Empath: A language for modeling digital pets

Jeremy Posner <jp2288@columbia.edu>
Nalini Kartha <nkk2106@columbia.edu>
Sampada Sonalkar <ss3119@columbia.edu>
William Mee <wjm2107@columbia.edu>

COMS 4115 Programming Languages and Translators

December 19, 2006

Contents

1 Introduction

The Empath language is designed for modeling digital pets. The pets which could be modeled include single entities, like a Tamagotchi-style creature, a virtual pet cat or a simulated human character in a game. In addition, entire ecosystems, such as a simulated garden, beehive or an ocean are possible. The Empath models are based on characteristics which vary over time, as well as discrete states in finite state automata.

1.1 Motivation

The Empath language is designed to allow rapid prototyping, development and testing of pet models and their interaction with human users. It is a domain-specific language with special constructs which make achievement of these objectives easier and more concise than programming them in a general-purpose language. The language abstracts the implementation how-tos from the pet designer and allows him/her to focus on the pet modeling. It includes necessary constructs to support its goal, such as time flow and mode switches based on conditions on the pet characteristics.

Empath allows users to interact with the pets through simple, programmer-defined actions, such as ”feed” or ”put to sleep.” The language also allows the programmer to create growth and decay formulae to define how a pet's needs change over time. Using Empath, a game designer can quickly answer questions like how robust is a character in a game? how interesting or fun is a virtual pet to interact with?

Empath is back-ended by the Java language, and is therefore usable not only on desktop computers, but on smaller hand-held systems as well. Empath supports both the definition of pet models as well as their execution. Although the user interface is presently a simple command line interface, the language offers an API for plugging in more complex GUI.

Empath is targeted at the game and broader entertainment industry. Virtual pets, for example, are big business. Bandai, the company behind the Tamagotchi and Digimon games, has already sold over 20 million units of its Tamagotchi Plus , and in 2005 recorded net sales of over 2 billion US dollars.

1.2 Concepts and Definitions

Within the context of Empath, we define an `entity` as a simplified model of a living pet.

Entities maintain a set of internal characteristics which indicate important attributes of the pet. Characteristics might, for example, indicate a hunger level, happiness, age, health level or simply a name. They can vary in a defined way over time, or as a result of external events (see below). Simple characteristics have a defined type, such as boolean flags, strings, ranges etc. Since characteristics can be hidden from the end users view, hidden characteristics can be a powerful means of creating complex internal processes for an entity.

A more complex type of a characteristic is a `state`. This is a reference to a finite state automaton with defined transitions between the states. States represent a mode that an entity could be in. A mode could be a stage in its life cycle, an activity in its daily routine, a state of its health, etc. For an entity modeling a frog, an important state would indicate whether the frog is currently an egg, a tadpole or an adult frog. Transitions between states represent mode switches. They are triggered as a result of a combination of other characteristics which reach defined threshold levels. For example, given the correct combination of warmth and time, the frog's state will change from egg to tadpole. While an entity can only be in one base state, a state can itself then be defined as a set of states, so that within the state of being an adult frog, the entity could be hopping, sleeping, eating or croaking. The concept of states are described in more detail in the section on finite state automata below.

Entities interact with their environment via `events`. These events can originate from the environment (such as the simple flow of time) or from a user interacting with the entity. Eventually the language may be expanded to support interaction between entities. An event could have some effect on one or more of the entity's characteristics; so stroking the virtual cat would increase its happiness level, possibly triggering it into the purring state (or into the scratching state; cats being non-deterministic). An event could be instantaneous (such as a feeding) or could occur over a period of time (such as getting tired over the course of a day).

1.3 Finite State Automata

Important characteristics of an entity, such as its life cycle, are modeled using finite state Mealy automata. A Mealy automaton is a set of states and transitions, with a single defined initial state.

The life cycle of an entity can typically be modeled as stages in its life, and additional activities in a particular stage. It is necessary to separate these in order to achieve a simplified intuitive design. We have introduced the notion of hierarchy of automata for this purpose. An entire finite state automaton can be embedded in a state of an automaton at a higher level. This nesting can be done to any level, although it is anticipated that 2-4 levels of nesting would be sufficient to describe the most complex of entities.

Semantically, when the top level parent state is active, one of the states in the child automaton is also active. If the parent state is inactive, the child automaton is also inactive. There is no notion of history viz, whenever the parent state is entered, the initial state of the child automaton is activated. Inter-level transitions can occur from a sub-state of one high level state directly to another high level state. Hence, the system does not require that all of the embedded automata go to the same depth. For example, an egg state may only have two states (waiting to hatch and hatching), while the upper level state into which the hatched egg transitions may have far more depth of detail.

The execution semantics of the automata is an instantaneous reaction to events to produce output actions and change state. The following are considered valid events for the transition firing:

1. user input events
2. reaching defined characteristic thresholds
3. boolean expressions formed from the above using and, or, and not operators

In response to user input events, the appropriate characteristics of the entity are modified.

States have onEntry() and onExit() functions for performing certain initializations and resets while entering/exiting the state. While the state is active, the onClockTick() function is executed at every clock cycle. When a transition fires, the onExit() of the source state is executed followed by the onEntry() of the destination state.

In a hierarchy, transitions are evaluated outside in and with preemption(if multiple transitions are simultaneously enabled). A happy puppy that becomes hungry at the same clock tick when it is supposed to grow into a dog will only go into the adult dog state and will not go into the hungry puppy state as the transition to dog state(at higher level) is given higher priority over the transition to the hungry puppy state(at lower level). Hence the transition from puppy to dog preempts the transition from happy puppy to hungry puppy, as the hunger level of the puppy becomes an obsolete characteristic.

1.4 Language Outline

Empath uses syntax based on that of C and Java. As with those languages, whitespace is generally unimportant, and blocks of code are enclosed in braces. Basic math functions (addition, subtraction, multiplication, division, modulo) and logic functions (and, or, not, etc.) are evaluated using symbols and syntax rules similar to those of Java. Additionally, flow control operators (if, while, for) are also set up to use syntax similar to that of Java. The idea is to create an environment that is relatively familiar to those who already know Java and C, reducing the learning curve for programmers transitioning to the new language. This syntactic style also has the advantage of being proven as extremely flexible for both simple and complex programming projects.

1.5 Limitations and Restriction

We have made several important simplifying limitations for Empath. Although it is feasible for entities to interact with each other, we do not consider this in the initial version of Empath. Furthermore, Empath only supports the use of a single automaton at any one level (although several automata can be nested).

1.6 Future Directions

Although the base Empath language includes only limited functionality, there is the potential for programmers to release libraries to allow for significantly increased capabilities. For example, although the language has no built-in capacity for artificial intelligence, the language does include sufficient math support to provide the foundation for a set of AI libraries. Since the language can handle multiple status values for a character, they can be set up to interact with each other in ways that allow for the programmer to define all sorts of hidden behavior, including the sorts of processes necessary for AI. Other libraries might include support for more sophisticated math and graphical interfaces. Because living pets are so complex, the capacity for creating them with Empath is limited only by the imagination of the programmers.

Another important area of future work would be to allow interaction between entities to be defined in the language. Networked toys, such as the new generation of the Tamagotchi or Nintendo's Nintendogs , are becoming increasingly popular.

2 Language Tutorial

An empath program consists of an entity specification which contains the variable declarations, function definitions, state declarations, transition definitions and nested state definitions.

2.1 A Simple Example

The state transition diagram for a simple frog is shown in Figure 1. The frog starts out in the egg state. On every clock cycle the age of the frog is internally increased. When the age reaches a specified threshold value, (defined as a trigger `hatch()`), the frog entity hatches out of its egg and becomes a tadpole. When the age of the tadpole reaches a threshold(defined as a trigger `isMature()`), then the tadpole matures into an Adult Frog. The Adult Frog finally dies when it reaches the age of 7.

A possible specification of this frog entity in an empath program is shown below-

```
/* this is a simple example of a frog entity
*/
entity BullFrog {

   int age = 0;

   function void onClockTick() {
```

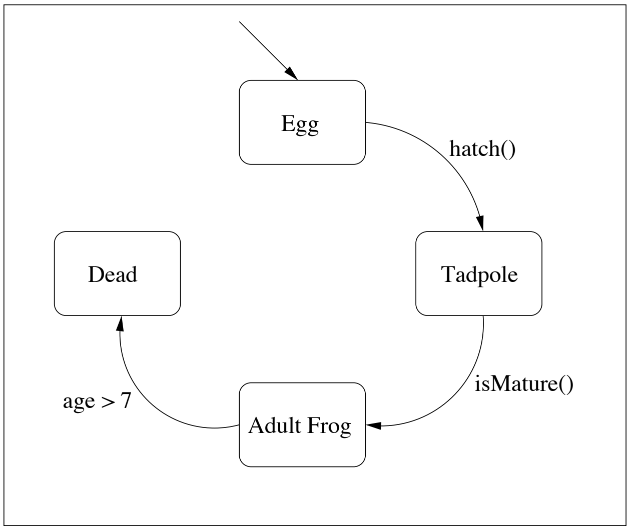


Figure 1: Lifecycle of a frog

```
   age++;
}

//condition on which the egg hatches into a tadpole
trigger hatch() {
   if (age > 2) {
      return true;
   }
   return false;
}

//condition on which the tadpole changes to a frog
trigger isMature() {
   if (age > 5) {
      return true;
   }
   return false;
}

state Egg, Tadpole, AdultFrog, Dead;
```

```
    transition Egg to Tadpole if (hatch());
    transition Tadpole to AdultFrog if (isMature());
    transition AdultFrog to Dead if (age > 7);

}
```

The frog characteristic variable `age` is defined first. Following this are the function definitions. The body of each function contains regular C or Java style statements. `onClockTick()` is a special function that defines what changes are to be made to the entity variables at every clock cycle(here the age of the frog is incremented).

The trigger functions `hatch()` and `isMature()` are used to transition between states. The special trigger functions by default must return boolean true or false and hence the return type is not part of the function definition. Also, since triggers will be evaluated on every clock cycle(transition conditions are evaluated every clock cycle to check if a new state must be entered), the values of characteristic variables of the entity or sub-state must not be changed in the body of a trigger function.

Following this is the list of states that the frog can be in. The first state in the list is taken to be the default state that the frog goes into at the beginning of execution. The list of transitions from Egg to Tadpole, from Tadpole to AdultFrog. etc are finally defined.

2.2 A More Complex Example

Next let us consider a more complex example of modeling a graduate student. The graduate student is in one of three states primarily - `Slaving` (symbolic of the daily routine during the course of the semester), `ChillingOut` (symbolic of the routine during breaks between semesters) or `Dead` (symbolic of the graduate student not getting fed often enough or failing to keep up with the course load). The `Slaving` state consists of a set of sub-states as shown below. Similarly, the `ChillingOut` state is also a complex state containing two sub-states. A possible representation of this entity in an Empath program is given below.

```
entity Grad_student {
```

```
/* Following are the characteristic variables of
*  the set of states at the highest(entity) level
*/
range [0:10] hunger = 0;
int assignments_due = 0;
boolean needs_a_break = false;

//This function will get executed every clock cycle
function void onClockTick() {
   hunger++;
   /*
   * The boolean expression in the following if
   * conditional evaluates to true for every 3rd clock cycle
   */
   if(tick 3) {
      assignments_due ++;
   }
}

//Events are special functions that specify a change on user input
event feed() {
   hunger -= 2;
}

event work() {
   assignments_due--;
}

event chill() {
   needs_a_break = true;
}

function void bury() {
   //sob sob :'(
}

state Slaving, ChillingOut, Dead;
```

```
transition
      Slaving to ChillingOut if(needs_a_break),
      ChillingOut to Slaving if(assignments_due >= 5);

//Nested definition of the Slaving state
Slaving   {

   boolean wake_up_alarm = false;

   event wake_up() {
      wake_up_alarm = true;
   }

   //A trigger which returns true if the student is overloaded
   trigger overload() {
      if((assignments_due == 10)||(hunger@max))
         return true;
      else
         return false;
   }

   state Sleeping, NeedToWork , DozingInClass , Hungry;

   transition
         Sleeping to DozingInClass if(wake_up_alarm),
         Sleeping to NeedToWork if (assignments_due >= 5),
         Sleeping to Hungry if (hunger>=8),
         DozingInClass to Hungry if (hunger >= 8),
         DozingInClass to NeedToWork if (assignments_due >= 5),
         Hungry to NeedToWork if (assignments_due >= 5),
         Hungry to Sleeping if (hunger <= 7),
         NeedToWork to Sleeping if (assignments_due <= 5),
         * to Dead if (overload()) / bury();


   DozingInClass {

      /*
```

```
         *  Special function onEntry which gets executed each time the
         *  DozingInClass state becomes the current active state
         */
         function void onEntry() {
            wake_up_alarm = false;
         }

      }

   }

   //Nested definition of state ChillingOut

   ChillingOut {

      int time = 0;

      function void onClockTick() {
         time += 10;
      }

      function void onEntry() {
         needs_a_break = false;
      }

      state Partying , init Comatose;

      transition Partying to Comatose if(time%7 == 0);
      transition Comatose to Partying if(time%10 == 0);

   }
}
```

The nested state definitions must be specified after the transition definitions of the parent state. The structure of the nested state definitions is identical to that of the parent entity.

An event is a function that becomes visible to the user as an input option. When the user chooses to execute that event, the code specified in the event function body is executed.

`onEntry()` and `onExit()` are special functions that are executed when the state in which they are defined is entered or exited respectively. They are generally used to specify initialization and clean-up operations needed for variable associated with a state.

In the `ChillingOut` state, we have used the `init` keyword to explicitly specify the default state of this automaton.

In the last transition defined in the `Slaving` state, the '`*`' wildcard operator is a shorthand for representing all the sub-states in the current state definition. Also we can specify an action function that must be executed when a transition is taken as shown in the transition to the Dead state. The action function `bury()` follows an optional '`/`' in the transition definition.

3 Language Reference Manual

3.1 Grammar notation

We use standard regular expression for expressing the production rules of the language. Symbols enclosed in single quotes are terminals. All other symbols are non-terminals. * represents zero or more occurrences of the preceding expression and + represents one or more occurrences. | represents a choice between the two expressions on either side of the | character and ? represents that the preceding expression is optional.

3.2 Lexical conventions

Comments

The // character combination represents a single line comment. For multi-line comments the /* character combination indicates start of comment block and */ character combination indicates end of comment block. The symbol sequence */ cannot appear within a multi-line comment.

Tokens

Tokens include identifiers, keywords, constants, operators, and punctuation.

Identifiers

Identifiers are represented by a sequence of letters and digits starting with an alphabetic character or an underscore character (‘_’). An identifier represents a name for one of the following constructs: variable, function, event, trigger, state.

digit → *(‘0’ .. ‘9’)*
letter → *(‘a’ .. ‘z’ | ‘A’ .. ‘Z’)*
id → *(‘_’ | letter) (‘_’ | letter | digit)**

Keywords

The following identifiers are reserved keywords and may not be used otherwise The keywords may only be used in the lower case alphabet. The language is case sensitive, so an identifier age is distinct from AGE or Age. The Java convention of identifiers starting with lower case alphabet and judicious use of upper case characters as separators is encouraged.

`boolean`	`entity`	`event`	`float`	`for`
`if`	`init`	`int`	`label`	`output`
`range`	`return`	`state`	`string`	`tick`
`to`	`transition`	`trigger`	`void`	`while`

Operators

The language supports a small set of mathematical and logical operators for manipulating entity characteristics. The precedence and associativity rules for these operators are the same as those of programming languages, hence we will not elucidate them in this manual.

Punctuation

The following characters are used for punctuation in the language

<table>
<tr><td>Arithmetic operators</td><td>+</td><td>−</td><td>*</td><td>/</td><td>%</td><td></td></tr>
<tr><td>Shorthand arithmetic operators</td><td>+ =</td><td>− =</td><td>∗ =</td><td>/=</td><td></td><td></td></tr>
<tr><td>Relational operators</td><td>&lt;</td><td>&gt;</td><td>&lt;=</td><td>&gt;=</td><td>==</td><td>! =</td></tr>
<tr><td>Logical operators</td><td>&amp;&amp;</td><td>||</td><td></td><td></td><td></td><td></td></tr>
</table>

<table>
<tr><td>;</td><td>statement end</td></tr>
<tr><td>,</td><td>argument list separator</td></tr>
<tr><td>=</td><td>declaration initializer</td></tr>
<tr><td>””</td><td>string literal</td></tr>
<tr><td>:</td><td>range specifier</td></tr>
<tr><td>[ ]</td><td>range delimiter</td></tr>
<tr><td>{ }</td><td>state definition, function body or block statement delimiter</td></tr>
<tr><td>( )</td><td>function parameter list delimiter</td></tr>
</table>

3.2.1 Constants

The language defines integer constants, real constants (including only a fractional part, no exponents), boolean constants and string constants

constant → intConst | realConst | boolConst | strConst
intConst → (‘+’ | ‘-’)? (digit)+
realConst → (+ | -)? (digit)* ‘ . ’ (digit)+
boolConst → (true | false)
strConst → ‘ " ’ (letter | digit | specialChar)* ‘ " ’
specialChar → ‘!’ | ‘#’ | ‘%’ | ‘ˆ’ | ‘&’ | ‘*’ | ‘()’ | ‘)’ | ‘–’ | ‘_’
| ‘=’ | ‘+’ | ‘˜ “’ | ”’ | ‘:’ | ‘;’ | ‘?’ | ‘/’ | ‘|’ | ‘\’
| ‘{’ | ‘}’ | ‘[’ | ‘]’ | ‘,’ | ‘.’ | ‘<’ | ‘>’ | ‘$’

A " can be included within a string by escaping it with another ", i.e. "".

Line Terminators and Whitespace

ASCII spaces, tabs, new line, carriage returns and form feeds are all considered as whitespace. Whitespace is ignored except as a token separator.

In-built functions

The following function names are not keywords but if present have a special meaning for the runtime environment:

- onClockTick
- onEntry
- onExit

3.3 Empath Specification Overview

An Empath program essentially consists of a description of the characteristics of an entity, a finite state machine (FSM) for representing its life cycle, and a set of options (known as events) available to the user for interacting with the entity.

In general, Empath is a C-like language (similar to Java or C#) in its use of curly braces, operators, function definitions etc. The nature of the Empath language necessitates the use of such constructs for performing simple arithmetic and logical operations on entity characteristics; and we did not want to reinvent the wheel in defining an unusual syntax.

Entity Specification

The entity specification gives a name to the entity being modeled and specifies the associated variables (usually characteristics of the entity), functions, states and transitions. The entity specification may also optionally include an embedded specification for one or more of the states of the entity. There can be only one entity specification in an empath program and the program must begin with it. All other constructs of the empath program are embedded in the entity specification block.

entitySpec → `‘entity’` *id* (`‘label’` *strConst)? programBody*

State Specification

A state specification is very similar to an entity specification. If a state is simple i.e. without any FSMs embedded inside it, the specification may include definition of the onClockTick(), onEntry() and onExit() functions, but no more.

If a state is complex (with a single FSM embedded inside it to form a hierarchy), the specification may include the above mentioned functions, but must also include a description of the embedded FSM. The dog example code in the appendix illustrates this. Here Dead is a simple state. The Puppy state is a complex state and includes a description of the day-to-day activities of a puppy.

stateSpec → *id* (`‘label’` *strConst*)? *programBody*
programBody → ‘{’ *decls funcDef stateDecls*
transDefs stateDefs ‘}’

The `label` keyword is used to specify a string that is passed to the user interface for display purpose. The *id* is used as an internal handle. Labels are used for states and functions, particularly events.

States

This lists the names of the set of possible states that the entity can be in. It works like a forward declaration for states, and is helpful to the programmer while specifying the transitions. The detailed definition for a state can be defined later in the *stateSpec* block. The initial state (the default state which the entity will be in at the beginning of execution of the program) is marked by including the keyword `init` before the corresponding state name. If the initial state is not thus explicitly marked then it is assumed to be the first state listed. The suggested convention for state names is that they be capitalized however this is not a requirement.

stateDecls → (*stateDecl*)*
stateDecl → `‘state’` *sDecl* (`‘,’` *sDecl*)* `‘;’`
sDecl → (`‘init’`)? *id* (`‘label’` *strConst*)?

Transition Specification

A transition specification defines the possible transitions that can occur between the entitys states. Each transition must specify a *from state*, a *to state*, a *trigger* and optionally, an *action*. The trigger may be a logical expression or a trigger function.

transDefs	→	(transDef)*
transDef	→	‘`transition`’ tDef (‘`,`’ tDef)* ‘`;`’
tDef	→	(id | *) ‘`to`’ id ‘`if`’ ‘`(`’ expression ‘`)`’ (‘`/`’ functionCall)?

A * in place of a *from state* is a shorthand operator signifies that a transition can be taken from any of the states in this FSM to a state outside the FSM. In general we allow inter-level transitions from an inner-level state to a higher level state. It would be useful to mention here that the semantics of execution of this FSM are as follows:

- Transitions are evaluated from outside to inside
- A transition triggered at an outer level preempts transitions triggered at an inner level

Variable Declarations

decls	→	(decl)*
decl	→	type id (‘`=`’ constant)? (‘`,`’ id (‘`=`’ constant)?)* ‘`;`’
type	→	‘`int`’ | ‘`float`’ | ‘`string`’ | ‘`boolean`’
		| ‘`range`’ ‘`[`’ constant ‘`:`’ constant ‘`]`’

Function Definitions

Event: An event is a special type of function which begins with the keyword `event`. These functions are marked to be visible to the user interface. They provide a means of interaction with the Empath program.
Trigger: A trigger is a special type of function which begins with the keyword `trigger`. These functions have the sole purpose of evaluating if a threshold has been reached and if the corresponding transition will be carried out. They also have the following restrictions

- cannot modify characteristics

- return type must be boolean
- can optionally modify variables in local scope but cannot modify those which have global scope

Action: An action is a function defined in the action part of a transition definition. The instructions in an action function will be executed if the trigger corresponding to the transition has been evaluated as true and the transition is fired. An action cannot return a value i.e. the return type of an action must be void. Any function whose return type is `void` can be an action.

```
funcDef    →   (func)*
func       →   ( ‘trigger’ | ‘event’ | ‘function’ retType )
               ( ‘label’ strConst)? id ‘(’ argsDef ‘)’ ‘{’ funcBody ‘}’
argsDef    →   (type id (‘,’ type id)*)?
funcBody   →   decls (stmt)*
retType    →   ‘void’ | type
stmt       →   expr ‘;’
               | ‘if’ ‘(’ expr ‘)’ stmtBlk ‘else’ stmtBlk)?
               | ‘for’ ‘(’ expr ‘;’ expr ‘;’ expr ‘)’ stmtBlk
               | ‘while’ ‘(’ expr ‘)’ stmtBlk
               | ‘output’ ‘(’ (constant | id) ‘)’ ‘;’
               | ‘return’ (expr)? ‘;’
stmtBlk    →   ( ‘{’ (stmt)+ ‘}’)
               | stmt)
```

Most of the statements and expressions are similar to those in the C or Java language. We describe the new ones:

- output(x): The parameter of this statement is passed to the UI for displaying
- tick n: This expression is useful in the onClockTick() method where on every nth clock tick some characteristic is changed.
- id @max: This expression checks whether a range type variable has attained the maximum value defined in the range.

expr → (*assExpr*
| ‘tick’ *intConst*)
assExpr → (*id* (‘=’ | ‘+=’ | ‘-=’ | ‘*=’ | ‘/=’ | ‘%=’))? *logOrExpr*
logOrExpr → *logAndExpr* (‘∥’ *logAndExpr*)*
logAndExpr → *eqExpr* (‘&&’ *eqExpr*)*
eqExpr → *relExpr* ((‘!=’| ‘==’) *relExpr*)*
relExpr → *addExpr* ((‘⟨’ | ‘⟩’ | ‘⟨=’ | ‘⟩=’) *addExpr*)*
addExpr → *mulExpr* ((‘+’| ‘-’) *mulExpr*)*
mulExpr → *uExpr* ((‘*’| ‘/’ | ‘%’) *uExpr*)*
uExpr → *pExpr* (‘++’ | ‘--’ | ‘@min’ | ‘@max’)?
pExpr → (*logOrExpr*) | *val*
val → (*constant* | *id* | *functionCall*)

- id @min: Similarly, this expression checks whether a variable has attained the minimum value defined in the range.

4 Project Plan

This section outlines the project planning aspects of Empath: organization of the team, environment used and further planning aspects.

4.1 Project Processes

Project planning was done within the framework of the deadlines. We tried to even the work out over the semester to avoid doing all the work right at the end, and on the whole this was successful. Our group had regular meetings on a Monday to plan the week ahead. We also met on a fortnightly basis with the TA. We often used these meetings as internal deadlines; for example we would aim to get a working target code example in a presentable state for the next TA meeting.

Specification of the language was done through a series of workshops which resulted initially in a fairly complex example (the dog example, submitted in the language reference manual hand in). From this, we formally specified the language grammar.

Specification of aspects like the static semantic analysis was done with unit tests, so that by the end of the project we had a suite of tests for what was and was not allowed on a semantic level in our language.

Development was done in two different directions: while one half of the team (Nalini and Sampada) worked on the parser and then on the tree walker, Jeremy grappled with the development of the (manually developed) target code that we wanted to eventually generate. At the end phase of the project, we joined these two together with the actual code generation. People had their own areas of specialization and owned parts of the project. For example, with the parser and walker, we divided the language between the functional logic (Nalini) and the state descriptions (Sampada).

Our project put a strong emphasis on testing, but it was for most part the responsibility of each developer to test their own code (details of this below). We did not have a large enough team to have code tested by someone other than the developer.

4.2 Programming Style Guide

We made the mistake in our project of not imposing a forming style guide until it was too late. We did not think it was necessary until it became apparent late in the process that people were using very different styles, and there was a difference in e.g. use of comments. The coding convention mailed around is based in part on the Sun convention (http://java.sun.com/docs/codeconv/) and in part on informal standards from industry.

The style guide that the project then adopted had the following points:

- commenting using Javadoc, at class level and method level - this was the most important point
- use of the *private* modifier for class variables, and make use of getter and setter access methods (which can be auto-generated in Eclipse)
- make methods *private* by default and only *protected* or *public* if needed
- use of the naming convention of ”_myvar” for class member variables this has the advantage of distinguishing member variables from variables at method scope. It also makes something like ”myobject._myvar” look ugly and should therefore be avoided.
- stick to indentation like this:

  ```
  while (foo) {
     //code
  }
  ```

  and not

  ```
  while (foo)
  {
     //code
  }
  ```

4.3 Project Time line

The project time line is shown on the following pages - the full file is available as a spreadsheet.

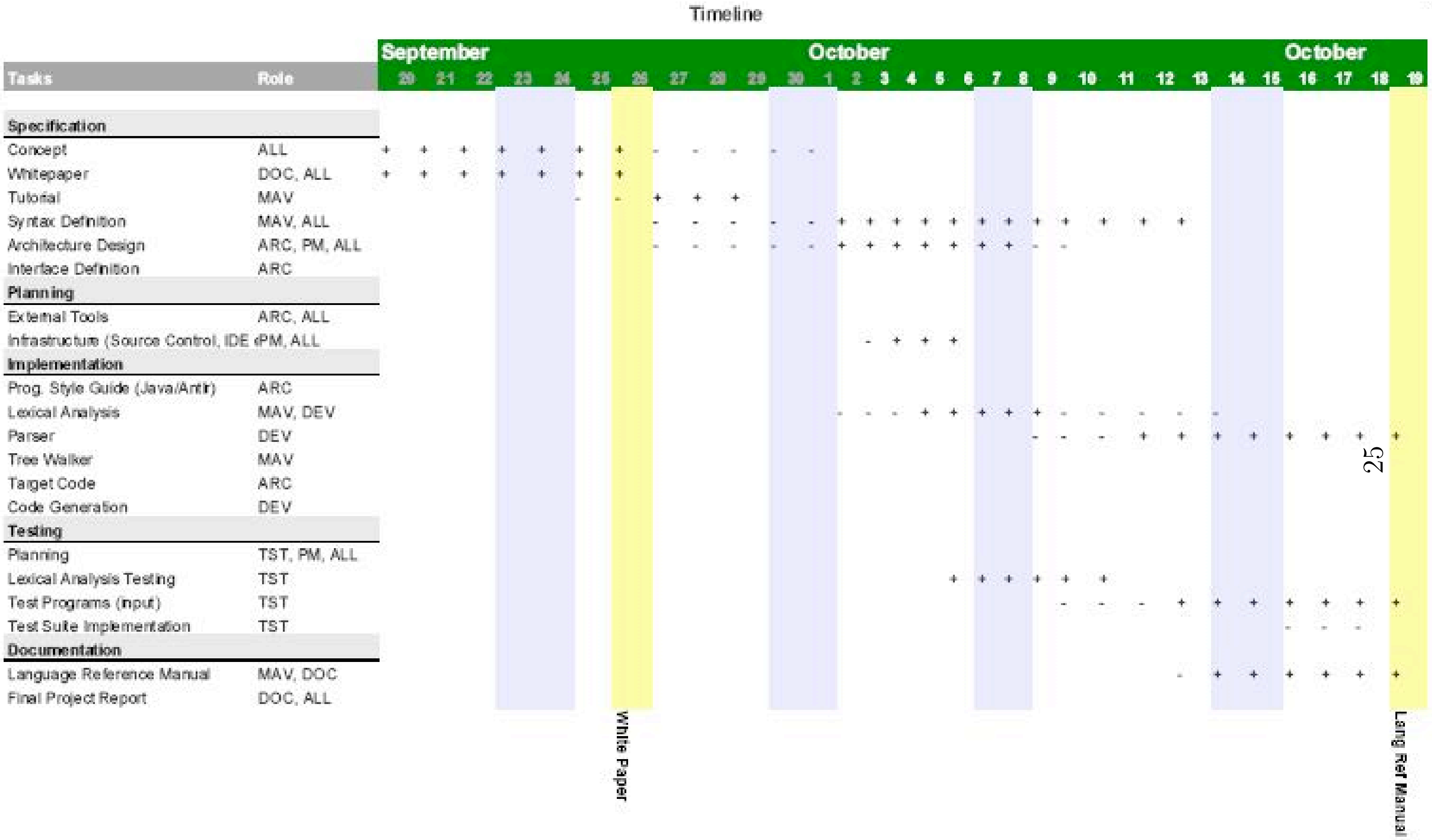
4.3.1 Project Log

Shown in the table below are some of the goals and milestones achieved for the project

Sep 20	kickoff: decision to develop a virtual pet language, assignment of roles
Sep 24	name Empath coined (Jeremy)
Sep 26	Empath white paper submitted
Oct 14	cvs repository and other infrastructure setup
Oct 18	work on lexer began
Oct 19	Language Reference Manual submitted
Nov 12	work on parser began
Nov 21	parser functioning
Nov 27	target code working
Nov 29	first lines of code generated
Nov 20	work on second tree walk (code generation) begun
Dec 12	user interface begun
Dec 14	first runnable code

4.4 Roles and Responsibilities

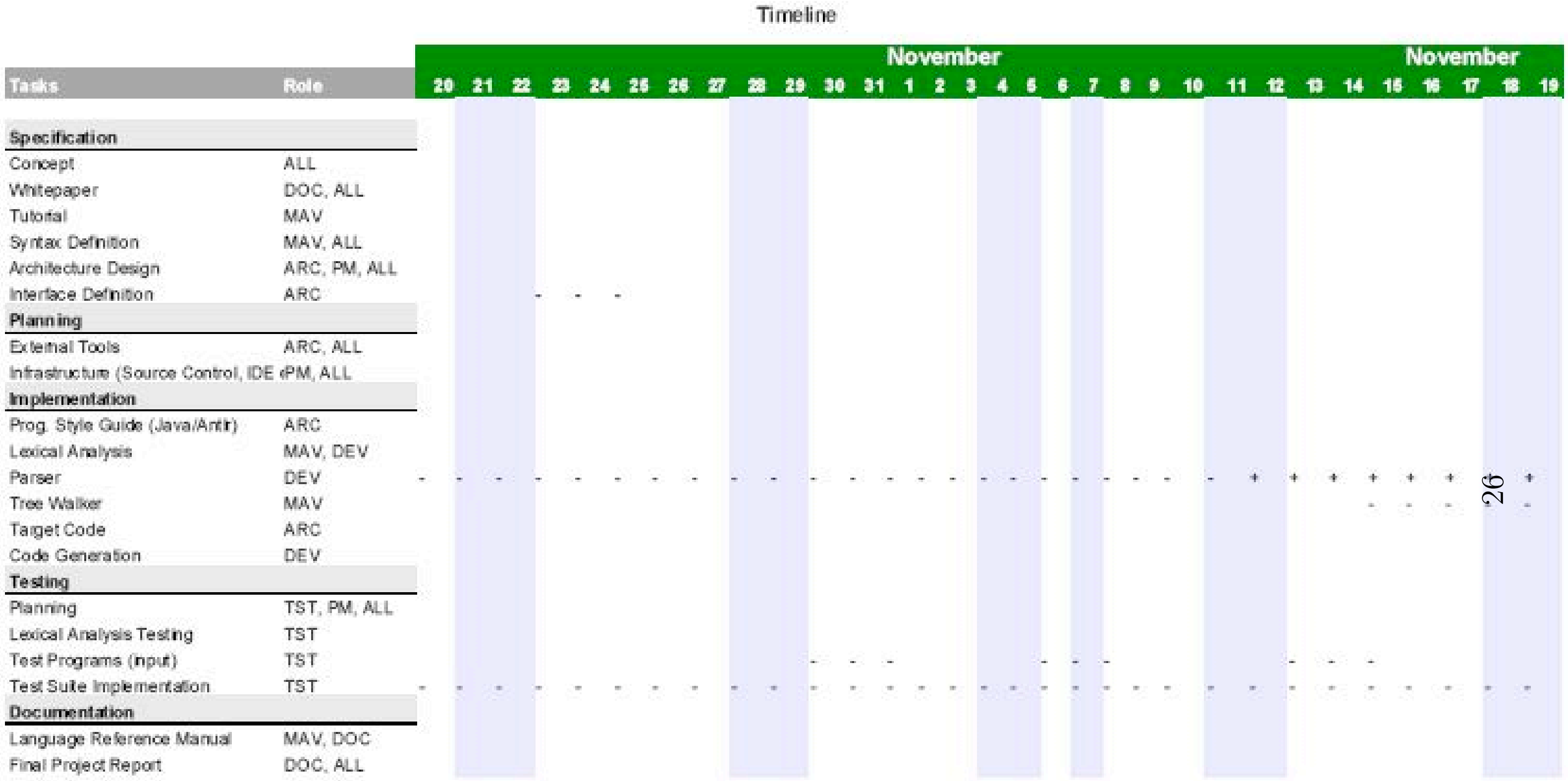
Roles were allocated early in the project. Some of the roles were assigned to a single person, others to two, either equally or headed by one person with support from the other.
Architecture Jeremy. This involved the decision on how interaction between runtime, entity and user interface
Development everyone
Documentation Sampada. This role included setting up document structure and helping others out with TeX.
Infrastructure Will. This included setting up of version control, finding plug ins etc.
Language Mavens Nalini and Sampada. Responsible not only for the language definition, but also the implementation (lexer,parser and tree walking), static semantic analysis etc.
Presentation Nalini
Project Manager Will
Testing Will with implementation support from everyone



Timeline

		September											October															October			
Tasks	Role	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Specification																															
Concept	ALL	+	+	+	+	+	+	+	-	-	-	-	-																		
Whitepaper	DOC, ALL	+	+	+	+	+	+	+																							
Tutorial	MAV						-	-	+	+	+																				
Syntax Definition	MAV, ALL								-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+						
Architecture Design	ARC, PM, ALL								-	-	-	-	-	+	+	+	+	+	+	+	-	-									
Interface Definition	ARC																														
Planning																															
External Tools	ARC, ALL																														
Infrastructure (Source Control, IDE	PM, ALL														-	+	+	+													
Implementation																															
Prog. Style Guide (Java/Antlr)	ARC																														
Lexical Analysis	MAV, DEV													-	-	-	+	+	+	+	+	-	-	-	-	-					
Parser	DEV																				-	-	-	+	+	+	+	+	+	+	+
Tree Walker	MAV																														
Target Code	ARC																														
Code Generation	DEV																														
Testing																															
Planning	TST, PM, ALL																														
Lexical Analysis Testing	TST																	+	+	+	+	+	+								
Test Programs (input)	TST																					-	-	-	+	+	+	+	+	+	+
Test Suite Implementation	TST																											-	-	-	
Documentation																															
Language Reference Manual	MAV, DOC																								-	+	+	+	+	+	+
Final Project Report	DOC, ALL																														
								White Paper																					Lang Ref Manual

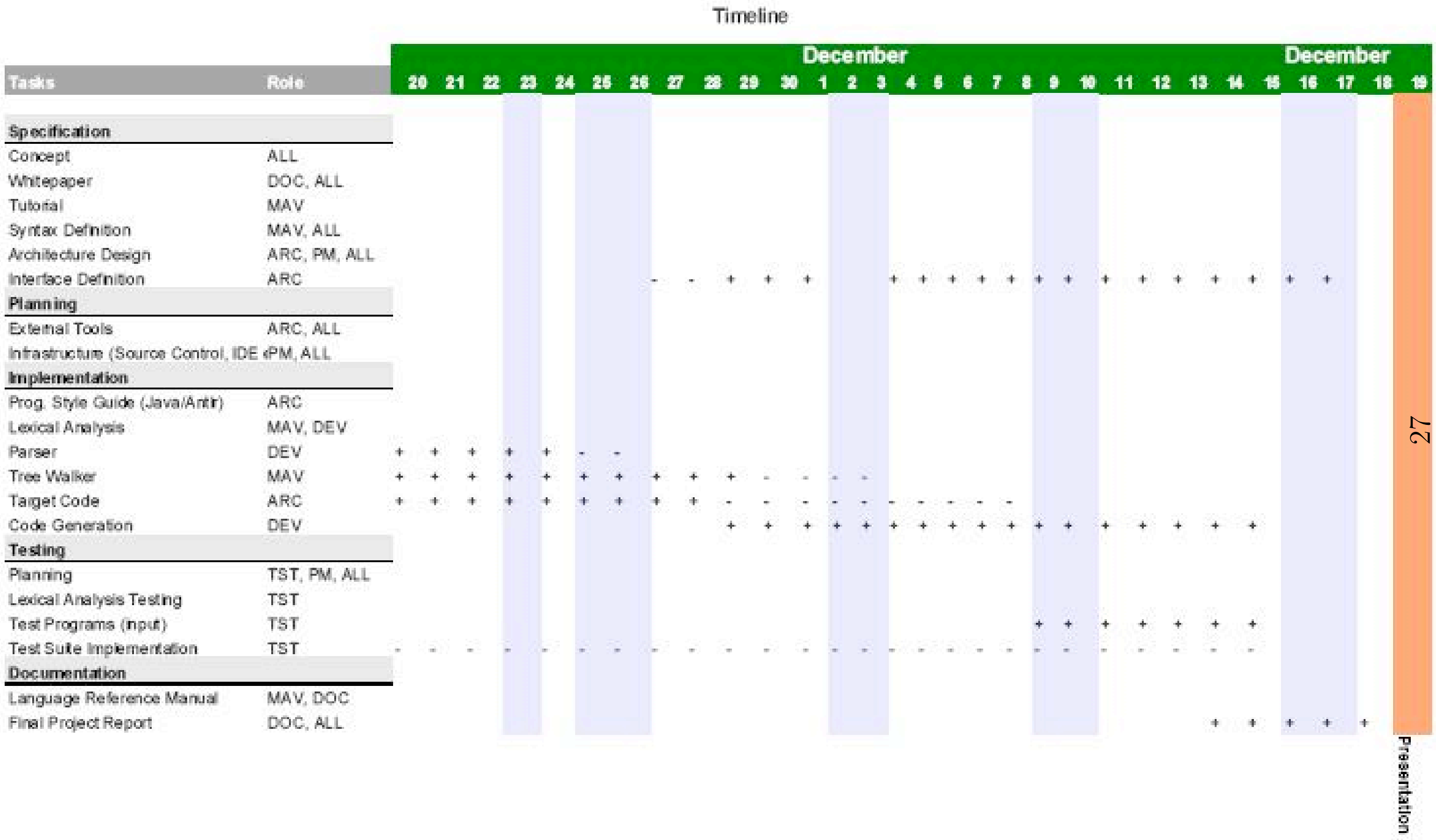
Figure 2: Planning 1



Timeline

																November														November		
Tasks	Role	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Specification																																
Concept	ALL																															
Whitepaper	DOC, ALL																															
Tutorial	MAV																															
Syntax Definition	MAV, ALL																															
Architecture Design	ARC, PM, ALL																															
Interface Definition	ARC				-	-	-																									
Planning																																
External Tools	ARC, ALL																															
Infrastructure (Source Control, IDE	PM, ALL																															
Implementation																																
Prog. Style Guide (Java/Antlr)	ARC																															
Lexical Analysis	MAV, DEV																															
Parser	DEV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+
Tree Walker	MAV																											-	-	-	-	-
Target Code	ARC																															
Code Generation	DEV																															
Testing																																
Planning	TST, PM, ALL																															
Lexical Analysis Testing	TST																															
Test Programs (input)	TST										-	-	-					-	-	-						-	-	-				
Test Suite Implementation	TST	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Documentation																																
Language Reference Manual	MAV, DOC																															
Final Project Report	DOC, ALL																															

Figure 3: Planning 2



Timeline

Tasks	Role	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		December																										December			
Specification																															
Concept	ALL																														
Whitepaper	DOC, ALL																														
Tutorial	MAV																														
Syntax Definition	MAV, ALL																														
Architecture Design	ARC, PM, ALL																														
Interface Definition	ARC								-	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		
Planning																															
External Tools	ARC, ALL																														
Infrastructure (Source Control, IDE	PM, ALL																														
Implementation																															
Prog. Style Guide (Java/Antlr)	ARC																														
Lexical Analysis	MAV, DEV																														
Parser	DEV	+	+	+	+	+	-	-																							
Tree Walker	MAV	+	+	+	+	+	+	+	+	+	+	-	-	-	-																
Target Code	ARC	+	+	+	+	+	+	+	+	+	-	-	-	-	-	-	-	-	-	-											
Code Generation	DEV										+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+				
Testing																															
Planning	TST, PM, ALL																														
Lexical Analysis Testing	TST																														
Test Programs (input)	TST																				+	+	+	+	+	+	+				
Test Suite Implementation	TST	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
Documentation																															
Language Reference Manual	MAV, DOC																														
Final Project Report	DOC, ALL																							+	+	+	+	+			
																															Presentation

Figure 4: Planning 3

4.5 Development Environment

Empath was implemented entirely in version 1.5 of the Java language. Having both the compiler implementation and the target code in Java simplified our development. We made use of Java generics, Enums and other constructs introduced in this version of the language. We used the prescribed Antlr toolset (version 2.7 because of the Eclipse plugin available, see below)

4.5.1 Version Control

We used CVS for version control. We wanted to use SubVersion initially, but had a problem with using it in combination with the NFS on the clic lab machines, so made the switch to CVS early.

4.5.2 Instant Messaging

We made extensive use of Instant Messaging (AIM in this case) to communicate with each other while developing. This worked well, since only three of the four team members lived near campus.

4.5.3 Integrated Development Environment

We did almost all development using the open-source Eclipse IDE version 3.2 (http://www.eclipse.org). This environment made a big difference to our development work because we could find good plugins for all the various parts of the development. In particular, we made use of the following Eclipse plugins:

1. CVS - this standard Eclipse plugin helped us with the version control

2. JUnit - this plugin, which is delivered with Eclipse, was used as the basis for our tests

3. ANTLR plugin for Eclipse - this plugin made our lives a lot easier because it has (http://antlreclipse.sourceforge.net/) features like syntax highlighting and error reporting, and also generates the

4. Texlipse - gave support for the documentation using TeX

Eclipse provided us with a lot of support and also the same environment for everyone, regardless of the operating system they were using (our team developed on a mix of Mac, Windows and Linux machines. Because of the system-independent nature of Java, and the Eclipse environment, this was not an issue).

4.5.4 Documentation

We chose to use TeX to do our documentation because it makes it easier to compare versions (in e.g. CVS), because it is easier to split up the document so different people authored different sections, and because of bad experiences with complex word processors such as MS Word in the past.

5 Architectural Design

The Empath system runs the programmer's code through a compiler, which produces Java code. The Java is then compiled and run through the Empath Runtime Environment, which has a swappable user interface module.

5.1 Compiler

The compiler has three main components

- Lexical analyzer
- Parser
- Abstract syntax tree walker

Lexical analyzer

The lexical analyzer performs the standard task of accepting an Empath file as input and generates a stream of tokens. It identifies and ignores line terminators, white spaces, single and multi-line comments. It identifies and tokenizes literals, operators, and identifiers. We did not do anything out of the ordinary in our implementation here.

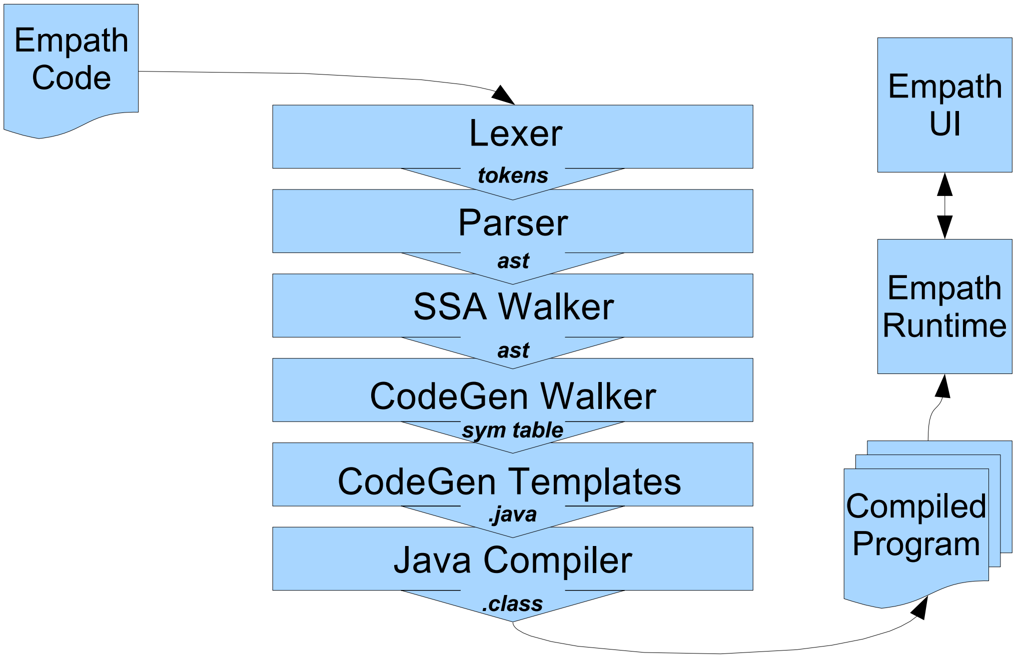


Figure 5: Architecture Overview

Parser

The parser receives the stream of tokens from the lexical analyzer and matches them against the grammar production rules of Empath. It generates the Abstract Syntax Tree (AST) representation for the Empath program. We pared down the tree as much as possible by dropping all structural punctuation. Again, this implementation using Antlr was standard.

One of the late surprises of Antlr for us was that we had to implement our own line numbering logic so that errors would correctly identify the origin of the error in the Empath source. This fundamental parser functionality was also not well documented in Antlr and we had to get the information from newsgroups. We extended the CommonAST class with our own class (BaseAST) which implemented all the line number logic.

Abstract syntax tree walker

The AST walker performs static semantic analysis while walking the AST and populates the symbol table. The code generation phase requires a second walk of the AST for building the function definitions.

5.1.1 Static semantic analysis

We put a lot of effort into the static semantic analysis to ensure that state definitions in the input were consistent. The following checks are made in static semantic analysis:

1. Re-declaration of states. States must have unique names in the program.

2. Multiple initial states

3. From and to states in a transition definition must be declared earlier.

4. The triggering condition on the transition must be either a logical expression or a trigger function(that returns boolean value).

5. The action part of the transition must be a function that returns void.

6. Forward declaration of states. States must be declared before defining the child FSM.

7. Multiple functions having the same function name cannot be defined in the same scope.

8. Trigger functions cannot modify global variables

9. For the special functions 'onClockTick', 'onExit' and 'onEntry'-

   (a) Function type cannot be a trigger or an event.

   (b) They must not return any value (must have return type void).

   (c) They cannot have argument lists

10. The variables listed in the argument list of a function definition cannot be already defined in the current scope.

11. There must be a return statement in the function body if the return type is not void.

12. The type of the returned value must match the defined return type.

13. For any reference to variables in the body of a function, there must be a variable of that name defined.

14. For variable declarations, there must not be another state, function or variable defined in the current scope with the same name.

15. When declaring variables of type range, the limit values must be integer literals.

16. Range must have valid bounds i.e. (lowerBound ⟨= upperBound).

17. The assigned literal in a variable declaration must be of the same type as the variable type.

18. A statement must not consist of a non-terminal expression, for example the following is not a valid statement: a+b;

19. Condition for an 'if' statement must be a boolean expression.

20. The second expression in a for loop (conditional) must be a boolean expression.

21. Condition for a 'while' loop must be a boolean expression.

22. The operand to an 'output' statement must be a string literal or string variable.

23. The operands to a mathematical operator must be of the same type and of one of the following types-

    (a) int

    (b) range

    (c) float

24. The operands to a logical operator must both be of boolean type.

25. The operands to the '@max' and '@min' operators must be variables of type range.

26. The operands to the increment and decrement operators must be variables.

27. Function calls must pass the following checks-

    (a) Function must have been defined previous to call.

    (b) The number, types and order of arguments in function call must match those of the function definition.

    (c) Return value of the function called must be assigned to a variable of the same type.

5.1.2 Symbol Table

The symbol table has entries for variable declarations, function definitions, and state declarations. It is implemented as a Java HashMap. Since we have a unified symbol table for variables, functions and states, the language has a single namespace.

The local variables of a function are stored in a separate symbol table and is referenced from the function entry in the parent symbol table.

The FSM hierarchy in a state is also maintained as a separate symbol table with a reference from the state entry in the parent symbol table.

5.2 Code Generator

The final section of our compilation process was the generation of compilable Java code.

We generated code in two different ways. A second walk of the abstract syntax tree was responsible for the conversion of empath functions into the Java equivalent, and these were combined with other information from the symbol table using a templating languages to produce Java source. These two code generation subsystems are explained in more detail below.

Function Code Generation

Although the Empath syntax for variable manipulation, control flow etc. was similar to Java, there were also some important differences.

We made use of a second tree walker to walk the Empath functions, doing necessary conversion and constructing the equivalent Java functions which were, on completion, attached to the function's entry in the symbol table.

While much of the work of this tree walker was the mechanical reconstruction of nested expressions etc. from the AST, and to some extend meant us undoing all the work we had done in the AST construction in the first place, it was also responsible for important conversions. For example, the onClockTick() function in Empath has no arguments and makes use of the **tick** keyword; the Java equivalent has the tick (clock value) given as an argument, uses the % modulus operator to implement the tick check and adds an implicit call to the onClockTick() of the superclass. Other important conversion includes the empath range type to a Java class, and the Empath string which is handled differently to the Java String class.

Templates

The final section of the code generation is the creation of Java source files. To do this, we made use of a templating language. Although there were many options here, such as the widely-used Velocity language, we ultimately chose to use the *String Template* templating language, because it is a sister project to Antlr. String Template has all the necessary constructs (conditions, loops etc) for us to do the final code generation.

Because we had designed the target code towards code which would be generated, we were able to do all the necessary code generation with just two templates:

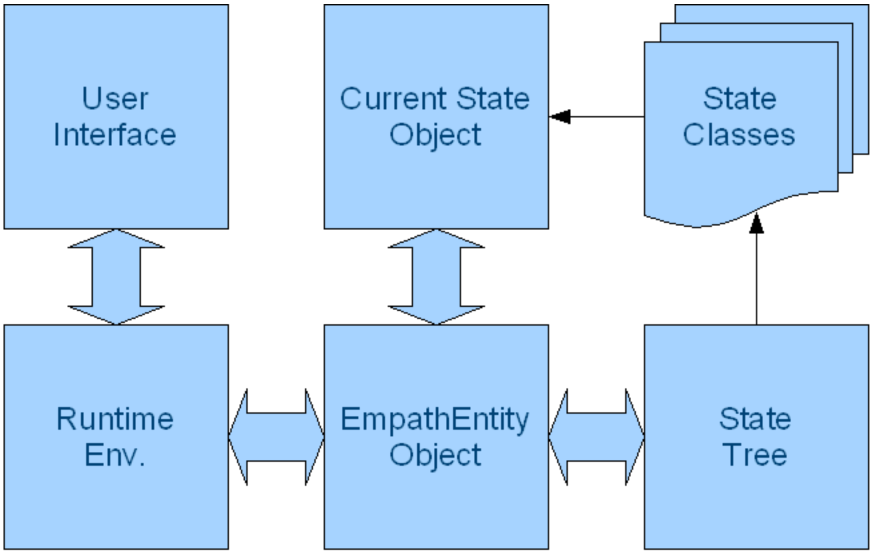


Figure 6: The Empath Runtime

- EntitySubclass.st generated the class to construct the tree of states and transitions between them
- StateSubclass.st generated a class corresponding to each state defined in the empath source

The symbol table obtained from the second (code generation) tree walk was processed recursively to populate these templates, which were then written to .java source files which could be compiled.

5.3 Running Empath Code

The compiled Java code from the code generation process is run through the Java Runtime Environment, which itself has two parts: the Runtime Environment itself and the User Interface module.

While we had initially anticipated relying on Java's Object model for inheritance characteristics, allowing each substate to inherit the variables of its parent. Ultimately, this proved not to be the case, as Java does not have an easy way of swapping one class for another even if they are both extensions of the same parent type. The runtime environment is a product of working around this limitation of Java's.

We also made the deliberate decision to have a runtime environment that was more complicated than was minimally necessary in order to simplify the process of generating code in the compiler. While the project could have been designed with the opposite distribution of complexity, this greatly simplified the debugging process.

Generated Code

The generated code includes an EmpathEntity class, and a group of EmpathState classes. When instantiated by the Runtime Environment, the EmpathEntity object contains an EmpathStateTree, each node of which contains a pointer to one of the State classes, as well as a representation of their relationships to each other within the hierarchy. Each tree node contains a list of transitions originating from the corresponding state, including the trigger for the transition, any action required when the transition fires, and the state class for the target state.

The EmpathEntity object acts as a layer of abstraction between the runtime environment and the currently active state object, allowing the runtime environment to always point to one object for all its interactions. The EmpathEntity object itself handles the handoff between different state objects during transitions, triggering the evaluation of the transitions stored in the state tree.

The state classes use Java's own rules of inheritance to handle Empath's own inheritance from root state to substate. A state's variables are stored as static variables in that state's class. Each substate extends the parent state, so that when a transition between substates of the same parent is processed, the variables stored at the parent state level (and above) do not need to be touched, they simply remain in place for the new state object to read and manipulate.

Runtime Environment

The Empath Runtime Environment directs traffic between the entity object and the user interface. The class consists mostly of functions that serve to pass information through from the entity to the interface and vice versa. Among the passthroughs, the runtime environment takes the list of event functions available to be called and replaces the function names with their labels for the user interface to display. The runtime environment then takes the signal to trigger the event and matches it once again with the actual function, soliciting the user interface for any necessary input argument.

The runtime environment contains the main method that is used to run an entity. It first instantiates the user environment, then the entity. It terminates when the entity reaches a terminal state, which is defined as a state from which there are no possible transitions out.

User Interface

The Empath User Interface is actually a Java abstract class, which allows developers to make their own customized front ends. The base UI caches the labels for the current state (and its events). Whenever the runtime environment processes a change of state, it triggers the UI to refresh the cache and then refresh the display. The current Empath UI is a text based UI, but it would be entirely possible to replace the TextUI class with a GUI if one were so inclined.

The UI waits to get input from the user, which will either tell it to call an event function or a clock tick. As with the runtime environment, the UI terminates when the entity reaches a terminal state.

6 Test Plan

6.1 Example Test Code

This report includes three examples of Empath code listed in full in the Appendix.

6.1.1 bullfrog.emp

This is a very simple hello-world style example which was used when we were generating our first code from empath source. It has the advantage of few states and easy transitions between the states. It is also used in the language tutorial.

6.1.2 dog.emp

The entire development cycle of our project was accompanied by man's best friend: dog.emp is an example which we first used to workshop the language and then to define the syntax. Later, dog.emp was used to develop the target code that we wanted to generate, to check the static semantic analysis and to test the generated code.
dog.emp is a realistic example which contains 3 levels of states with transitions between. It displays most of the features of the empath language, including wildcard transitions, inter-level transitions, chained onClockTick() methods and use of onExit, onEntry and transition actions.

6.1.3 undergrad.emp

The undergraduate is a moderately complex example which was added later in the project life cycle.

6.2 Test Automation

Our project make extensive use of test automation. At each stage of the compiler construction, we aimed to develop a set of unit tests which we could automate and use for testing new logic as well as for regression testing. We used the JUnit framework (http://junit.org) version 3.8 for all of our unit tests. We did not make use of the *assert* keyword in Java 1.5 because we used the JUnit asserts. Our test code was kept separate from our source code (in a separate 'tst' source tree).

Input for the tests was a mixture of strings holding snippets of code and small to medium-sized Empath code in individual files. The tests are stored in the `tst` directory, the input code used for the tests is in the `code` directory.

6.3 Test Suites

We aimed specific tests at specific states of the compilation process. All the tests are in the package

6.3.1 Lexer Tests: EmpathLexerTest

The lexer was tested by passing it strings with code snippets and examining the sequence of tokens which resulted. This allowed us to check e.g. multi-line comments or the tokenization of INT_LIT or INT_REAL tokens.

6.3.2 Parser Tests: EmpathParserTest and LineNumberTest

The parser was tested at a variety of levels. We tested that certain productions performed as expected against input strings, and that invalid input strings caused exceptions. We also ran tests against entire input files, to ensure each of these represented valid input.

The LineNumberTest targets the reporting of line numbers in errors.

6.3.3 Static Semantic Analysis Tests: FuncSSATest and StateTransTest

The logic of the static semantic analysis was checked by tests which ran the tree walker against a set of files containing Empath code. Some of these were expected to pass the static semantic analysis, others had specific errors in them which the static semantic analysis was expected to catch.

6.3.4 Code Generation Tests: CodeGenWalkerTest

The code generation of the functional logic was tested by generating code from empath input and comparing the generated code string against a string we were expecting.

6.3.5 Entire Test Suite: AllTests

All of the above tests are collected into a single suite
`edu.columbia.cs.coms4115.empath.AllTests`
which runs all 33 tests as a unit.

6.4 Manual Tests

While developing the parser, we made use of the simple *ASTViewer* class which displays the AST in either text or graphical format (using Antlr's ASTFrame class). This was very useful for intermediate testing when we were developing the parser.

The generation of the final classes and the state transition logic is something which could have been tested automatically but was not. We would have needed to run an entity in remote control (i.e. simulate clock ticks and user input) and test final state for this, and we did not have enough time to achieve this. Instead, the generated code was tested manually, by comparison against what we expected to be generated, and by running the generated entities in the runtime environment.

6.5 Test Responsibilities

We used a continual testing approach to ensure that the code we were developing was continually being verified. The task of setting up the testing approach and helping develop the skeleton unit tests was done by Will. The responsibility of implementing the tests was assumed by the person developing that piece of logic. This was not ideal - we should have had a different tester to coder - but the team was too small and the focuses too tight for this to be a practical option.

7 Lessons Learned

Going through this project was a learning experience for everyone involved. We have put together some of the comments from each team member here (without naming names) as to what they learnt, both in the technical and non-technical aspects of the project.

7.1 Technical Lessons

Work done in setting up the proper infrastructure at the beginning paid off - in particular, the combination of the Eclipse environment and CVS helped us in this project.

We learnt a significant amount about practical aspects of compilers in this project, and enjoyed much of the implementation work.

Test-orientated development worked well, giving us something to check at each stage and as a way of demonstrating problems or solutions.

On the negative side, we think we may have put too much effort into the language: it was too complex for the scope of this project and if we had simplified it we would have had more time to concentrate on other aspects without loosing out on overall quality. In particular, it seemed like a waste of effort to take Java-like code and generate Java code. We could have, as an alternative approach, taken Java snippets and put them into the language, although this would have introduced other problems.

We thought we were choosing something simple, but it was in fact difficult both in concept and implementation

Object Orientated languages are not suited for all purposes

7.2 Teamwork Lessons

We managed to work in a way so that there few dependencies of people on each other. There were exceptions to this (such as the target code), but in general people could work at their own pace and there were no significant delays because of waiting for other team members. Another positive example of orthogonal, collaborative work was the use of TeX to do the documentation, which allowed each team member to own a section of the document.

The team had several problems initially, but we think it came together well at the end. In particular, it was challenging to work in and coordinate a team in which people had different commitments and levels of engagement.

The quality of the team members was evident. People were in control of what they were doing.

We managed to pace ourselves well across the semester

Communication was difficult; as a result of using areas of specialty, one half of the team did not know what the other half was doing. In particular, the half of the team doing the implementation with Antlr was not aware of the issues of the team doing the code generation and user interface, and vice versa.

It was difficult, even in a relatively small group of talented people, to coordinate the work. People did not have the same motivation. It was, especially at the beginning of the project, difficult to get everyone together at a meeting and people did not want to take responsibility. Some people in the team felt they were criticized unfairly.

A Appendix

A.1 Empath Code Generating Walker (codegen.g)

```
/* codegen.g : the AST walker for code generation
 * Author Sampada Sonalkar (ss3119@columbia.edu)
 */

header {
  package edu.columbia.cs.coms4115.empath.antlr;
}

 {
   import java.util.*;
   import edu.columbia.cs.coms4115.empath.codegen.*;
 }

 class CodeGenWalker extends TreeParser;
 options  {
   importVocab = Empath;
   defaultErrorHandler=false;
 }

 {
/*  public String filename;
   SymbolTable entitySymTab=new SymbolTable();
   SymbolTable tempSymTab=entitySymTab;
   FunctionEntry fe = null;
  boolean inarglist = false;
  boolean isstmt = false;
  public SymbolTable getSymTable() {
    return entitySymTab;
  }*/
  public String filename;
  SymbolTable entitySymTab;
   SymbolTable tempSymTab;
  public void setSymTable(SymbolTable symTab)  {
    entitySymTab = symTab;
    tempSymTab = entitySymTab;
  }
  String range_lower=null, range_upper=null;
  int tabCount = 2;
  boolean funcCall = false;
  boolean funcCond = false;
  public SymbolTable getSymTable() {
    return entitySymTab;
  }
 }

 entitySpec
   : #("entity"
   {
     StringBuffer a;
   }
   entityName:ID {
   }
   ("label" STRING_LIT)?
   programBody
   )
   ;

 programBody
   {     StringBuffer a;  // dummy
    }
   : #(PROGRAM
    a=decls
    funcDef
    stateDecls
    transitionDefs
    stateDefs
   )
   ;

 stateDecls
   : #(STATE_DECLS
     (sDecl)*
   )
   ;

 sDecl {boolean initFlag = false;}
   : #(S_DECL ("init" {initFlag = true;})? stateName:ID ("label" STRIN
G_LIT)?
   )
   ;


 transitionDefs
   : #(TRANSITIONS
     (tDef)*
   )
   ;


 tDef
    {  StringBuffer a, b = null;
      funcCall = false;
      funcCond = false;
      SymbolTableEntry ste;
      SymbolTable st;
   }
   // instantiate and add transition to symbol table
   : #(T_DEF //{action = null;}
  ( fromState:ID
    {  ste = tempSymTab.getEntry(fromState.getText());
      st = tempSymTab;
    }
  | MULT
    {  ste = tempSymTab.getParentEntry();
      st = tempSymTab.getParent();
    }
  )
  toState:ID
  a = expression
    {  if(funcCall)
    funcCond = true;
  }
 ( b = functionCall)?
 )
  {  // Add transition to transition list of fromState or parent st
ate of this FSM
   Transition t;
   if(b==null)
    t = new Transition(((StateEntry)ste).getName(), toState.getTe
xt(), a.toString());
   else
    t = new Transition(((StateEntry)ste).getName(), toState.getTe
xt(), a.toString(), b.toString());
   if(funcCond)
    t.setFunctionCondition();
   ((StateEntry)ste).addTransition(t);
   funcCall = false;
   funcCond = false;
  }
  ;

stateDefs
 : #(STATE_DEFS
 (sDef)*
 )
 ;

sDef
 : #(S_DEF
 stateName:ID
 {
  SymbolTableEntry ste = tempSymTab.getEntry(stateName.getText());
  tempSymTab = ((StateEntry)ste).getChildSymTab();
 }
 ("label" STRING_LIT)?
 programBody
 )
 {  if (tempSymTab.parent!=null)
```

```
    tempSymTab = tempSymTab.parent;
  }
  ;

functionCall returns [StringBuffer r]
{
 r = new StringBuffer();
  boolean firstArg = true;
 StringBuffer a, b;
}
 : #(
  F_CALL
  fun:ID
  {  r.append(fun.getText());
   r.append('(');
  }
  (
   id:ID | a = constant
   {
    if(firstArg)  {
     r.append(id.getText());
     firstArg = false;
    }
    else  {
     r.append(", ");
     r.append(id.getText());
    }
   }
  )*
  {  r.append(')');
   funcCall = true;
  }
 )
  ;

funcDef
 : #(FUNCDEF
 (func)*)
```

```
  ;

func
    {  StringBuffer r = new StringBuffer();
      StringBuffer a, b, c;
    }
  : #(FUNC
  {  boolean isEvent=false;
    r.append("\tpublic ");  }
  ("trigger"
    | "event" {  isEvent = true;  }
    | "function" a = retType
  )
  ("label" STRING_LIT)?
  i:ID  {
    FunctionEntry fe = (FunctionEntry)tempSymTab.getEntry(i.getText())
;
    tempSymTab = fe.getlocalSymTab();
    r.append(fe.getRetType().getJavaType());
    r.append(’ ’);
    if(isEvent)  {
      r.append(CodeGenerator.PREFIX_EVENT);
      isEvent = false;
    }
    r.append(i.getText());
    r.append(’(’);
    if(i.getText().equals("onClockTick"))
      r.append("int tick");
  }
  b = argsDef
    {  r.append(b);  r.append(’)’);  }
  c = funcBody
    {  r.append("\t{\n");  r.append(c);
      if(i.getText().equals("onClockTick"))
        r.append("\t\tsuper.onClockTick(tick);\n");
      r.append("\t}\n\n");  }
  )
    {  fe.setDefinition(r.toString());
```

```
      if (tempSymTab.parent!=null)
        tempSymTab = tempSymTab.parent;
    }
  ;

retType returns [StringBuffer r]
  {
    r = new StringBuffer();
     StringBuffer a;
   }
  : ("void"
    {  r.append("void");  }
  )
  | (a=type
    {  r.append(a);  }
  )
    {  r.append(' ');  }
  ;

type returns [StringBuffer r]
  {  StringBuffer a, b;
    r = new StringBuffer();
  }
  : "int"
    {  r.append("int");  }
  | "float"
    {  r.append("float");  }
  | "string"
    {  r.append("String");  }
  | "boolean"
    {  r.append("boolean");  }
  | "range" a=constant b=constant
    {  r.append("Range");
      range_lower = a.toString();
      range_upper = b.toString();
    }
  ;
```

```
argsDef returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a;
    boolean firstArg = true;
  }

  : #(ARGS_DEF
    ( a=type
      {  if(firstArg)
          firstArg = false;
        else
          r.append(", ");
        r.append(a);
        r.append(’ ’);
      }
     i:ID
       {  r.append(i.getText());
       }
     )*
  )
  ;

funcBody returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a, b;
  }
  : #(FUNCBODY
  a = decls
    {  r.append(a);  }
  ( b=statement
    {  r.append(b);  }
  )*
  )
  ;

decls returns [StringBuffer r]
```

```
  {
    r = new StringBuffer();
    StringBuffer a;
  }
  : #(DECLS
  ( a = decl
    {  r.append(a);  }
  )*
  )
  ;

statement  returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a=null, b=null, c=null, d=null;
    for(int j=0;j<tabCount;j++)
      r.append(’\t’);
  }
  : a=expression
    { r.append(a);   r.append(";\n"); }
  | #( "if" a=expression b=statementBlock (c=statementBlock)? )
    {   r.append("if (");  r.append(a);  r.append(’)’);
      r.append(b);
      if(c!=null)  {
        for(int j=0;j<tabCount;j++)
          r.append(’\t’);
        r.append("else");
        r.append(c);
      }
    }
  | #( "for" a=expression b=expression c=expression  d=statementBlock
)
    {  r.append("for(");  r.append(a);  r.append("; ");
      r.append(b);  r.append("; ");    r.append(c);
      r.append(’)’);  r.append(d);  }
  | #("while" a=expression b=statementBlock )
    {  r.append("while(");    r.append(a);
      r.append(’)’);      r.append(b);  }
  | #("output"  {   r.append("output(");  }
    (a=constant   {  r.append(a);  }
    | i:ID      {  r.append(i.getText());  }
    ) {  r.append(");\n");  }
  )
  | #("return"   {  r.append("return ");  }
    (a=expression  {  r.append(a);  }
    )?  {  r.append(";\n");  }
  )
  ;

statementBlock returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a;
    r.append("\t{\n");
    tabCount++;
  }
  :
  #(STMTBLK
  (a=statement
    { r.append(a);  }
  )+
  )
  {  tabCount--;
    for(int i=0;i<tabCount;i++)
      r.append('\t');
    r.append("}\n");
  }
  ;

decl returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a, b=null;
  }
  :#(DECL
  a=type
```

```
 ( i:ID
  {  r.append("\t\t");
   r.append(a);
   r.append(’ ’);
   r.append(i.getText());
   if(a.toString().equals("Range"))  {
    r.append(" = new ");  r.append(a);  r.append("(");
    // check for valid range limits
    if(Integer.parseInt(range_lower)>Integer.parseInt(range_upper)
)
     throw new SemanticException("Invalid range: "+i.getText()+"[
"+range_lower+", "+range_upper+"]",
      filename, i.getLine(), i.getColumn());
    // append min,max of range
    r.append(range_lower);  r.append(", ");
    r.append(range_upper);  r.append(", ");
    VariableEntry ve = tempSymTab.getVarEntry(i.getText());
    ve.setRangeLower(range_lower);
    ve.setRangeUpper(range_upper);
   }
   else if(a.toString().equals("String"))  {
    r.append(" = new ");  r.append(a);  r.append("(");
   }
  }
 ( b=constant)?
  {
   VariableEntry ve = tempSymTab.getVarEntry(i.getText());
   if (b==null)  {
    if(a.toString().equals("Range"))  {
     r.append("0)");
    }
    else if(a.toString().equals("String"))
     r.append(’)’);
    else  {
     r.append(" = ");
     r.append(ve.getJavaInitialValue());
    }
   }
```

```
      else  {
        String initVal = b.toString();
        if(a.toString().equals("Range"))  {
          if(Integer.parseInt(initVal) > Integer.parseInt(range_upper)
)
            initVal = range_upper;
          else if(Integer.parseInt(initVal) < Integer.parseInt(range_l
ower))
            initVal = range_lower;
          r.append(initVal);  r.append(')');
        }
        else if(a.toString().equals("String"))  {
          r.append("\"");  r.append(initVal);  r.append("\")");
        }
        else  {
          r.append(" = ");  r.append(initVal);
        }
        ve.setInitialValue(initVal);
      }
      r.append(";\n");
    }
  )*)
    {  range_lower = null; range_upper = null;  }
  ;



expression returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a;
  }
  :#(EXPR
    (
    ("tick" i:INT_LIT)
      {  r.append("((tick % "+i.getText()+") == 0)");  }
    |
    a=assignmentExpression
```

```
      {  r.append(a);  }
    )
  )
  ;

assignmentExpression returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a;
  }
  :
  #(ASSIGN varname:ID a=lefthandexpr)
  {   if(((tempSymTab.getVarEntry(varname.getText()))).getIDType()==Da
taRetType.RANGE)  {
      r.append(varname.getText());  r.append(".newValue(");  r.append(
a);  r.append(’)’);
    }
    else  {
      r.append(varname.getText());  r.append(" = ");  r.append(a);
    }
  }
  |#(PLUSEQ varname1:ID a=lefthandexpr)
  {  if(((tempSymTab.getVarEntry(varname1.getText()))).getIDType()==Da
taRetType.RANGE)  {
      r.append(varname1.getText());  r.append(".newValue(");  r.append
(varname1.getText()); r.append(".getValue() + ");  r.append(a);  r.app
end(’)’);
    }
    else  {
      r.append(varname1.getText());  r.append(" += ");  r.append(a);

    }
  }
  | #(MINUSEQ varname2:ID a=lefthandexpr)
  {  if(((tempSymTab.getVarEntry(varname2.getText()))).getIDType()==Da
taRetType.RANGE)  {
      r.append(varname2.getText());  r.append(".newValue(");  r.append
(varname2.getText()); r.append(".getValue() - ");  r.append(a);  r.app
end(')');
    }
    else  {
      r.append(varname2.getText());  r.append(" -= ");  r.append(a);

    }
  }
  | #(MULTEQ varname3:ID a=lefthandexpr)
  {  if(((tempSymTab.getVarEntry(varname3.getText()))).getIDType()==Da
taRetType.RANGE)  {
      r.append(varname3.getText());  r.append(".newValue(");  r.append
(varname3.getText()); r.append(".getValue() * ");  r.append(a);  r.app
end(')');
    }
    else  {
      r.append(varname3.getText());  r.append(" *= ");  r.append(a);

    }
  }
  | #(DIVEQ varname4:ID a=lefthandexpr)
  {  if(((tempSymTab.getVarEntry(varname4.getText()))).getIDType()==Da
taRetType.RANGE)  {
      r.append(varname4.getText());  r.append(".newValue(");  r.append
(varname4.getText()); r.append(".getValue() / ");  r.append(a);  r.app
end(')');
    }
    else  {
      r.append(varname4.getText());  r.append(" /= ");  r.append(a);

    }
  }
  | #(MODEQ varname5:ID a=lefthandexpr)
  {  if(((tempSymTab.getVarEntry(varname5.getText()))).getIDType()==Da
taRetType.RANGE)  {
      r.append(varname5.getText());  r.append(".newValue(");  r.append
(varname5.getText()); r.append(".getValue() % ");  r.append(a);  r.app
end(')');
    }
    else  {
      r.append(varname5.getText());  r.append(" %= ");  r.append(a);

    }
  }
  |(a=lefthandexpr)
  {
    r.append(a);
  }
  ;

lefthandexpr returns [StringBuffer r]
  {
    r = new StringBuffer();
    StringBuffer a, b;
  }
  :(
  #(OR a=lefthandexpr b=lefthandexpr)
  { r.append(’(’);  r.append(a);  r.append(" || ");  r.append(b);  r.a
ppend(’)’);}
  | #(AND a=lefthandexpr b=lefthandexpr)
  { r.append(’(’);  r.append(a);  r.append(" && ");  r.append(b);  r.a
ppend(’)’);}
  | #(EQ a=lefthandexpr b=lefthandexpr)
  {   r.append(’(’);  r.append(a);
    VariableEntry ve = tempSymTab.getVarEntry(a.toString());
    if((ve!=null && ve.getIDType()==DataRetType.STRING)||(a.toString()
.startsWith("\"")&&a.toString().endsWith("\"")))  {
      r.append(".equals(");  r.append(b);  r.append(’)’);
    }
    else  {
      r.append(" == ");  r.append(b);
    }
    r.append(’)’);}
  | #(NEQ a=lefthandexpr b=lefthandexpr)
  { r.append(’(’);
    VariableEntry ve = tempSymTab.getVarEntry(a.toString());
    if((ve!=null && ve.getIDType()==DataRetType.STRING)||(a.toString()
.startsWith("\"")&&a.toString().endsWith("\""))) {
      r.append("!");   r.append(a);   r.append(".equals(");
      r.append(b);  r.append(')');
    }
    else {
      r.append(a);  r.append(" != ");
      r.append(b);
    }
    r.append(')');  }
  | #(LT a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" < ");  r.append(b);  r.ap
pend(')');}
  | #(LEQ a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" <= ");  r.append(b);  r.a
ppend(')');}
  | #(GT a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" > ");  r.append(b);  r.ap
pend(')');}
  | #(GEQ a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" >= ");  r.append(b);  r.a
ppend(')');}
  | #(PLUS a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" + ");  r.append(b);  r.ap
pend(')');}
  | #(MINUS a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" - ");  r.append(b);  r.ap
pend(')');}
  | #(MULT a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" * ");  r.append(b);  r.ap
pend(')');}
  | #(DIV a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" / ");  r.append(b);  r.ap
pend(')');}
  | #(MOD a=lefthandexpr b=lefthandexpr)
  { r.append('(');  r.append(a);  r.append(" % ");  r.append(b);  r.ap
pend(')');}
  | #(INCR a=lefthandexpr)
  {   if(a.toString().endsWith(".getValue()")) {
      String var = a.toString().substring(0,(a.toString().length()-11)
);
      r.append(var);
      r.append(".incrementValue()");
    }
    else  {
      r.append(a);
      r.append("++ ");
    }
  }
  | #(DECR a=lefthandexpr)
  {   if(a.toString().endsWith(".getValue()"))  {
      String var = a.toString().substring(0,(a.toString().length()-11)
);
      r.append(var);
      r.append(".decrementValue()");
    }
    else  {
      r.append(a);
      r.append("-- ");
    }
  }
  | #(AT_MAX a=lefthandexpr)
  {  if(a.toString().endsWith(".getValue()"))  {
      String var = a.toString().substring(0,(a.toString().length()-11)
);
      r.append(var);
      r.append(".atMax()");
    }
  }
  | #(AT_MIN a=lefthandexpr)
  {  if(a.toString().endsWith(".getValue()"))  {
      String var = a.toString().substring(0,(a.toString().length()-11)
);
      r.append(var);
      r.append(".atMin()");
    }
  }
  | a=val
  { r.append(a);  }
  )
  ;

val returns [StringBuffer r]
  {
    StringBuffer a;
    r = new StringBuffer();
  }
  : a=constant  {r.append(a);}
    |  i:ID
    {  r.append(i.getText());
      if(((tempSymTab.getVarEntry(i.getText()))).getIDType()==DataRetT
ype.RANGE)
        r.append(".getValue()");
    }
    | a=functionCall  {r.append(a);}
    ;

constant returns [StringBuffer r]
  {  StringBuffer a;
    r = new StringBuffer();
  }
  : #(UPLUS a = cons)    {  r.append('+');  r.append(a);  }
  | #(UMINUS a = cons)  {  r.append('-');  r.append(a);  }
  | a = cons        {  r.append(a);  }
  ;

cons  returns [StringBuffer r]
  {
    r = new StringBuffer();
    }
  : a:STRING_LIT  {r.append("\""); r.append(a.getText()); r.append("\"
");}
  | b:INT_LIT    {r.append(b.getText());}
  | c:REAL_LIT  {r.append(c.getText());}
    |   "true"    {r.append("true");}
    |   "false"    {r.append("false");}
    |   "null"    {r.append("null");}
  ;
```

A.2 Empath Grammar (empath.g)

```
/*
*  empath.g contains the lexer and parser
*  authors: sampada and nalini
*/

header {
  package edu.columbia.cs.coms4115.empath.antlr;
}

class EmpathParser extends Parser;

options  {
  k = 2;
  buildAST = true;
  exportVocab = Empath;
  defaultErrorHandler=false;
  ASTLabelType = "edu.columbia.cs.coms4115.empath.antlr.BaseAST";
}

tokens {
  AT_MAX;
  AT_MIN;
  PROGRAM;
  V_DECL;
  FUNCDEF;
  FUNC;
  FUNCNAME;
  FUNCBODY;
  F_CALL;
  TYPE;
  STMT;
  STMTBLK;
  DECLS;
  DECL;
  STATE_DECLS;
  S_DECL;
  TRANSITIONS;
  T_DEF;
  STATE_DEFS;
  S_DEF;
  EXPR;
  ARGS_DEF;
  ARGS_CALL;
  UPLUS;
  UMINUS;
}

entitySpec : "entity"^ ID ("label" STRING_LIT)? programBody ;

programBody : LCURLY!
        decls
        funcDef
        stateDecls
        transitionDefs
        stateDefs
        RCURLY!
        {#programBody = #([PROGRAM, "program"], #programBody);}
        ;

stateDecls : (stateDecl)*
      {#stateDecls = #([STATE_DECLS,"stateDecls"], #stateDecls);}
      ;

stateDecl : "state"! sDecl (COMMA! sDecl)* SCOLON! ;

sDecl : ("init")? ID ("label" STRING_LIT)?
      {#sDecl = #([S_DECL,"s_decl"],#sDecl);}
      ;

transitionDefs : (transitionDef)*
      {#transitionDefs = #([TRANSITIONS,"transitions"], #transitionDef
s);}
      ;
```

```
transitionDef : "transition"! tDef (COMMA! tDef)* SCOLON! ;

tDef : ( ID | MULT ) "to"! ID "if"! LPAREN! expression RPAREN! ( DIV!
functionCall )?
      {#tDef = #([T_DEF,"t_def"], #tDef);}
      ;

functionCall :  ID LPAREN! ((ID | constant) (COMMA! (ID | constant))*)
? RPAREN!
        {#functionCall = #([F_CALL, "f_call"], #functionCall);}
        ;

stateDefs : (stateDef)*
      {#stateDefs = #([STATE_DEFS,"stateDefs"], #stateDefs);}
      ;

stateDef : ID ("label" STRING_LIT)? programBody
      {#stateDef = #([S_DEF,"s_def"], #stateDef);}
      ;

constant
  : PLUS! cons
    {#constant = #([UPLUS,"UPLUS"], constant);}
  | MINUS! cons
    {#constant = #([UMINUS,"UMINUS"], constant);}
  | cons
  ;

cons
  : STRING_LIT
  | INT_LIT
  | REAL_LIT
  |  "true"
    |   "false"
    |   "null"
  ;
variableIdentifier : ID ;
```

```
funcDef : (func)*
{#funcDef = #([FUNCDEF, "funcDef"], #funcDef);}
;

func :
    ("trigger" | "event" | "function" retType )
    ("label" STRING_LIT)?
    ID
    LPAREN!
    argsDef
    RPAREN!
    LCURLY!
    funcBody
    RCURLY!
    {#func = #([FUNC, "func"], #func);}
;

argsDef : (type ID (COMMA! type ID)*)?
  {#argsDef = #([ARGS_DEF, "argsDef"], #argsDef);}
  ;


funcBody : decls (statement)*
  {#funcBody = #([FUNCBODY, "funcBody"], #funcBody);}
  ;

//not entirely sure about how to specify range
type : "int"
  | "float"
  | "string"
  | "boolean"
  | "range" LSQUARE! constant COLON! constant RSQUARE!
  //{#type = #([TYPE, "type"], #type);}
  ;

retType : "void" | type;

statement :
     expression SCOLON!
    | "if"^ LPAREN! expression RPAREN! statementBlock
      (options { greedy = true; }:
          "else"! statementBlock)?
    | "for"^ LPAREN! expression SCOLON! expression SCOLON! expression
RPAREN!
      statementBlock
    | "while"^ LPAREN! expression RPAREN!
      statementBlock
    | "output"^ LPAREN! (constant | ID) RPAREN! SCOLON!
    | "return"^ (expression)? SCOLON!

    ;


statementBlock :
  ( LCURLY! (statement)+ RCURLY!
  | statement)
  {#statementBlock = #([STMTBLK, "statementBlock"], #statementBlock);}

  ;

decls :
  (decl)*
  {#decls = #([DECLS, "decls"], #decls);}
  ;

decl : type ID (ASSIGN! constant)? (COMMA! ID (ASSIGN! constant)?)*  S
COLON!
  {#decl = #([DECL, "decl"], #decl);}
  ;

expression :
    (assignmentExpression
      | "tick" INT_LIT)
      {#expression = #([EXPR,"expression"],#expression);}
      ;
```

```
assignmentExpression :
    (ID
           (
            ASSIGN^
        |   PLUSEQ^
            |   MINUSEQ^
            |   MULTEQ^
            |   DIVEQ^
            |   MODEQ^
          )
        )?
       logicalOrExpression
    ;


logicalOrExpression :
  logicalAndExpression (OR^ logicalAndExpression)*
    ;


logicalAndExpression :
  equalityExpression (AND^ equalityExpression)*
    ;

equalityExpression :
    relationalExpression ((NEQ^ | EQ^) relationalExpression)*
    ;

relationalExpression :
    additiveExpression
    (
      (
        LT^
        |   GT^
        |   LEQ^
        |   GEQ^
      )
      additiveExpression
    )*
    ;


additiveExpression :
  multiplicativeExpression ((PLUS^ | MINUS^) multiplicativeExpression)
*
    ;

multiplicativeExpression :
  unaryExpression ((MULT^ | DIV^ | MOD^ ) unaryExpression)*
    ;

unaryExpression :
   primaryExpression ( INCR^ | DECR^ | AT_MIN^ | AT_MAX^ )?
    ;

primaryExpression :
     LPAREN! logicalOrExpression RPAREN!
     | val
  ;

val :
   constant
    |  ID
    | functionCall
    ;


class EmpathLexer extends Lexer;
options {
  k=2;
  charVocabulary = '\3'..'\377';
  exportVocab = Empath;
  testLiterals = false;
}
```

```
// newline
NEWLINE:  (
    ’\n’
    | ’\r’
    | (’\r’ ’\n’) => ’\r’ ’\n’
   ) {newline();$setType(Token.SKIP);};

WSPACE:  (’ ’|’\t’|’\f’)+{ _ttype = Token.SKIP; };


// multiline comment
ML_COMMENT:  "/*"
     ( // Prevent .* from eating the whole file
      options {greedy=false;}:
      (
       (’\r’ ’\n’) => ’\r’ ’\n’ { newline(); }
       | ’\r’ { newline(); }
       | ’\n’ { newline(); }
       | ~( ’\n’ | ’\r’ )
      )
     )*
    "*/" { $setType(Token.SKIP); };

// single line comment (~( ’\n’ | ’\r’ ))* (NL)
SL_COMMENT: "//" (~(’\n’ | ’\r’))* NEWLINE { _ttype = Token.SKIP; };

// variable identifiers
ID options { testLiterals=true; }: CHAR_ATOM (DIGIT|CHAR_ATOM)*;

// strings cannot span multiple lines
STRING_LIT:  ’"’! (
    ~(’"’|’\n’|’\r’)
    | ’"’! ’"’
   )* ’"’!;

INT_LIT: (DIGIT)+
   ( DOT (DIGIT)+ { $setType(REAL_LIT); }
//      | /* empty*/   { $setType(INT_LIT); }
      )?;

//REAL_LIT: (DIGIT)+ DOT (DIGIT)+;

//BOOL_LIT: "true" | "false";

// the delimiter we use
SCOLON:’;’;

//brackets etc
LCURLY  : ’{’;
RCURLY  : ’}’;
LPAREN  : ’(’;
RPAREN  : ’)’;
LSQUARE  : ’[’;
RSQUARE : ’]’;

// other punctuators
COMMA : ’,’;
DOT    : ’.’;
COLON : ’:’;

// arithmetic operators
PLUS   : ’+’;
MINUS   : ’-’;
MULT  : ’*’;
DIV    : ’/’;
MOD    : ’%’;
INCR  : "++";
DECR  : "--";
PLUSEQ  : "+=";
MINUSEQ  : "-=";
MULTEQ  : "*=";
DIVEQ  : "/=";
MODEQ  : "%=";

//logical operators
```

```
ASSIGN  : ’=’;
EQ    : "==";
NEQ    : "!=";
GT    : ’>’;
GEQ    : ">=";
LT    : ’<’;
LEQ    : "<=";
AND    : "&&";
OR    : "||";
NOT    : ’!’;
MIN_MAX  : ’@’
      ( ("max"|"MAX") { $setType(AT_MAX); }
      | ("min"|"MIN") { $setType(AT_MIN);}
      );

protected
DIGIT:(’0’..’9’);

protected
CHAR_ATOM: (’a’..’z’|’A’..’Z’|’_’);
```

A.3 Empath Tree Walker (walker.g)

```
/*
*  walker.g : the AST walker for static semantic analysis
*  authors: sampada and nalini
*/

header {
  package edu.columbia.cs.coms4115.empath.antlr;
}

 {
   import java.util.*;
 }

 class EmpathTreeWalker extends TreeParser;
 options  {
   importVocab = Empath;
   defaultErrorHandler=false;
   ASTLabelType = "edu.columbia.cs.coms4115.empath.antlr.BaseAST";
 }

 {
  public String filename;
   SymbolTable entitySymTab=new SymbolTable();
   SymbolTable tempSymTab = entitySymTab;
   FunctionEntry fe = null;
  boolean inarglist = false;
  boolean islogexpn = false;
  boolean returned = false;
  boolean argsisthere = false;
  boolean notvoid = false;
  boolean isnotfunc = false;
  boolean triggerDef = false;

  DataRetType cur_type;
  public SymbolTable getSymTable() {
    return entitySymTab;
  }
 }

 entitySpec
   : #("entity"
   entityName:ID {
     tempSymTab.setName(entityName.getText());
     // System.out.println("about to walk the entity");
   }
   ("label" s:STRING_LIT
     {  tempSymTab.setLabel(s.getText());  }
   )?
   programBody)
   {
     // System.out.println("walking the entity");

   }
   ;

 programBody
   : #(PROGRAM
    decls
    funcDef
    stateDecls
    transitionDefs
    stateDefs
   )
   ;

 stateDecls
   : #(STATE_DECLS
     (sDecl)*
   )
   ;

 sDecl {boolean initFlag = false;}
   : #(S_DECL ("init" {initFlag = true;})? stateName:ID ("label" STRIN
G_LIT)?
```

```
   )
   {
     // System.out.println("walking the state "+stateName.getText());
     StateEntry se = new StateEntry(stateName.getText(),initFlag);
     if (tempSymTab.hasEntry(stateName.getText()) || !entitySymTab.ver
ifyUniqueState(stateName.getText()))  {
    // error msg: state re-declaration
      throw new SemanticException("State: "+stateName.getText()+" else
where in the program", filename, stateName.getLine(), stateName.getCol
umn());
     }
    if (initFlag && tempSymTab.hasInitState())  {
    // error msg: more than one init states
      throw new SemanticException("More than one initial states: "+sta
teName.getText(), filename, stateName.getLine(), stateName.getColumn()
);
    }
     tempSymTab.addEntry(stateName.getText(), se);
   }
   ;


 transitionDefs
   : #(TRANSITIONS
     (tDef)*
   )
   ;


 tDef
   : #(T_DEF
   {  DataRetType expr, action;
     boolean starState = false;
   }
     ( fromState:ID
     | MULT   {  starState = true;  }
     )
     toState:ID
     expr = e:expression
     (action=f:functionCall
       {if(action!=null)
         throw new SemanticException("Invalid transition action: "+f.g
etText(), filename, f.getLine(), f.getColumn());
       }
     )?
     )
     {
       if (!starState && fromState.getType()==ID)
         if (!tempSymTab.hasEntry(fromState.getText()))
        // error msg: from state not declared
          throw new SemanticException("State "+fromState.getText()+" h
as not been declared", filename, fromState.getLine(), fromState.getCol
umn());


       if (!tempSymTab.verifyState(toState.getText()))
      // error msg: to state not declared
        throw new SemanticException("State "+toState.getText()+" has n
ot been declared", filename, toState.getLine(), toState.getColumn());

//       check if expr returns boolean value
      if(expr != DataRetType.BOOLEAN)  {
        // error msg: not a boolean expression
        throw new SemanticException("Invalid transition condition: "+e
.getText(), filename, e.getLine(), e.getColumn());
      }
      starState = false;
    }
     ;

 stateDefs
   : #(STATE_DEFS
   (sDef)*
   )
   ;
 sDef
  : #(S_DEF stateName:ID
  {
   // System.out.println("Current sym tab "+tempSymTab.name);
   SymbolTableEntry ste=null;
   if (tempSymTab.hasEntry(stateName.getText()))  {
    ste = tempSymTab.getEntry(stateName.getText());
    if(ste.getType()==Type.STATE)  {
     SymbolTable stateSymTab = new SymbolTable(stateName.getText()
, tempSymTab, ste);
     ((StateEntry)ste).setChildSymTab(stateSymTab);
     tempSymTab = stateSymTab;
     // System.out.println("adding new child symbol table for "+te
mpSymTab.name);
    }
    else  {
    // error msg: state not declared earlier
     throw new SemanticException("State "+stateName.getText()+" ha
s not been declared earlier", filename, stateName.getLine(), stateName
.getColumn());
    }
   }
   else  {
  // error msg: state not declared earlier
   throw new SemanticException("State "+stateName.getText()+" has n
ot been declared earlier", filename, stateName.getLine(), stateName.ge
tColumn());
   }

  }
  ("label" s:STRING_LIT
   {  ste.setLabel(s.getText());  tempSymTab.setLabel(s.getText());
 }
  )?
  programBody
  )
  {
   // System.out.println("End of rule symTab: "+tempSymTab.name);
```

```
      if (tempSymTab.parent!=null)
        tempSymTab = tempSymTab.parent;
      // System.out.println("Switch to symTab: "+tempSymTab.name);
    }
    ;

functionCall returns [DataRetType t = null]
{
  int count=0;
  String c = null;
  DataRetType argstype = null;
}
  : #(
  fun_beg:F_CALL
  fun:ID
    {

      FunctionEntry f = tempSymTab.verifyFunc(fun.getText());
      if (f == null)
       {
        //error msg: function not declared earlier but being called

        throw new SemanticException("Undeclared function: "+fun.getTex
t(), filename, fun.getLine(), fun.getColumn());
       }

       if(f.getRetType()==DataRetType.RANGE)
        t = DataRetType.INT;
      else if(f.getRetType()==DataRetType.VOID)
        t = null;
      else
        t = f.getRetType();
    }
  ((id:ID | c = constant)
    {

      if(c == null)
      {
        argstype = tempSymTab.verifyVar(id.getText());
        if(argstype == null)
         {
  //         error msg: ID has not been declared
          throw new SemanticException("Undeclared variable: "+ id.getT
ext(), filename, id.getLine(), id.getColumn());

        }
      }
      else
      {
        if(c == "string")
          argstype = DataRetType.STRING;
        else if (c == "int")
          argstype = DataRetType.INT;
        else if(c == "float")
          argstype = DataRetType.FLOAT;
        else if (c == "boolean")
          argstype = DataRetType.BOOLEAN;
      }

      if(argstype == DataRetType.RANGE)
        argstype = DataRetType.INT;

      if(!f.verifyArgs(count , argstype))
      {
        throw new SemanticException("Invalid argument list: "+id.getTe
xt(), filename, id.getLine(), id.getColumn());
      }
      count = count + 1;
    }

  )*
  {
    int len = f.getArgsLength();
    if(len!=count)
    {
      // too few arguments
      throw new SemanticException("Invalid argument list: "+id.getText
(), filename, id.getLine(), id.getColumn());
    }
  }
  )
  ;



funcDef
  : #(FUNCDEF (func)*)
  ;

func
  : #(FUNC
  {  fe = new FunctionEntry();  }
  ("trigger"
    {  triggerDef = true;
      fe.addRetType("boolean");
      isnotfunc = true;
    }
  | "event"
    {
      fe.addRetType("void");
      isnotfunc = true;
    }
  | ("function" retType)
  )
  ("label" s:STRING_LIT
    {  fe.setLabel(s.getText());  }
  )?
  funcName:ID
  {
    //use hasentry here because we allow functions with same name to b
e defined in different scopes
      if (tempSymTab.hasEntry(funcName.getText()))
     {
      //error msg: function re-declaration
      throw new SemanticException("Function redeclaration: "+funcName.
getText(), filename, funcName.getLine(), funcName.getColumn());
     }
    else
    {

      tempSymTab.addEntry(funcName.getText(), fe);
      SymbolTable funcSymTab = new SymbolTable(funcName.getText(), tem
pSymTab, fe);
      ((FunctionEntry)fe).setlocalSymTab(funcSymTab);
      tempSymTab = funcSymTab;

    }

   }
  argsDef
  {
    if(((funcName.getText().compareTo("onExit")) == 0)||((funcName.get
Text().compareTo("onEntry")) == 0)||((funcName.getText().compareTo("on
ClockTick")) == 0))
    {
      if(argsisthere)
      {
        throw new SemanticException("Function cannot have argument lis
t: "+funcName.getText(), filename, funcName.getLine(), funcName.getCol
umn());

      }
      if(notvoid)
      {
        throw new SemanticException("Invalid return type for function:
 "+funcName.getText(), filename, funcName.getLine(), funcName.getColum
n());
      }
      if(isnotfunc)
      {
        throw new SemanticException("Function cannot be a trigger or e
vent: "+funcName.getText(), filename, funcName.getLine(), funcName.get
Column());
      }
    }
    argsisthere = false;
    notvoid = false;
    isnotfunc = false;
  }
  funcBody
  )
  {
    //System.out.println("gOIONG TO CHECK");
    if((returned == false)&&(fe.getRetType() != DataRetType.VOID))
    {
      throw new SemanticException("Missing return statement: "+funcNam
e.getText(), filename, funcName.getLine(), funcName.getColumn());

    }
    returned = false;
    // System.out.println("End of rule symTab: "+tempSymTab.name);
     if (tempSymTab.parent!=null)
       tempSymTab = tempSymTab.parent;
     // System.out.println("Switched back to old symTab: "+tempSymTab.
name);
    fe = null;
    triggerDef = false;
  }
  ;

retType
{String t;}
  : ("void"
    {  fe.addRetType("void");  }
  )
  | (t=type
    {
      fe.addRetType(t);
      notvoid = true;
    }
  )
  ;


type returns [String s]
{  String a, b;  }
  : "int"
    {  s="int";  }
  | "float"
    {  s="float";  }
  | "string"
    {  s="string";  }
  | "boolean"
    {  s="boolean";  }
  | r:"range" a=constant b=constant
    {  s="range";
      if(!(a.equals("int") && b.equals("int")))
        throw new SemanticException("Illegal range value types "+a+",
"+b, filename, r.getLine(), r.getColumn());
    }
  ;

argsDef
  { String t; }
  : #(ARGS_DEF
    (t=type
      {
        fe.addArgs(t);
        argsisthere = true;
      }
     argName:ID
       {
         //add ID and type to localsymtab
         if (tempSymTab.hasEntry(argName.getText()))
         {
//           error msg: ID is already defined in current scope
          throw new SemanticException("Variable already defined in cur
rent scope: "+argName.getText(), filename, argName.getLine(), argName.
getColumn());
         }
        else
        {
          VariableEntry ve = new VariableEntry();
          tempSymTab.addEntry(argName.getText(), ve);
          ve.addIDType(t);
        }
      }
    )*
  )
  ;


funcBody
  : #(FUNCBODY
  decls
  (statement)*
  )
  ;

decls
  : #(DECLS
  (decl)*
  )
  ;


decl
{String t , c = null;}
  :#(DECL
  t=type
  ( declID:ID (c = constant)?
    {
      //add ID and type to localsymtab
      if (tempSymTab.hasEntry(declID.getText()))
       {
//         error msg: ID is already defined in current scope
```

```
    throw new SemanticException("Variable already defined in curre
nt scope: "+declID.getText(), filename, declID.getLine(), declID.getCo
lumn());
   }
   else
   {
    if((c==null)||(c==t)||((t=="range")&&(c=="int")))
    {
     VariableEntry ve = new VariableEntry();
     tempSymTab.addEntry(declID.getText(), ve);
     ve.addIDType(t);
    }
    else
    {
//        error msg: const assigned is not of same type as ID
     throw new SemanticException("Invalid value assigned to varia
ble: "+declID.getText(), filename, declID.getLine(), declID.getColumn(
));

    }
   }
  }
 )*)
 ;


statementBlock :
 #(STMTBLK
 (statement)+
 )
 ;


statement
{
 cur_type = null;
 DataRetType expr_type = null;
 DataRetType ret_type = null;
 String c = null;
}
 :
 ( expr_type = expr:expression
  {

   if(expr_type!=null)
   {
    //error msg:  stmt of type b+c;
    throw new SemanticException("Invalid statement: "+expr.getText
(), filename, expr.getLine(), expr.getColumn());
   }

  }
 )
 | #( "if"
 expr_type = e:expression
  {
   //checking that e's child is a logical operator
   if(expr_type != DataRetType.BOOLEAN)
   {
    throw new SemanticException("Invalid condition for if statemen
t: "+e.getText(), filename, e.getLine(), e.getColumn());
   }
  }
 statementBlock
 (statementBlock )?
 )
 | #( "for"
 expr_type = e1:expression
  {
   //check that e1's child is ASSIGN
   if(expr_type != null)
   {
    throw new SemanticException("Invalid initialization expression
 in for loop: "+e1.getText(), filename, e1.getLine(), e1.getColumn());

   }
```

```
    }
   expr_type = e2:expression
    {
      //check that e2's child is 1 of the comparison operators
      if(expr_type != DataRetType.BOOLEAN)
      {
        throw new SemanticException("Invalid conditional expression in
 for loop: "+e2.getText(), filename, e2.getLine(), e2.getColumn());
       }
    }
   expr_type = e3:expression
    {
      //check that e3's child is some sort of value updation
      if(expr_type != null)
      {
        throw new SemanticException("Invalid increment expression in f
or loop: "+e3.getText(), filename, e3.getLine(), e3.getColumn());
       }
    }
   statementBlock
   )
   | #("while"
   expr_type = ex:expression
    {
      //check that ex's child is a logical operator
      if(expr_type != DataRetType.BOOLEAN)
      {
        throw new SemanticException("Invalid condition for while loop:
 "+ex.getText(), filename, ex.getLine(), ex.getColumn());
       }
    }
   statementBlock
   )
   | #("output"
   {
     DataRetType optype = null;
   }
   (c=con:constant | outid:ID)
```

```
  {
   if(c == null)
   {
    optype = tempSymTab.verifyVar(outid.getText());
    if (optype == null)
     {

     throw new SemanticException("Variable not defined: "+ outid.
getText(), filename, outid.getLine(), outid.getColumn());
     }
     if(optype != DataRetType.STRING)
     {

     throw new SemanticException("Invalid output: "+ outid.getTex
t(), filename, outid.getLine(), outid.getColumn());
     }
   }
   else
   {
    if(c != "string")
    {
     //error msg: ID is already defined in current scope
     throw new SemanticException("Invalid output: "+ con.getText(
), filename, con.getLine(), con.getColumn());
    }
   }
  }
 )
 | #(retstmt:"return"
  {
   returned = true;
   ret_type = fe.getRetType();

   //check that if cur_type is void then there is no following expr

   AST ret_child = retstmt.getFirstChild();
   if((ret_type==DataRetType.VOID)&&(ret_child!=null))
   {
    //error msg: return value for function whose return value is v
oid
    throw new SemanticException("Invalid return value: "+retstmt.g
etText(), filename, retstmt.getLine(), retstmt.getColumn());
   }

  }
 (expr_type = expression
  {
   if(ret_type!=expr_type)
   {
    //error msg: invalid return value
    throw new SemanticException("Invalid return value: "+retstmt.g
etText(), filename, retstmt.getLine(), retstmt.getColumn());
   }
  }
 )?
 )
 ;


expression returns [DataRetType t = null]
 :#(EXPR
  (
  ("tick" INT_LIT
   {  t = DataRetType.BOOLEAN;  }
  )
  |
  t = assignmentExpression
  )
 )
 ;

assignmentExpression returns [DataRetType t = null]
{
 DataRetType idtype , rhs = null;
 int line = 0 , col = 0;
 String IDname = null , value = null;
```

```
}
  :
  (
  (
  #(ASSIGN
  varname1:ID
  {  boolean globalVarRef = false;
    if (!tempSymTab.hasEntry(varname1.getText()))  {  // global variab
le referenced
        globalVarRef = true;
      }
    if (triggerDef && globalVarRef)
      throw new SemanticException("Not allowed to change characteristi
c values in a trigger function "+varname1.getText(), filename, varname
1.getLine(), varname1.getColumn());
  }

  rhs = rhs1:rthandexpr
    {
      IDname = varname1.getText();
      line = varname1.getLine();
      col = varname1.getColumn();
      value = rhs1.getText();
    }
  )
  |#(PLUSEQ
  varname2:ID
  {  boolean globalVarRef = false;
    if (!tempSymTab.hasEntry(varname2.getText()))  {  // global variab
le referenced
        globalVarRef = true;
      }
    if (triggerDef && globalVarRef)
      throw new SemanticException("Not allowed to change characteristi
c values in a trigger function "+varname2.getText(), filename, varname
2.getLine(), varname2.getColumn());
  }
  rhs = rhs2:rthandexpr
    {
      IDname = varname2.getText();
      line = varname2.getLine();
      col = varname2.getColumn();
      value = rhs2.getText();
    }
  )
  | #(MINUSEQ
  varname3:ID
  {  boolean globalVarRef = false;
    if (!tempSymTab.hasEntry(varname3.getText()))  {  // global variab
le referenced
        globalVarRef = true;
      }
    if (triggerDef && globalVarRef)
      throw new SemanticException("Not allowed to change characteristi
c values in a trigger function "+varname3.getText(), filename, varname
3.getLine(), varname3.getColumn());
  }

  rhs = rhs3:rthandexpr
    {
      IDname = varname3.getText();
      line = varname3.getLine();
      col = varname3.getColumn();
      value = rhs3.getText();
    }
  )
  | #(MULTEQ
  varname4:ID
  {  boolean globalVarRef = false;
    if (!tempSymTab.hasEntry(varname4.getText()))  {  // global variab
le referenced
        globalVarRef = true;
      }
    if (triggerDef && globalVarRef)
```

```
   throw new SemanticException("Not allowed to change characteristi
c values in a trigger function "+varname4.getText(), filename, varname
4.getLine(), varname4.getColumn());
 }

 rhs = rhs4:rthandexpr
  {
   IDname = varname4.getText();
   line = varname4.getLine();
   col = varname4.getColumn();
   value = rhs4.getText();
  }
 )
 | #(DIVEQ
 varname5:ID
 {  boolean globalVarRef = false;
  if (!tempSymTab.hasEntry(varname5.getText()))  {  // global variab
le referenced
    globalVarRef = true;
   }
  if (triggerDef && globalVarRef)
   throw new SemanticException("Not allowed to change characteristi
c values in a trigger function "+varname5.getText(), filename, varname
5.getLine(), varname5.getColumn());
 }

 rhs = rhs5:rthandexpr
  {
   IDname = varname5.getText();
   line = varname5.getLine();
   col = varname5.getColumn();
   value = rhs5.getText();
  }
 )
 | #(MODEQ
 varname6:ID
 {  boolean globalVarRef = false;
  if (!tempSymTab.hasEntry(varname6.getText()))  {  // global variab
```

```
le referenced
        globalVarRef = true;
      }
    if (triggerDef && globalVarRef)
      throw new SemanticException("Not allowed to change characteristi
c values in a trigger function "+varname6.getText(), filename, varname
6.getLine(), varname6.getColumn());
  }

  rhs = rhs6:rthandexpr
    {
      IDname = varname6.getText();
      line = varname6.getLine();
      col = varname6.getColumn();
      value = rhs6.getText();
    }
  )

  |(t = lhe:rthandexpr)
  )
    {

      if(IDname!=null)
      {
        idtype = tempSymTab.verifyVar(IDname);
        if(idtype == null)
        {
  //          error msg: ID has not been declared
          throw new SemanticException("Undeclared variable: "+IDname,
filename, line, col);

        }

        if(((idtype==DataRetType.RANGE)&&(rhs!=DataRetType.INT))||((id
type!=DataRetType.RANGE)&&(rhs!=idtype)))
        {
          //error msg: invalid expression type
          throw new SemanticException("Mismatch in variable and assign
ed expression type: "+value, filename, line , col);
        }
        t = null;
      }
    }
  )
  ;

rthandexpr returns [DataRetType t = null]
{

  DataRetType leftsub = null , rightsub = null;
}
  :(
  #(OR
  leftsub = lt1:rthandexpr
  rightsub = rt1:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt1.getText(), filename , lt1.getLine(), lt1.getColumn());
      }
      if(leftsub!=DataRetType.BOOLEAN)
      {
        throw new SemanticException("Invalid operand: "+lt1.getText(),
 filename , lt1.getLine(), lt1.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(AND
  leftsub = lt2:rthandexpr
  rightsub = rt2:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt2.getText(), filename , lt2.getLine(), lt2.getColumn());
      }
      if(leftsub!=DataRetType.BOOLEAN)
      {
        throw new SemanticException("Invalid operand: "+lt2.getText(),
 filename , lt2.getLine(), lt2.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(EQ
  leftsub = lt3:rthandexpr
  rightsub = rt3:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt3.getText(), filename , lt3.getLine(), lt3.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(NEQ
  leftsub = lt4:rthandexpr
  rightsub = rt4:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt4.getText(), filename , lt4.getLine(), lt4.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(LT
  leftsub = lt5:rthandexpr
  rightsub = rt5:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt5.getText(), filename , lt5.getLine(), lt5.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt5.getText(),
 filename , lt5.getLine(), lt5.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(LEQ
  leftsub = lt6:rthandexpr
  rightsub = rt6:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt6.getText(), filename , lt6.getLine(), lt6.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt5.getText(),
 filename , lt5.getLine(), lt5.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(GT
  leftsub = lt7:rthandexpr
  rightsub = rt7:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt7.getText(), filename , lt7.getLine(), lt7.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt7.getText(),
 filename , lt7.getLine(), lt7.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
  )
  | #(GEQ
  leftsub = lt8:rthandexpr
  rightsub = rt8:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt8.getText(), filename , lt8.getLine(), lt8.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt8.getText(),
 filename , lt8.getLine(), lt8.getColumn());
      }
      t = DataRetType.BOOLEAN;
    }
   )
  | #(PLUS
  leftsub = lt9:rthandexpr
  rightsub = rt9:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt9.getText(), filename , lt9.getLine(), lt9.getColumn());
```

```
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt9.getText(),
 filename , lt9.getLine(), lt9.getColumn());
      }
      t = leftsub;
    }
  )
  | #(MINUS
  leftsub = lt10:rthandexpr
  rightsub = rt10:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt10.getText(), filename , lt10.getLine(), lt10.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt9.getText(),
 filename , lt9.getLine(), lt9.getColumn());
      }
      t = leftsub;
    }
  )
  | #(MULT
  leftsub = lt11:rthandexpr
  rightsub = rt11:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt11.getText(), filename , lt11.getLine(), lt11.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt9.getText(),
 filename , lt9.getLine(), lt9.getColumn());
      }
      t = leftsub;
    }
  )
  | #(DIV
  leftsub = lt12:rthandexpr
  rightsub = rt12:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt12.getText(), filename , lt12.getLine(), lt12.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt9.getText(),
 filename , lt9.getLine(), lt9.getColumn());
      }
      t = leftsub;
    }
  )
  | #(MOD
  leftsub = lt13:rthandexpr
  rightsub = rt13:rthandexpr
    {
      if(leftsub!=rightsub)
      {
        throw new SemanticException("Mismatching operands in expressio
n: "+lt13.getText(), filename , lt13.getLine(), lt13.getColumn());
      }
      if((leftsub==DataRetType.BOOLEAN)||(leftsub==DataRetType.STRING)
)
      {
        throw new SemanticException("Invalid operand: "+lt9.getText(),
 filename , lt9.getLine(), lt9.getColumn());
      }
      t = leftsub;
    }
  )
  | ( #(lhatmax:AT_MAX
    {
      AST atmax_child = lhatmax.getFirstChild();
      if(atmax_child.getType()!=ID)
      {
        //error msg: invalid operand to @max operator
        throw new SemanticException("Invalid OPERAND: "+lhatmax.getTex
t(), filename , lhatmax.getLine(), lhatmax.getColumn());
      }

      leftsub = tempSymTab.verifyVar(atmax_child.getText());
      if(leftsub!= DataRetType.RANGE)
      {
        //error msg: invalid use of @max on non-compatible type
        throw new SemanticException("Invalid operand: "+lhatmax.getTex
t(), filename , lhatmax.getLine(), lhatmax.getColumn());
      }

    }
  t = rthandexpr)
  {  t = DataRetType.BOOLEAN;  }
  )
  | ( #(lhatmin:AT_MIN
    {
      AST atmin_child = lhatmin.getFirstChild();
      if(atmin_child.getType()!=ID)
      {
        //error msg: invalid operand to @min operator
        throw new SemanticException("Invalid operand: "+lhatmin.getTex
t(), filename , lhatmin.getLine(), lhatmin.getColumn());
      }
      leftsub = tempSymTab.verifyVar(atmin_child.getText());
      if(leftsub != DataRetType.RANGE)
      {
        //error msg: invalid use of @min on non-compatible type
        throw new SemanticException("Invalid operand: "+lhatmin.getTex
t(), filename , lhatmin.getLine(), lhatmin.getColumn());
      }
    }
  t = rthandexpr)
  {  t = DataRetType.BOOLEAN;  }
  )
  |( #(lhincr:INCR
    {
      AST incr_child = lhincr.getFirstChild();
      if(incr_child.getType()!=ID)
      {
        //error msg: invalid operand to ++ operator
        throw new SemanticException("Invalid operand: "+lhincr.getText
(), filename , lhincr.getLine(), lhincr.getColumn());
      }

      leftsub = tempSymTab.verifyVar(incr_child.getText());
      if((leftsub == DataRetType.BOOLEAN)||(leftsub == DataRetType.STR
ING))
      {
        //error msg: invalid use of ++ on non-compatible type
        throw new SemanticException("Invalid operand: "+lhincr.getText
(), filename , lhincr.getLine(), lhincr.getColumn());
      }
      boolean globalVarRef = false;
      if (!tempSymTab.hasEntry(incr_child.getText()))  // global varia
ble referenced
          globalVarRef = true;
        if(triggerDef && globalVarRef)
        throw new SemanticException("Not allowed to change characteris
tic values in a trigger function "+incr_child.getText(), filename , lh
incr.getLine(), lhincr.getColumn());
      globalVarRef = false;
  }
 t = rthandexpr
  {
   t = null;
  }
 )
 )
 |( #(lhdecr:DECR
  {
   AST decr_child = lhdecr.getFirstChild();
   if(decr_child.getType()!=ID)
   {
    //error msg: invalid operand to -- operator
    throw new SemanticException("Invalid operand: "+lhdecr.getText
(), filename , lhdecr.getLine(), lhdecr.getColumn());
   }

   leftsub = tempSymTab.verifyVar(decr_child.getText());
   if((leftsub == DataRetType.BOOLEAN)||(leftsub == DataRetType.STR
ING))
   {
    //error msg: invalid use of -- on non-compatible type
    throw new SemanticException("Invalid operand: "+lhdecr.getText
(), filename , lhdecr.getLine(), lhdecr.getColumn());
   }
   boolean globalVarRef = false;
   if (!tempSymTab.hasEntry(decr_child.getText()))  // global varia
ble referenced
     globalVarRef = true;
    if(triggerDef && globalVarRef)
    throw new SemanticException("Not allowed to change characteris
tic values in a trigger function "+decr_child.getText(), filename , lh
decr.getLine(), lhdecr.getColumn());

  }
 t = rthandexpr
  {
      t = null;
    }
  )
  )
  | t = val
  )
  ;

val returns [DataRetType t = null]
:
{  String c;  }
  ( c = constant
     {
       if(c=="int")
      {
        t = DataRetType.INT;
      }
      if(c=="string")
      {
        t = DataRetType.STRING;
      }
      if(c=="float")
      {
        t = DataRetType.FLOAT;
      }
      if(c=="boolean")
      {
        t = DataRetType.BOOLEAN;
      }
     }
  )
    | ( var:ID
    {
      DataRetType idtype = null;
      idtype = tempSymTab.verifyVar(var.getText());
      if(idtype == null)
      {
        //error msg: undefined variable
      throw new SemanticException("Undefined variable: "+var.getText()
, filename , var.getLine(), var.getColumn());
      }
      else
      {
        if(idtype==DataRetType.RANGE)
        {
            t = DataRetType.INT;
        }
        else
        {
               t = idtype;
        }

      }
    }
    )
    | (t = functionCall
    {
      //t = null;
    }
  )
    ;

constant returns [String c]
  : #(UPLUS c = a:cons)
    { if (!(c.equals("int") || c.equals("float")))
      throw new SemanticException("Invalid operand to unary plus ",fil
ename,a.getLine(),a.getColumn());
    }
  | #(UMINUS c = cons)
    { if (!(c.equals("int") || c.equals("float")))
      throw new SemanticException("Invalid operand to unary minus ",fi
lename,a.getLine(),a.getColumn());
    }
  | c = cons
  ;
```

```
cons returns [String c]
  : ( STRING_LIT
  {  c = "string";  }
  )
  | ( INT_LIT
  {  c = "int";  }
  )
  | ( REAL_LIT
  {  c = "float";  }
  )
  | (  "true"
  {  c = "boolean";  }
  )
    | ( "false"
    {  c = "boolean";  }
  )
    | ( "null"
    {  c = "string";  }
  )
  ;
```

A.4 Example Empath Code: dog.emp

```
/* dog entity test for code generation */

entity Dog label "gnarly toothed shaggy mutt" {

  string name;
  float age = 0.0;
  range [0:10] happiness = 5;
  range [0:20] hunger = 4;

  // maximum age
  float MAX_AGE = 20.0;

  /* time trigger */
  function void onClockTick() {
    if (tick 2) {
      // increase hunger and age
      hunger++;
      age += 0.5;
    } else {
      // decrease happiness
      happiness--;
    }
  }

  /* feed event */
  event label "feed" feed(int quantity)  {
    hunger-=quantity;
  }

  /* trigger for the dog starving */
  trigger starve() {
    if (hunger>16) {
      return true;
    }
    return false;
  }
```

```
/* output of the dying transition */
function void bury() {
  output("the dog has been buried");
}

state init DogPuppy, DogAdult;
state DogDead;

/* transitions */
transition DogPuppy to DogDead if (starve()) / bury() ;
transition DogAdult to DogDead if (age>=MAX_AGE || hunger @max);


/* ------------------------ Puppy State ------------------------- */

DogPuppy label "puppy" {

  range [0:100] milkDrunk=0;

  function void onClockTick() {
    happiness--;
  }

  function void onEntry() {
    happiness = 10;
    output("the puppy has been born");
  }

  function void onExit() {
    output("the puppy has grown up");
  }

  // output if the puppy did not get enough milk by a certain age
  function void notEnoughMilk() {
    output("the puppy did not get enough milk");
  }
  event label "play with the puppy" play() {
   happiness = 10;
  }

  event label "give the puppy milk" suckle() {
   milkDrunk+=3;
  }

  /* states Whining and Hungry have no further definition */
  state init DogPuppyWaggingTail, DogPuppyWhining, DogPuppyHungry;

  transition
   DogPuppyWaggingTail to DogPuppyHungry if (hunger >= 8),
   DogPuppyWaggingTail to DogPuppyWhining if (happiness <=3),
   DogPuppyHungry to DogPuppyWaggingTail if (hunger < 3),
   DogPuppyWhining to DogPuppyWaggingTail if (happiness > 7),
   DogPuppyWhining to DogPuppyHungry if (hunger >=8);
  transition * to DogAdult if (milkDrunk>5 && age>=3.0);
  // the puppy dies if it doesn't get enough milk
  transition * to DogDead if (milkDrunk<=5 && age>=3.5) / notEnoughM
ilk();

  DogPuppyWaggingTail {
   function void onEntry() {
    happiness = 10;
   }
  }
 }

 /* ----------------------- Adult State ------------------------ */

 DogAdult label "grown dog" {

  int timeSinceLastWalk = 0;

  function void onClockTick() {
   hunger += 1;
   timeSinceLastWalk++;
```

```
    }

    event walk() {
      timeSinceLastWalk = 0;
    }

    state DogAdultBarking , DogAdultHungry , DogAdultNeedsWalk;

    transition
      DogAdultBarking to DogAdultHungry if (hunger >= 7),
      DogAdultHungry to DogAdultBarking if (hunger <= 3),
      DogAdultBarking to DogAdultNeedsWalk if (timeSinceLastWalk > 10
),
      DogAdultNeedsWalk to DogAdultBarking if (timeSinceLastWalk < 5),

      DogAdultNeedsWalk to DogAdultHungry if (hunger >=8);
    transition * to DogDead if (hunger @max);

    DogAdultBarking {

      function void onEntry() {
        timeSinceLastWalk = 0;
      }

    }
  }

  /* the Dead state */
  DogDead label "was once a dog" {

    function void onEntry() {
      /* make the owner feel bad if the dog starved to death */
      if (age<MAX_AGE)
        output ("Shame on you!! You killed the dog!");
      else
        output ("the dog died of old age");
    }
  }
```

```
}
```

A.5 Example Empath Code: frog.emp

```
/* a simple "hello world" level example
*/
entity Frog {
  int age = 0;

  function void onClockTick() {
    age++;
  }

  trigger hatch() {
    if (age > 2) {
      return true;
    }
    return false;
  }

  trigger isMature() {
    if (age > 5) {
      return true;
    }
    return false;
  }

  state Egg, Tadpole, AdultFrog, Dead;

  transition Egg to Tadpole if (hatch());
  transition Tadpole to AdultFrog if (age > 5);
  transition AdultFrog to Dead if (age > 7);

}
```

A.6 Example Empath Code: grad_student.emp

```
entity Grad_student {

  /* Following are the characteristic variables of
  *  the set of states at the highest(entity) level
  */
  range [0:10] hunger = 0;
  int assignments_due = 0;
  boolean needs_a_break = false;

  //This function will get executed every clock cycle
  function void onClockTick() {
    hunger++;
    /*
    * The boolean expression in the following if
    * conditional evaluates to true for every 3rd clock cycle
    */
    if(tick 3) {
      assignments_due ++;
    }
  }

  //Events are special functions that specify a change on user input
  event feed() {
    hunger -= 2;
  }

  event work() {
    assignments_due--;
  }

  event chill() {
    needs_a_break = true;
  }

  function void bury() {
    //sob sob :'(
  }

  state Slaving, ChillingOut, Dead;

  transition
      Slaving to ChillingOut if(needs_a_break),
      ChillingOut to Slaving if(assignments_due >= 5);

  //Nested definition of the Slaving state
  Slaving  {

    boolean wake_up_alarm = false;

    event wake_up() {
      wake_up_alarm = true;
    }

    //A trigger which returns true if the student is overloaded
    trigger overload() {
      if((assignments_due == 10)||(hunger@max))
        return true;
      else
        return false;
    }

    state Sleeping, NeedToWork , DozingInClass , Hungry;

    transition
        Sleeping to DozingInClass if(wake_up_alarm),
        Sleeping to NeedToWork if (assignments_due >= 5),
        Sleeping to Hungry if (hunger>=8),
        DozingInClass to Hungry if (hunger >= 8),
        DozingInClass to NeedToWork if (assignments_due >= 5),
        Hungry to NeedToWork if (assignments_due >= 5),
        Hungry to Sleeping if (hunger <= 7),
        NeedToWork to Sleeping if (assignments_due <= 5),
        * to Dead if (overload()) / bury();
    DozingInClass {

      /*
      *  Special function onEntry which gets executed each time the
      *  DozingInClass state becomes the current active state
      */
      function void onEntry() {
        wake_up_alarm = false;
      }

    }

  }

  //Nested definition of state ChillingOut

  ChillingOut {

    int time = 0;

    function void onClockTick() {
      time += 10;
    }

    function void onEntry() {
      needs_a_break = false;
    }

    state Partying , init Comatose;

    transition Partying to Comatose if(time%7 == 0);
    transition Comatose to Partying if(time%10 == 0);

  }
}
```

A.7 Example Empath Code: squirrel.emp

```
entity Squirrel label "squirrel" {

  int age=0;

  function void onClockTick() {
    age+=1;
  }

  state init SquirrelBaby, SquirrelAdult, SquirrelDead;

  transition SquirrelBaby to SquirrelAdult if (age>2);
  transition SquirrelAdult to SquirrelDead if (age>20);

  SquirrelAdult {

    string season="spring";
    range [1:10] hunger=0;
    int nutsGathered=0;


    event label "give nuts" feed(string nuttype, int nutcount) {
      hunger -= nutcount;
    }

    event label "gather nuts" gather(string nuttype, int nutcount) {
      nutsGathered += nutcount;
    }

    function void onClockTick() {

      if (tick 4) {

        if (season == "spring") {
          season = "summer";
        } else if (season == "summer") {
          season = "autumn";
        } else if (season == "autumn") {
          season = "winter";
        } else if (season == "winter") {
          season = "spring";
        }

      }

    }

    state SquirrelAdultGathering, SquirrelAdultHibernating, SquirrelAd
ultFrisky, init SquirrelAdultHungry;

    transition SquirrelAdultGathering to SquirrelAdultHibernating if (
season == "winter");
    transition SquirrelAdultHibernating to SquirrelAdultHungry if (sea
son == "spring");
    transition SquirrelAdultHungry to SquirrelAdultFrisky if (season =
= "summer");
    transition SquirrelAdultFrisky to SquirrelAdultGathering if (seaso
n == "autumn");

  }


}
```

A.8 Java Source: BaseAST

```
package edu.columbia.cs.coms4115.empath.antlr;

import antlr.CommonAST;
import antlr.Token;
import antlr.collections.AST;

/**
 * Provides line and column number definitions
 *
 * @author William Mee (wjm2107)
 *
 */
public class BaseAST extends CommonAST {

  /**
   * serial number
   */
  private static final long serialVersionUID = -5518412207944590562L;
  private int _line;
  private int _column;

  @Override
  public void initialize(Token token) {
    super.initialize(token);
    _line=token.getLine();
    _column=token.getColumn();
  }

  public int getColumn() {
    return _column;
  }

  public int getLine() {
    int line=_line;
    if (line!=0) {
      return line;
    }
    // try and find a child with non-zero line number
    AST child=getFirstChild();
    if (child!=null) {
      line=child.getLine();
      if (line!=0) {
        return line;
      }
    }
    // try and find a sibling with non-zero line number
    AST sibling = getNextSibling();
    while (sibling!=null) {
      line=sibling.getLine();
      if (line!=0) {
        return line;
      }
      sibling = getNextSibling();
    }
    return line;
  }

}
```

A.9 Java Source: CodeGenException

```
package edu.columbia.cs.coms4115.empath.codegen;

/**
 * This exception (or a subclass) is thrown during the code generation

 * @author Sharika
 *
 */
public class CodeGenException extends Exception {

  public CodeGenException(String message, Throwable cause) {
    super(message, cause);
  }

  public CodeGenException(String message) {
    super(message);
  }

  public CodeGenException(Throwable cause) {
    super(cause);
  }


}
```

A.10 Java Source: CodeGenWalker

```
// $ANTLR : "codegen.g" -> "CodeGenWalker.java"$

  package edu.columbia.cs.coms4115.empath.antlr;

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

   import java.util.*;
   import edu.columbia.cs.coms4115.empath.codegen.*;


public class CodeGenWalker extends antlr.TreeParser       implements C
odeGenWalkerTokenTypes
 {

/*  public String filename;
   SymbolTable entitySymTab=new SymbolTable();
   SymbolTable tempSymTab=entitySymTab;
   FunctionEntry fe = null;
  boolean inarglist = false;
  boolean isstmt = false;
  public SymbolTable getSymTable() {
    return entitySymTab;
  }*/
  public String filename;
  SymbolTable entitySymTab;
   SymbolTable tempSymTab;
  public void setSymTable(SymbolTable symTab)  {
    entitySymTab = symTab;
    tempSymTab = entitySymTab;
  }
  String range_lower=null, range_upper=null;
  int tabCount = 2;
  boolean funcCall = false;
  boolean funcCond = false;
  public SymbolTable getSymTable() {
    return entitySymTab;
  }
 public CodeGenWalker() {
  tokenNames = _tokenNames;
}

  public final void entitySpec(AST _t) throws RecognitionException {

    AST entitySpec_AST_in = (_t == ASTNULL) ? null : (AST)_t;
    AST entityName = null;

    AST __t2 = _t;
    AST tmp1_AST_in = (AST)_t;
    match(_t,LITERAL_entity);
    _t = _t.getFirstChild();

        StringBuffer a;

    entityName = (AST)_t;
    match(_t,ID);
    _t = _t.getNextSibling();


    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case LITERAL_label:
    {
      AST tmp2_AST_in = (AST)_t;
      match(_t,LITERAL_label);
      _t = _t.getNextSibling();
      AST tmp3_AST_in = (AST)_t;
      match(_t,STRING_LIT);
      _t = _t.getNextSibling();
      break;
    }
    case PROGRAM:
    {
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    }
    programBody(_t);
    _t = _retTree;
    _t = __t2;
    _t = _t.getNextSibling();
    _retTree = _t;
  }

  public final void programBody(AST _t) throws RecognitionException {

    AST programBody_AST_in = (_t == ASTNULL) ? null : (AST)_t;
        StringBuffer a;  // dummy


    AST __t5 = _t;
    AST tmp4_AST_in = (AST)_t;
    match(_t,PROGRAM);
    _t = _t.getFirstChild();
    a=decls(_t);
    _t = _retTree;
    funcDef(_t);
    _t = _retTree;
```

```
   stateDecls(_t);
   _t = _retTree;
   transitionDefs(_t);
   _t = _retTree;
   stateDefs(_t);
   _t = _retTree;
   _t = __t5;
   _t = _t.getNextSibling();
   _retTree = _t;
 }

 public final StringBuffer  decls(AST _t) throws RecognitionException
{
   StringBuffer r;

   AST decls_AST_in = (_t == ASTNULL) ? null : (AST)_t;

       r = new StringBuffer();
       StringBuffer a;


   AST __t54 = _t;
   AST tmp5_AST_in = (AST)_t;
   match(_t,DECLS);
   _t = _t.getFirstChild();
   {
   _loop56:
   do {
     if (_t==null) _t=ASTNULL;
     if ((_t.getType()==DECL)) {
       a=decl(_t);
       _t = _retTree;
         r.append(a);
     }
     else {
       break _loop56;
     }
```

```
  } while (true);
  }
  _t = __t54;
  _t = _t.getNextSibling();
  _retTree = _t;
  return r;
}

public final void funcDef(AST _t) throws RecognitionException {

  AST funcDef_AST_in = (_t == ASTNULL) ? null : (AST)_t;

  AST __t34 = _t;
  AST tmp6_AST_in = (AST)_t;
  match(_t,FUNCDEF);
  _t = _t.getFirstChild();
  {
  _loop36:
  do {
    if (_t==null) _t=ASTNULL;
    if ((_t.getType()==FUNC)) {
      func(_t);
      _t = _retTree;
    }
    else {
      break _loop36;
    }

  } while (true);
  }
  _t = __t34;
  _t = _t.getNextSibling();
  _retTree = _t;
}

public final void stateDecls(AST _t) throws RecognitionException {

  AST stateDecls_AST_in = (_t == ASTNULL) ? null : (AST)_t;
```

```
   AST __t7 = _t;
   AST tmp7_AST_in = (AST)_t;
   match(_t,STATE_DECLS);
   _t = _t.getFirstChild();
   {
   _loop9:
   do {
     if (_t==null) _t=ASTNULL;
     if ((_t.getType()==S_DECL)) {
       sDecl(_t);
       _t = _retTree;
     }
     else {
       break _loop9;
     }

   } while (true);
   }
   _t = __t7;
   _t = _t.getNextSibling();
   _retTree = _t;
 }

 public final void transitionDefs(AST _t) throws RecognitionException
{

   AST transitionDefs_AST_in = (_t == ASTNULL) ? null : (AST)_t;

   AST __t15 = _t;
   AST tmp8_AST_in = (AST)_t;
   match(_t,TRANSITIONS);
   _t = _t.getFirstChild();
   {
   _loop17:
   do {
     if (_t==null) _t=ASTNULL;
     if ((_t.getType()==T_DEF)) {
       tDef(_t);
       _t = _retTree;
     }
     else {
       break _loop17;
     }

   } while (true);
   }
   _t = __t15;
   _t = _t.getNextSibling();
   _retTree = _t;
 }

 public final void stateDefs(AST _t) throws RecognitionException {

   AST stateDefs_AST_in = (_t == ASTNULL) ? null : (AST)_t;

   AST __t23 = _t;
   AST tmp9_AST_in = (AST)_t;
   match(_t,STATE_DEFS);
   _t = _t.getFirstChild();
   {
   _loop25:
   do {
     if (_t==null) _t=ASTNULL;
     if ((_t.getType()==S_DEF)) {
       sDef(_t);
       _t = _retTree;
     }
     else {
       break _loop25;
     }

   } while (true);
   }
   _t = __t23;
   _t = _t.getNextSibling();
   _retTree = _t;
 }

 public final void sDecl(AST _t) throws RecognitionException {

   AST sDecl_AST_in = (_t == ASTNULL) ? null : (AST)_t;
   AST stateName = null;
   boolean initFlag = false;

   AST __t11 = _t;
   AST tmp10_AST_in = (AST)_t;
   match(_t,S_DECL);
   _t = _t.getFirstChild();
   {
   if (_t==null) _t=ASTNULL;
   switch ( _t.getType()) {
   case LITERAL_init:
   {
     AST tmp11_AST_in = (AST)_t;
     match(_t,LITERAL_init);
     _t = _t.getNextSibling();
     initFlag = true;
     break;
   }
   case ID:
   {
     break;
   }
   default:
   {
     throw new NoViableAltException(_t);
   }
   }
   }
   stateName = (AST)_t;
   match(_t,ID);
   _t = _t.getNextSibling();
   {
```

```
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case LITERAL_label:
  {
    AST tmp12_AST_in = (AST)_t;
    match(_t,LITERAL_label);
    _t = _t.getNextSibling();
    AST tmp13_AST_in = (AST)_t;
    match(_t,STRING_LIT);
    _t = _t.getNextSibling();
    break;
  }
  case 3:
  {
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  }
  _t = __t11;
  _t = _t.getNextSibling();
  _retTree = _t;
}

public final void tDef(AST _t) throws RecognitionException {

  AST tDef_AST_in = (_t == ASTNULL) ? null : (AST)_t;
  AST fromState = null;
  AST toState = null;
    StringBuffer a, b = null;
      funcCall = false;
      funcCond = false;
      SymbolTableEntry ste;
      SymbolTable st;
```

```
AST __t19 = _t;
AST tmp14_AST_in = (AST)_t;
match(_t,T_DEF);
_t = _t.getFirstChild();
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case ID:
{
  fromState = (AST)_t;
  match(_t,ID);
  _t = _t.getNextSibling();
    ste = tempSymTab.getEntry(fromState.getText());
        st = tempSymTab;

  break;
}
case MULT:
{
  AST tmp15_AST_in = (AST)_t;
  match(_t,MULT);
  _t = _t.getNextSibling();
    ste = tempSymTab.getParentEntry();
        st = tempSymTab.getParent();

  break;
}
default:
{
  throw new NoViableAltException(_t);
}
}
}
toState = (AST)_t;
match(_t,ID);
_t = _t.getNextSibling();
a=expression(_t);
```

```
    _t = _retTree;
      if(funcCall)
            funcCond = true;

    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case F_CALL:
    {
      b=functionCall(_t);
      _t = _retTree;
      break;
    }
    case 3:
    {
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    }
    _t = __t19;
    _t = _t.getNextSibling();
      // Add transition to transition list of fromState or parent stat
e of this FSM
          Transition t;
          if(b==null)
            t = new Transition(((StateEntry)ste).getName(), toState.ge
tText(), a.toString());
          else
            t = new Transition(((StateEntry)ste).getName(), toState.ge
tText(), a.toString(), b.toString());
          if(funcCond)
            t.setFunctionCondition();
          ((StateEntry)ste).addTransition(t);
          funcCall = false;
```

```
     funcCond = false;

  _retTree = _t;
 }

 public final StringBuffer  expression(AST _t) throws RecognitionExce
ption {
  StringBuffer r;

  AST expression_AST_in = (_t == ASTNULL) ? null : (AST)_t;
  AST i = null;

    r = new StringBuffer();
    StringBuffer a;


  AST __t76 = _t;
  AST tmp16_AST_in = (AST)_t;
  match(_t,EXPR);
  _t = _t.getFirstChild();
  {
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case LITERAL_tick:
  {
   {
   AST tmp17_AST_in = (AST)_t;
   match(_t,LITERAL_tick);
   _t = _t.getNextSibling();
   i = (AST)_t;
   match(_t,INT_LIT);
   _t = _t.getNextSibling();
   }
    r.append("((tick % "+i.getText()+") == 0)");
   break;
  }
  case AT_MAX:
  case AT_MIN:
```

```
case F_CALL:
case UPLUS:
case UMINUS:
case ID:
case STRING_LIT:
case MULT:
case DIV:
case PLUS:
case MINUS:
case INT_LIT:
case REAL_LIT:
case LITERAL_true:
case LITERAL_false:
case LITERAL_null:
case ASSIGN:
case PLUSEQ:
case MINUSEQ:
case MULTEQ:
case DIVEQ:
case MODEQ:
case OR:
case AND:
case NEQ:
case EQ:
case LT:
case GT:
case LEQ:
case GEQ:
case MOD:
case INCR:
case DECR:
{
  a=assignmentExpression(_t);
  _t = _retTree;
    r.append(a);
  break;
}
default:
  {
   throw new NoViableAltException(_t);
  }
  }
  }
  _t = __t76;
  _t = _t.getNextSibling();
  _retTree = _t;
  return r;
 }

 public final StringBuffer  functionCall(AST _t) throws RecognitionEx
ception {
  StringBuffer r;

  AST functionCall_AST_in = (_t == ASTNULL) ? null : (AST)_t;
  AST fun = null;
  AST id = null;

   r = new StringBuffer();
   boolean firstArg = true;
   StringBuffer a, b;


  AST __t30 = _t;
  AST tmp18_AST_in = (AST)_t;
  match(_t,F_CALL);
  _t = _t.getFirstChild();
  fun = (AST)_t;
  match(_t,ID);
  _t = _t.getNextSibling();
   r.append(fun.getText());
     r.append('(');

  {
  _loop32:
  do {
   if (_t==null) _t=ASTNULL;
   switch ( _t.getType()) {
   case ID:
   {
     id = (AST)_t;
     match(_t,ID);
     _t = _t.getNextSibling();
     break;
   }
   case UPLUS:
   case UMINUS:
   case STRING_LIT:
   case INT_LIT:
   case REAL_LIT:
   case LITERAL_true:
   case LITERAL_false:
   case LITERAL_null:
   {
     a=constant(_t);
     _t = _retTree;

             if(firstArg)  {
               r.append(id.getText());
               firstArg = false;
             }
             else  {
               r.append(", ");
               r.append(id.getText());
             }

     break;
   }
   default:
   {
     break _loop32;
   }
   }
  } while (true);
  }
```

```
      r.append(’)’);
          funcCall = true;

    _t = __t30;
    _t = _t.getNextSibling();
    _retTree = _t;
    return r;
  }

  public final void sDef(AST _t) throws RecognitionException {

    AST sDef_AST_in = (_t == ASTNULL) ? null : (AST)_t;
    AST stateName = null;

    AST __t27 = _t;
    AST tmp19_AST_in = (AST)_t;
    match(_t,S_DEF);
    _t = _t.getFirstChild();
    stateName = (AST)_t;
    match(_t,ID);
    _t = _t.getNextSibling();

        SymbolTableEntry ste = tempSymTab.getEntry(stateName.getText()
);
        tempSymTab = ((StateEntry)ste).getChildSymTab();

    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case LITERAL_label:
    {
      AST tmp20_AST_in = (AST)_t;
      match(_t,LITERAL_label);
      _t = _t.getNextSibling();
      AST tmp21_AST_in = (AST)_t;
      match(_t,STRING_LIT);
      _t = _t.getNextSibling();
      break;
    }
    case PROGRAM:
    {
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    }
    programBody(_t);
    _t = _retTree;
    _t = __t27;
    _t = _t.getNextSibling();
      if (tempSymTab.parent!=null)
          tempSymTab = tempSymTab.parent;

    _retTree = _t;
  }

  public final StringBuffer  constant(AST _t) throws RecognitionExcept
ion {
    StringBuffer r;

    AST constant_AST_in = (_t == ASTNULL) ? null : (AST)_t;
      StringBuffer a;
        r = new StringBuffer();


    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case UPLUS:
    {
      AST __t108 = _t;
      AST tmp22_AST_in = (AST)_t;
      match(_t,UPLUS);
      _t = _t.getFirstChild();
      a=cons(_t);
      _t = _retTree;
      _t = __t108;
      _t = _t.getNextSibling();
        r.append('+');  r.append(a);
      break;
    }
    case UMINUS:
    {
      AST __t109 = _t;
      AST tmp23_AST_in = (AST)_t;
      match(_t,UMINUS);
      _t = _t.getFirstChild();
      a=cons(_t);
      _t = _retTree;
      _t = __t109;
      _t = _t.getNextSibling();
        r.append('-');  r.append(a);
      break;
    }
    case STRING_LIT:
    case INT_LIT:
    case REAL_LIT:
    case LITERAL_true:
    case LITERAL_false:
    case LITERAL_null:
    {
      a=cons(_t);
      _t = _retTree;
        r.append(a);
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    _retTree = _t;
```

```
  return r;
}

public final void func(AST _t) throws RecognitionException {

  AST func_AST_in = (_t == ASTNULL) ? null : (AST)_t;
  AST i = null;
    StringBuffer r = new StringBuffer();
        StringBuffer a, b, c;


  AST __t38 = _t;
  AST tmp24_AST_in = (AST)_t;
  match(_t,FUNC);
  _t = _t.getFirstChild();
    boolean isEvent=false;
      r.append("\tpublic ");
  {
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case LITERAL_trigger:
  {
    AST tmp25_AST_in = (AST)_t;
    match(_t,LITERAL_trigger);
    _t = _t.getNextSibling();
    break;
  }
  case LITERAL_event:
  {
    AST tmp26_AST_in = (AST)_t;
    match(_t,LITERAL_event);
    _t = _t.getNextSibling();
      isEvent = true;
    break;
  }
  case LITERAL_function:
  {
    AST tmp27_AST_in = (AST)_t;
    match(_t,LITERAL_function);
    _t = _t.getNextSibling();
    a=retType(_t);
    _t = _retTree;
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  }
  {
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case LITERAL_label:
  {
    AST tmp28_AST_in = (AST)_t;
    match(_t,LITERAL_label);
    _t = _t.getNextSibling();
    AST tmp29_AST_in = (AST)_t;
    match(_t,STRING_LIT);
    _t = _t.getNextSibling();
    break;
  }
  case ID:
  {
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  }
  i = (AST)_t;
  match(_t,ID);
  _t = _t.getNextSibling();
```

```
    FunctionEntry fe = (FunctionEntry)tempSymTab.getEntry(i.getTex
t());
    tempSymTab = fe.getlocalSymTab();
    r.append(fe.getRetType().getJavaType());
    r.append(’ ’);
    if(isEvent)  {
     r.append(CodeGenerator.PREFIX_EVENT);
     isEvent = false;
    }
    r.append(i.getText());
    r.append(’(’);
    if(i.getText().equals("onClockTick"))
     r.append("int tick");

  b=argsDef(_t);
  _t = _retTree;
   r.append(b);  r.append(’)’);
  c=funcBody(_t);
  _t = _retTree;
   r.append("\t{\n");  r.append(c);
     if(i.getText().equals("onClockTick"))
      r.append("\t\tsuper.onClockTick(tick);\n");
     r.append("\t}\n\n");
  _t = __t38;
  _t = _t.getNextSibling();
   fe.setDefinition(r.toString());
     if (tempSymTab.parent!=null)
      tempSymTab = tempSymTab.parent;

  _retTree = _t;
 }

 public final StringBuffer  retType(AST _t) throws RecognitionExcepti
on {
  StringBuffer r;

  AST retType_AST_in = (_t == ASTNULL) ? null : (AST)_t;
```

```
    r = new StringBuffer();
    StringBuffer a;


if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case LITERAL_void:
{
  {
  AST tmp30_AST_in = (AST)_t;
  match(_t,LITERAL_void);
  _t = _t.getNextSibling();
    r.append("void");
  }
  break;
}
case LITERAL_int:
case LITERAL_float:
case LITERAL_string:
case LITERAL_boolean:
case LITERAL_range:
{
  {
  a=type(_t);
  _t = _retTree;
    r.append(a);
  }
    r.append(' ');
  break;
}
default:
{
  throw new NoViableAltException(_t);
}
}
_retTree = _t;
return r;
```

```
 }

 public final StringBuffer  argsDef(AST _t) throws RecognitionExcepti
on {
  StringBuffer r;

  AST argsDef_AST_in = (_t == ASTNULL) ? null : (AST)_t;
  AST i = null;

    r = new StringBuffer();
    StringBuffer a;
    boolean firstArg = true;


  AST __t46 = _t;
  AST tmp31_AST_in = (AST)_t;
  match(_t,ARGS_DEF);
  _t = _t.getFirstChild();
  {
  _loop48:
  do {
   if (_t==null) _t=ASTNULL;
   if (((_t.getType() >= LITERAL_int && _t.getType() <= LITERAL_ran
ge))) {
    a=type(_t);
    _t = _retTree;
     if(firstArg)
         firstArg = false;
        else
         r.append(", ");
        r.append(a);
        r.append(’ ’);

    i = (AST)_t;
    match(_t,ID);
    _t = _t.getNextSibling();
     r.append(i.getText());
   }
   else {
    break _loop48;
   }

  } while (true);
  }
  _t = __t46;
  _t = _t.getNextSibling();
  _retTree = _t;
  return r;
 }

 public final StringBuffer  funcBody(AST _t) throws RecognitionExcept
ion {
  StringBuffer r;

  AST funcBody_AST_in = (_t == ASTNULL) ? null : (AST)_t;

    r = new StringBuffer();
    StringBuffer a, b;


  AST __t50 = _t;
  AST tmp32_AST_in = (AST)_t;
  match(_t,FUNCBODY);
  _t = _t.getFirstChild();
  a=decls(_t);
  _t = _retTree;
   r.append(a);
  {
  _loop52:
  do {
   if (_t==null) _t=ASTNULL;
   if ((_tokenSet_0.member(_t.getType()))) {
    b=statement(_t);
    _t = _retTree;
     r.append(b);
   }
   else {
    break _loop52;
   }

  } while (true);
  }
  _t = __t50;
  _t = _t.getNextSibling();
  _retTree = _t;
  return r;
 }

 public final StringBuffer  type(AST _t) throws RecognitionException
{
  StringBuffer r;

  AST type_AST_in = (_t == ASTNULL) ? null : (AST)_t;
   StringBuffer a, b;
    r = new StringBuffer();


  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case LITERAL_int:
  {
   AST tmp33_AST_in = (AST)_t;
   match(_t,LITERAL_int);
   _t = _t.getNextSibling();
    r.append("int");
   break;
  }
  case LITERAL_float:
  {
   AST tmp34_AST_in = (AST)_t;
   match(_t,LITERAL_float);
   _t = _t.getNextSibling();
    r.append("float");
  break;
}
case LITERAL_string:
{
  AST tmp35_AST_in = (AST)_t;
  match(_t,LITERAL_string);
  _t = _t.getNextSibling();
    r.append("String");
  break;
}
case LITERAL_boolean:
{
  AST tmp36_AST_in = (AST)_t;
  match(_t,LITERAL_boolean);
  _t = _t.getNextSibling();
    r.append("boolean");
  break;
}
case LITERAL_range:
{
  AST tmp37_AST_in = (AST)_t;
  match(_t,LITERAL_range);
  _t = _t.getNextSibling();
  a=constant(_t);
  _t = _retTree;
  b=constant(_t);
  _t = _retTree;
    r.append("Range");
        range_lower = a.toString();
        range_upper = b.toString();

  break;
}
default:
{
  throw new NoViableAltException(_t);
}
}
```

```
    _retTree = _t;
    return r;
  }

  public final StringBuffer  statement(AST _t) throws RecognitionExcep
tion {
    StringBuffer r;

    AST statement_AST_in = (_t == ASTNULL) ? null : (AST)_t;
    AST i = null;

        r = new StringBuffer();
        StringBuffer a=null, b=null, c=null, d=null;
        for(int j=0;j<tabCount;j++)
          r.append('\t');


    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case EXPR:
    {
      a=expression(_t);
      _t = _retTree;
      r.append(a);   r.append(";\n");
      break;
    }
    case LITERAL_if:
    {
      AST __t58 = _t;
      AST tmp38_AST_in = (AST)_t;
      match(_t,LITERAL_if);
      _t = _t.getFirstChild();
      a=expression(_t);
      _t = _retTree;
      b=statementBlock(_t);
      _t = _retTree;
      {
      if (_t==null) _t=ASTNULL;
```

```
 switch ( _t.getType()) {
 case STMTBLK:
 {
  c=statementBlock(_t);
  _t = _retTree;
  break;
 }
 case 3:
 {
  break;
 }
 default:
 {
  throw new NoViableAltException(_t);
 }
 }
 }
 _t = __t58;
 _t = _t.getNextSibling();
  r.append("if (");  r.append(a);  r.append(’)’);
    r.append(b);
    if(c!=null)  {
     for(int j=0;j<tabCount;j++)
      r.append(’\t’);
     r.append("else");
     r.append(c);
    }

 break;
}
case LITERAL_for:
{
 AST __t60 = _t;
 AST tmp39_AST_in = (AST)_t;
 match(_t,LITERAL_for);
 _t = _t.getFirstChild();
 a=expression(_t);
 _t = _retTree;
```

```
  b=expression(_t);
  _t = _retTree;
  c=expression(_t);
  _t = _retTree;
  d=statementBlock(_t);
  _t = _retTree;
  _t = __t60;
  _t = _t.getNextSibling();
    r.append("for("); r.append(a);  r.append("; ");
        r.append(b);  r.append("; ");   r.append(c);
        r.append(’)’);  r.append(d);
  break;
}
case LITERAL_while:
{
  AST __t61 = _t;
  AST tmp40_AST_in = (AST)_t;
  match(_t,LITERAL_while);
  _t = _t.getFirstChild();
  a=expression(_t);
  _t = _retTree;
  b=statementBlock(_t);
  _t = _retTree;
  _t = __t61;
  _t = _t.getNextSibling();
    r.append("while(");   r.append(a);
        r.append(’)’);      r.append(b);
  break;
}
case LITERAL_output:
{
  AST __t62 = _t;
  AST tmp41_AST_in = (AST)_t;
  match(_t,LITERAL_output);
  _t = _t.getFirstChild();
    r.append("output(");
  {
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case UPLUS:
  case UMINUS:
  case STRING_LIT:
  case INT_LIT:
  case REAL_LIT:
  case LITERAL_true:
  case LITERAL_false:
  case LITERAL_null:
  {
    a=constant(_t);
    _t = _retTree;
      r.append(a);
    break;
  }
  case ID:
  {
    i = (AST)_t;
    match(_t,ID);
    _t = _t.getNextSibling();
      r.append(i.getText());
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  }
    r.append(");\n");
  _t = __t62;
  _t = _t.getNextSibling();
  break;
}
case LITERAL_return:
{
  AST __t64 = _t;
  AST tmp42_AST_in = (AST)_t;
```

```
    match(_t,LITERAL_return);
    _t = _t.getFirstChild();
      r.append("return ");
    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case EXPR:
    {
      a=expression(_t);
      _t = _retTree;
        r.append(a);
      break;
    }
    case 3:
    {
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    }
      r.append(";\n");
    _t = __t64;
    _t = _t.getNextSibling();
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  _retTree = _t;
  return r;
}

public final StringBuffer  decl(AST _t) throws RecognitionException
```

```
{
    StringBuffer r;

    AST decl_AST_in = (_t == ASTNULL) ? null : (AST)_t;
    AST i = null;

        r = new StringBuffer();
        StringBuffer a, b=null;


    AST __t71 = _t;
    AST tmp43_AST_in = (AST)_t;
    match(_t,DECL);
    _t = _t.getFirstChild();
    a=type(_t);
    _t = _retTree;
    {
    _loop74:
    do {
      if (_t==null) _t=ASTNULL;
      if ((_t.getType()==ID)) {
        i = (AST)_t;
        match(_t,ID);
        _t = _t.getNextSibling();
          r.append("\t\t");
              r.append(a);
              r.append(' ');
              r.append(i.getText());
              if(a.toString().equals("Range"))  {
                r.append(" = new ");  r.append(a);  r.append("(");
                // check for valid range limits
                if(Integer.parseInt(range_lower)>Integer.parseInt(rang
e_upper))
                  throw new SemanticException("Invalid range: "+i.getT
ext()+"["+range_lower+", "+range_upper+"]",
                    filename, i.getLine(), i.getColumn());
                // append min,max of range
                r.append(range_lower);  r.append(", ");
        r.append(range_upper);  r.append(", ");
        VariableEntry ve = tempSymTab.getVarEntry(i.getText())
;
        ve.setRangeLower(range_lower);
        ve.setRangeUpper(range_upper);
       }
       else if(a.toString().equals("String"))  {
        r.append(" = new ");  r.append(a);  r.append("(");

       }

    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case UPLUS:
    case UMINUS:
    case STRING_LIT:
    case INT_LIT:
    case REAL_LIT:
    case LITERAL_true:
    case LITERAL_false:
    case LITERAL_null:
    {
     b=constant(_t);
     _t = _retTree;
     break;
    }
    case 3:
    case ID:
    {
     break;
    }
    default:
    {
     throw new NoViableAltException(_t);
    }
    }
    }
```

```
              VariableEntry ve = tempSymTab.getVarEntry(i.getText());
              if (b==null)  {
                if(a.toString().equals("Range"))  {
                  r.append("0)");
                }
                else if(a.toString().equals("String"))
                  r.append(')');
                else  {
                  r.append(" = ");
                  r.append(ve.getJavaInitialValue());
                }
              }
              else  {
                String initVal = b.toString();
                if(a.toString().equals("Range"))  {
                  if(Integer.parseInt(initVal) > Integer.parseInt(rang
e_upper))
                    initVal = range_upper;
                  else if(Integer.parseInt(initVal) < Integer.parseInt
(range_lower))
                    initVal = range_lower;
                  r.append(initVal);  r.append(')');
                }
                else if(a.toString().equals("String"))  {
                  r.append("\"");  r.append(initVal);  r.append("\")")
;
                }
                else  {
                  r.append(" = ");  r.append(initVal);
                }
                ve.setInitialValue(initVal);
              }
              r.append(";\n");

      }
      else {
        break _loop74;
```

```
   }

  } while (true);
  }
  _t = __t71;
  _t = _t.getNextSibling();
   range_lower = null; range_upper = null;
  _retTree = _t;
  return r;
 }

 public final StringBuffer  statementBlock(AST _t) throws Recognition
Exception {
  StringBuffer r;

  AST statementBlock_AST_in = (_t == ASTNULL) ? null : (AST)_t;

    r = new StringBuffer();
    StringBuffer a;
    r.append("\t{\n");
    tabCount++;


  AST __t67 = _t;
  AST tmp44_AST_in = (AST)_t;
  match(_t,STMTBLK);
  _t = _t.getFirstChild();
  {
  int _cnt69=0;
  _loop69:
  do {
   if (_t==null) _t=ASTNULL;
   if ((_tokenSet_0.member(_t.getType()))) {
    a=statement(_t);
    _t = _retTree;
    r.append(a);
   }
   else {
        if ( _cnt69>=1 ) { break _loop69; } else {throw new NoViableAl
tException(_t);}
      }

      _cnt69++;
    } while (true);
    }
    _t = __t67;
    _t = _t.getNextSibling();
      tabCount--;
        for(int i=0;i<tabCount;i++)
          r.append('\t');
        r.append("}\n");

    _retTree = _t;
    return r;
  }

  public final StringBuffer  assignmentExpression(AST _t) throws Recog
nitionException {
    StringBuffer r;

    AST assignmentExpression_AST_in = (_t == ASTNULL) ? null : (AST)_t
;
    AST varname = null;
    AST varname1 = null;
    AST varname2 = null;
    AST varname3 = null;
    AST varname4 = null;
    AST varname5 = null;

        r = new StringBuffer();
        StringBuffer a;


    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case ASSIGN:
```

```
    {
      AST __t80 = _t;
      AST tmp45_AST_in = (AST)_t;
      match(_t,ASSIGN);
      _t = _t.getFirstChild();
      varname = (AST)_t;
      match(_t,ID);
      _t = _t.getNextSibling();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t80;
      _t = _t.getNextSibling();
        if(((tempSymTab.getVarEntry(varname.getText()))).getIDType()==
DataRetType.RANGE)  {
            r.append(varname.getText());  r.append(".newValue(");  r.a
ppend(a);  r.append(')');
          }
          else  {
            r.append(varname.getText());  r.append(" = ");  r.append(a
);
          }

      break;
    }
    case PLUSEQ:
    {
      AST __t81 = _t;
      AST tmp46_AST_in = (AST)_t;
      match(_t,PLUSEQ);
      _t = _t.getFirstChild();
      varname1 = (AST)_t;
      match(_t,ID);
      _t = _t.getNextSibling();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t81;
      _t = _t.getNextSibling();
        if(((tempSymTab.getVarEntry(varname1.getText()))).getIDType()=
=DataRetType.RANGE)  {
            r.append(varname1.getText());  r.append(".newValue(");  r.
append(varname1.getText()); r.append(".getValue() + ");  r.append(a);
 r.append(’)’);
          }
          else  {
            r.append(varname1.getText());  r.append(" += ");  r.append
(a);
          }

      break;
    }
    case MINUSEQ:
    {
      AST __t82 = _t;
      AST tmp47_AST_in = (AST)_t;
      match(_t,MINUSEQ);
      _t = _t.getFirstChild();
      varname2 = (AST)_t;
      match(_t,ID);
      _t = _t.getNextSibling();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t82;
      _t = _t.getNextSibling();
        if(((tempSymTab.getVarEntry(varname2.getText()))).getIDType()=
=DataRetType.RANGE)  {
            r.append(varname2.getText());  r.append(".newValue(");  r.
append(varname2.getText()); r.append(".getValue() - ");  r.append(a);
 r.append(’)’);
          }
          else  {
            r.append(varname2.getText());  r.append(" -= ");  r.append
(a);
          }

      break;
    }
    case MULTEQ:
    {
      AST __t83 = _t;
      AST tmp48_AST_in = (AST)_t;
      match(_t,MULTEQ);
      _t = _t.getFirstChild();
      varname3 = (AST)_t;
      match(_t,ID);
      _t = _t.getNextSibling();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t83;
      _t = _t.getNextSibling();
        if(((tempSymTab.getVarEntry(varname3.getText()))).getIDType()=
=DataRetType.RANGE)  {
            r.append(varname3.getText());  r.append(".newValue(");  r.
append(varname3.getText()); r.append(".getValue() * ");  r.append(a);
 r.append(’)’);
          }
          else  {
            r.append(varname3.getText());  r.append(" *= ");  r.append
(a);
          }

      break;
    }
    case DIVEQ:
    {
      AST __t84 = _t;
      AST tmp49_AST_in = (AST)_t;
      match(_t,DIVEQ);
      _t = _t.getFirstChild();
      varname4 = (AST)_t;
      match(_t,ID);
      _t = _t.getNextSibling();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t84;
```

```
   _t = _t.getNextSibling();
    if(((tempSymTab.getVarEntry(varname4.getText()))).getIDType()=
=DataRetType.RANGE)  {
      r.append(varname4.getText());  r.append(".newValue(");  r.
append(varname4.getText()); r.append(".getValue() / ");  r.append(a);
 r.append(’)’);
     }
     else  {
      r.append(varname4.getText());  r.append(" /= ");  r.append
(a);
     }

   break;
  }
  case MODEQ:
  {
   AST __t85 = _t;
   AST tmp50_AST_in = (AST)_t;
   match(_t,MODEQ);
   _t = _t.getFirstChild();
   varname5 = (AST)_t;
   match(_t,ID);
   _t = _t.getNextSibling();
   a=lefthandexpr(_t);
   _t = _retTree;
   _t = __t85;
   _t = _t.getNextSibling();
    if(((tempSymTab.getVarEntry(varname5.getText()))).getIDType()=
=DataRetType.RANGE)  {
      r.append(varname5.getText());  r.append(".newValue(");  r.
append(varname5.getText()); r.append(".getValue() % ");  r.append(a);
 r.append(’)’);
     }
     else  {
      r.append(varname5.getText());  r.append(" %= ");  r.append
(a);
     }
   break;
  }
  case AT_MAX:
  case AT_MIN:
  case F_CALL:
  case UPLUS:
  case UMINUS:
  case ID:
  case STRING_LIT:
  case MULT:
  case DIV:
  case PLUS:
  case MINUS:
  case INT_LIT:
  case REAL_LIT:
  case LITERAL_true:
  case LITERAL_false:
  case LITERAL_null:
  case OR:
  case AND:
  case NEQ:
  case EQ:
  case LT:
  case GT:
  case LEQ:
  case GEQ:
  case MOD:
  case INCR:
  case DECR:
  {
   {
   a=lefthandexpr(_t);
   _t = _retTree;
   }

     r.append(a);

   break;
```

```
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    _retTree = _t;
    return r;
  }

  public final StringBuffer  lefthandexpr(AST _t) throws RecognitionEx
ception {
    StringBuffer r;

    AST lefthandexpr_AST_in = (_t == ASTNULL) ? null : (AST)_t;

        r = new StringBuffer();
        StringBuffer a, b;


    {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType()) {
    case OR:
    {
      AST __t89 = _t;
      AST tmp51_AST_in = (AST)_t;
      match(_t,OR);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t89;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" || ");  r.append(b);  r
.append(’)’);
      break;
    }
    case AND:
    {
      AST __t90 = _t;
      AST tmp52_AST_in = (AST)_t;
      match(_t,AND);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t90;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" && ");  r.append(b);  r
.append(’)’);
      break;
    }
    case EQ:
    {
      AST __t91 = _t;
      AST tmp53_AST_in = (AST)_t;
      match(_t,EQ);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t91;
      _t = _t.getNextSibling();
        r.append(’(’);  r.append(a);
          VariableEntry ve = tempSymTab.getVarEntry(a.toString());
          if((ve!=null && ve.getIDType()==DataRetType.STRING)||(a.toSt
ring().startsWith("\"")&&a.toString().endsWith("\"")))  {
            r.append(".equals(");  r.append(b);  r.append(’)’);
          }
          else  {
            r.append(" == ");  r.append(b);
          }
          r.append(')');
      break;
    }
    case NEQ:
    {
      AST __t92 = _t;
      AST tmp54_AST_in = (AST)_t;
      match(_t,NEQ);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t92;
      _t = _t.getNextSibling();
      r.append('(');
          VariableEntry ve = tempSymTab.getVarEntry(a.toString());
          if((ve!=null && ve.getIDType()==DataRetType.STRING)||(a.toSt
ring().startsWith("\"")&&a.toString().endsWith("\"")))  {
            r.append("!");   r.append(a);   r.append(".equals(");
            r.append(b);  r.append(')');
          }
          else  {
            r.append(a);  r.append(" != ");
            r.append(b);
          }
          r.append(')');
      break;
    }
    case LT:
    {
      AST __t93 = _t;
      AST tmp55_AST_in = (AST)_t;
      match(_t,LT);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t93;
      _t = _t.getNextSibling();
      r.append('('); r.append(a); r.append(" < "); r.append(b); r.
append(')');
      break;
    }
    case LEQ:
    {
      AST __t94 = _t;
      AST tmp56_AST_in = (AST)_t;
      match(_t,LEQ);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t94;
      _t = _t.getNextSibling();
      r.append('('); r.append(a); r.append(" <= "); r.append(b); r
.append(')');
      break;
    }
    case GT:
    {
      AST __t95 = _t;
      AST tmp57_AST_in = (AST)_t;
      match(_t,GT);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t95;
      _t = _t.getNextSibling();
      r.append('('); r.append(a); r.append(" > "); r.append(b); r.
append(')');
      break;
    }
    case GEQ:
    {
      AST __t96 = _t;
      AST tmp58_AST_in = (AST)_t;
      match(_t,GEQ);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t96;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" >= ");  r.append(b);  r
.append(’)’);
      break;
    }
    case PLUS:
    {
      AST __t97 = _t;
      AST tmp59_AST_in = (AST)_t;
      match(_t,PLUS);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t97;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" + ");  r.append(b);  r.
append(’)’);
      break;
    }
    case MINUS:
    {
      AST __t98 = _t;
      AST tmp60_AST_in = (AST)_t;
      match(_t,MINUS);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t98;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" - ");  r.append(b);  r.
append(’)’);
      break;
    }
    case MULT:
    {
      AST __t99 = _t;
      AST tmp61_AST_in = (AST)_t;
      match(_t,MULT);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t99;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" * ");  r.append(b);  r.
append(’)’);
      break;
    }
    case DIV:
    {
      AST __t100 = _t;
      AST tmp62_AST_in = (AST)_t;
      match(_t,DIV);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t100;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" / ");  r.append(b);  r.
append(’)’);
      break;
    }
    case MOD:
    {
      AST __t101 = _t;
      AST tmp63_AST_in = (AST)_t;
      match(_t,MOD);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      b=lefthandexpr(_t);
      _t = _retTree;
      _t = __t101;
      _t = _t.getNextSibling();
      r.append(’(’);  r.append(a);  r.append(" % ");  r.append(b);  r.
append(’)’);
      break;
    }
    case INCR:
    {
      AST __t102 = _t;
      AST tmp64_AST_in = (AST)_t;
      match(_t,INCR);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t102;
      _t = _t.getNextSibling();
        if(a.toString().endsWith(".getValue()"))  {
            String var = a.toString().substring(0,(a.toString().length
()-11));
            r.append(var);
            r.append(".incrementValue()");
          }
          else  {
            r.append(a);
            r.append("++ ");
          }

      break;
    }
    case DECR:
    {
      AST __t103 = _t;
      AST tmp65_AST_in = (AST)_t;
      match(_t,DECR);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t103;
      _t = _t.getNextSibling();
        if(a.toString().endsWith(".getValue()"))  {
            String var = a.toString().substring(0,(a.toString().length
()-11));
            r.append(var);
            r.append(".decrementValue()");
          }
          else  {
            r.append(a);
            r.append("-- ");
          }

      break;
    }
    case AT_MAX:
    {
      AST __t104 = _t;
      AST tmp66_AST_in = (AST)_t;
      match(_t,AT_MAX);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t104;
      _t = _t.getNextSibling();
        if(a.toString().endsWith(".getValue()"))  {
            String var = a.toString().substring(0,(a.toString().length
()-11));
            r.append(var);
            r.append(".atMax()");
          }

      break;
    }
    case AT_MIN:
    {
      AST __t105 = _t;
      AST tmp67_AST_in = (AST)_t;
      match(_t,AT_MIN);
      _t = _t.getFirstChild();
      a=lefthandexpr(_t);
      _t = _retTree;
      _t = __t105;
      _t = _t.getNextSibling();
        if(a.toString().endsWith(".getValue()"))  {
            String var = a.toString().substring(0,(a.toString().length
()-11));
            r.append(var);
            r.append(".atMin()");
          }

      break;
    }
    case F_CALL:
    case UPLUS:
    case UMINUS:
    case ID:
    case STRING_LIT:
    case INT_LIT:
    case REAL_LIT:
    case LITERAL_true:
    case LITERAL_false:
```

```
  case LITERAL_null:
  {
    a=val(_t);
    _t = _retTree;
    r.append(a);
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  }
  _retTree = _t;
  return r;
}

public final StringBuffer  val(AST _t) throws RecognitionException {

  StringBuffer r;

  AST val_AST_in = (_t == ASTNULL) ? null : (AST)_t;
  AST i = null;

      StringBuffer a;
      r = new StringBuffer();


  if (_t==null) _t=ASTNULL;
  switch ( _t.getType()) {
  case UPLUS:
  case UMINUS:
  case STRING_LIT:
  case INT_LIT:
  case REAL_LIT:
  case LITERAL_true:
  case LITERAL_false:
  case LITERAL_null:
    {
      a=constant(_t);
      _t = _retTree;
      r.append(a);
      break;
    }
    case ID:
    {
      i = (AST)_t;
      match(_t,ID);
      _t = _t.getNextSibling();
        r.append(i.getText());
        if(((tempSymTab.getVarEntry(i.getText()))).getIDType()==DataRe
tType.RANGE)
          r.append(".getValue()");

      break;
    }
    case F_CALL:
    {
      a=functionCall(_t);
      _t = _retTree;
      r.append(a);
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
    }
    _retTree = _t;
    return r;
  }

  public final StringBuffer  cons(AST _t) throws RecognitionException
{
    StringBuffer r;
```

```
AST cons_AST_in = (_t == ASTNULL) ? null : (AST)_t;
AST a = null;
AST b = null;
AST c = null;

    r = new StringBuffer();


if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case STRING_LIT:
{
  a = (AST)_t;
  match(_t,STRING_LIT);
  _t = _t.getNextSibling();
  r.append("\""); r.append(a.getText()); r.append("\"");
  break;
}
case INT_LIT:
{
  b = (AST)_t;
  match(_t,INT_LIT);
  _t = _t.getNextSibling();
  r.append(b.getText());
  break;
}
case REAL_LIT:
{
  c = (AST)_t;
  match(_t,REAL_LIT);
  _t = _t.getNextSibling();
  r.append(c.getText());
  break;
}
case LITERAL_true:
{
  AST tmp68_AST_in = (AST)_t;
  match(_t,LITERAL_true);
```

```
    _t = _t.getNextSibling();
    r.append("true");
    break;
  }
  case LITERAL_false:
  {
    AST tmp69_AST_in = (AST)_t;
    match(_t,LITERAL_false);
    _t = _t.getNextSibling();
    r.append("false");
    break;
  }
  case LITERAL_null:
  {
    AST tmp70_AST_in = (AST)_t;
    match(_t,LITERAL_null);
    _t = _t.getNextSibling();
    r.append("null");
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  _retTree = _t;
  return r;
}


public static final String[] _tokenNames = {
  "<0>",
  "EOF",
  "<2>",
  "NULL_TREE_LOOKAHEAD",
  "AT_MAX",
  "AT_MIN",
  "PROGRAM",
  "V_DECL",
  "FUNCDEF",
  "FUNC",
  "FUNCNAME",
  "FUNCBODY",
  "F_CALL",
  "TYPE",
  "STMT",
  "STMTBLK",
  "DECLS",
  "DECL",
  "STATE_DECLS",
  "S_DECL",
  "TRANSITIONS",
  "T_DEF",
  "STATE_DEFS",
  "S_DEF",
  "EXPR",
  "ARGS_DEF",
  "ARGS_CALL",
  "UPLUS",
  "UMINUS",
  "\"entity\"",
  "ID",
  "\"label\"",
  "STRING_LIT",
  "LCURLY",
  "RCURLY",
  "\"state\"",
  "COMMA",
  "SCOLON",
  "\"init\"",
  "\"transition\"",
  "MULT",
  "\"to\"",
  "\"if\"",
  "LPAREN",
  "RPAREN",
  "DIV",
  "PLUS",
  "MINUS",
  "INT_LIT",
  "REAL_LIT",
  "\"true\"",
  "\"false\"",
  "\"null\"",
  "\"trigger\"",
  "\"event\"",
  "\"function\"",
  "\"int\"",
  "\"float\"",
  "\"string\"",
  "\"boolean\"",
  "\"range\"",
  "LSQUARE",
  "COLON",
  "RSQUARE",
  "\"void\"",
  "\"else\"",
  "\"for\"",
  "\"while\"",
  "\"output\"",
  "\"return\"",
  "ASSIGN",
  "\"tick\"",
  "PLUSEQ",
  "MINUSEQ",
  "MULTEQ",
  "DIVEQ",
  "MODEQ",
  "OR",
  "AND",
  "NEQ",
  "EQ",
  "LT",
  "GT",
    "LEQ",
    "GEQ",
    "MOD",
    "INCR",
    "DECR",
    "NEWLINE",
    "WSPACE",
    "ML_COMMENT",
    "SL_COMMENT",
    "DOT",
    "NOT",
    "MIN_MAX",
    "DIGIT",
    "CHAR_ATOM"
  };

  private static final long[] mk_tokenSet_0() {
    long[] data = { 4398063288320L, 60L, 0L, 0L};
    return data;
  }
  public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0())
;
  }
```

A.11 Java Source: CodeGenerator

```
package edu.columbia.cs.coms4115.empath.codegen;

import java.io.File;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.List;

import org.antlr.stringtemplate.StringTemplate;
import org.antlr.stringtemplate.StringTemplateGroup;

import edu.columbia.cs.coms4115.empath.antlr.FunctionEntry;
import edu.columbia.cs.coms4115.empath.antlr.StateEntry;
import edu.columbia.cs.coms4115.empath.antlr.SymbolTable;
import edu.columbia.cs.coms4115.empath.antlr.Transition;
import edu.columbia.cs.coms4115.empath.antlr.VariableEntry;

/**
 * Main code generation class
 * @author William Mee (wjm2107)
 */
public class CodeGenerator {

  public static final String TPLT_ENTITY_SUBCLASS="EntitySubclass";
  public static final String TPLT_STATE_SUBCLASS="StateSubclass";
  public static final String TRGT_PKG="edu.columbia.cs.coms4115.empath
.gen";
  public static final File TRGT_DIR=new File("trgt/edu/columbia/cs/com
s4115/empath/gen");
  public static final String STATE_SUPERCLASS="EmpathState";
  public static final String EXT_JAVA=".java";

  public static final String PREFIX_TRIGGER="_trigger_";
  public static final String PREFIX_EVENT="_event_";
  public static final String PREFIX_ENTITY="Entity";
  // template attributes
  public static final String ATTR_CLASS="class";
  public static final String ATTR_ROOT_CLASS="root_class";
  public static final String ATTR_PACKAGE="package";
  public static final String ATTR_SUPERCLASS="superclass";
  public static final String ATTR_FUNCTION="function";
  public static final String ATTR_VARIABLE="variable";
  public static final String ATTR_TRANS_FROM="trans_from";
  public static final String ATTR_TRANS_TO="trans_to";
  public static final String ATTR_TRANS_COND="trans_cond";
  public static final String ATTR_TRANS_OUTPUT="trans_out";
  public static final String ATTR_STATE_INIT="init_state";
  public static final String ATTR_STATE_INIT_FOR="init_state_for";
  public static final String ATTR_NO_ONENTRY="no_onentry";
  public static final String ATTR_NO_ONEXIT="no_onexit";
  public static final String ATTR_FUNC_LABEL="function_label";
  public static final String ATTR_LABELLED_FUNC="labelled_function";
  public static final String ATTR_STATE_LABEL="state_label";

  // special function names
  public static final String SPECIAL_FUNCTION_ONENTRY="onEntry";
  public static final String SPECIAL_FUNCTION_ONEXIT="onExit";
  public static final String SPECIAL_FUNCTION_ONCLOCKTICK="onClockTick
";

  SymbolTable _st;
  StringTemplateGroup _codegenGrp;
  List <StringTemplate> _stateTemplates=new ArrayList<StringTemplate>(
);
  StringTemplate _entitySubTplt;

  private boolean _debug=true;

  private int _funcCounter=0;

  public CodeGenerator(SymbolTable st) {
    _st=st;
    _codegenGrp = new StringTemplateGroup("codegen", "template");
    _entitySubTplt = _codegenGrp.getInstanceOf("EntitySubclass");
  }

  /**
   * Generates a subclass of @see edu.columbia.cs.coms4115.empath.mode
l.EmpathState
   * @param st
   */
  private void generate(SymbolTable st) throws CodeGenException {
    String initState=null;
    String parentName=st.getName();
    StringTemplate template = templateForState(st);
    _stateTemplates.add(template);

    if (st.getParent()!=null) {
      // get the state entry corresponding to this symbol table to loo
k at transitions
      //StateEntry myStateEntry=(StateEntry)st.getParent().getEntry(st
.getName());
      StateEntry myStateEntry=(StateEntry)st.getParentEntry();
      processTransitions(myStateEntry, template);
    }


    for (String state:st.stateIterable()) {
      StateEntry se = (StateEntry)st.getEntry(state);
      if (se==null) {
        throw new CodeGenException("could not find child symbol table
for "+state+" having parent "+parentName);
      }
      // set the initstate either to the first state or one marked exp
licitly as the first state
      if (se.isInitState() || initState==null) {
        initState=se.getName();
      }
      _entitySubTplt.setAttribute("state", state);
      _entitySubTplt.setAttribute("parent", parentName);
      SymbolTable childST = se.getChildSymTab();
      if (childST==null) {
        // this is a trivial state
        StringTemplate stateTemplate = _codegenGrp.getInstanceOf(TPLT_
STATE_SUBCLASS);
        stateTemplate.setAttribute(ATTR_CLASS,state);
        stateTemplate.setAttribute(ATTR_PACKAGE, TRGT_PKG);
        stateTemplate.setAttribute(ATTR_SUPERCLASS, parentName);
        stateTemplate.setAttribute(ATTR_NO_ONENTRY, "true");
        stateTemplate.setAttribute(ATTR_NO_ONEXIT, "true");

        _stateTemplates.add(stateTemplate);
        processTransitions(se, stateTemplate);
      } else {
        generate(childST);
      }
    }
    if (initState!=null) {
      _entitySubTplt.setAttribute(ATTR_STATE_INIT, initState);
      _entitySubTplt.setAttribute(ATTR_STATE_INIT_FOR, parentName);
    }
  }

  /**
   * Go through the transitions of a state, adding entries to template
s as needed
   *
   * @param entry
   * @param template
   */
  private void processTransitions(StateEntry entry, StringTemplate tem
plate) {
    List<Transition>transitions = entry.getTransitionList();
    if (transitions!=null) {
      for (Transition trans: transitions) {
        String funcName = null;
        if (!trans.isFunctionCondition()) {
          funcName = PREFIX_TRIGGER+_funcCounter;
          _funcCounter++;
```

```
          String funcBody = "public boolean "+funcName+"() { return("+
trans.getCondition()+");}";
          template.setAttribute(ATTR_FUNCTION, funcBody);
        } else {
          funcName = normalizeFunCall(trans.getCondition());
        }
        String transOut=trans.getAction()==null?"null":’"’+normalizeFu
nCall(trans.getAction())+’"’;
        _entitySubTplt.setAttribute(ATTR_TRANS_FROM, trans.getFromStat
e());
        _entitySubTplt.setAttribute(ATTR_TRANS_TO, trans.getToState())
;
        _entitySubTplt.setAttribute(ATTR_TRANS_COND, funcName);
        _entitySubTplt.setAttribute(ATTR_TRANS_OUTPUT, transOut);

      }
    }

  }

  /**
   * initialize a template for @see edu.columbia.cs.coms4115.empath.mo
del.EmpathState
   * @param st
   * @return
   */
  private StringTemplate templateForState(SymbolTable st) {
    boolean hasOnExit=false,hasOnEntry=false;
    StringTemplate template = _codegenGrp.getInstanceOf(TPLT_STATE_SUB
CLASS);
    //template.setAttribute(ATTR_CLASS,st.getName());
    template.setAttribute(ATTR_CLASS,st.getName());
    template.setAttribute(ATTR_PACKAGE, TRGT_PKG);
    if (st.getLabel()!=null) {
      template.setAttribute(ATTR_STATE_LABEL, st.getLabel());
    }
    SymbolTable parentTable = st.getParent();
    String superClass=(parentTable==null)?STATE_SUPERCLASS:parentTable
```

```
.getName();
    template.setAttribute(ATTR_SUPERCLASS, superClass);
    // go through the list of variables
    for (String varName: st.variableIterable()) {
      template.setAttribute(ATTR_VARIABLE, getVariable(varName,(Variab
leEntry)st.getEntry(varName)));
    }
    // go through the list of functions
    for (String funcName: st.functionIterable()) {
      FunctionEntry fe = (FunctionEntry) st.getEntry(funcName);
      String funcBody = fe.getDefinition();
      template.setAttribute(ATTR_FUNCTION, funcBody);
      if (SPECIAL_FUNCTION_ONENTRY.equals(funcName)) {
        hasOnEntry=true;
      }
      if (SPECIAL_FUNCTION_ONEXIT.equals(funcName)) {
        hasOnExit=true;
      }
      String label = fe.getLabel();
      if (label!=null) {
        // assume its a label
        template.setAttribute(ATTR_FUNC_LABEL, label);
        template.setAttribute(ATTR_LABELLED_FUNC, PREFIX_EVENT+funcNam
e);
      }
    }
    if (!hasOnEntry) {
      template.setAttribute(ATTR_NO_ONENTRY, "true");
    }
    if (!hasOnExit) {
      template.setAttribute(ATTR_NO_ONEXIT, "true");
    }
    return template;
  }

  /**
   * Gets the variable line
   * @param name the name of the variable
   * @param ventry the variable entry
   * @return
   */
  private String getVariable(String name, VariableEntry ventry) {
    StringBuffer buffer = new StringBuffer();
    buffer.append(ventry.getIDType().getJavaType()+" "+name);
    buffer.append("="+ventry.getJavaInitialValue());
    return buffer.toString();
  }

  /**
   * Generate all needed source files
   * @throws CodeGenException if there are problems during code genera
tion
   */
  public void generate() throws CodeGenException {
    String entityName=_st.getName();
    _entitySubTplt.setAttribute(ATTR_CLASS,"Entity"+entityName);
    _entitySubTplt.setAttribute(ATTR_PACKAGE,TRGT_PKG);
    _entitySubTplt.setAttribute(ATTR_ROOT_CLASS,entityName);
    generate(_st);
    try {
      createFiles();
    } catch (IOException e) {
      throw new CodeGenException(e);
    }
  }

  /**
   * Create the set of generated Java files
   * @throws IOException
   */
  private void createFiles() throws IOException {

    if (!TRGT_DIR.exists()) {
      boolean created=TRGT_DIR.mkdir();
      if (!created) {
        throw new IOException("could not create target code directory
"+TRGT_DIR);
      }
    }

    String className=(String)_entitySubTplt.getAttribute(ATTR_CLASS);
    File file=new File(TRGT_DIR,className+EXT_JAVA);
    PrintStream pstream = new PrintStream(file);
    pstream.println(_entitySubTplt);
    pstream.close();
    for(StringTemplate stateTplt:_stateTemplates) {
      className=(String)stateTplt.getAttribute(ATTR_CLASS);
      file = new File(TRGT_DIR,className+EXT_JAVA);
      if (_debug) {
        System.out.println("generating "+file);
      }
      pstream = new PrintStream(file);
      pstream.println(stateTplt);
      pstream.close();
    }
  }

  /**
    * Removes the braces from a function call, i.e. "foo()" becomes "fo
o"
    * @param call with optional braces
    * @return call without braces
    */
  private static String normalizeFunCall(String call) {
    if (call==null) {
      return null;
    }
    int braceIdx = call.indexOf('(');
    if (braceIdx!=-1) {
      return call.substring(0,braceIdx);
    }
    return call;
  }
```

```
}
```

A.12 Java Source: DataRetType

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.util.HashMap;
import java.util.Map;

/**
 * Data types
 * @author William Mee (wjm2107)
 *
 */
public enum DataRetType {
  STRING("String"),
  BOOLEAN("boolean"),
  INT("int"),
  FLOAT("float"),
  VOID("void") ,
  RANGE("Range");

  private String _javaType;

  private static Map<String, DataRetType> _map=new HashMap<String, Dat
aRetType>();
  static {
    _map.put("string", STRING);
    _map.put("boolean", BOOLEAN);
    _map.put("int", INT);
    _map.put("float", FLOAT);
    _map.put("void", VOID);
    _map.put("range", RANGE);
  }

  DataRetType(String type) {
    _javaType=type;
  }

  public String getJavaType() {
        return _javaType;
    }

  public static DataRetType byString(String string) {
    return _map.get(string);
  }

}
```

A.13 Java Source: EmpathCompiler

```
package edu.columbia.cs.coms4115.empath;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

import antlr.RecognitionException;
import antlr.SemanticException;
import antlr.TokenStreamException;
import antlr.collections.AST;
import edu.columbia.cs.coms4115.empath.antlr.BaseAST;
import edu.columbia.cs.coms4115.empath.antlr.CodeGenWalker;
import edu.columbia.cs.coms4115.empath.antlr.EmpathLexer;
import edu.columbia.cs.coms4115.empath.antlr.EmpathParser;
import edu.columbia.cs.coms4115.empath.antlr.EmpathTreeWalker;
import edu.columbia.cs.coms4115.empath.antlr.TestConstants;
import edu.columbia.cs.coms4115.empath.codegen.CodeGenException;
import edu.columbia.cs.coms4115.empath.codegen.CodeGenerator;

public class EmpathCompiler {


  private static void printError(String msg) {
    System.err.println(msg);
  }

  private static void bail(String msg) {
    printError(msg);
    System.exit(-1);
  }

  private void run(File ipfile) {

    EmpathParser parser = null;
    EmpathTreeWalker walker = new EmpathTreeWalker();
    walker.filename=ipfile.getPath();
    CodeGenWalker gen = new CodeGenWalker();
    gen.filename=ipfile.getPath();
    EmpathLexer lexer = null;
    try {
       lexer = new EmpathLexer(new FileInputStream(ipfile));
    } catch (FileNotFoundException e) {
      bail("file not found: "+ipfile);
    }
    parser = new EmpathParser(lexer);
    parser.setASTNodeClass(BaseAST.class.getName());
    try {
      parser.entitySpec();
    } catch (RecognitionException e) {
      bail(e.getMessage()+" at line "+e.getLine());
    } catch (TokenStreamException e) {
      bail(e.getMessage());
    }
    AST ast = parser.getAST();
    try {
      walker.entitySpec(ast);
      gen.setSymTable(walker.getSymTable());
      gen.entitySpec(ast);
    } catch (SemanticException e)  {
      bail(e.getMessage()+" at line "+e.getLine());
    } catch (RecognitionException e) {
      bail(e.getMessage()+" at line "+e.getLine());
    }
    //CodeGenWalker walker = walkFile(ipfile);
    CodeGenerator generator = new CodeGenerator(walker.getSymTable());

    try {
      generator.generate();
    } catch (CodeGenException e) {
      bail("code generation error: "+e.getMessage());
    }

  }
```

```
  /**
   * @param args
   */
  public static void main(String[] args) {
    if (args.length!=1) {
      bail("File name not specified");
    }
    String filename = args[0];
    File file = new File(filename);
    if (!file.isFile()) {
      bail("Specified command line argument is not a file");
    }
    EmpathCompiler compiler = new EmpathCompiler();
    compiler.run(file);
  }
}
```

A.14 Java Source: EmpathEntity

```
package edu.columbia.cs.coms4115.empath.model;

// import java.util.List;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import edu.columbia.cs.coms4115.empath.runtime.*;

/** EmpathEntity serves as the container class for an entity.
 *
 * @author jposner
 *
 */
public abstract class EmpathEntity {

  public final static String GENERATED_PACKAGE="edu.columbia.cs.coms41
15.empath.eg.";

  protected Map<String, EmpathStateTree> _allStates; // All possible s
tates.
  protected EmpathStateTree _currentStateTree; // The node in the stat
e tree representing the current node.
  protected EmpathState _currentState; // The EmpathState class for th
e current state.
  protected EmpathStateTree _root; // The root of the state tree.
  protected EmpathRuntime _runtimeEnv; // The runtime environment.
  protected int _clockTick; // The current clock tick value.


  /**
   * Constructor
   * @param runtime
   */
  public EmpathEntity(EmpathRuntime runtime) {
    // super();
    // addObserver(runtime);
    _runtimeEnv=runtime;
    _allStates=new HashMap<String,EmpathStateTree>();
  }

  /**
   * initialize the entity with the current state of the state tree
   * @param root the root of the state tree
   * @param currState the current state the entity is in
   */
  protected void initialize(EmpathStateTree root,EmpathState currState
) {
    _root = root;
    _currentStateTree = root;
    _currentState = currState;
  }

  /** onClockTick excecutes the entity's onClockTick code and incremen
ts clockTick.
   *
   *
   */
  public void onClockTick()
  {
    _currentState.onClockTick(_clockTick++);
  }

  /** isTerminal determines whether or not the current state has no po
ssible exit.
   *
   * @return isTerminal, is this a terminal state?
   */
  public boolean isTerminal()
  {
    return _currentStateTree.isTerminal();
  }
  /** getState returns the name of the current state.
   *
   * @return current state name.
   */
  public String getState()
  {
    if (_currentState.getLabel() != null)
    {
      return _currentState.getLabel();
    }

    return _currentState.getClass().getName();
  }

  public ArrayList<Method> getEvents()
  {
    return _currentStateTree.getEvents();
  }

  public String getEventLabel(Method event) {
    return _currentState.getEventLabel(event);
  }

  public void callEvent(Method event, Object[] args)
  {
    try
    {
      event.invoke(_currentState, args);
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
  }

  /** evaluateAndExecuteTransitions does the actual work of checking w
hether or not a transition should
```

```
   *  happen and changes from one state to another.
   * @return state name for the former state.
   */
  public String evaluateAndExecuteTransitions() {
    EmpathStateTree newState;
    Class stateToCall;
    EmpathStateTree lastState = null;

    String currState = _currentStateTree.getName();
    System.out.println("Current state is "+currState);
    debug("Evaluating transitions at time "+_clockTick);
    newState = _currentStateTree.evaluateTransitions(); // Target of a
ny transition. null if no transition.

    if (newState != null) {
      debug("making transition from "+currState+" to "+newState.getNam
e());
      _currentState.onExit(); // execute onExit if leaving the state.
      // sendOutput("Transitioning to state " + newState);

      lastState = newState;

      Constructor entityConstructor;

      try
      {
        stateToCall = newState.getState();

        entityConstructor = stateToCall.getConstructor((Class[])null);


        // Instantiate the new state as currentState.
        _currentState = (EmpathState) createObject(entityConstructor,
null);

        // Change the currentStateTree pointer to refer to the correct
 node.
        _currentStateTree = newState;
        // Run onEntry for the new state.
        _currentState.onEntry();
      }
      catch (NoSuchMethodException e)
      {
        System.out.println(e);
      }

      followInit();
    } else {
      debug("no transition made");
    }

    if (lastState != null) {
      _runtimeEnv.stateChanged();
      return lastState.getState().getName();
    }
    else
    {
      return null;
    }
  }

  public String followInit()
  {
    EmpathStateTree newState;
    Class stateToCall;
    EmpathStateTree lastState = null;

    newState = _currentStateTree.getInit();

    while (newState != null)
    {
      lastState = newState;

      Constructor entityConstructor;
```

```
      try
      {
        stateToCall = newState.getState();

        entityConstructor = stateToCall.getConstructor((Class[])null);


        _currentState = (EmpathState) createObject(entityConstructor,
null);

        _currentStateTree = newState;

        _currentState.onEntry();

        newState = _currentStateTree.getInit();
      }
      catch (NoSuchMethodException e)
      {
        System.out.println(e);
      }
    }

    if (lastState != null) {
      debug("initial state is "+lastState.getName());
      return lastState.getState().getName();
    }
    else
    {
      return null;
    }
  }

  /** createObject calls the constructor and creates an object. Used a
s support for the
   *  reflection above.
   *
   * @param A constructor method.
   * @param An array of arguments for the constructor method.
```

```
 * @return The object to be created.
 */
public static Object createObject(Constructor constructor,
          Object[] arguments)
{

  // System.out.println ("Constructor: " + constructor.toString());
  Object object = null;

  try {
    object = constructor.newInstance(arguments);
    // System.out.println ("Object: " + object.toString());
    return object;
  } catch (InstantiationException e) {
    System.out.println(e);
  } catch (IllegalAccessException e) {
    System.out.println(e);
  } catch (IllegalArgumentException e) {
    System.out.println(e);
  } catch (InvocationTargetException e) {
    System.out.println(e);
  }
  return object;
}

/**
 * register a tree for easy lookup later
 * @param name the name, corresponding to the name of the node
 * @param tree
 */
public void registerTree(EmpathStateTree tree) {
  _allStates.put(tree.getName(),tree);
}

/**
 * lookup a tree in the registry, based on name
 * @param name the name of the tree node
 * @return the tree, or null
```

```
   */
  public EmpathStateTree lookupTree(String name) {
    return _allStates.get(name);
  }

  /**
   * Sends a string to the runtime
   * @param msg
   */
  public void sendOutput(String msg) {
    _runtimeEnv.sendOutput(msg);
  }


  /**
   * Convenience method for doing debuging
   * @param msg the debug message
   */
  public void debug(String msg) {
    _runtimeEnv.debug(msg);
  }

}
```

A.15 Java Source: EmpathRuntime

```
package edu.columbia.cs.coms4115.empath.runtime;

import edu.columbia.cs.coms4115.empath.model.*;

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import edu.columbia.cs.coms4115.empath.ui.*;
import edu.columbia.cs.coms4115.empath.codegen.CodeGenerator;

/**
 * Empath runtime environment
 *
 * @author William Mee (wjm2107)
 */
public class EmpathRuntime{

  private EmpathEntity _entity;
  private EmpathUI _ui;
  private boolean _debug=false;

  public EmpathRuntime() {
    _ui = new TextUI(this);
  }

  public String getLabel() {
    return _entity.getState();
  }

  public ArrayList<String> getEvents() {
    ArrayList<Method> eventMethods = _entity.getEvents();
    ArrayList<String> eventLabels = new ArrayList<String>();

    int i = 0;
    while (i < eventMethods.size()) {
      eventLabels.add(i, _entity.getEventLabel(eventMethods.get(i)));

      i++;
    }

    return eventLabels;
  }

  public Class[] getEventArgs(int i) {
    Class[] paramTypes;
    paramTypes = _entity.getEvents().get(i).getParameterTypes();

    if (paramTypes.length == 0)
    {
      return (Class []) null;
    }

    return paramTypes;
  }

  public void stateChanged() {
    _ui.refreshCache();
    _ui.refreshDisplay();
  }

  public void callEvent(int event, Object[] params) {
    _entity.callEvent(_entity.getEvents().get(event), params);
  }

  public void callClockTick() {
    _entity.onClockTick();
    _entity.evaluateAndExecuteTransitions();
  }

  public void runEntity(EmpathEntity entity)
  {
    _entity=entity;
    while (_entity.followInit() != null);

    _ui.refreshCache();
    _ui.refreshDisplay();

    while (!_entity.isTerminal())
    {
      _ui.awaitInput();
    }

    _ui.terminate();
  }

  public void quit() {
    _ui.terminate();

    System.exit(0);
  }

  /**
   * Write a message to stderr and exit with an error code of -1
   * @param msg the message to show
   */
  public static void bale(String msg) {
    System.err.println(msg);
    System.exit(-1);
  }

  public static void syntaxAndBale() {
    bale("syntax: EmpathRuntime [-d] <entity-name>");
  }


  /**
   * @param args
   */
  public static void main(String[] args) {
    boolean debug=false;
    if (args.length ==0 || args.length > 2) {
      syntaxAndBale();
    }
    String entityClassName = null;
    if (args.length==1) {
       entityClassName = CodeGenerator.TRGT_PKG+"."+CodeGenerator.PREF
IX_ENTITY+args[0];
    } else {
      if (! "-d".equals(args[0])) {
        syntaxAndBale();
      }
       entityClassName = CodeGenerator.TRGT_PKG+"."+CodeGenerator.PREF
IX_ENTITY+args[1];
       debug=true;
    }

    try {
      Class entityState = Class.forName(entityClassName);
      Constructor entityConstructor;
      Class[] constructorParam = new Class[1];
      constructorParam[0] = EmpathRuntime.class;

      entityConstructor = entityState.getConstructor(constructorParam)
;

      EmpathRuntime runtime = new EmpathRuntime();
      runtime.setDebug(debug);

      Object[] containedERE = new Object[1];
      containedERE[0] = runtime;

      EmpathEntity myEntity = (EmpathEntity) createObject(entityConstr
uctor, containedERE);

      runtime.runEntity(myEntity);
    }
    catch (ClassNotFoundException e) {
      bale("No class "+entityClassName+" found for the entity " + args
[0]);
    }
    catch (NoClassDefFoundError e) {
      bale("No class "+entityClassName+" found for the entity " + args
[0]);
    }
    catch (NoSuchMethodException e) {
      bale("Unexpected exception: "+e.getMessage());
    }
  }

  public void sendOutput(String msg)
  {
    _ui.displayOutput(msg);
  }

  /** createObject calls the constructor and creates an object. Used a
s support for the
   *  reflection above.
   *
   * @param A constructor method.
   * @param An array of arguments for the constructor method.
   * @return The object to be created.
   */
  public static Object createObject(Constructor constructor,
            Object[] arguments)
  {
    Object object = null;

    try {
      object = constructor.newInstance(arguments);
      return object;
    } catch (InstantiationException e) {
      System.out.println(e);
    } catch (IllegalAccessException e) {
      System.out.println(e);
    } catch (IllegalArgumentException e) {
      System.out.println(e);
    } catch (InvocationTargetException e) {
      System.out.println(e);
    }
    return object;
  }

  public void debug(String msg) {
    if (_debug) {
      System.out.println("Debug: "+msg);
    }
  }

  public void setDebug(boolean debug) {
    debug("in debug mode");
    _debug = debug;
  }


}
```

A.16 Java Source: EmpathState

```
package edu.columbia.cs.coms4115.empath.model;

import java.util.HashMap;
import java.lang.reflect.Method;

/**
 * EmpathState models a state defined in Empath
 * i.e. there is a subclass of EmpathState for each state in
 * an Empath entity, including the top-level entity
 *
 * @author William Mee (wjm2107)
 */
public abstract class EmpathState{

  // protected Map<String, EmpathState> _subStates;
  // protected EmpathState _currState;
  protected static String _label;
  // protected static String _sendToState; /* The state name to be ret
urned by
  //                  secondaryTransition() */
  // protected static EmpathRuntime _runtimeEnv;
  protected static EmpathEntity _entity;

  protected HashMap<String,String> _eventLabels = new HashMap<String,S
tring>();


  public void onClockTick(int tick) {

  }

  /**
   * Sets the entity for communication back to the runtime
   *
   * @param entity
   */
```

```
public void setEntity(EmpathEntity entity) {
  _entity=entity;
}

public abstract void onEntry();
public abstract void onExit();

/**
 * Gets the state label
 * @return
 */
public String getLabel() {
  return _label;
}

/**
 * setter for label
 * @param label new value of label
 */
public void setLabel(String label) {
  _label=label;
}


protected void addEventLabel(String eventName, String label) {
  _eventLabels.put(eventName, label);
}

/**
 * gets a label for an event
 * @param event
 * @return
 */
public String getEventLabel(Method event) {
  String response;
  response = _eventLabels.get(event.getName());

  if (response == null)
```

```
    {
      response = event.getName();
    }

    return response;
  }

  public void output(String msg) {
    _entity.sendOutput(msg);
  }

}
```

A.17 Java Source: EmpathStateTree

```
package edu.columbia.cs.coms4115.empath.model;

import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.ListIterator;
import java.util.Map;
import java.util.Observable;
// import edu.columbia.cs.coms4115.empath.runtime.*;
// import edu.columbia.cs.coms4115.empath.eg.*;
import java.util.TreeMap;

public class EmpathStateTree extends Observable {
  protected Map<String, EmpathStateTree> _subStates;
  protected EmpathStateTree _parentState;
  protected String _name;
  protected Class _thisState;
  protected static EmpathEntity _entity;
//  protected boolean _terminalState;
  protected ArrayList<EmpathTransition> _transitions;
  protected ArrayList<Method> _events;
  protected EmpathStateTree _initState;

  /**
   * Constructor
   * @param entity the entity which this tree is associated with
   * @param name the name of the tree node
   * @param state the corresponding state class
   * @param parent parent tree
   */
  public EmpathStateTree(String name, Class state, EmpathStateTree par
ent) {
    _parentState = parent;
    _thisState = state;
    _name = name;
    _subStates = new TreeMap<String,EmpathStateTree>();
    _transitions = new ArrayList<EmpathTransition>();
    _events = null;
    _initState = null;

    if (_parentState != null)
    {
      _parentState.addSubState(name, this);
    }
  }


  /**
   * Constructor
   * @param entity the entity which this tree is associated with
   * @param state the corresponding state class
   * @param parent parent tree
   */
  public EmpathStateTree(Class state, EmpathStateTree parent) {
    this(state.getSimpleName(),state,parent);
  }

  public void setInit(EmpathStateTree initState)
  {
    _initState = initState;
  }

  public EmpathStateTree getInit()
  {
    return _initState;
  }

  public void addTransition(Method trigger, Method action, EmpathState
Tree target) {
    _transitions.add(new EmpathTransition(trigger, action, target));
  }

  public void addTransition(Class triggerClass, String triggerName, Cl
ass actionClass, String actionName, EmpathStateTree target)   {
    _transitions.add(new EmpathTransition(triggerClass, triggerName, a
ctionClass, actionName, target));
  }

  /**
   * Add a new transition using the underlying class in this tree
   * @param triggerName name of the trigger function
   * @param actionName name of the action
   * @param target the target tree
   */
  public void addTransition(String triggerName, String actionName, Emp
athStateTree target)   {
    _transitions.add(new EmpathTransition(_thisState, triggerName, act
ionName, target));
  }


  public ArrayList<Method> getEvents()
  {
    Method[] allMethods;
    int i;

    if (_events == null)
    {
      _events = new ArrayList<Method>();
      allMethods = _thisState.getMethods();

      i = 0;

      while (i < allMethods.length)
      {
        if (allMethods[i].getName().startsWith("_event_"))
        {
          _events.add(allMethods[i]);
        }

        i++;
      }
    }
    return _events;
  }

  public EmpathStateTree evaluateTransitions()
  {
    ListIterator cursor = _transitions.listIterator();
    EmpathTransition transition;
    EmpathStateTree response = null;

    if (_parentState != null) {
      response = _parentState.evaluateTransitions();

      if (response != null) {
        return response;
      }
    }

    while (cursor.hasNext()) {
      transition = (EmpathTransition) cursor.next();
      response = transition.evaluate(_entity._currentState);
      if (response != null) {
        debug("\t"+transition+" (true)");
        return response;
      } else {
        debug("\t"+transition+" (false)");
      }
    }

    return response;
  }

  public void addSubState(String stateName, EmpathStateTree subState)

  {
    _subStates.put(stateName, subState);
  }
```

```
public EmpathStateTree getParent()
{
  return _parentState;
}

public boolean isTerminal()
{
  if(! _transitions.isEmpty())
  {
    return false;
  }

  if (_parentState == null)
  {
    return true;
  }

  return _parentState.isTerminal();
}

public Class getState()
{
  return _thisState;
}

/**
 * Gets the name of the node
 * @return
 */
public String getName() {
  return _name;
}


/**
 * Get the entity this tree is associated with
 * @return
 */
```

```
  public static EmpathEntity getEntity() {
    return _entity;
  }


  /**
   * Sets the entity this tree is associated with
   * @param _entity
   */
  public static void setEntity(EmpathEntity _entity) {
    EmpathStateTree._entity = _entity;
  }

  /**
   * convenience method for doing debuging
   * @param msg debug message
   */
  public void debug(String msg) {
    _entity.debug(msg);
  }



}
```

A.18 Java Source: EmpathTransition

```
package edu.columbia.cs.coms4115.empath.model;


import java.lang.reflect.Method;

public class EmpathTransition {
  private Method _trigger;
  private Method _action;
  private EmpathStateTree _target;

  EmpathTransition (Method trigger, Method action, EmpathStateTree tar
get) {
    _trigger = trigger;
    _action = action;
    _target = target;
  }

  /**
   * constructor
   * @param clazz the class with which trigger and action are associat
ed
   * @param triggerName name of the trigger function
   * @param actionName name of the action function
   * @param target the target in the tree
   */
  EmpathTransition (Class clazz, String triggerName, String actionName
, EmpathStateTree target)   {
    try {
      _trigger = clazz.getMethod(triggerName, (Class[]) null);
      if (actionName != null) {
        _action = clazz.getMethod(actionName, (Class[]) null);
      }
    }
    catch (NoSuchMethodException e)
    {
      System.out.println(e);
  }

  _target = target;
 }


 EmpathTransition (Class triggerClass, String triggerName, Class acti
onClass, String actionName, EmpathStateTree target)   {
  try {
   _trigger = triggerClass.getMethod(triggerName, (Class[]) null);
   if (actionClass == null || actionName == null) {
    _action = null;
   } else {
    _action = actionClass.getMethod(actionName, (Class[]) null);
   }
  }
  catch (NoSuchMethodException e)
  {
   System.out.println(e);
  }

  _target = target;
 }

 public EmpathStateTree evaluate(EmpathState state) {
  Boolean result=null;
  try {
   result = (Boolean)_trigger.invoke(state, (Object [])null);
  } catch (Exception e) {
   e.printStackTrace();
   return null;
  }
  if (result.booleanValue())  {
   if (_action != null) {
    try {
     _action.invoke(state,(Object[]) null);
    } catch (Exception e) {
     e.printStackTrace();
```

```
        return null;
      }
    }

    return _target;
  } else {
    return null;
  }
}

@Override
public String toString() {
  return _trigger.getName()+" -> "+_target.getName();
}

}
```

A.19 Java Source: EmpathUI

```
package edu.columbia.cs.coms4115.empath.ui;

import java.lang.reflect.Method;
import java.util.ArrayList;
import edu.columbia.cs.coms4115.empath.runtime.*;


public abstract class EmpathUI {
  protected ArrayList<String> _eventList;
  protected EmpathRuntime _runtimeEnv;
  protected String _currentLabel;

  public abstract void displayOutput(String msg);

  public abstract void awaitInput();

  public void refreshCache() {
    _currentLabel = _runtimeEnv.getLabel();
    _eventList = _runtimeEnv.getEvents();
  }

  public abstract void refreshDisplay();
  public abstract void terminate();
}
```

A.20 Java Source: FunctionEntry

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.util.*;

import edu.columbia.cs.coms4115.empath.antlr.Type;

public class FunctionEntry extends SymbolTableEntry {

  DataRetType retType;
  ArrayList<DataRetType> argList;
  SymbolTable localVar;
  String  definition;

  public FunctionEntry()  {
    super(Type.FUNCTION);
    localVar = new SymbolTable();
    argList = new ArrayList<DataRetType>();
    definition = null;
  }

  public void addRetType(String t)  {
    if((t.equals("int"))||(t.equals("range")))
    {
      retType=DataRetType.INT;
    }
    if(t.equals("string"))
    {
      retType=DataRetType.STRING;
    }
    if(t.equals("float"))
    {
      retType=DataRetType.FLOAT;
    }
    if(t.equals("boolean"))
    {
      retType=DataRetType.BOOLEAN;
    }
    if(t.equals("void"))
    {
      retType=DataRetType.VOID;
    }
  }

  public void addArgs(String arg)
  {
    DataRetType dt = null;
    if(arg.equals("int"))
    {
      dt=DataRetType.INT;
    }
    if(arg.equals("string"))
    {
      dt=DataRetType.STRING;
    }
    if(arg.equals("float"))
    {
      dt=DataRetType.FLOAT;
    }
    if(arg.equals("boolean"))
    {
      dt=DataRetType.BOOLEAN;
    }
    if(arg.equals("range"))
    {
      dt=DataRetType.RANGE;
    }
    argList.add(dt);
  }

  public DataRetType getRetType()  {
    return retType;
  }

  /**
```

```
 *
 * @param def Generated code for function definition
 */
public void setDefinition(String def)  {
  definition = def;
}

/**
 *
 * @return function definition
 */
public String getDefinition()  {
  return definition;
}

public void setlocalSymTab(SymbolTable childSymTab)  {
  this.localVar = childSymTab;
}

public SymbolTable getlocalSymTab()  {
  return localVar;
}


public boolean verifyArgs(int pos , DataRetType drt)
{

  if(pos > argList.size())
    return false;
  if(argList.get(pos) !=  drt)
  {
    return false;
  }
  else
  {
    return true;
  }
  }

  public int getArgsLength()
  {
    return argList.size();
  }
}
```

A.21 Java Source: Range

```
package edu.columbia.cs.coms4115.empath.runtime;

public class Range {
  private int value;
  private int min;
  private int max;

  public Range(int minimum, int maximum, int initialValue)
  {
    min = minimum;
    max = maximum;
    value = initialValue;

    if (value > maximum)
    {
      value = maximum;
    }

    if (value < minimum)
    {
      value = minimum;
    }
  }

  public int newValue (int newValue)
  {
    value = newValue;

    if (value > max)
    {
      value = max;
    }

    if (value < min)
    {
      value = min;
    }

    return value;
  }

  public int getValue ()
  {
    return value;
  }

  public int incrementValue()
  {
    int oldValue = value;

    if (value < max)
    {
      value++;
    }

    return oldValue;
  }

  public int decrementValue()
  {
    int oldValue = value;

    if (value > min)
    {
      value --;
    }

    return oldValue;
  }

  /**
   * returns true if the value is at the maximum
   * @return
   */
```

```
  public boolean atMax() {
    return (max == value);
  }

  /**
   * returns true if the
   * @return
   */
  public boolean atMin()
  {
    return (min == value);
  }

  public int getMax() {
    return max;
  }

  public int getMin() {
    return min;
  }
}
```

A.22 Java Source: Returntype

```
package edu.columbia.cs.coms4115.empath.antlr;

public enum Returntype {STRING, BOOLEAN, INT, FLOAT, RANGE , VOID}
```

A.23 Java Source: StateEntry

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.util.*;

/**
 * State entry in the symbol table
 *
 * @author William Mee (wjm2107)
 */
public class StateEntry extends SymbolTableEntry {
  String name;
  boolean initFlag;
  SymbolTable childSymTab;
  List<Transition> transList;

  public StateEntry()  {
    super(Type.STATE);
    name = null;
    initFlag = false;
    childSymTab = null;
    transList = new ArrayList<Transition>();
  }

  public StateEntry(String name, boolean init)  {
    super(Type.STATE);
    this.name = name;
    initFlag = init;
    childSymTab = null;
    transList = new ArrayList<Transition>();
  }

  public boolean isInitState()  {
    return initFlag;
  }

  public void setChildSymTab(SymbolTable childSymTab)  {
    this.childSymTab = childSymTab;
  }

  /**
   * @return the child symbol table
   */
  public SymbolTable getChildSymTab()  {
    return childSymTab;
  }


  public void addTransition(Transition t)  {
    transList.add(t);
  }

  public List<Transition> getTransitionList() {
    return transList;
  }

  /**
   *
   * @return the state name
   */
  public String getName()  {
    return name;
  }

}
```

A.24 Java Source: SymbolTable

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.util.*;

/**
 * A symbol table containing variables, function and states for static
 semantic analysis
 * and code generation
 *
 * @author William Mee (wjm2107)
 */
public class SymbolTable {
  String name;
  String label=null;
  HashMap<String,SymbolTableEntry> symTab;
  // set of all state entries for easy lookup
  private Map<String,StateEntry> _states=new TreeMap<String,StateEntry
>();
  private List<String> _statesList = new ArrayList<String>();
  // map of all functions for easy lookup
  private Map<String,FunctionEntry> _functions = new TreeMap<String,Fu
nctionEntry>();
  // map of all variables
  private Map<String,VariableEntry> _variables = new TreeMap<String,Va
riableEntry>();
  SymbolTable parent;
  SymbolTableEntry parentEntry;

  public SymbolTable()  {
    symTab = new HashMap<String,SymbolTableEntry>();
    parent = null;
    parentEntry = null;
  }

  public SymbolTable(String name)  {
    this();
    this.name = name;
  }

  public SymbolTable(String name, SymbolTable parent, SymbolTableEntry
 parentEntry)  {
    symTab = new HashMap<String,SymbolTableEntry>();
    this.name = name;
    this.parent = parent;
    this.parentEntry = parentEntry;
  }

  /**
   * Add a new state entry to the symbol table
   * @param key the name of the state
   * @param value the StateEntry entry
   * @return true if added, false if already exists and not added
   */
  public boolean addEntry(String key, StateEntry value)  {
    // do not overwrite an exsting symbol
    if (symTab.get(key)!=null) {
      return false;
    }
    _states.put(key,value);
    _statesList.add(key);
    symTab.put(key, (SymbolTableEntry)value);
    return true;
  }


  /**
   * add a new function to the symbol table
   * @param key the function name
   * @param entry the function
   * @return true if added, false if the name already exists (so not a
dded)
   */
  public boolean addEntry(String key, FunctionEntry value) {
    if (symTab.get(key)!=null) {
```

```
    return false;
  }
  _functions.put(key,value);
  symTab.put(key, (SymbolTableEntry)value);
  return true;
}

/**
 * Add a variable to the symbol table
 * @param key
 * @param value
 * @return
 */
public boolean addEntry(String key, VariableEntry value) {
  if (symTab.get(key)!=null) {
    return false;
  }
  _variables.put(key,value);
  symTab.put(key, (SymbolTableEntry)value);
  return true;
}

/**
 * add a new general entry
 * @param key
 * @param value
 * @return
 */
public boolean addEntry(String key, SymbolTableEntry value)  {
  if (symTab.get(key)!=null) {
    return false;
  }
  symTab.put(key, value);
  return true;
}

public SymbolTableEntry getEntry(String key)  {
  return symTab.get(key);
  }

  public boolean hasEntry(String key)  {
    return symTab.containsKey(key);
  }

  public boolean hasInitState()  {
    Collection<SymbolTableEntry> cste = symTab.values();
    Iterator<SymbolTableEntry> itste = cste.iterator();
    while(itste.hasNext())  {
      SymbolTableEntry ste = itste.next();
      if (ste.getType()==Type.STATE && ((StateEntry)ste).isInitState()
)
        return true;
    }
    return false;
  }

  /**
   * Return an iterator over all state labes stored in this symbol tab
le
   * @return
   */
  public Iterator<String> variableIterator() {
    return _variables.keySet().iterator();
  }

  /**
   * @return an Iterable over the state lables
   */
  public Iterable<String> stateIterable() {
    return _statesList;
  }

  /**
   * @return an iterator over all state labes stored in this symbol ta
ble
   */
  public Iterator<String> stateIterator() {
    return _statesList.iterator();
  }

  /**
   * @returns an iterator over all the function labels stored in this
symbol table
   */
  public Iterator<String> functionIterator() {
    return _functions.keySet().iterator();
  }

  /**
   * @return an iterable for the functions
   */
  public Iterable<String> functionIterable() {
    return _functions.keySet();
  }

  /**
   * @return an iterable for variables
   */
  public Iterable<String> variableIterable() {
    return _variables.keySet();
  }

  /**
   * @return the name of this symbol table
   */
  public String getName() {
    return name;
  }

  /**
   *
   * @param name set the name of parent entity/state
   */
  public void setName(String name)  {
    this.name = name;
  }

  /**
   * @return the parent symbol table, or null if there is none (root)
   */
  public SymbolTable getParent() {
    return parent;
  }

  /**
   * @return the parent symbol table entry, or null if there is none (
root)
   */
  public SymbolTableEntry getParentEntry() {
    return parentEntry;
  }

  /**
   *
   * @param l label of this state/function
   */
  public void setLabel(String l)  {
    label = l;
  }

  public DataRetType verifyVar(String varname)
  {
    //boolean found = false;
    SymbolTable temp = null;
    temp = this;
    do
    {
      if((temp.hasEntry(varname))&&(temp.getEntry(varname).getType()==
Type.VARIABLE))
      {
        //found = true;
        //break;
    VariableEntry v = (VariableEntry) temp.getEntry(varname);
    return v.getIDType();
   }
   temp = temp.parent;

  }while((temp)!=null);
  return null;
  //return found;
 }

 public FunctionEntry verifyFunc(String funcname)
 {
  //boolean found = false;
  SymbolTable temp = null;
  temp = this;
  do
  {
   if((temp.hasEntry(funcname))&&(temp.getEntry(funcname).getType()
==Type.FUNCTION))
   {
    return ((FunctionEntry) temp.getEntry(funcname));

    //found = true;
    //break;
   }
   temp = temp.parent;

  }while((temp)!=null);
  return null;
  //return found;
 }

 public boolean verifyState(String stateName)  {
  SymbolTable temp = null;
  temp = this;
  do  {
   if((temp.hasEntry(stateName))&&(temp.getEntry(stateName).getType
()==Type.STATE))  {
        return true;
      }
      temp = temp.parent;
    }
    while(temp!=null);
    return false;
  }

  public boolean verifyUniqueState(String stateName)  {
    for (String state:this.stateIterable()) {
      if(stateName.equals(state))
        return false;
      else  {
        StateEntry se = (StateEntry)this.getEntry(state);
        if(se.getChildSymTab()!=null)
          if(!(se.getChildSymTab().verifyUniqueState(stateName)))
            return false;
      }
    }
    return true;
  }

  public VariableEntry getVarEntry(String varname)
  {
    //boolean found = false;
    SymbolTable temp = null;
    temp = this;
    do
    {
      if((temp.hasEntry(varname))&&(temp.getEntry(varname).getType()==
Type.VARIABLE))
      {
        //found = true;
        //break;
        VariableEntry v = (VariableEntry) temp.getEntry(varname);
        return v;
      }
      temp = temp.parent;
    }while((temp)!=null);
    return null;
    //return found;
  }

  /**
   *
   * @return label of this state
   */
  public String getLabel()  {
    return label;
  }

}
```

A.25 Java Source: SymbolTableEntry

```
package edu.columbia.cs.coms4115.empath.antlr;

public abstract class SymbolTableEntry {
  Type type;
  String label=null;

  protected SymbolTableEntry(Type t) {
    this.type = t;
  }

  public Type getType()  {
    return type;
  }


  /**
   *
   * @param l label of this state/function
   */
  public void setLabel(String l)  {
    label = l;
  }

  /**
   *
   * @return label of this state
   */
  public String getLabel()  {
    return label;
  }

}
```

A.26 Java Source: TextUI

```
package edu.columbia.cs.coms4115.empath.ui;

import java.lang.reflect.Method;
import java.util.Arrays;
import edu.columbia.cs.coms4115.empath.runtime.*;
import java.util.ListIterator;
import java.io.*;

public class TextUI extends EmpathUI {

  public static final String OUTPUT_MARKER="Output: ";
  public static final String REFRESH_MARKER="-------------------------
-----------\n";

  public TextUI(EmpathRuntime ERE) {
    _runtimeEnv = ERE;
  }

  public void displayOutput(String msg) {
    System.out.println(OUTPUT_MARKER+msg);
  }

  public void refreshDisplay() {
    System.out.println(REFRESH_MARKER+"Current state is " + _currentLa
bel);

    int i = 0;

    while (i < _eventList.size())
    {
      System.out.println("Enter " + i + " to " + _eventList.get(i));

      i++;
    }

    System.out.println("Enter m to display this menu.");
  System.out.println("Enter q to quit.");
  System.out.println("Enter nothing to advance clock");
 }

 public void awaitInput() {
  BufferedReader reader = new BufferedReader(new InputStreamReader(S
ystem.in));

  try
  {
   String input = reader.readLine();

   while (input.compareTo("M") == 0 || input.compareTo("m") == 0) {


    refreshDisplay();

    input = reader.readLine();
   }

   if (input.compareTo("Q") == 0 || input.compareTo("q") == 0) {
    _runtimeEnv.quit();
   }

   while (input.length() > 0) {
    int eventRequest = Integer.parseInt(input);

    if (eventRequest < _eventList.size())
    {
     Class[] param;
     param =_runtimeEnv.getEventArgs(eventRequest);

     Object[] eventParams;

     if (param != null){
      eventParams = new Object[param.length];

      int i = 0;
      System.out.println("looking for " + param.length + " argum
ents.");

      while (i < param.length) {
       System.out.print("Please enter a value of type " + param
[i].getName() + ": ");

       if (param[i].equals(String.class))
       {
        eventParams[i] = reader.readLine();
       }

       if (param[i].getName().compareTo("int") == 0)
       {
        eventParams[i] = Integer.parseInt(reader.readLine());
       }

       if (param[i].getName().compareTo("float") == 0)
       {
        eventParams[i] = Float.valueOf(reader.readLine());
       }

       i++;
      }
     }
     else
     {
      eventParams = null;
     }

     _runtimeEnv.callEvent(eventRequest, (Object[]) eventParams);


    }
    else
    {
     System.out.println("The number you entered is not a valid op
tion.");
        }

        System.out.println("Enter another event number, m for the menu
, q to quit, or press enter to increment clock...");
        input = reader.readLine();

        while (input.compareTo("M") == 0 || input.compareTo("m") == 0)
 {

          refreshDisplay();

          input = reader.readLine();
        }

        if (input.compareTo("Q") == 0 || input.compareTo("q") == 0) {
          _runtimeEnv.quit();
        }
      }

      System.out.println("Advancing clock");
      _runtimeEnv.callClockTick();
    }
    catch (NumberFormatException e)
    {
      System.out.println("Your entry was not recognized as a number. P
lease request an action again.");
    }
    catch (NullPointerException e)
    {
      System.out.println("Your entry was not recognized as a number. P
lease request an action again.");
    }
    catch (IOException e)
    {
      System.out.println(e);
    }
```

```
  }

  public void terminate() {
    System.out.println("Exiting.");
  }
}
```

A.27 Java Source: Transition

```
package edu.columbia.cs.coms4115.empath.antlr;

public class Transition {
  String fromState, toState;
  String condition;    //  everything inside of the braces
  String action;      // action (output)
  boolean functionCondition;  // flag indicating whether this is an ex
pression or a function reference

  public Transition()  {}

  public Transition(String fromState, String toState, String condition
)  {
    this.fromState = fromState;
    this.toState = toState;
    this.condition = condition;
    this.action = null;
  }

  public Transition(String fromState, String toState, String condition
, String action)  {
    this.fromState = fromState;
    this.toState = toState;
    this.condition = condition;
    this.action = action;
  }

  public void setFunctionCondition()  {
    this.functionCondition = true;
  }


  /**
   * @return true if the condition triggering the state change is a fu
nction
   */
  public boolean isFunctionCondition() {
    return functionCondition;
  }

  /**
   * gets the condition, i.e. the stuff between the braces of the tran
sition condition
   * @return
   */
  public String getCondition() {
    return condition;
  }

  public String getAction() {
    return action;
  }

  public String getFromState() {
    return fromState;
  }

  public String getToState() {
    return toState;
  }

  @Override
  public String toString() {
    return fromState+"->"+toState+" on "+condition;
  }



}
```

A.28 Java Source: Type

```
package edu.columbia.cs.coms4115.empath.antlr;

public enum Type {VARIABLE, FUNCTION, STATE}
```

A.29 Java Source: VariableEntry

```
package edu.columbia.cs.coms4115.empath.antlr;

public class VariableEntry extends SymbolTableEntry {
  DataRetType datatype;
  String initialValue = null;
  String rangeLower = null;  // only for range data type
  String rangeUpper = null;  // only for range data type

  VariableEntry() {
    super(Type.VARIABLE);
  }

  public void addIDType(String t)
  {
    if(t.equals("int"))
    {
      datatype=DataRetType.INT;
    }
    if(t.equals("string"))
    {
      datatype=DataRetType.STRING;
    }
    if(t.equals("float"))
    {
      datatype=DataRetType.FLOAT;
    }
    if(t.equals("boolean"))
    {
      datatype=DataRetType.BOOLEAN;
    }
    if(t.equals("range"))
    {
      datatype=DataRetType.RANGE;
    }

  }
  public DataRetType getIDType()  {
    return datatype;
  }

  public String getInitialValue()  {
    return initialValue;
  }

  public void setInitialValue(String initialValue)  {
    this.initialValue = initialValue;
  }

  public String getRangeLower()  {
    return rangeLower;
  }

  public void setRangeLower(String rangeLower)  {
    this.rangeLower = rangeLower;
  }

  public String getRangeUpper()  {
    return rangeUpper;
  }

  public void setRangeUpper(String rangeUpper)  {
    this.rangeUpper = rangeUpper;
  }

  /**
   * @return a string which can be used in Java code for the initial v
alue of this entry
   */
  public String getJavaInitialValue() {
    switch (datatype) {
    case STRING:   return initialValue==null?"null":initialValue;
    case INT:    return initialValue==null?"0":initialValue;
    case FLOAT:   return initialValue==null?"0.0f":initialValue+"f";
    case BOOLEAN:   return initialValue==null?"false":initialValue;
    case RANGE:
      if (initialValue==null) {
        return "null";
      }
      String lower=rangeLower==null?"0":rangeLower;
      String upper=rangeUpper==null?"0":rangeUpper;
      return "new Range("+lower+","+upper+","+initialValue+")";
    }
    return null;
  }

}
```

A.30 Java Test Source: ASTViewer

```
package edu.columbia.cs.coms4115.empath.tst;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import edu.columbia.cs.coms4115.empath.antlr.BaseAST;
import edu.columbia.cs.coms4115.empath.antlr.EmpathLexer;
import edu.columbia.cs.coms4115.empath.antlr.EmpathParser;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;

public class ASTViewer {

  File _file;

  public ASTViewer(File file) {
    _file=file;
  }

  /**
   * Print message and exit
   * @param msg
   */
  private static void bale(String msg) {
    System.err.println(msg);
    System.exit(1);
  }

  private static void syntaxAndBale() {
    bale("Syntax: astviewer [-t|-g] source-file\n\t-t text\n\t-g graph
ical (default)");
  }

  private AST getAST() {
```

```
  FileInputStream fis=null;
  try {
     fis = new FileInputStream(_file);
  } catch (FileNotFoundException e) {
    bale("could not find file");
  }
  EmpathLexer lexer = new EmpathLexer(fis);
  EmpathParser parser=new EmpathParser(lexer);
  parser.setASTNodeClass(BaseAST.class.getName());
  try {
    parser.entitySpec();
  } catch (Exception e) {
    bale(e.getMessage());
  }
  try {
    fis.close();
  } catch (IOException e) {
    bale(e.getMessage());
  }
  AST ast = parser.getAST();
  return ast;
}

private void viewGraphical() {
  AST ast = getAST();
  ASTFrame frame = new ASTFrame("Empath AST", ast);
  frame.setVisible(true);
}

public void viewText() {
  AST ast = getAST();
  System.out.println(ast.toStringList());
}

/**
 * @param args
 */
public static void main(String args[]) {
    boolean graphical=false;
    File sourceFile=null;
    if (args.length==1) {
      sourceFile = new File(args[0]);
      graphical=true;
    } else if (args.length==2) {
      sourceFile = new File(args[1]);
      if ("-t".equals(args[0])) {
        graphical=false;
      } else if ("-g".equals(args[0])) {
        graphical=true;
      } else {
        syntaxAndBale();
      }
    } else {
      syntaxAndBale();
    }
    if (!sourceFile.exists()) {
      bale("could not open file "+args[0]);
    }
    ASTViewer viewer = new ASTViewer(sourceFile);
    if (graphical) {
      viewer.viewGraphical();
    } else {
      viewer.viewText();
    }
  }
}
```

A.31 Java Test Source: AllTests

```
package edu.columbia.cs.coms4115.empath;

import edu.columbia.cs.coms4115.empath.antlr.CodeGenWalkerTest;
import edu.columbia.cs.coms4115.empath.antlr.EmpathLexerTest;
import edu.columbia.cs.coms4115.empath.antlr.EmpathParserTest;
import edu.columbia.cs.coms4115.empath.antlr.FuncSSATest;
import edu.columbia.cs.coms4115.empath.antlr.LineNumberTest;
import edu.columbia.cs.coms4115.empath.antlr.SymbolTableTest;
import junit.framework.Test;
import junit.framework.TestSuite;

/**
 * Suite of all unit tests for Empath
 *
 * @author William Mee (wjm2107)
 */
public class AllTests {

  public static Test suite() {
    TestSuite suite = new TestSuite("Empath Test Suite");
    // lexer tests
    suite.addTestSuite(EmpathLexerTest.class);
    // parser tests
    suite.addTestSuite(EmpathParserTest.class);
    // test  of reporting line numbers / BaseAST class
    suite.addTestSuite(LineNumberTest.class);
    // function-based ssa
    suite.addTestSuite(FuncSSATest.class);
    // state-based ssa
    suite.addTestSuite(SymbolTableTest.class);
    // code generation tests
    suite.addTestSuite(CodeGenWalkerTest.class);
    return suite;
  }

}
```

A.32 Java Test Source: BaseCodeGenTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

import antlr.RecognitionException;
import antlr.SemanticException;
import antlr.TokenStreamException;
import antlr.collections.AST;

public class BaseCodeGenTest extends BaseParserTest {

  protected CodeGenWalker generate(File file)  {
    EmpathParser parser = null;
    EmpathTreeWalker walker = new EmpathTreeWalker();
    walker.filename=file.getPath();
    CodeGenWalker gen = new CodeGenWalker();
    gen.filename=file.getPath();

    // Parse the file
    try {
      parser = makeParser(new FileInputStream(file));
    } catch (FileNotFoundException e) {
      fail(e.getMessage());
    }
    try {
      parser.entitySpec();
    } catch (RecognitionException e) {
      fail(e.getMessage());
    } catch (TokenStreamException e) {
      fail(e.getMessage());
    }

    AST ast = parser.getAST();
    // Walk the AST, do static semantic analysis and generate the symb
ol table
    try {
      walker.entitySpec(ast);
    } catch (SemanticException e)  {
      fail(e.getMessage()+" in line "+e.getLine());
    } catch (RecognitionException e) {
      fail(e.getMessage()+" in line "+e.getLine());
    }

    // Get the symbol table from SSA walker and pass it to the code ge
nerator walker
    gen.setSymTable(walker.getSymTable());

    // Walk the AST, generate code for functions
    try {
      gen.entitySpec(ast);
    } catch (SemanticException e)  {
      fail(e.getMessage()+" in line "+e.getLine());
    } catch (RecognitionException e) {
      fail(e.getMessage()+" in line "+e.getLine());
    }

    return gen;
  }


}
```

A.33 Java Test Source: BaseParserTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.StringReader;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamRecognitionException;
import antlr.collections.AST;
import junit.framework.TestCase;

public class BaseParserTest extends TestCase {

  protected void setUp() throws Exception {
    super.setUp();
  }

  protected void tearDown() throws Exception {
    super.tearDown();
  }

  /**
   * Construct a parser from the given String input
   * @param input
   * @return
   */
  protected EmpathParser makeParser(String input) {
    EmpathLexer lexer = new EmpathLexer(new StringReader(input));
    EmpathParser parser = new EmpathParser(lexer);
    return parser;
  }
  /**
   * Run a production with a given input
   * @param production the production to run
   * @param input the string input
   * @return
   */
  protected AST assertProductionOk(String production, String input) {
    EmpathParser parser = makeParser(input);
    parser.setASTNodeClass(BaseAST.class.getName());
    Method method=null;
    AST ast = null;
      try {
        method=parser.getClass().getMethod(production, (Class[]) null)
;
      } catch (NoSuchMethodException e) {
        fail("could not find production "+production);
      }
      try {
        ast=(AST)method.invoke(parser, (Object[]) null);
    } catch (IllegalArgumentException e) {
      fail(e.getMessage());
    } catch (IllegalAccessException e) {
      fail(e.getMessage());
    } catch (InvocationTargetException e) {
      Throwable cause = e.getCause();
      fail(cause.getClass()+": "+cause.getMessage());
    }
    try {
      parser.match(EmpathParser.EOF);
    } catch (Exception e) {
      fail(e.getClass()+":"+e.getMessage());
    }
    return ast;
  }

  /**
   * Check that a production does not succeed against the given input
   * @param production the production to run
   * @param input the string input
   * @return
   */
  protected void assertProductionNotOk(String production, String input
) {
    EmpathParser parser = makeParser(input);
    parser.setASTNodeClass(BaseAST.class.getName());
    Method method=null;
      try {
        method=parser.getClass().getMethod(production, (Class[]) null)
;
      } catch (NoSuchMethodException e) {
        fail("could not find production "+production);
      }
      try {
        method.invoke(parser, (Object[]) null);
      fail("could run production "+production+" against invalid input"
);
    } catch (IllegalArgumentException e) {
      fail(e.getMessage());
    } catch (IllegalAccessException e) {
      fail(e.getMessage());
    } catch (InvocationTargetException e) {
      Throwable cause = e.getCause();
      //expected
    }
  }



  /**
   * Create a parser from a File
   * @param inputFile the input code file
   * @return an EmpathParser object from the code
   */
  protected EmpathParser makeParser(FileInputStream fis) {
    EmpathParser parser=null;
    EmpathLexer lexer = new EmpathLexer(fis);
    parser = new EmpathParser(lexer);
    parser.setASTNodeClass(BaseAST.class.getName());
    return parser;
  }

  /**
   * Parse a file and check that is has valid content
   * @param file
   * @return
   */
  protected AST assertParsedOk(File file) {
    FileInputStream fis=null;
    try {
      fis = new FileInputStream(file);
    } catch (FileNotFoundException e) {
      fail(e.getMessage());
    }
    EmpathParser parser = makeParser(fis);
    parser.setASTNodeClass(BaseAST.class.getName());
    try {
      parser.entitySpec();
    } catch (TokenStreamRecognitionException e) {
      fail(file.getName()+":"+e.getMessage()+" line "+e.recog.line+" p
os "+e.recog.column);
    } catch (RecognitionException e) {
      fail(file.getName()+": "+e.getClass()+":"+e.getMessage());
    } catch (TokenStreamException e) {
      fail(file.getName()+": "+e.getClass()+":"+e.getMessage());
    }
    AST ast = parser.getAST();
    assertNotNull(ast);
    assertEquals("entity",ast.getText());
    try {
      fis.close();
    } catch (IOException e) {
      fail(e.getMessage());
    }
    return ast;
  }

  /**
   * check that a string is valid Empath code
   * @param input
   * @return
   */
  protected AST assertParsedOk(String input) {
    EmpathParser parser=makeParser(input);
    try {
      parser.entitySpec();
    } catch (Exception e) {
      fail(e.getClass()+":"+e.getMessage());
    }
    try {
      parser.match(EmpathParser.EOF);
    } catch (Exception e) {
      fail(e.getClass()+":"+e.getMessage());
    }
    AST ast = parser.getAST();
    assertNotNull(ast);
    assertEquals(ast.getText(),"entity");
    return ast;
  }

  /**
   * Check that a string is NOT valid Empath code
   * @param input
   */
  public void assertNotParsedOk(String input) {
    EmpathParser parser=makeParser(input);
    try {
      parser.entitySpec();
      fail("could parse invalid input!");
    } catch (Exception e) {
      //expected
    }
```

```
    try {
      parser.match(EmpathParser.EOF);
    } catch (Exception e) {
      //expected
    }
    AST ast = parser.getAST();
    assertNull(ast);
  }



}
```

A.34 Java Test Source: BaseWalkerTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

import antlr.RecognitionException;
import antlr.SemanticException;
import antlr.TokenStreamException;
import antlr.collections.AST;

public abstract class BaseWalkerTest extends BaseParserTest {

  protected EmpathTreeWalker walkFile(File file) {
    EmpathParser parser = null;
//    parser.setASTNodeClass(BaseAST.class.getName());
    EmpathTreeWalker walker = new EmpathTreeWalker();
    walker.filename=file.getPath();
    try {
      parser = makeParser(new FileInputStream(file));
    } catch (FileNotFoundException e) {
      fail(e.getMessage());
    }
    try {
      parser.entitySpec();
    } catch (RecognitionException e) {
      fail(e.getMessage());
    } catch (TokenStreamException e) {
      fail(e.getMessage());
    }
    AST ast = parser.getAST();
    try {
      walker.entitySpec(ast);
    } catch (SemanticException e)  {
      fail(e.getMessage()+" in line "+e.getLine());
    } catch (RecognitionException e) {
      fail(e.getMessage()+" in line "+e.getLine());
    }
    return walker;
  }


  protected EmpathTreeWalker walkFileForSSA(File file) throws Semantic
Exception {
    EmpathParser parser = null;
    EmpathTreeWalker walker = new EmpathTreeWalker();
    walker.filename=file.getPath();
    try {
      parser = makeParser(new FileInputStream(file));
    } catch (FileNotFoundException e) {
      fail(e.getMessage());
    }
    try {
      parser.entitySpec();
    } catch (RecognitionException e) {
      fail(e.getMessage());
    } catch (TokenStreamException e) {
      fail(e.getMessage());
    }
    AST ast = parser.getAST();
    try {
      walker.entitySpec(ast);
    } catch (SemanticException e)  {
      throw e;
    } catch (RecognitionException e) {
      fail(e.getMessage()+" in line "+e.getLine());
    }
    return walker;
  }


}
```

A.35 Java Test Source: CodeGenRunner

```
package edu.columbia.cs.coms4115.empath.codegen;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

import antlr.RecognitionException;
import antlr.SemanticException;
import antlr.TokenStreamException;
import antlr.collections.AST;
import edu.columbia.cs.coms4115.empath.antlr.BaseAST;
import edu.columbia.cs.coms4115.empath.antlr.CodeGenWalker;
import edu.columbia.cs.coms4115.empath.antlr.EmpathLexer;
import edu.columbia.cs.coms4115.empath.antlr.EmpathParser;
import edu.columbia.cs.coms4115.empath.antlr.EmpathTreeWalker;
import edu.columbia.cs.coms4115.empath.antlr.TestConstants;

public class CodeGenRunner {

  private void fail(String msg) {
    System.err.println(msg);
    System.exit(1);
  }


  /**
   * Create a parser from a File
   * @param inputFile the input code file
   * @return an EmpathParser object from the code
   */
  private EmpathParser makeParser(FileInputStream fis) {
    EmpathParser parser=null;
    EmpathLexer lexer = new EmpathLexer(fis);
    parser = new EmpathParser(lexer);
    parser.setASTNodeClass(BaseAST.class.getName());
    return parser;
```

```
}

protected CodeGenWalker walkFile(File file) {
  EmpathParser parser = null;
  EmpathTreeWalker walker = new EmpathTreeWalker();
  walker.filename=file.getPath();
  CodeGenWalker gen = new CodeGenWalker();
  gen.filename=file.getPath();
  try {
    parser = makeParser(new FileInputStream(file));
  } catch (FileNotFoundException e) {
    fail(e.getMessage());
  }
  try {
    parser.entitySpec();
  } catch (RecognitionException e) {
    fail(e.getMessage());
  } catch (TokenStreamException e) {
    fail(e.getMessage());
  }
  AST ast = parser.getAST();
  try {
    walker.entitySpec(ast);
    gen.setSymTable(walker.getSymTable());
    gen.entitySpec(ast);
  } catch (SemanticException e)  {
    fail(e.getMessage()+" in line "+e.getLine());
  } catch (RecognitionException e) {
    fail(e.getMessage()+" in line "+e.getLine());
  }
  return gen;
}

private void run(File file) {
  CodeGenWalker walker = walkFile(file);
  CodeGenerator generator = new CodeGenerator(walker.getSymTable());

  try {
      generator.generate();
    } catch (CodeGenException e) {
      e.printStackTrace();
    }

  }

  /**
   * @param args
   */
  public static void main(String[] args) {
    if (args.length!=1) {
      System.err.println("need empath input file");
      System.exit(-1);
    }
    File file = new File(args[0]);
    CodeGenRunner runner = new CodeGenRunner();
    runner.run(file);

  }

}
```

A.36 Java Test Source: CodeGenWalkerTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import antlr.RecognitionException;
import antlr.collections.AST;

public class CodeGenWalkerTest extends BaseCodeGenTest  {

  public void testCG1()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F1);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("f1");
    String def = fe.getDefinition();
//    System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
//    System.out.println(def);
    assertEquals("public boolean f1(){return (a == b);}", def);
  }

  public void testCG2()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F2);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("onClockTick");
    String def = fe.getDefinition();
//    System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
//    System.out.println(def);
    assertEquals("public void onClockTick(int tick){age++ ;hunger.incr
ementValue();super.onClockTick(tick);}", def);
  }

  public void testCG3()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F3);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("f3");
    String def = fe.getDefinition();
//    System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
//    System.out.println(def);
    assertEquals("public String f3(int x, String s){if (hunger.atMax()
){x += 5;hunger.newValue(hunger.getValue() - ((5 * 2) - 3));hunger.dec
rementValue();output(\"hunger@max\");}else{output(s);}return s;}", def
);
  }

  public void testCG4()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F4);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("f4");
    String def = fe.getDefinition();
//    System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
//    System.out.println(def);
    assertEquals("public void f4(){int i = 0;for(i = 0; (i < 10); i++
){if (((tick % 5) == 0)){hunger.newValue(f());}else{output(\"Tick not
5\");}}}", def);
  }

  public void testCG5()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F5);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("f5");
    String def = fe.getDefinition();
//    System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
```

```
//    System.out.println(def);
    assertEquals("public void _event_f5(boolean c, Range d){boolean a
= false;boolean b = false;r.newValue(-10);while((r.getValue() < 11)){i
f (((r.getValue() % 2) == 0)){a = (b && c);}else{a = (b || (r.getValue
() < 5));}r.incrementValue();}r.newValue(r.getValue() - -10);}", def);

  }

  public void testCG6()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F6);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("onClockTick");
    String def = fe.getDefinition();
  //  System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
    System.out.println(def);
    assertEquals("public void onClockTick(int tick){if (((tick % 4) ==
 0)){if ((\"spring\".equals(season))){season = \"summer\";}else{if ((s
eason.equals(\"summer\"))){season = \"autumn\";}else{if ((season.equal
s(\"autumn\"))){season = \"winter\";}else{if ((season.equals(\"winter\
"))){season = \"spring\";}}}}}super.onClockTick(tick);}", def);
  }
/*
  public void testCG7()  {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_F7);
    SymbolTable st = gen.getSymTable();

    FunctionEntry fe = (FunctionEntry)st.getEntry("f7");
    String def = fe.getDefinition();
    System.out.println(def);
    def = def.replaceAll("\t", "");
    def = def.replaceAll("\n", "");
    System.out.println(def);
    assertEquals("public boolean f7(){return (a == b);}", def);
  }
```

```
public void testCG8()  {
  CodeGenWalker gen = generate(TestConstants.CODE_GEN_F8);
  SymbolTable st = gen.getSymTable();

  FunctionEntry fe = (FunctionEntry)st.getEntry("f8");
  String def = fe.getDefinition();
  System.out.println(def);
  def = def.replaceAll("\t", "");
  def = def.replaceAll("\n", "");
  System.out.println(def);
  assertEquals("public boolean f8(){return (a == b);}", def);
}

public void testCG9()  {
  CodeGenWalker gen = generate(TestConstants.CODE_GEN_F9);
  SymbolTable st = gen.getSymTable();

  FunctionEntry fe = (FunctionEntry)st.getEntry("f9");
  String def = fe.getDefinition();
  System.out.println(def);
  def = def.replaceAll("\t", "");
  def = def.replaceAll("\n", "");
  System.out.println(def);
  assertEquals("public boolean f9(){return (a == b);}", def);
}

public void testCG10()  {
  CodeGenWalker gen = generate(TestConstants.CODE_GEN_F10);
  SymbolTable st = gen.getSymTable();

  FunctionEntry fe = (FunctionEntry)st.getEntry("f10");
  String def = fe.getDefinition();
  System.out.println(def);
  def = def.replaceAll("\t", "");
  def = def.replaceAll("\n", "");
  System.out.println(def);
  assertEquals("public boolean f10(){return (a == b);}", def);
}
```

```
  */

  /*  public void testCodeGen() {
    CodeGenWalker gen = generate(TestConstants.CODE_GEN_FROG);
    SymbolTable st = gen.getSymTable();

    for(String funcName:st.functionIterable())  {
      FunctionEntry fe = (FunctionEntry)st.getEntry(funcName);
      System.out.println("Function: "+funcName);
      System.out.println(fe.getDefinition());
    }

  }
  */
//  public void testExpression() {
//    EmpathParser parser = makeParser("a=2+2;");
//    EmpathParser parser = makeParser("a=2*3+2;");
//    EmpathParser parser = makeParser("a=2>2;");  // should catch thi
s in SSA
//    EmpathParser parser = makeParser("a+=2+2;");
//    EmpathParser parser = makeParser("a=a || b;");
//  }

}
```

A.37 Java Test Source: EmpathLexerTest

```
package edu.columbia.cs.coms4115.empath.antlr;


import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;

import antlr.ANTLRTokenTypes;
import antlr.Token;
import antlr.TokenStreamException;

import junit.framework.TestCase;

/**
 * Tests the lexer
 * @author William Mee (wjm2107)
 *
 */
public class EmpathLexerTest extends TestCase {

  public void setUp() {
  }


  /**
   * Returns a lexer
   * @param input
   * @return
   */
  private EmpathLexer getLexer(String input) {
        EmpathLexer lexer = new EmpathLexer(new StringReader(input));
        return lexer;
  }


  /**
   * Returns the first token produced from the given string
   * @param input string
   * @return a Token if matched
   * @throws TokenStreamException
   */
  private Token lexString(String input) throws TokenStreamException {
        EmpathLexer lexer = new EmpathLexer(new StringReader(input));
        Token token = lexer.nextToken();
        return token;

  }

  private void assertValidInput(String input,int type) {
    assertValidInput(input,type,null);
  }

  private void assertValidInput(String input,int type, String text) {
    Token token=null;
    try {
      token = lexString(input);
    } catch (TokenStreamException e) {
      fail(e.getMessage());
    }
    assertNotNull(token);
    if (text!=null) {
      assertEquals(text,token.getText());

    }
    assertEquals(type,token.getType());

  }


  private void assertValidInput(String input, int [] types) {
        EmpathLexer lexer = new EmpathLexer(new StringReader(input));
    Token token=null;
    List tokens=new ArrayList();
    int tokenNum=1;
```

```
    boolean end=false;
    do {
      try {
        token = lexer.nextToken();
        tokenNum++;
        assertNotNull(token);
      } catch (TokenStreamException e) {
        fail("token #"+tokenNum+" " + e.getMessage());
      }
      end=(token.getType()==ANTLRTokenTypes.EOF);
      if (!end) {
        tokens.add(token);
      }
    } while (!end);
    // are the two lists the same size
    if (types.length!=tokens.size()) {
      assertEquals("did not get expected number of tokens",types.lengt
h,tokens.size());
    }
    for (int i=0;i<types.length;i++) {
      token=(Token)tokens.get(i);
      int type = types[i];
      assertEquals("failed on token with text "+token.getText(),type,t
oken.getType());
    }

  }


  private void assertInvalidInput(String input) {
    try {
      lexString(input);
      fail("accepted token "+input+", should not have");
    } catch (TokenStreamException e) {
      // expected
    }

  }
  public void testPunctuation() {
    assertValidInput("=",EmpathLexer.ASSIGN);
    assertValidInput(".",EmpathLexer.DOT);
    //assertValidInput("..",EmpathLexer.DBLDOT);
  }

  /**
   * Tests that the lexer ignores newline characters
   *
   */
  public void testNewlines() {
    int [] expectedTypes={EmpathLexer.ID,EmpathLexer.SCOLON,EmpathLexe
r.ID,EmpathLexer.SCOLON};
    assertValidInput("myVar;\nmyOtherVar;",expectedTypes);
  }

  public void testIdentifier() {
    assertValidInput("myVar",EmpathLexer.ID);
  }


  public void testSimpleComment() {
    int [] expectedTypes={EmpathLexer.ID,EmpathLexer.ASSIGN,EmpathLexe
r.INT_LIT,EmpathLexer.SCOLON,EmpathLexer.SCOLON};
    assertValidInput("myVar=5; //my inline comment\n;",expectedTypes);

  }

  public void testMultilineComment() {
    int [] expectedTypes={EmpathLexer.ID,EmpathLexer.ASSIGN,EmpathLexe
r.INT_LIT,EmpathLexer.SCOLON,EmpathLexer.SCOLON};
    assertValidInput("myVar=5; /*my inline comment\ncarrieson*/;",expe
ctedTypes);
  }


  public void testString() {
    assertValidInput("\"foo bar\"",EmpathLexer.STRING_LIT,"foo bar");
    assertValidInput("\"he said \"\"hi\"\"\"",EmpathLexer.STRING_LIT,"
he said \"hi\"");
  }

  public void testNumbers() {
    assertValidInput("1.3",EmpathLexer.REAL_LIT);
    assertValidInput("1",EmpathLexer.INT_LIT);
    int [] expectedTypes=new int[] {EmpathLexer.INT_LIT,EmpathLexer.CO
LON,EmpathLexer.INT_LIT};
  }

  public void testMinMax() {
    assertValidInput("@min", EmpathLexer.AT_MIN);
    assertValidInput("@MIN", EmpathLexer.AT_MIN);
    assertValidInput("@max", EmpathLexer.AT_MAX);
    assertValidInput("@MAX", EmpathLexer.AT_MAX);
    int [] expectedTypes={EmpathLexer.ID,EmpathLexer.AT_MAX};
    assertValidInput("hunger @MAX", expectedTypes);
  }

  /**
  public void testRange() {
    int [] expectedTypes={EmpathLexer.ID,EmpathLexer.DBLDOT};
    assertValidInput("a..", expectedTypes);
    expectedTypes=new int[] {EmpathLexer.INT_LIT,EmpathLexer.DBLDOT,Em
pathLexer.INT_LIT};
    assertValidInput("1..10", expectedTypes);
    expectedTypes=new int[] {EmpathLexer.LITERAL_range,EmpathLexer.LSQ
UARE,EmpathLexer.INT_LIT,EmpathLexer.DBLDOT,EmpathLexer.INT_LIT,Empath
Lexer.RSQUARE};
    assertValidInput("range[1 .. 10]", expectedTypes);
    assertValidInput("range[1..10]", expectedTypes);
  }*/

  /**
   * Test that the tokens are getting the line numbers correctly
   */
  public void testLineNumbers() {
    EmpathLexer lexer = getLexer("id1\nid2\nid3");
    Token token = null;
    try {
      token = lexer.nextToken();
      assertNotNull(token);
      assertEquals(1,token.getLine());
      token=lexer.nextToken();
      assertNotNull(token);
      assertEquals(2,token.getLine());
    } catch (TokenStreamException e) {
      fail(e.getMessage());
    }

  }

}
```

A.38 Java Test Source: EmpathParserTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.io.File;

import antlr.collections.AST;


/**
 * Tests for the Parser
 * @author William Mee (wjm2107)
 */
public class EmpathParserTest extends BaseParserTest {


  /**
   * Simple valid test
   *
   */
  public void testValid() {
//    AST ast = assertParsedOk("entity egg {state broken;}");
//    assertEquals(ast.getText(),"entity");
    assertProductionOk("entitySpec", "entity dog {}");
    assertProductionOk("entitySpec", "entity dog label \"Tommy\" {}");


    assertProductionOk("stateDecl", "state s;");
    assertProductionOk("stateDecl", "state init s;");
    assertProductionOk("stateDecl", "state s label \"S\", t;");
    assertProductionOk("stateDecl", "state s, init t label \"T\";");

    assertProductionOk("stateDecls", "");
    assertProductionOk("stateDecls", "state s;\n state t;");
    assertProductionOk("stateDecls", "state init s; state init t;");
// catch this in tree-walking

    assertProductionOk("transitionDef", "transition a to b if (c());")
```

```
;
    assertProductionOk("transitionDef", "transition * to b if (c()), d
 to e if (1) / g();");

    assertProductionOk("transitionDefs", "transition a to b if (c());
transition d to e if (1) / g();");

    assertProductionOk("stateDef", "pup { }");
    assertProductionOk("stateDef", "pup label \"Pup\" {state init s; s
tate t; transition * to b if (c()), d to e if (1) / g();}");
//    runProduction("tDef","x to y if (a())");
  }

  /**
   * Tests invalid input
   */
  public void testInvalid() {
    assertNotParsedOk("entity malformed {x}");
    assertProductionNotOk("entitySpec", "dog label \"Tommy\" {}");
    assertProductionNotOk("entitySpec", "entity {}");
    assertProductionNotOk("entitySpec", "entity dog label {}");
    assertProductionNotOk("entitySpec", "entity dog label Tommy {}");
    assertProductionNotOk("entitySpec", "entity dog label \"Tommy\" }"
);
    assertProductionNotOk("entitySpec", "entity dog label \"Tommy\" {{
");

    assertProductionNotOk("stateDecl", "state s");
    assertProductionNotOk("stateDecl", "state ;");
    assertProductionNotOk("stateDecl", "state init;");
    assertProductionNotOk("stateDecl", "state s, ;");
    assertProductionNotOk("stateDecl", "state s, init;");

    assertProductionNotOk("stateDecls", "state s, state, x, y;");

    assertProductionNotOk("transitionDef", "transition to b if (c());"
);
    assertProductionNotOk("transitionDef", "transition a to to if (c()
);");
    /*this should be caught in static semantic analyis
     * assertProductionNotOk("transitionDef", "transition a to b if (c
=1);");*/
    assertProductionNotOk("transitionDef", "transition a to b if (c())
 /;");

    assertProductionNotOk("transitionDefs", "transition a to b if (c()
), transition d to e if (1) / g();");

    assertProductionNotOk("stateDef", " { }");
    assertProductionNotOk("stateDef", "pup label { }");
    assertProductionNotOk("expression", "a=b=c");
    assertProductionNotOk("functionCall" , "f(a,)");
  }

  /**
   * Tests all the files in code/valid for parsability
   *
   */
  public void testAllValid() {
    assertTrue(TestConstants.DIR_PARSER_TEST_INPUT.isDirectory());
    for (String file:TestConstants.DIR_PARSER_TEST_INPUT.list()) {
      File codeFile=new File(TestConstants.DIR_PARSER_TEST_INPUT,file)
;
      if (codeFile.isFile()) {
        assertParsedOk(codeFile);
      }
    }
  }


  public void testLineNumbers() {
    AST ast = assertParsedOk(new File(TestConstants.DIR_PARSER_TEST_IN
PUT,"linenumtest.emp"));
    assertTrue(ast instanceof BaseAST);
    assertEquals(4,ast.getLine());
  }
```

```
}
```

A.39 Java Test Source: FuncSSATest

```
package edu.columbia.cs.coms4115.empath.antlr;

import antlr.SemanticException;

/**
 * Tests for Static Semantic Analysis of functions
 * @author Sharika
 *
 */
public class FuncSSATest extends BaseWalkerTest {


  public void testPosSSA() {
    try {
      EmpathTreeWalker walker = walkFileForSSA(TestConstants.SSAFUNC_T
EST1);
      // should not be any errors
    } catch (SemanticException e) {
      fail(e.getMessage());
    }
  }

  public void testPosSSAbullfrog() {
    try {
      EmpathTreeWalker walker = walkFileForSSA(TestConstants.CODE_GEN_
FROG);
      // should not be any errors
    } catch (SemanticException e) {
      fail(e.getMessage());
    }
  }

  public void testPosSSAGrad() {
    try {
      EmpathTreeWalker walker = walkFileForSSA(TestConstants.SSAFUNC_T
EST14);
```

```
      // should not be any errors
    } catch (SemanticException e) {
      fail(e.getMessage());
      System.err.println(e.getLine());
    }
  }

  public void testNegSSA1() {
    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST2);
      fail("could walk function with bad type assignment");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("Mismatch in variable"));
    }
  }

  public void testNegSSA2() {
    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST6);
      fail("could walk function with bad type assignment");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid condition for whil
e loop"));
    }
  }
  public void testFunctionReturnValues() {
    // check the return value of a trigger
    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST4);
      // should not be any errors
    } catch (SemanticException e) {
      fail(e.getMessage());
    }
    // check the return value of a trigger
    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST3);
      // should not be any errors
    } catch (SemanticException e) {
      fail(e.getMessage()+" at "+e.getLine());

    }

  }



  public void testIntegerMaths() {
    // check the return value of a trigger
    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST5);
      // should not be any errors
    } catch (SemanticException e) {
      e.printStackTrace();
      fail(e.getMessage()+" at "+e.getLine());
    }
  }

  public void testoutputstmt() {
    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST7);
      fail("could walk function with invalid output value");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid output"));
    }
  }

  public void testfunctioncallargs()
  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST8);
      fail("could walk function call with invalid args list");
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid argument list"));
    }
  }

  public void testfunctiondefined()
  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST10);
      fail("could walk function call with invalid args list");
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid argument list"));
    }
  }

  public void testargsinspecialfunc()
  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST11);
      fail("could walk function defn of onclocktick, etc. with args li
st");
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      assertTrue(e.getMessage().startsWith("Function cannot have argum
ent list"));
    }
  }

  public void testretspecialfunc()
  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST12);
      fail("could walk function with invalid return type (non-void for
 onexit, etc.)");
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid return type for fu
nction"));
    }
  }

  public void testtypeofspecial()
  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST13);
      fail("could walk trigger or event with name onexit, etc.");
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      assertTrue(e.getMessage().startsWith("Function cannot be a trigg
er or event"));
    }
  }
  public void testNegCharAssignInTrigger()  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST15);
      fail("could walk trigger that modifies characteristics ");
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      assertTrue(e.getMessage().startsWith("Not allowed to change char
acteristic values in a trigger function "));
    }
  }

  public void testPosCharAssignInTrigger()  {
    try
    {
      walkFileForSSA(TestConstants.SSAFUNC_TEST16);
      // should not be any errors
    }
    catch (SemanticException e)
    {
      // expected
      fail(e.getMessage());
    }
  }
}
```

A.40 Java Test Source: FunctionTester

```
package edu.columbia.cs.coms4115.empath.antlr;


public class FunctionTester extends BaseParserTest {


  /**
   * Run the "func" production on the given input
   * @param input input string
   */
  private void assertFunctionOk(String input) {
    assertProductionOk("func",input);
  }

  public void testNegative() {
    assertProductionNotOk("expression","tick hjgf");

  }

  /**
   * Test a simple function parse
   */
  public void testSimple() {
    assertFunctionOk("function string  thinkAboutLifeALittle() { }");
    assertFunctionOk("function string  label \"test\" thinkAboutLifeAL
ittle() {int a; }");
    assertFunctionOk("trigger label \"test\" thinkAboutLifeALittle() {
int a; }");
    assertFunctionOk("trigger thinkAboutLifeALittle() {int a; }");
    assertFunctionOk("function void bury() {a=10;}");
    //assertFunctionOk("trigger diesFromTooMuchSushi()");
    //assertFunctionOk("string label x thinkAboutLifeALittle()");
    //assertFunctionOk("input int stroke()");
    //assertFunctionOk("trigger diesFromTooMuchSushi()");
  }
  public void testStmts() {
    assertProductionOk("statement" , "hunger++; ");
    assertProductionOk("statement" , "hunger @max; ");
    assertProductionOk("statement" , "return 10; ");
    assertProductionOk("statement" , "a=10;");
    //assertProductionNotOk("statement" , "a; ");

  }

  public void testExpressions() {
    assertProductionOk("expression" , "a(b,10) ");
    //assertProductionOk("expression" , "a");

  }

  /**
   * Tests functions with an argument
   */
  public void xtestWithArguments() {
    assertFunctionOk("stroke(int speed) {}");
  }


  public void xtestWithBody() {
    assertFunctionOk("fatten() {bodyFat=bodyFat+10}");
  }

  public void testType() {
    assertProductionOk("decl","range[1 .. 10] happiness;");
    assertProductionOk("decl","range[1 ..10] happiness;");

  }

}
```

A.41 Java Test Source: LineNumberTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import antlr.SemanticException;

/**
 * Tests of the line number thrown by a Semantic Exception
 *
 * @author William Mee (wjm2107)
 */
public class LineNumberTest extends BaseWalkerTest {

  public void testSSAErrors() {

    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST2);
      fail("could walk invalid function");
    } catch (SemanticException e) {
      // expected
      assertEquals(12,e.getLine());
    }

    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST6);
      fail("could walk invalid function");
    } catch (SemanticException e) {
      // expected
      assertEquals(13,e.getLine());
    }

    try {
      walkFileForSSA(TestConstants.SSAFUNC_TEST9);
      fail("could walk invalid function");
    } catch (SemanticException e) {
      // expected
      assertEquals(6,e.getLine());
    }
    }

}
```

A.42 Java Test Source: StateTransTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import antlr.SemanticException;

/**
 * Tests for static semantic analysis of the state logic
 *
 * @author William Mee (wjm2107)
 */
public class StateTransTest extends BaseWalkerTest {
  public void testStarState()  {
    EmpathTreeWalker walker = walkFile(TestConstants.SSASTATE_TEST11);

  }

  public void testNeg1()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST1);
      fail("could walk state with re-declaration");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("State: "));
    }
  }
  public void testNeg2()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST2);
      fail("could walk state without declaration");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("State "));
    }
  }
  public void testNeg3()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST3);
      fail("could walk transition with bad condition");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid transition conditi
on: "));
    }
  }
  public void testNeg4()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST4);
      fail("could walk transition with bad condition");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("Undeclared function: "));
    }
  }
  public void testNeg5()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST5);
      fail("could walk state multiple inits");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("More than one initial stat
es: "));
    }
  }
  public void testNeg6()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST6);
      fail("could walk transition with bad condition");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("Invalid transition conditi
on: "));
    }
  }
  public void testPos7()  {
    try {
      EmpathTreeWalker walker = walkFileForSSA(TestConstants.SSASTATE_
TEST7);
      // should not be any errors
    } catch (SemanticException e) {
      fail(e.getMessage());
    }
  }
  public void testNeg8()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST8);
      fail("could walk state without declaration");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("State "));
    }
  }
  public void testNeg9()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST9);
      fail("could walk state without declaration");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("State: "));
    }
  }
  public void testNeg10()  {
    try {
      EmpathTreeWalker walker=walkFileForSSA(TestConstants.SSASTATE_TE
ST10);
      fail("could walk state without declaration");
      // should not be any errors
    } catch (SemanticException e) {
      // expected
      assertTrue(e.getMessage().startsWith("State: "));
    }
  }
}
```

A.43 Java Test Source: SymbolTableTest

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.util.Iterator;

/**
 * Tests for the symbol table
 *
 * @author William Mee (wjm2107)
 */
public class SymbolTableTest extends BaseWalkerTest {

  public void testEntries() {
    EmpathTreeWalker walker = walkFile(TestConstants.FROG);
    SymbolTable table = walker.entitySymTab;
    assertNotNull(table);
    assertTrue(table.symTab.size()>0);
    StateEntry adultFrog=(StateEntry)table.getEntry("AdultFrog");
    assertNotNull(adultFrog);
    SymbolTable childTable=adultFrog.childSymTab;
    assertNotNull(childTable);
    assertTrue(childTable.symTab.size()>0);
    StateEntry tadpoleFrog=(StateEntry)table.getEntry("Tadpole");
    assertNotNull(tadpoleFrog);
    childTable=tadpoleFrog.childSymTab;
    assertNotNull(childTable);
    assertTrue(childTable.symTab.size()>0);
  }


  private void recurseIntoTable(SymbolTable table) {
    Iterator<String> stateIter = table.stateIterator();
    while (stateIter.hasNext()) {
      String stateLabel = stateIter.next();
      assertNotNull(stateLabel);
      StateEntry se = (StateEntry)table.getEntry(stateLabel);
      assertNotNull("child for "+stateLabel+" does not exist",se);
```

```
        if (se.getChildSymTab()!=null) {
          recurseIntoTable(se.getChildSymTab());
        }
      }
    }

    /**
     * Tests the state iterator
     */
    public void testStateList() {
      EmpathTreeWalker walker = walkFile(TestConstants.FROG);
      SymbolTable table = walker.entitySymTab;
      assertNotNull(table);
      recurseIntoTable(table);
    }



  }
```

A.44 Java Test Source: TestConstants

```
package edu.columbia.cs.coms4115.empath.antlr;

import java.io.File;

public class TestConstants {

  // parser tests
  public static File DIR_PARSER_TEST_INPUT=new File("code/parser");
  // walker tests
  public static File DIR_WALKER_TEST_INPUT=new File("code/walker");
  // codegen tests
  public static File DIR_CODEGEN_TEST_INPUT=new File("code/codegen");

  public static File FROG=new File(DIR_PARSER_TEST_INPUT,"frog.emp");

  public static File CODE_GEN_FROG=new File(DIR_CODEGEN_TEST_INPUT,"bu
llfrog.emp");
  public static File CODE_GEN_F1=new File(DIR_CODEGEN_TEST_INPUT,"f1.e
mp");
  public static File CODE_GEN_F2=new File(DIR_CODEGEN_TEST_INPUT,"f2.e
mp");
  public static File CODE_GEN_F3=new File(DIR_CODEGEN_TEST_INPUT,"f3.e
mp");
  public static File CODE_GEN_F4=new File(DIR_CODEGEN_TEST_INPUT,"f4.e
mp");
  public static File CODE_GEN_F5=new File(DIR_CODEGEN_TEST_INPUT,"f5.e
mp");
  public static File CODE_GEN_F6=new File(DIR_CODEGEN_TEST_INPUT,"f6.e
mp");
  public static File CODE_GEN_F7=new File(DIR_CODEGEN_TEST_INPUT,"f7.e
mp");
  public static File CODE_GEN_F8=new File(DIR_CODEGEN_TEST_INPUT,"f8.e
mp");
  public static File CODE_GEN_F9=new File(DIR_CODEGEN_TEST_INPUT,"f9.e
mp");
  public static File CODE_GEN_F10=new File(DIR_CODEGEN_TEST_INPUT,"f10
.emp");

  public static File CROCODILE=new File(DIR_PARSER_TEST_INPUT,"crocodi
le.emp");

  public static File SSAFUNC_TEST1=new File(DIR_WALKER_TEST_INPUT,"ssa
functest1.emp");
  public static File SSAFUNC_TEST2=new File(DIR_WALKER_TEST_INPUT,"ssa
functest2.emp");
  public static File SSAFUNC_TEST3=new File(DIR_WALKER_TEST_INPUT,"ssa
functest3.emp");
  public static File SSAFUNC_TEST4=new File(DIR_WALKER_TEST_INPUT,"ssa
functest4.emp");
  public static File SSAFUNC_TEST5=new File(DIR_WALKER_TEST_INPUT,"ssa
functest5.emp");
  public static File SSAFUNC_TEST6=new File(DIR_WALKER_TEST_INPUT,"ssa
functest6.emp");
  public static File SSAFUNC_TEST7=new File(DIR_WALKER_TEST_INPUT,"ssa
functest7.emp");
  public static File SSAFUNC_TEST8=new File(DIR_WALKER_TEST_INPUT,"ssa
functest8.emp");
  public static File SSAFUNC_TEST9=new File(DIR_WALKER_TEST_INPUT,"ssa
functest9.emp");
  public static File SSAFUNC_TEST10=new File(DIR_WALKER_TEST_INPUT,"ss
afunctest10.emp");
  public static File SSAFUNC_TEST11=new File(DIR_WALKER_TEST_INPUT,"ss
afunctest11.emp");
  public static File SSAFUNC_TEST12=new File(DIR_WALKER_TEST_INPUT,"ss
afunctest12.emp");
  public static File SSAFUNC_TEST13=new File(DIR_WALKER_TEST_INPUT,"ss
afunctest13.emp");
  public static File SSAFUNC_TEST14=new File("code/grad_student.emp");

  public static File SSAFUNC_TEST15=new File(DIR_WALKER_TEST_INPUT,"ss
afunctest15.emp");
  public static File SSAFUNC_TEST16=new File(DIR_WALKER_TEST_INPUT,"ss
afunctest16.emp");
```

```
  public static File SSASTATE_TEST1=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest1.emp");
  public static File SSASTATE_TEST2=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest2.emp");
  public static File SSASTATE_TEST3=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest3.emp");
  public static File SSASTATE_TEST4=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest4.emp");
  public static File SSASTATE_TEST5=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest5.emp");
  public static File SSASTATE_TEST6=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest6.emp");
  public static File SSASTATE_TEST7=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest7.emp");
  public static File SSASTATE_TEST8=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest8.emp");
  public static File SSASTATE_TEST9=new File(DIR_WALKER_TEST_INPUT,"ss
astatetest9.emp");
  public static File SSASTATE_TEST10=new File(DIR_WALKER_TEST_INPUT,"s
sastatetest10.emp");
  public static File SSASTATE_TEST11=new File(DIR_WALKER_TEST_INPUT,"s
sastatetest11.emp");
}
```