "+"
20   | Minus -> "-"
21   | Mult  -> "*"

```

```

22 | Divide -> "/"
23
24
25 let rec string_of_expr = function
26   Binop(e1, o, e2) -> (string_of_expr e1) ^ (string_of_operator o) ^ (string_of_expr e2)
27 | ELit(i) -> string_of_int i
28 | EVar(s) -> s
29 | EStr(s) -> s
30 | EId(s) -> s
31
32 let rec string_of_param = function
33   Lit(i) -> string_of_int i
34 | Sym(s) -> s
35 | Var(s) -> s
36 | Str(s) -> "\"" ^ s ^ "\""
37 | Arr(a) -> "[" ^ string_of_params a ^ "]"
38 | Ques -> "?"
39
40
41 and string_of_params = function
42   Params(pList) -> String.concat "," (List.map string_of_param pList)
43 | Array(pList) -> String.concat "," (List.map string_of_param pList)
44
45 let rec string_of_stmts = function
46   Stmts(sList) -> String.concat "\n" (List.map string_of_stmt sList)
47 and string_of_stmt = function
48   Block(red, stmts) -> "{" ^ red ^ ":\n" ^ (string_of_stmts stmts) ^ "\n}"
49 | Comp(e1, c, e2) -> (string_of_expr e1) ^ (string_of_compoperator c)
50   ^ (string_of_expr e2) ^ ";";
51 | Eval(name, ps) -> name ^ "(" ^ (string_of_params ps) ^ ");";
52 | NEval(name1, ps1) -> "!" ^ name1 ^ "(" ^ (string_of_params ps1) ^ ");";
53 | DirectiveStudy(name, stmts) -> "@" ^ name ^ "(" ^
54   (string_of_stmts (Stmts (List.map (fun a -> Eval(a)) stmts))) ^ ")"
55 | Directive(name, params) -> "@" ^ name ^ "(" ^ (string_of_params params) ^ ")"
56 | Dot1(str1, str2, stmts) -> str1 ^ "." ^ "@" ^ str2 ^ "(" ^
57   (string_of_stmts (Stmts (List.map (fun a -> Eval(a)) stmts))) ^ ")"
58 | Dot2(str1, str2, ps) -> str1 ^ "." ^ str2 ^ "(" ^ (string_of_params ps) ^ ");";
59 | NDot2(str1, str2, ps) -> "!" ^ str1 ^ "." ^ str2 ^ "(" ^ (string_of_params ps) ^ ");";
60
61
62 let string_of_ruleFact = function
63   Rule(name, params, stmt) -> name ^ "(" ^ (string_of_params params) ^ "),"
64   ^ (string_of_stmt stmt)
65 | Fact(name, params) -> name ^ "(" ^ (string_of_params params) ^ ");";
66 | GlobalDirective(name, ps) -> "@" ^ name ^ "(" ^ (string_of_params ps) ^ ");";
67
68 let string_of_program = function
69   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  Copied and modified from ast.mli
8
9  Written by John Demme
10
11 *)
12 type param =
13   Lit   of int
14   | Sym of string
15   | Var of int
16   | Anon
17   | Str of string
18   | Arr of param list
19
20 and params = param list
21
22 type expr =
23   | ELit   of int
24
25 type eval = string*params
26 type var  = int
27
28 type stmt =
29   Block of string*stmts                                     (* {.....} *)
30   | Comp of var*Ast.compoperator*expr                       (* $5+5<$4  $a=5,$b=6; *)
31   | StrComp of var*string
32   | SymComp of var*string
33   | NEval of eval                                           (* !wall(4,5) *)
34   | Eval of eval                                             (* wall(4,5) *)
35   | DirectiveStudy of string*(eval list)                    (* @learn(wall(4,5);) *)
36   | Directive of string*params                               (* @print("dfdsf"); *)
37   | Dot1 of int*string*(eval list)                           (* $agent.@learn(wall(4,5);) *)
38   | Dot2 of int*string*params                                 (* env.view($X,$Y,$Obj) *)
39   | NDot2 of int*string*params                               (* !env.view($X,$Y,$Obj) *)
40
41 and stmts=stmt list    (* statement1;statement2;statement3; *)
42
43 type ruleFact =
44   | Rule of string * params * int * stmt * stmt list
45   | Fact of string * params
46
47
48 type program = ruleFact list

```

B.6 trans.ml

```
1  (* Functions to modify the AST slightly
2     to make parsing it easier for the interpreter.
3
4     Static checking happens 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 paramaters 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 simplyify 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   <teqdruidd@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 obataining
5  * information about agents and environment(grid) and simulate
6  * their behaviors and the other being output driver.
7  *
8  * Original authors:
9  *   Cheng Cheng
10 *   (worked on communicating with interpreter and
11 *    simulating the interaction)
12 *   Nishant Shah
13 *   (worked on integrating the output driver with the simulator)
14 *
15 * Support for loading multiple agents and separating the programs
16 *   added by John Demme
17 *)
18
19 open Interp
20 open Ast
21
22 (* define global references to the parameters of environment*)
23 let grid_size_ref=ref 1;;
24 let grid_x_size_ref=ref 1;;
25 let grid_y_size_ref=ref 1;;
26 let goal_x_ref=ref 1;;
27 let goal_y_ref=ref 1;;
28
29 (* define data structure of agent*)
30 type sim_agent = {

```

```

31  x    : int;
32  y    : int;
33  sym  : char;
34  db   : database
35 }
36
37 (* define a global array to restore information of wall and positions of agents *)
38 (* maximum environment size is 100*100 *)
39 (* '.' represents empty grid, '|' represents wall*)
40 let record=
41   let f index='.' in
42     Array.init 10000 f ;;
43
44 let clear_array a=
45   for index=0 to (Array.length a)-1 do
46     if index=(!grid_y_size_ref- !goal_y_ref)* !grid_x_size_ref+ !goal_x_ref-1 then
47       begin
48         a.(index)<-'#'
49       end
50     else a.(index)<- '.'
51   done
52 ;;
53
54 let sim_exit s =
55   Printf.printf "\nSimulation over: %s\n\n" s;
56   exit(1)
57 ;;
58
59 (* set the size of environment *)
60 let rec set_size nxt=
61   match nxt with
62   | NoSolution-> ()
63   | Solution(c,n)->
64     (match c with
65      [ CEqInt(x); CEqInt(y)]-> if x<1||x>100 then failwith "the_length_of_grid_is_
66      else if y<1||y>100 then failwith "the_width_of_grid_is_not_illegal!!!"
67      else
68        begin
69          grid_x_size_ref:=x;
70          grid_y_size_ref:=y;
71          grid_size_ref:=x*y
72        end
73      | _-> ())
74 ;;
75
76 (* set the goal agents try to reach*)
77 let rec set_goal nxt=
78   match nxt with
79   | NoSolution-> ()
80   | Solution(c,n)->
81     (match c with

```

```

82         [CEqlInt(x);CEqlInt(y)]-> if x<1||x> !grid_x_size_ref then failwith "illegal_
83         else if y<1||y> !grid_x_size_ref then failwith "illegal_goal_y_position"
84         else
85             begin
86                 goal_x_ref:=x;
87                 goal_y_ref:=y;
88             end
89         |_->())
90 ;;
91
92 (* display and output the results after interactions*)
93 let print_grid oc arr =
94     for a=0 to !grid_y_size_ref-1
95     do
96         for j= !grid_x_size_ref*(a) to !grid_x_size_ref*(a+1)-1
97         do
98             Printf.fprintf oc "%c_" arr.(j)
99         done;
100         Printf.fprintf oc "\n"
101     done
102 ;;
103
104 let print_file j arr =
105     let file = "Agent"^string_of_int(j)^".dat" in (* Write message to file *)
106     let oc = open_out file in (* create or truncate file , return channel *)
107     (print_grid oc arr;
108      close_out oc)
109 ;;
110
111 let print_stdout j arr =
112     Printf.printf "\n=====Turn_%d=====\\n" j;
113     print_grid stdout arr;
114     print_string "\n"
115 ;;
116
117 (* create wall in environment*)
118 let create_wall x_start x_end y_start y_end =
119     if x_start<1 || x_end> !grid_x_size_ref then
120         failwith "Creating_Wall_:x_position_of_wall_exceeds_the_grids"
121     else if y_start<1 || y_end> !grid_y_size_ref then
122         failwith "Creating_Wall_:y_position_of_wall_exceeds_the_grids"
123     else if x_start>x_end || y_start>y_end then failwith "Creating_Wall:wrong_range!!!"
124     else for i=x_start to x_end do
125         for j=y_start to y_end do
126             record.((!grid_y_size_ref-j)* !grid_x_size_ref+i-1)<-'|'
127         done
128     done
129 ;;
130
131
132 (* obtain wall information from interpretor and create walls*)

```

```

133 let rec iter_wall nxt=
134     match nxt with
135     | NoSolution -> ()
136     | Solution(c,n) ->
137         (match c with
138         | [Any; Any]-> create_wall 1 !grid_x_size_ref 1 !grid_y_size_ref
139         | [CEqInt(x);Any]-> create_wall x x 1 !grid_y_size_ref
140         | [CLT(x);Any]-> create_wall 1 (x-1) 1 !grid_y_size_ref
141         | [CGT(x);Any]-> create_wall (x+1) !grid_x_size_ref 1 !grid_y_size_ref
142         | [CRange(x1,x2);Any]->create_wall (x1+1) (x2-1) 1 !grid_y_size_ref
143         | [Any;CEqInt(y)]-> create_wall 1 !grid_x_size_ref y y
144         | [Any;CLT(y)]-> create_wall 1 !grid_y_size_ref 1 (y-1)
145         | [Any;CGT(y)]-> create_wall 1 !grid_x_size_ref (y+1) !grid_y_size_ref
146         | [Any;CRange(y1,y2)]-> create_wall 1 !grid_x_size_ref (y1+1) (y2-1)
147         | [CEqInt(x);CEqInt(y)]-> create_wall x x y y
148         | [CEqInt(x);CLT(y)]-> create_wall x x 1 (y-1)
149         | [CEqInt(x);CGT(y)]-> create_wall x x (y+1) !grid_y_size_ref
150         | [CEqInt(x);CRange(y1,y2)]-> create_wall x x (y1+1) (y2-1)
151         | [CLT(x);CEqInt(y)]-> create_wall 1 (x-1) y y
152         | [CLT(x);CLT(y)]-> create_wall 1 (x-1) 1 (y-1)
153         | [CLT(x);CGT(y)]-> create_wall 1 (x-1) (y+1) !grid_y_size_ref
154         | [CLT(x);CRange(y1,y2)]-> create_wall 1 (x-1) (y1+1) (y2-1)
155         | [CGT(x);CEqInt(y)]-> create_wall (x+1) !grid_x_size_ref y y
156         | [CGT(x);CLT(y)]-> create_wall (x+1) !grid_x_size_ref 1 (y-1)
157         | [CGT(x);CGT(y)]-> create_wall (x+1) !grid_x_size_ref (y+1) !grid_y_size_ref
158         | [CGT(x);CRange(y1,y2)]-> create_wall (x+1) !grid_x_size_ref (y1+1) (y2-1)
159         | [CRange(x1,x2);CEqInt(y)]-> create_wall (x1+1) (x2-1) y y
160         | [CRange(x1,x2);CLT(y)]-> create_wall (x1+1) (x2-1) 1 (y-1)
161         | [CRange(x1,x2);CGT(y)]-> create_wall (x1+1) (x2-1) (y+1) !grid_y_size_ref
162         | [CRange(x1,x2);CRange(y1,y2)]->create_wall (x1+1) (x2-1) (y1+1) (y2-1)
163         | _ -> ());
164     iter_wall (n ())
165 ;;
166
167 (* agent moves towards to a direction*)
168 let agent_move a direction =
169     Printf.printf "%c:_Moving_%s\n" a.sym direction;
170     match direction with
171     | "UP" -> {x = a.x; y = a.y + 1; db = a.db; sym = a.sym}
172     | "DOWN" -> {x = a.x; y = a.y - 1; db = a.db; sym = a.sym}
173     | "LEFT" -> {x = a.x - 1; y = a.y; db = a.db; sym = a.sym}
174     | "RIGHT"-> {x = a.x + 1; y = a.y; db = a.db; sym = a.sym}
175     | _ ->failwith "No_such_a_direction!"
176 ;;
177
178 (*simulator stores the information of agent's move *)
179 (*if agent reaches the goal or hits wall, simulator terminates *)
180 let do_agent_move a =
181     let array_index=(!grid_y_size_ref - a.y)* !grid_x_size_ref + a.x-1 in
182     if a.x < 1|| a.x > !grid_x_size_ref then (*x position is beyond range *)
183         begin

```

```

184         let str=(Char.escaped a.sym)^" _hits_the_y_margin_and_Game_over!!! " in
185         sim_exit str
186     end
187 else if a.y < 1 || a.y > !grid_y_size_ref then (*y position is beyond range *)
188     begin
189         let str=(Char.escaped a.sym)^" _hits_the_x_margin_and_Game_over!!! " in
190         sim_exit str
191     end
192 else if Array.get record array_index = '|' then
193     begin
194         let str=(Char.escaped a.sym)^" _hits_the_wall_and_Game_over!!!" in
195         sim_exit str
196     end
197 else if Array.get record array_index = '#' then
198     begin
199         let str=(Char.escaped a.sym)^" _wins!!! Successfully_reach_the_goal_at_position_(" ^ s
200 ^", " ^ string_of_int (!goal_y_ref)^")"
201         in
202         sim_exit str
203     end
204 else if (Array.get record array_index) != '.' then
205     begin
206         let str="Game_over!!! Agents_crash!!! _at_position_(" ^ string_of_int(a.x)^", " ^ string_of_int(a.y)^")"
207         in
208         sim_exit str
209     end
210 else record.(array_index)<- a.sym
211 ;;
212 (* obtain current position of agent from interpreter and make it move*)
213 let iter_move agent nxt =
214     match nxt with
215     | NoSolution -> failwith "No_Solution"
216     | Solution ([CEqlStr(dir)], -) ->
217         let new_agent = agent_move agent dir in
218         ignore (do_agent_move new_agent);
219         new_agent
220     | _ -> failwith "Invalid_(or_no)_move"
221 ;;
222 (* load the databases of all agents in environment*)
223 let my_loc_db agent all env =
224     ref ([Interp.Fact({name = "loc"; params = [CEqlInt(agent.x); CEqlInt(agent.y)]});
225         Interp.Fact({name = "env"; params = [CEqlAgent(env)]});
226         @
227         (List.map
228             (fun other ->
229                 Interp.Fact({name = "agent"; params = [CEqlAgent(other.db)]})
230             ) (List.filter (fun a -> a != agent) all)))
231 ;;
232 (* simulation function*)
233 let simulation envDB agents =

```



```

234   let rec loop i agents =
235       let sGen_size=query envDB (ref []) "size" 2 in
236       set_size sGen_size;
237       let sGen_goal=query envDB (ref []) "goal" 2 in
238       set_goal sGen_goal;
239       clear_array record;
240       let sGen_wall=query envDB (ref []) "wall" 2 in
241       iter_wall sGen_wall;
242       let new_agents =
243           List.map
244               (fun agent ->
245                   let sGen_move = query agent.db (my_loc_db agent agents envDB) "move" 1 in
246                   iter_move agent sGen_move)
247               agents
248       in
249       print_stdout i record;
250       if i>100 then sim_exit "You lose! Can not reach the goal with in 100 steps"
251       else loop (i+1) new_agents
252   in loop 1 agents
253 ;;
254
255
256 let load_agent db_loc =
257     match db_loc with
258     (c, s) ->
259         let lexbuf1 = Lexing.from_channel (open_in s) in
260         let program = Parser.program Scanner.token lexbuf1 in
261         {x=1; y=1; sym = (String.get c 0); db = Interp.parseDB(program)}
262 ;;
263
264 (*load database of rules and facts for a single agent*)
265 let load_db db_loc =
266     let lexbuf1 = Lexing.from_channel (open_in db_loc) in
267     let program = Parser.program Scanner.token lexbuf1 in
268     {x=1; y=1; sym = 'x'; db = Interp.parseDB(program)}
269 ;;
270
271
272 let get_agent_locs db =
273     let rec gal_int res =
274         match res with
275         NoSolution -> []
276         | Solution([CEqlStr(c); CEqlStr(s)], nxt) -> (c,s) :: (gal_int (nxt()))
277         | _ -> failwith "Failed to load agent"
278     in
279     gal_int (query db (ref []) "agent" 2)
280 ;;
281
282
283 let _ =
284     let envDB = load_db Sys.argv.(1) in

```

```

285   let agent_locs = get_agent_locs envDB.db in
286   if 0 == (List.length agent_locs)
287   then simulation envDB.db [envDB]
288   else
289     let agentDBs = List.map load_agent agent_locs in
290     simulation envDB.db agentDBs
291 ;;

```

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 *   This whole guy is written for composability. Each function takes a database and
11 *   variable constraints and returns type "next". Each composite function (like evals,
12 *   and blocks) run their sub functions, look at the results, mutate then appropriately
13 *   and return the results as "next"s. All of this happens lazily.
14 *
15 *   John Demme
16 *
17 *)
18
19 (* Each variable can be constrained in any of these ways *)
20 type var_cnst =
21   Any
22 | FalseSol
23 | CEqlSymbol of string
24 | CEqlInt    of int
25 | CEqlStr    of string
26 | CLT        of int
27 | CGT        of int
28 | CRange     of int*int
29 | CEqlAgent  of database
30
31 (* Constraints is a list of variables *)
32 and cnst = var_cnst list
33
34 and signature = {
35   name   : string;
36   params : cnst
37 }

```

```

38
39 (* This guy is how we do our lazy evaluation *)
40 and next =
41   NoSolution
42   | Solution of cnst * (unit -> next)
43
44 (* A list of these guys makes up our database *)
45 and rule_fact =
46   Fact of signature
47   | Rule of signature * (database -> database -> cnst -> next)
48
49 and database = rule_fact list ref
50 ;;
51
52
53 let string_of_cnst = function
54   Any -> "Any"
55   | FalseSol -> "False"
56   | CEqlSymbol(s) -> s
57   | CEqlStr(s) -> "\"" ^ s ^ "\""
58   | CEqlInt(i) -> string_of_int i
59   | CLT(i) -> "<" ^ (string_of_int i)
60   | CGT(i) -> ">" ^ (string_of_int i)
61   | CRange(a,b)-> (string_of_int a) ^ ".." ^ (string_of_int b)
62   | CEqlAgent(a) -> "Agent"
63 ;;
64
65 let string_of_eval name vars =
66   name ^ "(" ^ String.concat "," (List.map string_of_cnst vars) ^ ")\n"
67 ;;
68
69 (* AND two variable constraints together *)
70 let cAnd a b =
71   let rec and_int a b t =
72     match (a, b) with
73     | (Any, _) -> b
74     | (_, Any) -> a
75     | (CEqlAgent(a1), CEqlAgent(a2)) when (a1 == a2) -> a
76     | (CEqlSymbol(s1), CEqlSymbol(s2)) when (0 == String.compare s1 s2) -> a
77     | (CEqlStr(s1), CEqlStr(s2)) when (0 == String.compare s1 s2) -> a
78     | (CEqlInt(i1), CEqlInt(i2)) when (i2 == i2) -> a
79     | (CEqlInt(i1), CGT(i2)) when (i1 > i2) -> a
80     | (CEqlInt(i1), CLT(i2)) when (i1 < i2) -> a
81     | (CLT(i1), CLT(i2)) -> CLT(min i1 i2)
82     | (CGT(i1), CGT(i2)) -> CLT(max i1 i2)
83     | (CGT(i1), CLT(i2)) when (i1 < i2) -> CRange(i1, i2)
84     | (CRange(l, u), CEqlInt(i)) when (i > l && i < u) -> b
85     | (CRange(l, u), CGT(i)) when (i < u - 1) -> CRange((max l i), u)
86     | (CRange(l, u), CLT(i)) when (i > l + 1) -> CRange(l, (min u i))
87     | (CRange(l1, u1), CRange(l2, u2)) when (u1 > l2 && u2 > l1) -> CRange((max l1 l2),
88     | (_, _) when t -> and_int b a false

```

```

89     | (-, -) -> FalseSol
90   in
91     and_int a b true
92   ;;
93
94   let range_to_int c =
95     match c with
96       CRange(l, u) when (l + 2 == u) -> CEqlInt(l+1)
97     | - -> c
98   ;;
99
100  let int_to_range c =
101    match c with
102      CEqlInt(i) -> CRange(i-1, i+1)
103    | - -> c
104  ;;
105
106  (* For each constraint, subtract the second from the first *)
107  (* TODO: There are off-by-one errors in here... Fix when you have a clearer head *)
108  let cMinus b s =
109    let cmi b s =
110      (* Too many combinations and no play makes Johnny go something something *)
111      match (b, s) with
112        (-, Any) ->
113          []
114      | (Any, -) ->
115          failwith "Unsupported_subtraction_-_need_!=_constraint"
116      | (CEqlSymbol(s1), CEqlSymbol(s2)) when (0 != String.compare s1 s2) ->
117          [b; s]
118      | (CEqlStr(s1), CEqlStr(s2)) when (0 != String.compare s1 s2) ->
119          [b; s]
120      | (CEqlInt(i1), CEqlInt(i2)) when (i1 != i2) ->
121          [b; s]
122      | (CEqlAgent(a1), CEqlAgent(a2)) when (a1 != a2) ->
123          [b; s]
124      | (CRange(bl, bu), CRange(sl, su)) when (bl < sl && bu > su) ->
125          [CRange(bl, sl); CRange(su, bu)]
126      | (CRange(bl, bu), CRange(sl, su)) when (bl < sl && bu < su) ->
127          [CRange(bl, min sl bu)]
128      | (CRange(bl, bu), CRange(sl, su)) when (bl < sl && bu > su) ->
129          [CRange(max bl su, bu)]
130      | (CRange(bl, bu), CRange(sl, su)) when (bu > sl || bl > su) ->
131          [b]
132      | (CGT(bi), CLT(si)) -> [CGT(max bi si)]
133      | (CLT(bi), CGT(si)) -> [CLT(min bi si)]
134      | (CGT(bi), CGT(si)) when (bi < si) -> [CRange(bi, si)]
135      | (CLT(bi), CLT(si)) when (bi < si) -> [CRange(si, bi)]
136    | - -> []
137  in
138    List.map range_to_int (cmi (int_to_range b) (int_to_range s))
139  ;;

```

```

140
141 (* Why isn't this in the list module?
142  *
143  * This guy iterates through all the elements of a list like map,
144  * but the function emits a list which are all appended together.
145  *)
146 let list_acc mapper list =
147   let rec acc list ret =
148     match list with
149     [] -> ret
150     | hd :: tl -> acc tl ((mapper hd) @ ret)
151   in
152     acc list []
153 ;;
154
155 (* return the first n elements of list *)
156 let rec list_first n list =
157   match list with
158   [] -> []
159   | hd :: tl when n > 0 -> hd :: list_first (n - 1) tl
160   | _ -> []
161 ;;
162
163 (* Return a list with 'number' items duplicated *)
164 let rec list_fill item number =
165   if number <= 0
166   then []
167   else item :: (list_fill item (number - 1));;
168
169
170 (* Pad the shorter or the two lists to make them the
171  * same size. Pad with 'Any's'
172  *)
173 let cnst_extend a b =
174   let delta = (List.length a) - (List.length b) in
175   if delta > 0
176   then (a, List.append b (list_fill Any delta))
177   else if delta < 0
178   then (List.append a (list_fill Any (delta * -1)), b)
179   else (a,b)
180 ;;
181
182 (* Pad a constraint list to ensure it is length l *)
183 let cnst_extend_to a l =
184   let delta = l - (List.length a) in
185   if delta > 0
186   then List.append a (list_fill Any delta)
187   else a
188 ;;
189
190 (* Extend aC and bC to be the same length and AND them *)

```

```

191 let cnstAndAll aC bC =
192   let (aC, bC) = cnst_extend aC bC in
193   List.map2 cAnd aC bC
194 ;;
195
196 (* Does signature match the term 'name' with inputs 'vars'? *)
197 let match_signature signature name vars =
198   let match_params param vars =
199     let anded = cnstAndAll param vars in
200     List.for_all (fun a -> a != FalseSol) anded
201   in
202   (String.compare signature.name name == 0) &&
203   ((List.length signature.params) == (List.length vars)) &&
204   (match_params signature.params vars)
205 ;;
206
207 (* *)
208 let remove_fact_all db pred cnsts =
209   (* print_string ("Removing: " ^ (string_of_eval pred cnsts) ^ "\n"); *)
210   List.filter
211     (fun curr ->
212       match curr with
213       | Fact(signature) when
214         match_signature signature pred cnsts -> false
215       | _ -> true)
216   db
217 ;;
218
219 (* Remove a single matching fact from a database, returning the new DB *)
220 let rec remove_fact1 db pred cnsts =
221   (* print_string ("Removing: " ^ (string_of_eval pred cnsts) ^ "\n"); *)
222   match db with
223   | [] -> []
224   | Fact(sign) :: tl when match_signature sign pred cnsts -> tl
225   | hd :: tl -> hd :: (remove_fact1 tl pred cnsts)
226 ;;
227
228 (* Return a new list of constraints specified by a parameter list *)
229 let cnst_of_params params env =
230   let param_to_cnst = function
231     | Tst.Lit(i) -> CEqlInt (i)
232     | Tst.Sym(s) -> CEqlSymbol (s)
233     | Tst.Var(i) -> List.nth env i
234     | Tst.Str(s) -> CEqlStr (s)
235     | Tst.Anon -> Any
236     | Tst.Arr(a) -> failwith "Arrays are not supported yet"
237   in
238   (* print_string (string_of_eval "cop-env" env);
239     print_string ("cop: " ^ (Printer.string_of_params (Tst.Params(params)))) ^ "\n"); *)
240   List.map param_to_cnst params
241 ;;

```

```

242
243 (* Convert a param list to a list of constraints *)
244 let sig_to_cnst signature =
245   let param_to_cnst = function
246     Tst.Lit(i) -> CEqlInt      (i)
247   | Tst.Sym(s) -> CEqlSymbol  (s)
248   | Tst.Var(i) -> Any
249   | Tst.Anon   -> Any
250   | Tst.Str(s) -> CEqlStr     (s)
251   | Tst.Arr(a) -> failwith "Arrays_are_not_supported_yet"
252   in
253     List.map param_to_cnst signature
254   ;;
255
256
257 (* Evaluate a query *)
258 let rec run_eval db addDB name vars =
259   let rec run_gen tail nextGen =
260     let sols = (nextGen ()) in
261     match sols with
262     | NoSolution -> eval_loop tail
263     | Solution (cnst, gen) ->
264       Solution(
265         (list_first (List.length vars) cnst),
266         (fun unit -> run_gen tail gen))
267   and eval_loop e =
268     match e with
269     | [] -> NoSolution
270     | Fact (signature) :: tail
271       when match_signature signature name vars ->
272         Solution (cnstAndAll vars signature.params,
273           (fun unit -> eval_loop tail))
274     | Rule (signature, exec) :: tail
275       when match_signature signature name vars ->
276         let matchedVars = cnstAndAll vars signature.params in
277         run_gen tail (fun unit -> exec db addDB matchedVars)
278     | head :: tail -> eval_loop tail
279   in
280     (*print_string ("In: " ^ (string_of_eval name vars));*)
281     eval_loop (!addDB @ !db)
282   ;;
283
284 (* Replace the i'th element with e in list
285  * Horribly wasteful, but oh well
286  *)
287 let rec list_replace i e list =
288   match list with
289   | [] -> []
290   | hd :: tl ->
291     if i == 0
292     then e :: tl

```

```

293         else hd :: (list_replace (i - 1) e tl)
294     ;;
295
296
297 (* The function that should be run when I type [i..j] *)
298 let rec range i j = if i >= j then [] else i :: (range (i+1) j)
299
300
301 let parseDB (prog) =
302     (*
303      *   All of the parse functions take the information regarding
304      *   their statement and return a function of the type
305      *   database -> database -> cnsts -> next
306      *
307      *   Which correspond to
308      *   primary db -> add-on db -> the scope's constraints
309      *
310      *   And they return a lazy solution iterator.
311      *
312      *   These same functions which are returned by the query method
313      *   are used internally to compose everything. Makes it
314      *   (relatively) easy to implement new functionality since nobody
315      *   needs to know anything about their parents or children except
316      *   that they conform to this interface.
317      *)
318
319
320
321 (* Our only compiler directive is print, for now.
322    learn/forget have a special syntax *)
323 let parseCompilerDirective name params =
324     let nc = String.compare name in
325     (* Print something... probably just "Hello World" *)
326     if (nc "print") = 0
327     then
328         fun db addDB cnst ->
329             let print_param param =
330                 match param with
331                 | Tst.Lit(i) -> print_int i
332                 | Tst.Str(s) -> print_string s
333                 | Tst.Sym(s) -> print_string s
334                 | Tst.Var(i) -> print_string (string_of_cnst (List.nth cnst i))
335                 | - -> ()
336             in
337             (List.iter print_param params;
338              print_string "\n";
339              NoSolution)
340     else
341         (print_string "Unknown_compiler_directive";
342          fun db addDB cnst ->
343              NoSolution)

```



```

344   in
345
346
347   (* Compute AND blocks by cANDing all the solutions in each row
348   *   of the cross product of all the possible solutions
349   *)
350   let rec parseAndBlock stmts =
351     match stmts with
352     [] -> (fun db addDB cnst -> Solution (cnst, fun unit -> NoSolution))
353     | stmt :: tail ->
354       let nextStatement = (parseAndBlock tail) in
355       let thisStatement = (parseStatement stmt) in
356       fun db addDB cnst ->
357         let nextGenMain = (nextStatement db addDB) in
358         let rec runThisGens thisGen =
359           match (thisGen ()) with
360           NoSolution -> NoSolution
361           | Solution (thisCnsts, thisGenNxt) ->
362             let rec runNextGens nextGen =
363               match (nextGen ()) with
364               NoSolution ->
365                 runThisGens thisGenNxt
366               | Solution (nextCnsts, nextGenNxt) ->
367                 Solution (nextCnsts, fun unit -> runNextGens nextGenNxt)
368             in
369             runNextGens (fun unit -> nextGenMain thisCnsts)
370         in
371         runThisGens (fun unit -> thisStatement db addDB cnst)
372
373
374   (* Return all the solutions from one, then go to the next *)
375   and parseOrBlock stmts =
376     match stmts with
377     [] -> (fun db addDB cnst -> NoSolution)
378     | stmt :: tail ->
379       let nextStmt = (parseOrBlock tail) in
380       let currStmt = (parseStatement stmt) in
381       fun db addDB cnst ->
382         let rec runOr nxt =
383           match nxt with
384           NoSolution -> nextStmt db addDB cnst
385           | Solution (vars, nxt) -> Solution (vars,
386             (fun unit -> runOr (nxt ())))
387         in
388         runOr (currStmt db addDB cnst)
389
390
391
392   (* Return the results from a query *)
393   and parseEval name params =
394     let param_var_index var_idx =

```

```

395     let rec pvi_iter plist idx =
396         match plist with
397         | [] -> -1
398         | Tst.Var(i) :: tl when i == var_idx -> idx
399         | _ :: tl -> pvi_iter tl (idx + 1)
400     in
401         pvi_iter params 0
402 in
403 fun db addDB cnst ->
404     let cnsts = cnst_of_params params cnst in
405     (* Map the slots returned from the eval into our slot-space *)
406     let revMap rCnsts =
407         List.map2
408         (fun cnst idx ->
409             let pIdx = param_var_index idx in
410             if pIdx == -1
411             then cnst
412             else cAnd cnst (List.nth rCnsts pIdx))
413         cnst
414         (range 0 (List.length cnst))
415     in
416     (* Run the eval, then send back the results, reverse mapping the slots as we go *)
417     let nxt = run_eval db addDB name cnsts in
418     let rec doNxt nxt =
419         match nxt with
420         | NoSolution -> NoSolution
421         | Solution(rCnsts, nxt) ->
422             (* print_string (string_of_eval name rCnsts); *)
423             let rCnsts = revMap (list_first (List.length params) rCnsts) in
424             (* print_string (string_of_eval name rCnsts); *)
425             Solution(rCnsts, (fun unit -> doNxt (nxt ())))
426     in
427     doNxt nxt
428
429
430 (* ***** BEHOLD — The bane of my existence!!!!!! *)
431
432 (* A dumber man could not have written this function...
433  * ... A smarter man would have known not to.
434  *)
435 and parseNotEval name params =
436     let eval = parseEval name params in
437     fun db addDB cnsts ->
438         (* It probably will help to think of this function as a binary blob...
439          * I blacked out while I was writing it, but I remember it having
440          * something to do with lazily-generated cross products. ~John
441          *)
442         let rec iter_outs bigList =
443             (* Printf.printf "%s\n" (string_of_eval "Level:" (List.hd bigList)); *)
444             match bigList with
445             | [] -> failwith "Internal_error_23"

```

```

446 | myRow :: [] ->
447   let rec linearGen myList =
448     match myList with
449       [] -> NoSolution
450       | hd :: tl -> Solution([hd], fun unit -> linearGen tl)
451   in
452     linearGen myRow
453 | myRow :: tl ->
454   let tlGenMain = iter_outs tl in
455   let rec twoGen myList nxtGen =
456     match myList with
457       [] -> NoSolution
458       | myHd :: myTl ->
459         match nxtGen with
460           NoSolution ->
461             twoGen myTl tlGenMain
462           | Solution(sol, nxtGen) ->
463             Solution(myHd :: sol, fun unit -> twoGen myList (nxtGen()))
464   in
465     twoGen myRow tlGenMain
466 in
467
468 (* Iterate through all the solutions, subtracting all the new solutions
469  * from the existing ones being stored in 'outs'
470  *)
471 let rec minus nxt outs =
472   match nxt with
473     NoSolution ->
474       iter_outs outs
475     | Solution(evCnsts, nxt) ->
476       minus
477         (nxt())
478         (* (list_acc (fun out -> List.map2 cMinus out evCnsts) outs) *)
479         (List.map2
480           (fun out evCnst ->
481             list_acc (fun o -> cMinus o evCnst) out)
482           outs
483           evCnsts)
484 in
485   (* Start with the input solution, and subtract all the results *)
486   minus (eval db addDB cnsts) (List.map (fun c -> [c]) cnsts)
487
488
489 (* Run an eval in somebody else's database *)
490 and parseDot2 v pred params =
491   let eval = parseEval pred params in
492   (fun db addDB cnst ->
493     match (List.nth cnst v) with
494       CEqlAgent(adb) ->
495         eval adb (ref []) cnst
496     | a -> (Printf.printf

```

```

497         "Warning: attempted_dot_( '.' )_on_a_non-agent: %s\n"
498         (string_of_cnst a);
499         NoSolution))
500
501 and parseNDot2 v pred params =
502   let eval = parseNotEval pred params in
503   (fun db addDB cnst ->
504     match (List.nth cnst v) with
505     | CEqlAgent(adb) ->
506       eval adb (ref []) cnst
507     | a -> (Printf.printf
508       "Warning: attempted_dot_( '.' )_on_a_non-agent: %s\n"
509       (string_of_cnst a);
510       NoSolution))
511
512
513 and doAnd myCnsts db addDB cnst =
514   let sol = cnstAndAll myCnsts cnst in
515   (* (print_string (string_of_eval "" myCnsts));
516     (print_string (string_of_eval "" cnst)); *)
517   if List.for_all (fun a -> a != FalseSol) sol
518   then Solution(sol, fun () -> NoSolution)
519   else NoSolution
520
521 and parseCompOp op v e2 =
522   let compOp i =
523     match op with
524     | Ast.Lt -> CLT(i)
525     | Ast.Gt -> CGT(i)
526     | Ast.Leq -> CLT(i + 1)
527     | Ast.Geq -> CGT(i - 1)
528     | Ast.Eq -> CEqlInt(i)
529     | _ -> failwith "Unsupported_comparison_operator"
530   in
531   match e2 with
532   | Tst.ELit(i) ->
533     doAnd ((list_fill Any v) @ [(compOp i)])
534
535 and parseStrComp v s =
536   doAnd ((list_fill Any v) @ [CEqlStr(s)])
537
538 and parseSymComp v s =
539   doAnd ((list_fill Any v) @ [CEqlSymbol(s)])
540
541 and parseLearnForget name statements =
542   let remove_facts db addDB cnsts =
543     let remove_fact (name, params) =
544       db := remove_fact_all !db name (cnst_of_params params cnsts)
545     in
546     List.iter remove_fact statements
547   in

```

```

548   let remove_fact1 db addDB cnsts =
549       let remove_fact (name,params) =
550           db := remove_fact1 !db name (cnst_of_params params cnsts)
551       in
552           List.iter remove_fact statements
553   in
554   let add_facts db addDB cnsts =
555       let add_fact (name,params) =
556           db := Fact({name = name; params = (cnst_of_params params cnsts)}) :: !db
557       in
558           List.iter add_fact statements
559   in
560   let nm = String.compare name in
561       if (nm "learn") == 0
562       then (fun db addDB cnsts -> add_facts db addDB cnsts; NoSolution)
563       else if (nm "forget") == 0
564       then (fun db addDB cnsts -> remove_facts db addDB cnsts; NoSolution)
565       else if (nm "forget1") == 0
566       then (fun db addDB cnsts -> remove_fact1 db addDB cnsts; NoSolution)
567       else failwith ("Invalid_directive:_" ^ name)
568
569   and parseDot1 v dname statements =
570       let study = parseLearnForget dname statements in
571       (fun db addDB cnst ->
572           match (List.nth cnst v) with
573           | CEqlAgent(adb) ->
574               study adb (ref []) cnst
575           | a -> (Printf.printf
576               "Warning:_attempted_@_dot_( '.' )_on_a_non-agent:_%s\n"
577               (string_of_cnst a);
578               NoSolution))
579
580   and parseStatement statement =
581       match statement with
582       | Tst.Block (redOp, statements)
583         when 0 == (String.compare redOp "AND") ->
584           parseAndBlock statements
585       | Tst.Block (redOp, statements)
586         when 0 == (String.compare redOp "OR") ->
587           parseOrBlock statements
588       | Tst.Block (redOp, statements) ->
589           (Printf.printf "Invalid_reduction_operator_%s\n" redOp;
590            (fun db addDB cnst -> NoSolution))
591       | Tst.Eval (name, params) ->
592           parseEval name params
593       | Tst.NEval (name, params) ->
594           parseNotEval name params
595       | Tst.Directive (name, params) ->
596           parseCompilerDirective name params
597       | Tst.Comp(e1, compOp, e2) ->
598           parseCompOp compOp e1 e2

```

```

599 | Tst.DirectiveStudy(name, statements) ->
600 |   parseLearnForget name statements
601 | Tst.StrComp(v, s) ->
602 |   parseStrComp v s
603 | Tst.SymComp(v, s) ->
604 |   parseSymComp v s
605 | Tst.Dot1(v, dname, statements) ->
606 |   parseDot1 v dname statements
607 | Tst.Dot2(v, pred, params) ->
608 |   parseDot2 v pred params
609 | Tst.NDot2(v, pred, params) ->
610 |   parseNDot2 v pred params
611 in
612
613 let parseRule stmt slots actions =
614   fun db addDB inCnsts ->
615     let rec runPer sols nxt =
616       match nxt with
617       | NoSolution -> NoSolution
618       | Solution(outCnsts, nxt) ->
619         (* Have we already given this solution? *)
620         if (List.mem outCnsts sols)
621         then runPer sols (nxt())
622         else
623           (List.iter
624            (fun action ->
625              (ignore (action db addDB outCnsts)))
626            actions;
627            Solution(outCnsts, fun () -> runPer (outCnsts :: sols)(nxt())))
628     in
629       (* print_string ("Num slots: " ^ (string_of_int slots) ^ "\n"); *)
630       runPer [] (stmt db addDB (cnst_extend_to inCnsts slots))
631 in
632
633 let parseRF = function
634   Tst.Rule (name, parms, numVars, statement, nseStmt) ->
635     Rule ({ name = name; params = (sig_to_cnst parms)};
636           (parseRule (parseStatement statement) numVars
637                     (List.map parseStatement nseStmt)))
638   | Tst.Fact (name, parms) ->
639     Fact ({ name = name; params = (sig_to_cnst parms)})
640 in
641
642 let tProg = Trans.translate(prog) in
643   ref (List.map parseRF tProg)
644 ;;
645
646 (* Primary entry point into the database. Specify the
647 * term to query and the number of variables to pass in.
648 *
649 * db is the database to query, and addDB is the "add on"

```

```

650  * database so the caller can pass information into the program
651  *)
652  let query db addDB pred numVars =
653    run_eval db addDB pred (list_fill Any numVars)
654  ;;
655
656
657  (* Print all the rules and facts in a DB- for debugging *)
658  let rec dump_db db =
659    let print_sig s =
660      Printf.printf "%s(%s)"
661        s.name
662        (String.concat "," (List.map string_of_cnst s.params))
663    in
664    let dump_rf rf =
665      match rf with
666      | Fact(s) ->
667          print_sig s;
668          print_string ";\n"
669      | Rule(s, f) ->
670          print_sig s;
671          print_string "⊆{}\n"
672    in
673    List.iter dump_rf db
674  ;;

```

Listing 56: C μ LOG Interpreter