Reinforcement Learning Language

Michael Groble (michael.groble@motorola.com)

December 15, 2006

1 Introduction

The Reinforcement Learning Language (RLL) is used to specify and simulate various reinforcement learning algorithms. Reinforcement learning is described in [SB98]. The language will initially focus on the application of Reinforcement Learning to finite Markov Decision Processes (MDPs). Briefly, a finite MDP is specified by the tuple $< \mathcal{S}, \mathcal{A}_s, \mathcal{P}^a_{ss'}, \mathcal{R}^a_{ss'} >$ where $\mathcal{S}$ is the finite state space, $\mathcal{A}_s$ is the finite action space (in general, the allowable actions depends on the current state $s \in \mathcal{S}$, $a \in \mathcal{A}_s$), $\mathcal{P}^a_{ss'}$ is the environment transition probabilities

$$\mathcal{P}^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

in other words, the probability that a certain state will be reached (s') as a result of taking a specific action (a) in a specific starting state (s) and finally $\mathcal{R}^a_{ss'}$ are the expected action rewards

$$\mathcal{R}^a_{ss'} = E[r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s']$$

in other words, the expectation of the reward (r_{t+1}) as a result of taking action (a) in a specific starting state (s) which results in a specific subsequent state (s').

The reinforcement learning problem is that of finding a policy π which in some sense optimizes our reward. Problems can be categorized into ones with finite ends (termed *episodic*) and those without end (*continuous*). The language will support the typical reward formulation, one of maximizing the expected discounted future rewards. In other words, the policy attempts to maximize the *return* R_t where

$$R_t = E\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}\right]$$

The term $0 < \gamma \leq 1$ is called a discount factor (it discounts future rewards). For continuous tasks, $T = \infty$ and the discount factor must be less than 1 to ensure the return is bounded. For episodic tasks, the termination of the episode corresponds to reaching a particular subset of the state space. These terminal

states are not considered part of $\mathcal{S}$. The set $\mathcal{S}^+$is used to denote the union of $\mathcal{S}$ and the terminal states. Technically in the definition of $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ above, $s \in \mathcal{S}$ and $s' \in \mathcal{S}^+$.

The purpose of the RLL is to specify finite MDP problems and the solution and simulation of both episodic and continuous problems using standard reinforcement learning algorithms. What was completed in the course of this project was specification of MDP problems and interpretation of them to generate a standard file format for MDP and POMDP solvers [Cas].

2 Tutorial

The tutorial walks through a few example problems. It starts with a simple deterministic decision problem. It then describes two MDP examples from [SB98] and one from [TBF05].

2.1 Deterministic Decision Problem

```
1   mdp Deterministic(goal=10)
2       states x {0:goal}
3       actions i {0:goal}
4       reward r
5
6       episode
7           isStart = x == 0
8           isTerminal = x' == goal
9
10      environment
11          x' = x + i
12          r = 1 if isTerminal else -1
```

The first line defines an MDP with name Deterministic and the parameter `goal` which has a default value of 10.

Lines 2 through 4 declare the states, actions and reward. In this case, they are all scalars. The states and actions are declared by providing an identifier and a bound on the range of values. In this case, both range from 0 to `goal` inclusive.

This MDP is episodic, meaning there is a special state condition that terminates the decision process. The predicates `isStart` and `isTerminal` are used to define start state and terminal state conditions. By default, they are defined so that `isStart` is always true and `isTerminal` is always false. the Lines 6 through 8 specify the episode by stating the only valid start state is 0 and the only valid terminal state is `goal`.

Lines 10 through 12 define the environment update. The environment section can be thought of as a function that is given the initial state and an action and must compute the updated state and reward. The tick mark is used to

reference the updated state (which is why line 8 references `x'` for the terminal predicate). The last line also shows a Python-like conditional statement of the form $<$ `true` $-$ `value` $>$ `if` $<$ `condition` $>$ `else` $<$ `false` $-$ `value` $>$.

With this definition, the MDP is defining a game in which the purpose is to reach `goal` as quickly as you can. Your action is incremented deterministically to your current state. A policy of providing `goal` as your first action results in an accumulated (non-discounted) return of 1 for the whole "game", while a policy of providing 1 as an action 10 times in a row would result in a return of -8 (when `goal` is the default value of 10).

The utility `generate-pomdp` is run by providing a file to read (which would contain definitions of MDPs), the name of an MDP to run and it's arguments. The following is an example of running Deterministic with a goal of 3.

```
$generate-pomdp deterministic.rll Deterministic 3
############
discount: 1.0
values: reward
# states
# 1: x (4 values)
states: 4
# actions
# 1: i (4 values)
actions: 4
start:
1 0 0 0
T: 0 : 0 : 0     1.0000000   # a= 0  s= 0  s'= 0
R: 0 : 0 : 0    -1.0000000   # a= 0  s= 0  s'= 0
T: 1 : 0 : 1     1.0000000   # a= 1  s= 0  s'= 1
R: 1 : 0 : 1    -1.0000000   # a= 1  s= 0  s'= 1
T: 2 : 0 : 2     1.0000000   # a= 2  s= 0  s'= 2
R: 2 : 0 : 2    -1.0000000   # a= 2  s= 0  s'= 2
T: 3 : 0 : 3     1.0000000   # a= 3  s= 0  s'= 3
R: 3 : 0 : 3     1.0000000   # a= 3  s= 0  s'= 3
T: 0 : 1 : 1     1.0000000   # a= 0  s= 1  s'= 1
R: 0 : 1 : 1    -1.0000000   # a= 0  s= 1  s'= 1
T: 1 : 1 : 2     1.0000000   # a= 1  s= 1  s'= 2
R: 1 : 1 : 2    -1.0000000   # a= 1  s= 1  s'= 2
T: 2 : 1 : 3     1.0000000   # a= 2  s= 1  s'= 3
R: 2 : 1 : 3     1.0000000   # a= 2  s= 1  s'= 3
T: 0 : 2 : 2     1.0000000   # a= 0  s= 2  s'= 2
R: 0 : 2 : 2    -1.0000000   # a= 0  s= 2  s'= 2
T: 1 : 2 : 3     1.0000000   # a= 1  s= 2  s'= 3
R: 1 : 2 : 3     1.0000000   # a= 1  s= 2  s'= 3
T: 0 : 3 : 3     1.0000000   # a= 0  s= 3  s'= 3
R: 0 : 3 : 3     1.0000000   # a= 0  s= 3  s'= 3
```

See [Cas] for details of this file format, but note the T entries are transition probabilities $\mathcal{P}^a_{ss'}$ and R entries are expected rewards $\mathcal{R}^a_{ss'}$. The comments at the end of each line translate state and action index values to values of action, start state and end state. In this case, the values and indices are the same.

2.2 Gambler's Problem

The is a simple gambling game where the probability of winning any particular turn is a random coin flip. The objective is to acquire a total amount matching a set goal. On each turn, the gamble must bet at least 1 and at most the total holdings (up to the amount that would give them the goal if they were to win). The learning problem is to determine the optimal gambling policy, in other words what amount should be bet for each amount the gambler is holding.

```
1   mdp Gamblers(maxCapital = 100, pSuccess = 0.5)
2       states capital {0:maxCapital}
3       actions stake {1:maxCapital}
4       reward  r
5
6       validAction = 0 < stake && \
7                     stake <= min(capital, maxCapital - capital)
8
9       episode
10          isStart = 0 < capital && capital < maxCapital
11          isTerminal = capital' == 0 || capital' == maxCapital
12
13      environment
14          random win = binomial(pSuccess)
15
16          capital' = capital + stake * (1 if win else -1)
17          r = 1 if capital' == maxCapital else 0
```

There are two more predefined predicates in MDP specifications. They are `validState` denoting $\mathcal{S}$ and `validAction` denoting $\mathcal{A}_s$. This is required since the simple bounds cannot capture all such constraints. Line 6 shows an example of valid actions, specifying essentially that you can't bet more than you have and your ultimate winnings have a cap. Lines 9 through 11 define the episodic predicates. In this case the terminal states are 0 (bust) or the max winning and the start states are all but those terminals.

Line 14 defines a random variable within the scope of the environment. In this particular case, the random variable is drawn from a binomial distribution. Since it is defined in the scope of the environment, a new value is drawn at each turn. Line 16 shows how capital is updated and line 17 shows how the reward is computed.

Below shows the output for the case where the limit is 5 and the probability favors the house.

```
$generate-pomdp gamblers.rll Gamblers 5 0.4
############
discount: 1.0
values: reward
# states
# 1: capital (6 values)
states: 6
# actions
# 1: stake (5 values)
actions: 5
start:
0 0.25 0.25 0.25 0.25 0
T: 0 : 1 : 0     0.6000000   # a= 1  s= 1  s'= 0
T: 0 : 1 : 2     0.4000000   # a= 1  s= 1  s'= 2
T: 0 : 2 : 1     0.6000000   # a= 1  s= 2  s'= 1
T: 0 : 2 : 3     0.4000000   # a= 1  s= 2  s'= 3
T: 1 : 2 : 0     0.6000000   # a= 2  s= 2  s'= 0
T: 1 : 2 : 4     0.4000000   # a= 2  s= 2  s'= 4
T: 0 : 3 : 2     0.6000000   # a= 1  s= 3  s'= 2
T: 0 : 3 : 4     0.4000000   # a= 1  s= 3  s'= 4
T: 1 : 3 : 1     0.6000000   # a= 2  s= 3  s'= 1
T: 1 : 3 : 5     0.4000000   # a= 2  s= 3  s'= 5
R: 1 : 3 : 5     1.0000000   # a= 2  s= 3  s'= 5
T: 0 : 4 : 3     0.6000000   # a= 1  s= 4  s'= 3
T: 0 : 4 : 5     0.4000000   # a= 1  s= 4  s'= 5
R: 0 : 4 : 5     1.0000000   # a= 1  s= 4  s'= 5
```

2.3 Jack's Car Rental

This next example is a car rental company with two sites. People stop at each site to request rental cars and return cars. At night, the owner has the discretion of transferring up to 5 cars from one site to another. In this case the policy we want to learn is how many cars to transfer given the number of cars remaining at each location. For simplicity, cars at each location are limited to 20. Any transferred or returned over 20 effectively disappear. Each rental provides a reward of +10 and each car transfer incurs a cost of 2 (provides a reward of -2).

```
1   mdp CarRental(maxCars = 20, maxTransfer = 5)
2       states cars[2] {0:maxCars}
3       actions netTransfer {-maxTransfer:maxTransfer}
4       reward  r
5
6       validAction = -min(5,cars[1]) < netTransfer && \
7                     netTransfer < min(5,cars[0])
8
9       environment
```

```
10          random returns  = [poisson(3);poisson(2)]
11          random requests = [poisson(3);poisson(4)]
12
13          nextMorn = min( cars + [-netTransfer;netTransfer], 20)
14          rentals = min(requests, nextMorn)
15          cars' = min(nextMorn + returns - rentals, 20)
16          r = rentals * 10 - 2 * abs(netTransfer)
```

The first thing to notice about this case is that the state is no longer a scalar, but is a vector. The notation for vector quantities and operations match that of Matlab in spirit. Line 2 declares the state element cars as a vector with two elements (one for each rental location). Line 3 defines the action as a net transfer, the number of cars transferred from location 1 to location 2 (a negative value indicates transfer from 2 to 1). RLL is currently limited to explicit dimensions. In other words, it is not possible to have a vector dimension depend on a parameter.

The next thing to notice about this case is that it is continuous, not episodic, so there is no episodic section.

Lines 10, 11 and 13 all show the notation for array literals, square brackets with elements separated by semicolons. In this case, the random variables declared in lines 10 and 11 are both random vectors with two elements each.

Listed below is an initial portion of the POMDP file with a max of 5 cars at each location and a maximum transfer of 2 (the full file is almost 4000 lines long).

```
$generate-pomdp carrental.rll CarRental 5 2
############
discount: 0.9
values: reward
# states
# 1: cars[0] (6 values)
# 2: cars[1] (6 values)
states: 36
# actions
# 1: netTransfer (5 values)
actions: 5
start:
uniform
T: 3 :  2 :  0     0.0062815   # a=  1  s= 2 0  s'= 0 0
T: 3 :  2 :  1     0.0191738   # a=  1  s= 2 0  s'= 1 0
T: 3 :  2 :  2     0.0292546   # a=  1  s= 2 0  s'= 2 0
T: 3 :  2 :  3     0.0297484   # a=  1  s= 2 0  s'= 3 0
T: 3 :  2 :  4     0.0226817   # a=  1  s= 2 0  s'= 4 0
T: 3 :  2 :  5     0.0138312   # a=  1  s= 2 0  s'= 5 0
T: 3 :  2 :  6     0.0126803   # a=  1  s= 2 0  s'= 0 1
T: 3 :  2 :  7     0.0387055   # a=  1  s= 2 0  s'= 1 1
```

```
T: 3 :  2 :  8     0.0590551   # a=  1  s= 2 0  s'= 2 1
T: 3 :  2 :  9     0.0600520   # a=  1  s= 2 0  s'= 3 1
T: 3 :  2 : 10     0.0457867   # a=  1  s= 2 0  s'= 4 1
T: 3 :  2 : 11     0.0279206   # a=  1  s= 2 0  s'= 5 1
T: 3 :  2 : 12     0.0127975   # a=  1  s= 2 0  s'= 0 2
T: 3 :  2 : 13     0.0390633   # a=  1  s= 2 0  s'= 1 2
...
```

The most important thing to note about this output is that now with the non-scalar state vector, the comments are more helpful. They identify the state vector values, rather than the index required by the POMDP file format.

2.4 Heaven or Hell

This is an example of a "gridworld", an environment where the agent's actions allow it to move about a 2-dimensional grid. In this case, there is one start position, two terminal positions (Heaven and Hell) and one Map position. The reward for Heaven is +100 while the reward for Hell is -100. The act of moving has a reward of -1. Two locations in the world are fixed for Heaven and Hell, but the agent cannot tell which is which. Each episode has Heaven and Hell randomly assigned to the two locations. If the agent enters the Map location, it is told which location is Heaven. Also, the action is not always carried out as desired. With some probability, the real direction is chosen randomly from the 3 directions other than the desired one.

```
1   mdp HH(pA = 0.5,pMove=1.0)
2       diagram grid
3           #012345678
4           " A     B " #0
5           " ........ " #1
6           "     .    " #2
7           "     .    " #3
8           "     S    " #4
9           "     .    " #5
10          "     .    " #6
11          "     ...X " #7
12
13      states
14          position[2] grid
15          map {unknown, a, b}
16      actions direction {n,s,e,w}
17      reward r
18
19      validState = position in grid(".SX")
20
21      episode
```

```
22          isStart = position in grid("S") && map == unknown
23          isTerminal = position' in grid("AB")
24          random aIsHeaven = binomial(pA)
25
26      environment
27          random moveAsIntended = binomial(pMove)
28          random offset = uniform(1,3)
29
30          d = direction if moveAsIntended else (direction + offset)
31          dir = d if d < 4 else d-4
32          next = [position[0] + (-1 if dir == w else
33                                 (1 if dir == e else 0));
34                  position[1] + (-1 if dir == n else
35                                 (1 if dir == s else 0))]
36          position' = next if next in grid(".SXAB") else position
37          map' = map if position' not in grid("X") else \
38                 (a if aIsHeaven else b)
39          inHeaven = position' in grid("A") && aIsHeaven || \
40                     position' in grid("B") && !aIsHeaven
41          inHell   = position' in grid("B") && aIsHeaven || \
42                     position' in grid("A") && !aIsHeaven
43          r = 100 if inHeaven else (-100 if inHell else -1)
```

This example shows quite a few new aspects of the language. First is that states can be structured. In this case states are represented by a vector component (position) and a scalar component (map). Second is that states and actions can consist of sets of nominal named values rather than integer values. The map state can have the values "unknown" (meaning the location of Heaven is unknown), "a" or "b". Similarly, the action in line 16 is movement in one of the compass directions "n", "s", "e" or "w".

Third, shown in lines 2 through 11, is a shorthand notation to ease creating gridworlds by depicting them with ASCII diagrams. The diagram represents a set of two dimensional locations which can be indexed by strings. The position [0;0] is the upper left corner of the diagram with the first index incrementing to rightward and the second index incrementing downward (the index values are noted in comments). Line 19 defines a valid state predicate using a set membership notation. In this case, valid states are the 16 positions within the diagram that contain any of the characters ".", "S" or "X".

Lines 27 through 31 allow movement to be non-deterministic.

There is no predefined behavior for detecting movement outside of the valid states. Lines 32 through 36 describe how the position is updated to stay within the valid region.

Lines 21 through 24 show an episode section. In this case, a random variable is declared in the episode section, it is therefore drawn at the beginning of each episode and remains constant during the turns within an episode.

Below is some sample output for running this with a slight preference for the "A" location being Heaven and a slight chance of moving in a random direction other than the one intended. The start section has been truncated since there are 300 states in the actual output.

```
$generate-pomdp hh.rll HH 0.6 0.9
############
discount: 1.0
values: reward
# states
# 1: position[0] (10 values)
# 2: position[1] (10 values)
# 3: map (3 values)
states: 300
# actions
# 1: direction (4 values)
actions: 4
start:
0 0 0 0 0 0 0 ...
T: 0 :  11 :   1     0.9000000   # a= 0  s= 1 1 0  s'= 1 0 0
R: 0 :  11 :   1    20.0000000   # a= 0  s= 1 1 0  s'= 1 0 0
T: 0 :  11 :  11     0.0666667   # a= 0  s= 1 1 0  s'= 1 1 0
R: 0 :  11 :  11    -1.0000000   # a= 0  s= 1 1 0  s'= 1 1 0
T: 0 :  11 :  12     0.0333333   # a= 0  s= 1 1 0  s'= 2 1 0
R: 0 :  11 :  12    -1.0000000   # a= 0  s= 1 1 0  s'= 2 1 0
T: 1 :  11 :   1     0.0333333   # a= 1  s= 1 1 0  s'= 1 0 0
R: 1 :  11 :   1    20.0000000   # a= 1  s= 1 1 0  s'= 1 0 0
T: 1 :  11 :  11     0.9333333   # a= 1  s= 1 1 0  s'= 1 1 0
R: 1 :  11 :  11    -1.0000000   # a= 1  s= 1 1 0  s'= 1 1 0
T: 1 :  11 :  12     0.0333333   # a= 1  s= 1 1 0  s'= 2 1 0
R: 1 :  11 :  12    -1.0000000   # a= 1  s= 1 1 0  s'= 2 1 0
T: 2 :  11 :   1     0.0333333   # a= 2  s= 1 1 0  s'= 1 0 0
R: 2 :  11 :   1    20.0000000   # a= 2  s= 1 1 0  s'= 1 0 0
T: 2 :  11 :  11     0.0666667   # a= 2  s= 1 1 0  s'= 1 1 0
R: 2 :  11 :  11    -1.0000000   # a= 2  s= 1 1 0  s'= 1 1 0
T: 2 :  11 :  12     0.9000000   # a= 2  s= 1 1 0  s'= 2 1 0
R: 2 :  11 :  12    -1.0000000   # a= 2  s= 1 1 0  s'= 2 1 0
T: 3 :  11 :   1     0.0333333   # a= 3  s= 1 1 0  s'= 1 0 0
R: 3 :  11 :   1    20.0000000   # a= 3  s= 1 1 0  s'= 1 0 0
...
```

Again, the comments help identify the states, but only the values are currently shown, not the names (e.g. n, s, e, w).

3 Language Manual

For reference, the full formal definition of the lexical and syntactic elements is listed in the appendix.

3.1 Lexical conventions

The rll language has lexical elements for whitespace, identifiers, keywords, literals, operators and delimiters.

3.1.1 Whitespace

Similar to Python [vR], there are some cases where whitespaces are significant in rll specifications. In particular, indentation is used to delineate blocks of statements. To handle this concept, the language uses the tokens `NEWLINE`, `INDENT` and `DEDENT`.

There is also a distinction between physical source lines and logical source lines. A backslash `\` acts as a logical line continuation character. A physical line ending in a backslash will be joined with the next physical line to represent a single logical line. Logical line continuations also exist for content between the delimiter pairs `()`, `{}` and `[]`. The following code example is treated as a single logical line.

```
a = [1;
     2]
```

The `NEWLINE` token represents the end of a logical line. The example above therefore consists of the following tokens `a = [ 1 ; 2 ] NEWLINE`.

3.1.2 Comments

Comments start with a `#` character and continue to the end of the physical line.

3.1.3 Identifiers

Identifiers are sequences of letters (both upper and lower case), numbers and the underscore character `_`. The first character of an identifier must be a letter. Identifiers are case sensitive.

3.1.4 Keywords

The following are keywords in the rll language.

```
actions diagram else elsif environment episode for if in
mdp not random reward set states valid while
```

3.1.5 Literals

There are three types of literals in rll, integer numbers, floating point numbers and strings. Integers are sequences of digits. Floating point numbers are defined as in C. String literals are delimited by double quotes, for example "a string".

3.1.6 Operators

The following tokens are used as operators.

```
+ - * / = += -= *= /= < > == != <= >= && || !  '
```

3.1.7 Delimiters

The following tokens are used as delimiters.

```
( ) [ ] { } < > :  , ;
```

3.2 Types

integer an integer value

floating point a floating point value

string a string

array a fixed dimension array

set a set

diagram a named rectangular character grid which acts as a set of two dimensional arrays

function a function

state a named discrete scalar or single-dimension array

action a named discrete scalar or single-dimension array

reward a named floating point scalar

mdp a markov decision process

3.3 Expressions

3.3.1 Assignable Expressions

Assignable expressions are those expressions that may exist on the left hand side of an assignment (also known as "lvalues"). In rll, atomic assignable expressions consist of an identifier, optionally followed by the ' operator, optionally followed by an array index.

The ' operator is valid only for state types and is used to denote the updated value of the state.

An array index is an integer surrounded by square brackets. The following is an example using both.

```
s'[1] = s[1] + 2
```

Function calls may return multiple values. The individual return values are separated by commas and the entire group is enclosed in angle brackets, for example.

```
<s'[0],returns> = f()
```

3.3.2 Primary Expressions

Primary expressions consist of atomic assignable expressions, literals, parenthesized expressions, function calls, anonymous sets, anonymous arrays and intervals.

Function calls consist of an identifier followed by a possibly empty list of arguments surrounded by parentheses. The argument list can consist of both positional and named arguments, all separated by commas. Positional arguments may be any expression while a named argument is an identifier followed by `=` followed by an expression. For example.

```
sarsa(Racetrack, 0.6, episodes = 1000)
```

Anonymous sets are a comma delimited list of expressions surrounded by curly braces, for example.

```
{1,10,a}
```

Anonymous arrays consist of semicolon delimited rows surrounded by square brackets. Each row is a comma separated list of expressions. Each row must have the same number of elements.

```
[ a, 1; 0, b]
```

Intervals consist of two numbers separated by a colon. The following set has an interval as its first member.

```
{0:10,15}
```

3.3.3 Unary Expressions

Unary expressions are primary expressions prefixed by one of the operators `!` `+` `-` denoting logical negation, unary positive and unary negative respectively.

3.3.4 Multiplicative Expression

A multiplicative expression consists of a sole unary expression or two unary expressions separated by one of the binary operators `*` `/` denoting multiplication and division respectively.

3.3.5 Additive Expression

An additive expression consists of a sole multiplicative expression or two multiplicative expressions separated by one of the binary operators `+ -` denoting addition and subtraction respectively.

3.3.6 Boolean Relation Expression

A boolean relation expression consists of a sole additive expression or two additive expressions separated by one of the binary operators `> < >= <=` or the keywords `in` or `not in`. These represent the logical relations greater than, less than, greater than or equal, less than or equal, in (set membership) and not in (set exclusion) respectively.

3.3.7 Boolean Equality Expression

A boolean equality expression consists of a sole boolean relation expression or two boolean relation expressions separated by one of the binary operators `==` `!=` representing equals to or not equals to respectively.

3.3.8 Boolean And Expression

A boolean and expression consists of a sole boolean equality expression or two boolean equality expressions separated by the binary operator `&&` representing logical and.

3.3.9 Boolean Or Expression

A boolean or expression consists of a sole boolean and expression or two boolean and expressions separated by the binary operator `||` representing logical or.

3.3.10 Expression

An expression consists of a sole boolean or expression or a ternary expression of the form

booleanOrExpression `if` *booleanOrExpression* `else` *booleanOrExpression*

The middle expression is the test condition. The first expression is the value of the expression if the test is true, the last expression is the value if the test is false.

3.4 Statements

At the top level, there are two types of statements in rll, one type for defining markov decision processes and one type for defining procedural statements.

3.4.1 MDP Definition

An MDP definition consists of a line of the form

`mdp` *identifier* `(` *parameter list* `)`

followed by an indented block of statements which consist of state declaration, action declaration, reward declaration, diagram declaration, constraint declaration, episode declaration or environment declaration.

Parameter list is a comma separated list of named parameters with default values, for example.

```
mdp Gamblers(maxCapital = 100, pSuccess = 0.5)
```

3.4.2 State Declaration

A state declaration consists of the keyword `states` followed by either a single variable declaration or an indented block of variable declaration statements. States are treated as integer values.

```
states
    position[2] {0:10}
    velocity[2] {-2:2}
```

The declaration introduces identifiers for the states and also introduces identifiers for the final states denoted with tick marks, e.g. `position'` and `velocity'` in the example above.

3.4.3 Action Declaration

An action declaration consists of the keyword `actions` followed by either a single variable declaration or an indented block of variable declaration statements. Actions are treated as integer values.

3.4.4 Variable Declaration

A variable declaration consists of an identifier followed by an optional dimension (an integer surrounded by square brackets) and a set expression which describes the range the variable is allowed to have. There is no notation to provide separate ranges for the different elements of a vector. In the example below, both elements of the position vector have the same range, 0 to 10 inclusive.

```
position[2] {0:10}
```

Within the range statement, identifiers can be introduced for named values. The example

```
direction {n,s,e,w}
```

Creates four new identifiers "n", "s", "e", "w", and gives them the values 0 through 3. It is an error to mix already defined variables and undeclared names in this way. All elements must be undefined (in which case they are created as described), or all must be previously defined.

3.4.5 Reward Declaration

A reward declaration consists of the keyword `reward` followed by an identifier. Only a single reward declaration can exist in an MDP definition. Rewards have floating point type.

3.4.6 Diagram Declaration

A diagram declaration is used to create a graphical representation of a 2 dimensional gridworld. A diagram declaration consists of the keyword diagram on a line followed by an identifier followed by an indented block of string literals. For example

```
diagram grid
    " A     B "
    " ....... "
    "    .    "
    "    .    "
    "    S    "
    "    .    "
    "    .    "
    "    ...X "
```

This declaration introduces a function named "grid" (in this example) which takes a string argument and returns a type that is considered a set of vectors. The set consists of the positions of each of the characters passed in string argument. Position [0;0] is the top left corner of the diagram. The first index increments rightward and the second increments downward. The expression `grid("ABX")` is therefore the set

```
{[1;0],[7;0],[7;7]}
```

A diagram declaration is an optional element of an MDP definition.

3.4.7 Constraint Declaration

A constraint declaration is used to define predicates and consists of an identifier followed by an equal sign followed by a boolean expression. A constraint currently must be one of the predefined identifiers `isStart`, `isTerminal`, `validState` or `validAction`.

3.4.8 Episode Declaration

An episode declaration is used to define episodic nature of the MDP. It consists of the keyword `episode` on a line followed by an indented block of statements consisting of constraint declarations or random declarations. An episode declaration is an optional element of an MDP definition.

3.4.9 Environment Declaration

An environment declaration is used to define the state and reward update functions of the MDP. It consists of the keyword `environment` on a line followed by an indented block of statements consisting of random declarations and assignment statements. An MDP definition must have one environment declaration. The assignment statements in the environment declaration must assign values to the reward identifier and must assign values to the next state value for each of the state variables.

3.4.10 Random Declaration

A random declaration declares a random variable with a particular distribution and is of the form `random` *identifier* `=` *expression* `NEWLINE`.

3.4.11 Assignment Statement

An assignment statement is of the form *assignableExpression* op *expression* `NEWLINE` where op may be any of the assignment operators `= += -= *= /=`. Assignement statements can introduce new variables. The type of the variable is inferred from the right hand expression. Within an MDP, initial states and actions are not assignable, only final states and rewards are.

3.4.12 Function Call Statement

A function call statement is simply a function call expression followed by a `NEWLINE`.

3.4.13 For Statement

A for statement consists of a line `for (` *iteratorExpression* `)` `NEWLINE` followed by an indented block of procedural statements. An iterator expression is an expression of the form *identifier* `=` *expression* `:` *expression*, or of the form *identifier* `in` *set*. In the first case, the iterator takes on the values in the range from the first expression to the second. In the second case, the iterator takes on each of the values in the set. The indented block of statements is executed once for each iterator value.

3.4.14 While Statement

A while statement consists of the line `while (` *expression* `)` `NEWLINE` followed by an indented block of procedural statements. The block is executed while the expression evaluates to true.

3.4.15 If Statement

An if statement consists of three different types of chunks, an if chunk followed by an arbitrary number of elsif chunks followed by an optional else chunk. An if

chunk is of the form `if ( expression ) NEWLINE` followed by an indented block of procedural statements. An elsif chunk is of the form `elsif ( expression ) NEWLINE` followed by an indented block of procedural statements. An else chunk is of the form `else NEWLINE` followed by an indented block of procedural statements.

3.5 Built-in Functions

There are a number of pre-defined functions available in both MPD definitions and procedural statements. Mathematical functions are available in both and consist of

`max(a,b)` maximum value of `a` and `b`

`min(a,b)` minimum value of `a` and `b`

`abs(a)` absolute value of `a`

The mathematical operators work on both scalar and array types. On arrays, the operations are performed element-wise.

Discrete random variable distributions are available in MPD definitions and consist of

`uniform(n,m)` uniform distribution from `n` to `m`

`binomial(p)` binomial distribution with probability of success `p`

`poisson(n)` poisson distribution with expectation `n`

4 Project Plan

4.1 Process

Since this project was done by just myself, I did not require formal process definition and planning to make sure dependencies were met in time to complete the project. The primary process focus was on iterative and test driven development. By the time the whitepaper was submitted, I had developed the set of input files that served as the base test cases throughout the project. These served as the primary test cases through the whole project with different levels of fidelity (parsing, abstract syntax tree generation, code generation, run-time interpretation). Other tests were added as needed to prove out specifics of the type implementation, tree walker and interpretation engine.

4.2 Plan

I did not define dates for plan milestones, but had the following ordering of planned activities.

- Define lexer and parser
- Define AST node structure
- Implement tree walker and static semantic error checking
- Implement run-time interpretation engine
- Generate probability tables for deterministic decision problem
- Implement random distributions and validate computed probabilities for non-deterministic cases
- Implement simulation interpretation functionality
- Integrate interpreter with existing reinforcement learning code
- Validate learned policies against previously computed solutions
- Write final report

4.3 Environment

The following tools were used as part of the environment.

ANTLR 2.7.6 Parser generator language/code

g++ 4.1.1 C++ compiler

Boost 1.33.1 C++ libraries

Loki 0.1.5 C++ libraries

CppUnit 1.10.2 C++ unit test framework

Eclipse 3.2.0 IDE (including the CDT plugin for C++ support and the ANTLR Eclipse plugin for ANTLR support)

Highlight 2.4.8 Source code to LaTeX translator

In addition, parts of the parser use some code from the Python lexer created by Terence Parr and Loring Craymer. The files rll.g, RllTokenStream.hpp and RllTokenStream.cpp contain this code, which is clearly delimited. LineCountingAST was adapted for C++ from code that comes with ANTLR. Most of the code in util.hpp existed prior to this project. All other code was developed solely by myself during the course of this project.

4.4 Programming Style

The programming style is shown by the small examples below. First a class declaration.

```
class IntervalSet
{
public:
    typedef int Index;
    typedef int Value;
    typedef boost::numeric::interval<Value> Interval;
    typedef IntervalSetConstIterator<Value> const_iterator;

    IntervalSet();
    void insert(Value const& v);
    void insert(Interval const& interval);
    void insert(std::pair<Value,Value> const& interval);

    int cardinality() const;
    Value valueOfIndex(const Index& index) const;
    Index indexOfValue(const Value& value) const;
    bool contains(Value const& value) const;

    const_iterator begin() const;
    const_iterator end() const;

private:
    typedef std::list<Interval> IntervalList;
    IntervalList _intervals;
    int _cardinality;

    void IntervalSet::updateCardinality();
};
```

In general, typedefs are required for any container used in method arguments or members. Methods need to strive for const correctness. Template definitions and method arguments follow the recommendations in [VJ03]. Namespaces are always fully qualified in header files, "using" declarations are not used (this is also mostly true in the source code as well).

Then an implementation example.

```
TernaryType::TernaryType(Scope& parent)
  : _parent(parent)
  , _trueScope(parent.makeChildScope())
  , _falseScope(parent.makeChildScope())
  , _return()
  , _condition(new BooleanType())
```

```
{
}

bool
TernaryType::setReturn(RllType::Ptr type)
{
    bool result = true;

    if (!_return) {
        _return = type;
    }
    else {
        // return can't be redefined
        result = false;
    }

    return result;
}
```

Constructors list elements one per line, method types exist by themselves on a line before the method name.

4.5 Project Log

The project did not complete the plan. Some elements were not as "linear" as planned and some were not completed. In particular, the following plan steps were performed at same time iteratively.

- Implement tree walker and static semantic error checking
- Implement run-time interpretation engine
- Generate probability tables for deterministic decision problem
- Implement random distributions and validate computed probabilities for non-deterministic cases

My procedure for working through this process was to implement a error reporting mechanism for static semantic errors. I then defined special TODO errors for cases that I hadn't implemented yet. As I was filling in the implementation of the tree walker, I sprinkled TODO errors for every branch I had not implemented yet. I would turn the TODO errors off with a switch and then run my automated tests until they ran clean (didn't report any true errors). Then I turned the TODO errors back on and saw what was failing next. There was so much interaction between the type system, instruction implementation and tree walker, that I couldn't cleanly focus on one aspect at a time (e.g. deterministic vs. non-deterministic behavior).

Also, the following steps were not completed, although the first is trivial given the existing implementation that correctly executes all of the MDP code to generate the POMDP files.

- Implement simulation interpretation functionality
- Integrate interpreter with existing reinforcement learning code
- Validate learned policies against previously computed solutions

5 Architectural Design

The main procedural components are the lexer, parser and tree walker and they work in a pipeline. I chose a register-based architecture for the interpreter. The main components in the interpreter are RllType, Instruction and Scope.

5.1 RllType

RllType is the base type for all types supported by the system. The relevant interfaces are briefly described below.

```
RllType(bool isConstant=false);
bool isConstant() const;
```

The base type tracks whether it is a compile time constant. While not implemented yet, the mechanism is in place to allow optimizations like constant propagation.

```
void setName(std::string const& name);
std::string const& name() const;
```

All types have names. By default, each instance is given a unique name so it can be tracked when logging instructions. When a type is added to a symbol table, it's name is updated to reflect what was provided in the symbol table for it.

```
virtual Ptr newSameType() const;
```

One desired feature of MDP specifications was that they would be as functional as possible. In particular, new variables can be created without explicit type specifications, the types are inferred from the relevant portions of the AST. This method allows a simple way to create a variable which is of the same dynamic type as an existing variable.

5.2 Instruction

Instruction is the base type for all instructions that can be executed by the runtime. The only significant interface it has is to execute itself.

```
virtual void operator()() const = 0;
```

The interesting code for instructions come from the subclasses. Instructions store the variables they operate on as shared pointers. When instructions are optimized away, the corresponding temporary variables will be deleted from the heap by the shared pointer memory management. The body of the operator instruction is implemented in terms of the concrete types for speed considerations. An add instruction, for example, knows it is adding an IntegerType with a DoubleType and putting the result in a DoubleType. The interpreted instruction operates directly on the base C++ types, e.g. as if the code were as follows.

```
void add(int& lhs, double& rhs, double& out) {out = lhs + rhs;}
```

See Instruction.hpp for more details.

5.3 Scope

Scope is responsible for maintaining a symbol table and storing instructions. The relevant interfaces are briefly described below.

```
virtual Scope makeChildScope(std::string const& name = "");
```

Scopes are organized in a hierarchical tree. The name of the scope helps determine error locations (a typical use is to set it to the name of the file being parsed).

```
virtual bool addSymbol(std::string const& name,
                       TypePtr symbol,
                       bool okToEclipse = false);
virtual TypePtr getSymbol(std::string const& s);
```

When adding symbols, the relevant thing to note is that the interface allows a name to eclipse a definition made at a higher scope. This was used primarily for parameters of function and MDP definitions.

```
virtual void addInstruction(InstructionPtr instruction);
virtual void evaluateInstructions() const;
```

The interfaces for adding and evaluating instructions are straighforward.

```
virtual void reportError(std::string const& implementationFile,
                         int implementationLine,
                         std::string const& message,
                         int sourceLine);
```

The tree walker made a best effort to identify all semantic errors, not stop on the first one found. The scope was used to report errors so they could be gathered and reported consistently.

6 Test Plan

6.1 Automated Tests

The following lists the automated tests used for development.

Test	Purpose
testMulti	Test the basic multi-dispatch mechanism to ensure that instructions would get created correctly.
testSymmetricMulti	Test multi-dispatch for swapping symmetric arguments.
testBasicMath	Test instruction execution for basic math.
testArray	Test instruction execution for array math.
testInterval	Test instruction execution for intervals.
testFunctionCall	Test instruction execution for function calls.
testTernary	Test instruction execution for ternary operations.
testDiagram	Test instruction execution for diagram creation and set membershipt tests.
testDistribution	Test instruction execution for creation of random distributions and drawing random variables.
testIntervals	Test interval type implementation
testElementIterate	Test iterating over the elements in an set.
testVectorIterate	Test iterating over the elements of a vector.
testUniform	Test the probabilities and expectations of uniform generator.
testBinomial	Test the probabilities and expectations of binomial generator.
testPoisson	Test the probabilities and expectations of poisson generator.
testVector	Test the probabilities and expectations of a random vector.
testInstructions	Test execution of instruction block within a scope.
testNaming	Test scope name propagation.
testLookup	Test symbol table lookup.
testBaseDeterministic	Full parse and instruction generation test on base_deterministic_test.rll.
testGamblers	Full parse and instruction generation test on gamblers.rll.
testCarrental	Full parse and instruction generation test on carrental.rll.
testHh	Full parse and instruction generation test on hh.rll.
testSolver	Full parse and instruction generation test on solver.rll.

6.2 Manual Tests

The results were shown in the tutorial section, but POMDP files were created for all four MDPs and the corresponding transition probabilities and reward

expectations were checked.

6.3 Test Status

All automated tests pass except testSolver since the tree walker does not fully implement the statements in that file. I have spot checked the output of the manual tests against expected values, and in the case of carrental.rll, values I generated independently from a separate program and all of those tests pass.

7 Lessons Learned

I have mixed feelings about a couple of the choices I made. Two choices in particular led to significant effort on my part. First was the implementation language choice of C++ and second was the decision that instructions operate at run time on concrete types, I didn't want dispatching and type resolution going on during interpretation. On the one hand, these were good choices since, even with all the performance focus, it still takes over 10 minutes to compute the nearly 2 million transition elements for the carrental case. The interpreter would never be able to complete if I had implemented in Java. On the other hand, the C++ and concrete type choices led to some pretty hairy template code that took quite a while to work right, and more importantly work with minimal annotation to create all possible type-specific variants of instructions. I toyed with the idea of implementing this in OCAML. Now that I am done, I realize I was right in not trying to tackle a new language on top of all the code I had to write, but that neither C++ or Java are the right languages to be writing this type of code. I really want a functional language and all the Boost components for doing that are impressive in their ambition, but they still seem too constrained by C++ syntax and baggage.

Another regret is not looking into the ANTLR tree walking syntax more. I wanted to provide very detailed and complete error checking in my tree walker. I wasn't exactly sure how to make that work in ANTLR. I also did not like that the ANTLR plugin was tuned to expect Java code in the ANTLR files. I really didn't want to do significant editing of C++ in the ANTLR editor. So in the end I wound up implementing my own tree walker. I used macros as much as I could to simplify the syntax, but it was still quite a bit of code to write and is fairly messy. I am not sure doing it in ANTLR would have made it much cleaner, but I am guessing it might have went smoother if I did.

References

[Cas] Tony Cassandra, *POMDP File Specification*, http://www.cs.brown.edu/research/ai/pomdp/examples/pomdp-file-spec.html.

[SB98] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, MA, 1998.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic Robotics*, The MIT Press, Cambridge, MA, 2005.

[VJ03] David Vandevoorde and Nicolai M. Josuttis, *C++ Templates: The Complete Guide*, Addison-Wesley, Boston, MA, 2003.

[vR] Guido van Rossum, *The Python Programming Language*, http://www.python.org/.

A Appendix: RLL Language Specification

```
class RllLexer extends Lexer;
OPEN_PAREN :
        '('
                ;
CLOSE_PAREN :
        ')'
                ;
OPEN_BRACK :
        '['
                ;
CLOSE_BRACK :
        ']'
                ;
OPEN_CURLY :
        '{'
                ;
CLOSE_CURLY :
        '}'
                ;
COLON :
        ':'
                ;
COMMA :
        ','
                ;
SEMI :
        ';'
                ;
PLUS :
        '+'
                ;
MINUS :
```

```
        '-'
                ;
STAR :
        '*'
                ;
SLASH :
        '/'
                ;
ASSIGN :
        '='
                ;
PLUS_ASSIGN :
        "+="
                ;
MINUS_ASSIGN :
        "-="
                ;
STAR_ASSIGN :
        "*="
                ;
SLASH_ASSIGN :
        "/="
                ;
LESS_THAN :
        '<'
                ;
GREATER_THAN :
        '>'
                ;
EQUAL :
        "=="
                ;
NOT_EQUAL :
        "!="
                ;
LESS_OR_EQUAL :
        "<="
                ;
GREATER_OR_EQUAL :
        ">="
                ;
LOGICAL_AND :
        "&&"
                ;
LOGICAL_OR :
        "||"
```

```
                ;
LOGICAL_NOT :
        '!'
                ;
TICK :
        '\''
                ;
NUMBER :
        '.' Int
        ( Exponent )?

        | Int
        ( '.' ( Int )? ( Exponent )? | Exponent | )
        ;
Int :
        ( '0'..'9' )+
        ;
Exponent :
        ( 'e' | 'E' )
        ( '+' | '-' )?
        Int
        ;
ID :
        ( 'a'..'z' | 'A'..'Z' )
        ( 'a'..'z' | 'A'..'Z' | '_' | '0'..'9' )*
        ;
STRING_LITERAL :
        '"'
        ( ~( '"' | '\n' | '\r' ) )*
        '"'
        ;
CONTINUED_LINE :
        '\\'
        ( '\r' )?
        '\n'
        ( ' ' | '\t' )*
        ;
LEADING_WS :
        {getColumn()==1}?
        ( ' ' | '\t' | '\014' )+
        ( {implicitLineJoiningLevel==0}? ( '\r' )? '\n'
          | '#' ( ~'\n' )* ( '\n' )+ )?
        ;
COMMENT :
        '#'
        ( ~'\n' )*
```

```
        ( {startCol==1}? ( '\n' )+ )?
        ;
NEWLINE :
        ( ( '\r' )? '\n' )+
        ;
WS :
        ( ' ' | '\t' )+
        ;
class RllParser extends Parser;
fileInput :
        ( NEWLINE | statement )*
        EOF
        ;
statement :
        mdpDefinition
        | proceduralStatement
        ;
proceduralStatementList :
        ( proceduralStatement )+
        ;
proceduralStatement :
        assignmentStatement
        | functionCallStatement
        | forStatement
        | whileStatement
        | ifStatement
        ;
mdpDefinition :
        "mdp" ID OPEN_PAREN
        ( parameterList )?
        CLOSE_PAREN NEWLINE INDENT mdpBody DEDENT
        ;
parameterList :
        parameter
        ( COMMA parameter )*
        ;
parameter :
        ID
        ( ASSIGN expression )?
        ;
mdpBody :
        ( diagramDeclaration )?
        stateDeclaration actionDeclaration rewardDeclaration
        ( constraintDeclaration )*
        ( episodeDeclaration )?
        environmentDeclaration
```

```
        ;
stateDeclaration :
        "states"
        ( variableDeclarationStatement
        | NEWLINE INDENT ( variableDeclarationStatement )+ DEDENT )
        ;
actionDeclaration :
        "actions"
        ( variableDeclarationStatement
        | NEWLINE INDENT ( variableDeclarationStatement )+ DEDENT )
        ;
rewardDeclaration :
        "reward" ID NEWLINE
        ;
constraintDeclaration :
        simpleAssignmentStatement
        ;
episodeDeclaration :
        "episode" NEWLINE INDENT
        ( episodeStatement )+
        DEDENT
        ;
diagramDeclaration :
        "diagram" ID NEWLINE INDENT
        ( STRING_LITERAL NEWLINE )+
        DEDENT
        ;
environmentDeclaration :
        "environment" NEWLINE INDENT
        ( environmentStatement )+
        DEDENT
        ;
variableDeclarationStatement :
        variableDeclaration variableRangeConstraint NEWLINE
        ;
episodeStatement :
        constraintDeclaration
        | randomDeclaration
        ;
setDeclaration :
        "set" ID ASSIGN expression NEWLINE
        ;
environmentStatement :
        randomDeclaration
        | assignmentStatement
        ;
```

```
randomDeclaration :
        "random" variableDeclaration ASSIGN expression NEWLINE
        ;
forStatement :
        "for" OPEN_PAREN iteratorExpression CLOSE_PAREN NEWLINE
         INDENT proceduralStatementList DEDENT
        ;
whileStatement :
        "while" OPEN_PAREN expression CLOSE_PAREN NEWLINE
         INDENT proceduralStatementList DEDENT
        ;
ifStatement :
        "if" OPEN_PAREN expression CLOSE_PAREN NEWLINE
         INDENT proceduralStatementList DEDENT
        ( "elsif" OPEN_PAREN expression CLOSE_PAREN NEWLINE
         INDENT proceduralStatementList DEDENT )*
        ( "else" NEWLINE INDENT proceduralStatementList DEDENT )?
        ;
simpleAssignmentStatement :
        assignableExpression ASSIGN expression NEWLINE
        ;
assignmentStatement :
        assignableExpression
        ( ASSIGN | PLUS_ASSIGN | MINUS_ASSIGN
        | STAR_ASSIGN | SLASH_ASSIGN )
        expression NEWLINE
        ;
expression :
        booleanOrExpression
        ( "if" booleanOrExpression "else" booleanOrExpression )?
        ;
booleanOrExpression :
        booleanAndExpression
        ( LOGICAL_OR booleanAndExpression )*
        ;
booleanAndExpression :
        booleanEqualityExpression
        ( LOGICAL_AND booleanEqualityExpression )*
        ;
booleanEqualityExpression :
        booleanRelationExpression
        ( ( EQUAL | NOT_EQUAL ) booleanRelationExpression )*
        ;
booleanRelationExpression :
        additiveExpression
        ( ( LESS_THAN | GREATER_THAN | LESS_OR_EQUAL
```

```
          | GREATER_OR_EQUAL | "in" | "not" "in" ) additiveExpression )*
        ;
additiveExpression :
        multiplicativeExpression
        ( ( PLUS | MINUS ) multiplicativeExpression )*
        ;
multiplicativeExpression :
        unaryExpression
        ( ( STAR | SLASH ) unaryExpression )*
        ;
unaryExpression :
        LOGICAL_NOT valueExpression
        |
        ( PLUS | MINUS )
        valueExpression
        | valueExpression
        ;
valueExpression :
        atomicAssignableExpression
        | functionCall
        | set
        | array
        | INT
        | FLOAT
        | STRING_LITERAL
        | OPEN_PAREN expression CLOSE_PAREN
        ;
assignableExpression :
        atomicAssignableExpression
        | assignableExpressionList
        ;
atomicAssignableExpression :
        atomicIndexableAssignableExpression
        ( OPEN_BRACK expression CLOSE_BRACK )?
        ;
atomicIndexableAssignableExpression :
        ID
        ( TICK )?
        ;
assignableExpressionList :
        LESS_THAN atomicAssignableExpression
        ( COMMA atomicAssignableExpression )*
        GREATER_THAN
        ;
iteratorExpression :
        ID ASSIGN rangeExpression
```

```
        | ID "in" set
        ;
rangeExpression :
        expression COLON expression
        ;
functionCallStatement :
        functionCall NEWLINE
        ;
functionCall :
        ID OPEN_PAREN
        ( argumentList )?
        CLOSE_PAREN
        ;
argumentList :
        argument
        ( COMMA argument )*
        ;
argument :
        positionalArgument
        | namedArgument
        ;
positionalArgument :
        expression
        ;
namedArgument :
        ID ASSIGN expression
        ;
variableDeclaration :
        ID
        ( variableDimension )?
        ;
variableRangeConstraint :
        bounds
        | ID
        ;
variableDimension :
        OPEN_BRACK INT CLOSE_BRACK
        ;
array :
        OPEN_BRACK arrayRow
        ( SEMI arrayRow )*
        CLOSE_BRACK
        ;
set :
        OPEN_CURLY setElement
        ( COMMA setElement )*
```

```
        CLOSE_CURLY
        ;
bounds :
        OPEN_CURLY setElement
        ( COMMA setElement )*
        CLOSE_CURLY
        ;
setElement :
        expression
        ( COLON expression )?
        ;
arrayRow :
        expression
        ( COMMA expression )*
        ;
```

Contents

1 rll.g

```
1  header "post_include_hpp"{
2  #include "LineCountingAST.hpp"
3  }
4
5
6  options {
7      language="Cpp";
8  }
9
10
11 /*
12   The following lexer was based on the Python lexer created by Terence Parr
13   and Loring Craymer.  In particular the constructs used to correctly
14   handle Python's way of using indentation to delimit compound statements
15
16   I converted it to C++ and modified it to support the rll language.
17   Mike Groble  Oct 1 2006
18 */
19
20
21 /*
22 [The "BSD licence"]
23 Copyright (c) 2004 Terence Parr and Loring Craymer
24 All rights reserved.
25
26 Redistribution and use in source and binary forms, with or without
27 modification, are permitted provided that the following conditions
28 are met:
```

```
1.  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
2.  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.
3.  The name of the author may not be used to endorse or promote products
    derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/



class RllLexer extends Lexer;

options {
    k=2;
    testLiterals=false; // have to test in identifier rule
}

{
/** Handles context-sensitive lexing of implicit line joining such as
*  the case where newline is ignored in cases like this:
*  a = [3,
*       4]
*/
   private:
   int implicitLineJoiningLevel;

   public:
   void init() {
      implicitLineJoiningLevel = 0;
   }
}

OPEN_PAREN:  '('{implicitLineJoiningLevel++;};
CLOSE_PAREN: ')'{implicitLineJoiningLevel--;};

OPEN_BRACK:  '['{implicitLineJoiningLevel++;};
CLOSE_BRACK: ']'{implicitLineJoiningLevel--;};

OPEN_CURLY:  '{'{implicitLineJoiningLevel++;};
CLOSE_CURLY: '}'{implicitLineJoiningLevel--;};

COLON: ':';
COMMA: ',';
SEMI:  ';';

PLUS:   '+';
MINUS:  '-';
STAR:   '*';
SLASH : '/';
ASSIGN: '=';

PLUS_ASSIGN:  "+=";
MINUS_ASSIGN: "-=";
STAR_ASSIGN:  "*=";
SLASH_ASSIGN: "/=";

LESS_THAN:         '<';
GREATER_THAN:      '>';
EQUAL:             "==";
NOT_EQUAL:         "!=";
LESS_OR_EQUAL:     "<=";
GREATER_OR_EQUAL: ">=";

LOGICAL_AND: "&&";
LOGICAL_OR:  "||";
LOGICAL_NOT: '!';

TICK: '\'';

```

```
NUMBER
  :  '.'Int (Exponent)?  {$setType(FLOAT);}
  |Int ( '.'(Int)?  (Exponent)?  {$setType(FLOAT);}
       |Exponent {$setType(FLOAT);}
       |/*nothing*/{$setType(INT);})
  ;

protected
Int
  :  ('0'..'9')+
  ;

protected
Exponent
  :  ('e'|'E') ('+'|'-')?  Int
  ;

ID
options {
    testLiterals = true;
}
  :  ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*
  ;

STRING_LITERAL
options {
    testLiterals = false;
}
  :  '"'!  (~('"'|'\n'|'\r'))* '"'!
  ;

/** Consume a newline and any whitespace at start of next line */
CONTINUED_LINE
  :  '\\'('\r')?  '\n'(' '|'\t')* { newline(); $setType(antlr::Token::SKIP); }
  ;

/** Grab everything before a real symbol.  Then if newline, kill it
*  as this is a blank line.  If whitespace followed by comment, kill it
*  as it's a comment on a line by itself.
*
*  Ignore leading whitespace when nested in [..], (..), {..}.
*/
LEADING_WS
{
    int spaces = 0;
}
  :  {getColumn()==1}?
     // match spaces or tabs, tracking indentation count
    ( ' '  { spaces++; }
    |'\t'{ spaces += 8; spaces -= (spaces % 8); }
    |'\014'// formfeed is ok
    )+
    {
      if ( implicitLineJoiningLevel>0 ) {
          // ignore ws if nested
          $setType(antlr::Token::SKIP);
      }
      else {
          std::string s(spaces,' ');
          $setText(s);
      }
    }
    // kill trailing newline or comment
    ( {implicitLineJoiningLevel==0}?  ('\r')?  '\n'{newline();}
      {$setType(antlr::Token::SKIP);}

    |  // if comment, then only thing on a line; kill so we
       // ignore totally also wack any following newlines as
       // they cannot be terminating a statement
      '#'(~'\n')* ('\n'{newline();})+
      {$setType(antlr::Token::SKIP);}
    )?
  ;

/** Comments not on line by themselves are turned into newlines because
    sometimes they are newlines like

    b = a # end of line comment

    or
```

```
    a = [1, # weird
         2]

    This rule is invoked directly by nextToken when the comment is in
    first column or when comment is on end of nonwhitespace line.

    The problem is that then we have lots of newlines heading to
    the parser.  To fix that, column==1 implies we should kill whole line.

    Consume any newlines following this comment as they are not statement
    terminators.  Don't let NEWLINE token handle them.
*/

COMMENT
{
    int startCol = getColumn();
}
  :  '#'(~'\n')* // let NEWLINE handle \n unless column = 1 for '#'
    { $setType(antlr::Token::SKIP); }
    ( {startCol==1}?  ('\n'{newline();})+ )?
  ;

/** Treat a sequence of blank lines as a single blank line.  If
*  nested within a (..), {..}, or [..], then ignore newlines.
*  If the first newline starts in column one, they are to be ignored.
*/
NEWLINE
{
    int startCol = getColumn();
}
  :  (options{greedy=true;}:('\r')?  '\n'{newline();})+
    {if ( startCol==1 ||implicitLineJoiningLevel>0 )
        $setType(antlr::Token::SKIP);
    }
  ;

WS
  :  (' '|'\t')+ {$setType(antlr::Token::SKIP);}
  ;


class RllParser extends Parser;

options {
    k=2;
    buildAST = true;
    ASTLabelType = "LineCountingAST::Ptr";
    defaultErrorHandler=false;
}

tokens {
    FILE;
    PARAMS;
    ARGS;
    MDP_BODY;
    SET;
    ARRAY;
    ROW;
    DIM;
    CALL;
    LHS_LIST;
    INDEX;
    DECL;
    BOUNDS;
    IF_EXPR;
    UNARY_PLUS;
    UNARY_MINUS;
    PS_LIST;
    ITER;
    NARG;
}

fileInput
  :  (NEWLINE |statement)* EOF!
    {#fileInput = #([FILE,"FILE"], fileInput); }
  ;

statement
  :  mdpDefinition
  |proceduralStatement
  ;
```

```
272
273 proceduralStatementList
274   :  (proceduralStatement)+
275     {#proceduralStatementList = #([PS_LIST,"PS_LIST"], proceduralStatementList);}
276   ;
277
278 proceduralStatement
279   :  assignmentStatement
280   |functionCallStatement
281   |forStatement
282   |whileStatement
283   |ifStatement
284   ;
285
286 mdpDefinition
287   :  "mdp"^ID OPEN_PAREN! (parameterList)?  CLOSE_PAREN! NEWLINE!
288      INDENT! mdpBody DEDENT!
289   ;
290
291 parameterList
292   :  parameter (COMMA! parameter)*
293     {#parameterList = #([PARAMS,"PARAMS"],parameterList);}
294   ;
295
296 parameter
297   :  ID^(ASSIGN^expression)?
298   ;
299
300 mdpBody
301   :  (diagramDeclaration)?
302     stateDeclaration
303     actionDeclaration
304     rewardDeclaration
305     (constraintDeclaration)*
306     (episodeDeclaration)?
307     environmentDeclaration
308     {#mdpBody = #([MDP_BODY,"MDP_BODY"],mdpBody);}
309   ;
310
311 stateDeclaration
312   :  "states"^
313       ( variableDeclarationStatement
314       |NEWLINE!
315         INDENT! (variableDeclarationStatement)+ DEDENT!
316       )
317   ;
318
319
320 actionDeclaration
321   :  "actions"^
322       ( variableDeclarationStatement
323       |NEWLINE!
324         INDENT! (variableDeclarationStatement)+ DEDENT!
325       )
326   ;
327
328 rewardDeclaration
329   :  "reward"^ID NEWLINE!
330   ;
331
332 constraintDeclaration
333   :  simpleAssignmentStatement
334   ;
335
336 episodeDeclaration
337   :  "episode"^NEWLINE!
338     INDENT! (episodeStatement)+ DEDENT!
339   ;
340
341 diagramDeclaration
342   :  "diagram"^ID NEWLINE!
343     INDENT! (STRING_LITERAL NEWLINE!)+ DEDENT!
344   ;
345 environmentDeclaration
346   :  "environment"^NEWLINE!
347     INDENT! (environmentStatement)+ DEDENT!
348   ;
349
350 variableDeclarationStatement
351   :  variableDeclaration variableRangeConstraint NEWLINE!
352    {#variableDeclarationStatement = #([DECL,"DECL"],variableDeclarationStatement);}
```

```
353   ;
354
355 episodeStatement
356   :  constraintDeclaration
357   |randomDeclaration
358   ;
359
360 setDeclaration
361   :  "set"^ID ASSIGN! expression NEWLINE!
362   ;
363
364 environmentStatement
365   :  randomDeclaration
366   |assignmentStatement
367   ;
368
369 randomDeclaration
370   :  "random"^variableDeclaration ASSIGN! expression NEWLINE!
371   ;
372
373 forStatement
374   :  "for"^OPEN_PAREN! iteratorExpression CLOSE_PAREN! NEWLINE!
375     INDENT! proceduralStatementList DEDENT!
376   ;
377
378 whileStatement
379   :  "while"^OPEN_PAREN! expression CLOSE_PAREN! NEWLINE!
380     INDENT! proceduralStatementList DEDENT!
381   ;
382
383 ifStatement
384   :  "if"^OPEN_PAREN! expression CLOSE_PAREN! NEWLINE!
385     INDENT! proceduralStatementList DEDENT!
386     ("elsif"OPEN_PAREN! expression CLOSE_PAREN! NEWLINE!
387      INDENT! proceduralStatementList DEDENT!)*
388     ("else"  NEWLINE!
389      INDENT! proceduralStatementList DEDENT!)?
390   ;
391
392 simpleAssignmentStatement
393   :  assignableExpression ASSIGN^expression NEWLINE!
394   ;
395
396 assignmentStatement
397   :  assignableExpression
398     ( ASSIGN^
399     |PLUS_ASSIGN^
400     |MINUS_ASSIGN^
401     |STAR_ASSIGN^
402     |SLASH_ASSIGN^
403     )
404    expression NEWLINE!
405   ;
406
407 expression
408   :  booleanOrExpression ("if"!  booleanOrExpression "else"!  booleanOrExpression
409                           {#expression = #([IF_EXPR,"IF_EXPR"],expression);})?
410   ;
411
412 booleanOrExpression
413   :  booleanAndExpression (LOGICAL_OR^booleanAndExpression)*
414   ;
415
416 booleanAndExpression
417   :  booleanEqualityExpression (LOGICAL_AND^booleanEqualityExpression)*
418   ;
419
420 booleanEqualityExpression
421   :  booleanRelationExpression ((EQUAL^|NOT_EQUAL^) booleanRelationExpression)*
422   ;
423
424 booleanRelationExpression
425   :  additiveExpression ((LESS_THAN^
426                          |GREATER_THAN^
427                          |LESS_OR_EQUAL^
428                          |GREATER_OR_EQUAL^
429                          |"in"^
430                          |"not"^"in"!)  additiveExpression)*
431   ;
432
433 additiveExpression
```

```
434   :  multiplicativeExpression ((PLUS^|MINUS^) multiplicativeExpression)*
435   ;
436
437 multiplicativeExpression
438   :  unaryExpression ((STAR^|SLASH^) unaryExpression)*
439   ;
440
441 unaryExpression
442   :  LOGICAL_NOT^valueExpression
443   |(PLUS^{#PLUS->setType(UNARY_PLUS);}
444     |MINUS^{#MINUS->setType(UNARY_MINUS);}) valueExpression
445   |valueExpression
446   ;
447
448 valueExpression
449   :  atomicAssignableExpression
450   |functionCall
451   |set
452   |array
453   |INT
454   |FLOAT
455   |STRING_LITERAL
456   |OPEN_PAREN^expression CLOSE_PAREN!
457   ;
458
459
460 assignableExpression
461   :  atomicAssignableExpression
462   |assignableExpressionList
463   ;
464
465 atomicAssignableExpression
466   :  atomicIndexableAssignableExpression
467     (OPEN_BRACK! expression CLOSE_BRACK! {#atomicAssignableExpression = #([INDEX,"INDEX"], atomicAssignableExpression);})?
468   ;
469
470 atomicIndexableAssignableExpression
471   :  ID (TICK^)?
472   ;
473
474 assignableExpressionList
475   :  LESS_THAN! atomicAssignableExpression (COMMA! atomicAssignableExpression)* GREATER_THAN!
476   {#assignableExpressionList = #([LHS_LIST,"LHS_LIST"], assignableExpressionList);}
477   ;
478
479 iteratorExpression
480   :  ID ASSIGN! rangeExpression {#iteratorExpression = #([ITER,"ITER"], iteratorExpression);}
481   |ID "in"!  set {#iteratorExpression = #([ITER,"ITER"], iteratorExpression);}
482   ;
483
484 rangeExpression
485   :  expression COLON^expression
486   ;
487
488 functionCallStatement
489   :  functionCall NEWLINE!
490   ;
491
492 functionCall
493   :  ID OPEN_PAREN! (argumentList)?  CLOSE_PAREN!
494     {#functionCall = #([CALL,"CALL"],functionCall);}
495   ;
496
497 argumentList
498   :  argument (COMMA! argument)*
499   {#argumentList = #([ARGS,"ARGS"],argumentList);}
500   ;
501
502 argument
503   :  positionalArgument
504   |namedArgument
505   ;
506
507 positionalArgument
508   :  expression
509   ;
510
511 namedArgument
512   :  ID ASSIGN! expression {#namedArgument = #([NARG,"NARG"], #namedArgument);}
513   ;
514
```

```
515 variableDeclaration
516   :  ID^(variableDimension)?
517   ;
518
519 variableRangeConstraint
520   :  bounds
521   |ID^
522   ;
523
524 variableDimension
525   :  OPEN_BRACK! INT CLOSE_BRACK!
526     {#variableDimension = #([DIM,"DIM"],variableDimension);}
527   ;
528
529 array
530   :  OPEN_BRACK! arrayRow (SEMI! arrayRow)* CLOSE_BRACK!
531     {#array = #([ARRAY,"ARRAY"], array);}
532   ;
533
534 set
535   :  OPEN_CURLY! setElement (COMMA! setElement)* CLOSE_CURLY!
536     {#set = #([SET,"SET"], set);}
537   ;
538
539 bounds
540   :  OPEN_CURLY! setElement (COMMA! setElement)* CLOSE_CURLY!
541     {#bounds = #([BOUNDS,"BOUNDS"],bounds);}
542   ;
543
544 setElement
545   :  expression (COLON^expression)?
546   ;
547
548 arrayRow
549   :  expression (COMMA! expression)*
550     {#arrayRow = #([ROW,"ROW"], arrayRow);}
551   ;
552
553 //class RllWalker extends TreeParser;
554 //options {
555 //  ASTLabelType = "antlr::RefCommonAST";
556 //}
557 //
558 //fileInput [Scope & scope] returns [ RllType r ]
559 //{
560 //  RllType a;
561 //}
562 //  :  #(FILE a=mdp[scope])
563 //  ;
564 //
565 //mdp  [Scope & scope] returns [RllType r]
566 //{
567 //  MarkovDecisionProcessType::Ptr mdp(new MarkovDecisionProcessType());
568 //  std::string name;
569 //}
570 //  :  #("mdp" id:ID (param:PARAMS { })?  body:MDP_BODY { })
571 //  {
572 //      scope.addSymbol(id->getText(),mdp);
573 //  }
574 //
575 //  ;
576 //
577
```

2 BinomialRandom.hpp

```
1 #ifndef BINOMIALRANDOM_HPP_
2 #define BINOMIALRANDOM_HPP_
3
4
5 #include "DiscreteRandom.hpp"
6
7
8 class BinomialRandom :  public DiscreteRandom
9 {
10 public:
11     typedef boost::bernoulli_distribution<> Distribution;
12     typedef boost::variate_generator<RandomGenerator&,Distribution> Generator;
13
```

```
14      BinomialRandom(RandomGenerator& generator, double p);
15
16      virtual int operator() ();
17
18      virtual double probability(int i) const;
19
20      virtual ~BinomialRandom();
21
22  private:
23      Generator _generator;
24  };
25
26  #endif /*BINOMIALRANDOM_HPP_*/
```

3 Diagram.hpp

```
1   #ifndef DIAGRAM_HPP_
2   #define DIAGRAM_HPP_
3
4   #include <valarray>
5   #include <map>
6   #include <set>
7
8   class Diagram
9   {
10  public:
11      Diagram();
12
13      bool addRow(std::string const& row);
14
15      bool contains(std::valarray<int> const& vector, std::string const & chars);
16
17      int rows() const;
18      int cols() const;
19
20  private:
21      int _numCols;
22      int _numRows;
23      typedef std::map<char,std::set<long> > Positions;
24      Positions _positions;
25
26      long index(int row, int col) const;
27  };
28
29  #endif /*DIAGRAM_HPP_*/
```

4 DiscreteRandom.hpp

```
1   #ifndef DISCRETERANDOM_HPP_
2   #define DISCRETERANDOM_HPP_
3
4   #include "boost/random.hpp"
5   #include "IntervalSet.hpp"
6
7   class DiscreteRandom :  public IntervalSet
8   {
9   public:
10      typedef boost::mt19937 RandomGenerator;
11
12      virtual int operator() () const;
13
14      virtual double probability(int i) const;
15
16      virtual ~DiscreteRandom();
17
18  protected:
19      DiscreteRandom(RandomGenerator& generator, IntervalSet::Interval interval);
20
21      RandomGenerator& _numberGenerator;
22  };
23
24  #endif /*DISCRETERANDOM_HPP_*/
```

5 Dispatcher.hpp

```
#ifndef DISPATCHER_HPP_
#define DISPATCHER_HPP_

#include "Types.hpp"
#include "Scope.hpp"
#include "RllLexerTokenTypes.hpp"
#include "MultiMethods.hpp"
#include "SingleMethods.hpp"
#include "util.hpp"
#include "InstructionFactory.hpp"

class Dispatcher :  public RllLexerTokenTypes
{
public:
    enum {
        FAKE_MAX = 1000,
        FAKE_MIN,
        FAKE_ABS
    };

    Dispatcher();

    RllType::Ptr dispatchOps(RllType::Ptr in, int type, Scope & scope);
    RllType::Ptr dispatchOps(RllType::Ptr lhs, RllType::Ptr rhs, int type, Scope & scope);
    bool dispatchInstructions(RllType::Ptr lhs, RllType::Ptr rhs, int type, Scope & scope);

private:
    typedef std::map<int,UnaryDispatcher<boost::shared_ptr> > UnaryDispatchers;
    UnaryDispatchers _unaryDispatchers;
    typedef std::map<int,FnDispatcher<void,boost::shared_ptr> > UnaryInstructionDispatchers;
    UnaryInstructionDispatchers _unaryInstructionDispatchers;
    typedef std::map<int,FnDispatcher<RllType::Ptr,boost::shared_ptr> > BinaryDispatchers;
    BinaryDispatchers _binaryDispatchers;

    template <typename I, typename O, OpReturn (*Op)(typename I::Ref,typename O::Ref)>
    void addUnary(int op) {
        UnaryDispatchers::iterator dispatcher = findOrCreate(_unaryDispatchers,op);
        dispatcher->second.add<I,O,addUnaryOp<I,O,Op> >();
    }

    template <typename I, typename O, OpReturn (*Op)(typename I::Ref,typename O::Ref)>
    void addUnaryInstruction(int op) {
        UnaryInstructionDispatchers::iterator dispatcher = findOrCreate(_unaryInstructionDispatchers,op);
        dispatcher->second.add<I,O,addUnaryOpInstruction<I,O,Op> >();
    }

    template <typename L, typename R, typename O, OpReturn (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
    void addBinary(int op) {
        BinaryDispatchers::iterator dispatcher = findOrCreate(_binaryDispatchers,op);
        dispatcher->second.add<L,R,O,addBinaryOp<L,R,O,Op> >();
    }

    template <typename L, typename R, typename O, OpReturn (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
    void addLhsArrayBinary(int op) {
        BinaryDispatchers::iterator dispatcher = findOrCreate(_binaryDispatchers,op);
        dispatcher->second.add<L,R,O,addLhsArrayBinaryOp<L,R,O,Op> >();
    }

    template <typename L, typename R, typename O, OpReturn (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
    void addRhsArrayBinary(int op) {
        BinaryDispatchers::iterator dispatcher = findOrCreate(_binaryDispatchers,op);
        dispatcher->second.add<L,R,O,addRhsArrayBinaryOp<L,R,O,Op> >();
    }

    template <typename L, typename R, typename O, OpReturn (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
    void addArrayBinary(int op) {
        BinaryDispatchers::iterator dispatcher = findOrCreate(_binaryDispatchers,op);
        dispatcher->second.add<L,R,O,addArrayBinaryOp<L,R,O,Op> >();
    }

};

#endif /*DISPATCHER_HPP_*/
```

6 Exceptions.hpp

```
#ifndef EXCEPTIONS_HPP_
#define EXCEPTIONS_HPP_

#include <stdexcept>
#include <string>

class InvalidTreeStructureException :  public std::logic_error
{
public:
    InvalidTreeStructureException(std::string const& s);
    virtual ~InvalidTreeStructureException() throw();
};

class ModifyConstantException :  public std::logic_error
{
public:
    ModifyConstantException(std::string const& s);
    virtual ~ModifyConstantException() throw();
};


class UnimplementedOperationException :  public std::logic_error
{
public:
    UnimplementedOperationException(std::string const& s);
    virtual ~UnimplementedOperationException() throw();
};

#endif /*EXCEPTIONS_HPP_*/
```

7 FunctionCall.hpp

```
#ifndef FUNCTIONCALL_HPP_
#define FUNCTIONCALL_HPP_

#include "Instruction.hpp"
#include "Types.hpp"

class FunctionCall :  public Instruction
{
public:
    FunctionCall(FunctionType::Ptr fn);
    virtual void operator()() const;
    virtual ~FunctionCall();

private:
    FunctionType::Ptr _function;
};

class TernaryCall :  public Instruction
{
public:
    TernaryCall(TernaryType::Ptr fn);
    virtual void operator()() const;
    virtual ~TernaryCall();

private:
    TernaryType::Ptr _function;
};

class DiagramMemberInstruction :  public Instruction
{
public:
    DiagramMemberInstruction(DiagramType::Ptr diagram, IntegerArrayType::Ptr array, std::string const& chars, BooleanType::Ptr
out);
    virtual void operator()() const;
    virtual ~DiagramMemberInstruction();

private:
    DiagramType::Ref _diagramRef;
    IntegerArrayType::Ref _arrayRef;
    std::string _chars;
    BooleanType::Ref _outRef;
    DiagramType::Ptr _diagramVariable;
    IntegerArrayType::Ptr _arrayVariable;
    BooleanType::Ptr _outVariable;
};
```

```
45
46 class RandomCreateInstruction :  public Instruction
47 {
48 public:
49     RandomCreateInstruction(RandomType::Ptr random, DiscreteRandom::RandomGenerator & generator);
50     virtual void operator()() const;
51     virtual ~RandomCreateInstruction();
52
53 private:
54     RandomType::Ptr _random;
55     DiscreteRandom::RandomGenerator & _generator;
56 };
57
58 class RandomDrawInstruction :  public Instruction
59 {
60 public:
61     RandomDrawInstruction(RandomType::Ptr random);
62     virtual void operator()() const;
63     virtual ~RandomDrawInstruction();
64
65 private:
66     RandomType::Ptr _random;
67 };
68
69
70 #endif /*FUNCTIONCALL_HPP_*/
```

8 Instruction.hpp

```
 1 #ifndef INSTRUCTION_HPP_
 2 #define INSTRUCTION_HPP_
 3
 4 #include <iostream>
 5 #include <vector>
 6 #include <valarray>
 7
 8 #include "boost/smart_ptr.hpp"
 9
10 //#define ENABLE_INST_LOG
11
12 #ifdef ENABLE_INST_LOG
13 typedef std::string OpReturn;
14 #define RETURN(s) return s
15 #else
16 typedef void OpReturn;
17 #define RETURN(s)
18 #endif
19
20 class Instruction
21 {
22 public:
23     virtual void operator()() const = 0;
24
25     virtual ~Instruction() {}
26 };
27
28 typedef boost::shared_ptr<Instruction> InstructionPtr;
29 typedef std::vector<InstructionPtr> InstructionBlock;
30
31 template <typename T> inline void printValue(std::ostream & out, T & v) {}
32 template <> inline void printValue(std::ostream & out, int & v) {out << " = "<< v;}
33 template <> inline void printValue(std::ostream & out, double & v) {out << " = "<< v;}
34 template <> inline void printValue(std::ostream & out, bool & v) {out << " = "<< v;}
35
36
37
38 template <typename I, typename O, OpReturn (*Op)(typename I::Ref,typename O::Ref)>
39 class UnaryInstruction :  public Instruction
40 {
41 public:
42     UnaryInstruction(typename I::Ptr in, typename O::Ptr out)
43       :  _inRef(in->ref())
44       , _outRef(out->ref())
45       , _inVariable(in)
46       , _outVariable(out)
47     {}
48
49     virtual void operator()() const
50   {
```

```
51      ;
52  #ifdef ENABLE_INST_LOG
53      std::cout << Op(_inRef,_outRef) << " in:  "<< _inVariable->name();
54      printValue(std::cout, _inRef);
55      std::cout << " out:  "<< _outVariable->name();
56      printValue(std::cout, _outRef);
57      std::cout << std::endl;
58  #else
59      Op(_inRef,_outRef);
60  #endif
61    }
62
63    virtual ~UnaryInstruction() {}
64
65  private:
66      typename I::Ref _inRef;
67      typename O::Ref _outRef;
68      typename I::Ptr _inVariable;
69      typename O::Ptr _outVariable;
70  };
71
72  template <typename L, typename R, typename O, OpReturn (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
73  class BinaryInstruction :  public Instruction
74  {
75  public:
76      BinaryInstruction(typename L::Ptr lhs, typename R::Ptr rhs, typename O::Ptr out)
77       :  _lhsRef(lhs->ref())
78       , _rhsRef(rhs->ref())
79       , _outRef(out->ref())
80       , _lhsVariable(lhs)
81       , _rhsVariable(rhs)
82       , _outVariable(out)
83      {}
84
85      virtual void operator()() const
86    {
87  #ifdef ENABLE_INST_LOG
88      std::cout <<
89      std::cout << Op(_lhsRef,_rhsRef,_outRef) << " lhs:  "<< _lhsVariable->name();
90      printValue(std::cout, _lhsRef);
91      std::cout << " rhs:  "<< _rhsVariable->name();
92      printValue(std::cout, _rhsRef);
93      std::cout << " out:  "<< _outVariable->name();
94      printValue(std::cout, _outRef);
95      std::cout << std::endl;
96  #else
97      Op(_lhsRef,_rhsRef,_outRef);
98  #endif
99    }
100
101   virtual ~BinaryInstruction() {}
102
103 private:
104     typename L::Ref _lhsRef;
105     typename R::Ref _rhsRef;
106     typename O::Ref _outRef;
107     typename L::Ptr _lhsVariable;
108     typename R::Ptr _rhsVariable;
109     typename O::Ptr _outVariable;
110 };
111
112 template <class I, class O> inline OpReturn assign  (I & in, O & out) {out = in;RETURN("assign");}
113 template <class I, class O> inline OpReturn negate  (I & in, O & out) {out = -in;RETURN("negate");}
114 template <class I, class O> inline OpReturn lnot    (I & in, O & out) {out = !in;RETURN("lnot");}
115 template <class I, class O> inline OpReturn abs     (I & in, O & out) {out = std::abs(in);RETURN("abs");}
116
117 template <class E, class C> inline OpReturn insert  (E & element, C & container) {container.insert(element);RETURN("insert");}
118
119 template <class L, class R, class O> inline OpReturn add      (L & lhs, R & rhs, O & out) {out = lhs + rhs;RETURN("add");}
120 template <class L, class R, class O> inline OpReturn subtract (L & lhs, R & rhs, O & out) {out = lhs - rhs;RETURN("subtract");}
121 template <class L, class R, class O> inline OpReturn multiply (L & lhs, R & rhs, O & out) {out = lhs * rhs;RETURN("multiply");}
122 template <class L, class R, class O> inline OpReturn divide   (L & lhs, R & rhs, O & out) {out = lhs / rhs;RETURN("divide");}
123
124 template <class L, class R, class O> inline OpReturn lessThan      (L & lhs, R & rhs, O & out) {out = lhs < rhs;RETURN("lessThan");}
125 template <class L, class R, class O> inline OpReturn greaterThan   (L & lhs, R & rhs, O & out) {out = lhs > rhs;RETURN("greaterThan");}
126 template <class L, class R, class O> inline OpReturn lessEqThan    (L & lhs, R & rhs, O & out) {out = lhs <= rhs;RETURN("lessEqThan");}
127 template <class L, class R, class O> inline OpReturn greaterEqThan (L & lhs, R & rhs, O & out) {out = lhs >= rhs;RETURN("greaterEqThan");}
128 template <class L, class R, class O> inline OpReturn equals        (L & lhs, R & rhs, O & out) {out = lhs == rhs;RETURN("equals");}
129 template <class L, class R, class O> inline OpReturn notEquals     (L & lhs, R & rhs, O & out) {out = lhs != rhs;RETURN("notEquals");}
130
131 template <class L, class R, class O> inline OpReturn lessThan      (std::valarray<L> & lhs, R & rhs, std::valarray<O>
```

```
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] < rhs;RETURN("lessThan");}
  132 template <class L, class R, class O> inline OpReturn greaterThan   (std::valarray<L> & lhs, R & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] > rhs;RETURN("greaterThan");}
  133 template <class L, class R, class O> inline OpReturn lessEqThan    (std::valarray<L> & lhs, R & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] <= rhs;RETURN("lessEqThan");}
  134 template <class L, class R, class O> inline OpReturn greaterEqThan (std::valarray<L> & lhs, R & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] >= rhs;RETURN("greaterEqThan");}
  135 template <class L, class R, class O> inline OpReturn equals        (std::valarray<L> & lhs, R & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] == rhs;RETURN("equals");}
  136 template <class L, class R, class O> inline OpReturn notEquals     (std::valarray<L> & lhs, R & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] != rhs;RETURN("notEquals");}
  137
  138 template <class L, class R, class O> inline OpReturn lessThan      (L & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs < rhs[i];RETURN("lessThan");}
  139 template <class L, class R, class O> inline OpReturn greaterThan   (L & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs > rhs[i];RETURN("greaterThan");}
  140 template <class L, class R, class O> inline OpReturn lessEqThan    (L & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs <= rhs[i];RETURN("lessEqThan");}
  141 template <class L, class R, class O> inline OpReturn greaterEqThan (L & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs >= rhs[i];RETURN("greaterEqThan");}
  142 template <class L, class R, class O> inline OpReturn equals        (L & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs == rhs[i];RETURN("equal");}
  143 template <class L, class R, class O> inline OpReturn notEquals     (L & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs != rhs[i];RETURN("notEquals");}
  144
  145 template <class L, class R, class O> inline OpReturn lessThan      (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] < rhs[i];RETURN("lessThan");}
  146 template <class L, class R, class O> inline OpReturn greaterThan   (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] > rhs[i];RETURN("greaterThan");}
  147 template <class L, class R, class O> inline OpReturn lessEqThan    (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] <= rhs[i];RETURN("lessEqThan");}
  148 template <class L, class R, class O> inline OpReturn greaterEqThan (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] >= rhs[i];RETURN("greaterEqThan");}
  149 template <class L, class R, class O> inline OpReturn equals        (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] == rhs[i];RETURN("equal");}
  150 template <class L, class R, class O> inline OpReturn notEquals     (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out) {for (std::size_t i = 0; i < out.size(); ++i) out[i] = lhs[i] != rhs[i];RETURN("notEquals");}
  151
  152 template <class L, class R, class O> inline OpReturn subscript  (L & lhs, R & rhs, O & out) {out = lhs[rhs];RETURN("subscript");}
  153 template <class L, class R, class O> inline OpReturn arrayAssign(L & lhs, R & rhs, O & out) {out[lhs] = rhs;RETURN("arrayAssign");}
  154
  155 template <class L, class R, class O> inline OpReturn intervalBounds (L & lhs, R & rhs, O & out) {out.assign(lhs,rhs);RETURN("intervalBounds")
  156
  157 template <class L, class R, class O> inline OpReturn in       (L & lhs, R & rhs, O & out) {out = rhs.contains(lhs);RETURN("in");}
  158 template <class L, class R, class O> inline OpReturn notIn    (L & lhs, R & rhs, O & out) {out = !rhs.contains(lhs);RETURN("notIn");}
  159
  160 template <class L, class R, class O> inline OpReturn max      (L & lhs, R & rhs, O & out) {out = lhs > rhs ?  lhs :  rhs;RETURN("max");}
  161 template <class L, class R, class O> inline OpReturn min      (L & lhs, R & rhs, O & out) {out = lhs < rhs ?  lhs :  rhs;RETURN("min");}
  162
  163 template <class L, class R, class O> inline OpReturn max      (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out)
  164                                                              {out = lhs;
  165                                                               for (std::size_t i = 0; i < out.size(); ++i) {
  166                                                                   if (out[i] < rhs[i]) {
  167                                                                       out[i] = rhs[i];
  168                                                                   }
  169                                                               }
  170                                                               RETURN("max");
  171                                                              }
  172 template <class L, class R, class O> inline OpReturn min      (std::valarray<L> & lhs, std::valarray<R> & rhs, std::valarray<O>
& out)
  173                                                               {out = lhs;
  174                                                                for (std::size_t i = 0; i < out.size(); ++i) {
  175                                                                    if (out[i] > rhs[i]) {
  176                                                                        out[i] = rhs[i];
  177                                                                    }
  178                                                                }
  179                                                                RETURN("min");
  180                                                               }
  181 template <class L, class R, class O> inline OpReturn max      (L & lhs, std::valarray<R> & rhs, std::valarray<O> & out)
  182                                                              {out = rhs;
  183                                                               for (std::size_t i = 0; i < out.size(); ++i) {
  184                                                                   if (out[i] < lhs) {
  185                                                                       out[i] = lhs;
  186                                                                   }
  187                                                               }
  188                                                               RETURN("max");
  189                                                              }
  190 template <class L, class R, class O> inline OpReturn min      (L & lhs, std::valarray<R> & rhs, std::valarray<O> & out)
  191                                                               {out = rhs;
  192                                                                for (std::size_t i = 0; i < out.size(); ++i) {
```

```
                                                                    if (out[i] > lhs) {
                                                                        out[i] = lhs;
                                                                    }
                                                                }
                                                                RETURN("min");
                                                            }

template <class L, class R, class O> inline OpReturn max    (std::valarray<L> & lhs, R & rhs, std::valarray<O> & out)
                                                            {out = lhs;
                                                             for (std::size_t i = 0; i < out.size(); ++i) {
                                                                 if (out[i] < rhs) {
                                                                     out[i] = rhs;
                                                                 }
                                                             }
                                                             RETURN("max");
                                                            }
template <class L, class R, class O> inline OpReturn min    (std::valarray<L> & lhs, R & rhs, std::valarray<O> & out)
                                                            {out = lhs;
                                                             for (std::size_t i = 0; i < out.size(); ++i) {
                                                                 if (out[i] > rhs) {
                                                                     out[i] = rhs;
                                                                 }
                                                             }
                                                             RETURN("min");
                                                            }

#undef RETURN
#endif /*INSTRUCTION_HPP_*/
```

9 InstructionFactory.hpp

```
#ifndef INSTRUCTIONFACTORY_HPP_
#define INSTRUCTIONFACTORY_HPP_

#include "Instruction.hpp"
#include "Scope.hpp"
#include "Types.hpp"

template <typename I, typename O, OpReturn (*Op)(typename I::Ref,typename O::Ref)>
void
addUnaryOpInstruction(typename I::Ptr in, typename O::Ptr out, Scope& scope)
{
    InstructionPtr inst(new UnaryInstruction<I,O,Op>(in,out));
    scope.addInstruction(inst);
}

template <typename I, typename O, OpReturn (*Op)(typename I::Ref,typename O::Ref)>
typename O::Ptr
addUnaryOp(typename I::Ptr in, Scope& scope)
{
    typename O::Ptr result(new O());
    addUnaryOpInstruction<I,O,Op>(in,result,scope);
    return result;
}

template <typename L, typename R, typename O, OpReturn (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
void
addBinaryOpInstruction(typename L::Ptr lhs, typename R::Ptr rhs, typename O::Ptr out, Scope& scope)
{
    InstructionPtr inst(new BinaryInstruction<L,R,O,Op>(lhs,rhs,out));
    scope.addInstruction(inst);
}

template <typename L, typename R, typename O, OpReturn (* Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
typename O::Ptr
addBinaryOp(typename L::Ptr lhs, typename R::Ptr rhs, Scope& scope)
{
    typename O::Ptr result(new O());
    addBinaryOpInstruction<L,R,O,Op>(lhs,rhs,result,scope);
    return result;
}

template <typename L, typename R, typename O, OpReturn (* Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
typename O::Ptr
addLhsArrayBinaryOp(typename L::Ptr lhs, typename R::Ptr rhs, Scope& scope)
{
    typename O::Ptr result(new O(lhs->size()));
    addBinaryOpInstruction<L,R,O,Op>(lhs,rhs,result,scope);
    return result;
```

```
49 }
50
51 template <typename L, typename R, typename O, OpReturn (* Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
52 typename O::Ptr
53 addRhsArrayBinaryOp(typename L::Ptr lhs, typename R::Ptr rhs, Scope& scope)
54 {
55     typename O::Ptr result(new O(rhs->size()));
56     addBinaryOpInstruction<L,R,O,Op>(lhs,rhs,result,scope);
57     return result;
58 }
59
60 template <typename L, typename R, typename O, OpReturn (* Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
61 typename O::Ptr
62 addArrayBinaryOp(typename L::Ptr lhs, typename R::Ptr rhs, Scope& scope)
63 {
64     typename O::Ptr result;
65     if (rhs->size() == lhs->size()) {
66         result = typename O::Ptr(new O(rhs->size()));
67         addBinaryOpInstruction<L,R,O,Op>(lhs,rhs,result,scope);
68     }
69     return result;
70 }
71
72
73 #endif /*INSTRUCTIONFACTORY_HPP_*/
```

10 IntervalSet.hpp

```
1  #ifndef INTERVALSET_HPP_
2  #define INTERVALSET_HPP_
3
4  #include <list>
5  #include <utility>
6
7  #include "boost/numeric/interval.hpp"
8  #include "IntervalSetIterator.hpp"
9
10 class IntervalSet
11 {
12 public:
13     typedef int Index;
14     typedef int Value;
15     typedef boost::numeric::interval<Value> Interval;
16     typedef IntervalSetConstIterator<Value> const_iterator;
17
18     IntervalSet();
19
20     void insert(const Value& v);
21     void insert(const Interval& interval);
22     void insert(std::pair<Value,Value> const& interval);
23
24     int cardinality() const;
25     Value valueOfIndex(const Index& index) const;
26     Index indexOfValue(const Value& value) const;
27     bool contains(Value const& value) const;
28
29     const_iterator begin() const;
30     const_iterator end() const;
31
32 private:
33     typedef std::list<Interval> IntervalList;
34     IntervalList _intervals;
35     int _cardinality;
36
37     void IntervalSet::updateCardinality();
38 };
39
40 #endif /*INTERVALSET_HPP_*/
```

11 IntervalSetIterator.hpp

```
1  #ifndef INTERVALSETITERATOR_HPP_
2  #define INTERVALSETITERATOR_HPP_
3
4  #include <list>
5
```

```
#include "boost/numeric/interval.hpp"
#include "boost/iterator/iterator_facade.hpp"

template <typename Value>
class IntervalSetConstIterator
  :  public boost::iterator_facade<
        IntervalSetConstIterator<Value>
      , Value
      , boost::forward_traversal_tag
    >
{
public:
    IntervalSetConstIterator(typename std::list<boost::numeric::interval<Value> >::const_iterator iterator)
     :  _i(iterator),
       _offset(0)
    {}

 private:
    friend class boost::iterator_core_access;

    void increment() {
        ++_offset;
        if (_offset > _i->upper() - _i->lower()) {
            ++_i;
            _offset = 0;
        }
    }

    Value& dereference() const
    {
        _v = _i->lower() + _offset;

        return _v;
    }

    bool equal(IntervalSetConstIterator const& other) const
    {
        return this->_i == other._i && this->_offset == other._offset;
    }

    typename std::list<boost::numeric::interval<Value> >::const_iterator _i;
    int _offset;
    mutable Value _v;
};

#endif /*INTERVALSETITERATOR_HPP_*/
```

12 IntervalSetVector.hpp

```
#ifndef INTERVALSETVECTOR_HPP_
#define INTERVALSETVECTOR_HPP_

#include <vector>

#include "IntervalSet.hpp"
#include "IntervalSetVectorIterator.hpp"

class IntervalSetVector
{
public:

    typedef IntervalSetVectorConstIterator const_iterator;

    IntervalSetVector();

    void addSet(const IntervalSet& set);

    IntervalSet& operator [](std::size_t i);
    const IntervalSet& operator [](std::size_t i) const;

    std::valarray<int> valueOfIndex(int i) const;
    int indexOfValue(std::valarray<int> const& v) const;

    const_iterator begin() const;
    const_iterator end() const;

private:
    typedef std::vector<IntervalSet> SetList;

```

```
    SetList _sets;
};

#endif /*INTERVALSETVECTOR_HPP_*/
```

13 IntervalSetVectorIterator.hpp

```
#ifndef INTERVALSETVECTORITERATOR_HPP_
#define INTERVALSETVECTORITERATOR_HPP_

#include <iostream>
#include <list>
#include <valarray>

#include "boost/iterator/iterator_facade.hpp"

class IntervalSetVectorConstIterator
  :  public boost::iterator_facade<
        IntervalSetVectorConstIterator
      , std::valarray<int>
      , boost::forward_traversal_tag
    >
{
public:
    typedef std::list<IntervalSet::const_iterator> IteratorList;
    typedef std::vector<IntervalSet> ElementList;

    IntervalSetVectorConstIterator(const ElementList& elements,
                                   IteratorList iterators)
     :  _elements(elements)
     , _iterators(iterators)
     , _v(elements.size())
    {
        int index = 0;
        for (IteratorList::iterator i = _iterators.begin(); i != _iterators.end(); ++i) {
            _v[index] = **i;
            ++index;
        }
    }

 private:

    friend class boost::iterator_core_access;

    void increment() {
        increment(_elements.begin(), _iterators.begin(), 0);
    }

    void increment(ElementList::const_iterator element, IteratorList::iterator iterator, int index) {
        if (element != _elements.end()) {
            ++(*iterator);
            if (*iterator == element->end()) {
                *iterator = element->begin();
                _v[index] = **iterator;
                increment(++element, ++iterator, ++index);
            }
            else {
                _v[index] = **iterator;
            }
        }
        else {
            // we have reached the end, make sure all iterators are also at the end
            // to match expected end() equality test
            ElementList::const_iterator e = _elements.begin();
            for (IteratorList::iterator i = _iterators.begin(); i != _iterators.end(); ++i) {
                *i = e->end();
                ++e;
            }
        }
    }

    std::valarray<int>& dereference() const
    {
        return _v;
    }

    bool equal(IntervalSetVectorConstIterator const& other) const
    {
        bool result =
```

```
73            &_elements == &other._elements &&
74            std::equal(_iterators.begin(),_iterators.end(),other._iterators.begin());
75
76        return result;
77    }
78
79    const ElementList& _elements;
80    IteratorList _iterators;
81    mutable std::valarray<int> _v;
82 };
83
84 #endif /*INTERVALSETVECTORITERATOR_HPP_*/
```

14 LineCountingAST.hpp

```
1  #ifndef LINECOUNTINGAST_HPP_
2  #define LINECOUNTINGAST_HPP_
3
4  #include "antlr/CommonAST.hpp"
5
6  class LineCountingAST : public antlr::CommonAST {
7  public:
8      typedef antlr::ASTRefCount<LineCountingAST> Ptr;
9
10    // copy constructor
11    LineCountingAST( const LineCountingAST& other )
12    :  CommonAST(other)
13    , line(other.line)
14    {
15    }
16
17    // Default constructor
18    LineCountingAST() :  CommonAST(), line(0) {}
19    virtual ~LineCountingAST() {}
20
21    // get the line number of the node (or try to derive it from the child node
22    virtual int getLine() const
23    {
24       // most of the time the line number is not set if the node is a
25       // imaginary one.  Usually this means it has a child.  Refer to the
26       // child line number.  Of course this could be extended a bit.
27       // based on an example by Peter Morling.
28       if ( line != 0 )
29          return line;
30       if( getFirstChild() )
31          return ( Ptr(getFirstChild())->getLine() );
32       return 0;
33    }
34
35    virtual void setLine( int l )
36    {
37       line = l;
38    }
39
40    /** the initialize methods are called by the tree building constructs
41     * depending on which version is called the line number is filled in.
42     * e.g.  a bit depending on how the node is constructed it will have the
43     * line number filled in or not (imaginary nodes!).
44     */
45    virtual void initialize(int t, std::string const& txt)
46    {
47       CommonAST::initialize(t,txt);
48       line = 0;
49    }
50
51    virtual void initialize( antlr::RefToken t )
52    {
53       CommonAST::initialize(t);
54       line = t->getLine();
55    }
56
57    virtual void initialize( Ptr ast )
58    {
59       CommonAST::initialize(antlr::RefAST(ast));
60       line = ast->getLine();
61    }
62
63    // for convenience will also work without
64    void addChild( Ptr c )
```

```
65    {
66       BaseAST::addChild( antlr::RefAST(c) );
67    }
68
69    // for convenience will also work without
70    void setNextSibling( Ptr c )
71    {
72       BaseAST::setNextSibling( antlr::RefAST(c) );
73    }
74
75    // provide a clone of the node (no sibling/child pointers are copied)
76    virtual antlr::RefAST clone()
77    {
78       return antlr::RefAST(new LineCountingAST(*this));
79    }
80
81    static antlr::RefAST factory()
82    {
83       return antlr::RefAST(Ptr(new LineCountingAST()));
84    }
85
86 private:
87    int line;
88 };
89
90 #endif /*LINECOUNTINGAST_HPP_*/
```

15 MarkovDecisionProcessType.hpp

```
1  #ifndef MARKOVDECISIONPROCESSTYPE_HPP_
2  #define MARKOVDECISIONPROCESSTYPE_HPP_
3
4  #include <list>
5
6  #include "boost/smart_ptr.hpp"
7  #include "Types.hpp"
8  #include "Scope.hpp"
9  #include "StateVector.hpp"
10 #include "IntervalSet.hpp"
11 #include "IntervalSetVector.hpp"
12 #include "Dispatcher.hpp"
13
14
15 class MarkovDecisionProcessType :  public RllType
16 {
17 public:
18
19     typedef boost::shared_ptr<MarkovDecisionProcessType> Ptr;
20     typedef std::list<RllType::Ptr> ArgumentList;
21
22     MarkovDecisionProcessType(Scope &parent, Dispatcher & dispatcher);
23
24     bool defineParameter(const std::string& name);
25     bool defineParameter(const std::string& name, RllType::Ptr defaultValue);
26     bool defineStateVariable(const std::string& name, int dimension, SetType::Ptr bounds);
27     bool defineActionVariable(const std::string& name, int dimension, SetType::Ptr bounds);
28     bool defineReward(const std::string& name);
29     bool defineRandomVariable(const std::string& name, RandomType::Ptr distribution);
30     bool defineRandomVariable(const std::string& name, std::list<RandomType::Ptr> & distributions);
31     void enterBodyDeclaration();
32     void enterEpisodeDeclaration();
33     void enterEnvironmentDeclaration();
34
35     bool printPomdp(std::ostream & out, ArgumentList & arguments);
36
37     RllType::Ptr getParameter(std::size_t position);
38     std::size_t  numParameters() const;
39
40     Scope & getScope();
41     Scope const& getScope() const;
42
43     Scope & getParameterScope();
44     Scope const& getParameterScope() const;
45
46     virtual ~MarkovDecisionProcessType();
47
48 private:
49
50     void initializePredicate(Scope & scope, std::string const& name, bool value);
```

```
51
52      DiscreteRandom::RandomGenerator _generator;
53      Dispatcher & _dispatcher;
54      Scope _parameterScope;
55      Scope _initialScope;
56      Scope _episodeScope;
57      Scope _environmentScope;
58      Scope * _currentScope;
59
60      typedef std::pair<RllType::Ptr,int> VectorIndex;
61      typedef std::vector<SetType::Ptr> Bounds;
62      typedef std::vector<VectorIndex> Indices;
63      typedef std::vector<RllType::Ptr> Parameters;
64      typedef std::vector<RandomType::Ptr> Randoms;
65
66      Bounds _stateBounds;
67      Indices _stateIndices;
68      Indices _finalStateIndices;
69
70      Bounds _actionBounds;
71      Indices  _actionIndices;
72      DoubleType::Ptr _reward;
73
74      void set(Indices & indices, std::valarray<int> const& values);
75      void get(Indices & indices, std::valarray<int> & values);
76
77      Parameters _parameters;
78
79      Randoms _episodeRandoms;
80      Randoms _environmentRandoms;
81  };
82
83  #endif /*MARKOVDECISIONPROCESSTYPE_HPP_*/
```

16 MultiMethods.hpp

```
1   #ifndef MULTIMETHODS_HPP_
2   #define MULTIMETHODS_HPP_
3
4   // Adapted from Loki
5
6   #include "boost/smart_ptr.hpp"
7   #include "loki/Typelist.h"
8   #include "loki/LokiTypeInfo.h"
9   #include "loki/Functor.h"
10  #include "loki/AssocVector.h"
11
12  #include <sstream>
13  #include "boost/smart_ptr.hpp"
14  #include "Types.hpp"
15  #include "Scope.hpp"
16  #include "InstructionFactory.hpp"
17
18
19  template <typename ResultType, template <typename> class Pointer = boost::shared_ptr>
20  class BasicDispatcher
21  {
22      typedef Pointer<RllType> BaseLhs;
23      typedef Pointer<RllType> BaseRhs;
24      typedef ResultType ( * CallbackType)(BaseLhs, BaseRhs, Scope &);
25
26      typedef std::pair<Loki::TypeInfo,Loki::TypeInfo> KeyType;
27      typedef CallbackType MappedType;
28      typedef Loki::AssocVector<KeyType, MappedType> MapType;
29      MapType callbackMap_;
30
31      void doAdd(Loki::TypeInfo lhs, Loki::TypeInfo rhs, CallbackType fun);
32      bool doRemove(Loki::TypeInfo lhs, Loki::TypeInfo rhs);
33
34  public:
35      template <class SomeLhs, class SomeRhs>
36      void add(CallbackType fun)
37      {
38          doAdd(typeid(typename SomeLhs::element_type), typeid(typename SomeRhs::element_type), fun);
39      }
40
41      template <class SomeLhs, class SomeRhs>
42      bool remove()
43      {
```

```
44          return doRemove(typeid(SomeLhs::element_type), typeid(SomeRhs::element_type));
45      }
46
47      ResultType go(BaseLhs lhs, BaseRhs rhs, Scope &scope);
48  };
49
50  template <typename ResultType, template <typename> class Pointer>
51  void
52  BasicDispatcher<ResultType,Pointer>::doAdd(Loki::TypeInfo lhs, Loki::TypeInfo rhs, CallbackType fun)
53  {
54      callbackMap_[KeyType(lhs, rhs)] = fun;
55  }
56
57  template <typename ResultType, template <typename> class Pointer>
58  bool
59  BasicDispatcher<ResultType,Pointer>::doRemove(Loki::TypeInfo lhs, Loki::TypeInfo rhs)
60  {
61      return callbackMap_.erase(KeyType(lhs, rhs)) == 1;
62  }
63
64  template <typename ResultType, template <typename> class  Pointer>
65  ResultType
66  BasicDispatcher<ResultType,Pointer>::go(BaseLhs lhs,BaseRhs rhs,Scope &scope)
67  {
68      typename MapType::key_type k(typeid(*lhs),typeid(*rhs));
69      typename MapType::iterator i = callbackMap_.find(k);
70
71      if (i != callbackMap_.end())
72      {
73          return (i->second)(lhs, rhs,scope);
74      }
75      return ResultType();
76  }
77
78  ////////////////////////////////////////////////////////////////////////////////
79  // class template Private::FnDispatcherHelper
80  // Implements trampolines and argument swapping used by FnDispatcher
81  ////////////////////////////////////////////////////////////////////////////////
82
83  namespace Private
84  {
85      template <
86      class SomeLhs, class SomeRhs,
87          typename SomeResult,
88          typename BaseResult,
89          template <typename> class  Pointer,
90          SomeResult (*Callback)(Pointer<SomeLhs>, Pointer<SomeRhs>,Scope &)>
91      struct FnDispatcherHelper
92      {
93          typedef  Pointer<RllType> BaseLhs;
94          typedef  Pointer<RllType> BaseRhs;
95          static BaseResult Trampoline(BaseLhs lhs, BaseRhs rhs,Scope &scope)
96          {
97              return Callback(boost::dynamic_pointer_cast<SomeLhs>(lhs), boost::dynamic_pointer_cast<SomeRhs>(rhs),scope);
98          }
99          static BaseResult TrampolineR(BaseRhs rhs, BaseLhs lhs,Scope &scope)
100         {
101             return Trampoline(lhs, rhs,scope);
102         }
103     };
104
105 //    template <typename L, typename R, typename O,  template <typename> class  Pointer, void (*Op)(typename L::Ref,typename
R::Ref,typename O::Ref)>
106 //    struct BinaryOpHelper
107 //    {
108 //        typedef  Pointer<RllType> BaseLhs;
109 //        typedef  Pointer<RllType> BaseRhs;
110 //        typedef  Pointer<RllType> ResultType;
111 //        static ResultType Trampoline(BaseLhs lhs, BaseRhs rhs,Scope &scope)
112 //        {
113 //            return addBinaryOp<L,R,O,Op>(boost::dynamic_pointer_cast<L>(lhs), boost::dynamic_pointer_cast<R>(rhs),scope);
114 //        }
115 //        static ResultType TrampolineR(BaseRhs rhs, BaseLhs lhs,Scope &scope)
116 //        {
117 //            return Trampoline(lhs, rhs,scope);
118 //        }
119 //    };
120 }
121
122 ////////////////////////////////////////////////////////////////////////////////
123 // class template FnDispatcher
```

```
124 // Implements an automatic logarithmic double dispatcher for functions
125 // Features automated conversions
126 ////////////////////////////////////////////////////////////////////////////////
127
128 template <typename ResultType, template <typename> class Pointer = boost::shared_ptr>
129 class FnDispatcher
130 {
131     typedef  Pointer<RllType> BaseLhs;
132     typedef  Pointer<RllType> BaseRhs;
133     typedef ResultType ( * CallbackType)(BaseLhs, BaseRhs,Scope &);
134
135     BasicDispatcher<ResultType,Pointer> backEnd_;
136
137 public:
138     template <class SomeLhs, class SomeRhs>
139     void add(ResultType (*pFun)(BaseLhs, BaseRhs,Scope &))
140     {
141         return backEnd_.template add<Pointer<SomeLhs>, Pointer<SomeRhs> >(pFun);
142     }
143
144     template <class SomeIn, class SomeOut,
145         void (*callback)(Pointer<SomeIn>, Pointer<SomeOut>,Scope &)>
146     void add()
147     {
148     typedef Private::FnDispatcherHelper<
149                 SomeIn, SomeOut,
150                 void,
151                 void,
152                 Pointer,
153                 callback> Local;
154
155         add<SomeIn, SomeOut>(&Local::Trampoline);
156     }
157
158
159     template <class SomeLhs, class SomeRhs, class SomeResult,
160         Pointer<SomeResult> (*callback)(Pointer<SomeLhs>, Pointer<SomeRhs>,Scope &)>
161     void add()
162     {
163     typedef Private::FnDispatcherHelper<
164                 SomeLhs, SomeRhs,
165                 Pointer<SomeResult>,
166                 ResultType,
167                 Pointer,
168                 callback> Local;
169
170         add<SomeLhs, SomeRhs>(&Local::Trampoline);
171     }
172
173 //    template <typename L, typename R, typename O, void (*Op)(typename L::Ref,typename R::Ref,typename O::Ref)>
174 //    void add()
175 //    {
176 //    typedef Private::BinaryOpHelper<
177 //                L, R,
178 //                O,
179 //                Pointer,
180 //                Op > Local;
181 //
182 //        add<L, R>(&Local::Trampoline);
183 //    }
184
185     template <class SomeLhs, class SomeRhs, class SomeResult,
186         Pointer<SomeResult> (*callback)(Pointer<SomeLhs>, Pointer<SomeRhs>,Scope &),
187         bool symmetric>
188     void add(bool = true) // [gcc] dummy bool
189     {
190     typedef Private::FnDispatcherHelper<
191                 SomeLhs, SomeRhs,
192                 Pointer<SomeResult>,
193                 ResultType,
194                 Pointer,
195                 callback> Local;
196
197         add<SomeLhs, SomeRhs>(&Local::Trampoline);
198         if (symmetric)
199         {
200             add<SomeRhs, SomeLhs>(&Local::TrampolineR);
201         }
202     }
203
204     template <class SomeLhs, class SomeRhs>
```

```
205     void remove()
206     {
207         backEnd_.template remove<Pointer<SomeLhs>, Pointer<SomeRhs> >();
208     }
209 
210     ResultType go(BaseLhs lhs, BaseRhs rhs,Scope &scope)
211     {
212         return backEnd_.go(lhs, rhs,scope);
213     }
214 
215 };
216 
217 #endif /*MULTIMETHODS_HPP_*/
```

17 PoissonRandom.hpp

```
1  #ifndef POISSONRANDOM_HPP_
2  #define POISSONRANDOM_HPP_
3  
4  #include <vector>
5  
6  #include "DiscreteRandom.hpp"
7  
8  
9  class PoissonRandom :  public DiscreteRandom
10 {
11 public:
12     typedef boost::uniform_real<> Distribution;
13     typedef boost::variate_generator<RandomGenerator&,Distribution> Generator;
14 
15     PoissonRandom(RandomGenerator& generator, unsigned expected);
16 
17     virtual int operator() ();
18 
19     virtual double probability(int i) const;
20 
21     virtual ~PoissonRandom();
22 
23 private:
24     typedef std::vector<double> Probabilities;
25     Generator _generator;
26     Probabilities _probabilities;
27 };
28 
29 #endif /*POISSONRANDOM_HPP_*/
```

18 random.hpp

```
1 #ifndef RANDOM_HPP_
2 #define RANDOM_HPP_
3 
4 #include "DiscreteRandom.hpp"
5 #include "BinomialRandom.hpp"
6 #include "UniformRandom.hpp"
7 #include "PoissonRandom.hpp"
8 
9 #endif /*RANDOM_HPP_*/
```

19 RandomVector.hpp

```
1  #ifndef RANDOMVECTOR_HPP_
2  #define RANDOMVECTOR_HPP_
3  
4  #include <valarray>
5  
6  #include "boost/smart_ptr.hpp"
7  #include "DiscreteRandom.hpp"
8  #include "RandomVectorIterator.hpp"
9  
10 class RandomVector
11 {
12 public:
13     typedef boost::shared_ptr<DiscreteRandom> RandomPtr;
14     typedef RandomVectorConstIterator const_iterator;
```

```
15
16      RandomVector();
17
18      void addRandomVariable(RandomPtr x);
19
20      std::valarray<int> operator ()() const;
21
22      double probability(const std::valarray<int>& x) const;
23
24      const_iterator begin() const;
25      const_iterator end() const;
26
27  private:
28      typedef std::list<RandomPtr> VariableList;
29
30      VariableList _variables;
31  };
32
33  #endif /*RANDOMVECTOR_HPP_*/
```

20 RandomVectorIterator.hpp

```
1   #ifndef RANDOMVECTORITERATOR_HPP_
2   #define RANDOMVECTORITERATOR_HPP_
3
4   #include <iostream>
5   #include <list>
6   #include <valarray>
7
8   #include "boost/smart_ptr.hpp"
9   #include "boost/iterator/iterator_facade.hpp"
10  #include "DiscreteRandom.hpp"
11
12  class RandomVectorConstIterator
13    :  public boost::iterator_facade<
14          RandomVectorConstIterator
15        , std::valarray<int>
16        , boost::forward_traversal_tag
17      >
18  {
19  public:
20      typedef boost::shared_ptr<DiscreteRandom> RandomPtr;
21      typedef std::list<DiscreteRandom::const_iterator> IteratorList;
22      typedef std::list<RandomPtr> ElementList;
23
24      RandomVectorConstIterator(const ElementList& elements,
25                                IteratorList iterators)
26       :  _elements(elements)
27       , _iterators(iterators)
28       , _v(elements.size())
29      {
30          int index = 0;
31          for (IteratorList::iterator i = _iterators.begin(); i != _iterators.end(); ++i) {
32              _v[index] = **i;
33              ++index;
34          }
35      }
36
37   private:
38
39      friend class boost::iterator_core_access;
40
41      void increment() {
42          increment(_elements.begin(), _iterators.begin(), 0);
43      }
44
45      void increment(ElementList::const_iterator element, IteratorList::iterator iterator, int index) {
46          if (element != _elements.end()) {
47              ++(*iterator);
48              if (*iterator == (*element)->end()) {
49                  *iterator = (*element)->begin();
50                  _v[index] = **iterator;
51                  increment(++element, ++iterator, ++index);
52              }
53              else {
54                  _v[index] = **iterator;
55              }
56          }
57          else {
```

```
58              // we have reached the end, make sure all iterators are also at the end
59              // to match expected end() equality test
60              ElementList::const_iterator e = _elements.begin();
61              for (IteratorList::iterator i = _iterators.begin(); i != _iterators.end(); ++i) {
62                  *i = (*e)->end();
63                  ++e;
64              }
65          }
66      }
67
68      std::valarray<int>& dereference() const
69      {
70          return _v;
71      }
72
73      bool equal(RandomVectorConstIterator const& other) const
74      {
75          bool result =
76              &_elements == &other._elements &&
77              std::equal(_iterators.begin(),_iterators.end(),other._iterators.begin());
78
79          return result;
80      }
81
82      const ElementList& _elements;
83      IteratorList _iterators;
84      mutable std::valarray<int> _v;
85  };
86
87  #endif /*RANDOMVECTORITERATOR_HPP_*/
```

21 RllTokenStream.hpp

```
1  #ifndef RLL_TOKEN_STREAM_HPP
2  #define RLL_TOKEN_STREAM_HPP
3
4  /*
5  [The "BSD licence"]
6  Copyright (c) 2004 Terence Parr and Loring Craymer
7  All rights reserved.
8
9  Redistribution and use in source and binary forms, with or without
10 modification, are permitted provided that the following conditions
11 are met:
12 1.  Redistributions of source code must retain the above copyright
13     notice, this list of conditions and the following disclaimer.
14 2.  Redistributions in binary form must reproduce the above copyright
15     notice, this list of conditions and the following disclaimer in the
16     documentation and/or other materials provided with the distribution.
17 3.  The name of the author may not be used to endorse or promote products
18     derived from this software without specific prior written permission.
19
20 THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
21 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
22 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
23 IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
24 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
25 NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
26 DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
27 THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
28 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
29 THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30 */
31
32 #include <queue>
33 #include "antlr/TokenStream.hpp"
34 #include "RllLexer.hpp"
35
36
37 class RllTokenStream :  public  antlr::TokenStream {
38
39 public:
40     RllTokenStream(RllLexer& lexer);
41     virtual antlr::RefToken nextToken();
42     virtual ~RllTokenStream();
43
44 protected:
45     void insertImaginaryIndentDedentTokens();
46     void push(int i);
```

```
47      int pop();
48      int peek() const;
49      /** Return the index on stack of previous indent level == i else -1 */
50      int findPreviousIndent(int i) const;
51
52  private:
53      const static int MAX_INDENTS = 100;
54      const static int FIRST_COLUMN = 1;
55
56      /** The stack of indent levels (column numbers) */
57      int* _indentStack;
58      /** stack pointer */
59      int _sp; // grow upwards
60
61      /** The queue of tokens */
62      std::queue<antlr::RefToken> _tokens;
63
64      /** We pull real tokens from this lexer */
65      RllLexer& _lexer;
66  };
67
68  #endif
```

22 RllTreeWalker.hpp

```
1   #ifndef RLLTREEWALKER_HPP_
2   #define RLLTREEWALKER_HPP_
3
4   #include <string>
5
6   #include "LineCountingAST.hpp"
7   #include "RllLexerTokenTypes.hpp"
8   #include "Types.hpp"
9   #include "types/MarkovDecisionProcessType.hpp"
10  #include "Scope.hpp"
11  #include "Dispatcher.hpp"
12
13  class RllTreeWalker :  public RllLexerTokenTypes
14  {
15  public:
16      typedef LineCountingAST::Ptr AST;
17
18      RllTreeWalker(Dispatcher & dispatcher);
19
20      virtual RllType::Ptr fileInput(AST ast, Scope &scope);
21      virtual RllType::Ptr mdp(AST ast, Scope &scope);
22      virtual void setMdpParameters(AST ast, MarkovDecisionProcessType::Ptr mdp);
23      virtual void fillMdpBody(AST ast, MarkovDecisionProcessType::Ptr mdp);
24      virtual void addMdpDiagram(AST ast, MarkovDecisionProcessType::Ptr mdp);
25      virtual void addMdpStates(AST ast, MarkovDecisionProcessType::Ptr mdp);
26      virtual void addMdpActions(AST ast, MarkovDecisionProcessType::Ptr mdp);
27      virtual void addMdpReward(AST ast, MarkovDecisionProcessType::Ptr mdp);
28      virtual void addMdpRandom(AST ast, MarkovDecisionProcessType::Ptr mdp);
29      virtual void addMdpConstraint(AST ast, MarkovDecisionProcessType::Ptr mdp);
30      virtual void addMdpEpisode(AST ast, MarkovDecisionProcessType::Ptr mdp);
31      virtual void addMdpEnvironment(AST ast, MarkovDecisionProcessType::Ptr mdp);
32
33      virtual RllType::Ptr expression(AST ast, Scope & scope);
34      virtual RllType::Ptr functionCall(AST ast, Scope & scope);
35      virtual bool assign(AST ast, Scope & scope);
36
37      virtual RllType::Ptr ternary(AST test, AST ifTrue, AST ifFalse, Scope & scope);
38      virtual ~RllTreeWalker();
39
40  private:
41
42      RllType::Ptr dispatchOps(AST ast, Scope & scope);
43      RllType::Ptr dispatchOps(AST ast, int type, Scope & scope);
44
45      void error(std::string const& file, int fileLine, std::string const& message, int sourceLine, Scope & scope);
46      bool expected(std::string const& file, int fileLine, AST ast, int expected,std::string const& expectedString, Scope &
scope);
47
48      template <typename Type>
49      typename Type::Ptr
50      expectedType(std::string const& file, int fileLine, AST ast, RllType::Ptr ptr, Scope & scope);
51
52      Dispatcher & _dispatcher;
53  };
```

```
54
55 #endif /*RLLTREEWALKER_HPP_*/
```

23 Scope.hpp

```
1  #ifndef SCOPE_HPP_
2  #define SCOPE_HPP_
3
4  #include <string>
5  #include <map>
6  #include <list>
7  #include "boost/smart_ptr.hpp"
8  #include "Instruction.hpp"
9  #include "SemanticError.hpp"
10
11 class RllType;
12 typedef boost::shared_ptr<RllType> TypePtr;
13
14 class Scope
15 {
16 public:
17     typedef std::list<SemanticError> ErrorList;
18
19     Scope(std::string const& name = "");
20
21     virtual TypePtr getSymbol(std::string const& s);
22     //virtual void addTemporary(TypePtr temp);
23     virtual bool addSymbol(std::string const& name, TypePtr symbol, bool okToEclipse = false);
24     virtual bool overwriteSymbol(std::string const& name, TypePtr symbol);
25     virtual void addInstruction(InstructionPtr instruction);
26     virtual void evaluateInstructions() const;
27     virtual Scope makeChildScope(std::string const& name = "");
28     virtual void reportError(std::string const& implementationFile, int implementationLine, std::string const& message, int
sourceLine);
29     virtual bool hasErrors();
30     virtual ErrorList const& getErrors();
31     virtual ~Scope();
32     virtual std::string const& getName() const;
33
34 private:
35     typedef std::map<std::string,TypePtr> NamedTypeMap;
36     typedef std::list<TypePtr> RegisterList;
37
38     Scope(Scope* parent, std::string const& name = "");
39     void reportError(std::string const& implementationFile, int implementationLine, std::string const& message, std::string
const& scope, int sourceLine);
40
41     Scope* _parent;
42     std::string _name;
43     NamedTypeMap _symbols;
44     NamedTypeMap _literals;
45 //    RegisterList _registers;
46     InstructionBlock _instructions;
47     ErrorList _errors;
48 };
49
50 #endif /*SCOPE_HPP_*/
```

24 SemanticError.hpp

```
1  #ifndef SEMANTICERROR_HPP_
2  #define SEMANTICERROR_HPP_
3
4  #include <string>
5  #include <ostream>
6
7  class SemanticError
8  {
9  public:
10     SemanticError(std::string const& implementationFile, int implementationLine,  std::string const& message, std::string
const& scope, int sourceLine);
11
12     std::string const& getImplementationFile() const;
13     int getImplementationLine() const;
14     std::string const& getScope() const;
15     std::string const& getMessage() const;
```

```
16      int getSourceLine() const;
17
18  private:
19      const std::string _implementationFile;
20      const int         _implementationLine;
21      const std::string _message;
22      const std::string _scope;
23      const int         _sourceLine;
24  };
25
26  std::ostream & operator<<(std::ostream & s, SemanticError const& error);
27
28  #endif /*SEMANTICERROR_HPP_*/
```

25 SingleMethods.hpp

```
1   #ifndef SINGLEMETHODS_HPP_
2   #define SINGLEMETHODS_HPP_
3
4   // Adapted from Loki
5
6   #include "boost/smart_ptr.hpp"
7   #include "loki/Typelist.h"
8   #include "loki/LokiTypeInfo.h"
9   #include "loki/Functor.h"
10  #include "loki/AssocVector.h"
11
12  #include <sstream>
13  #include "boost/smart_ptr.hpp"
14  #include "Types.hpp"
15  #include "Scope.hpp"
16  #include "InstructionFactory.hpp"
17
18
19
20  template <template <typename> class Pointer = boost::shared_ptr>
21  class BaseUnaryDispatcher
22  {
23      typedef Pointer<RllType> BaseIn;
24      typedef Pointer<RllType> ResultType;
25      typedef ResultType (* CallbackType)(BaseIn, Scope &);
26
27      typedef Loki::TypeInfo KeyType;
28      typedef CallbackType MappedType;
29      typedef Loki::AssocVector<KeyType, MappedType> MapType;
30      MapType callbackMap_;
31
32      void doAdd(Loki::TypeInfo in, CallbackType fun);
33      bool doRemove(Loki::TypeInfo in);
34
35  public:
36      template <class SomeIn>
37      void add(CallbackType fun)
38      {
39          doAdd(typeid(typename SomeIn::element_type), fun);
40      }
41
42      template <class SomeIn>
43      bool remove()
44      {
45          return doRemove(typeid(SomeIn::element_type));
46      }
47
48      ResultType go(BaseIn in, Scope &scope);
49  };
50
51  template <template <typename> class Pointer>
52  void
53  BaseUnaryDispatcher<Pointer>::doAdd(Loki::TypeInfo in, CallbackType fun)
54  {
55      callbackMap_[in] = fun;
56  }
57
58  template <template <typename> class Pointer>
59  bool
60  BaseUnaryDispatcher<Pointer>::doRemove(Loki::TypeInfo in)
61  {
62      return callbackMap_.erase(in) == 1;
63  }
```

```
64
65 template <template <typename> class  Pointer>
66 typename BaseUnaryDispatcher<Pointer>::ResultType
67 BaseUnaryDispatcher<Pointer>::go(BaseIn in,Scope &scope)
68 {
69     typename MapType::key_type k(typeid(*in));
70     typename MapType::iterator i = callbackMap_.find(k);
71     if (i == callbackMap_.end())
72     {
73         std::ostringstream stream;
74         stream << "Function not found "<<typeid(*in).name();
75             throw std::runtime_error(stream.str());
76     }
77     return (i->second)(in,scope);
78 }
79
80 ////////////////////////////////////////////////////////////////////////////////
81 // class template Private::FnDispatcherHelper
82 // Implements trampolines and argument swapping used by FnDispatcher
83 ////////////////////////////////////////////////////////////////////////////////
84
85 namespace Private
86 {
87     template <
88     class SomeIn,
89         typename SomeResult,
90         template <typename> class  Pointer,
91         Pointer<SomeResult> (*Callback)(Pointer<SomeIn>,Scope &)>
92     struct UnaryDispatcherHelper
93     {
94         typedef  Pointer<RllType> BaseIn;
95         static Pointer<RllType> Trampoline(BaseIn in, Scope &scope)
96         {
97             return Callback(boost::dynamic_pointer_cast<SomeIn>(in),scope);
98         }
99     };
100 }
101
102 ////////////////////////////////////////////////////////////////////////////////
103 // class template FnDispatcher
104 // Implements an automatic logarithmic double dispatcher for functions
105 // Features automated conversions
106 ////////////////////////////////////////////////////////////////////////////////
107
108 template <template <typename> class Pointer = boost::shared_ptr>
109 class UnaryDispatcher
110 {
111     typedef  Pointer<RllType> BaseIn;
112     typedef  Pointer<RllType> ResultType;
113     typedef ResultType ( * CallbackType)(BaseIn,Scope &);
114
115     BaseUnaryDispatcher<Pointer> backEnd_;
116
117 public:
118     template <class SomeIn>
119     void add(ResultType (*pFun)(BaseIn,Scope &))
120     {
121         return backEnd_.template add<Pointer<SomeIn> >(pFun);
122     }
123
124     template <class SomeIn, class SomeResult,
125         Pointer<SomeResult> (*callback)(Pointer<SomeIn>, Scope &)>
126     void add()
127     {
128     typedef Private::UnaryDispatcherHelper<
129                 SomeIn,
130                 SomeResult,
131                 Pointer,
132                 callback> Local;
133
134         add<SomeIn>(&Local::Trampoline);
135     }
136
137     template <class SomeIn>
138     void remove()
139     {
140         backEnd_.template remove<Pointer<SomeIn> >();
141     }
142
143     ResultType go(BaseIn in,Scope &scope)
144     {
```

```
145         return backEnd_.go(in,scope);
146     }
147
148 };
149
150 #endif /*SINGLEMETHODS_HPP_*/
```

26 StateVector.hpp

```
1 #ifndef STATEVECTOR_HPP_
2 #define STATEVECTOR_HPP_
3
4 #include <valarray>
5
6 typedef std::valarray<int> StateVector;
7
8 #endif /*STATEVECTOR_HPP_*/
```

27 Types.hpp

```
1 #ifndef TYPES_HPP_
2 #define TYPES_HPP_
3
4 #include <string>
5 #include <sstream>
6 #include <valarray>
7 #include "boost/smart_ptr.hpp"
8 #include "Scope.hpp"
9 #include "IntervalSet.hpp"
10 #include "Diagram.hpp"
11 #include "random.hpp"
12
13 class RllType
14 {
15 public:
16     typedef boost::shared_ptr<RllType> Ptr;
17
18     RllType(bool isConstant=false);
19     bool isConstant() const;
20
21     void setName(std::string const& name);
22     std::string const& name() const;
23
24     virtual Ptr newSameType() const;
25
26     virtual ~RllType();
27
28 protected:
29     void makeConstant();
30
31 private:
32     static int _instances;
33     bool _isConstant;
34     std::string _name;
35 };
36
37 template <typename Value,int Unique=0>
38 class WrapperType :  public RllType
39 {
40 public:
41     typedef boost::shared_ptr<WrapperType<Value,Unique> > Ptr;
42     typedef Value& Ref;
43
44     WrapperType()
45       :  RllType(false)
46       , _value()
47     {}
48
49     WrapperType(Value const& constantValue)
50       :  RllType(true)
51       , _value(constantValue)
52     {}
53
54     WrapperType(std::string const& constantString)
55       :  RllType(true)
56       , _value()
```

```
57      {
58          std::istringstream buffer(constantString);
59          buffer >> _value;
60          if (!buffer.eof()) {
61              throw std::invalid_argument("can't initialize "+ std::string(typeid(Value).name()) + " with "+ constantString);
62          }
63      }
64
65      void setConstant(Value const& value)
66      {
67          makeConstant();
68          _value = value;
69      }
70
71      void set(Value const& value)
72      {
73          _value = value;
74      }
75
76      Value get() const
77      {
78          return _value;
79      }
80
81      Value& ref()
82      {
83          return _value;
84      }
85
86      virtual RllType::Ptr newSameType() const {
87          return Ptr(new WrapperType<Value,Unique>());
88      }
89
90      virtual ~WrapperType() {}
91
92  private:
93      Value _value;
94  };
95
96  template <typename Value,int Unique=0>
97  class ArrayWrapperType :  public RllType
98  {
99  public:
100     typedef boost::shared_ptr<ArrayWrapperType<Value,Unique> > Ptr;
101     typedef std::valarray<Value>& Ref;
102     typedef Value& ElementRef;
103
104     ArrayWrapperType(std::size_t n)
105       :  RllType(false)
106       , _array(n)
107     {}
108     ArrayWrapperType(Value const& constantValue, std::size_t n)
109       :  RllType(true)
110       , _array(constantValue,n)
111     {}
112
113     void setConstant()
114     {
115         makeConstant();
116     }
117
118     std::size_t size() const
119     {
120         return _array.size();
121     }
122
123     void set(std::size_t i, Value const& value)
124     {
125         _array[i] = value;
126     }
127
128     Value get(std::size_t i) const
129     {
130         return _array[i];
131     }
132
133     Ref ref()
134     {
135         return _array;
136     }
137
```

```
138     virtual RllType::Ptr newSameType() const {
139         return Ptr(new ArrayWrapperType<Value,Unique>(_array.size()));
140     }
141 
142     virtual ~ArrayWrapperType() {}
143 
144 private:
145 
146     std::valarray<Value> _array;
147 };
148 
149 
150 class FunctionType :  public RllType
151 {
152 public:
153     typedef boost::shared_ptr<FunctionType> Ptr;
154 
155     FunctionType(Scope& parent);
156 
157     bool addParameter(std::string const& name, RllType::Ptr type);
158     bool setReturn(RllType::Ptr type);
159 
160     RllType::Ptr getParameter(std::size_t position);
161     std::size_t  numParameters() const;
162     RllType::Ptr getReturn();
163 
164     Scope& getScope();
165     void execute() const;
166 
167     virtual RllType::Ptr newSameType() const;
168 
169     virtual ~FunctionType();
170 
171 private:
172     typedef std::vector<RllType::Ptr> Parameters;
173     Scope& _parent;
174     Scope _scope;
175     RllType::Ptr _return;
176     Parameters _parameters;
177 };
178 
179 
180 
181 typedef WrapperType<bool> BooleanType;
182 typedef WrapperType<int> IntegerType;
183 typedef WrapperType<double> DoubleType;
184 //typedef WrapperType<std::string> StringType;
185 typedef WrapperType<Diagram> DiagramType;
186 typedef ArrayWrapperType<bool> BooleanArrayType;
187 typedef ArrayWrapperType<int> IntegerArrayType;
188 typedef ArrayWrapperType<double> DoubleArrayType;
189 typedef WrapperType<IntervalSet> SetType;
190 typedef WrapperType<IntervalSet::Interval> IntervalType;
191 
192 
193 class TernaryType :  public RllType
194 {
195 public:
196     typedef boost::shared_ptr<TernaryType> Ptr;
197 
198     TernaryType(Scope& parent);
199 
200     bool setReturn(RllType::Ptr type);
201     RllType::Ptr getReturn();
202     BooleanType::Ptr getCondition();
203 
204     Scope& getTrueScope();
205     Scope& getFalseScope();
206 
207     void execute() const;
208 
209     virtual RllType::Ptr newSameType() const;
210 
211     virtual ~TernaryType();
212 
213 private:
214     Scope& _parent;
215     Scope _trueScope;
216     Scope _falseScope;
217     RllType::Ptr _return;
218     BooleanType::Ptr _condition;
```

```
219 };
220
221 class RandomType :  public RllType
222 {
223 public:
224     typedef boost::shared_ptr<RandomType> Ptr;
225
226     RandomType();
227
228     IntegerType::Ptr getReturn();
229
230     virtual void create(DiscreteRandom::RandomGenerator & generator);
231     virtual void draw();
232
233     virtual ~RandomType();
234     virtual boost::shared_ptr<DiscreteRandom> distribution() = 0;
235
236 protected:
237     IntegerType::Ptr _return;
238 };
239
240
241 class BinomialRandomType :  public RandomType
242 {
243 public:
244     typedef boost::shared_ptr<BinomialRandomType> Ptr;
245
246     BinomialRandomType(DoubleType::Ptr p);
247     BooleanType::Ptr getBoolReturn();
248     virtual void create(DiscreteRandom::RandomGenerator & generator);
249     virtual void draw();
250     virtual boost::shared_ptr<DiscreteRandom> distribution();
251
252     virtual ~BinomialRandomType();
253
254 private:
255     DoubleType::Ptr _p;
256     BooleanType::Ptr _boolResult;
257     boost::shared_ptr<BinomialRandom> _binomial;
258 };
259
260 class UniformRandomType :  public RandomType
261 {
262 public:
263     typedef boost::shared_ptr<UniformRandomType> Ptr;
264
265     UniformRandomType(IntegerType::Ptr min, IntegerType::Ptr max);
266
267     virtual void create(DiscreteRandom::RandomGenerator & generator);
268     virtual void draw();
269     virtual boost::shared_ptr<DiscreteRandom> distribution();
270
271     virtual ~UniformRandomType();
272
273 private:
274     IntegerType::Ptr _min;
275     IntegerType::Ptr _max;
276     boost::shared_ptr<UniformRandom> _uniform;
277 };
278
279 class PoissonRandomType :  public RandomType
280 {
281 public:
282     typedef boost::shared_ptr<PoissonRandomType> Ptr;
283
284     PoissonRandomType(IntegerType::Ptr expected);
285
286     virtual void create(DiscreteRandom::RandomGenerator & generator);
287     virtual void draw();
288     virtual boost::shared_ptr<DiscreteRandom> distribution();
289
290     virtual ~PoissonRandomType();
291
292 private:
293     IntegerType::Ptr _expected;
294     boost::shared_ptr<PoissonRandom> _poisson;
295 };
296
297 #endif /*TYPES_HPP_*/
```

28 UniformRandom.hpp

```cpp
#ifndef UNIFORMRANDOM_HPP_
#define UNIFORMRANDOM_HPP_

#include "DiscreteRandom.hpp"


class UniformRandom :  public DiscreteRandom
{
public:
    typedef boost::uniform_int<> UniformDistribution;
    typedef boost::variate_generator<RandomGenerator&,UniformDistribution> UniformGenerator;

    UniformRandom(RandomGenerator& generator, int min, int max);

    virtual int operator() ();

    virtual double probability(int i) const;

    virtual ~UniformRandom();

private:
    UniformGenerator _uniformGenerator;
};

#endif /*UNIFORMRANDOM_HPP_*/
```

29 util.hpp

```cpp
#ifndef UTIL_HPP_
#define UTIL_HPP_

#include <algorithm>

#include "boost/numeric/interval.hpp"

template<class T> T
limit(T lower, T value, T upper) {
    return std::min(upper,std::max(lower,value));
}

template<typename Map>
typename Map::iterator
findOrCreate(Map& m, const typename Map::key_type& k) {
    typename Map::iterator lower = m.lower_bound(k);

    if (lower != m.end() &&
        !(m.key_comp()(k,lower->first))) {
        return lower;
    }
    else {
        return m.insert(lower, typename Map::value_type(k, typename Map::mapped_type()));
    }
}

template<typename Map, typename Key, typename Value>
bool
insertIfUnique(Map& m, Key k, Value v) {
    typename Map::iterator lower = m.lower_bound(k);

    if (lower != m.end() &&
        !(m.key_comp()(k,lower->first))) {
        return false;
    }
    else {
        m.insert(lower, typename Map::value_type(k, v));
        return true;
    }
}

template<typename Map, typename Key, typename Value>
typename Map::iterator
insertOrUpdate(Map& m, Key k, Value v) {
    typename Map::iterator lower = m.lower_bound(k);

    if (lower != m.end() &&
        !(m.key_comp()(k,lower->first))) {
        lower->second = v;
```

```
50          return lower;
51      }
52      else {
53          return m.insert(lower, typename Map::value_type(k, v));
54      }
55  }
56
57  #endif /*UTIL_HPP_*/
```

30 BinomialRandom.cpp

```
1   #include "BinomialRandom.hpp"
2
3   BinomialRandom::BinomialRandom(RandomGenerator& generator, double p)
4     :  DiscreteRandom(generator,IntervalSet::Interval(0,1))
5     , _generator(_numberGenerator,Distribution(p))
6   {
7   }
8
9   int
10  BinomialRandom::operator()()
11  {
12      return _generator();
13  }
14
15  double
16  BinomialRandom::probability(int i) const
17  {
18      double probability = 0.0;
19      if (i == 0) {
20          probability = 1.0 - _generator.distribution().p();
21      }
22      else if (i == 1) {
23          probability = _generator.distribution().p();
24      }
25
26      return probability;
27  }
28
29
30  BinomialRandom::~BinomialRandom()
31  {
32  }
33
```

31 Diagram.cpp

```
1   #include "Diagram.hpp"
2
3   #include <stdexcept>
4   #include <iostream>
5
6   #include "util.hpp"
7
8   Diagram::Diagram()
9     :  _numCols(-1)
10    , _numRows(0)
11    , _positions()
12  {
13  }
14
15  bool
16  Diagram::addRow(std::string const& row) {
17      bool result = false;
18      if (_numCols == -1) {
19          _numCols = row.size();
20      }
21      if (row.size() == _numCols) {
22          int col = 0;
23          for (std::string::const_iterator c = row.begin(); c != row.end(); ++c) {
24              Positions::iterator cSet = findOrCreate(_positions,*c);
25              long k = index(_numRows,col);
26              cSet->second.insert(k);
27              //std::cout << "ins " << k << " " << _numRows << " >" << *c << "<" << std::endl;
28              ++col;
29          }
```

```
30          ++_numRows;
31          result = true;
32      }
33      return result;
34  }
35
36  bool
37  Diagram::contains(std::valarray<int> const& vector, std::string const & chars) {
38      bool result = false;
39      if (vector.size() == 2) {
40          long k = index(vector[1],vector[0]);
41          //std::cout << k << " >" << chars << "<" << std::endl;
42          for (std::string::const_iterator c = chars.begin(); c != chars.end(); ++c) {
43              Positions::iterator cSet = _positions.find(*c);
44              if (cSet != _positions.end()) {
45                  result = cSet->second.find(k) != cSet->second.end();
46                  //std::cout << "hit:" << *c << " " << result << std::endl;
47              }
48              if (result) {
49                  break;
50              }
51          }
52      }
53      else {
54          throw std::logic_error("diagram contains needs vector dim 2");
55      }
56      return result;
57  }
58
59  int
60  Diagram::rows() const
61  {
62      return _numRows;
63  }
64
65  int
66  Diagram::cols() const
67  {
68      int cols = _numCols;
69      if (cols < 0) cols = 0;
70      return cols;
71  }
72
73  long
74  Diagram::index(int row, int col) const
75  {
76      return col + row * _numCols;
77  }
78
```

32 DiscreteRandom.cpp

```
1  #include "DiscreteRandom.hpp"
2
3  DiscreteRandom::DiscreteRandom(RandomGenerator& generator, IntervalSet::Interval interval)
4    :  _numberGenerator(generator)
5  {
6      insert(interval);
7  }
8
9  int
10 DiscreteRandom::operator() () const
11 {
12     return 0;
13 }
14
15 double
16 DiscreteRandom::probability(int i) const
17 {
18     if (i == 0) {
19         return 1.0;
20     }
21     return 0.0;
22 }
23
24
25 DiscreteRandom::~DiscreteRandom()
26 {}
```

33 Dispatcher.cpp

```
#include "Dispatcher.hpp"


#define NUMERIC_UNARY(op,token)                                  \
    addUnary<IntegerType,IntegerType,op>(token);                  \
    addUnary<DoubleType,DoubleType,op>(token);

#define NUMERIC_BINARY(op,token)                                 \
    addBinary<IntegerType,IntegerType,IntegerType,op>(token); \
    addBinary<IntegerType,DoubleType,DoubleType,op>(token);   \
    addBinary<DoubleType,IntegerType,DoubleType,op>(token);   \
    addBinary<DoubleType,DoubleType,DoubleType,op>(token);    \
    addRhsArrayBinary<IntegerType,IntegerArrayType,IntegerArrayType,op>(token); \
    addLhsArrayBinary<IntegerArrayType,IntegerType,IntegerArrayType,op>(token); \
    addArrayBinary<IntegerArrayType,IntegerArrayType,IntegerArrayType,op>(token);

#define NUMERIC_COMPARISON_BINARY(op,token)                   \
    addBinary<IntegerType,IntegerType,BooleanType,op>(token); \
    addBinary<IntegerType,DoubleType,BooleanType,op>(token);  \
    addBinary<DoubleType,IntegerType,BooleanType,op>(token);  \
    addBinary<DoubleType,DoubleType,BooleanType,op>(token);   \
    addRhsArrayBinary<IntegerType,IntegerArrayType,BooleanArrayType,op>(token);    \
    addLhsArrayBinary<IntegerArrayType,IntegerType,BooleanArrayType,op>(token);    \
    addArrayBinary<IntegerArrayType,IntegerArrayType,BooleanArrayType,op>(token);  \
    addRhsArrayBinary<DoubleType,DoubleArrayType,BooleanArrayType,op>(token);    \
    addLhsArrayBinary<DoubleArrayType,DoubleType,BooleanArrayType,op>(token);    \
    addArrayBinary<DoubleArrayType,DoubleArrayType,BooleanArrayType,op>(token);


Dispatcher::Dispatcher()
  :  _unaryDispatchers()
  , _unaryInstructionDispatchers()
  , _binaryDispatchers()
{
    NUMERIC_UNARY(negate,UNARY_MINUS);
    addUnary<IntegerType,IntegerType,abs>(FAKE_ABS);
    addUnary<BooleanType,BooleanType,lnot>(LOGICAL_NOT);

    addUnaryInstruction<IntegerType,IntegerType,::assign>(ASSIGN);
    addUnaryInstruction<IntegerType,DoubleType,::assign>(ASSIGN);
    addUnaryInstruction<DoubleType,IntegerType,::assign>(ASSIGN);
    addUnaryInstruction<DoubleType,DoubleType,::assign>(ASSIGN);
    addUnaryInstruction<BooleanType,BooleanType,::assign>(ASSIGN);
    addUnaryInstruction<IntegerArrayType,IntegerArrayType,::assign>(ASSIGN);
    addUnaryInstruction<DoubleArrayType,DoubleArrayType,::assign>(ASSIGN);
    addUnaryInstruction<BooleanArrayType,BooleanArrayType,::assign>(ASSIGN);

    NUMERIC_BINARY(add,PLUS);
    NUMERIC_BINARY(subtract,MINUS);
    NUMERIC_BINARY(multiply,STAR);
    NUMERIC_BINARY(divide,SLASH);
    NUMERIC_BINARY(max,FAKE_MAX);
    NUMERIC_BINARY(min,FAKE_MIN);

    NUMERIC_COMPARISON_BINARY(lessThan,LESS_THAN);
    NUMERIC_COMPARISON_BINARY(greaterThan,GREATER_THAN);
    NUMERIC_COMPARISON_BINARY(lessEqThan,LESS_OR_EQUAL);
    NUMERIC_COMPARISON_BINARY(greaterEqThan,GREATER_OR_EQUAL);
    NUMERIC_COMPARISON_BINARY(equals,EQUAL);
    NUMERIC_COMPARISON_BINARY(notEquals,NOT_EQUAL);
    // booleans aren't numeric, but can be equality compared so add explicitly
    addBinary<BooleanType,BooleanType,BooleanType,equals>(EQUAL);
    addBinary<BooleanType,BooleanType,BooleanType,notEquals>(NOT_EQUAL);

    addBinary<IntegerType,IntegerType,IntervalType,intervalBounds>(COLON);

    addBinary<IntegerArrayType,IntegerType,IntegerType,subscript>(INDEX);
    addBinary<DoubleArrayType,IntegerType,DoubleType,subscript>(INDEX);
    addBinary<BooleanArrayType,IntegerType,BooleanType,subscript>(INDEX);

    addBinary<IntegerType,SetType,BooleanType,in>(LITERAL_in);
    addBinary<IntegerType,SetType,BooleanType,notIn>(LITERAL_not);
}

RllType::Ptr
Dispatcher::dispatchOps(RllType::Ptr in, int type, Scope & scope)
{
    RllType::Ptr result;
    UnaryDispatchers::iterator i = _unaryDispatchers.find(type);
```

```
80      if (i != _unaryDispatchers.end()) {
81          result = i->second.go(in,scope);
82      }
83      return result;
84  }
85
86  RllType::Ptr
87  Dispatcher::dispatchOps(RllType::Ptr lhs, RllType::Ptr rhs, int type, Scope & scope)
88  {
89      RllType::Ptr result;
90      BinaryDispatchers::iterator i = _binaryDispatchers.find(type);
91      if (i != _binaryDispatchers.end()) {
92          result = i->second.go(lhs,rhs,scope);
93      }
94      return result;
95  }
96
97  bool
98  Dispatcher::dispatchInstructions(RllType::Ptr lhs, RllType::Ptr rhs, int type, Scope & scope)
99  {
100     bool result = false;
101     UnaryInstructionDispatchers::iterator i = _unaryInstructionDispatchers.find(type);
102     if (i != _unaryInstructionDispatchers.end()) {
103         i->second.go(lhs,rhs,scope);
104         result = true;
105     }
106     return result;
107 }
```

34 Exceptions.cpp

```
1  #include "Exceptions.hpp"
2
3  InvalidTreeStructureException::InvalidTreeStructureException(std::string const& s)
4    :  logic_error(s)
5  {
6  }
7
8  InvalidTreeStructureException::~InvalidTreeStructureException() throw()
9  {
10 }
11
12 ModifyConstantException::ModifyConstantException(std::string const& s)
13   :  logic_error(s)
14 {
15 }
16
17 ModifyConstantException::~ModifyConstantException() throw()
18 {
19 }
20
21 UnimplementedOperationException::UnimplementedOperationException(std::string const& s)
22   :  logic_error(s)
23 {
24 }
25
26 UnimplementedOperationException::~UnimplementedOperationException() throw()
27 {
28 }
```

35 FunctionCall.cpp

```
1  #include "FunctionCall.hpp"
2
3  FunctionCall::FunctionCall(FunctionType::Ptr fn)
4    :  _function(fn)
5  {
6  }
7
8  void
9  FunctionCall::operator()() const
10 {
11 #ifdef ENABLE_INST_LOG
12      std::cout << "function "<< _function->name() << std::endl;
13 #endif
14     _function->execute();
```

```
15 }
16
17 FunctionCall::~FunctionCall()
18 {
19 }
20
21
22 TernaryCall::TernaryCall(TernaryType::Ptr fn)
23   :  _function(fn)
24 {
25 }
26
27 void
28 TernaryCall::operator()() const
29 {
30 #ifdef ENABLE_INST_LOG
31      std::cout << "ternary "<< std::endl;
32 #endif
33     _function->execute();
34 }
35
36 TernaryCall::~TernaryCall()
37 {
38 }
39
40 DiagramMemberInstruction::DiagramMemberInstruction(DiagramType::Ptr diagram, IntegerArrayType::Ptr array, std::string const&
chars, BooleanType::Ptr out)
41   :  _diagramRef(diagram->ref())
42   , _arrayRef(array->ref())
43   , _chars(chars)
44   , _outRef(out->ref())
45   , _diagramVariable(diagram)
46   , _arrayVariable(array)
47   , _outVariable(out)
48 {}
49
50 void
51 DiagramMemberInstruction::operator()() const
52 {
53     _outRef = _diagramRef.contains(_arrayRef,_chars);
54 #ifdef ENABLE_INST_LOG
55      std::cout << "diagramMember  diagram:  "<< _diagramVariable->name() << " chars:  "<< _chars << " array:  "<<
_arrayVariable->name() << " [";
56      for (std::size_t i = 0; i < _arrayRef.size(); ++i) {
57         std::cout << _arrayRef[i] << " ";
58      }
59      std::cout << "] out:  "<< _outVariable->name() << " = "<< _outRef << std::endl;
60 #endif
61
62 }
63
64 DiagramMemberInstruction::~DiagramMemberInstruction()
65 {}
66
67
68 RandomCreateInstruction::RandomCreateInstruction(RandomType::Ptr random, DiscreteRandom::RandomGenerator & generator)
69   :  _random(random)
70   , _generator(generator)
71 {}
72
73 void
74 RandomCreateInstruction::operator()() const
75 {
76     _random->create(_generator);
77
78 #ifdef ENABLE_INST_LOG
79      std::cout << "randomCreate  random:  "<< _random->name() << std::endl;
80 #endif
81 }
82
83 RandomCreateInstruction::~RandomCreateInstruction()
84 {}
85
86 RandomDrawInstruction::RandomDrawInstruction(RandomType::Ptr random)
87   :  _random(random)
88 {}
89
90 void
91 RandomDrawInstruction::operator()() const
92 {
93     _random->draw();
```

```
#ifdef ENABLE_INST_LOG
     std::cout << "randomDraw  random:  "<< _random->name() << " = "<< _random->getReturn()->ref() << std::endl;
#endif
}

RandomDrawInstruction::~RandomDrawInstruction()
{}
```

36 IntervalSet.cpp

```
#include <limits>
#include <iostream>

#include "IntervalSet.hpp"

IntervalSet::IntervalSet()
:  _intervals(),
   _cardinality(0)
{
}

void
IntervalSet::insert(const Value& v)
{
    Interval interval(v,v);
    insert(interval);
}

void
IntervalSet::insert(std::pair<Value,Value> const& pair)
{
    Interval interval(pair.first,pair.second);
    insert(interval);
}

void
IntervalSet::insert(const Interval& interval)
{
    Interval h = interval;

    for (IntervalList::iterator i = _intervals.begin(); i != _intervals.end(); /* increment in loop */) {
        if (overlap(h, *i)) {
            // need to merge intervals
            h = hull(*i,h);
            _intervals.erase(i++); //increment before erasing
        }
        else if (h < *i) {
            // need to add before i
            _intervals.insert(i,h);
            updateCardinality();
            return;
        }
        else {
            ++i;
        }
    }
    // add to end
    _intervals.push_back(h);
    updateCardinality();
    //std::cout << interval.lower() << " " << interval.upper() << " " << _cardinality << std::endl;
}

int
IntervalSet::cardinality() const
{
    return _cardinality;
}

IntervalSet::Value
IntervalSet::valueOfIndex(const Index& index) const
{
    int startIndex = 0;
    for (IntervalList::const_iterator i = _intervals.begin(); i != _intervals.end(); ++i) {
        int endIndex = startIndex + 1 + i->upper() - i->lower();
        if (index >= startIndex && index < endIndex) {
            Value v = i->lower() + index - startIndex;
            return v;
        }
        startIndex = endIndex;
```

```
70      }
71
72      return -666;  // TODO throw exception
73  }
74
75  IntervalSet::Index
76  IntervalSet::indexOfValue(const Value& value) const
77  {
78      int startIndex = 0;
79      for (IntervalList::const_iterator i = _intervals.begin(); i != _intervals.end(); ++i) {
80          if (in(value,*i)) {
81              Index index = startIndex + value - i->lower();
82              return index;
83          }
84          startIndex += 1 + i->upper() - i->lower();
85      }
86
87      return -666;  // TODO throw exception
88  }
89
90  bool
91  IntervalSet::contains(Value const& value) const
92  {
93      return indexOfValue(value) >= 0;
94  }
95
96  IntervalSet::const_iterator
97  IntervalSet::begin() const
98  {
99      return const_iterator(_intervals.begin());
100 }
101
102
103 IntervalSet::const_iterator
104 IntervalSet::end() const
105 {
106     return const_iterator(_intervals.end());
107 }
108
109 void
110 IntervalSet::updateCardinality() {
111
112     int count = 0;
113     for (IntervalList::iterator i = _intervals.begin(); i != _intervals.end(); ++i) {
114         count += 1 + i->upper() - i->lower();
115     }
116     _cardinality = count;
117 }
```

37 IntervalSetVector.cpp

```
1  #include <valarray>
2  #include <iostream>
3
4  #include "IntervalSetVector.hpp"
5  #include "IntervalSetVectorIterator.hpp"
6
7  IntervalSetVector::IntervalSetVector()
8  :  _sets()
9  {
10 }
11
12 void
13 IntervalSetVector::addSet(const IntervalSet& set) {
14     _sets.push_back(set);
15 }
16
17 IntervalSet&
18 IntervalSetVector::operator [](std::size_t i) {
19     return _sets[i];
20 }
21
22 const IntervalSet&
23 IntervalSetVector::operator [](std::size_t i) const{
24     return _sets[i];
25 }
26
27 IntervalSetVector::const_iterator
28 IntervalSetVector::begin() const
```

```
{
    std::list<IntervalSet::const_iterator> iterators;
    for (SetList::const_iterator i = _sets.begin(); i != _sets.end(); ++i) {
        iterators.push_back(i->begin());
    }
    return IntervalSetVectorConstIterator(_sets, iterators);
}

IntervalSetVector::const_iterator
IntervalSetVector::end() const {
    std::list<IntervalSet::const_iterator> iterators;
    for (SetList::const_iterator i = _sets.begin(); i != _sets.end(); ++i) {
        iterators.push_back(i->end());
    }
    return IntervalSetVectorConstIterator(_sets, iterators);
}

std::valarray<int>
IntervalSetVector::valueOfIndex(int index) const
{
    std::valarray<int> value(_sets.size());

    for (int i = 0; i < _sets.size(); ++i) {
        int elementIndex = index % _sets[i].cardinality();
        value[i] = _sets[i].valueOfIndex(elementIndex);
        index /= _sets[i].cardinality();
    }
    return value;
}

int
IntervalSetVector::indexOfValue(std::valarray<int> const& v) const
{
    int index = 0;
    int factor = 1;
    for (int i = 0; i < v.size(); ++i) {
        index += factor * _sets[i].indexOfValue(v[i]);
        factor *= _sets[i].cardinality();
    }
    return index;
}

```

38 Main.cpp

```

#include <iostream>
#include <fstream>
#include <string>

#include "boost/smart_ptr.hpp"

#include "antlr/AST.hpp"
#include "antlr/ASTRefCount.hpp"
#include "antlr/CommonAST.hpp"

#include "RllLexer.hpp"
#include "RllParser.hpp"
#include "RllTokenStream.hpp"
#include "RllTreeWalker.hpp"
#include "LineCountingAST.hpp"

#include "Types.hpp"
#include "types/MarkovDecisionProcessType.hpp"

int main(int argc, const char* argv[] )
{
    std::string usage("Usage:  rll file.rll mdp_name [mdp_arg ...]");
    const int baseArgs = 3;

    if (argc < baseArgs) {
        std::cerr << usage << std::endl;
        return -1;
    }
    try {
        std::ifstream file(argv[1]);

        if (!file) {
            std::cerr << "can't open "<< argv[1] << std::endl;
```

```
35              return -1;
36          }
37
38          antlr::ASTFactory factory( "LineCountingAST", LineCountingAST::factory );
39
40          RllLexer lexer(file);
41          lexer.init();
42          RllTokenStream tokenStream(lexer);
43          RllParser parser(tokenStream);
44
45          parser.initializeASTFactory(factory);
46          parser.setASTFactory(&factory);
47
48          parser.fileInput();
49
50          LineCountingAST::Ptr ast(parser.getAST());
51          if (ast) {
52              Scope scope;
53              Dispatcher disp;
54              RllTreeWalker walker(disp);
55              walker.fileInput(ast,scope);
56              if (scope.hasErrors()) {
57                  Scope::ErrorList const& errors = scope.getErrors();
58                  for (Scope::ErrorList::const_iterator i = errors.begin(); i != errors.end(); ++i) {
59                      std::cerr << (*i) << std::endl;
60                  }
61                  return -1;
62              }
63              std::string name(argv[2]);
64
65              MarkovDecisionProcessType::Ptr mdp = boost::dynamic_pointer_cast<MarkovDecisionProcessType>(scope.getSymbol(name));
66
67              int numArgs = argc - baseArgs;
68              if (mdp->numParameters() != numArgs) {
69                  std::cerr << name << " requires "<< mdp->numParameters() << std::endl;
70              }
71              else {
72                  std::list<RllType::Ptr> arguments;
73                  for (int i = 0; i < numArgs; ++i) {
74                      RllType::Ptr param = mdp->getParameter(i);
75                      bool inserted = false;
76                      try {
77                          IntegerType::Ptr intParam(new IntegerType(argv[baseArgs+i]));
78                          arguments.push_back(intParam);
79                          inserted = true;
80                      }
81                      catch (std::invalid_argument & ex) {
82                      }
83                      if (!inserted) {
84                          try {
85                              DoubleType::Ptr doubleParam(new DoubleType(argv[baseArgs+i]));
86                              arguments.push_back(doubleParam);
87                          }
88                          catch (std::invalid_argument & ex) {
89                              std::cerr << ex.what() << std::endl;
90                              return -1;
91                          }
92                      }
93                  }
94                  mdp->printPomdp(std::cout,arguments);
95                  for (Scope::ErrorList::const_iterator i = scope.getErrors().begin(); i != scope.getErrors().end(); ++i) {
96                      std::cout << (*i) << std::endl;
97                  }
98              }
99          }
100     }
101     catch( antlr::ANTLRException& e )
102     {
103         std::cerr << "exception:  "<< e.getMessage() << std::endl;
104         return -1;
105     }
106     catch( std::exception& e )
107     {
108         std::cerr << "exception:  "<< e.what() << std::endl;
109
110         std::cerr << usage << std::endl;
111
112         return -1;
113     }
114     return 0;
115 }
```

39 MarkovDecisionProcessType.cpp

```cpp
#include <stdexcept>
#include <sstream>
#include <iomanip>
#include <algorithm>

#include "types/MarkovDecisionProcessType.hpp"
#include "InstructionFactory.hpp"
#include "FunctionCall.hpp"
#include "RandomVector.hpp"

MarkovDecisionProcessType::MarkovDecisionProcessType(Scope & parent, Dispatcher & dispatcher)
  :  _generator(345376543u)
  , _dispatcher(dispatcher)
  , _parameterScope(parent.makeChildScope())
  , _initialScope(_parameterScope.makeChildScope())
  , _episodeScope(_initialScope.makeChildScope())
  , _environmentScope(_episodeScope.makeChildScope())
  , _currentScope(&_parameterScope)
  , _stateBounds()
  , _stateIndices()
  , _finalStateIndices()
  , _actionBounds()
  , _actionIndices()
  , _reward()
  , _parameters()
  , _episodeRandoms()
  , _environmentRandoms()
{
    initializePredicate(_initialScope,"validState",true);
    initializePredicate(_initialScope,"validAction",true);
    initializePredicate(_episodeScope,"isStart",true);
    initializePredicate(_episodeScope,"isTerminal",false);

    // make random distribution generators
}

bool
MarkovDecisionProcessType::defineParameter(const std::string& name)
{
    IntegerType::Ptr param(new IntegerType());
    bool result = _parameterScope.addSymbol(name,param);
    if (result) {
        _parameters.push_back(param);
    }
    return result;
}


bool
MarkovDecisionProcessType::defineParameter(const std::string& name, RllType::Ptr defaultValue)
{
    bool result = false;
    RllType::Ptr param = defaultValue->newSameType();
    if (_parameterScope.addSymbol(name,param) &&
        _dispatcher.dispatchInstructions(defaultValue,param,Dispatcher::ASSIGN,_parameterScope)) {
        _parameters.push_back(param);
        result = true;
    }
    return result;
}



Scope &
MarkovDecisionProcessType::getScope()
{
    return *_currentScope;
}

Scope const&
MarkovDecisionProcessType::getScope() const
{
    return *_currentScope;
}

Scope &
```

```
77 MarkovDecisionProcessType::getParameterScope()
78 {
79     return _parameterScope;
80 }
81
82 Scope const&
83 MarkovDecisionProcessType::getParameterScope() const
84 {
85     return _parameterScope;
86 }
87
88 bool
89 MarkovDecisionProcessType::defineStateVariable(const std::string& name, int dimension, SetType::Ptr bounds)
90 {
91     RllType::Ptr state;
92     RllType::Ptr finalState;
93     if (dimension == 1) {
94         state = IntegerType::Ptr(new IntegerType());
95         finalState = IntegerType::Ptr(new IntegerType());
96     }
97     else {
98         state = IntegerArrayType::Ptr(new IntegerArrayType(dimension));
99         finalState = IntegerArrayType::Ptr(new IntegerArrayType(dimension));
100    }
101
102    for (int i = 0; i < dimension; ++i) {
103        _stateBounds.push_back(bounds);
104        _stateIndices.push_back(VectorIndex(state,i));
105        _finalStateIndices.push_back(VectorIndex(finalState,i));
106    }
107
108    bool result = _initialScope.addSymbol(name,state);
109    if (result) {
110        result = _initialScope.addSymbol(name + "'",finalState);
111    }
112    return result;
113 }
114
115 bool
116 MarkovDecisionProcessType::defineActionVariable(const std::string& name, int dimension, SetType::Ptr bounds)
117 {
118     RllType::Ptr action;
119     if (dimension == 1) {
120         action = IntegerType::Ptr(new IntegerType());
121     }
122     else {
123         action = IntegerArrayType::Ptr(new IntegerArrayType(dimension));
124     }
125
126     for (int i = 0; i < dimension; ++i) {
127         _actionBounds.push_back(bounds);
128         _actionIndices.push_back(VectorIndex(action,i));
129     }
130
131     return _initialScope.addSymbol(name,action);
132 }
133
134 bool
135 MarkovDecisionProcessType::defineReward(const std::string& name)
136 {
137     bool result = true;
138     if (!_reward) {
139         _reward = DoubleType::Ptr(new DoubleType());
140     }
141     else {
142         // reward can't be redefined
143         result = false;
144     }
145     if (result) {
146         result = _initialScope.addSymbol(name,_reward);
147     }
148     return result;
149 }
150
151 bool
152 MarkovDecisionProcessType::defineRandomVariable(const std::string& name, RandomType::Ptr distribution)
153 {
154     bool result = false;
155     if (_currentScope == &_episodeScope ||_currentScope == &_environmentScope) {
156         BinomialRandomType::Ptr binomial = boost::dynamic_pointer_cast<BinomialRandomType>(distribution);
157         if (binomial) {
```

```
158             result = _currentScope->addSymbol(name,binomial->getBoolReturn());
159         }
160         else {
161             result = _currentScope->addSymbol(name,distribution->getReturn());
162         }
163         if (result) {
164             InstructionPtr createInst(new RandomCreateInstruction(distribution,_generator));
165             _initialScope.addInstruction(createInst);
166 
167             if (_currentScope == &_episodeScope) {
168                 _episodeRandoms.push_back(distribution);
169             }
170             else {
171                 _environmentRandoms.push_back(distribution);
172             }
173         }
174     }
175     return result;
176 }
177 
178 bool
179 MarkovDecisionProcessType::defineRandomVariable(const std::string& name, std::list<RandomType::Ptr> & distributions)
180 {
181     bool result = false;
182     IntegerArrayType::Ptr variable(new IntegerArrayType(distributions.size()));
183     result = _currentScope->addSymbol(name,variable);
184     if (result) {
185         int index = 0;
186         for (std::list<RandomType::Ptr>::iterator d = distributions.begin(); d != distributions.end(); ++d) {
187             InstructionPtr createInst(new RandomCreateInstruction(*d,_generator));
188             _initialScope.addInstruction(createInst);
189 
190             if (_currentScope == &_episodeScope) {
191                 _episodeRandoms.push_back(*d);
192             }
193             else {
194                 _environmentRandoms.push_back(*d);
195             }
196 
197             IntegerType::Ptr i(new IntegerType(index));
198             addBinaryOpInstruction<IntegerType,IntegerType,IntegerArrayType,arrayAssign>(i,(*d)->getReturn(),variable,*_currentScope);
199             ++index;
200         }
201     }
202     return result;
203 }
204 
205 void
206 MarkovDecisionProcessType::enterBodyDeclaration()
207 {
208     _currentScope = &_initialScope;
209 }
210 
211 void
212 MarkovDecisionProcessType::enterEpisodeDeclaration()
213 {
214     _currentScope = &_episodeScope;
215 }
216 
217 void
218 MarkovDecisionProcessType::enterEnvironmentDeclaration()
219 {
220     _currentScope = &_environmentScope;
221 }
222 
223 
224 void
225 MarkovDecisionProcessType::initializePredicate(Scope & scope, std::string const& name, bool value)
226 {
227     FunctionType::Ptr fn(new FunctionType(scope));
228     BooleanType::Ptr result(new BooleanType());
229     result->set(value);
230     fn->setReturn(result);
231     scope.addSymbol(name,fn,true);
232 }
233 
234 
235 RllType::Ptr
236 MarkovDecisionProcessType::getParameter(std::size_t position)
237 {
238     RllType::Ptr result;
```

```
239
240     if (position < _parameters.size()) {
241         result = _parameters[position];
242     }
243
244     return result;
245 }
246
247 std::size_t
248 MarkovDecisionProcessType::numParameters() const
249 {
250     return _parameters.size();
251 }
252
253 bool
254 MarkovDecisionProcessType::printPomdp(std::ostream & out, ArgumentList & arguments)
255 {
256     if (arguments.size() != _parameters.size()) {
257         std::cerr << "num parameters "<< _parameters.size() << " doesn't match num arguments "<< arguments.size() <<
" for printPomdp"<< std::endl;
258         return false;
259     }
260
261     {
262         Parameters::iterator p = _parameters.begin();
263         ArgumentList::iterator a = arguments.begin();
264         while( p != _parameters.end()) {
265             if (!_dispatcher.dispatchInstructions(*a,*p,Dispatcher::ASSIGN,_parameterScope)) {
266                 std::cerr << "argument type "<< typeid(*a).name() << " doesn't match parameter type "<< typeid(*p).name()
<< " for printPomdp"<< std::endl;
267                 return false;
268             }
269             ++p;
270             ++a;
271         }
272     }
273     _parameterScope.evaluateInstructions();
274     _initialScope.evaluateInstructions();
275     _episodeScope.evaluateInstructions(); // set isStart and isTerminal functions
276
277     out << "############"<< std::endl;
278     out << "discount:  0.9"<< std::endl;
279     out << "values:  reward"<< std::endl;
280
281     std::vector<int> stateWidths(_stateIndices.size());
282     IntervalSetVector startStates;
283     out << std::endl << "# states"<< std::endl;
284     std::size_t totalStates = 0;
285     for (std::size_t i = 0; i < _stateIndices.size(); ++i) {
286         out << "# "<< i+1 << ":  "<< _stateIndices[i].first->name();
287         if (boost::dynamic_pointer_cast<IntegerArrayType>(_stateIndices[i].first)) {
288            out << "["<< _stateIndices[i].second << "]";
289         }
290         out << " ("<< _stateBounds[i]->ref().cardinality() << " values)"<< std::endl;
291         if (totalStates == 0) {
292             totalStates = 1;
293         }
294         totalStates *= _stateBounds[i]->ref().cardinality();
295         startStates.addSet(_stateBounds[i]->ref());
296
297         std::ostringstream buffer;
298         for (IntervalSet::const_iterator v = _stateBounds[i]->ref().begin(); v != _stateBounds[i]->ref().end(); ++v) {
299             buffer << *v;
300             stateWidths[i] = std::max(stateWidths[i],(int)buffer.str().size());
301             buffer.str("");
302         }
303     }
304
305     out << "states:  "<< totalStates << std::endl;
306
307     std::vector<int> actionWidths(_actionIndices.size());
308     IntervalSetVector actions;
309     out << std::endl << "# actions"<< std::endl;
310     std::size_t totalActions = 0;
311     for (std::size_t i = 0; i < _actionIndices.size(); ++i) {
312         out << "# "<< i+1 << ":  "<< _actionIndices[i].first->name();
313         if (boost::dynamic_pointer_cast<IntegerArrayType>(_actionIndices[i].first)) {
314            out << "["<< _actionIndices[i].second << "]";
315         }
316         out << " ("<< _actionBounds[i]->ref().cardinality() << " values)"<< std::endl;
317         if (totalActions == 0) {
```

```
318             totalActions = 1;
319         }
320         totalActions *= _actionBounds[i]->ref().cardinality();
321         actions.addSet(_actionBounds[i]->ref());
322 
323         std::ostringstream buffer;
324         for (IntervalSet::const_iterator v = _actionBounds[i]->ref().begin(); v != _actionBounds[i]->ref().end(); ++v) {
325             buffer << *v;
326             actionWidths[i] = std::max(actionWidths[i],(int)buffer.str().size());
327             buffer.str("");
328         }
329     }
330 
331     out << "actions:  "<< totalActions << std::endl;
332 
333 //    out << std::endl << "# observations" << std::endl;
334 //    std::size_t totalObservations = 0;
335 //    for (std::size_t i = 0; i < _stateIndices.size(); ++i) {
336 //        out << "# " << i+1 << ":  " << _stateIndices[i].first->name();
337 //        if (boost::dynamic_pointer_cast<IntegerArrayType>(_stateIndices[i].first)) {
338 //           out << "[" << _stateIndices[i].second << "]";
339 //        }
340 //        out << " (" << _stateBounds[i]->ref().cardinality() << " values)" << std::endl;
341 //        if (totalObservations == 0) {
342 //            totalObservations = 1;
343 //        }
344 //        totalObservations *= _stateBounds[i]->ref().cardinality();
345 //    }
346 //
347 //    out << "observations:  " << totalObservations << std::endl;
348 
349     std::ostringstream buffer;
350     buffer << totalActions;
351 
352     int actionWidth = buffer.str().size();
353 
354     buffer.str("");
355 
356     buffer << totalStates;
357     int stateWidth = buffer.str().size();
358 
359     FunctionType::Ptr validStateFn = boost::dynamic_pointer_cast<FunctionType>(_episodeScope.getSymbol("validState"));
360     FunctionType::Ptr validActionFn = boost::dynamic_pointer_cast<FunctionType>(_episodeScope.getSymbol("validAction"));
361     bool & validState = boost::dynamic_pointer_cast<BooleanType>(validStateFn->getReturn())->ref();
362     bool & validAction = boost::dynamic_pointer_cast<BooleanType>(validActionFn->getReturn())->ref();
363 
364 
365     {
366         std::valarray<bool> isStart(false,totalStates);
367         FunctionType::Ptr fn = boost::dynamic_pointer_cast<FunctionType>(_episodeScope.getSymbol("isStart"));
368         std::size_t startIndex = 0;
369         std::size_t numTrue = 0;
370         for (IntervalSetVector::const_iterator startState = startStates.begin(); startState != startStates.end(); ++startState)
{
371             std::valarray<int> start = *startState;
372             set(_stateIndices,start);
373             fn->execute();
374             BooleanType::Ptr result = boost::dynamic_pointer_cast<BooleanType>(fn->getReturn());
375             isStart[startIndex] = result->get();
376             if (result->get()) {
377                 ++numTrue;
378             }
379             ++startIndex;
380         }
381         out << "start:"<< std::endl;
382         if (numTrue == isStart.size()) {
383             out << "uniform";
384         }
385         else {
386             for (std::size_t i = 0; i < isStart.size(); ++i) {
387                 out << (isStart[i] ?  1.0 :  0.0) / numTrue << " ";
388             }
389         }
390         out << std::endl << std::endl;
391     }
392 
393     RandomVector randoms;
394     for (std::size_t i = 0; i < _episodeRandoms.size(); ++i) {
395         //out << "add random" << std::endl;
396         randoms.addRandomVariable(_episodeRandoms[i]->distribution());
397     }
```

```
398     for (std::size_t i = 0; i < _environmentRandoms.size(); ++i) {
399         randoms.addRandomVariable(_environmentRandoms[i]->distribution());
400     }
401     int sIndex = 0;
402     for (IntervalSetVector::const_iterator startState = startStates.begin(); startState != startStates.end(); ++startState)
{
403         std::valarray<int> start = *startState;
404         set(_stateIndices,start);
405         validStateFn->execute();
406         if (validState) {
407             int aIndex = 0;
408             for (IntervalSetVector::const_iterator a = actions.begin(); a != actions.end(); ++a) {
409                 std::valarray<int> action = *a;
410                 set(_actionIndices,action);
411                 validActionFn->execute();
412                 if (validAction) {
413                     std::valarray<double> p(totalStates);
414                     std::valarray<double> r(totalStates);
415 
416                     bool randomRan = false;
417                     for (RandomVector::const_iterator rnd = randoms.begin(); rnd != randoms.end(); ++rnd) {
418                         std::valarray<int> random = *rnd;
419                         for (std::size_t i = 0; i < _episodeRandoms.size(); ++i) {
420                             BinomialRandomType::Ptr binom = boost::dynamic_pointer_cast<BinomialRandomType>(_episodeRandoms[i]);
421                             if (binom) {
422                                 binom->getBoolReturn()->set(random[i]);
423                                 //out << "random " << i << " " << random[i] <<  " " << binom->getBoolReturn()->get()
<< " " << binom->getBoolReturn()->name() << std::endl;
424                             }
425                             else {
426                                 _episodeRandoms[i]->getReturn()->set(random[i]);
427                             }
428                         }
429                         for (std::size_t i = 0; i < _environmentRandoms.size(); ++i) {
430                             BinomialRandomType::Ptr binom = boost::dynamic_pointer_cast<BinomialRandomType>(_environmentRandoms[i]);
431                             if (binom) {
432                                 binom->getBoolReturn()->set(random[i+_episodeRandoms.size()]);
433                             }
434                             else {
435                                 _environmentRandoms[i]->getReturn()->set(random[i+_episodeRandoms.size()]);
436                             }
437                         }
438 
439                         randomRan = true;
440                         _environmentScope.evaluateInstructions();
441 
442                         std::valarray<int> endState(_finalStateIndices.size());
443                         get(_finalStateIndices,endState);
444                         // ok to use start states for this since bounds are the same
445                         int finalIndex = startStates.indexOfValue(endState);
446                         if (finalIndex >= 0 && finalIndex < totalStates) {
447                             double prob = randoms.probability(random);
448                             p[finalIndex] += prob;
449                             r[finalIndex] += prob * _reward->get();
450                         }
451                     }
452 
453                     if (!randomRan) {
454                         _environmentScope.evaluateInstructions();
455                         std::valarray<int> endState(_finalStateIndices.size());
456                         get(_finalStateIndices,endState);
457                         // ok to use start states for this since bounds are the same
458                         int finalIndex = startStates.indexOfValue(endState);
459                         if (finalIndex >= 0 && finalIndex < totalStates) {
460                             p[finalIndex] = 1;
461                             r[finalIndex] = _reward->get();
462                         }
463                     }
464 
465                     for (std::size_t f = 0; f < totalStates; ++f) {
466                         if (p[f] > 0) {
467                             out << "T: "<< std::setw(actionWidth) << aIndex << " :  "<< std::setw(stateWidth) << sIndex
<< " :  "<<
468                                                 std::setw(stateWidth) << f << "  "<< std::setw(12) << std::fixed << std::setprecision(7)
<< p[f] << "   # a= ";
469                             for (std::size_t i = 0; i < action.size(); ++i) {
470                                 out << std::setw(actionWidths[i]) << action[i] << " ";
471                             }
472                             out << " s= ";
473                             for (std::size_t i = 0; i < start.size(); ++i) {
474                                 out << std::setw(stateWidths[i]) << start[i] << " ";
```

```
475                             }
476                             std::valarray<int> e = startStates.valueOfIndex(f);
477                             out << " e= ";
478                             for (std::size_t i = 0; i < e.size(); ++i) {
479                                 out << std::setw(stateWidths[i]) << e[i] << " ";
480                             }
481                             out << std::endl;
482                             double reward = r[f] / p[f];
483                             if (reward != 0.0) {
484                                 out << "R: "<< std::setw(actionWidth) << aIndex << " :  "<< std::setw(stateWidth) <<
sIndex << " :  "<<
485                                                                     std::setw(stateWidth) << f << "  "<< std::setw(12) << std::fixed <<
std::setprecision(7) << reward << "   # a= ";
486                                 for (std::size_t i = 0; i < action.size(); ++i) {
487                                     out << std::setw(actionWidths[i]) << action[i] << " ";
488                                 }
489                                 out << " s= ";
490                                 for (std::size_t i = 0; i < start.size(); ++i) {
491                                     out << std::setw(stateWidths[i]) << start[i] << " ";
492                                 }
493                                 out << " e= ";
494                                 for (std::size_t i = 0; i < e.size(); ++i) {
495                                     out << std::setw(stateWidths[i]) << e[i] << " ";
496                                 }
497                                 out << std::endl;
498                             }
499                         }
500                     }
501                 }
502                 ++aIndex;
503             }
504         }
505         ++sIndex;
506     }
507
508     return true;
509 }
510
511 void
512 MarkovDecisionProcessType::set(Indices & indices, std::valarray<int> const& values)
513 {
514     for (std::size_t i = 0; i < values.size(); ++i) {
515         if (IntegerArrayType::Ptr s = boost::dynamic_pointer_cast<IntegerArrayType>(indices[i].first)) {
516             s->set(indices[i].second,values[i]);
517         }
518         else {
519             IntegerType::Ptr s = boost::dynamic_pointer_cast<IntegerType>(indices[i].first);
520             s->set(values[i]);
521         }
522     }
523 }
524
525 void
526 MarkovDecisionProcessType::get(Indices & indices, std::valarray<int> & values)
527 {
528     for (std::size_t i = 0; i < values.size(); ++i) {
529         if (IntegerArrayType::Ptr s = boost::dynamic_pointer_cast<IntegerArrayType>(indices[i].first)) {
530             values[i] = s->get(indices[i].second);
531         }
532         else {
533             IntegerType::Ptr s = boost::dynamic_pointer_cast<IntegerType>(indices[i].first);
534             values[i] = s->get();
535         }
536     }
537 }
538
539
540 MarkovDecisionProcessType::~MarkovDecisionProcessType()
541 {
542 }
```

40 PoissonRandom.cpp

```
1 #include <algorithm>
2 #include <stdexcept>
3 #include <iostream>
4
5 #include "PoissonRandom.hpp"
6
```

```
 7
 8 PoissonRandom::PoissonRandom(RandomGenerator& generator, unsigned expected)
 9   :  DiscreteRandom(generator,IntervalSet::Interval(0,0))
10   , _generator(_numberGenerator,Distribution())
11   , _probabilities()
12 {
13     double n = expected;
14     double probability = std::exp(-n);
15     double sum = probability;
16     while (sum < 0.9999) {
17         _probabilities.push_back(sum);
18         //std::cout << expected << " " << _probabilities.size() << " " << probability << std::endl;
19         probability *= n / _probabilities.size();
20         sum += probability;
21     }
22     insert(IntervalSet::Interval(0,_probabilities.size()-1));
23 }
24
25 int
26 PoissonRandom::operator()()
27 {
28     double probability = _generator();
29
30     Probabilities::iterator i = std::lower_bound(_probabilities.begin(),_probabilities.end(),probability);
31     int diff = i - _probabilities.begin();
32     if (i == _probabilities.end()) {
33         --diff;
34     }
35     if (diff < 0) {
36         throw std::logic_error("poisson less than 0");
37     }
38     return diff;
39 }
40
41 double
42 PoissonRandom::probability(int i) const
43 {
44     double probability = 0.0;
45     if (i >= 0 && i < _probabilities.size()) {
46         probability = _probabilities[i];
47         if (i > 0) {
48             probability -= _probabilities[i-1];
49         }
50     }
51
52     return probability;
53 }
54
55
56 PoissonRandom::~PoissonRandom()
57 {
58 }
59
```

41 RandomVector.cpp

```
 1 #include "RandomVector.hpp"
 2
 3 RandomVector::RandomVector()
 4   :  _variables()
 5 {
 6 }
 7
 8 void
 9 RandomVector::addRandomVariable(RandomPtr x)
10 {
11     _variables.push_back(x);
12 }
13
14 std::valarray<int>
15 RandomVector::operator ()() const
16 {
17     std::valarray<int> x(_variables.size());
18
19     int index = 0;
20     for (VariableList::const_iterator v = _variables.begin(); v != _variables.end(); ++v) {
21         x[index] = (**v)();
22         ++index;
23     }
```

```
    return x;
}

double
RandomVector::probability(const std::valarray<int>& x) const
{
    double probability = 1.0;
    int index = 0;

    for (VariableList::const_iterator v = _variables.begin(); v != _variables.end(); ++v) {
        probability *= (*v)->probability(x[index]);
        ++index;
    }

    return probability;
}


RandomVector::const_iterator
RandomVector::begin() const
{
    std::list<DiscreteRandom::const_iterator> iterators;
    for (VariableList::const_iterator i = _variables.begin(); i != _variables.end(); ++i) {
        iterators.push_back((*i)->begin());
    }
    return RandomVectorConstIterator(_variables, iterators);
}

RandomVector::const_iterator
RandomVector::end() const
{    std::list<DiscreteRandom::const_iterator> iterators;
    for (VariableList::const_iterator i = _variables.begin(); i != _variables.end(); ++i) {
        iterators.push_back((*i)->end());
    }
    return RandomVectorConstIterator(_variables, iterators);
}
```

42 RllTokenStream.cpp

```
/*
[The "BSD licence"]
Copyright (c) 2004 Terence Parr and Loring Craymer
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1.  Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
2.  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.
3.  The name of the author may not be used to endorse or promote products
    derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "antlr/CommonToken.hpp"
#include "RllLexer.hpp"
#include "RllParser.hpp"
#include "RllTokenStream.hpp"

/** Python does not explicitly provide begin and end nesting signals.
Rather, the indentation level indicates when you begin and end.
This is an interesting lexical problem because multiple DEDENT
tokens should be sent to the parser sometimes without a corresponding
input symbol!  Consider the following example:

a=1
```

```
if a>1:
print a
b=3

Here the "b" token on the left edge signals that a DEDENT is needed
after the "print a \n" and before the "b".  The sequence should be

...  1 COLON NEWLINE INDENT PRINT a NEWLINE DEDENT b ASSIGN 3 ...

For more examples, see the big comment at the bottom of this file.

This TokenStream normally just passes tokens through to the parser.
Upon NEWLINE token from the lexer, however, an INDENT or DEDENT token
may need to be sent to the parser.  The NEWLINE is the trigger for
this class to do it's job.  NEWLINE is saved and then the first token
of the next line is examined.  If non-leading-whitespace token,
then check against stack for indent vs dedent.  If LEADING_WS, then
the column of the next non-whitespace token will dictate indent vs
dedent.  The column of the next real token is number of spaces
in the LEADING_WS token + 1 (to move past the whitespace).  The
lexer grammar must set the text of the LEADING_WS token to be
the proper number of spaces (and do tab conversion etc...).

A stack of column numbers is tracked and used to detect changes
in indent level from one token to the next.

A queue of tokens is built up to hold multiple DEDENT tokens that
are generated.  Before asking the lexer for another token via
nextToken(), the queue is flushed first one token at a time.

Terence Parr and Loring Craymer
February 2004
*/


RllTokenStream::RllTokenStream(RllLexer& lexer)
:_indentStack(new int[MAX_INDENTS]),
  _sp(-1),
  _tokens(),
  _lexer(lexer)
{
    push(FIRST_COLUMN); // "state" of indent level is FIRST_COLUMN
}

/** From http://www.python.org/doc/2.2.3/ref/indentation.html

"Before the first line of the file is read, a single zero is
pushed on the stack; this will never be popped off again.  The
numbers pushed on the stack will always be strictly increasing
from bottom to top.  At the beginning of each logical line, the
line's indentation level is compared to the top of the
stack.  If it is equal, nothing happens.  If it is larger, it is
pushed on the stack, and one INDENT token is generated.  If it
is smaller, it must be one of the numbers occurring on the
stack; all numbers on the stack that are larger are popped
off, and for each number popped off a DEDENT token is
generated.  At the end of the file, a DEDENT token is generated
for each number remaining on the stack that is larger than
zero."

I use columns 1..n instead.

The DEDENTS possibly needed at EOF are gracefully handled by forcing
EOF to have column 1 even though with UNIX it's hard to get EOF
at a non left edge.
*/

antlr::RefToken
RllTokenStream::nextToken() {
    // if something in queue, just remove and return it
    if ( _tokens.size()>0 ) {
        antlr::RefToken t = _tokens.front();
        _tokens.pop();
        // System.out.println(t);
        return t;
    }

    insertImaginaryIndentDedentTokens();

    return nextToken();
}
```

```
122
123 void
124 RllTokenStream::insertImaginaryIndentDedentTokens()
125 {
126     antlr::RefToken t = _lexer.nextToken();
127
128     // if not a NEWLINE, doesn't signal indent/dedent work; just enqueue
129     if ( t->getType()!= RllLexer::NEWLINE ) {
130         _tokens.push(t);
131         return;
132     }
133     // save NEWLINE in the queue
134     _tokens.push(t);
135     // grab first token of next line
136     t = _lexer.nextToken();
137
138     // compute col as the column number of next non-WS token in line
139     int col = t->getColumn(); // column dictates indent/dedent
140
141     if ( t->getType() == RllLexer::EOF_) {
142         col = 1; // pretend EOF always happens at left edge
143     }
144     else if ( t->getType()==RllLexer::LEADING_WS ) {
145         col = t->getText().length() + 1; // col is 1 spot after size
146     }
147
148     // compare to last indent level
149     int lastIndent = peek();
150     if ( col > lastIndent ) { // they indented; track and gen INDENT
151         push(col);
152         antlr::RefToken indent(new antlr::CommonToken(RllParser::INDENT,""));
153         indent->setColumn(t->getColumn());
154         indent->setLine(t->getLine());
155         _tokens.push(indent);
156     }
157     else if ( col < lastIndent ) { // they dedented
158         // how far back did we dedent?
159         int prevIndex = findPreviousIndent(col);
160         // generate DEDENTs for each indent level we backed up over
161         for (int d=_sp-1; d>=prevIndex; d--) {
162             antlr::RefToken dedent(new antlr::CommonToken(RllParser::DEDENT,""));
163             dedent->setColumn(t->getColumn());
164             dedent->setLine(t->getLine());
165             _tokens.push(dedent);
166         }
167         _sp = prevIndex; // pop those off indent level
168     }
169     if ( t->getType()!=RllLexer::LEADING_WS ) { // discard WS
170         _tokens.push(t);
171     }
172 }
173
174 //  T O K E N  S T A C K  M E T H O D S
175
176 void
177 RllTokenStream::push(int i) {
178     if (_sp>=MAX_INDENTS) {
179         throw std::logic_error("stack overflow");
180     }
181     ++_sp;
182     _indentStack[_sp] = i;
183 }
184
185 int
186 RllTokenStream::pop() {
187     if (_sp<0) {
188         throw std::logic_error("stack underflow");
189     }
190     int top = _indentStack[_sp];
191     --_sp;
192     return top;
193 }
194
195 int RllTokenStream::peek() const {
196     return _indentStack[_sp];
197 }
198
199 /** Return the index on stack of previous indent level == i else -1 */
200 int
201 RllTokenStream::findPreviousIndent(int i) const {
202     for (int j=_sp-1; j>=0; j--) {
```

```
203         if ( _indentStack[j]==i ) {
204             return j;
205         }
206     }
207     return -1;
208 }
209
210 RllTokenStream::~RllTokenStream() {
211     delete [] _indentStack;
212 }
213
214 /* More example input / output pairs with code simplified to single chars
215 ------- t1 -------
216 a a
217 b b
218 c
219 d
220 a a \n INDENT b b \n c \n DEDENT d \n EOF
221 ------- t2 -------
222 a  c
223 b
224 c
225 a c \n INDENT b \n DEDENT c \n EOF
226 ------- t3 -------
227 a
228 b
229 c
230 d
231 a \n INDENT b \n INDENT c \n DEDENT DEDENT d \n EOF
232 ------- t4 -------
233 a
234 c
235 d
236 e
237 f
238 g
239 h
240 i
241 j
242 k
243 a \n INDENT c \n INDENT d \n DEDENT e \n f \n INDENT g \n h \n i \n INDENT j \n DEDENT DEDENT k \n DEDENT EOF
244 ------- t5 -------
245 a
246 b
247 c
248 d
249 e
250 a \n INDENT b \n c \n INDENT d \n e \n DEDENT DEDENT EOF
251 */
```

43 RllTreeWalker.cpp

```
1  #include "RllTreeWalker.hpp"
2
3  #include <sstream>
4
5  #include "Exceptions.hpp"
6  #include "InstructionFactory.hpp"
7  #include "FunctionCall.hpp"
8
9  #define SHOW_TODO
10 #ifdef SHOW_TODO
11 #define TODO(ast,scope) error(__FILE__,__LINE__,(ast)->getText() + " not implemented yet",(ast)->getLine(),scope)
12 #else
13 #define TODO(ast,scope)
14 #endif
15
16 #define OPTIONAL(ast,value,scope) ((ast) && (ast)->getType() == value)
17 #define EXPECTED(ast,value,scope) expected(__FILE__,__LINE__,ast,value,#value, scope)
18 #define EXPECTED_TYPE(ast,ptr,type,scope) expectedType<type>(__FILE__,__LINE__,ast,ptr, scope)
19 #define ERROR(message,ast,scope) error(__FILE__,__LINE__,message,(ast)->getLine(),scope)
20 #define REDEFINE_ERROR(ast,scope) error(__FILE__,__LINE__,(ast)->getText() + " is already defined",(ast)->getLine(),scope)
21 #define INVALID(ast,scope) error(__FILE__,__LINE__,"invalid tree element "+(ast)->getText(),(ast)->getLine(),scope)
22
23 RllTreeWalker::RllTreeWalker(Dispatcher & dispatcher)
24   :  _dispatcher(dispatcher)
25 {
26 }
27
```

```
28 RllType::Ptr
29 RllTreeWalker::fileInput(AST ast, Scope &scope)
30 {
31     RllType::Ptr lastType;
32     if (EXPECTED(ast,FILE,scope)) {
33          for (AST child(ast->getFirstChild()); child; child = AST(child->getNextSibling())) {
34             if (child->getType() == LITERAL_mdp) {
35                 lastType = mdp(child,scope);
36             }
37             else if (child->getType() == ASSIGN ||
38                      child->getType() == PLUS_ASSIGN ||
39                      child->getType() == MINUS_ASSIGN ||
40                      child->getType() == STAR_ASSIGN ||
41                      child->getType() == SLASH_ASSIGN) {
42                 TODO(child,scope);
43             }
44             else if (child->getType() == CALL) {
45                 TODO(child,scope);
46             }
47             else if (child->getType() == LITERAL_for) {
48                 TODO(child,scope);
49             }
50             else if (child->getType() == LITERAL_while) {
51                 TODO(child,scope);
52             }
53             else if (child->getType() == LITERAL_if) {
54                 TODO(child,scope);
55             }
56             else {
57                 INVALID(child,scope);
58             }
59          }
60     }
61     return lastType;
62 }
63
64 RllType::Ptr
65 RllTreeWalker::mdp(AST ast, Scope &scope)
66 {
67     MarkovDecisionProcessType::Ptr mdp(new MarkovDecisionProcessType(scope, _dispatcher));
68     ast = AST(ast->getFirstChild());
69     if (EXPECTED(ast,ID,scope)) {
70         std::string name = ast->getText();
71
72         ast = ast->getNextSibling();
73         if (OPTIONAL(ast, PARAMS, mdp->getScope())) {
74             setMdpParameters(ast,mdp);
75             ast = ast->getNextSibling();
76         }
77         if (EXPECTED(ast, MDP_BODY, mdp->getScope())) {
78             for (AST body(ast->getFirstChild()); body; body = AST(body->getNextSibling())) {
79                 fillMdpBody(body,mdp);
80             }
81         }
82         scope.addSymbol(name,mdp);
83     }
84     return mdp;
85 }
86
87 void
88 RllTreeWalker::setMdpParameters(AST ast, MarkovDecisionProcessType::Ptr mdp)
89 {
90     for (ast = AST(ast->getFirstChild()); ast; ast = AST(ast->getNextSibling())) {
91         if (ast->getType() == ID) {
92             if (!mdp->defineParameter(ast->getText())) {
93                 REDEFINE_ERROR(ast,mdp->getScope());
94             }
95         }
96         else if (ast->getType() == ASSIGN) {
97             AST id(ast->getFirstChild());
98             if (EXPECTED(id,ID,mdp->getScope())) {
99                 std::string name = id->getText();
100                AST initNode(id->getNextSibling());
101                RllType::Ptr initType = expression(initNode,mdp->getScope());
102                if (!initType->isConstant()) {
103                    ERROR("Parameter must have a constant initializer",initNode,mdp->getScope());
104                }
105                if (!mdp->defineParameter(name,initType)) {
106                    REDEFINE_ERROR(id,mdp->getScope());
107                }
108            }
```

```
109         }
110         else {
111             INVALID(ast,mdp->getScope());
112         }
113     }
114 }
115
116 void
117 RllTreeWalker::fillMdpBody(AST ast, MarkovDecisionProcessType::Ptr mdp)
118 {
119     mdp->enterBodyDeclaration();
120
121     if (ast->getType() == LITERAL_diagram) {
122         addMdpDiagram(ast,mdp);
123     }
124     else if (ast->getType() == LITERAL_states) {
125         addMdpStates(ast,mdp);
126     }
127     else if (ast->getType() == LITERAL_actions) {
128         addMdpActions(ast,mdp);
129     }
130     else if (ast->getType() == LITERAL_reward) {
131         addMdpReward(ast,mdp);
132     }
133     else if (ast->getType() == ASSIGN) {
134         addMdpConstraint(ast,mdp);
135     }
136     else if (ast->getType() == LITERAL_episode) {
137         addMdpEpisode(ast,mdp);
138     }
139     else if (ast->getType() == LITERAL_environment) {
140         addMdpEnvironment(ast,mdp);
141     }
142     else {
143         INVALID(ast,mdp->getScope());
144     }
145 }
146
147 void
148 RllTreeWalker::addMdpDiagram(AST ast, MarkovDecisionProcessType::Ptr mdp)
149 {
150     DiagramType::Ptr diagram(new DiagramType());
151     AST id(ast->getFirstChild());
152     if (EXPECTED(id,ID,mdp->getScope())) {
153         for (AST row(id->getNextSibling()); row; row = row->getNextSibling()) {
154            if (EXPECTED(row,STRING_LITERAL,mdp->getScope())) {
155                 if (!diagram->ref().addRow(row->getText())) {
156                     ERROR("Diagram rows must be the same size",row,mdp->getScope());
157                 }
158             }
159         }
160         if (!mdp->getScope().addSymbol(id->getText(),diagram)) {
161             REDEFINE_ERROR(id,mdp->getScope());
162         }
163     }
164 }
165
166 void
167 RllTreeWalker::addMdpStates(AST ast, MarkovDecisionProcessType::Ptr mdp)
168 {
169     for (AST decl(ast->getFirstChild()); decl; decl = decl->getNextSibling()) {
170         if (EXPECTED(decl,DECL,mdp->getScope())) {
171             AST child(decl->getFirstChild());
172             if (EXPECTED(child,ID,mdp->getScope())) {
173                 std::string id = child->getText();
174                 int dimension = 1;
175                 AST dim(child->getFirstChild());
176                 if (dim && EXPECTED(dim,DIM,mdp->getScope())) {
177                     AST value(dim->getFirstChild());
178                     if (EXPECTED(value,INT,mdp->getScope())) {
179                         std::istringstream buffer(value->getText());
180                         buffer >> dimension;
181                     }
182                 }
183                 child = AST(child->getNextSibling());
184                 RllType::Ptr bounds = expression(child,mdp->getScope());
185                 SetType::Ptr typedBounds = boost::dynamic_pointer_cast<SetType>(bounds);
186                 if (!typedBounds) {
187                     if (DiagramType::Ptr diagram = EXPECTED_TYPE(child,bounds,DiagramType,mdp->getScope())) {
188                         if (dimension != 2) {
189                             ERROR("State must have dimension 2 to use diagram as bounds",child,mdp->getScope());
```

```
190                         }
191                         else {
192                             typedBounds = SetType::Ptr(new SetType());
193                             // we don't track separate bounds for each element of vector so add both intervals
194                             typedBounds->ref().insert(IntervalSet::Interval(0,diagram->ref().rows()));
195                             typedBounds->ref().insert(IntervalSet::Interval(0,diagram->ref().cols()));
196                         }
197                     }
198                 }
199 
200                 if (!typedBounds) {
201                     ERROR(id + " bounds must be set or diagram",child,mdp->getScope());
202                 }
203                 else if (!mdp->defineStateVariable(id,dimension,typedBounds)) {
204                     ERROR(id + " already defined",decl,mdp->getScope());
205                 }
206             }
207         }
208     }
209 }
210 
211 
212 void
213 RllTreeWalker::addMdpActions(AST ast, MarkovDecisionProcessType::Ptr mdp)
214 {
215     for (AST decl(ast->getFirstChild()); decl; decl = decl->getNextSibling()) {
216         if (EXPECTED(decl,DECL,mdp->getScope())) {
217             AST child(decl->getFirstChild());
218             if (EXPECTED(child,ID,mdp->getScope())) {
219                 std::string id = child->getText();
220                 int dimension = 1;
221                 AST dim(child->getFirstChild());
222                 if (dim && EXPECTED(dim,DIM,mdp->getScope())) {
223                     AST value(dim->getFirstChild());
224                     if (EXPECTED(value,INT,mdp->getScope())) {
225                         std::istringstream buffer(value->getText());
226                         buffer >> dimension;
227                     }
228                 }
229                 child = AST(child->getNextSibling());
230                 RllType::Ptr bounds = expression(child,mdp->getScope());
231                 if (SetType::Ptr typedBounds = EXPECTED_TYPE(child,bounds,SetType,mdp->getScope()))
232                 {
233                     if (!mdp->defineActionVariable(id,dimension,typedBounds)) {
234                         ERROR(id + " already defined",decl,mdp->getScope());
235                     }
236                 }
237             }
238         }
239     }
240 }
241 
242 void
243 RllTreeWalker::addMdpReward(AST ast, MarkovDecisionProcessType::Ptr mdp)
244 {
245     ast = AST(ast->getFirstChild());
246     if (EXPECTED(ast,ID,mdp->getScope())) {
247         if (!mdp->defineReward(ast->getText())) {
248             ERROR(ast->getText() + " already defined or mdp already has a reward",ast,mdp->getScope());
249         }
250     }
251 }
252 
253 void
254 RllTreeWalker::addMdpConstraint(AST ast, MarkovDecisionProcessType::Ptr mdp)
255 {
256     AST id(ast->getFirstChild());
257     if (EXPECTED(id,ID,mdp->getScope())) {
258         FunctionType::Ptr predicate(new FunctionType(mdp->getScope()));
259         BooleanType::Ptr result(new BooleanType());
260         predicate->setReturn(result);
261 
262         AST expr(id->getNextSibling());
263         RllType::Ptr rhs = expression(expr,predicate->getScope());
264         if (EXPECTED_TYPE(expr,rhs,BooleanType,predicate->getScope())) {
265             _dispatcher.dispatchInstructions(rhs,result,ASSIGN,predicate->getScope());
266         }
267         mdp->getScope().overwriteSymbol(id->getText(),predicate);
268     }
269 }
270 
```

```
271 void
272 RllTreeWalker::addMdpEpisode(AST ast, MarkovDecisionProcessType::Ptr mdp)
273 {
274     mdp->enterEpisodeDeclaration();
275 
276     for (AST child(ast->getFirstChild()); child; child = AST(child->getNextSibling())) {
277         if (child->getType() == ASSIGN) {
278             addMdpConstraint(child,mdp);
279         }
280         else if (child->getType() == LITERAL_random) {
281             addMdpRandom(child,mdp);
282         }
283         else {
284             INVALID(child,mdp->getScope());
285         }
286     }
287 }
288 
289 void
290 RllTreeWalker::addMdpEnvironment(AST ast, MarkovDecisionProcessType::Ptr mdp)
291 {
292     mdp->enterEnvironmentDeclaration();
293 
294     for (AST child(ast->getFirstChild()); child; child = AST(child->getNextSibling())) {
295         if (child->getType() == ASSIGN) {
296             assign(child,mdp->getScope());
297         }
298         else if (child->getType() == LITERAL_random) {
299             addMdpRandom(child,mdp);
300         }
301         else {
302             INVALID(child,mdp->getScope());
303         }
304     }
305 }
306 
307 void
308 RllTreeWalker::addMdpRandom(AST ast, MarkovDecisionProcessType::Ptr mdp)
309 {
310     AST id(ast->getFirstChild());
311     if (EXPECTED(id,ID,mdp->getScope())) {
312         AST distribution(id->getNextSibling());
313         if (!distribution) {
314              ERROR("random variable declaration needs distribution",ast,mdp->getScope());
315         }
316         else if (distribution->getType() == ARRAY) {
317             std::list<RandomType::Ptr> distributions;
318             for (AST row(distribution->getFirstChild()); row; row = AST(row->getNextSibling())) {
319                 if (row->getNumberOfChildren() != 1) {
320                     ERROR("Single dimension arrays only",row,mdp->getScope());
321                 }
322                 else {
323                     AST d(row->getFirstChild());
324                     RandomType::Ptr random = EXPECTED_TYPE(d,expression(d,mdp->getParameterScope()),RandomType,mdp->getParameterScope());
325                     if (random) {
326                        distributions.push_back(random);
327                     }
328                 }
329             }
330             if (!mdp->defineRandomVariable(id->getText(),distributions)) {
331                 REDEFINE_ERROR(id,mdp->getParameterScope());
332             }
333         }
334         else {
335             // use initial scope only to resolve arguments to distribution
336             RandomType::Ptr random = EXPECTED_TYPE(distribution,expression(distribution,mdp->getParameterScope()),RandomType,mdp->getParamete
337             if (random) {
338                 if (!mdp->defineRandomVariable(id->getText(),random)) {
339                     REDEFINE_ERROR(id,mdp->getParameterScope());
340                 }
341             }
342         }
343     }
344 }
345 
346 
347 RllType::Ptr
348 RllTreeWalker::expression(AST ast, Scope & scope)
349 {
350     RllType::Ptr result;
351 
```

```
352     if (ast->getType() == OPEN_PAREN) {
353         result = expression(AST(ast->getFirstChild()),scope);
354     }
355     else if (ast->getType() == TICK) {
356         AST id(ast->getFirstChild());
357         if (EXPECTED(id,ID,scope)) {
358             result = scope.getSymbol(id->getText()+"'");
359         }
360     }
361     else if (ast->getType() == ID) {
362         result = scope.getSymbol(ast->getText());
363         if (FunctionType::Ptr function = boost::dynamic_pointer_cast<FunctionType>(result)) {
364             if (function->numParameters() == 0) {
365                 // this is a 0 param function being used as variable, so generate a funcion call and
366                 // return result
367                 result = function->getReturn();
368                 InstructionPtr call(new FunctionCall(function));
369                 scope.addInstruction(call);
370             }
371         }
372     }
373     else if (ast->getType() == CALL) {
374         result = functionCall(ast,scope);
375     }
376     else if (ast->getType() == INT) {
377         result = RllType::Ptr(new IntegerType(ast->getText()));
378     }
379     else if (ast->getType() == FLOAT) {
380         result = RllType::Ptr(new DoubleType(ast->getText()));
381     }
382     else if (ast->getType() == SET ||ast->getType() == BOUNDS) {
383         SetType::Ptr set(new SetType());
384         bool isConstant = true;
385         bool areLiterals = false;
386         int literalValue = 0;
387         if (ast->getType() == BOUNDS) {
388             areLiterals = true;
389             for (AST child(ast->getFirstChild()); child; child = AST(child->getNextSibling())) {
390                 if (child->getType() != ID ||scope.getSymbol(child->getText())) {
391                     areLiterals = false;
392                 }
393             }
394         }
395         for (AST child(ast->getFirstChild()); child; child = AST(child->getNextSibling())) {
396             if (areLiterals) {
397                 IntegerType::Ptr literal(new IntegerType(literalValue));
398                 ++literalValue;
399                 if (!scope.addSymbol(child->getText(),literal)) {
400                     throw std::logic_error("symbol creation failed when areLiterals true");
401                 }
402                 addUnaryOpInstruction<IntegerType,SetType,::insert>(literal,set,scope);
403             }
404             else {
405                 RllType::Ptr element = expression(child,scope);
406                 if (!element) {
407                     INVALID(child,scope);
408                 }
409                 else if (IntegerType::Ptr typedElement = boost::dynamic_pointer_cast<IntegerType>(element)) {
410                     addUnaryOpInstruction<IntegerType,SetType,::insert>(typedElement,set,scope);
411                     isConstant = isConstant && typedElement->isConstant();
412                 }
413                 else if (IntervalType::Ptr typedElement = boost::dynamic_pointer_cast<IntervalType>(element)) {
414                     addUnaryOpInstruction<IntervalType,SetType,::insert>(typedElement,set,scope);
415                     isConstant = isConstant && typedElement->isConstant();
416                 }
417                 else {
418                     INVALID(child,scope);
419                 }
420             }
421         }
422         result = set;
423     }
424     else if (ast->getType() == ARRAY) {
425         IntegerArrayType::Ptr array(new IntegerArrayType(ast->getNumberOfChildren()));
426         bool isConstant = true;
427         int i = 0;
428         for (AST child(ast->getFirstChild()); child; child = AST(child->getNextSibling())) {
429             if (child->getType() != ROW ||child->getNumberOfChildren() != 1) {
430                 INVALID(child,scope);
431             }
432             else {
```

```
433                 AST elementNode(child->getFirstChild());
434                 RllType::Ptr element = expression(elementNode,scope);
435                 if (!element) {
436                     INVALID(elementNode,scope);
437                 }
438                 else if (IntegerType::Ptr typedElement = boost::dynamic_pointer_cast<IntegerType>(element)) {
439                     IntegerType::Ptr index(new IntegerType(i));
440                     addBinaryOpInstruction<IntegerType,IntegerType,IntegerArrayType,arrayAssign>(index,typedElement,array,scope);
441                     isConstant = isConstant && typedElement->isConstant();
442                 }
443                 else {
444                     INVALID(elementNode,scope);
445                 }
446             }
447             ++i;
448         }
449         result = array;
450     }
451     else if (ast->getType() == UNARY_PLUS) {
452         // ignore
453     }
454     else if (ast->getType() == LOGICAL_AND &&
455              ast->getNumberOfChildren() == 2) {
456         AST lhs(ast->getFirstChild());
457         AST rhs(lhs->getNextSibling());
458         result = ternary(lhs,rhs,AST(),scope);
459     }
460     else if (ast->getType() == LOGICAL_OR &&
461              ast->getNumberOfChildren() == 2) {
462         AST lhs(ast->getFirstChild());
463         AST rhs(lhs->getNextSibling());
464         result = ternary(lhs,AST(),rhs,scope);
465     }
466     else if (ast->getType() == IF_EXPR &&
467              ast->getNumberOfChildren() == 3) {
468         AST ifTrue(ast->getFirstChild());
469         AST cond(ifTrue->getNextSibling());
470         AST ifFalse(cond->getNextSibling());
471         result = ternary(cond,ifTrue,ifFalse,scope);
472     }
473     else if ((ast->getType() == LITERAL_in ||ast->getType() == LITERAL_not) &&
474              ast->getNumberOfChildren() == 2) {
475         // handle arrays in diagram as special case
476         AST arrayNode(ast->getFirstChild());
477         AST diagramNode(arrayNode->getNextSibling());
478         RllType::Ptr arrayExp = expression(arrayNode,scope);
479         RllType::Ptr diagramExp = expression(diagramNode,scope);
480         if (arrayExp && diagramExp) {
481             IntegerArrayType::Ptr array = boost::dynamic_pointer_cast<IntegerArrayType>(arrayExp);
482             DiagramType::Ptr diagram = boost::dynamic_pointer_cast<DiagramType>(diagramExp);
483             if (array && diagram) {
484                 // dig into diagram node (which is really parsed as function call) to get arg string
485                 AST id(diagramNode->getFirstChild());
486                 AST args(id->getNextSibling());
487                 if (args->getNumberOfChildren() != 1) {
488                     ERROR("diagram membership must have one string argument", args,scope);
489                 }
490                 else {
491                     AST s(args->getFirstChild());
492                     if (EXPECTED(s,STRING_LITERAL,scope)) {
493                         BooleanType::Ptr b(new BooleanType());
494                         InstructionPtr inst(new DiagramMemberInstruction(diagram,array,s->getText(),b));
495                         scope.addInstruction(inst);
496                         if (ast->getType() == LITERAL_not) {
497                             addUnaryOpInstruction<BooleanType,BooleanType,lnot>(b,b,scope);
498                         }
499                         result = b;
500                     }
501                 }
502             }
503             else {
504                 // not special case, so dispatch
505                 result = dispatchOps(ast,scope);
506             }
507         }
508         else {
509             // not special case, so dispatch
510            result = dispatchOps(ast,scope);
511         }
512     }
513     else {
```

```
514         result = dispatchOps(ast,scope);
515     }
516     return result;
517 }
518 
519 RllType::Ptr
520 RllTreeWalker::functionCall(AST ast, Scope & scope)
521 {
522     RllType::Ptr result;
523 
524     AST id(ast->getFirstChild());
525     if (EXPECTED(id,ID,scope)) {
526         AST args(id->getNextSibling());
527         if ("min"== id->getText()) {
528             result = dispatchOps(AST(id->getNextSibling()),Dispatcher::FAKE_MIN,scope);
529         }
530         else if ("max"== id->getText()) {
531             result = dispatchOps(AST(id->getNextSibling()),Dispatcher::FAKE_MAX,scope);
532         }
533         else if ("abs"== id->getText()) {
534             result = dispatchOps(AST(id->getNextSibling()),Dispatcher::FAKE_ABS,scope);
535         }
536         else if ("binomial"== id->getText()) {
537             if (args->getNumberOfChildren() != 1) {
538                 ERROR("binomial distribution needs 1 argument",id,scope);
539             }
540             else {
541                 AST pNode(args->getFirstChild());
542                 DoubleType::Ptr p = EXPECTED_TYPE(pNode,expression(pNode,scope),DoubleType,scope);
543                 if (p) {
544                     BinomialRandomType::Ptr random(new BinomialRandomType(p));
545                     result = random;
546                 }
547             }
548         }
549         else if ("uniform"== id->getText()) {
550             if (args->getNumberOfChildren() != 2) {
551                 ERROR("uniform distribution needs 2 arguments",id,scope);
552             }
553             else {
554                 AST minNode(args->getFirstChild());
555                 AST maxNode(minNode->getNextSibling());
556                 IntegerType::Ptr min = EXPECTED_TYPE(minNode,expression(minNode,scope),IntegerType,scope);
557                 IntegerType::Ptr max = EXPECTED_TYPE(maxNode,expression(maxNode,scope),IntegerType,scope);
558                 if (min && max) {
559                     UniformRandomType::Ptr random(new UniformRandomType(min,max));
560                     result = random;
561                 }
562             }
563         }
564         else if ("poisson"== id->getText()) {
565             if (args->getNumberOfChildren() != 1) {
566                 ERROR("poisson distribution needs 1 argument",id,scope);
567             }
568             else {
569                 AST expectedNode(args->getFirstChild());
570                 IntegerType::Ptr expected = EXPECTED_TYPE(expectedNode,expression(expectedNode,scope),IntegerType,scope);
571                 if (expected) {
572                     PoissonRandomType::Ptr random(new PoissonRandomType(expected));
573                     result = random;
574                 }
575             }
576         }
577         else {
578             RllType::Ptr exp = scope.getSymbol(id->getText());
579             if (DiagramType::Ptr diagram = boost::dynamic_pointer_cast<DiagramType>(exp)) {
580                 result = diagram;
581             }
582             else {
583                 FunctionType::Ptr function =  EXPECTED_TYPE(id,exp,FunctionType,scope);
584 
585                 if (function) {
586                     int numArgs = 0;
587                     if (args && EXPECTED(args,ARGS,scope)) {
588                         numArgs = args->getNumberOfChildren();
589                     }
590                     if (function->numParameters() != numArgs) {
591                         std::ostringstream buffer;
592                             buffer << id->getText() << " has "<< function->numParameters()
593                                    << " but called with "<< args->getNumberOfChildren();
594                         ERROR(buffer.str(),args,scope);
```

```
595                     }
596                     else if (args) {
597                         for (AST arg(args->getFirstChild()); arg; arg = AST(arg->getNextSibling())) {
598                             if (arg->getType() == ASSIGN) {
599                                 // don't handle named function arguments yet
600                                 TODO(arg,scope);
601                             }
602                             else {
603                                 TODO(arg,scope);
604                             }
605                         }
606                     }
607                     result = function->getReturn();
608                     InstructionPtr call(new FunctionCall(function));
609                     scope.addInstruction(call);
610                 }
611             }
612         }
613     }
614     return result;
615 }
616
617 bool
618 RllTreeWalker::assign(AST ast, Scope & scope) {
619     bool result = false;
620     AST lhsNode(ast->getFirstChild());
621     AST rhsNode(lhsNode->getNextSibling());
622     RllType::Ptr rhs = expression(rhsNode,scope);
623     if (!rhs) {
624         INVALID(rhsNode,scope);
625     }
626     else {
627         if (lhsNode->getType() == INDEX) {
628             AST arrayNode(lhsNode->getFirstChild());
629             AST indexNode(arrayNode->getNextSibling());
630             IntegerArrayType::Ptr array = EXPECTED_TYPE(arrayNode,expression(arrayNode,scope),IntegerArrayType,scope);
631             IntegerType::Ptr index = EXPECTED_TYPE(indexNode,expression(indexNode,scope),IntegerType,scope);
632             if (array && index) {
633                 if (IntegerType::Ptr typedRhs = boost::dynamic_pointer_cast<IntegerType>(rhs)) {
634                     addBinaryOpInstruction<IntegerType,IntegerType,IntegerArrayType,arrayAssign>(index,typedRhs,array,scope);
635                     result = true;
636                 }
637                 else if (DoubleType::Ptr typedRhs = boost::dynamic_pointer_cast<DoubleType>(rhs)) {
638                     addBinaryOpInstruction<IntegerType,DoubleType,IntegerArrayType,arrayAssign>(index,typedRhs,array,scope);
639                     result = true;
640                 }
641                 else {
642                     ERROR("Only handle assigning to int array elements",ast,scope);
643                 }
644             }
645         }
646         else {
647             // not an array element on lhs so do plain dispatch
648             RllType::Ptr lhs = expression(lhsNode,scope);
649             if (!lhs) {
650                 if (lhsNode->getType() == ID) {
651                     // create variable
652                     lhs = rhs->newSameType();
653                     if (!scope.addSymbol(lhsNode->getText(),lhs)) {
654                         ERROR("Can't resolve or create symbol",lhsNode,scope);
655                     }
656                 }
657                 else {
658                     ERROR("Can't resolve or create",lhsNode,scope);
659                 }
660             }
661             if (lhs) {
662                 result = _dispatcher.dispatchInstructions(rhs,lhs,ASSIGN,scope);
663             }
664         }
665     }
666     return result;
667 }
668
669 RllType::Ptr
670 RllTreeWalker::ternary(AST test, AST ifTrue, AST ifFalse, Scope & scope)
671 {
672     RllType::Ptr result;
673
674     TernaryType::Ptr ternary(new TernaryType(scope));
675
```

```
676     // always evaluate condition in top-level scope
677     BooleanType::Ptr condition = EXPECTED_TYPE(test,expression(test,scope),BooleanType,scope);
678     if (condition) {
679         BooleanType::Ptr param = ternary->getCondition();
680         addUnaryOpInstruction<BooleanType,BooleanType,::assign>(condition,param,scope);
681         RllType::Ptr trueResult;
682         RllType::Ptr falseResult;
683         if (ifTrue) {
684             trueResult = expression(ifTrue,ternary->getTrueScope());
685         }
686         if (ifFalse) {
687             falseResult = expression(ifFalse,ternary->getFalseScope());
688         }
689 
690         if (trueResult && !falseResult) {
691             RllType::Ptr result = trueResult->newSameType();
692             ternary->setReturn(result);
693             _dispatcher.dispatchInstructions(trueResult,result,ASSIGN,ternary->getTrueScope());
694             _dispatcher.dispatchInstructions(param,result,ASSIGN,ternary->getFalseScope());
695         }
696         else if (!trueResult && falseResult) {
697             RllType::Ptr result = falseResult->newSameType();
698             ternary->setReturn(result);
699             _dispatcher.dispatchInstructions(param,result,ASSIGN,ternary->getTrueScope());
700             _dispatcher.dispatchInstructions(falseResult,result,ASSIGN,ternary->getFalseScope());
701         }
702         if (trueResult && falseResult) {
703             //make sure we have common type
704             if (typeid(trueResult) != typeid(falseResult)) {
705                 ERROR("must have same types, not"+ std::string(typeid(trueResult).name())+ " "+std::string(typeid(falseResult).name()),test,sc
706             }
707             else {
708                 RllType::Ptr result = trueResult->newSameType();
709                 ternary->setReturn(result);
710                 _dispatcher.dispatchInstructions(trueResult,result,ASSIGN,ternary->getTrueScope());
711                 _dispatcher.dispatchInstructions(falseResult,result,ASSIGN,ternary->getFalseScope());
712             }
713         }
714         result = ternary->getReturn();
715         InstructionPtr call(new TernaryCall(ternary));
716         scope.addInstruction(call);
717     }
718     return result;
719 }
720 
721 RllType::Ptr
722 RllTreeWalker::dispatchOps(AST ast, Scope & scope)
723 {
724     return dispatchOps(ast, ast->getType(), scope);
725 }
726 
727 RllType::Ptr
728 RllTreeWalker::dispatchOps(AST ast, int type, Scope & scope)
729 {
730     RllType::Ptr result;
731 
732     if (ast->getNumberOfChildren() == 1) {
733         AST child(ast->getFirstChild());
734         RllType::Ptr in = expression(child,scope);
735         if (!in) {
736             INVALID(child,scope);
737         }
738         else {
739             result = _dispatcher.dispatchOps(in,type,scope);
740             if (!result) {
741                 std::ostringstream buffer;
742                 buffer << "Can't dispatch "<< type << " for "<< typeid(*in).name();
743                 ERROR(buffer.str(),child,scope);
744             }
745         }
746     }
747     else if (ast->getNumberOfChildren() == 2) {
748         AST child(ast->getFirstChild());
749         RllType::Ptr lhs = expression(child,scope);
750         if (!lhs) {
751             INVALID(child,scope);
752         }
753         else {
754             child = AST(child->getNextSibling());
755             RllType::Ptr rhs = expression(child,scope);
756             if (!rhs) {
```

```
757                 INVALID(child,scope);
758             }
759             else {
760                 result = _dispatcher.dispatchOps(lhs,rhs,type,scope);
761                 if (!result) {
762                     std::ostringstream buffer;
763                     buffer << "Can't dispatch "<< type << " for "<< typeid(*lhs).name() << " "<< typeid(*rhs).name();
764                     ERROR(buffer.str(),child,scope);
765                 }
766             }
767         }
768     }
769     else {
770         INVALID(ast,scope);
771     }
772 
773     return result;
774 }
775 
776 
777 void
778 RllTreeWalker::error(std::string const& file, int fileLine, std::string const& message, int sourceLine, Scope & scope)
779 {
780     scope.reportError(file,fileLine,message,sourceLine);
781 }
782 
783 bool
784 RllTreeWalker::expected(std::string const& file, int fileLine, AST ast, int expected, std::string const& expectedString, Scope
& scope) {
785     bool result = ast;
786     if (!result) {
787         error(file, fileLine, "expecting "+expectedString+" in mdp tree structure, not null", ast->getLine(),scope );
788     }
789     else {
790         result = ast->getType() == expected;
791         if (!result) {
792             error(file, fileLine, "expecting "+expectedString+" in mdp tree structure, not "+ast->getText(), ast->getLine(),scope
);
793         }
794     }
795     return result;
796 }
797 
798 template <typename Type>
799 typename Type::Ptr
800 RllTreeWalker::expectedType(std::string const& file, int fileLine, AST ast, RllType::Ptr ptr, Scope & scope)
801 {
802     if (!ptr) {
803         error(file, fileLine, "expecting "+std::string(typeid(Type).name())+" in mdp tree structure "+ast->getText()+", not
null pointer", ast->getLine(),scope );
804         return typename Type::Ptr();
805     }
806     else {
807         typename Type::Ptr result = boost::dynamic_pointer_cast<Type>(ptr);
808         if (!result) {
809             error(file, fileLine, "expecting "+std::string(typeid(Type).name())+" in mdp tree structure "+ast->getText()+",
not "+std::string(typeid(*ptr).name()), ast->getLine(),scope );
810         }
811         return result;
812     }
813 }
814 
815 RllTreeWalker::~RllTreeWalker()
816 {
817 }
```

44 Scope.cpp

```
1 #include <iostream>
2 
3 #include "Scope.hpp"
4 #include "Types.hpp"
5 #include "util.hpp"
6 
7 
8 Scope::Scope(std::string const& name)
9   :  _parent(0)
10   , _name(name)
11   , _symbols()
```

```
12    , _literals()
13 //  , _registers()
14    , _instructions()
15    , _errors()
16 {
17 }
18
19 Scope::Scope(Scope* parent, std::string const& name)
20    :  _parent(parent)
21    , _name(name)
22    , _symbols()
23    , _literals()
24 //  , _registers()
25    , _instructions()
26    , _errors()
27 {
28     if (_name.size() == 0 && _parent) {
29         _name = _parent->_name;
30     }
31 }
32
33 std::string const&
34 Scope::getName() const
35 {
36     return _name;
37 }
38
39
40 TypePtr
41 Scope::getSymbol(std::string const& s)
42 {
43     TypePtr result;
44     NamedTypeMap::iterator i = _symbols.find(s);
45
46     if (i != _symbols.end()) {
47         result = i->second;
48     }
49     else if (_parent) {
50         result = _parent->getSymbol(s);
51     }
52     return result;
53 }
54
55 //void
56 //Scope::addTemporary(TypePtr temp)
57 //{
58 //    _registers.push_back(temp);
59 //}
60
61 bool
62 Scope::addSymbol(std::string const& name, TypePtr symbol, bool okToEclipse)
63 {
64     bool result = false;
65     if (okToEclipse ||!_parent ||!_parent->getSymbol(name)) {
66         result = insertIfUnique(_symbols,name,symbol);
67         if (result) {
68             symbol->setName(name);
69         }
70     }
71     return result;
72 }
73
74 bool
75 Scope::overwriteSymbol(std::string const& name, TypePtr symbol)
76 {
77     symbol->setName(name);
78     _symbols[name] = symbol;
79     return true;
80 }
81
82 void
83 Scope::addInstruction(InstructionPtr instruction)
84 {
85     _instructions.push_back(instruction);
86 }
87
88 void
89 Scope::evaluateInstructions() const
90 {
91     for (InstructionBlock::const_iterator i = _instructions.begin(); i != _instructions.end(); ++i) {
92         (**i)();
```

```
    }
}

Scope
Scope::makeChildScope(std::string const& name) {
    return Scope(this,name);
}

void
Scope::reportError(std::string const& implementationFile, int implementationLine, std::string const& message, int sourceLine)
{
    reportError(implementationFile, implementationLine, message, _name, sourceLine);
}

void
Scope::reportError(std::string const& implementationFile, int implementationLine, std::string const& message, std::string
const& scope, int sourceLine)
{
    if (_parent) {
        _parent->reportError(implementationFile, implementationLine, message, scope, sourceLine);
    }
    else {
        SemanticError error(implementationFile, implementationLine, message, scope, sourceLine);
        _errors.push_back(error);
        std::cout << error << std::endl;
    }
}

bool
Scope::hasErrors() {
    if (_parent) {
        return _parent->hasErrors();
    }
    else {
        return _errors.size() > 0;
    }
}

Scope::ErrorList const&
Scope::getErrors()
{
    return _errors;
}

Scope::~Scope()
{
}
```

45 SemanticError.cpp

```
#include "SemanticError.hpp"

SemanticError::SemanticError(std::string const& implementationFile, int implementationLine, std::string const& message, std::string
const& scope, int sourceLine)
  :  _implementationFile(implementationFile)
  , _implementationLine(implementationLine)
  , _message(message)
  , _scope(scope)
  , _sourceLine(sourceLine)
{
}

std::string const&
SemanticError::getImplementationFile() const
{
    return _implementationFile;
}

int
SemanticError::getImplementationLine() const
{
    return _implementationLine;
}

std::string const&
SemanticError::getScope() const
{
    return _scope;
}
```

```
29
30 std::string const&
31 SemanticError::getMessage() const
32 {
33     return _message;
34 }
35
36 int
37 SemanticError::getSourceLine() const
38 {
39     return _sourceLine;
40 }
41
42 std::ostream & operator<<(std::ostream & s, SemanticError const& error)
43 {
44    return s << error.getScope() << ":"<< error.getSourceLine() << " "
45             << error.getMessage() << " found at "
46             << error.getImplementationFile() << ":"<< error.getImplementationLine();
47 }
48
```

46 Types.cpp

```
1 #include <sstream>
2 #include "Types.hpp"
3 #include "Exceptions.hpp"
4
5 int RllType::_instances = 0;
6
7 RllType::RllType(bool isConstant)
8   :  _isConstant(isConstant)
9   , _name()
10 {
11     std::ostringstream buffer;
12     buffer << "temp"<< _instances;
13     _name = buffer.str();
14     ++_instances;
15 }
16
17 RllType::~RllType()
18 {
19 }
20
21 bool
22 RllType::isConstant() const
23 {
24     return _isConstant;
25 }
26
27 void
28 RllType::makeConstant()
29 {
30     if (_isConstant) {
31         throw ModifyConstantException("Tried to reset a constant");
32     }
33     _isConstant = true;
34 }
35
36 void
37 RllType::setName(std::string const& name)
38 {
39     _name = name;
40 }
41
42 std::string const&
43 RllType::name() const
44 {
45     return _name;
46 }
47
48 RllType::Ptr
49 RllType::newSameType() const
50 {
51     throw UnimplementedOperationException("Trying to clone base RllType");
52 }
53
54 FunctionType::FunctionType(Scope& parent)
55   :  _parent(parent)
56   , _scope(parent.makeChildScope())
```

```
57    , _return()
58    , _parameters()
59 {
60 }
61
62 bool
63 FunctionType::addParameter(std::string const& name, RllType::Ptr type)
64 {
65     bool result = _scope.addSymbol(name, type, true);
66
67     if (result) {
68         _parameters.push_back(type);
69     }
70     return result;
71 }
72
73 bool
74 FunctionType::setReturn(RllType::Ptr type)
75 {
76     bool result = true;
77
78     if (!_return) {
79         _return = type;
80     }
81     else {
82         // return can't be redefined
83         result = false;
84     }
85
86     return result;
87 }
88
89
90 RllType::Ptr
91 FunctionType::getParameter(std::size_t position)
92 {
93     RllType::Ptr result;
94
95     if (position < _parameters.size()) {
96         result = _parameters[position];
97     }
98
99     return result;
100 }
101
102 std::size_t
103 FunctionType::numParameters() const
104 {
105     return _parameters.size();
106 }
107
108 //RllType::Ptr
109 //FunctionType::getArgument(std::string const& name) {
110 //    return _scope.getSymbol(name);
111 //}
112
113
114 RllType::Ptr
115 FunctionType::getReturn() {
116     return _return;
117 }
118
119 Scope&
120 FunctionType::getScope() {
121     return _scope;
122 }
123
124 void
125 FunctionType::execute() const {
126     _scope.evaluateInstructions();
127 }
128
129 RllType::Ptr
130 FunctionType::newSameType() const
131 {
132     FunctionType::Ptr result(new FunctionType(_parent));
133     return result;
134 }
135
136
137 FunctionType::~FunctionType()
```

```
{}


TernaryType::TernaryType(Scope& parent)
  :  _parent(parent)
  , _trueScope(parent.makeChildScope())
  , _falseScope(parent.makeChildScope())
  , _return()
  , _condition(new BooleanType())
{
}

bool
TernaryType::setReturn(RllType::Ptr type)
{
    bool result = true;

    if (!_return) {
        _return = type;
    }
    else {
        // return can't be redefined
        result = false;
    }

    return result;
}

RllType::Ptr
TernaryType::getReturn()
{
    return _return;
}

BooleanType::Ptr
TernaryType::getCondition()
{
    return _condition;
}


Scope&
TernaryType::getTrueScope()
{
    return _trueScope;
}

Scope&
TernaryType::getFalseScope()
{
    return _falseScope;
}

void
TernaryType::execute() const
{
    if (_condition->get()) {
        _trueScope.evaluateInstructions();
    }
    else {
        _falseScope.evaluateInstructions();
    }
}


RllType::Ptr
TernaryType::newSameType() const {
    return RllType::Ptr(new TernaryType(_parent));
}


TernaryType::~TernaryType()
{}

RandomType::RandomType()
  :  _return(new IntegerType())
{}


IntegerType::Ptr
```

```
219 RandomType::getReturn()
220 {
221     return _return;
222 }
223
224 void
225 RandomType::create(DiscreteRandom::RandomGenerator & generator)
226 {
227     throw UnimplementedOperationException("Trying to create base RandomType");
228 }
229
230 void
231 RandomType::draw()
232 {
233     throw UnimplementedOperationException("Trying to create base RandomType");
234 }
235
236 RandomType::~RandomType()
237 {}
238
239 BinomialRandomType::BinomialRandomType(DoubleType::Ptr p)
240   :  RandomType()
241   , _p(p)
242   , _boolResult(new BooleanType())
243   , _binomial()
244 {}
245
246 void
247 BinomialRandomType::create(DiscreteRandom::RandomGenerator & generator)
248 {
249     if (_binomial) {
250         throw std::logic_error("Called create twice on binomial random type");
251     }
252     else {
253         _binomial = boost::shared_ptr<BinomialRandom>(new BinomialRandom(generator,_p->get()));
254     }
255 }
256
257 void
258 BinomialRandomType::draw()
259 {
260     if (_binomial) {
261         _return->set((*_binomial)());
262         _boolResult->set(_return->get());
263     }
264     else {
265         throw std::logic_error("attempting to draw random variable from uncreated distribution");
266     }
267 }
268
269 BooleanType::Ptr
270 BinomialRandomType::getBoolReturn() {
271     return _boolResult;
272 }
273
274 boost::shared_ptr<DiscreteRandom>
275 BinomialRandomType::distribution()
276 {
277     return _binomial;
278 }
279
280
281 BinomialRandomType::~BinomialRandomType()
282 {
283 }
284
285 UniformRandomType::UniformRandomType(IntegerType::Ptr min, IntegerType::Ptr max)
286   :  RandomType()
287   , _min(min)
288   , _max(max)
289   , _uniform()
290 {}
291
292 void
293 UniformRandomType::create(DiscreteRandom::RandomGenerator & generator)
294 {
295     if (_uniform) {
296         throw std::logic_error("Called create twice on uniform random type");
297     }
298     else {
299         _uniform = boost::shared_ptr<UniformRandom>(new UniformRandom(generator,_min->get(),_max->get()));
```

```
300     }
301 }
302
303 void
304 UniformRandomType::draw()
305 {
306     if (_uniform) {
307         _return->set((*_uniform)());
308     }
309     else {
310         throw std::logic_error("attempting to draw random variable from uncreated distribution");
311     }
312 }
313
314 boost::shared_ptr<DiscreteRandom>
315 UniformRandomType::distribution()
316 {
317     return _uniform;
318 }
319
320 UniformRandomType::~UniformRandomType()
321 {
322 }
323
324
325 PoissonRandomType::PoissonRandomType(IntegerType::Ptr expected)
326   :  RandomType()
327   , _expected(expected)
328   , _poisson()
329 {}
330
331 void
332 PoissonRandomType::create(DiscreteRandom::RandomGenerator & generator)
333 {
334     if (_poisson) {
335         throw std::logic_error("Called create twice on poisson random type");
336     }
337     else {
338         _poisson = boost::shared_ptr<PoissonRandom>(new PoissonRandom(generator,_expected->get()));
339     }
340 }
341
342 void
343 PoissonRandomType::draw()
344 {
345     if (_poisson) {
346         _return->set((*_poisson)());
347     }
348     else {
349         throw std::logic_error("attempting to draw random variable from uncreated distribution");
350     }
351 }
352
353
354 boost::shared_ptr<DiscreteRandom>
355 PoissonRandomType::distribution()
356 {
357     return _poisson;
358 }
359
360 PoissonRandomType::~PoissonRandomType()
361 {
362 }
```

47 UniformRandom.cpp

```
1  #include "UniformRandom.hpp"
2
3  UniformRandom::UniformRandom(RandomGenerator& generator, int min, int max)
4    :  DiscreteRandom(generator,IntervalSet::Interval(min,max))
5    , _uniformGenerator(_numberGenerator,UniformDistribution(min,max))
6  {
7  }
8
9  int
10 UniformRandom::operator()()
11 {
12     return _uniformGenerator();
13 }
```

```
14
15 double
16 UniformRandom::probability(int i) const
17 {
18     double probability = 0.0;
19     int min = _uniformGenerator.distribution().min();
20     int max = _uniformGenerator.distribution().max();
21     if ( i >= min && i <= max ) {
22         probability = 1.0 / (1.0 + max - min);
23     }
24     return probability;
25 }
26
27
28 UniformRandom::~UniformRandom()
29 {
30 }
31
```