

# **OGEL:**

## A LEGO Mindstorm Robot Controller Language

Matt Kalish - Michael Ching - Gerardo Flores - Charles O'Donnell

13 May 2003

# Table of Contents

<b>1 An Introduction to OGEL</b>	<b>6</b>
1.1 Background	6
1.2 Related Work	6
1.3 Goals of OGEL	6
1.3.1 Object-Oriented	6
1.3.2 Hierarchical Structure	7
1.3.3 Scalable	7
1.3.4 Flexible	7
1.3.5 Rapid Development	7
1.3.6 Event Driven and Concurrent Processing	8
1.3.7 Legacy Support	8
<b>2 Tutorial</b>	<b>9</b>
2.1 Preliminary Information	9
2.2 A First Example	9
2.3 Compiling and Running OGEL Source Code	12
<b>3 Language Reference Manual</b>	<b>13</b>
3.1 Grammar Notation	13
3.2 Lexical Rules	13
3.2.1 Comments	13
3.2.2 Whitespace	13
3.2.3 Tokens	13
3.2.4 Identifiers	14
3.2.5 Keywords	14

3.2.6 Line Terminators	14
3.2.7 Embedded NQC Blocks	14
3.2.8 Separators	14
3.3 Program Structure	15
3.3.1 Header	15
3.3.1.1 Includes	15
3.3.1.2 Defines	15
3.3.1.3 RCX Declaration	16
3.3.1.4 Output and Input Portmappings	16
3.3.1.5 Message Declaration	16
3.3.2 Tasks	16
3.3.2.1 Standard Code and Function Calls	17
3.3.2.2 When Directives	18
3.3.2.3 Messaging	18
3.3.3 Functions	19
3.3.4 Main()	20
<b>4 Project Plan</b>	<b>21</b>
4.1 Team Responsibilities	21
4.2 Project Timeline	21
4.3 Software Development Environment	22
4.4 Project Log	22
<b>5 Architectural Design</b>	<b>24</b>
5.1 Architecture	24
5.2 Work Division	26
<b>6 Testing Plan</b>	<b>27</b>
6.1 Goals	27

6.2 Hypothesis	27
6.3 Methods	27
6.3.1 Phase I – Isolated Module Testing	27
6.3.2 Phase II – Integration Testing	28
6.3.3 Phase III – Final Testing	28
6.4 Tools	28
6.5 Implementation	28
6.5.1 Phase I – Isolated Module Testing	29
6.5.2 Phase II – Integration Testing	29
6.5.3 Phase III – Final Testing	29
6.6 Test Case Compilation Results	30
6.5.1 Test Case 1	31
6.5.2 Test Case 2	31
6.7 Division of Work	33
<b>7 Lessons Learned</b>	<b>34</b>
7.1 Charles O'Donnell	34
7.2 Matt Kalish	34
7.3 Michael Ching	34
7.4 Gerardo Flores	35
<b>Appendices</b>	
<b>A Complete OGEL Grammar</b>	<b>36</b>
<b>B Code Style Conventions</b>	<b>40</b>
B.1 Introduction	40

B.2 General Principles	40
B.3 Tabs	40
B.4 Documentation Comments	40
C Complete Code Listing	41

# Chapter 1

## An Introduction to OGEL

The OGEL language is intended for the development of programs to be executed on Lego Mindstorm robots. A single OGEL program, which uses a simple syntax very similar to that of Java, can easily and efficiently generate native code for one or more Lego Mindstorm microprocessors at the same time. The purpose of OGEL will therefore be to offer programmers a more powerful alternative to the current development tools and interfaces available for coding these Mindstorm robots.

### 1.1 Background

Lego Mindstorm is a product which allows users to turn Lego building blocks into responsive, autonomous robots, able to interact with their environment via predefined and/or computer assisted responses. The basic components in building Lego Mindstorm robots include the RCX 1.0 controller, miniature motors, sensors, and regular Lego blocks. The RCX 1.0 runs on firmware that is uploaded along with the user program through an infrared tower to the infrared interface on the controller.

### 1.2 Related Work

Several environments currently exist which facilitate the programming of Lego Mindstorm robots. Current efforts range from drag and drop interfaces targeted at novice programmers to the Not Quite C (NQC) language targeted at more experienced C programmers. NQC contains a feature-rich library for interfacing with the RCX controller. This provides a well-developed foundation to use as a means of communicating with the RCX.

### 1.3 Goals of OGEL

OGEL is designed to merge the best traits of all currently available technology, creating a language that is both easy and intuitive to use, while providing superior power and control. It is designed to help developers rapidly design, prototype and simulate their robotic entities. OGEL will ultimately compile down to NQC code.

### **1.3.1 Object Oriented**

An object oriented language allows for a wide variety of enhancements over the current programming architecture of NQC. By mimicking traits of an object-oriented language, we sizably extend both the ease of programming and the power of drag-and-drop and NQC development solutions. The object-oriented nature of OGEL will serve as a layer of abstraction that simplifies the development cycle for programmers. They will be able to better focus on the task at hand and not be concerned with the physical constraints of the hardware nor the exact location and intent of the underlying code. By separating the ideas of control and physical blocks, one can easily reuse the components in multiple situations.

### **1.3.2 Hierarchical Architecture**

OGEL creates a hierarchical development environment that facilitates the user's understanding and organization of the program. Robots have similar procedures: transportation movement, non-transportation movements (e.g. arm swings), and physical sensor detection. OGEL recognizes these similarities in programming development and supports a variety of keywords that organize these similar robot requirements into distinct classes. Thus, a programmer will be able to quickly analyze his/her program and verify if all necessary parts have been included in the robot just by looking to see if a movement class, arm-swinging class or sensor detection class have been created.

### **1.3.3 Scalable**

OGEL will also take advantage of the inherent scalable nature of our hierarchical object-oriented approach. Employing the same building block based approach, large-scale programming projects can be easily designed and implemented using preexisting code and frameworks. By taking the details of the hardware and hiding it behind the abstraction of OGEL, large-scale projects can more easily be segmented as well.

### **1.3.4 Flexible**

Although our language is designed for the RCX 1.0 controller, it would be ignorant of us to ignore possible (and probable) advances in the actual hardware units. Any system-specific traits will be user configurable. The language should also be general enough to be able to describe any goal a programmer wants a robot to achieve. This can be done by refraining from overly controller-specific references, similar to the outdated 'register' keyword in C.

### **1.3.5 Rapid Development**

OGEL provides several capabilities that facilitate the rapid development of user code, such as modular and reusable code. The increased amount of development in the arena of Lego Mindstorms for research and recreational purposes, leads to large amounts of

code being created every day. In addition, the similarity in the movements and reactions of robots to stimuli beg the question, “Why must programmers re-create the wheel?” OGEL’s design recognizes that it is geared toward a programming-savvy community, and a major goal of the language is to allow a task to be described and programmed with relative ease. High level abstractions and simple calls to standard library routines aid in this quest.

### **1.3.6 Event Driven and Concurrent Processing**

OGEL’s second goal is to be able to leverage the development of user-defined movements and combine them to create better event-based programming. One popular area in the robotics community is the creation of a soccer team of robots that will compete against a team of international superstars by 2050. To achieve this goal, event based programming must become more elaborate, robust and specific. OGEL does not attempt to completely bridge this gap in development efforts, but recognizes it as an issue and addresses it accordingly.

A physical system can be seen as nothing more than stimulus and response. Ex: A robot touches a wall, it moves the other direction. The RCX is also inherently multitasking and can have multiple control flows processing at the same time. Events and message passing internally between these tasks and externally with the outside world would ease development and present a clean model of Event-Response control to the programmer. In addition, this ability to program simultaneous threads of logic is an invaluable resource to programmers as they will inevitably desire their robots to perform several tasks at a time. Although more advanced systems have rudimentary notion of 'events' (RCX 2.0) RCX 1.0 does not. A generalized notion of events and messaging is built into OGEL.

### **1.3.7 Legacy Support**

There exist countless NQC programs, which are already written and yet could benefit from the features introduced in OGEL. There also may be many opportunities where a specific algorithm has already been worked out in NQC and it would be unreasonable to require it to be rewritten. Therefore, OGEL will be able to understand blocks of NQC source code, and allow for NQC-style library calls to be handled in the same manner as well.



# Chapter 2

## Tutorial of OGEL

### 2.1 Preliminary Information

OGEL, while being very powerful and versatile, was designed with ease of implementation in mind. This tutorial will make it abundantly obvious how time-efficient and programmer-friendly it is to code in OGEL.

### 2.2 A First Example

Imagine a scenario where you would like to program two separate Lego Mindstorm robots to each perform a task. Each robot is made up of two RCX 1.0 processors – one which controls the upper body, and one which controls the lower body. The programmer can accomplish this via a single OGEL program, something that is currently impossible in other languages.

Examine the following code:

```
// RCX Declarations.  
rcx robotUpper {SENSOR_TOUCH touch, out arm}  
rcx robotLower {SENSOR_TOUCH bumper, out leftLeg, out rightLeg}  
  
// Message Declarations.  
message alert1;  
message alert2;
```

This code is essentially known as the header section of an OGEL program. Its purpose is to declare the RCX units that are to be used and the messages that they are eligible to pass and receive while executing. In the RCX declarations, one must describe the inputs and outputs controlled by each RCX unit – ie, robotUpper has an arm that it controls, and it receives input from a sensor.

Now the programmer is free to implement whatever tasks will be necessary to achieve the program's ultimate goal. Tasks can be utilized by any or all of the RCX units declared in the program's header. Functions can also be used to accomplish key parts of a task. Examine the following code:

```

// A task, makes the robot move forward until a bump is encountered, then stops it.
task runUntilBump(in bumper)(out leftLeg, out rightLeg)
{
    // Block of native NQC code.
    <%
    setPower (leftLeg, OUT_FULLL);
    setPower (rightLeg, OUT_FULLL);
    %>

    // When directive, stops run when the sensor gets activated (a bump occurs).
    when (bumpSensor == 1) do stopMovement();
}

// A function, stops the current robot's motion.
void stopMovement()(out leftLeg, out rightLeg)
{
    nqc.Off (leftLeg);
    nqc.Off (rightLeg);
}

```

The task “runUntilBump” makes the robot move forward until a bump is felt by the sensor, then the function stopMovement() is called and the motors are shut off. Important things to notice here are

- 1) All native NQC methods are available to the programmer and must simply be prefixed by “nqc.” as nqc.Off is. NQC code that you simply want to include must be escaped by a <% ... %>.
- 2) The when directive controls the motion of the robot – it keeps the motors running under full power until the sensor changes. When it does, control flow moves to the function.
- 3) A function's port mappings must be a subset of the calling task's port mappings. Other integer parameters may be passed to a function and they would be specified in the first set of parenthesis in the declaration.
- 4) OGEL programs can contain as many functions and tasks as desired, allowing of course for inevitable memory constraints.

Each OGEL program will contain exactly one task called main(), which serves to assign tasks and functions to individual RCX units as needed. These tasks are automatically activated at the start of execution. Examine the following code:

```

// The main task.
task main()
{
    // RCX instantiations.
    rcx robotUpper TimTorso;
    rcx robotLower TimLegs;
    rcx robotUpper JerryTorso;
}

```

```

    rcx robotLower JerryLegs;

    // Add task calls.
    TimLegs.addTask(runUntilBump);
    JerryLegs.addTask(runUntilBump);
}

```

Note that the four initially declared RCX units are now instantiated. It is next required that the RCX's are assigned the tasks that they will be responsible for running. Note that it is important that the task has the port mappings that are supported by the RCX – otherwise the application will not make sense semantically, although it looks syntactically correct.

So, the first complete OGEL program is as follows:

```

// RCX Declarations.
rcx robotOneUpper {SENSOR_TOUCH touch, out arm}
rcx robotOneLower {SENSOR_TOUCH bumper, out leftLeg, out rightLeg}

// Message Declarations.
message alert1;
message alert2;

// A task, makes the robot move forward until a bump is encountered, then stops it.
task runUntilBump (in bumper)(out leftLeg, out rightLeg)
{
    // Block of native NQC code.
    <%
    setPower (leftLeg, OUT_FULLL);
    setPower (rightLeg, OUT_FULLL);
    %>

    // When directive, stops run when the sensor gets activated (a bump occurs).
    when (bumpSensor == 1) do stopMovement();
}

// A function, stops the current robot's motion.
void stopMovement()(out leftLeg, out rightLeg)
{
    nqc.Off (leftLeg);
    nqc.Off (rightLeg);
}

// The main task.
task main()
{
    // RCX instantiations

```

```

rcx robotUpper TimTorso;
rcx robotLower TimLegs;
rcx robotUpper JerryTorso;
rcx robotLower JerryLegs;

// Add task calls.
TimLegs.addTask(runUntilBump);
JerryLegs.addTask(runUntilBump);
}

```

When executed properly, it will make two separate robots power their motors and begin walking until a bump is encountered, at which time they stop. This could easily be a subset of a larger, more useful OGEL program, such as a “showdown” program where two robots begin in the center of a walled room, and they walk until the wall is hit, then turn and fire some projectile at the opposing robot. As you can see from the sheer size and capability of the program above, OGEL provides a very powerful tool for programming Lego Mindstorm robots..

## 2.3 Compiling and Running OGEL Source Code

Source code files will be identified with a “.ogel” extension. A few requirements are necessary for the NQC compiler to function. First and foremost, you must be able to invoke antlr.Tool from the current compilation space and be able to run a bash script. The bash script is used as the driver for the whole process. After a program is written in OGEL, compilation is initiated with the command “ogelc filename.ogel”, where filename is the name of the source file. Assuming that no compile-time errors occur, the compile will produce a number of output files, with the extension “.nqc”. The number of files generated by the compiler will correspond to the number of RCX units instantiated in the main task.

To ultimately run the code in a real-life LEGO Mindstorm robot, however, each of these target files will need to be compiled using an existing NQC compiler. A good NQC website, containing links to NQC compilers, can be found at: <http://www.baumfamily.org/nqc/index.html>. Once the NQC files are compiled, the code must be uploaded onto the RCX units. Check LEGO Mindstorm documentation for details on this stage. Upon uploading the programs to each RCX involved in the application, everything is prepared to run the OGEL program on the robot(s).

# Chapter 3

## Language Reference Manual

### 3.1 Grammar Notation

The grammatical symbols are introduced in stages throughout this document, presenting only the symbols relevant to each subsection. Italicized symbols represent non-terminals, while those in quotes are terminal. Concatenated symbols imply concatenated tokens, while the '|' character implies a choice. Other regular expression notation used include the Kleene \*, Kleene +, and ? optional character. Parentheses group symbols and apply precedence. Finally, the symbols *letter* and *digit* are defined to represent characters in the Roman alphabet plus underscores and Arabic numerals.

### 3.2 Lexical Rules

#### 3.2.1 Comments

OGEL supports two types of comments. The first form begins with `/*` and ends with `*/`. This form of comment is allowed to cover multiple lines, but cannot be nested. The second type of comment begins with `//` and ends at a new line. Comments are ignored by the compiler and allow the programmer to document the source code.

#### 3.2.2 Whitespace

Whitespace is defined as the ASCII space, horizontal tab and form feed characters, as well as line terminators and comments. As long as tokens are distinguishable and separated by at least one whitespace, the addition or subtraction of whitespace has no effect on the meaning of the program.

#### 3.2.3 Tokens

There are several types of tokens: identifiers, keywords, constants, separators, and embedded NQC blocks. Whitespace is ignored except as a token separator. Whitespace is sometimes required to separate adjacent tokens that might otherwise be combined into one token (i.e. identifiers, keywords and constants). Token formation is greedy: the input

is searched for the longest string of characters that would constitute a token. Tokens are comprised of ASCII characters only.

### 3.2.4 Identifiers

Identifiers are used for variable, task, RCX, and function names. The first character must be an upper or lower case letter or the underscore character. The remaining part of the identifier can be a combination of letter numbers and an underscore, but may not be separated by whitespace.

### 3.2.5 Keywords

OGEL reserves certain identifiers from use by a programmer. These reserved words are all keywords and may not be used as identifiers. A list is below:

bool	in	return	throw
break	int	task	catch
const	message	time	
do	out	true	
else	pass	void	
false	rcx	when	
if	receive	while	

### 3.2.6 Line Terminators

Input characters are divided into lines by line terminators. Lines are terminated by the ASCII characters carriage return or linefeed. The carriage return linefeed combination is counted as one terminator, not two.

### 3.2.7 Embedded NQC blocks

Embedded NQC blocks allow the programmer to reuse working code and insert it accordingly into the new program. Embedded NQC blocks begin with the character sequence `<%` and end with `%>`. The compiler assumes that the embedded NQC code is fully functional and will not issue any errors or warnings. The embedded code is not allowed to contain the string `'%>'`, otherwise it would effectively prematurely end the block of embedded code.

### 3.2.8 Separators

The following ASCII characters are separators:

{ } [ ] ; : . ,

## 3.3 Program Structure

An OGEL robot control *program* consists of one *source* file and any number of *library* files which all adhere to the following format: At the top of the file there is a header block, followed by any number of tasks or functions in any order. A source file will contain *one* main function located after the header, while library files cannot contain a main function. The source file can *include* other library files so that their header definition, tasks, and functions are available for use. Libraries serve the purpose to group high-level goals, robot type information, or any other broad categories.

Programs describe goals which encompass the use of multiple robots and RCX units, and are automatically compiled down to many Not Quite C (NQC) source files, one for each RCX unit involved. These NQC source files are guaranteed to be operational (except for embedded NQC blocks).

*file* → *includeDecls rcxDecls messageDecls functionList  
taskDeclaratonList mainDeclaration*

### 3.3.1 Header

A header section is included in every OGEL source and library file. The header's primary purpose is to organize the naming and defining of multiple RCXs, and to define messages which are used to communicate between RCX units. This facilitates the ability to extend control to multiple RCXs and allows for different RCX input and output port configurations. Source files can use any number of includes to load library files, while library files are prohibited from using such a keyword.

#### 3.3.1.1 Includes

A source file may include any number of library files. The files to be included with a given source file will be declaring “#include filename”, where filename is to have a .lib extension.

#### 3.3.1.2 RCX Declaration

The RCX declaration is the next part of the header and it associates an RCX type with a name. A header file may have multiple RCX definitions, but each RCX must have a unique name throughout all source and library files. Following the RCX declaration is a code block encapsulated by the separators: '{' and '}' which is where the Output and Input Port mappings are made. An RCX must declare at least one port.

*rcxDecls* → *(rcxDeclaration)\**  
*rcxDeclaration* → *“rcx” rcxIdentifier “{(inputPortMapping |  
outputPortMapping)+ “}”*  
*rcxIdentifier* → *IDENTIFIER*

### 3.3.1.3 Output and Input Port mappings

Output and Input Port mappings are used to associate a name with each unique port. The RCX supports up to a maximum of three output and three input ports, limiting the number of ports the user can make in the RCX declaration block. An output port mapping is created by listing an output port= '\$A','\$B' or '\$C' with a unique name within the current RCX declaration block. Similarly, an input port mapping is created by listing an Input Port= '\$1','\$2' or '\$3' with a unique name.

<i>outputPortMapping</i>	→	<i>outPortMap</i> “=” ( <i>OUTA</i>   <i>OUTB</i>   <i>OUTC</i> ) “;”
<i>inputPortMapping</i>	→	<i>inPortMap</i> “=” ( <i>IN1</i>   <i>IN2</i>   <i>IN3</i> ) “;”
<i>outPortMap</i>	→	IDENTIFIER
<i>inPortMap</i>	→	IDENTIFIER

### 3.3.1.4 Message Declaration

A message declaration gives a name for a message that will be accessible by all RCXs. The message declaration is created by using the 'message' keyword coupled with a unique message name. All RCXs defined within the header have access to the message.

<i>messageDecls</i>	→	( <i>messageDeclaration</i> )*
<i>messageDeclaration</i>	→	“message” <i>messageIdentifier</i> “;”
<i>messageIdentifier</i>	→	IDENTIFIER

### 3.3.2 Tasks

A task declaration defines the most high-level objectives to be performed by a single RCX unit. These objectives can be accomplished through OGEL's event-driven features or through more conventional signal checking and function calling. A task is defined by the keyword **task** followed by the task's identifier and a parenthesized list of portmaps. Each portmap within the list is specified to be either of type **in** or **out**. There can be anywhere from zero to three specified in or out portmaps. A task code block is encapsulated by the separators '{' and '}'. The block can begin by the declaration of all **when** directives, followed by the declaration of all local variables. There can also be any combination of standard code, function calls, embedded NQC code, or **pass/receive** calls. Embedded NQC calls, though self-sufficient, may also rely on the globally declared port mappings of an rcx and not have to use the low-level port mappings set up by default in NQC.

<i>taskDeclarationList</i>	→	( <i>task</i> )*
<i>task</i>	→	<i>taskDeclaration</i> “{“ <i>taskBody</i> “}”
<i>taskDeclaration</i>	→	“task” <i>taskIdentifier</i> <i>inOutList</i>
<i>taskBody</i>	→	( <i>taskBodyStatements</i> )*
<i>taskBodyStatements</i>	→	<i>whenStatement</i>   <i>standardStatement</i>   <i>NQCBLOCK</i>



*taskIdentifier* → *IDENTIFIER*

### 3.3.2.1 Standard Code and Function Calls

Source code placed within task blocks, after *when* directives, is sequentially executed, beginning as soon as a task is activated. Lines may include any type of *Statements*, such as variable assignment incorporated with expression evaluation, which are terminated by the ';' separator. *Statements* also include **if** and **while** block statements, which can be used to check an in port mapping against a certain value once. Functions can be called simply by the function identifier, followed by a parenthesized list of arguments, comma separated. The port mappings for the task are automatically matched with those of the function definition. The port mappings for the function must be a subset of the port mapping for the task which must invoke it.

*statementList* → (*standardStatement*)\*  
*standardStatement* → *whileBlock*  
 | "do" "{ (*standardStatement*)\* "}" "while" "{ (*whileCondition* "}"  
 | "break" " ;"  
 | "if" "{ (*ifCondition* "}" "{ (*standardStatement*)\* "}"  
*elseBlock*  
 | "return" " ;"  
 | *functionCall* " ;"  
 | *variableDeclaration*  
 | *variableAssignment*  
 | *variableDeclaredAssignment*  
 | *messageStatement*  
 | *nqcFunctionCall*  
*elseBlock* → ("else" "{ *statementList* "}")?  
*whileBlock* → "while" "{ (*whileCondition* "}" "{ *whileStatements* "}"  
*whileStatements* → (*standardStatement*)\*  
*whileCondition* → *condition*  
*ifCondition* → *condition*  
*condition* → *conditionStatement*  
 | (*PARALLEL* | *NOT* | *AND*) *conditionStatement*\*  
 | ("true" | "false")  
*conditionStatement* → *expression* (*LESSTHAN* | *GREATERTHAN* | *LEQUAL* | *GEQUAL* | *CEQUAL* | *NEQUAL*)  
*expression* → *sumExpr*  
*sumExpr* → *prodExpr* ((*PLUS* | *DASH*) *prodExpr*)\*  
*prodExpr* → *powExpr* ((*STAR* | *SLASH* | *MOD*) *powExpr*)\*  
*powExpr* → *atom* (*POW* *atom*)?  
*atom* → (*NUMBER* | *IDENTIFIER*)

<i>variableDeclaredAssignment</i>	→	<i>variableIdentifier</i> ( <i>EQUALS</i>   <i>PLUSEQ</i>   <i>MINUSEQ</i>   <i>TIMESEQ</i>   <i>DIVIDEEQ</i> ) <i>rValue</i> “;”
<i>variableAssignment</i>	→	<i>variableType</i> <i>variableIdentifier</i> “=” <i>rValue</i> “;”
<i>variableDeclaration</i>	→	<i>variableType</i> <i>variableIdentifier</i> “;”
<i>variableType</i>	→	("int"   "const"   "time"   "bool")
<i>variableIdentifier</i>	→	<i>IDENTIFIER</i>
<i>rValue</i>	→	( <i>NUMBER</i>   <i>IDENTIFIER</i>   “true”   “false”)

### 3.3.2.2 When Directives

When directives serve as a monitor for sensor (*in*) data and will call a given function when the 'when' expression evaluates to true. (This operates differently than unary, inline if-tests of in port maps in standard code.) The format is **when** followed by a boolean expression equating a in port map to a constant or variable, followed by a **do** *FunctionIdentifier* where the function identifier is followed by only a parenthesized list of arguments. The port mappings are again matched from the task. When directives operate concurrent to the task, and will call the specified function as long as the expression evaluates to true (i.e. the function will be called multiple times if the expression remains true). In the event of multiple 'when' declarations evaluating to true on a given RCX unit, the order the functions are called in is undefined.

<i>whenStatement</i>	→	“when” <i>whenCondition</i> <i>whenStatementDo</i> “;”
<i>whenCondition</i>	→	“(“ <i>inPortMap</i> ( <i>LESSTHAN</i>   <i>GREATERTHAN</i>   <i>LEQUAL</i>   <i>GEQUAL</i>   <i>CEQUAL</i>   <i>NEQUAL</i> ) ( <i>NUMBER</i>   <i>variableIdentifier</i> ) “)”
<i>whenStatementDo</i>	→	“do” <i>functionCall</i>

### 3.3.2.3 Messaging

OGEL offers a superior method of message passing compared to NQC because it creates a formal communication standard that facilitates increased notification of events between multiple RCXs. OGEL's messaging architecture relies upon the message names created in the header file and the **pass** and **receive** keywords.

<i>messageStatement</i>	→	( <i>passMessage</i>   <i>recieveMessage</i> )
-------------------------	---	--

Pass allows a RCX to express that a certain event has occurred. The construct is to use the pass keyword in addition to the name of the message that is to be sent.

<i>passMessage</i>	→	“pass” <i>messageIdentifier</i> “;”
--------------------	---	-------------------------------------

Receive allows an RCX to search for a particular message from another RCX. The construct is to use the receive keyword with the name of the message, followed by a code block encapsulated by the separators: '{' and '}' that contains the RCXs response once the message has been received. The receive is blocking until the specific identifier has been

received.

*receiveMessage* → “receive” *messageIdentifier* “;”

### 3.3.3 Functions

A function is a block of code called from within a task or other function, which performs some activity. A function name can be named any legal identifier. Its declaration takes both int type parameters and in/out type parameters, which can then be used within the body of the function. Functions do not have the capability to return any values, and thus are defined using the void keyword.

By their nature, a function’s primary utility is performing a physical action, such as making a sound or powering a motor in a certain direction. In other words, functions generally perform the lower-level actions required by a task or larger function. It is important to note that functions can only be called by tasks or other functions that have a matching in/out type parameter list. This is because when making a function call, the only parameters required to be explicitly stated will be those of type int. The in/out type variables will be implicitly inherited as parameters from the calling function or task.

In OGEL, we elected to do away with NQC’s concept of subroutines, instead choosing to use only functions in their place. We felt that this design kept it much simpler for the coder, as the differences between functions and subroutines in NQC are quite minute. Furthermore, there were so many restrictions placed on the use of subroutines in NQC that we found them to be too burdensome to include in OGEL. However, we reserve the right to use subroutines in the compilation from OGEL source to NQC.

<i>functionList</i>	→	<i>(function)*</i>
<i>function</i>	→	<i>functionDeclaration</i> “{“ <i>statementList</i> “}”
<i>functionDeclaration</i>	→	“void” <i>functionIdentifier</i> <i>parameterList</i> <i>inOutList</i>
<i>functionCall</i>	→	<i>functionIdentifier</i> <i>functionParamsPasng</i>
<i>functionParamsPasng</i>	→	“(“ ( <i>NUMBER</i>   <i>IDENTIFIER</i> ) ( “;” ( <i>NUMBER</i>   <i>IDENTIFIER</i> ) ) * ) ? “)”
<i>functionIdentifier</i>	→	<i>IDENTIFIER</i>
<i>parameterList</i>	→	“(“ ( <i>arg</i> (“;” <i>arg</i> ) * ) ? “)”
<i>arg</i>	→	(“int”   “const”) <i>IDENTIFIER</i>

### 3.3.4 main()

Every OGEL program starts with the execution of the main task, and it is therefore mandatory. Within the main task, previously defined RCXs and functions (whether within the file or in a separate header file) may be utilized. Its purpose is to associate RCXs with tasks in our event driven model and accomplish all the necessary type checking between disparate files in a directory. Central to the restructuring of the 'main' task is the fact that this new structure is scalable in that several control tasks may be

associated with individual RCXs within one file. These tasks may therefore be bound to different RCXs and make use of their modularity without having to recreate the code blocks in separate files. To instantiate an RCX, one would simply state the RCX id followed by a valid identifier.

RCX id identifier;

The "addtask(taskList)" method would then be used to associate an RCX with a task.

RCX\_identifier.addTask(task\_identifier);

Unlike traditional Not Quite C syntax, the calling of functions and listing of statements within main is not allowed as there is no direct link between the running main task and a particular RCX. Control structures such as loops would, therefore, have to be included in the logic of other tasks, which are called from main and would reside on an RCX. Main functions act primarily as an "init" task in which a configuration for a robot, or set of robots, is initiated and the inherent multi-tasking nature of the Lego RCXs may be taken advantage of.

<i>mainDeclaration</i>	→	“task” “main” (“( “)”) “{ “ <i>mainBody</i> “}”
<i>mainBody</i>	→	<i>rcxInstantiations rcxAddTaskList</i>
<i>rcxInstantiations</i>	→	( <i>rcxInstantiation</i> )+
<i>rcxInstantiation</i>	→	“rcx” <i>rcxClass rcxInstanceIdentifier</i> “;”
<i>rcxClass</i>	→	<i>IDENTIFIER</i>
<i>rcxInstanceIdentifier</i>	→	<i>IDENTIFIER</i>
<i>rcxAddTaskList</i>	→	( <i>rcxAddTaskCall</i> )+
<i>rcxAddTaskCall</i>	→	<i>rcxInstanceIdentifier</i> “.” “addTask” (“( “ <i>taskIdentifierList</i> “)”) “;” <i>taskIdentifierList</i> “)”) “;”

# Chapter 4

## Project Plan

### 4.1 Team Responsibilities

Group members were assigned primary responsibilities with some aspects of the project. Although a large portion of decision-making and other work was done together as a group, the following describes each group member's primary tasks:

Matt Kalish	Test File Creation, Documentation Creation
Michael Ching	Lexer Creation, Parser, Final Completion and Correctness of Semantic Analysis, Intermediate Testing Documentation
Charles O'Donnell	Entire Back-End, Glue Stages, Code Generation, Documentation
Gerardo Flores	Lexer, Parsing, Semantic Analysis, Intermediate Testing, Final Documentation, Creation of Final Presentation

### 4.2 Project Timeline

The project was developed under a strict, predetermined timeline for reaching specific milestones. It factors in the natural tendency for groups to put work off until near the deadline, however, every effort was made throughout the process to get ahead of schedule. Because of this, the project log ideally will show dates much sooner than the flexible ones declared here. The schedule used is as follows:

Rough language specification, Whitepaper complete	February 18 <sup>th</sup> , 2003
Development environment and version control software chosen	March 1 <sup>st</sup> , 2003
High-level syntax determined	March 21 <sup>st</sup> , 2003
Grammar complete	March 24 <sup>th</sup> , 2003
Language Reference Manual complete	March 27 <sup>th</sup> , 2003
Parser completed	April 15 <sup>th</sup> , 2003
Static-semantic checker done	May 1 <sup>st</sup> , 2003
Back-end complete	May 8 <sup>th</sup> , 2003
Final testing phase complete, compiler code finalized	May 11 <sup>th</sup> , 2003
Finalize documentation, project complete	May 13 <sup>th</sup> , 2003

## 4.3 Software Development Environment

This project will be developed on both Linux and Solaris using Java 1.3.1. CVS will be used as the version control software, and ANTLR will aid in code generation and semantic checking in several parts of our compiler. Testing will be carried out both in UNIX and in practice using actual RCX 1.0 processors on Lego Mindstorm robots.

## 4.4 Project Log

The following describes the actual dates that we completed each project milestone:

Rough language specification, Whitepaper complete	February 15 <sup>th</sup> , 2003
Development environment and version control software chosen	February 20 <sup>th</sup> , 2003
High-level syntax determined	March 20 <sup>th</sup> , 2003
Grammar complete	March 22 <sup>nd</sup> , 2003
Language Reference Manual complete	March 26 <sup>th</sup> , 2003
Parser completed	April 14 <sup>th</sup> , 2003
Static-semantic checker done	May 10 <sup>th</sup> , 2003
Back-end complete	May 10 <sup>th</sup> , 2003
Final testing phase complete, compiler code finalized	May 11 <sup>th</sup> , 2003
Finalize documentation, project complete	May 12 <sup>th</sup> , 2003

# Chapter 5

## Architectural Design

### 5.1 Architecture

The OGEL compiler contains all of the features you would expect to find in a generic compiler. Essentially, these are the lexer, parser, static-semantic checker, and code generator. In OGEL's case, there is also a preprocessing stage, and the code generation is split into two stages, the first stage producing architecturally independent source code. The flow of the OGEL compiler's components can be seen in Figure 5.1 below.

The compiler's five stages will take an OGEL source code file as an input, and output a file, or series of files, in the NQC target language. First, the preprocessor allows any number of OGEL library files to be included (`#include`) into the main source, giving access to any number of predefined functions, tasks, or RCX types. It also allows for a simple `#define` construct which will automatically replace any definitions found later in the source files. Finally, because of the preprocessor, original OGEL source code is can have tasks, functions, rcx units, and messages defined in any order. The output of the preprocessor is a ".master" file which contains all necessary information for the parser in one file.

The lexer and parser were coded using the ANTLR tool, which generates Java source code for many of the tedious tasks required by a compiler. This stage checks for syntactic correctness, and will generate an error message if any keywords or identifiers are found in the incorrect placement. The lexer and parser creates an abstract syntax tree, which encapsulates all the necessary information about the source file, without the bloat of brackets, and other unnecessary characters. This AST is passed to the static-semantic checker for the next stage of compilation.

The static-semantic checker will walk the AST and ensure that all keywords and identifiers have the correct semantic meaning. To this end, it ensures that all variables used have been previously defined, and that all tasks and functions are connected to valid RCX units according to their port definitions. No changes are actually made to the AST, but if the static-semantic checker completes successfully, the AST is guaranteed to be completely valid and ready to be walked for code generation.

The first code generation stage is meant to generate a single UNI-OGEL output file, which contains architecturally independent code, which is designed to span all LEGO

Mindstorm platforms (eg. The RCX 1.0, RCX2, CyberMaster, Scout, and Spybotics). UNI-OGEL also contains information about the source in general which second stage translators can use to determine compatibility with a given architecture, or to help with optimizations. Some of the information provided in the “.uniogel” file includes identification of recursive loops (disallowed on some architectures), and reference counting of function use (since LEGO Mindstorm units have different numbers of available “subfunctions”, optimizing compilers can place frequently used functions as subfunctions, and flatten any other non-recursive functions). This results in a single file which still contains all information about all RCX units used, but duplicates source code and maps ports where required. The UNI-OGEL file is designed to be easily parsed by the architecture-specific translators, and formats its information in a convenient manner.

The final translator takes this .uniogel file and will produce one or more Not-Quite-C source files (a .nqc files for each RCX unit instantiated in the OGEL main task). Architecture specific requirements are checked against, such as limitations to the number of tasks, and architecturally independent OGEL constructs such as “whens” and messaging are converted to NQC source (for example, the RCX 1.0 and RCX2 have different abilities for messaging). We have created only RCX 1.0 compilers, however the creation of compilers for other platforms is relatively easy task. All NQC files leaving this stage are guaranteed to compile in the NQC compiler (Windows software) except for possible bugs in native-NQC code blocks. The NQC IDE can then upload these files to an RCX unit via the IR port.



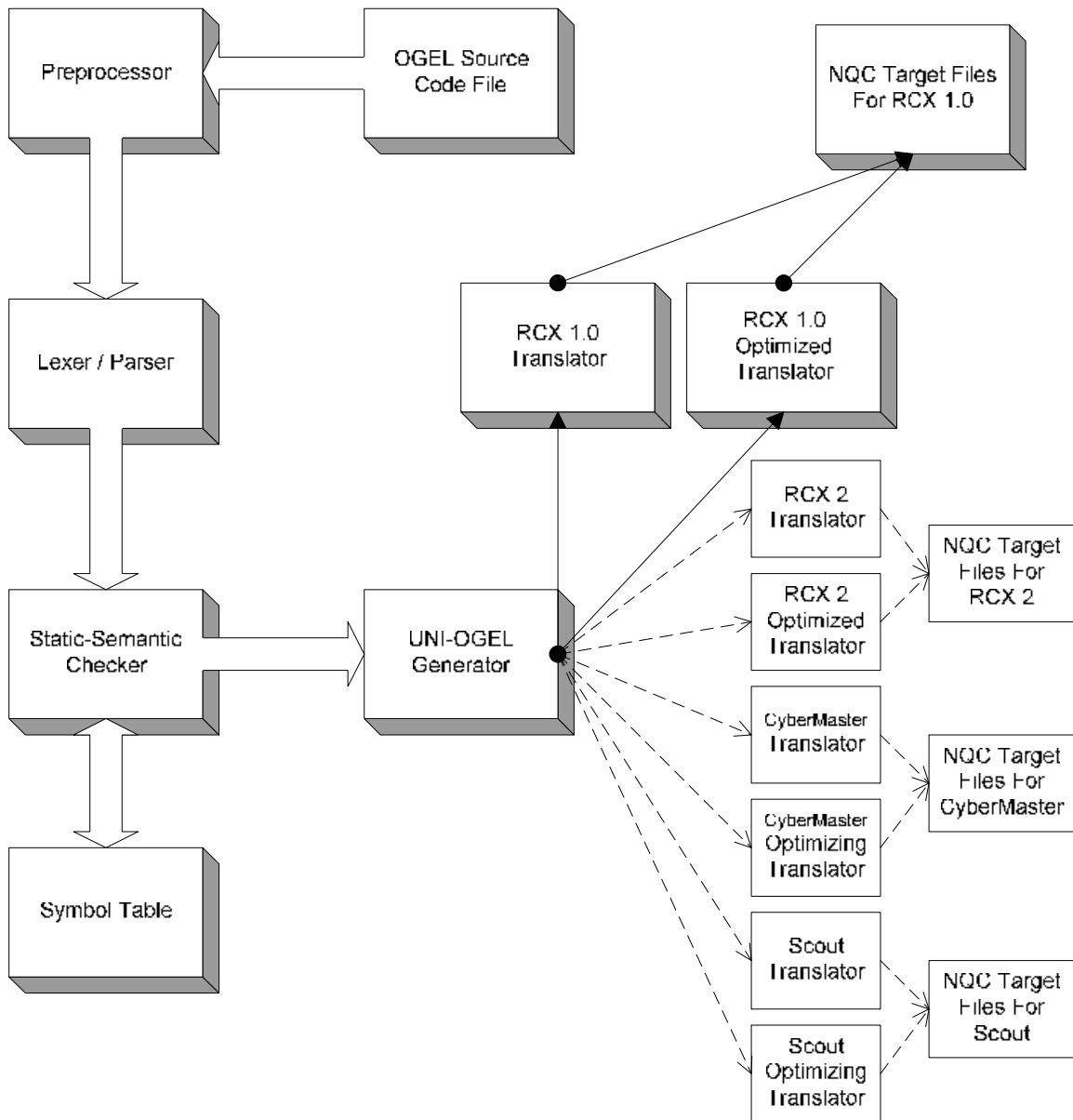


Figure 5.1 – The OGEL Compiler Flow Chart

## 5.2 Work Division

Many aspects of the compiler implementation were a combined group effort, at least as far as planning and overall organization are concerned. However, the major work on each module was done as follows:

The lexer and parser were constructed mainly by Gerardo Flores and Michael Ching. The Static-Semantic Checker was a combined effort. The back-end / code generation was done mostly by Charles O'Donnell.

# Chapter 6

## Testing Plan

### 6.1 Goals

Our primary objective in test planning for the OGEL compiler is to create a well-thought out set of test cases from which we can deduce if errors exist in the code. While it is impossible to test all possible inputs for this implementation, it is certainly reasonable to develop a strategy to accurately locate and repair code bugs, which will ultimately lead to a final product that is sufficiently robust and reliable. Please note that this documents only the major testing sequences that the OGEL group performed – it will not include details on the frequent, extremely small-scale tests that naturally occur in the development of any computer program.

### 6.2 Hypothesis

Through use of thoughtfully created test cases, it will be trivial to prove the correctness of our implementation after any change, major or minor, in the code. If bugs do exist, we will not only be able to realize this, but also have insight as to where the specific errors are located in the code due to the highly specialized nature of the tests.

### 6.3 Methods

In general, the only way to effectively code a large project is to test each significant module as it is completed, and only when satisfied with the results should one continue on to the later stages. Keeping this in mind, the following test phases were utilized:

#### 6.3.1 Phase I – Isolated Module Testing

In this phase, which will be performed after each module is completed, the group will test an individual subset of the OGEL compiler in isolation from other aspects of the project. These major subsets can be determined as the development process moves along, however a natural divide of modules for a generic compiler seems to be: lexer, parser, AST walker / static-semantic checker, and code generator. Certainly, as the OGEL group

deems necessary, these large subsets will be modularized into smaller portions that lend themselves better to isolation testing.

### **6.3.2 Phase II – Integration Testing**

This is the second phase of the OGEL group’s testing strategy, which will ensue after Phase I is complete for two connectable modules, and will then repeat after each iteration of Phase I until completion, where relevant and possible. This phase will be the first attempt to string together some sequence of modules into a unified program. Ideally, since all modules involved in the integration test should be completely functional, this phase will be brief. However, it is very necessary, not only to ensure that the program is progressing nicely, but also to catch any major errors made in the project planning. This must be a concern whenever a group gets together and divides the work up into “independent” pieces, and its better to catch such an error early rather than to complete all modules first to only then find out that they cannot be integrated. This phase will not handle large OGEL code examples, but rather complete skeletal source files that allow the group to ensure that the applicable sections of the compiler are in sync. When Phase II is completed successfully for an integration of all the project’s modules, Phase III will commence.

### **6.3.3 Phase III – Final Testing**

Phase III is the final stage, meant to test the compiler in real-life scenarios that can and will arise when developing OGEL programs. The OGEL group will ensure that every aspect of the OGEL compiler, from installation to execution, is satisfactory and as straightforward as possible. Phase III will also be the stage for elaborate OGEL programs to be tested. Ideally, the entire OGEL grammar will have been exercised by the time this phase is completed, and the group will be confident that the final compiler is the best one possible given the resources and time available for development.

## **6.4 Tools**

The tests were performed manually for the most part. The use of makefiles and bash scripts automated the tast somewhat, but the actual test files were created such that the maximum amount of code coverage was achieved.

## **6.5 Implementation**

The actual implementation of the OGEL group’s testing plan proved to be challenging and tedious, but overall it made our development process run smoothly and relatively problem-free throughout the development cycle. Details of the testing follow.

## 6.5.1 Phase I – Isolated Module Testing

Phase I seeks to test each module independent of the other aspects of the compiler. This can be done by challenging each non-trivial method of the code thoroughly to ensure that they function as expected. Here is a brief overview:

- 1) Syntax Checker – To test the lexer and parser of our compiler (as generated by ANTLR), the group will run several test cases, spanning most of the OGEL grammar, through the module. The group will also run tests using several files with incorrect syntax. This phase of testing ceases only when all of the carefully selected test cases perform as expected when executed.
- 2) Static-Semantic Checker – For this stage, we will need to use a thorough sample of AST's, both correct and semantically erroneous, to test if the checker correctly identifies all semantic errors in the source file. For this stage, once we know the Syntax Checker is fully functional, we will probably move on to Phase II to do the majority of the testing by linking the modules together. This is because hand-constructing AST's and Symbol Tables is probably not a reasonable task. However, any and all subsets of the Static-Semantic Checker are eligible for isolation testing, and will be done as the OGEL group sees fit.
- 3) Code Generator – Here we need to be sure that the NQC program(s) outputted are representative of the OGEL source file. Again, moving to Phase II and III is probably the most effective way to test this on a large scale, but the group reserves the right to test any subset of this module in isolation.

## 6.5.2 Phase II – Integration Testing

In this stage, we elected to integrate the entire parser first (i.e., lexer and parser), ensuring that it was functional for all test cases presented. The group had previously considered these two components as one big module, but during implementation we found that they were quite distinct. Therefore, some time was spent integrating them together and therefore that integration had to be thoroughly tested. Although we were able to look at an AST representation, interpreting this result for each case was somewhat time-consuming and difficult. Therefore, the most meaningful results would not be found until the code generator was finally integrated.

Nevertheless, we spent time first integrating and testing the addition of the AST walker to the parser. The final stage of this testing phase was the addition of the code generator, which was last to be implemented. The group thoroughly tested the first completely integrated compiler with many skeletal examples, and after some debugging and additions to the code, the group felt confident that it was time for testing phase III.

## 6.5.3 Phase III – Final Testing

This final stage proved relatively easy, due to the fact that Phase II eliminated many of the major problems that existed in the OGEL compiler at that time. However, many problematic bugs and quirks in the code were ironed out by finally attempting to compile large sample programs.

After this stage was completed, the OGEL group felt confident that the compiler set forth was the best product possible given our constraints. The next section will contain two examples of successful compilations, and the target NQC code that resulted from this testing.

## 6.6 Test Case Compilation Results

The following two segments exhibit the work of our compiler on two significant test programs. First the OGEL source code will be shown, immediately followed by the target NQC files that result.

### 6.6.1 Test Case 1

#### OGEL CODE:

```
#define TIME_REV 100
message done;
rcx demo {
    SENSOR_TOUCH hit = $1;
    left_motor = $A;
    right_motor = $C;
}
task go(in hit)(out left_motor, out right_motor) {
    when (hit>0) do reverse();
    nqc.OnFwd(left_motor + right_motor);
}
void turn()(out left_motor, out right_motor) {
    nqc.OnRev(left_motor);
    nqc.OnFwd(right_motor);
    nqc.Wait(TIME_REV);
    nqc.Off(left_motor + right_motor);
    throw done;
}
void reverse()(out left_motor, out right_motor) {
    nqc.OnRev(left_motor + right_motor);
    nqc.Wait(TIME_REV);
    nqc.Off(left_motor + right_motor);
    turn();
}
task rev_task()(out left_motor, out right_motor) {
    int x = 1;
```

```

while (x > 0) {
    catch done;
    reverse();
}
}
task main() {
    rcx demo first_born;
    first_born.addTask(rev_task);
    first_born.addTask(go);
}

```

### Generated NQC:

```

void reverse ( ) {
    OnRev ( OUT_A + OUT_C );
    Wait ( 100 );
    Off ( OUT_A + OUT_C );
    turn ( );
}
void turn ( ) {
    OnRev ( OUT_A );
    OnFwd ( OUT_C );
    Wait ( 100 );
    Off ( OUT_A + OUT_C );
}
task uni_ogel_whens_task() {
    while(1) {
        if ( (SENSOR_1> 0) ) { reverse( ); }
    }
}
task main() {
    start uni_ogel_whens_task;
    OnFwd ( OUT_A + OUT_C );
}

```

## 6.6.2 Test Case 2

### OGEL CODE:

```
#define TIME_REV 100
```

```
message done;
```

```

rcx demo {
    SENSOR_TOUCH hit = $1;
    left_motor = $A;
    right_motor = $C;
}

```

```

task go(in hit)(out left_motor, out right_motor) {
    when (hit>0) do reverse();
    nqc.OnFwd(left_motor + right_motor);
}

task other(in hit)(out left_motor, out right_motor) {
    <% int a=0; if (a==0) { } %>
    receive done;
    reverse();
}

void turn()(out left_motor, out right_motor) {
    nqc.OnRev(left_motor);
    nqc.OnFwd(right_motor);
    nqc.Wait(TIME_REV);
    nqc.Off(left_motor + right_motor);
    pass done;
}

void reverse()(out left_motor, out right_motor) {
    nqc.OnRev(left_motor + right_motor);
    nqc.Wait(TIME_REV);
    nqc.Off(left_motor + right_motor);
    turn();
}

task main() {
    rcx demo first_born_son;
    rcx demo worthless;
    first_born_son.addTask(go);
    worthless.addTask(other);
}

```

### Generated NQC:

#### 2 Files:

```

int done=0;
void reverse ( ) {
    OnRev ( OUT_A + OUT_C );
    Wait ( 100 );
    Off ( OUT_A + OUT_C );
    turn ( );
}
void turn ( ) {
    OnRev ( OUT_A );
    OnFwd ( OUT_C );
}

```

```

    Wait ( 100 );
    Off ( OUT_A + OUT_C );
    SendMessage(1);
}
task uni_ogel_whens_task() {
    while(1) {
        if ( (SENSOR_1> 0) ) { reverse( ); }
    }
}
task main() {
    start uni_ogel_whens_task;
    OnFwd ( OUT_A + OUT_C );
}

int done=0;
void reverse ( ) {
    OnRev ( OUT_A + OUT_C );
    Wait ( 100 );
    Off ( OUT_A + OUT_C );
    turn ( );
}
void turn ( ) {
    OnRev ( OUT_A );
    OnFwd ( OUT_C );
    Wait ( 100 );
    Off ( OUT_A + OUT_C );
    SendMessage(1);
}
task main() {
    { int a = 0 ; if ( a == 0 ) { } }
    ClearMessage();
    until (Message() == 1);
    reverse ( );
}

```

## 6.7 Work Division

Test plans with adequate code coverage developed by Matt Kalish. The large test cases were also written by Matt Kalish, with some smaller ones being written by each Gerardo Flores, Charles O'Donnell, and Micheal Ching as they were deemed necessary. The LEGO Mindstorm robots used in testing were constructed by Matt Kalish and Michael Ching. Actual Phase III testing was a cumulative group effort, with fixes being done by the author of the problematic module.



# Chapter 7

## Lessons Learned

### 7.1 Charles O'Donnell

One of the more interesting things I found when writing this compiler was the tremendous impact small, seemingly insignificant errors can have. Every error must be checked and rechecked for typos and minor semantic errors. Even a dull compiler design can involve an amazing amount of time testing and analyzing. Another issue revisited during this project was the need to define specific tasks and goals to people early on, and to define their interaction well. This, along with good milestones should ensure all people are caught up on each part of the project. Advice given for writing a compiler would be to incrementally create fully working versions, which can successively contain more and more features.

### 7.2 Matt Kalish

The most important lesson I learned is how to effectively manage my time. I feel like the raw amount of time it takes to produce anything meaningful in Computer Science is so much that without having good skills in time management, it would be impossible to finish a project of this magnitude. Furthermore, I learned how easy it is for a group, even a small one, to have misunderstandings and communication problems. I respect the difficulty of small group work much more after participating on this project.

If I could give advice to future classes, I would say that the most important thing is to be honest about your abilities, and communicate problems you are having as soon as they occur. It only hurts your group if you claim to be able to do something you can't handle, or if you are too shy or embarrassed to admit when you are incapable of completing a certain task assigned to you.

### 7.3 Michael Ching

The most important lesson I gained from this experience is to maintain a good, professional working relationship with the group. While at times it is easy to get frustrated with another person's performance, the key thing to remember is that in a long

term project, its not worth causing conflicts over small issues. In the end, everything works out best if you stay on task and professional.

My advice to the future classes is to pay close attention to the examples given in class. A lot of great help is supplied in lecture, I recall the segments showing the Esterel compiler's components as being particularly useful. So, I'd say pay attention in class and attend as often as possible.

## **7.4 Gerardo Flores**

As is usually the case, the most important thing in really making efficient use of the available time is keeping the lines of communication open. One must make sure that everyone is on the same page at all times. The most important lesson that I learned throughout this project is really how important "face time" is. It is not enough to simply instant message each other and assign tasks to one another, especially in a project of this magnitude. It is far too easy for people to shirk their tasks and never really contribute. Further, in this type of project, most of the stages are extremely interdependent and it is necessary to keep in constant contact so as not to damage the interface between stages.

# Appendix A

## Complete OGEL Grammar

<i>file</i>	→	<i>rcxDecls messageDecls functionList taskDeclaratonList mainDeclaration</i>
<i>messageDecls</i>	→	<i>(messageDeclaration)*</i>
<i>messageDeclaration</i>	→	<i>“message” messageIdentifier “;”</i>
<i>messageStatement</i>	→	<i>(passMessage   recieveMessage)</i>
<i>passMessage</i>	→	<i>“pass” messageIdentifier “;”</i>
<i>receiveMessage</i>	→	<i>“receive” messageIdentifier “;”</i>
<i>messageIdentifier</i>	→	<i>IDENTIFIER</i>
<i>taskDeclarationList</i>	→	<i>(task)*</i>
<i>task</i>	→	<i>taskDeclaration “{“ taskBody “}”</i>
<i>taskDeclaration</i>	→	<i>“task” taskIdentifier inOutList</i>
<i>taskBody</i>	→	<i>(taskBodyStatements)*</i>
<i>taskBodyStatements</i>	→	<i>whenStatement   standardStatement   NQCBLOCK</i>
<i>taskIdentifierList</i>	→	<i>(taskIdentifier (“,” taskIdentifier)*)+</i>
<i>taskIdentifier</i>	→	<i>IDENTIFIER</i>
<i>whenStatement</i>	→	<i>“when” whenCondition whenStatementDo “;”</i>
<i>whenCondition</i>	→	<i>“(“ inPortMap (LESSTHAN   GREATERTHAN   LEQUAL   GEQUAL   CEQUAL   NEQUAL) ( NUMBER   variableIdentifier) “)”</i>
<i>whenStatementDo</i>	→	<i>“do” functionCall</i>
<i>mainDeclaration</i>	→	<i>“task” “main” “(“ “)” “{“ mainBody “}”</i>
<i>mainBody</i>	→	<i>rcxInstantiations rcxAddTaskList</i>
<i>rcxInstantiations</i>	→	<i>(rcxInstantiation)+</i>
<i>rcxInstantiation</i>	→	<i>“rcx” rcxClass rcxInstanceIdentifier “;”</i>
<i>rcxClass</i>	→	<i>IDENTIFIER</i>
<i>rcxInstanceIdentifier</i>	→	<i>IDENTIFIER</i>
<i>rcxAddTaskList</i>	→	<i>(rcxAddTaskCall)+</i>
<i>rcxAddTaskCall</i>	→	<i>rcxInstanceIdentifier “.” “addTask” “(“ taskIdentifierList “)” “;”</i>
<i>rcxDecls</i>	→	<i>(rcxDeclaration)*</i>
<i>rcxDeclaration</i>	→	<i>“rcx” rcxIdentifier “{“(inputPortMapping   outputPortMapping)+ “}”</i>
<i>rcxIdentifier</i>	→	<i>IDENTIFIER</i>
<i>outputPortMapping</i>	→	<i>outPortMap “=” (OUTA   OUTB   OUTC) “;”</i>
<i>inputPortMapping</i>	→	<i>inPortMap “=” (IN1   IN2   IN3) “;”</i>
<i>outPortMap</i>	→	<i>IDENTIFIER</i>
<i>inPortMap</i>	→	<i>IDENTIFIER</i>

<i>functionList</i>	→	<i>(function)*</i>
<i>function</i>	→	<i>functionDeclaration</i> “{“ <i>statementList</i> “}”
<i>functionDeclaration</i>	→	“void” <i>functionIdentifier</i> <i>parameterList</i> <i>inOutList</i>
<i>functionCall</i>	→	<i>functionIdentifier</i> <i>functionParamsPasng</i>
<i>functionParamsPasng</i>	→	“(“ ( <i>(NUMBER   IDENTIFIER)</i> ( “,” <i>(NUMBER   IDENTIFIER)</i> )* )? “)”
<i>functionIdentifier</i>	→	<i>IDENTIFIER</i>
<i>nqcFunctionCall</i>	→	<i>NQCFUNCTION IDENTIFIER LPAREN RPAREN</i>
<i>statementList</i>	→	<i>(standardStatement)*</i>
<i>standardStatement</i>	→	<i>whileBlock</i>   “do” “{“ <i>(standardStatement)*</i> “}” “while” “(“ <i>whileCondition</i> “)”   “break” “;”   “if” “(“ <i>ifCondition</i> “)” “{“ <i>(standardStatement)*</i> “}” <i>elseBlock</i>   “return” “;”   <i>functionCall</i> “;”   <i>variableDeclaration</i>   <i>variableAssignment</i>   <i>variableDeclaredAssignment</i>   <i>messageStatement</i>   <i>nqcFunctionCall</i>
<i>elseBlock</i>	→	(“else” “{“ <i>statementList</i> “}”)?
<i>whileBlock</i>	→	“while” “(“ <i>whileCondition</i> “)” “{“ <i>whileStatements</i> “}”
<i>whileStatements</i>	→	<i>(standardStatement)*</i>
<i>whileCondition</i>	→	<i>condition</i>
<i>ifCondition</i>	→	<i>condition</i>
<i>condition</i>	→	<i>conditionStatement</i>   <i>(PARALLEL   NOT   AND) conditionStatement)*</i>   (“true”   “false”)
<i>conditionStatement</i>	→	<i>expression ( LESSTHAN   GREATERTHAN   LEQUAL   GEQUAL   CEQUAL   NEQUAL) expression</i>
<i>expression</i>	→	<i>sumExpr</i>
<i>sumExpr</i>	→	<i>prodExpr ((PLUS   DASH) prodExpr)*</i>
<i>prodExpr</i>	→	<i>powExpr ((STAR   SLASH   MOD) powExpr)*</i>
<i>powExpr</i>	→	<i>atom (POW atom)?</i>
<i>atom</i>	→	<i>(NUMBER   IDENTIFIER)</i>
<i>parameterList</i>	→	“(“ ( <i>arg</i> (“,” <i>arg</i> )* )? “)”
<i>inOutList</i>	→	<i>inList outList</i>
<i>inList</i>	→	“(“ “in” <i>inPortMap</i> (“,” “in” <i>inPortMap</i> )* )? “)”
<i>outList</i>	→	( (“out” <i>outPortMap</i> (“,” “out” <i>outPortMap</i> )* )? “)”
<i>arg</i>	→	(“int”   “const”) <i>IDENTIFIER</i>
<i>variableDeclaredAssignment</i>	→	<i>variableIdentifier (EQUALS   PLUSEQ   MINUSEQ</i>   <i>TIMESEQ   DIVIDEEQ) rValue</i> “;”
<i>variableAssignment</i>	→	<i>variableType variableIdentifier “=” rValue</i> “;”

<i>variableDeclaration</i>	→	<i>variableType</i> <i>variableIdentifier</i> “;”
<i>variableType</i>	→	("int"   "const"   "time"   "bool")
<i>variableIdentifier</i>	→	<i>IDENTIFIER</i>
<i>rValue</i>	→	( <i>NUMBER</i>   <i>IDENTIFIER</i>   “true”   “false”)
<i>PERIOD</i>	→	“.”
<i>POUND</i>	→	“#”
<i>PLUS</i>	→	“+”
<i>DASH</i>	→	“-“
<i>SLASH</i>	→	“/”
<i>STAR</i>	→	“*”
<i>PARALLEL</i>	→	“  ”
<i>LESSTHAN</i>	→	“<”
<i>GREATERTHAN</i>	→	“>”
<i>COMMA</i>	→	“,”
<i>EQUALS</i>	→	“=”
<i>SEMI</i>	→	“;”
<i>COLON</i>	→	“:”
<i>COLEQUALS</i>	→	“:=”
<i>LPAREN</i>	→	“(“
<i>RPAREN</i>	→	)”
<i>LBRACKET</i>	→	[“
<i>RBRACKET</i>	→	]”
<i>QUESTION</i>	→	“?”
<i>DQUESTION</i>	→	“??”
<i>LEQUAL</i>	→	“<=”
<i>GEQUAL</i>	→	“>=”
<i>NEQUAL</i>	→	“!=”
<i>CEQUAL</i>	→	“==”
<i>NOT</i>	→	“!”
<i>AND</i>	→	“&&”
<i>IMPLIES</i>	→	“=>”
<i>PLUSEQ</i>	→	“+=”
<i>MINUSEQ</i>	→	”-=“
<i>TIMSEQ</i>	→	“*=“
<i>DIVIDEEQ</i>	→	“/=“
<i>NQCFUNCTION</i>	→	“nqc.”
<i>LCURLY</i>	→	{“
<i>RCURLY</i>	→	}”
<i>IN1</i>	→	“\$1”
<i>IN2</i>	→	“\$2”
<i>IN3</i>	→	“\$3”
<i>OUTA</i>	→	“\$A”
<i>OUTB</i>	→	“\$B”
<i>OUTC</i>	→	“\$C”
<i>LIBRARYENDING</i>	→	“.lib”
<i>IDENTIFIER</i>	→	(“a” ..”z” ”A” ..”Z” ”_”)(“a” ..”z” ”A” ..”Z” ”_” ”0” ..”9”)*

<i>NUMBER</i>	→	("0".."9")+
<i>WS</i>	→	(" "   "\t"   "\n"   "\r")
<i>SL_COMMENT</i>	→	"//" (~"\n")* "\n"
<i>ML_COMMENT</i>	→	"/*" anythingButEnd1 "*/"
<i>NQCBLOCK</i>	→	"<% " anythingButEnd2 "%>"

# Appendix B

## Code Style Conventions

### B.1 Introduction

This section is intended to guide this group's programmers via stylistic guidelines that should be followed while coding the OGEL compiler. While the group respects the individual's right to have their own programming style, and therefore chose to not restrict the minute points of coding, we feel that some conventions are necessary to produce a fluid, professional-looking final product. What follows are the code style conventions adopted by the OGEL group for this project.

### B.2 General Principles

The code should be as human readable as possible, without compromising creativity. If an elegant (and therefore confusing) solution to a problem is implemented, it must be thoroughly documented in the source file, and / or a README. Consistency with indentation, formatting, and variable declaring is to be prioritized.

### B.3 Tabs

Tabs are to be used mainly for the purpose of consistent indentation of source code. While caution should be used since tabs may be displayed differently in different editors, they are nonetheless a valid whitespace character and should be used as such.

### B.4 Documentation Comments

Documentation must be valued as a necessary component of any compiler's source code files. Because of this, we will make every effort to thoroughly comment at least every method and variable in the code. In addition, interesting situations will be documented with the programmer's approach to the problem, and how a solution was implemented.

# Appendix C

## Complete Code Listing

### Grammar: Lexar and Parser

```
/**
@author Michael Ching, Gerardo Flores
*/
class OGELParser extends Parser;
options {
    buildAST = true;
    exportVocab = OGEL;
    k = 4;
    defaultErrorHandler = false;
}
tokens{
    FILE;
    RCXDECLS;
    PARAMETERLIST;
    INLIST;
    OUTLIST;
    RCXINSTANTIATIONS;
    TASKIDENTIFIERLIST;
    MESSAGEDECLS;
    RCXADDTASKLIST;
    RCXSTARTTASKLIST;
    RCXSTOPTASKLIST;
    TASKDECLARATIONLIST;
    FUNCTIONLIST;
    FUNCTIONPARAMSPASSING;
    INCLUDEDECLS;
    WHILECONDITION;
    IFCONDITION;
    RCXINSTANTIATION;
    STANDARDSTATEMENT;
    VARIABLEDECLARATION;
    VARIABLEASSIGNMENT;
    FUNCTIONDECLARATION;
    FUNCTION;
    STATEMENTLIST;
    FUNCTIONCALL;
    TASKDECLARATION;
    WHILESTATEMENTS;
    TASK;
    TASKBODY;
    TASKBODYSTATEMENTS;
    VARIABLEDECLAREDASSIGNMENT;
```



```

PASSMESSAGE;
THROWMESSAGE;
CATCHMESSAGE;

RECEIVEMESSAGE;
MESSAGESTATEMENT;
RCXADDTASKCALL;
RCXSTOPTASK;
WHENCONDITION;
NQCFUNCTIONCALL;
WHENSTATEMENTDO;
NQCBLOCKSTATEMENT;
NQCPARAMETERS;
NQCPARAMETERS1;
NQCPARAMETERS2;
}

file
: rcxDecls messageDecls functionList
  taskDeclarationList mainDeclaration EOF!{#file = #([FILE,
"file"], file);}
;

//MESSAGING

messageDecls
: (messageDeclaration)*
  {#messageDecls = #([MESSAGEDECLS, "messageDecls"],
messageDecls);}
;

messageDeclaration
: "message"^ messageIdentifier SEMI!
;

messageStatement
: (passMessage | receiveMessage | throwMessage | catchMessage)
;

passMessage
: "pass" messageIdentifier SEMI!
  {#passMessage = #([PASSMESSAGE, "passMessage"], passMessage);}
  //{#PASSMESSAGE = #([PASSMESSAGE, "passMessage"], passMessage);}
;

receiveMessage
: "receive" messageIdentifier SEMI!
  {#receiveMessage = #([RECEIVEMESSAGE, "receiveMessage"],
receiveMessage);}
;

throwMessage
: "throw" messageIdentifier SEMI!
  {#throwMessage = #([THROWMESSAGE, "throwMessage"],
throwMessage);}
;

```

```

catchMessage
    : "catch" messageIdentifier SEMI!
    {#catchMessage = #([CATCHMESSAGE, "catchMessage"],
catchMessage);}
    ;

messageIdentifier
    : IDENTIFIER
    ;

//TASK STUFF

taskDeclarationList
    : (task)*
    {#taskDeclarationList = #([TASKDECLARATIONLIST,
"taskDeclarationList"], taskDeclarationList);}
    ;

task
    : taskDeclaration LCURLY! taskBody RCURLY!
    {#task = #([TASK, "task"], task);}
    ;

taskDeclaration
    : "task"! taskIdentifier inOutList
    {#taskDeclaration = #([TASKDECLARATION, "taskDeclaration"],
taskDeclaration);}
    ;

taskBody
    : (taskBodyStatements)*
    {#taskBody = #([TASKBODY, "taskBody"], taskBody);}
    ;

taskBodyStatements
    : whenStatement
    | standardStatement
    ;

taskIdentifierList
    : (taskIdentifier (COMMA! taskIdentifier)*)+
    {#taskIdentifierList = #([TASKIDENTIFIERLIST,
"taskIdentifierList"], taskIdentifierList);}
    ;

taskIdentifier
    : IDENTIFIER
    ;

whenStatement
    : "when"^ LPAREN! whenCondition RPAREN! whenStatementDo SEMI!
    ;

```

```

whenStatementDo
    : "do"! functionCall
      {#whenStatementDo = #([WHENSTATEMENTDO, "whenStatementDo"],
whenStatementDo);}
    ;

whenCondition
    : whenConditionStatement
      (options { greedy = true;}
      : (PARALLEL | NOT | AND) whenConditionStatement)*
      {#whenCondition = #([WHENCONDITION, "whenCondition"],
whenCondition);}
    ;

whenConditionStatement
    : inPortMap ( LESSTHAN^ | GREATERTHAN^ | LEQUAL^ | GEQUAL^ |
CEQUAL^ | NEQUAL^ ) ( NUMBER | variableIdentifier)
    ;

//MAIN

mainDeclaration
    : "task"! "main"^ LPAREN! RPAREN! LCURLY! mainBody RCURLY!
    ;

mainBody: rcxInstantiations rcxAddTaskList
    ;

rcxInstantiations
    : (rcxInstantiation)+
      {#rcxInstantiations = #([RCXINSTANTIATIONS, "rcxInstantiations"],
rcxInstantiations);}
    ;

rcxInstantiation
    : "rcx"! rcxClass rcxInstanceIdentifier SEMI!
      {#rcxInstantiation = #([RCXINSTANTIATION, "rcxInstantiation"],
rcxInstantiation);}
    ;

rcxClass
    : IDENTIFIER
    ;

rcxInstanceIdentifier
    : IDENTIFIER
    ;

rcxAddTaskList
    : (rcxAddTaskCall)+
      {#rcxAddTaskList = #([RCXADDTASKLIST, "rcxAddTaskList"],
rcxAddTaskList);}
    ;

```

```

rcxAddTaskCall
    : rcxInstanceIdentifier PERIOD! "addTask"! LPAREN!
taskIdentifierList RPAREN! SEMI!
    {#rcxAddTaskCall = #([RCXADDTASKCALL, "rcxAddTaskCall"],
rcxAddTaskCall);}
    ;

rcxStartTaskList
    : (rcxStartTask)+
    {#rcxStartTaskList = #([RCXSTARTTASKLIST, "rcxStartTaskList"],
rcxStartTaskList);}
    ;

rcxStartTask
    : rcxInstanceIdentifier PERIOD! "start"! LPAREN!
taskIdentifierList RPAREN! SEMI!
    ;

rcxStopTaskList
    : (rcxStopTask)+
    {#rcxStopTaskList = #([RCXSTOPTASKLIST, "rcxStopTaskList"],
rcxStopTaskList);}
    ;

rcxStopTask
    : rcxInstanceIdentifier PERIOD! "stop"^ LPAREN!
taskIdentifierList RPAREN! SEMI!
    {#rcxStopTask = #([RCXSTOPTASK, "rcxStopTask"], rcxStopTask);}
    ;

rcxDecls
    : (rcxDeclaration)*
    {#rcxDecls = #([RCXDECLS, "rcxDecls"], rcxDecls);}
    ;

rcxDeclaration
    : "rcx"^ rcxIdentifier LCURLY! (inputPortMapping |
outputPortMapping)+ RCURLY!
    ;

rcxIdentifier
    : IDENTIFIER
    ;

outputPortMapping
    : outPortMap EQUALS^ (OUTA|OUTB|OUTC) SEMI!
    ;

inputPortMapping
    : IDENTIFIER inPortMap EQUALS^ (IN1|IN2|IN3) SEMI!
    ;

outPortMap
    : IDENTIFIER

```

```

;

inPortMap
  : IDENTIFIER
  ;

//FUNCTION STUFF

functionList
  : (function)*
  {#functionList = #([FUNCTIONLIST,"functionList"], functionList);}
  ;

function
  : functionDeclaration LCURLY! statementList RCURLY!
  {#function = #([FUNCTION, "function"], function);}
  ;

functionDeclaration
  : "void"! functionIdentifier parameterList inOutList
  {#functionDeclaration = #([FUNCTIONDECLARATION,
"functionDeclaration"], functionDeclaration);}
  ;

functionCall
  : functionIdentifier functionParamsPassing
  {#functionCall = #([FUNCTIONCALL, "functionCall"],
functionCall);}
  ;

nqcFunctionCall
  : NQCFUNCTION! IDENTIFIER LPAREN! (nqcParameters | nqcParameters1
| nqcParameters2) RPAREN!
  {#nqcFunctionCall = #([NQCFUNCTIONCALL, "nqcFunctionCall"],
nqcFunctionCall);}
  ;

nqcParameters2
  : NUMBER
  {#nqcParameters2 = #([NQCPARAMETERS2, "nqcParameters2"],
nqcParameters2);}
  ;

nqcParameters1
  : (NUMBER COMMA NUMBER)
  {#nqcParameters1 = #([NQCPARAMETERS1, "nqcParameters1"],
nqcParameters1);}
  ;

nqcParameters
  : (outPortMap (PLUS outPortMap)*)?

```

```

        {#nqcParameters = #([NQCParameters, "nqcParameters"],
nqcParameters);}
    ;

functionParamsPassing
    : LPAREN! ( (NUMBER|IDENTIFIER) ( COMMA! (NUMBER|IDENTIFIER) )*
)? RPAREN!
    {#functionParamsPassing =
#([FUNCTIONPARAMSPASSING, "functionParamsPassing"],
functionParamsPassing); }
    ;

functionIdentifier
    : IDENTIFIER
    ;

statementList
    : (standardStatement)*
    {#statementList = #([STATEMENTLIST, "statementList"],
statementList);}
    ;

standardStatement
    : whileBlock
    | "do"^ LCURLY! (standardStatement)* RCURLY! "while"! LPAREN!
whileCondition RPAREN!
    | "break"^ SEMI!
    | "if"^ LPAREN! ifCondition RPAREN! LCURLY! (standardStatement)*
RCURLY! elseBlock
    | "return"^ SEMI!
    | functionCall SEMI!
    | variableDeclaration
    | variableAssignment
    | variableDeclaredAssignment
    | messageStatement
    | nqcFunctionCall SEMI!
    | nqcBlockStatement
    ;

nqcBlockStatement
    : NQCBLOCK
    {#nqcBlockStatement = #([NQCBLOCKSTATEMENT, "nqcBlockStatement"],
nqcBlockStatement);}
    ;

elseBlock
    : ("else"^ LCURLY! statementList RCURLY!)?
    ;

whileBlock
    : "while"^ LPAREN! whileCondition RPAREN! LCURLY! whileStatements
RCURLY!
    ;

whileStatements
    : (standardStatement)*

```

```

        {#whileStatements = #([WHILESTATEMENTS, "whileStatements"],
whileStatements);}
        ;

whileCondition
    : condition
    {#whileCondition = #([WHILECONDITION, "whileCondition"],
whileCondition);}
    ;

ifCondition
    : condition
    {#ifCondition = #([IFCONDITION, "ifCondition"], ifCondition);}
    ;

condition
    : conditionStatement
    ( options { greedy = true;}
    : (PARALLEL | NOT | AND) conditionStatement)*
    | ("true" | "false")
    ;

conditionStatement
    : expression ( LESSTHAN^ | GREATERTHAN^ | LEQUAL^ | GEQUAL^ |
CEQUAL^ | NEQUAL^ ) expression
    ;

expression
    : sumExpr
    ;

sumExpr
    : prodExpr ((PLUS^|DASH^) prodExpr)*
    ;

prodExpr
    : powExpr ((STAR^ | SLASH^ | MOD^) powExpr)*
    ;

powExpr
    : atom (POW^ atom)?
    ;

atom
    : NUMBER
    | IDENTIFIER
    ;

parameterList
    : LPAREN! ( arg (COMMA! arg)* )? RPAREN!
    { #parameterList = #([PARAMETERLIST, "parameterList"],
parameterList); }
    ;

inOutList

```

```

        :  inList outList
        ;

inList
    : LPAREN! ("in"! inPortMap (COMMA! "in"! inPortMap)* )? RPAREN!
    {#inList = #([INLIST, "inList"], inList);}
    ;

outList
    : LPAREN! ("out"! outPortMap (COMMA! "out"! outPortMap)* )?
RPAREN!
    { #outList = #([OUTLIST,"outList"], outList);}
    ;

arg
    : ("int"|"const") IDENTIFIER
    ;

variableDeclaredAssignment
    : variableIdentifier (EQUALS^ | PLUSEQ^ | MINUSEQ^ | TIMESEQ^ |
DIVIDEEQ^ )
    rValue SEMI!
    {#variableDeclaredAssignment = #([VARIABLEDECLAREDASSIGNMENT,
"variableDeclaredAssignment"], variableDeclaredAssignment);}
    ;

variableAssignment: variableType variableIdentifier EQUALS^ rValue
SEMI!
    {#variableAssignment = #([VARIABLEASSIGNMENT,
"variableAssignment"], variableAssignment);}
    ;

variableDeclaration : variableType variableIdentifier SEMI!
    {#variableDeclaration = #([VARIABLEDECLARATION,
"variableDeclaration"], variableDeclaration);}
    ;

variableType : ("int" | "const" | "time" | "bool")
    ;

variableIdentifier: IDENTIFIER
    ;

rValue: (NUMBER | IDENTIFIER | "true" | "false")
    ;

class OGELLexer extends Lexer;

options {
    k = 4; // Lookahead to distinguish, e.g., :
and := */
charVocabulary = '\3'..\377';

    exportVocab = OGEL; // Export these token types for tree
walkers

```



```

    testLiterals = false;           // Disable checking every rule against
keywords
}

```

```

PERIOD :      '.' ;
POUND :      '#' ;
PLUS :       '+' ;
DASH :       '-' ;
SLASH :      '/' ;
STAR :       '*' ;
PARALLEL :   "||" ;
LESSTHAN :   '<' ;
GREATERTHAN : '>' ;
COMMA :      ',' ;
EQUALS :     '=' ;
SEMI :       ';' ;
COLON :      ':' ;
COLEQUALS :  ":@" ;
LPAREN :     '(' ;
RPAREN :     ')' ;
LBRACKET :   '[' ;
RBRACKET :   ']' ;
QUESTION :   '?' ;
DQUESTION :  "??" ;
LEQUAL :     "<=" ;
GEQUAL :     ">=" ;
NEQUAL :     "!=" ;
CEQUAL :     "==" ;
NOT :        '!' ;
AND :        "&&" ;
IMPLIES :    ">=" ;
PLUSEQ :     "+=" ;
MINUSEQ :    "-=" ;
TIMESEQ :    "*=" ;
DIVIDEEQ :   "/=" ;
NQCFUNCTION : "nqc." ;
LCURLY :     '{' ;
RCURLY :     '}' ;
IN1 :        "$1" ;
IN2 :        "$2" ;
IN3 :        "$3" ;
OUTA :       "$A" ;
OUTB :       "$B" ;
OUTC :       "$C" ;

```

IDENTIFIER

```

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

```

NUMBER

```

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

```

```

WS : (' ' | '\t')

```

```

|      '\n'  {newline();}
|      '\r'
|      {$setType(Token.SKIP); }
;

SL_COMMENT :
  "//"
  (~'\n')* '\n'
  { $setType(Token.SKIP); newline(); }
;

// multiple-line comments
ML_COMMENT: "/*"(options{generateAmbigWarnings=false;}
:
{ LA(2)!='/' }? '*'
|      '\r' '\n'      {newline();}
|      '\r'          {newline();}
|      '\n'          {newline();}
|      ~('*'|'\n'|'\r')
)*
"*/"
      {$setType(Token.SKIP);}
;

NQCBLOCK: "<%"
      (options {generateAmbigWarnings=false;}
:
      { LA(2)!='<'|'>' }? '%'
|      '\r' '\n'      {newline();}
|      '\r'          {newline();}
|      '\n'          {newline();}
|      ~('%'|'\n'|'\r')
)*
"%>"
      //{$setType(Token.SKIP);}
;

```

## Semantic Tree Walker

```
/**
```

```
@author Michael Ching, Gerardo Flores
```

```
*/
```

```

header{
import java.util.*;
}

class OGELSemanticWalker extends TreeParser;
options {
    importVocab = OGEL;
}
{

```

```

        //vectors that hold persistent information that's needed
throughout
        //semantic walking
        Vector RCXs = new Vector();
        Vector Funcs = new Vector();
        Vector Messages = new Vector();
        Vector Tasks = new Vector();
        Vector Main = new Vector();
        Vector NQCFuncs = new Vector();

    }

file: #(FILE setRCXs setMessages setFunction setTask
setMainDeclaration)
    ;

//rule that traverses all "rcx" nodes under RCXDECLS
setRCXs
{ //name of RCX class
    String className;
    //port mappings
    String port1 = null;
    String port2= null;
    String port3= null;

    String portA= null;
    String portB= null;
    String portC= null;
}

: #(RCXDECLS
    (#("rcx" className=getClass

    { //count the number of in and out ports declared
        int inports = 0;
        int outports = 0;
    }
    #(d:EQUALS c:IDENTIFIER
    {
    AST a;
    AST b;

    if(d.getNumberOfChildren() > 2){
        {
            a=c.getNextSibling();
            b=a.getNextSibling();

        }
    }
    else{
        a = c;
        b = a.getNextSibling();
    }

    }

    if(d.getNumberOfChildren() > 2){

```

```

String sensType = c.getText();
boolean checkSens = false;

if(sensType.equals("SENSOR_TOUCH")){
    checkSens = true;
}
if(sensType.equals("SENSOR_LIGHT")){
    checkSens = true;
}
if(sensType.equals("SENSOR_ROTATION")){
    checkSens = true;
}
if(sensType.equals("SENSOR_CELSIUS")){
    checkSens = true;
}
if(sensType.equals("SENSOR_FAHRENHEIT")){
    checkSens = true;
}
if(sensType.equals("SENSOR_PULSE")){
    checkSens = true;
}
if(sensType.equals("SENSOR_EDGE")){
    checkSens = true;
}

if(!checkSens){
    System.err.println("Error: Undefined sensor type
declaration: "+sensType+". Please indicate SENSOR_TOUCH,
SENSOR_LIGHT, SENSOR_ROTATION, SENSOR_CELSIUS, SENSOR_FAHRENHEIT,
SENSOR_PULSE or SENSOR_EDGE");
    System.exit(1);
}

String temp = b.getText();

if(temp.equals("$1")){
    port1 = a.getText();
    inports++;
}
if(temp.equals("$2")){
    port2 = a.getText();
    inports++;
}
if(temp.equals("$3")){
    port3 = a.getText();
    inports++;
}
if(temp.equals("$A")){

```

```

        portA = a.getText();
        outports++;
    }
    if(temp.equals("$B")){
        portB = a.getText();
        outports++;
    }
    if(temp.equals("$C")){
        portC = a.getText();
        outports++;
    }
    //else the port is not $abc123
    else if(!temp.equals("$A") && !temp.equals("$B") &&
!temp.equals("$C")
        && !temp.equals("$1") && !temp.equals("$2") &&
!temp.equals("$3")){
        System.err.println("Port misdeclared: "+a.getText()+
        "cannot be assigned to "+b.getText());
        System.exit(1);
    }
    //ends the String temp
}))*
//checks to see if too many ports have been declared
{
    if((inports > 3) || (outports > 3)){
        System.err.println("Too many ports declared");
        System.exit(1);
    }
}

)

//when traversed one RCX, just create a new RCX object with the
//required port mapping and names and insert it into the global
//RCXs vector which will hold all RCX objects
{RCX temp = new RCX(className, port1, port2, port3, portA, portB,
    portC);
    RCXs.add(temp);
}

)*
;

//returns the class name of the rcx
getClass returns [String name]
{name = null;}
:IDENTIFIER {name = #IDENTIFIER.getText();}
;

//parses the messages that are declared and adds them to the Messages
vector
setMessages: #(MESSAGEDECLS (#("message" a:IDENTIFIER
    {Messages.add(a.getText()); }))*
;

setFunction: #(FUNCTIONLIST (#(FUNCTION
    //for each function make a function object

```

```

    {Function f = new Function();}
    parseFuncDeclaration[f] parseStatementList[f]))* )
;

parseFuncDeclaration [Function f]
: #(FUNCTIONDECLARATION a:IDENTIFIER
{f.name = a.getText();}
setParameterList[f] setInList[f] setOutList[f]
{
f.loadPortsAndParams();
Funcs.add(f);}
)
;

setParameterList [Function f]
: #(PARAMETERLIST (("int"|"const") a:IDENTIFIER
{f.addParam(a.getText());}) * )

;

setInList [Function f]
: #(a:INLIST
{
if((a.getNumberOfChildren() >= 0) && (a.getNumberOfChildren() <
4) )
{
AST b = a.getFirstChild();
for(int i = 0; i < a.getNumberOfChildren(); i++)
{
f.inports[i] = b.getText();
b = b.getNextSibling();
}
}
else{
System.err.println("Error: too many input port mappings
defined: "+f.name);
System.exit(1);
}
}
)
;

setOutList [Function f]
: #(a:OUTLIST
{if((a.getNumberOfChildren() > 0) && (a.getNumberOfChildren() <
4) )
{
AST b = a.getFirstChild();
for(int i = 0; i < a.getNumberOfChildren(); i++)
{
f.outports[i] = b.getText();
b = b.getNextSibling();
}
}
else{

```

```

        System.err.println("Error: too many output port mappings
defined: "+f.name);
        System.exit(1);
    }

}
)
;

```

```

parseStatementList [Function f]
: # (a: STATEMENTLIST
{int children = a.getNumberOfChildren();

if(children > 0){
    AST b = a.getFirstChild();
    for(int i =0; i < children; i++){
        String temp = b.getText();
        if(temp.equals("while"))
            parseWhileBlock(b, f);
        else if(temp.equals("do"))
            parseDo(b, f);
        else if(temp.equals("if"))
            parseIf(b, f);
        else if(temp.equals("variableAssignment"))
            parseVariableAssignment(b, f);
        else if(temp.equals("variableDeclaredAssignment"))
            parseVariableDeclaredAssignment(b, f);
        else if(temp.equals("variableDeclaration"))
            parseVariableDeclaration(b, f);
        else if(temp.equals("functionCall"))
            parseFunctionCall(b, f);
        else if(temp.equals("passMessage"))
            parsePassMessage(b);
        else if(temp.equals("receiveMessage"))
            parseReceiveMessage(b);
        else if(temp.equals("throwMessage"))
            parseThrowMessage(b);
        else if(temp.equals("catchMessage"))
            parseCatchMessage(b);
        else if(temp.equals("nqcFunctionCall"))
            parseNQCFunctionCall(b, f);
        b = b.getNextSibling();
    }
}
})
;

```

```

parseThrowMessage
: # (a: THROWMESSAGE
{
    AST intermediate = a.getFirstChild();
    AST c = (AST)intermediate.getNextSibling();
    String messageName = c.getText();
    int messageVecSize = Messages.size();
    boolean check = false;

```

```

    for(int i=0; i < messageVecSize; i++){
        String tempString = (String)Messages.elementAt(i);
        if(tempString.equals(messageName)){
            check = true;
            break;
        }
    }
    if(!check){
        System.err.println("Message identifier: "+messageName+"
:undefined");
        System.exit(1);
    }
});

```

parseCatchMessage

```

: #(a:CATCHMESSAGE
{
AST intermediate = a.getFirstChild();
AST c = (AST)intermediate.getNextSibling();
String messageName = c.getText();
int messageVecSize = Messages.size();
boolean check = false;
for(int i=0; i < messageVecSize; i++){
    String tempString = (String)Messages.elementAt(i);
    if(tempString.equals(messageName)){
        check = true;
        break;
    }
}
if(!check){
    System.err.println("Message identifier: "+messageName+"
:undefined");
    System.exit(1);
}
});

```

parseNQCFUNCTIONCALL [Function f]

```

: #(a:NQCFUNCTIONCALL
{
AST b = a.getFirstChild();
AST params = b.getNextSibling();
int numParams = params.getNumberOfChildren();

String func = b.getText();
NQCFUNCTIONCALL tempNQC = f.nqcFunctionList;
String [] tempFunctions = tempNQC.functions;
int length = tempFunctions.length;
boolean found = false;
int index = 0;
for(int i=0; i < length; i++){
    String checkFunc = tempFunctions[i];
    if(func.equals(checkFunc)){
        found = true;
    }
}
}
});

```



```

        index = i;
        break;
    }
}
if(found == false){
    System.err.println("Attempting to call a non NQC function
as a NQC function: "+func+" in function: "+f.name);
    System.exit(1);
}

//must now verify the function
//On function
if(((index >= 0) && (index < 8)) || (index == 11)){
    if((index >= 0) && (index < 8)){
        if((params.getFirstChild().getType()) == NUMBER){
            System.err.println("Cannot pass NQC function:
"+f.nqcFunctionList.functions[index]+"() an int parameter");
            System.exit(1);
        }
    }
    parseNQCFunctionRegular(params, f, index);
}
if(((index == 10) || (index == 13) || (index == 15)) &&
(numParams == 0)){
    parseNQCFunctionNoParam(params, f, index);
}

if(((index == 10) || (index == 13) || (index == 15)) &&
(numParams != 0)){
    System.err.println("Cannot pass a paramter with NQC
function: "+f.nqcFunctionList.functions[index]+"()");
    System.exit(1);
}
if(index == 8){
    parseNQCFunctionPlaySound(params, f, index);
}

if(index == 9){
    parseNQCFunctionParam1(params, f, index);
}
if(((index == 12) || (index == 14))){
    int child = params.getNumberOfChildren();
    if( child != 1){
        System.err.println("Must pass exactly one parameter
to NQC function: "+f.nqcFunctionList.functions[index]+"() in function:
"+f.name);
        System.exit(1);
    }
    AST param2 = params.getFirstChild();
    int type = param2.getType();
    if(type == NUMBER)
        parseNQCFunctionParam2(params, f, index);
    if(type != NUMBER){
        if(index == 12){
            System.err.println("Cannot pass NQC function
Wait() variable: "+param2.getText()+" in function: "+f.name+". Must
pass an 'int'");

```

```

        System.exit(1);
    }
    if(index == 14){
        System.err.println("Cannot pass NQC function
SetSleepTime() variable: "+param2.getText()+" in function: "+f.name+".
Must pass an 'int'");
        System.exit(1);
    }
}
})
;

```

```

parseNQCFunctionParam2 [Function f, int index]
:#{a:NQCPARAMETERS2
{

    int children = a.getNumberOfChildren();
    if(children != 1){
        System.err.println("Error: Can only pass one parameter to
NQC function: "+f.nqcFunctionList.functions[index]+"() in function:
"+f.name);
        System.exit(1);
    }
    int type = a.getFirstChild().getType();
    if(!(type == NUMBER)){
        System.err.println("Error: Can only pass an int to NQC
function: "+f.nqcFunctionList.functions[index]+"() in function:
"+f.name);
        System.exit(1);
    }
});

```

```

//takes 2 number parameters
parseNQCFunctionParam1 [Function f, int index]
:#{a:NQCPARAMETERS1
{

    int children = a.getNumberOfChildren();

    if(children != 3){
        System.err.println("Error: Must pass two parameters to
PlayTone()");
        System.exit(1);
    }
    int temp = a.getFirstChild().getType();
    if(!(temp==NUMBER)){
        System.err.println("Error: Parameter:
"+a.getFirstChild().getText()+" in function: "+f.name+" must be of type
int");
        System.exit(1);
    }
    AST intermediate = a.getFirstChild().getNextSibling();
    AST b = intermediate.getNextSibling();

```

```

        int temp2 = b.getType();
        if(!(temp2==NUMBER)){
            System.err.println("Error: Parameter: "+b.getText()+" in
function: "+f.name+" must be of type int");
            System.exit(1);
        }
    });

//looking for one parameter, of type keyword for sound
parseNQCFunctionPlaySound [Function f, int index]
    :#(a:NQCPARAMETERS
    {

        int children = a.getNumberOfChildren();
        if(children != 1){
            System.err.println("Error: Must list one parameter for NQC
function: "+ f.nqcFunctionList.functions[index]+" () in function:
"+f.name);
            System.exit(1);
        }
        String sound = a.getFirstChild().getText();

        boolean check = false;
        if((sound.equals("SOUND_CLICK")) ||
(sound.equals("SOUND_DOUBLE_BEEP")) || (sound.equals("SOUND_DOWN")) ||
(sound.equals("SOUND_UP")) || (sound.equals("SOUND_LOW_BEEP")) ||
(sound.equals("SOUND_FAST_UP"))){
            check = true;
        }
        if(!check){
            System.err.println("Error: Incorrect PlaySound() parameter:
"+sound+" in function: "+f.name+". Use: SOUND_CLICK,
SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP,
SOUND_FAST_UP");
            System.exit(1);
        }
    });

//nqc function with no parameters
parseNQCFunctionNoParam [Function f, int index]
    :#(a:NQCPARAMETERS
    {

        int children = a.getNumberOfChildren();
        if(children > 0){
            System.err.println("Error: NQC function:
"+f.nqcFunctionList.functions[index]+" in function: "+f.name+" does not
take any parameters");
            System.exit(1);
        }
    });
;

//reads in a bunch of output ports

```

```

parseNQCFunctionRegular [Function f, int index]
    :#(a:NQCPARAMETERS
    {

        int children = a.getNumberOfChildren();
        if((children ==1) && (index == 11)){
            boolean check = false;
            String temp = a.getFirstChild().getText();
            if( (temp.equals("TX_POWER_LO")) ||
(temp.equals("TX_POWER_HI"))){
                check = true;
            }
            if(!check){

                System.err.println("Error: Parameter:
"+a.getFirstChild().getText()+" in NQC function: SetTxPower() in
function: "+f.name+" must be TX_POWER_LO or TX_POWER_HI");
                System.exit(1);
            }
            }
            else{

                if((children < 1) || (children > 5)){
                    System.err.println("Error: incorrect number of parameters
for NQC function: "+f.nqcFunctionList.functions[index]+"() in function:
"+f.name+". Must have at least one or up to three output ports");
                    System.exit(1);
                }
                AST temp = a.getFirstChild();
                for(int i=0; i < children; i++){
                    String outPortPassed = temp.getText();
                    if(!(temp.getType() == PLUS)){
                        boolean verify = false;
                        for(int j=0; j < 3; j++){
                            if(f.outports[j] != null){
                                String tempOutPort = f.outports[j];
                                if(outPortPassed.equals(tempOutPort)){
                                    verify = true;
                                }
                            }
                        }

                        if(!verify){
                            System.err.println("Error: attempting to use
undeclared output port: "+outPortPassed);
                            System.exit(1);
                        }
                    }
                }
                temp = temp.getNextSibling();
            }
        }
    }
}
;

```

```

parseWhileCondition [Function f]
  :#(a:WHILECONDITION
  {
  int numConditions = a.getNumberOfChildren();

  AST condition = a.getFirstChild();

  for(int i = 0; i < numConditions; i++){
    boolean verify = false;
    if((condition.getNumberOfChildren() != 0)
    {

      String variable = condition.getFirstChild().getText();
      ListIterator itr = f.symbolTable.listIterator(0);
      Hashtable current = (Hashtable)f.symbolTable.getFirst();

      if(current.containsKey(variable))
        verify = true;
      else{while( !current.containsKey(variable) && itr.hasNext()
    ){
          current = (Hashtable)itr.next();
          if(current.containsKey(variable)){
            verify = true;
            break;
          }
        }
        if (verify == false){
          System.err.println("Error: Variable "+variable+" has
not been defined");
          System.exit(1);
        }
      }
      condition = condition.getNextSibling();
    }

  });

```

```

parseWhileBlock [Function f]
  :#(a:"while" {
  Hashtable t = new Hashtable();
  f.symbolTable.addFirst(t);
  AST whileCondition = a.getFirstChild();
  AST whileStatements = whileCondition.getNextSibling();

  int numConditions = whileCondition.getNumberOfChildren();

  AST condition = whileCondition.getFirstChild();

  parseWhileCondition(whileCondition, f);

```

```

//whileStatement Checking
int children = whileStatements.getNumberOfChildren();
AST b = whileStatements.getFirstChild();
for (int j = 0; j < children; j++){
    String temp = b.getText();
    if(temp.equals("while"))
        parseWhileBlock(b, f);
    else if(temp.equals("do"))
        parseDo(b, f);
    else if(temp.equals("if"))
        parseIf(b, f);
    else if(temp.equals("variableAssignment"))
        parseVariableAssignment(b, f);
    else if(temp.equals("variableDeclaredAssignment"))
        parseVariableDeclaredAssignment(b, f);
    else if(temp.equals("variableDeclaration"))
        parseVariableDeclaration(b, f);
    else if(temp.equals("functionCall"))
        parseFunctionCall(b, f);
    else if(temp.equals("passMessage"))
        parsePassMessage(b);
    else if(temp.equals("receiveMessage"))
        parseReceiveMessage(b);
    else if(temp.equals("nqcFunctionCall"))
        parseNQCFunctionCall(b, f);
    else if(temp.equals("throwMessage"))
        parseThrowMessage(b);
    else if(temp.equals("catchMessage"))
        parseCatchMessage(b);
    b = b.getNextSibling();
}

f.symbolTable.removeFirst();
});

parseDo [Function f]
: # (a: "do"
{
//add a new context
Hashtable t = new Hashtable();
f.symbolTable.addFirst(t);

int children = a.getNumberOfChildren();
AST b = a.getFirstChild();
for(int i =0; i < children; i++){
    String temp = b.getText();
    if(temp.equals("while"))
        parseWhileBlock(b, f);
    else if(temp.equals("do"))
        parseDo(b, f);
    else if(temp.equals("if"))
        parseIf(b, f);
    else if(temp.equals("variableAssignment"))
        parseVariableAssignment(b, f);
    else if(temp.equals("variableDeclaredAssignment"))
        parseVariableDeclaredAssignment(b, f);
    else if(temp.equals("variableDeclaration"))

```

```

        parseVariableDeclaration(b, f);
    else if(temp.equals("functionCall"))
        parseFunctionCall(b, f);
    else if(temp.equals("whileCondition"))
        parseWhileCondition(b, f);
    else if(temp.equals("passMessage"))
        parsePassMessage(b);
    else if(temp.equals("receiveMessage"))
        parseReceiveMessage(b);
    else if(temp.equals("nqcFunctionCall"))
        parseNQCFunctionCall(b, f);
    else if(temp.equals("throwMessage"))
        parseThrowMessage(b);
    else if(temp.equals("catchMessage"))
        parseCatchMessage(b);
    b = b.getNextSibling();
}
f.symbolTable.removeFirst();
})
;

parseIf [Function f]
: #(a:"if" b:IFCONDITION
{
    parseIfCondition(b, f);

    //add a new context
    Hashtable t = new Hashtable();
    f.symbolTable.addFirst(t);

    int children = a.getNumberOfChildren();
    boolean done = false;

    //if there is actually something inside the if like standard
    //statements then we traverse them.  else do nothing
    if(children > 1){
        b = b.getNextSibling();
        for(int i =0; i < children-1; i++){
            String temp = b.getText();
            if(temp.equals("while"))
                parseWhileBlock(b, f);
            else if(temp.equals("do"))
                parseDo(b, f);
            else if(temp.equals("if"))
                parseIf(b, f);
            else if(temp.equals("variableAssignment"))
                parseVariableAssignment(b, f);
            else if(temp.equals("variableDeclaredAssignment"))
                parseVariableDeclaredAssignment(b, f);
            else if(temp.equals("variableDeclaration"))
                parseVariableDeclaration(b, f);
            else if(temp.equals("functionCall"))
                parseFunctionCall(b, f);
            else if(temp.equals("nqcFunctionCall"))
                parseNQCFunctionCall(b, f);
            else if(temp.equals("else")){
                f.symbolTable.removeFirst();

```





```

        System.exit(1);
    }
}
condition = condition.getNextSibling();
}

});

parseFunctionCall [Function f]
: #(FUNCTIONCALL a:IDENTIFIER b:FUNCTIONPARAMSPASSING
{
//check to see if the function exists
boolean found = false;
Object[] fa = Funcs.toArray();
for(int i = 0; i < fa.length ; i++){
    if( ((Function)fa[i]).name.equals(a.getText()) )
        found = true;
}
if(found){

//check for matching number of parameters
//now check for matching number of parameters
int index = 0;
int funcSize = Funcs.size();
for(int i=0; i < funcSize; i++){
    Function tempFunc = (Function)Funcs.elementAt(i);
    String functionName = tempFunc.name;
    if(a.equals(functionName)){
        index = i;
    }
}

int numParams = b.getNumberOfChildren();

Function tempFunc2 = (Function)Funcs.elementAt(index);
int checkParam = tempFunc2.counter;

if(numParams != checkParam){
    System.err.println("Error: mismatched parameters "+a);
    System.exit(1);
}

int children = b.getNumberOfChildren();
if(children != 0){
AST c = b.getFirstChild();

//iterate through each child and check that the parameter has
already
//been defined
for(int i =0; i < children; i++){
    if(c.getType() == IDENTIFIER){
        boolean verify = false;
        String variable = c.getText();
        ListIterator itr = f.symbolTable.listIterator(0);
        Hashtable current =
(Hashtable)f.symbolTable.getFirst();

```

```

        if(current.containsKey(variable)){
            Variable temp =
(Variable)current.get(variable);
            String type = temp.type;
            if((type.equals("in")) ||
(type.equals("out"))){
                System.err.println("Cannot pass a portmap
identifier to a function: "+variable);
                System.exit(1);
            }
            else{
                verify = true;
            }
        }
        else{
            while( !current.containsKey(variable) &&
itr.hasNext() ){
                current = (Hashtable)itr.next();
                if(current.containsKey(variable)){
                    Variable temp2 =
(Variable)current.get(variable);
                    String type2 = temp2.type;
                    if((type2.equals("in")) ||
(type2.equals("out"))){
                        System.err.println("Cannot
pass a portmap identifier to a function: "+variable);
                        System.exit(1);
                    }
                    else{
                        verify = true;
                    }
                }
            }
        }
    }
    if (verify == false){
        System.err.println("Error: Variable
"+variable+" has not been defined and therefore cannot be passed to a
function.");
        System.exit(1);
    }
}
c = c.getNextSibling();
}

}

//check output ports when functions are called within functions
//now check whether the function's input port and output port
mappings are a subset of the tasks'
for(int j=0; j < 3; j++){
    String tempFuncIn = tempFunc2.inports[j];
    if(tempFuncIn != null){

```

```

        boolean match = false;
        for(int m=0; m < 3; m++){
            String tempTaskIn = f.inports[m];
            if(tempFuncIn.equals(tempTaskIn)){
                match = true;
                break;
            }
        }
        if (!match){
            System.err.println("Mismatched inport mappings
between function: "+tempFunc2.name+" and task: "+f.name);
            System.exit(1);
        }
    }
}

//now check the output mappings
for(int n=0; n < 3; n++){
    String tempFuncIn = tempFunc2.outports[n];
    if(tempFuncIn != null){
        boolean match2 = false;
        for(int q=0; q < 3; q++){
            String tempTaskIn = f.outports[q];
            if(tempFuncIn.equals(tempTaskIn)){
                match2 = true;
                break;
            }
        }
        if (!match2){
            System.err.println("Mismatched outport mappings
between function: "+tempFunc2.name+" and task: "+f.name);
            System.exit(1);
        }
    }
}

}

else{
    System.err.println("Error: Undelcared function: "+a.getText());
    System.exit(1);
}

})

;

parseVariableDeclaration [Function f]
: #(a:VARIABLEDECLARATION
{
//first get type and name
AST b = a.getFirstChild();
String type = b.getText();
String variable = b.getNextSibling().getText();

//make a new Variable with the type and name and see if
//it's already in the context symbol table.  If it is, throw an
//error, if not insert into context
Variable v = new Variable(type, variable);

```

```

    Hashtable t = (Hashtable)f.symbolTable.getFirst();

    //if variable is already declared
    if(t.containsKey(variable)){
        System.err.println("Error: Variable "+variable+" already
declared");
        System.exit(1);
    }
    else
        t.put(variable, v);
    }
)
;

parseVariableAssignment [Function f]
: #(VARIABLEASSIGNMENT a:EQUALS
{
//get the three children which are type, variable name, and
//value
AST b = a.getFirstChild();
String type = b.getText();
b = b.getNextSibling();
String name = b.getText();
b = b.getNextSibling();
String value = b.getText();

//get the current context hashtable
Hashtable current = (Hashtable)f.symbolTable.getFirst();

//check to see if the type matches the value
if(type.equals("int")){
    //if int is being assigned to another variable, then check
    //the variables type
    if(b.getType() == IDENTIFIER){

        //get the current context hashtable

        ListIterator itr = f.symbolTable.listIterator(0);

        boolean verify = false;
        while( !current.containsKey(value) && itr.hasNext() ){
            current = (Hashtable)itr.next();
            if(current.containsKey(value)){
                verify = true;
                break;
            }
        }
        if(current.containsKey(value)){
            verify = true;
        }
        if (verify == false){
            System.err.println("Error: Variable "+value+" has not
been defined");
            System.exit(1);
        }
    }
}
}

```

```

        else if(value.equals("true")||value.equals("false")){
            System.err.println("Error: Cannot assign a boolean
value to an int");
            System.exit(1);
        }
    }
    if(type.equals("const")){
        //if variable is of type const int, it can only be assigned
        //to a number
        if(b.getType() != NUMBER){
            System.err.println("Error: Cannot assign anything but
a constant to a const int");
            System.exit(1);
        }
    }
    if(type.equals("time")){
        if(b.getType() != NUMBER){
            System.err.println("Error: Cannot assign a non number
to a time variable");
            System.exit(1);
        }
        else if ((b.getType() == NUMBER) &&
(Integer.parseInt(b.getText()) < 0)){
            System.err.println("Error: Cannot assign a negative
number to a time variable");
            System.exit(1);
        }
    }
    if(type.equals("bool")){
        if(!value.equals("true") && !value.equals("false")){
            System.err.println("Error: Cannot assign a non
boolean value to a bool");
            System.exit(1);
        }
    }

    //check to see if that variable name has already been used in
this
    //context. If not, then make a new variable
    if( ((Hashtable)f.symbolTable.getFirst()).containsKey(name) ){
        System.err.println("Error: Cannot reassign variable
"+name);
        System.exit(1);
    }
    else{
        ((Hashtable)f.symbolTable.getFirst()).put(name, new
Variable(type, name));
    }
}
)
;

parseVariableDeclaredAssignment [Function f]
:#{a:VARIABLEDECLAREDASSIGNMENT
{

```

```

AST b = a.getFirstChild();
String variable = b.getFirstChild().getText();
AST c = b.getFirstChild().getNextSibling();
String value = c.getText();

//get the current context hashtable
Hashtable current = (Hashtable)f.symbolTable.getFirst();

ListIterator itr = f.symbolTable.listIterator(0);

boolean verify = false;
while( !current.containsKey(variable) && itr.hasNext() ){
    current = (Hashtable)itr.next();
    if(current.containsKey(variable)){
        verify = true;
        break;
    }
}
if(current.containsKey(variable)){
    verify = true;
}
if (verify == false){
    System.err.println("Error: Variable "+variable+" has
not been defined");
    System.exit(1);
}

else{
String hashtype = ((Variable)current.get(variable)).type;

//check to see if the previously declared type matches the value
if(hashtype.equals("int")){
    //if int is being assigned to another variable, then check
    //the variables type
    if(c.getType() == IDENTIFIER){
        //get the current context hashtable
        current = (Hashtable)f.symbolTable.getFirst();

        ListIterator itr2 = f.symbolTable.listIterator(0);

        boolean verify2 = false;
        while( !current.containsKey(value) && itr2.hasNext()
){
            current = (Hashtable)itr2.next();
            if(current.containsKey(value)){
                verify2 = true;
                break;
            }
        }
        if(current.containsKey(value)){
            verify = true;
        }

        if (verify2 == false){
            System.err.println("Error: Variable "+value+" has not
been defined");

```



```

        :#(a:"main" parseRCXInstantiations parseRCXAddTaskList)
        ;

parseRCXInstantiations
    :#(a:RCXINSTANTIATIONS
    {
    //number of instantiations
    int children = a.getNumberOfChildren();

    //name of the class and variable
    String variable = "";

    //boolean used to see if the rcx class was found
    boolean found = false;

    //get the array of declared RCXs
    Object[] rcxs = RCXs.toArray();

    //get the RCXInstantiation
    AST inst = a.getFirstChild();
    String className = inst.getFirstChild().getText();

    //iterate through number of instantiations
    for(int j =0; j < children ; j++){
        if(j !=0){
            inst = inst.getNextSibling();
            className = inst.getFirstChild().getText();
        }
        //iterate through declared rcxs
        for(int i = 0; i < rcxs.length ; i++){
            if( ((RCX)rcxs[i]).className.equals(className) ){
                found = true;
                variable =
inst.getFirstChild().getNextSibling().getText();
                Main.add(new MainRCX(variable, (RCX)rcxs[i]));
            }
        }
        //if the class name of the RCX that you're trying to
        instantiate is not
        //defined, then throw an error
        if(!found){
            System.err.println("Error: Cannot instantiate an RCX
that has not been defined");
            System.exit(1);
        }
        found = false;

    }
    }
    )
    ;

parseRCXAddTaskList
    :#(a:RCXADDTASKLIST
    {int children = a.getNumberOfChildren();

```



```

Object[] mainRCX = Main.toArray();

//get each child and check its rcx name and the task that's being
added
AST b = a.getFirstChild();

for(int i =0; i <children; i++)
{
    boolean foundRCX = false;
    String rcxVar = b.getFirstChild().getText();

    String addedTask = null;
    //iterate through the rcx's that have been declared and see
if it matches
    String retrievedName = null;

    for(int j = 0; j < mainRCX.length; j++)
    {
        retrievedName = ((MainRCX)mainRCX[j]).variable;

        if(retrievedName.equals(rcxVar) && !foundRCX){
            foundRCX = true;

            parseTaskIdentifierList(b.getFirstChild().getNextSibling(), (MainR
CX)mainRCX[j] );
        }
        if(!foundRCX){
            System.err.println("Error: Cannot add a task to an RCX:
"+retrievedName+" that has not been defined");
            System.exit(1);
        }
        b = b.getNextSibling();
    }
}
)
;

parseTaskIdentifierList [MainRCX rcx]
: #(a:TASKIDENTIFIERLIST
{
//get the number of tasks that are trying to be added
int children = a.getNumberOfChildren();

//get the first child to start traversing
AST b = a.getFirstChild();
String child = b.getText();

//grab the already declared tasks
Object[] tasks = Tasks.toArray();

//iterate for each child
for(int i = 0; i < children; i++){
boolean foundTask = false;
int index = 0;
//iterate through the tasks looking for child

```

```

    for(int k =0; k < tasks.length; k++){
        //if we find a task with the specified name
        if(((Task)tasks[k]).name.equals(child)){
            foundTask = true;
            index = k;

            //found the task, now check that the ports match
            //task port mappings must represent a subset of rcx
port mappings

            //check inport mappings

            for(int j=0; j < 3; j++){
                String tempFuncIn =
((Task)tasks[k]).inports[j];

                if(tempFuncIn != null){
                    boolean match = false;
                    String tempTaskIn = rcx.rcx.port1;

                    if(tempFuncIn.equals(tempTaskIn)){
                        match = true;

                    }
                    tempTaskIn = rcx.rcx.port2;
                    if(tempFuncIn.equals(tempTaskIn)){
                        match = true;

                    }
                    tempTaskIn = rcx.rcx.port3;
                    if(tempFuncIn.equals(tempTaskIn)){
                        match = true;

                    }

                    if (!match){
                        System.err.println("Mismatched inport
mappings between RCX: "+rcx.rcx.className+" and task:
"+((Task)tasks[index]).name);
                        //System.exit(1);
                    }
                }
            }

            //check outport mappings

            for(int m=0; m < 3; m++){
                String tempFuncIn =
((Task)tasks[k]).outports[m];

                if(tempFuncIn != null){
                    boolean match = false;
                    String tempTaskIn = rcx.rcx.portA;
                    if(tempFuncIn.equals(tempTaskIn)){
                        match = true;

                    }
                }
            }

```

```

        tempTaskIn = rcx.rcx.portB;
        if(tempFuncIn.equals(tempTaskIn)){
            match = true;
        }
        tempTaskIn = rcx.rcx.portC;
        if(tempFuncIn.equals(tempTaskIn)){
            match = true;
        }

        if (!match){
            System.err.println("Mismatched outpost
mappings between RCX: "+rcx.rcx.className+" and task:
"+((Task)tasks[index]).name);
            //System.exit(1);
        }
        }
        }

        //add the task to the vector
        rcx.addTask(((Task)tasks[index]).name);
    }
    if(!foundTask){
        System.err.println("Error: Cannot add an undeclared
task to an rcx");
        System.exit(1);
    }

    b = b.getNextSibling();
    if(b !=null)
        child = b.getText();
    }
    })
    ;

setTask: #(TASKDECLARATIONLIST (#(TASK
//for each task make a new task
{ Task t = new Task();}
parseTaskDeclaration[t] parseTaskBody[t]))*)
;

parseTaskDeclaration [Task t]
: #(TASKDECLARATION a:IDENTIFIER
{t.name = a.getText();}
setTaskInList[t] setTaskOutList[t]
{t.loadPorts();}
Tasks.add(t);}
)
;

setTaskInList [Task t]
: #(a:INLIST
{if((a.getNumberOfChildren() > 0) && (a.getNumberOfChildren() <
4) )

```

```

    {
        AST b = a.getFirstChild();
        for(int i = 0; i < a.getNumberOfChildren(); i++)
        {
            t.inports[i] = b.getText();
            b = b.getNextSibling();
        }
    }
)
;

setTaskOutList [Task t]
: # (a:OUTLIST
4) )
{
    AST b = a.getFirstChild();
    for(int i = 0; i < a.getNumberOfChildren(); i++)
    {
        t.outports[i] = b.getText();
        b = b.getNextSibling();
    }
}
)
;

parseTaskBody [Task t]
: # (a:TASKBODY
{
int children = a.getNumberOfChildren();
if(children > 0){
    AST b = a.getFirstChild();
    for (int i=0; i < children; i++){

        String temp = b.getText();
        if(temp.equals("when"))
            parseWhenCondition(b, t);
        else if(temp.equals("if"))
            parseIfTask(b, t);
        else if(temp.equals("do"))
            parseDoTask(b, t);
        else if(temp.equals("while"))
            parseWhileBlockTask(b, t);
        else if(temp.equals("variableDeclaredAssignment"))
            parseVariableDeclaredAssignmentTask(b, t);
        else if(temp.equals("variableAssignment"))
            parseVariableAssignmentTask(b, t);
        else if(temp.equals("variableDeclaration"))
            parseVariableDeclarationTask(b, t);
        else if(temp.equals("passMessage"))
            parsePassMessage(b);
        else if(temp.equals("receiveMessage"))
            parseReceiveMessage(b);
        else if(temp.equals("functionCall"))

```

```

        parseFunctionCallTask(b, t);
    else if(temp.equals("nqcFunctionCall"))
        parseNQCFunctionCallTask(b, t);
    else if(temp.equals("throwMessage"))
        parseThrowMessage(b);
    else if(temp.equals("catchMessage"))
        parseCatchMessage(b);

    b = b.getNextSibling();
}
}
})
;

```

```

parseNQCFunctionCallTask [Task t]
: #(a:NQCFUNCTIONCALL
{
AST b = a.getFirstChild();
AST params = b.getNextSibling();
int numParams = params.getNumberOfChildren();

String func = b.getText();
NQC tempNQC = t.nqcFunctionList;
String [] tempFunctions = tempNQC.functions;
int length = tempFunctions.length;
boolean found = false;
int index = 0;
for(int i=0; i < length; i++){
    String checkFunc = tempFunctions[i];
    if(func.equals(checkFunc)){
        found = true;
        index = i;
        break;
    }
}
if(found == false){
    System.err.println("Attempting to call a non NQC function
as a NQC function: "+func+" in function: "+t.name);
    System.exit(1);
}

//must now verify the function
//On function
if(((index >= 0) && (index < 8)) || (index == 11))
    parseNQCFunctionRegularTask(params, t, index);

if(((index == 10) || (index == 13) || (index == 15)) &&
(numParams == 0)){
    parseNQCFunctionNoParamTask(params, t, index);
}

if(((index == 10) || (index == 13) || (index == 15)) &&
(numParams != 0)){

```

```

        System.err.println("Cannot pass a paramter with NQC
function: "+t.nqcFunctionList.functions[index]+"() in task: "+t.name);
        System.exit(1);
    }
    if(index == 8){
        parseNQCFunctionPlaySoundTask(params, t, index);
    }

    if(index == 9){
        parseNQCFunctionParam1Task(params, t, index);
    }
    if(((index == 12) || (index == 14))){
        int child = params.getNumberOfChildren();
        if(child != 1){
            System.err.println("Must pass exactly one parameter
to NQC function: "+t.nqcFunctionList.functions[index]+"() in task:
"+t.name);
            System.exit(1);
        }
        AST param2 = params.getFirstChild();
        int type = param2.getType();
        if(type == NUMBER)
            parseNQCFunctionParam2Task(params, t, index);
        if(type != NUMBER){
            if(index == 12){
                System.err.println("Cannot pass NQC function
Wait() with variable: "+param2.getText()+" in task: "+t.name+". Must
pass an 'int'");
                System.exit(1);
            }
            if(index == 14){
                System.err.println("Cannot pass NQC function
SetSleepTime() variable: "+param2.getText()+" in task: "+t.name+".
Must pass an 'int'");
                System.exit(1);
            }
        }
    }
}

})
;

```

```

parseNQCFunctionParam2Task [Task t, int index]
: # (a: NQC PARAMETERS2
{
    int children = a.getNumberOfChildren();
    if(children != 1){
        System.err.println("Error: Can only pass one parameter to
NQC function: "+t.nqcFunctionList.functions[index]+"() in task:
"+t.name);
        System.exit(1);
    }
    int type = a.getFirstChild().getType();

```

```

        if(!(type == NUMBER)){
            System.err.println("Error: Can only pass an int to NQC
function: "+t.nqcFunctionList.functions[index]+"() in task: "+t.name);
            System.exit(1);
        }
    });

//takes 2 number parameters
parseNQCFunctionParam1Task [Task t, int index]
    :#(a:NQCPARAMETERS1
    {

        int children = a.getNumberOfChildren();

        if(children != 3){
            System.err.println("Error: Must pass two parameters to NQC
function PlayTone() in task: "+t.name);
            System.exit(1);
        }
        int temp = a.getFirstChild().getType();
        if(!(temp==NUMBER)){
            System.err.println("Error: Parameter:
"+a.getFirstChild().getText()+" in NQC function:
"+t.nqcFunctionList.functions[index]+" in task: "+t.name+" must be of
type int");
            System.exit(1);
        }
        AST intermediate = a.getFirstChild().getNextSibling();
        AST b = intermediate.getNextSibling();
        int temp2 = b.getType();
        if(!(temp2==NUMBER)){
            System.err.println("Error: Parameter: "+b.getText()+" in
NQC function: "+t.nqcFunctionList.functions[index]+" in task:
"+t.name+" must be of type int");
            System.exit(1);
        }
    });

//looking for one parameter, of type keyword for sound
parseNQCFunctionPlaySoundTask [Task t, int index]
    :#(a:NQCPARAMETERS
    {

        int children = a.getNumberOfChildren();
        if(children != 1){
            System.err.println("Error: Must list one parameter for NQC
function: "+ t.nqcFunctionList.functions[index]+"() in task: "+t.name);
            System.exit(1);
        }
        String sound = a.getFirstChild().getText();

        boolean check = false;
        if((sound.equals("SOUND_CLICK")) ||
(sound.equals("SOUND_DOUBLE_BEEP")) || (sound.equals("SOUND_DOWN")) ||

```

```

(sound.equals("SOUND_UP")) || (sound.equals("SOUND_LOW_BEEP")) ||
(sound.equals("SOUND_FAST_UP"))){
    check = true;
}
if(!check){
    System.err.println("Error: Incorrect parameter: "+sound+"
in NQC function PlaySound in task: "+t.name+". Use: SOUND_CLICK,
SOUND_DOUBLE_BEEP, SOUND_DOWN, SOUND_UP, SOUND_LOW_BEEP,
SOUND_FAST_UP");
    System.exit(1);
}
});

//nqc function with no parameters
parseNQCFunctionNoParamTask [Task t, int index]
:#{a:NQCPARAMETERS
{

    int children = a.getNumberOfChildren();
    if(children > 0){
        System.err.println("Error: NQC function:
"+t.nqcFunctionList.functions[index]+" in task: "+t.name+" does not
take any parameters");
        System.exit(1);
    }
})
;

//reads in a bunch of output ports
parseNQCFunctionRegularTask [Task t, int index]
:#{a:NQCPARAMETERS
{

    int children = a.getNumberOfChildren();
    if((children ==1) && (index == 11)){
        boolean check = false;
        String temp = a.getFirstChild().getText();
        if( (temp.equals("TX_POWER_LO")) ||
(temp.equals("TX_POWER_HI"))){
            check = true;
        }
        if(!check){
            System.err.println("Error: Parameter:
"+a.getFirstChild().getText()+" in task: "+t.name+" must be
TX_POWER_LOW or TX_POWER_HI");
            System.exit(1);
        }
    }
else{

        if((children < 1) || (children > 5)){
            System.err.println("Error: incorrect number of parameters
for NQC function: "+t.nqcFunctionList.functions[index]+"() in task:
"+t.name+". Must have at least one or up to three output ports");
            System.exit(1);
        }
    }
}
}
;

```



```

    }
    AST temp = a.getFirstChild();
    for(int i=0; i < children; i++){
        String outPortPassed = temp.getText();
        if(!(temp.getType() == PLUS)){
            boolean verify = false;
            for(int j=0; j < 3; j++){
                if(t.outports[j] != null){
                    String tempOutPort = t.outports[j];
                    if(outPortPassed.equals(tempOutPort)){
                        verify = true;
                    }
                }
            }

            if(!verify){
                System.err.println("Error: attempting to use
undeclared output port: "+outPortPassed);
                System.exit(1);
            }
        }
        temp = temp.getNextSibling();
    }
}
})
;

parseWhileConditionTask [Task t]
: #(a: WHILECONDITION
{
int numConditions = a.getNumberOfChildren();

AST condition = a.getFirstChild();

for(int i = 0; i < numConditions; i++){
    boolean verify = false;
    if((condition.getNumberOfChildren()) != 0)
    {

        String variable = condition.getFirstChild().getText();
        ListIterator itr = t.symbolTable.listIterator(0);
        Hashtable current = (Hashtable)t.symbolTable.getFirst();

        if(current.containsKey(variable))
            verify = true;
        else{while( !current.containsKey(variable) && itr.hasNext()
){
            current = (Hashtable)itr.next();
            if(current.containsKey(variable)){
                verify = true;
                break;
            }
        }
        if (verify == false){
            System.err.println("Error: Variable "+variable+" has
not been defined");

```

```

        }
    }
    condition = condition.getNextSibling();
}

});

parseWhileBlockTask [Task t]

:#{a:"while" {
    Hashtable h = new Hashtable();
    t.symbolTable.addFirst(h);
    AST whileCondition = a.getFirstChild();
    AST whileStatements = whileCondition.getNextSibling();

    int numConditions = whileCondition.getNumberOfChildren();

    AST condition = whileCondition.getFirstChild();

    parseWhileConditionTask(whileCondition, t);

    //whileStatement Checking
    int children = whileStatements.getNumberOfChildren();
    AST b = whileStatements.getFirstChild();
    for (int j = 0; j < children; j++){
        String temp = b.getText();
        if(temp.equals("while"))
            parseWhileBlockTask(b, t);
        else if(temp.equals("do"))
            parseDoTask(b, t);
        else if(temp.equals("if"))
            parseIfTask(b, t);
        else if(temp.equals("variableAssignment"))
            parseVariableAssignmentTask(b, t);
        else if(temp.equals("variableDeclaredAssignment"))
            parseVariableDeclaredAssignmentTask(b, t);
        else if(temp.equals("variableDeclaration"))
            parseVariableDeclarationTask(b, t);
        else if(temp.equals("functionCall"))
            parseFunctionCallTask(b, t);
        else if(temp.equals("nqcFunctionCall"))
            parseNQCFunctionCallTask(b, t);
        else if(temp.equals("passMessage"))
            parsePassMessage(b);
        else if(temp.equals("receiveMessage"))
            parseReceiveMessage(b);
        else if(temp.equals("throwMessage"))
            parseThrowMessage(b);
        else if(temp.equals("catchMessage"))
            parseCatchMessage(b);
        b = b.getNextSibling();
    }

    t.symbolTable.removeFirst();
});

```

```

parseDoTask [Task t]
  :#(a:"do"
  {
  //add a new context
  Hashtable h = new Hashtable();
  t.symbolTable.addFirst(h);

  int children = a.getNumberOfChildren();
  AST b = a.getFirstChild();
  for(int i =0; i < children; i++){
    String temp = b.getText();
    if(temp.equals("while"))
      parseWhileBlockTask(b, t);
    else if(temp.equals("do"))
      parseDoTask(b, t);
    else if(temp.equals("if"))
      parseIfTask(b, t);
    else if(temp.equals("variableAssignment"))
      parseVariableAssignmentTask(b, t);
    else if(temp.equals("variableDeclaredAssignment"))
      parseVariableDeclaredAssignmentTask(b, t);
    else if(temp.equals("variableDeclaration"))
      parseVariableDeclarationTask(b, t);
    else if(temp.equals("functionCall"))
      parseFunctionCallTask(b, t);
    else if(temp.equals("nqcFunctionCall"))
      parseNQCFunctionCallTask(b, t);
    else if(temp.equals("whileCondition"))
      parseWhileConditionTask(b, t);
    else if(temp.equals("passMessage"))
      parsePassMessage(b);
    else if(temp.equals("receiveMessage"))
      parseReceiveMessage(b);
    else if(temp.equals("throwMessage"))
      parseThrowMessage(b);
    else if(temp.equals("catchMessage"))
      parseCatchMessage(b);
    b = b.getNextSibling();
  }
  t.symbolTable.removeFirst();
  })
  ;

```

```

parseIfTask [Task t]
  : #(a:"if" b:IFCONDITION
  {
  parseIfConditionTask(b, t);

  //add a new context
  Hashtable h = new Hashtable();
  t.symbolTable.addFirst(h);

  int children = a.getNumberOfChildren();
  boolean done = false;

```

```

//if there is actually something inside the if like standard
//statements then we traverse them. else do nothing
if(children > 1){
    b = b.getNextSibling();
    for(int i =0; i < children-1; i++){
        String temp = b.getText();
        if(temp.equals("while"))
            parseWhileBlockTask(b, t);
        else if(temp.equals("do"))
            parseDoTask(b, t);
        else if(temp.equals("if"))
            parseIfTask(b, t);
        else if(temp.equals("variableAssignment"))
            parseVariableAssignmentTask(b, t);
        else if(temp.equals("variableDeclaredAssignment"))
            parseVariableDeclaredAssignmentTask(b, t);
        else if(temp.equals("variableDeclaration"))
            parseVariableDeclarationTask(b, t);
        else if(temp.equals("functionCall"))
            parseFunctionCallTask(b, t);
        else if(temp.equals("nqcFunctionCall"))
            parseNQCFunctionCallTask(b, t);
        else if(temp.equals("else")){
            t.symbolTable.removeFirst();
            done=true;
            parseElseTask(b, t);
        }
        else if(temp.equals("passMessage"))
            parsePassMessage(b);
        else if(temp.equals("receiveMessage"))
            parseReceiveMessage(b);
        else if(temp.equals("throwMessage"))
            parseThrowMessage(b);
        else if(temp.equals("catchMessage"))
            parseCatchMessage(b);

        b = b.getNextSibling();
    }
}
if(!done){t.symbolTable.removeFirst();}

}
)
;

```

```

parseStatementListTask [Task t]
    : #(a:STATEMENTLIST
    {
    int children = a.getNumberOfChildren();
    if(children > 0){
        AST b = a.getFirstChild();
        for (int i=0; i < children; i++){
            String temp = b.getText();
            if(temp.equals("when"))
                parseWhenCondition(b, t);

```

```

        else if(temp.equals("if"))
            parseIfTask(b, t);
        else if(temp.equals("do"))
            parseDoTask(b, t);
            else if(temp.equals("while"))
                parseWhileBlockTask(b, t);
        else if(temp.equals("variableDeclaredAssignment"))
            parseVariableDeclaredAssignmentTask(b, t);
        else if(temp.equals("variableAssignment"))
            parseVariableAssignmentTask(b, t);
        else if(temp.equals("variableDeclaration"))
            parseVariableDeclarationTask(b, t);
        else if(temp.equals("passMessage"))
            parsePassMessage(b);
        else if(temp.equals("receiveMessage"))
            parseReceiveMessage(b);
    else if(temp.equals("functionCall"))
        parseFunctionCallTask(b, t);
        else if(temp.equals("nqcFunctionCall"))
            parseNQCFunctionCallTask(b, t);
    else if(temp.equals("throwMessage"))
        parseThrowMessage(b);
    else if(temp.equals("catchMessage"))
        parseCatchMessage(b);

        b = b.getNextSibling();
    }
}
})
;

parseElseTask [Task t]
: #(a:"else" b:STATEMENTLIST
{parseStatementListTask(b, t);}
)
;

parseIfConditionTask [Task t]
: #(a:IFCONDITION
{
int numConditions = a.getNumberOfChildren();
AST condition = a.getFirstChild();

for(int i = 0; i < numConditions; i++){
    boolean verify = false;
    if((condition.getNumberOfChildren() != 0)
    {

String variable = condition.getFirstChild().getText();
ListIterator itr = t.symbolTable.listIterator(0);
Hashtable current = (Hashtable)t.symbolTable.getFirst();

if(current.containsKey(variable))
    verify = true;
else{while( !current.containsKey(variable) && itr.hasNext()
) {

```

```

        current = (Hashtable)itr.next();
        if(current.containsKey(variable)){
            verify = true;
            break;
        }
    }
    if (verify == false){
        System.err.println("Error: Variable "+variable+" has
not been defined");
    }
}
condition = condition.getNextSibling();
}

});

```

parsePassMessage

```

: #(a:PASSMESSAGE
{
AST intermediate = a.getFirstChild();
AST c = (AST)intermediate.getNextSibling();
String messageName = c.getText();
int messageVecSize = Messages.size();
boolean check = false;
for(int i=0; i < messageVecSize; i++){
    String tempString = (String)Messages.elementAt(i);
    if(tempString.equals(messageName)){
        check = true;
        break;
    }
}
if(!check){
    System.err.println("Message identifier: "+messageName+"
:undefined");
    System.exit(1);
}
});

```

parseReceiveMessage

```

: #(a:RECEIVEMESSAGE
{
AST intermediate = a.getFirstChild();
AST c = (AST)intermediate.getNextSibling();
String messageName = c.getText();
int messageVecSize = Messages.size();
boolean check = false;
for(int i=0; i < messageVecSize; i++){
    String tempString = (String)Messages.elementAt(i);
    if(tempString.equals(messageName)){
        check = true;
        break;
    }
}
if(!check){
    System.err.println("Message identifier: "+messageName+"
:undefined");
}

```

```

        System.exit(1);
    }
});

parseVariableDeclarationTask [Task t]
: #(a:VARIABLEDECLARATION
{
//first get type and name
AST b = a.getFirstChild();
String type = b.getText();
AST c = b.getNextSibling();
String variable = c.getText();

//make a new Variable with the type and name and see if
//it's already in the context symbol table.  If it is, throw an
//error, if not insert into context
Variable v = new Variable(type, variable);
Hashtable h = (Hashtable)t.symbolTable.getFirst();

//if variable is already declared
if(h.containsKey(variable)){
    System.err.println("Error: Variable "+variable+" already
declared");
    System.exit(1);
}
else
    h.put(variable, v);
}
)
;

parseVariableAssignmentTask [Task t]
: #(VARIABLEASSIGNMENT a:EQUALS
{
//get the three children which are type, variable name, and
//value
AST b = a.getFirstChild();
String type = b.getText();
b = b.getNextSibling();
String name = b.getText();
b = b.getNextSibling();
String value = b.getText();

//get the current context hashtable
Hashtable current = (Hashtable)t.symbolTable.getFirst();

//check to see if the type matches the value
if(type.equals("int")){
    //if int is being assigned to another variable, then check
    //the variables type
    if(b.getType() == IDENTIFIER){

        //get the current context hashtable
        ListIterator itr = t.symbolTable.listIterator(0);

```

```

        boolean verify = false;
        while( !current.containsKey(value) && itr.hasNext() ){
            current = (Hashtable)itr.next();
            if(current.containsKey(value)){
                verify = true;
                break;
            }
        }
        if(current.containsKey(value)){
            verify = true;
        }
        if (verify == false){
            System.err.println("Error: Variable "+value+" has not
been defined");
            System.exit(1);
        }

        else if(value.equals("true")||value.equals("false")){
            System.err.println("Error: Cannot assign a boolean
value to an int");
            System.exit(1);
        }
    }
    if(type.equals("const")){
        //if variable is of type const int, it can only be assigned
        //to a number
        if(b.getType() != NUMBER){
            System.err.println("Error: Cannot assign anything but
a constant to a const int");
            System.exit(1);
        }
    }
    if(type.equals("time")){
        if(b.getType() != NUMBER){
            System.err.println("Error: Cannot assign a non number
to a time variable");
            System.exit(1);
        }
    }
    if(type.equals("bool")){
        if(!value.equals("true") && !value.equals("false")){
            System.err.println("Error: Cannot assign a non
boolean value to a bool");
            System.exit(1);
        }
    }
}

//check to see if that variable name has already been used in
this
//context. If not, then make a new variable
if( ((Hashtable)t.symbolTable.getFirst()).containsKey(name) ){
    System.err.println("Error: Cannot reassign variable
"+name);
    System.exit(1);
}
else{

```



```

        ((Hashtable)t.symbolTable.getFirst()).put(name, new
Variable(type, name));
    }

}
)
;

parseVariableDeclaredAssignmentTask [Task t]
: # (a: VARIABLEDECLAREDASSIGNMENT
{
AST b = a.getFirstChild();
String variable = b.getFirstChild().getText();
AST c = b.getFirstChild().getNextSibling();
String value = c.getText();

//get the current context hashtable
Hashtable current = (Hashtable)t.symbolTable.getFirst();

ListIterator itr = t.symbolTable.listIterator(0);

boolean verify = false;
while( !current.containsKey(variable) && itr.hasNext() ){
    current = (Hashtable)itr.next();
    if(current.containsKey(variable)){
        verify = true;
        break;
    }
}
if(current.containsKey(variable)){
    verify = true;
}
if (verify == false){
    System.err.println("Error: Variable "+variable+" has
not been defined");
    System.exit(1);
}

else{
String hashtype = ((Variable)current.get(variable)).type;

//check to see if the previously declared type matches the value
if(hashtype.equals("int")){
    //if int is being assigned to another variable, then check
    //the variables type
    if(c.getType() == IDENTIFIER){
        //get the current context hashtable
        current = (Hashtable)t.symbolTable.getFirst();

        ListIterator itr2 = t.symbolTable.listIterator(0);

        boolean verify2 = false;
        while( !current.containsKey(value) && itr2.hasNext()
){
            current = (Hashtable)itr2.next();
            if(current.containsKey(value)){

```



```

    }
    )
    ;

parseFunctionCallTask [Task t]
: #(FUNCTIONCALL a:IDENTIFIER b:FUNCTIONPARAMSPASSING
{
//check to see if the function exists
boolean found = false;
Object[] fa = Funcs.toArray();
for(int i = 0; i < fa.length ; i++){
    if( ((Function)fa[i]).name.equals(a.getText()) )
        found = true;
}
if(found){

//check for matching number of parameters
//now check for matching number of parameters
    int index = 0;
    int funcSize = Funcs.size();
    for(int i=0; i < funcSize; i++){
        Function tempFunc = (Function)Funcs.elementAt(i);
        String functionName = tempFunc.name;
        if(a.equals(functionName)){
            index = i;
        }
    }

    int numParams = b.getNumberOfChildren();

    Function tempFunc2 = (Function)Funcs.elementAt(index);
    int checkParam = tempFunc2.counter;

    if(numParams != checkParam){
        System.err.println("Error: mismatched parameters
function: "+a+" task: "+t.name);
        System.exit(1);
    }

//check the parameters that are being passed
    int children = b.getNumberOfChildren();

    if(children != 0){
        AST c = b.getFirstChild();

//iterate through each child and check that the parameter has
already
//been defined
        for(int i=0; i < children; i++){
            if(c.getType() == IDENTIFIER){
                boolean verify = false;
                String variable = c.getText();

//finding parameter names
                ListIterator itr = t.symbolTable.listIterator(0);

```

```

        Hashtable current =
(Hashtable)t.symbolTable.getFirst();

        if(current.containsKey(variable)){
            Variable temp =
(Variable)current.get(variable);
            String type = temp.type;

            if((type.equals("in") ||
(type.equals("out"))){
                System.err.println("Cannot pass a portmap
identifier to a function: "+variable);
                System.exit(1);
            }
        }
        else{
            verify = true;
        }

        if (verify == false){
            System.err.println("Error: Variable
"+variable+" has not been defined and therefore cannot be passed to a
function.");
            System.exit(1);
        }
    }
    c = c.getNextSibling();
}

//now check whether the function's input port and output port
mappings are a subset of the tasks'
for(int j=0; j < 3; j++){
    String tempFuncIn = tempFunc2.inports[j];
    if(tempFuncIn != null){
        boolean match = false;
        for(int m=0; m < 3; m++){
            String tempTaskIn = t.inports[m];
            if(tempFuncIn.equals(tempTaskIn)){
                match = true;
                break;
            }
        }
        if (!match){
            System.err.println("Mismatched inport mappings
between function: "+tempFunc2.name+" and task: "+t.name);
            System.exit(1);
        }
    }
}

//now check the output mappings
for(int n=0; n < 3; n++){
    String tempFuncIn = tempFunc2.outports[n];
    if(tempFuncIn != null){

```

```

        boolean match2 = false;
        for(int q=0; q < 3; q++){
            String tempTaskIn = t.outports[q];
            if(tempFuncIn.equals(tempTaskIn)){
                match2 = true;
                break;
            }
        }
        if (!match2){
            System.err.println("Mismatched outport mappings
between function: "+tempFunc2.name+" and task: "+t.name);
            System.exit(1);
        }
    }
}

}

}
else{
System.err.println("Error: Undelcared function: "+a.getText());
System.exit(1);
}

})
;

parseWhenCondition [Task t]
: #(b:"when" a:WHENCONDITION
{
    int numConditions = a.getNumberOfChildren();
    //grab the first condition
    AST condition = a.getFirstChild();
    //grab the do part of the when statement
    AST whenStatementDo = a.getNextSibling();

    //iterate through the when conditions checking if the inport
mapping
//is an identifier and if it has been initialized

    for(int i = 0; i < numConditions; i++){
        boolean verify = false;
        if((condition.getNumberOfChildren() != 0)
        {

            //find string representation of input port mapping
            String variable = condition.getFirstChild().getText();

            //iterate through all previous context's looking to see
//if the variable has been used
            ListIterator itr = t.symbolTable.listIterator(0);
            Hashtable current = (Hashtable)t.symbolTable.getFirst();
            if(current.containsKey(variable)){
                Variable tempVar = (Variable)current.get(variable);

                String type = tempVar.type;

```



```

        }
    }
    if (verify == false){
        System.err.println("Error: Attempting to call
undeclared function "+variable);
        System.exit(1);
    }

    //now check for matching number of parameters

    AST intermediate = function.getFirstChild();
    AST funcParamsPass = intermediate.getNextSibling();
    int numParams = funcParamsPass.getNumberOfChildren();

    Function tempFunc = (Function)Funcs.elementAt(index);

    if(numParams != 0){
        System.err.println("Should not pass a function with
parameters to a when condition. Function: "+variable);
        System.exit(1);
    }

    AST c = funcParamsPass.getFirstChild();

    //iterate through each child and check that the parameter has
already
//been defined

    for(int p=0; p < numParams; p++){
        if(c.getType() == IDENTIFIER){
            boolean verify3 = false;
            String variable2 = c.getText();

            //finding parameter names
            ListIterator itr = t.symbolTable.listIterator(0);
            Hashtable current =
(Hashtable)t.symbolTable.getFirst();

            if(current.containsKey(variable2)){
                verify3 = true;
            }
            else{
                while( !current.containsKey(variable2) &&
itr.hasNext() ){
                    current = (Hashtable)itr.next();
                    if(current.containsKey(variable2)){
                        verify3 = true;
                        break;
                    }
                }
            }
        }
    }

    if (verify3 == false){

```





```
});
```

## **Supporting Structures all created by Gerardo Flores and Michael Ching as deemed necessary.**

```
import java.util.*;

public class Function{
    //name of function
    String name;

    //possible parameters
    String[] parameters;
    int counter = 0;

    //port mappings
    String[] inports;
    String[] outports;
    NQC nqcFunctionList;
    //linked list to be used for storing contexts
    LinkedList symbolTable;
    Hashtable globalTable;

    public Function(){

        name = null;
        parameters = new String[100];
        inports = new String[3];
        outports = new String[3];

        nqcFunctionList = new NQC();
        for(int i =0; i< 100; i++)
            parameters[i] = null;

        for(int i =0; i< 3; i++)
            inports[i] = null;

        for(int i =0; i< 3; i++)
            outports[i] = null;
        //make linked list and outer hashtable to store variables global
to the
        //function...then add this hashtable to the linked list
        symbolTable = new LinkedList();
        globalTable = new Hashtable();
        symbolTable.addFirst(globalTable);
    }

    public void addParam(String a){
        parameters[counter] = a;
        counter++;
    }

    public int checkParams(){
```

```

int paramSize = parameters.length;
int counter = 0;
for(int i=0; i < paramSize; i++){
    String p = parameters[i];
    if(p != null)
        counter++;
}
return counter;
}

//method that inserts port mapping and parameters into hash table
public void loadPortsAndParams(){

    //load params into context hash table
    for(int i = 0; i< counter; i++){
        //check whether the parameter is already in the global table
        if(globalTable.containsKey(parameters[i])){
            System.err.println("Error: Variable: "+parameters[i]+" has
already been defined");
            System.exit(1);
        }
        else{
            globalTable.put(parameters[i], new
Variable("int",parameters[i]));
        }
    }

    for(int i =0; i< 3; i++){
        if(inports[i] != null){
            if(globalTable.containsKey(inports[i])){
                System.err.println("Error: Variable: "+inports[i]+" has
already been defined");
                System.exit(1);
            }
            else
            {
                globalTable.put(inports[i], new
Variable("in",inports[i]));
            }
        }
    }

    for(int i =0; i< 3; i++){
        if(outports[i] != null){
            if(globalTable.containsKey(outports[i])){
                System.err.println("Error: Variable: "+outports[i]+"
has already been defined");
                System.exit(1);
            }
            else{
                globalTable.put(outports[i], new
Variable("out",outports[i]));
            }
        }
    }
}

```

```

        }
    }
}

public void insertContext(Hashtable t){
    symbolTable.addFirst(t);
}
}

```

---

```

import java.util.*;
public class MainRCX{

    RCX rcx;
    String variable;
    Vector addedTasks;

    public MainRCX(String variable, RCX className){
        this.rcx = className;
        this.variable = variable;
        addedTasks = new Vector();
    }

    public void addTask(String t){
        addedTasks.add(t);
    }

}

```

---

```

import java.util.*;

public class NQC{
    //Vector that will hold all NQC functions
    String [] functions;

    public NQC(){
        functions = new String[16];

        functions[0] = "On";
        functions[1] = "Off";
        functions[2] = "Float";
        functions[3] = "Fwd";
        functions[4] = "Rev";
        functions[5] = "Toggle";
        functions[6] = "OnFwd";
        functions[7] = "OnRev";
        functions[8] = "PlaySound";
        functions[9] = "PlayTone";
        functions[10] = "ClearMessage";
        functions[11] = "SetTxPower";
    }
}

```

```

        functions[12] = "Wait";
        functions[13] = "StopAllTasks";
        functions[14] = "SetSleepTime";
        functions[15] = "SleepNow";
    }
}

```

---

```

public class RCX{
    //name of RCX class
    String className;

    //port mappings
    String port1;
    String port2;
    String port3;

    String portA;
    String portB;
    String portC;

    public RCX(String name, String a, String b, String c, String d,
String e, String f){
        className = name;
        port1 = a;
        port2 = b;
        port3 = c;
        portA = d;
        portB = e;
        portC = f;
    }
}

```

---

```

import java.util.*;

public class Task{
    //name of task
    String name;

    //port mappings
    String[] inports;
    String[] outports;
    String[] parameters;
    //linked list to be used for storing contexts
    LinkedList symbolTable;
    Hashtable globalTable;
    NQC nqcFunctionList;

    public Task(){
        name = null;

```

```

inports = new String[3];
outports = new String[3];
nqcFunctionList = new NQC();

for(int i =0; i< 3; i++)
    inports[i] = null;

for(int i =0; i< 3; i++)
    outports[i] = null;
//make linked list and outer hashtable to store variables global
to the
//function...then add this hashtable to the linked list
symbolTable = new LinkedList();
globalTable = new Hashtable();
symbolTable.addFirst(globalTable);
}

//method that inserts port mapping and parameters into hash table
public void loadPorts(){

    for(int i =0; i< 3; i++){
        if(inports[i] != null){
            globalTable.put(inports[i], new Variable("in",inports[i]));

        }
    }
    for(int i =0; i< 3; i++){
        if(outports[i] != null)
            globalTable.put(outports[i], new
Variable("out",outports[i]));
    }
}

public void print(){
    System.out.println("Function name "+name);
    for(int i =0; i< 3; i++){
        if(inports[i] != null)
            System.out.println("in port: "+inports[i]);
    }
    for(int i =0; i< 3; i++){
        if(outports[i] != null)
            System.out.println("out port: "+outports[i]);
    }
}
}

```

---

```

public class Variable{
    public Variable(String type, String name){
        this.name = name;
        this.type = type;
    }
    public String name;
    public String type;
}

```

---

**From this point on, all code was written by Charles O'Donnell as part of the back-end**

```
/******  
* This is the main java source for running our  
* Parser, Static-Semantic Checker, and UNI-OGEL writer  
*  
* This is copied from the -INFAMOUS- gerardo.java...  
* taking weeks to research and compile, this class represents  
* the absolute best the OGEL team has to offer in innovation,  
* complexity, completeness, and clarity  
*  
* Author: Gerardo Flores <gf141@columbia.edu>  
* Author: Charles O'Donnell <cwo4@columbia.edu>  
*/  
  
import java.io.*;  
import javax.swing.*;  
import antlr.debug.misc.ASTFrame;  
import antlr.CommonAST;  
import antlr.collections.AST;  
import antlr.DumpASTVisitor;  
  
public class ogelgen {  
  
    public static void main(String[] args){  
  
        if (args.length < 1) {  
            System.out.println("No input files given");  
            System.exit(0);  
        }  
        try {  
            OGELLexer simpleLexer = new OGELLexer(new  
FileInputStream(args[0])); //piping screws with some testing  
            OGELParser p = new OGELParser(simpleLexer);  
  
            p.file();  
            CommonAST t = (CommonAST)p.getAST();  
            OGELSemanticWalker sw = new OGELSemanticWalker();  
            sw.file(t);  
  
            OGELWriterWalker ww = new OGELWriterWalker();  
            if (args.length > 1) {  
                ww.setOutputName(args[1]);  
            }  
            ww.file(t);  
  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    } //end main()  
  
} //end class ogelgen
```

---

```

//*****
// This is the Tree Walker which reads the tree *AFTER* the static-
semantic
// checker... it will read the tree and output into our
// UNI-OGEL intermediary language
//
// Author: Charles O'Donnell <cwo4@columbia.edu>
//
//*****
header {
    import java.util.*;
    import java.io.*;
}

class OGELWriterWalker extends TreeParser;

options {
    importVocab = OGEL;
}

{

    //prefix string Stage 4 uses to determine if a subfunction is being
called from a task...
    //stage 4 will do this: when it see the string below, it
automatically deletes/etc the 'next'
    //space-delineated token...
    public final String SUBFUNC_CALL = "--FUNCCALL--";
    public final String WHEN_TOK_SEPARATOR = "~~~";
    public final String ENDLINE = "\n";
    public final int INDENT_INC = 3;

    //Name of file which will be written to with uni-ogel data
    public String output_file_name = "uni-out.uniogel";
    public void setOutputName(String out_name) { output_file_name =
out_name; }

    //Class variables
    private Map all_rcxs = new TreeMap();           //key=String type-name,
val=wRCX obj
    private Map inst_rcxs = new TreeMap();         //key=String rcx-
instance-name, val=wRCX obj
    private Set all_portmap_names = new TreeSet(); //holds 'all'
portmapping names possible

    private Map all_msgs = new TreeMap();         //key=String message-
name, val=Integer obj (msg #)
    private int global_msg_cnt = 0;

    private Map all_funcs = new TreeMap();       //key=String func-
ident, val=Func Obj

```

```

    private Map all_tasks = new TreeMap();           //key=String task-
ident, val=wTask Obj

    /**
     * This method checks to see if the input string (variable) matches
     any of the portmappings
     * -ever- found... ifso, it returns a constant mangling of the name,
     else it returns the name
     */
    String checkvar(String invar) {
        return invar;
        //special circumstance is we want true and false reserved words
to be numbers
        /*if ("true".equals(invar))
            return "1";
        else if ("false".equals(invar))
            return "0";
        else {
            String invar_bak = "".concat(invar); //ensure saved copy
            int cnt = 0;
            while (all_portmap_names.contains(invar)) { //FIXME: should I
also check against messages??
                invar = invar_bak + "_" + cnt;
                ++cnt;
            }
            return invar;
        }*/
    } //end checkvar()

    /**
     * Method to check a type String to perform any needed remappings
     (eg. time->int)
     */
    String checktype(String intype) {

        if ("time".equals(intype))
            return "int";
        else if ("bool".equals(intype))
            return "int";
        else
            return intype;
    } //end checktype

    /**
     * Method finds which subfunctions use recursion... ie. they *MUST*
     be subfunctions
     * in the final implementation can CANNOT be flattened...
     *
     * This fills the wRCX class's internal structure with the String
     names of the subfuncs
     *
     * This ALSO updates the RCX-global count of number of subfunctions
     per task/rcx
     * to add to the UNI-OGEL specification so the last stage can
     optimize knowing this

```



```

    * information (so this list is the list of all known functions)
    *
    * This is a driver function for the other recursive function
    */
void find_recursion(wRCX rcx, wTask task) {

    Iterator subfuncs_itr =
task.subfunc_ref_cnt.entrySet().iterator();

    while (subfuncs_itr.hasNext()) {
        Map.Entry subfuncs = (Map.Entry) subfuncs_itr.next();
        String func_name = (String) ((Map.Entry) subfuncs).getKey();
        wFunction next_func = (wFunction) all_funcs.get(func_name);

        int task_count =
((Integer) ((Map.Entry) subfuncs).getValue()).intValue();
        int rcx_count = 0;

        if (rcx.subfunc_ref_cnt.containsKey(func_name)) {
            rcx_count =
((Integer) rcx.subfunc_ref_cnt.get(func_name)).intValue();
        }

        rcx.subfunc_ref_cnt.put(func_name, new Integer(task_count +
rcx_count));

        //if the trail finds it, add it to the wRCX struct
        if (find_recursion_recur(rcx, func_name, new TreeSet(),
next_func)) {
            rcx.recur_func_names.add(func_name);
        }
    } //end for each func the task calls

    // also check when's functions
    Iterator when_funcs_itr = task.when_funcs.entrySet().iterator();

    while (when_funcs_itr.hasNext()) {
        Map.Entry subfuncs = (Map.Entry) when_funcs_itr.next();
        String func_name = (String) subfuncs.getKey();
        wFunction next_func = (wFunction) subfuncs.getValue();

        int rcx_count = 0;

        if (rcx.subfunc_ref_cnt.containsKey(func_name)) {
            rcx_count =
((Integer) rcx.subfunc_ref_cnt.get(func_name)).intValue();
        }

        rcx.subfunc_ref_cnt.put(func_name, new Integer(rcx_count));

        //if the trail finds it, add it to the wRCX struct
        if (find_recursion_recur(rcx, func_name, new TreeSet(),
next_func)) {
            rcx.recur_func_names.add(func_name);
        }
    }
}

```

```

    } //end find_recursion
    boolean find_recursion_recur(wRCX rcx, String func_name, Set
found_funcs, wFunction f) {

        if (f.subfunc_ref_cnt.containsKey(func_name)) {
            return true;
        }

        //we knows its not here now, so recur unless we've already
tried a path
        Iterator funcs_itr = f.subfunc_ref_cnt.entrySet().iterator();

        while (funcs_itr.hasNext()) {
            Map.Entry funcs = (Map.Entry) funcs_itr.next();
            String next_func_name =
(String) ((Map.Entry) funcs).getKey();
            wFunction next_func =
(wFunction) all_funcs.get(next_func_name);

            int task_count =
((Integer) ((Map.Entry) funcs).getValue()).intValue();
            int rcx_count = 0;

            if (rcx.subfunc_ref_cnt.containsKey(next_func_name)) {
                rcx_count =
((Integer) rcx.subfunc_ref_cnt.get(next_func_name)).intValue();
            }

            rcx.subfunc_ref_cnt.put(next_func_name, new
Integer(task_count + rcx_count));

            if (found_funcs.contains(next_func_name)) {
                return false;
            } else {
                found_funcs.add(next_func_name);
                return find_recursion_recur(rcx, func_name, found_funcs,
next_func);
            }

        } //end for each function this subfunc calls

        return false;
    } //end fine_recursion_recur()

/**
 * This binds a wTask to a given wRCX for later processing...
 * evaluates whens, etc...
 */
void bind_task(wRCX rcx, wTask task) {

    rcx.task_bindings.add(task);

    Map regex_names = new TreeMap();           //key=String regex-for-
port    val=replacement

```

```

//iterate through INs
Iterator ins_itr = task.in_ports.iterator();
while (ins_itr.hasNext()) {
    String fake_name = (String)ins_itr.next();
    String replc_name = (String)rcx.portmaps.get(fake_name);
    regex_names.put("\\s"+fake_name+"\\s", replc_name);
}

//iterate through OUTs
Iterator outs_itr = task.out_ports.iterator();
while (outs_itr.hasNext()) {
    String fake_name = (String)outs_itr.next();
    String replc_name = (String)rcx.portmaps.get(fake_name);
    regex_names.put("\\s"+fake_name+"\\s", replc_name);
}

//Now save the WHEN conditions to the wRCX object, portmapping
everything...
Iterator whens_itr = task.when_pairs.entrySet().iterator();
while (whens_itr.hasNext()) {
    Map.Entry whens = (Map.Entry) whens_itr.next();
    String when_cond = (String)((Map.Entry)whens).getKey();
    String when_func_call = (String)((Map.Entry)whens).getValue();

    Iterator regex_names_itr = regex_names.entrySet().iterator();
    while (regex_names_itr.hasNext()) {
        Map.Entry regex_pair = (Map.Entry) regex_names_itr.next();
        String regex = (String)regex_pair.getKey();
        String replc = (String)regex_pair.getValue();
        when_cond = when_cond.replaceAll(regex, replc);
        when_func_call = when_func_call.replaceAll(regex, replc);
    }
    rcx.when_pairs.put(when_cond, when_func_call);
} //end for every when list

} //end bind_task()

/**
 * this method writes out all task methods for a given RCX...
 * and writes out a UNI-OGEL specification into the given Pipe,
performing
 * appropriate mappings between wRCX and Task
 *
 */
void write_rcx_tasks(String rcx_id, wRCX rcx, Writer pipe) throws
IOException {

    //For Every wTask this wRCX is bound to, do the following...
    Iterator task_itr = rcx.task_bindings.iterator();
    while (task_itr.hasNext()) {
        wTask task = (wTask)task_itr.next();

        //start with wTask decl
        pipe.write(rcx_id + "-task:"+task.name+":    task " + task.name
+ " () { "+ENDLINE);
        pipe.flush();

```

```

//Now write out lines which setup any sensors...
Iterator types_itr = rcx.typemaps.entrySet().iterator();
while (types_itr.hasNext()) {
    Map.Entry types_elm = (Map.Entry)types_itr.next();
    String port_name =
(String)rcx.portmaps.get(types_elm.getKey());
    String sense_name = (String)types_elm.getValue();
    pipe.write(rcx_id+"-task:"+task.name+":
SetSensor("+port_name+", "+sense_name+");");
    pipe.flush();
} //end for each SENSOR_TYPE

//For every task's portmapping, find their real names, build
regex string

port    Map regex_names = new TreeMap();          //key=String regex-for-
        val=replacement

//iterate through INs
Iterator ins_itr = task.in_ports.iterator();
while (ins_itr.hasNext()) {
    String fake_name = (String)ins_itr.next();
    String replc_name = (String)rcx.portmaps.get(fake_name);
    regex_names.put("\\s"+fake_name+"\\s", replc_name);
}

//iterate through OUTs
Iterator outs_itr = task.out_ports.iterator();
while (outs_itr.hasNext()) {
    String fake_name = (String)outs_itr.next();
    String replc_name = (String)rcx.portmaps.get(fake_name);
    regex_names.put("\\s"+fake_name+"\\s", replc_name);
}

//iterate through all the statements in the task
Iterator statements_itr = task.contents_list.iterator();

while (statements_itr.hasNext()) {

    String new_line_str = (String)statements_itr.next();

    Iterator regex_names_itr =
regex_names.entrySet().iterator();
    while (regex_names_itr.hasNext()) {
        Map.Entry regex_pair =
(Map.Entry)regex_names_itr.next();
        String regex = (String)regex_pair.getKey();
        String replc = (String)regex_pair.getValue();
        new_line_str = ((String)new_line_str).replaceAll(regex,
replc);
    }

    pipe.write(rcx_id + "-task:"+task.name+": " +
new_line_str + ENDLINE);
    pipe.flush();
}

```

```

        } //end for every statement in task

        //write final close-brackets
        pipe.write(rcx_id + "-task:" + task.name + ":    } "+ENDLINE);
        pipe.flush();

    } //end for every task

} //end write_rcx_tasks

/**
 * This method writes out all the essential wRCX info such as
subfunc ref counts, etc
 *
 */
void write_rcx_info (String rcx_id, wRCX rcx, Writer pipe) throws
IOException {

    //Print the name of the RCX
    pipe.write("new-rcx:" + rcx_id + ":" + ENDLINE);
    pipe.flush();

    //Print out the total number of tasks this wRCX unit has to
support (excluding whens)
    pipe.write(rcx_id + "-total-tasks:
"+rcx.task_bindings.size() + ENDLINE);
    pipe.flush();

    //Nomatter what RCX, always tell what are our GLOBAL Message
variables...
    Iterator msg_var_names_itr = all_msgs.keySet().iterator();
    while (msg_var_names_itr.hasNext()) {
        String msg_name = (String)msg_var_names_itr.next();
        pipe.write(rcx_id + "-message: " + msg_name + ENDLINE);
        pipe.flush();
    } //end for each message

    //Print out one line which lists all the recursive functions
    pipe.write(rcx_id + "-recur-funcs: ");
    pipe.flush();
    Iterator recur_names_itr = rcx.recur_func_names.iterator();
    while (recur_names_itr.hasNext()) {
        String recur_names = (String)recur_names_itr.next();
        pipe.write((String)recur_names + " ");
        pipe.flush();
    } //end for each recursive name
    pipe.write(ENDLINE);
    pipe.flush();

    //Print every functions reference count
    Iterator ref_cnt_itr = rcx.subfunc_ref_cnt.entrySet().iterator();
    while (ref_cnt_itr.hasNext()) {
        Map.Entry ref_cnt = (Map.Entry) ref_cnt_itr.next();
        String subfunc_name = (String)((Map.Entry)ref_cnt).getKey();
        int count =
((Integer)((Map.Entry)ref_cnt).getValue()).intValue();

```

```

        pipe.write(rcx_id+"-subfunc-ref:"+subfunc_name+":
"+count+ENDLINE);
        pipe.flush();
    } //end for each subfunc reference

    //Print out WHEN conditions for this RCX
    Iterator whens_itr = rcx.when_pairs.entrySet().iterator();
    while (whens_itr.hasNext()) {
        Map.Entry whens = (Map.Entry)whens_itr.next();
        String when_cond = (String)((Map.Entry)whens).getKey();
        String when_func_call = (String)((Map.Entry)whens).getValue();
        pipe.write(rcx_id+"-
when:"+WHEN_TOK_SEPARATOR+when_cond+WHEN_TOK_SEPARATOR+when_func_call+E
NDLINE);
        pipe.flush();
    }

    } //end write_rcx_info()

/**
 * This method writes out all the functions wich 'this' wRCX
requires... WITH proper Portmappings
 */
void write_rcx_subfuncs (String rcx_id, wRCX rcx, Writer pipe)
throws IOException {

    //iterate through every references subfunction
    Iterator subfunc_refs_itr =
rcx.subfunc_ref_cnt.entrySet().iterator();
    while (subfunc_refs_itr.hasNext()) {
        Map.Entry subfunc_refs = (Map.Entry) subfunc_refs_itr.next();
        String func_name = (String)((Map.Entry)subfunc_refs).getKey();
        wFunction subfunc = (wFunction)all_funcs.get(func_name);

        Map regex_names = new TreeMap();          //key=String regex-for-
port    val=replacement

        //iterate through INs
        Iterator ins_itr = subfunc.in_ports.iterator();
        while (ins_itr.hasNext()) {
            String fake_name = (String)ins_itr.next();
            String replc_name = (String)rcx.portmaps.get(fake_name);
            regex_names.put("\\s"+fake_name+"\\s", replc_name);
        }

        //iterate through OUTs
        Iterator outs_itr = subfunc.out_ports.iterator();
        while (outs_itr.hasNext()) {
            String fake_name = (String)outs_itr.next();
            String replc_name = (String)rcx.portmaps.get(fake_name);
            regex_names.put("\\s"+fake_name+"\\s", replc_name);
        }

        //first print out the function header

```

```

        pipe.write(rcx_id+"-subfunc:"+func_name+":    void
"+func_name+" ( ");
        pipe.flush();
        Iterator param_itr = subfunc.paramnames.entrySet().iterator();
        while (param_itr.hasNext()) {
            Map.Entry param = (Map.Entry) param_itr.next();
            String type = (String)((Map.Entry)param).getValue();
            String var = (String)((Map.Entry)param).getKey();
            if (param_itr.hasNext()) {
                pipe.write(type+" "+var+", ");
            } else {
                pipe.write(type+" "+var+" ");
            }
            pipe.flush();
        }
        pipe.write(" ) { "+ENDLINE);
        pipe.flush();

        //iterate through all the statements in the task
        Iterator statements_itr = subfunc.contents_list.iterator();
        while (statements_itr.hasNext()) {

            String new_line_str = (String)statements_itr.next();

            Iterator regex_names_itr =
regex_names.entrySet().iterator();
            while (regex_names_itr.hasNext()) {
                Map.Entry regex_names_lp =
(Map.Entry)regex_names_itr.next();
                Map.Entry regex_pair = (Map.Entry)regex_names_lp;
                String regex = (String)regex_pair.getKey();
                String replc = (String)regex_pair.getValue();
                new_line_str = ((String)new_line_str).replaceAll(regex,
replc);
            }

            pipe.write(rcx_id+"-subfunc:"+func_name+":
"+new_line_str+ENDLINE);
            pipe.flush();

        } //end for every statement in task

        //write final close-brackets
        pipe.write(rcx_id + "-subfunc:"+func_name+":    } "+ENDLINE);
        pipe.flush();

    } //end for every subfunction

} //end write_rcx_subfuncs()

/**
 * So I could reuse the conditional Code, I had to move this out of
the root-structure
 * because antlr puked on "#(a:(foo|bar)" ??
 */
String parseCondition(AST a) {

```

```

String expr_buf = "( "; //LPAREN
int numConditions = a.getNumberOfChildren();

AST condition = a.getFirstChild();

for(int i = 0; i < numConditions; ++i){

    //multiple conditions are delineated by having children or
not
    if((condition.getNumberOfChildren()) != 0) {

        expr_buf += "( ";
        expr_buf +=
checkvar(condition.getFirstChild().getText()); //lvalue
        expr_buf += " " + condition.getText() + " ";
//condition
        expr_buf +=
checkvar(condition.getFirstChild().getNextSibling().getText());
//rvalue
        expr_buf += " )";

        } else { //is a condition binder
            expr_buf += condition.getText();
        }

        condition = condition.getNextSibling();
    } //end for each condition

    expr_buf += " )"; //RPAREN

    return expr_buf;
} //end parseCondition()

} //end prefecing java source

//*****
// Here we start off parsing our tree
//
// We go through, add all wRCX and messages to arrays
// Then add all tasks to arrays, filling up their string buffers with
the actual CODE (statements)
// which they contain, same with functions
//
// One we reach main, we'll see that a task is bound to an RCX, write
out all information
// to the UNI-OGEL file... continue for each task and each RCX
//
//*****
file
: #(FILE readRCXs readMessages readFunctions readTasks readMain
{
    //POST Processing
}

```



```

    )
    ;

//*****
// Here handle building up the wRCX struct predefines with all
necessary info
// (not rcx instances, but defs)
//*****
readRCXs
    : #(RCXDECLS
        (#("rcx"
            {
                String className;
                wRCX next_rcx = new wRCX();
            }
            className=getClass
            getPorts[next_rcx]

            //when traversed one RCX, just create a new wRCX object with
the
            //required port mapping and names and insert it into the
global
            //RCXs vector which will hold all wRCX objects
            {
                next_rcx.update_portmap();
                all_rcxs.put(className, next_rcx);
            }
        )*)
    );

//returns the class name of the rcx
getClass returns [String name]
    {name = null;}
    :IDENTIFIER {name = #IDENTIFIER.getText();}
    ;

//grab portnames... dont worry about correctness
getPorts [wRCX next_rcx]
    :(#(EQUALS a:IDENTIFIER
        {
            String type = "";
            String name = "";
            String port = "";
            AST b=a.getNextSibling();
            AST c=b.getNextSibling();
            if (c==null) { //this is an OUTPUT port
                name=a.getText();
                port=b.getText();
            } else {
                type = a.getText();
                name = b.getText();
                port = c.getText();
            }
            if(port.equals("$1")){
                next_rcx.one_port = name;
            }
        }
    )

```

```

        next_rcx.type_one_port = type;
    } else if(port.equals("$2")){
        next_rcx.two_port = name;
        next_rcx.type_two_port = type;
    } else if(port.equals("$3")){
        next_rcx.three_port = name;
        next_rcx.type_three_port = type;
    } else if(port.equals("$A")){
        next_rcx.a_port = name;
    } else if(port.equals("$B")){
        next_rcx.b_port = name;
    } else if(port.equals("$C")){
        next_rcx.c_port = name;
    }

    //System.out.println("FOUND PORT "+name+ " "+port);
    //System.out.println("B: "+next_rcx.b_port);
    all_portmap_names.add(name); //safe since static-semantic says
this'll always give 'something'
    }
    ))*
    ;

//*****
// Here handle adding messages to the universal context
//*****
readMessages
    : #(MESSAGEDECLS
        (#("message" a:IDENTIFIER
            {
                all_msgs.put(a.getText(), new Integer(++global_msg_cnt));
                //System.out.println("OGELWriter: Parsed Message
Declaration " + a.getText() +
                // " number=" + (global_msg_cnt-1));
            }
        ))*
    );

//*****
// Here begin translations of functions abstractly into the structs
//*****
readFunctions
    : #(FUNCTIONLIST
        (#(FUNCTION
            {
                wFunction f = new wFunction();
            }
            parseFunctionDeclaration[f]
            parseStatementList[f, INDENT_INC]
        ))*
    );

parseFunctionDeclaration [wFunction f]

```

```

: #(FUNCTIONDECLARATION
  a:IDENTIFIER
  {
    f.name = checkvar(a.getText());
    all_funcs.put(f.name, f);
  }
  readParameterList[f]
  readInList[f]
  readOutList[f]
  {
    //System.out.print("OGELwriter: Parsed wFunction "+f.name+"
parameters: ");
    Iterator param_itr = f.paramnames.entrySet().iterator();
    while (param_itr.hasNext()) {
      Map.Entry param = (Map.Entry)param_itr.next();
      //System.out.print(param.getValue() + " " + param.getKey() + "
");
    }
    //System.out.println("");
  }
);

readParameterList [wFunction f]
: #(PARAMETERLIST
  ("int"
  b:IDENTIFIER
  {
    f.paramnames.put(b.getText(), "int");
  }
  ) | ("const"
  c:IDENTIFIER
  {
    f.paramnames.put(c.getText(), "const");
  }
  ))*
);

readInList [SourceHolder s]
: #(a:INLIST
  {
    AST b = a.getFirstChild();
    for (int i = 0; i < a.getNumberOfChildren(); ++i) {
      s.in_ports.add(b.getText());
      b = b.getNextSibling();
    }
  }
);

readOutList [SourceHolder s]
: #(a:OUTLIST
  {
    AST b = a.getFirstChild();
    for (int i=0; i < a.getNumberOfChildren(); ++i) {
      s.out_ports.add(b.getText());
      b = b.getNextSibling();
    }
  }
);

```

```

);

//*****
// This is where I parse out real statements for both task and
// functions
// Items are simply parsed into verbatim final code (except for fake
// portmappings)
// which is saved in the sourceholder buffer

parseStatementList[SourceHolder s, int indent]
: #(a:STATEMENTLIST
  {
    int children = a.getNumberOfChildren();
    AST b = a.getFirstChild();
    for (int i=0; i < children; ++i) {

      String temp = b.getText();
      if(temp.equals("while"))
        parseWhileBlock(b, s, indent);
      else if(temp.equals("do"))
        parseDo(b, s, indent);
      else if(temp.equals("if"))
        parseIf(b, s, indent);
      else if(temp.equals("variableAssignment"))
        parseVariableAssignment(b, s, indent);
      else if(temp.equals("variableDeclaredAssignment"))
        parseVariableDeclaredAssignment(b, s, indent);
      else if(temp.equals("variableDeclaration"))
        parseVariableDeclaration(b, s, indent);
      else if(temp.equals("functionCall"))
        parseFunctionCall(b, s, indent);
      else if(temp.equals("passMessage"))
        parsePassMessage(b, s, indent);
      else if(temp.equals("receiveMessage"))
        parseReceiveMessage(b, s, indent);
      else if (temp.equals("nqcBlockStatement"))
        parseNQCBlockStatement(b, s, indent);
      else if (temp.equals("nqcFunctionCall"))
        parseNQCFunctionCall(b, s, indent);
      else if (temp.equals("catchMessage"))
        parseCatchMessage(b, s, indent);
      else if (temp.equals("throwMessage"))
        parseThrowMessage(b, s, indent);

      b = b.getNextSibling();
    } //end for each child
  }
);

//Statement While block
parseWhileBlock [SourceHolder s, int indent]
: #(a:"while"
  {
    AST whileCondition = a.getFirstChild();
    AST whileStatements = whileCondition.getNextSibling();

```

```

int numConditions = whileCondition.getNumberOfChildren();

AST condition = whileCondition.getFirstChild();

String while_cond = parseCondition(whileCondition);

String contents_str = "";
for (int i=0; i<indent; ++i) { contents_str += " "; };
contents_str += "while "+while_cond+" { ";
s.contents_list.add(contents_str);
indent += INDENT_INC;

//whileStatement Checking
int children = whileStatements.getNumberOfChildren();
AST b = whileStatements.getFirstChild();
for (int j = 0; j < children; ++j){
    String temp = b.getText();
    if(temp.equals("while"))
        parseWhileBlock(b, s, indent);
    else if(temp.equals("do"))
        parseDo(b, s, indent );
    else if(temp.equals("if"))
        parseIf(b, s, indent);
    else if(temp.equals("variableAssignment"))
        parseVariableAssignment(b, s, indent);
    else if(temp.equals("variableDeclaredAssignment"))
        parseVariableDeclaredAssignment(b, s, indent );
    else if(temp.equals("variableDeclaration"))
        parseVariableDeclaration(b, s, indent);
    else if(temp.equals("functionCall"))
        parseFunctionCall(b, s, indent );
    else if(temp.equals("passMessage"))
        parsePassMessage(b, s, indent);
    else if(temp.equals("receiveMessage"))
        parseReceiveMessage(b, s, indent);
    else if (temp.equals("nqcBlockStatement"))
        parseNQCBlockStatement(b, s, indent);
    else if (temp.equals("nqcFunctionCall"))
        parseNQCFunctionCall(b, s, indent);
    else if (temp.equals("catchMessage"))
        parseCatchMessage(b, s, indent);
    else if (temp.equals("throwMessage"))
        parseThrowMessage(b, s, indent);

    b = b.getNextSibling();
} //end for each child

String end_contents_str = "";
for (int i=0; i<(indent-INDENT_INC); ++i) { end_contents_str
+= " "; };
end_contents_str += "}";
s.contents_list.add(end_contents_str);
}
);

//Statement Do block
parseDo [SourceHolder s, int indent]

```

```

: # (a: "do"
{

String contents_str = "";
for (int i=0; i<indent; ++i) { contents_str += " "; };
contents_str += "do { ";
s.contents_list.add(contents_str);
indent += INDENT_INC;

int children = a.getNumberOfChildren();
AST b = a.getFirstChild();
for(int i =0; i < children; ++i){
    String temp = b.getText();
    if(temp.equals("while"))
        parseWhileBlock(b, s, indent);
    else if(temp.equals("do"))
        parseDo(b, s, indent);
    else if(temp.equals("if"))
        parseIf(b, s, indent);
    else if(temp.equals("variableAssignment"))
        parseVariableAssignment(b, s, indent);
    else if(temp.equals("variableDeclaredAssignment"))
        parseVariableDeclaredAssignment(b, s, indent);
    else if(temp.equals("variableDeclaration"))
        parseVariableDeclaration(b, s, indent);
    else if(temp.equals("functionCall"))
        parseFunctionCall(b, s, indent);
    else if(temp.equals("whileCondition")) {
        String while_cond = parseCondition(b);
        String end_contents_str = "";
        for (int j=0; j<indent; ++j) { end_contents_str += " ";

        end_contents_str += "} while "+while_cond+"";
        s.contents_list.add(end_contents_str);
    }
    else if(temp.equals("passMessage"))
        parsePassMessage(b, s, indent);
    else if(temp.equals("receiveMessage"))
        parseReceiveMessage(b, s, indent);
    else if (temp.equals("nqcBlockStatement"))
        parseNQCBlockStatement(b, s, indent);
    else if (temp.equals("nqcFunctionCall"))
        parseNQCFunctionCall(b, s, indent);
    else if (temp.equals("catchMessage"))
        parseCatchMessage(b, s, indent);
    else if (temp.equals("throwMessage"))
        parseThrowMessage(b, s, indent);

    b = b.getNextSibling();
}
}
);

//Statement IF block
parseIf [SourceHolder s, int indent]
: # (a: "if"

```

```

b:IFCONDITION
{
String if_cond = parseCondition(b);
String contents_str = "";
for (int i=0; i<indent; ++i) { contents_str += " "; };
contents_str += "if "+if_cond+" {";
s.contents_list.add(contents_str);
indent += INDENT_INC;

int children = a.getNumberOfChildren();

//if there is actually something inside the if like standard
//statements then we traverse them. else do nothing
if(children > 1){
    b = b.getNextSibling();
    for(int i =0; i < children-1; ++i){
        String temp = b.getText();
        if(temp.equals("while"))
            parseWhileBlock(b, s, indent);
        else if(temp.equals("do"))
            parseDo(b, s, indent);
        else if(temp.equals("if"))
            parseIf(b, s, indent);
        else if(temp.equals("variableAssignment"))
            parseVariableAssignment(b, s, indent);
        else if(temp.equals("variableDeclaredAssignment"))
            parseVariableDeclaredAssignment(b, s, indent);
        else if(temp.equals("variableDeclaration"))
            parseVariableDeclaration(b, s, indent);
        else if(temp.equals("functionCall"))
            parseFunctionCall(b, s, indent);
        else if(temp.equals("else"))
            parseElse(b, s, indent);
        else if(temp.equals("passMessage"))
            parsePassMessage(b, s, indent);
        else if(temp.equals("receiveMessage"))
            parseReceiveMessage(b, s, indent);
        else if (temp.equals("nqcBlockStatement"))
            parseNQCBlockStatement(b, s, indent);
        else if (temp.equals("nqcFunctionCall"))
            parseNQCFunctionCall(b, s, indent);
        else if (temp.equals("catchMessage"))
            parseCatchMessage(b, s, indent);
        else if (temp.equals("throwMessage"))
            parseThrowMessage(b, s, indent);

        b = b.getNextSibling();
    }
}

String end_contents_str = "";
for (int i=0; i<(indent-INDENT_INC); ++i) { end_contents_str
+= " "; };
end_contents_str += "}";
s.contents_list.add(end_contents_str);
}

```

```

);

parseElse [SourceHolder s, int indent]
: #(a:"else"
b:STATEMENTLIST
{
String contents_str = "";
for (int i=0; i<(indent-INDENT_INC); ++i) { contents_str += "
"; }; //else indenting is diff
contents_str += "} else {";
s.contents_list.add(contents_str);

parseStatementList(b, s, indent);
}
);

parseFunctionCall [SourceHolder s, int indent]
: #(FUNCTIONCALL
a:IDENTIFIER
b:FUNCTIONPARAMSPASSING
{

String func_name = checkvar(a.getText());

String call_str= "";
for (int i=0; i<indent; ++i) { call_str += " "; };
call_str += SUBFUNC_CALL+" "+func_name+" (";
indent += INDENT_INC;

//parameters that are being passed
int children = b.getNumberOfChildren();
if(children != 0) {
AST c = b.getFirstChild();

//iterate through each child
for(int i =0; i < children; ++i){

//for last guy, don't add the comma
if (i == (children-1))
call_str += c.getText() + " ";
else
call_str += c.getText() + " , ";

c = c.getNextSibling();
}

}

s.contents_list.add(call_str + " );");

//also keep track of reference counting
if (s.subfunc_ref_cnt.containsKey(func_name)) {
int count =
((Integer)s.subfunc_ref_cnt.get(func_name)).intValue();
s.subfunc_ref_cnt.put(func_name, new Integer(++count));
} else {
s.subfunc_ref_cnt.put(func_name, new Integer(1));
}
}
);

```



```

    }
    }
);

parseVariableDeclaration [SourceHolder s, int indent]
: #(a:VARIABLEDECLARATION
  {
    //first get type and name
    AST b = a.getFirstChild();
    String type = checktype(b.getText());
    String variable = b.getNextSibling().getText();

    String contents_str = "";
    for (int i=0; i<indent; ++i) { contents_str += " "; };
    contents_str += type+" "+variable+";";
    s.contents_list.add(contents_str);

  }
);

parseVariableAssignment [SourceHolder s, int indent]
: #(VARIABLEASSIGNMENT
a:EQUALS
  {
    //get the three children which are type, variable name, and
value
    AST b = a.getFirstChild();
    String type = checktype(b.getText());
    b = b.getNextSibling();
    String name = b.getText();
    String assignment = a.getText();
    b = b.getNextSibling();
    String value = b.getText();

    String contents_str = "";
    for (int i=0; i<indent; ++i) { contents_str += " "; };
    contents_str += type+" "+name+" "+assignment+" "+value+";";
//unknown var prior can 'only' use "="
    s.contents_list.add(contents_str);
  }
);

parseVariableDeclaredAssignment [SourceHolder s, int indent]
: #(a:VARIABLEDECLAREDASSIGNMENT
  {
    AST b = a.getFirstChild();
    String variable = b.getFirstChild().getText();
    String assignment = b.getText();
    AST c = b.getFirstChild().getNextSibling();
    String value = c.getText();

    String contents_str = "";
    for (int i=0; i<indent; ++i) { contents_str += " "; };
    contents_str += variable+" "+assignment+" "+value+" ";
    s.contents_list.add(contents_str);
  }
);

```

```

parseNQCBlockStatement [SourceHolder s, int indent]
  :# (a:NQCBLOCKSTATEMENT
    {
      AST block = a.getFirstChild();
      String nqc_str = block.getText();

      nqc_str = nqc_str.replaceAll("<%", "{");
      nqc_str = nqc_str.replaceAll("%>", "}");
      nqc_str = nqc_str.replaceAll("\\n", "");

      //replace control characters so our regex works later on...
      //ANTLR sucks ass and barfs on a \ ... so must be hokey here

      //NO COMMENTS ALLOWED WITHIN NQC CODE!!

      nqc_str = nqc_str.replaceAll("\\{", " { ");
      nqc_str = nqc_str.replaceAll("\\}", " } ");
      nqc_str = nqc_str.replaceAll("\\"+"(", " ( ");
      nqc_str = nqc_str.replaceAll("\\"+")", " ) ");
      nqc_str = nqc_str.replaceAll("\\"+"+"+"\\"+"+", " ++ ");
      // -(?!=)
      nqc_str = nqc_str.replaceAll("-(?!=)", " - ");
      nqc_str = nqc_str.replaceAll("~", " ~ ");
      // !(?!=)
      nqc_str = nqc_str.replaceAll("(?!=)", " ! ");
      // *(?!=)
      nqc_str = nqc_str.replaceAll("\\"+"*(?!=)", " * ");
      // /(?!/)+(?!=)
      nqc_str = nqc_str.replaceAll("/((?!/)+(?!=))", " / ");
      nqc_str = nqc_str.replaceAll("%", " % ");
      // (?<!+)((?!+)+(?!=)
      nqc_str =
nqc_str.replaceAll("(?!\\"+"+"+"\\"+"+"+(?!"+"+"\\"+"+"+(?!=))", " + ");
      nqc_str = nqc_str.replaceAll("<<", " << ");
      nqc_str = nqc_str.replaceAll(">>", " >> ");
      // (?<!<)<((?!<)+(?!=)
      nqc_str = nqc_str.replaceAll("(?!<)<((?!<)+(?!=))", " < ");
      // (?<!>)>((?!>)+(?!=)
      nqc_str = nqc_str.replaceAll("(?!>)>((?!>)+(?!=))", " > ");
      nqc_str = nqc_str.replaceAll(">=", " >=");
      nqc_str = nqc_str.replaceAll("<=", " <=");
      nqc_str = nqc_str.replaceAll("==", " == ");
      nqc_str = nqc_str.replaceAll("!=", " != ");
      // (?<!&)&(?<!&
      nqc_str = nqc_str.replaceAll("(?!&)&(?<!&)", " & ");
      nqc_str = nqc_str.replaceAll("\\"+"^", " ^ ");
      // (?<!|)|(?<!|)
      nqc_str = nqc_str.replaceAll("(?!\\"+"+"+"|)"+"\\"+"+"+"|"(?!\\"+"+"+"|)",
" | ");
      nqc_str = nqc_str.replaceAll("&&", " && ");
      nqc_str = nqc_str.replaceAll("\\"+"+"+"|"+"\\"+"+"+"|", " || ");
      nqc_str = nqc_str.replaceAll("\\"+"+"+"?", " ? ");
      nqc_str = nqc_str.replaceAll(":", " : ");
      nqc_str = nqc_str.replaceAll(";", " ; ");
      nqc_str = nqc_str.replaceAll(",", " , ");
      nqc_str = nqc_str.replaceAll("\\"+"+"+"=", " += ");

```

```

        nqc_str = nqc_str.replaceAll("-", " -");
        nqc_str = nqc_str.replaceAll("\\\\"+"*=", " *= ");
        nqc_str = nqc_str.replaceAll("/=", " /=");

        // ((?!<)+(?!=)+(?>)+(?!!)+(?!+)+(?!-
)+(?!*)+(?!/)+(?!=))=?!=)
        nqc_str = nqc_str.replaceAll(
            "((?!<)+(?!=)+(?>)+(?!!\\\\"+"!")+(?!\\\\"+"")+)+(?!-
)+(?!\\\\"+"*")+(?!\\\\"+""/)+(?!=))=?!=)", " = ");

        s.contents_list.add(nqc_str);
    }
);

parseReceiveMessage [SourceHolder s, int indent]
: #(a:RECEIVEMESSAGE
{
    String messageName =
a.getFirstChild().getNextSibling().getText();

    String contents_str1 = "";
    String contents_str2 = "";
    for (int i=0; i<indent; ++i) { contents_str1 += " ";
contents_str2 += " "; }
    contents_str1 += "ClearMessage()";
    contents_str2 += "until (Message() ==
"+(((Integer)all_msgs.get(messageName)).intValue()+"));";
    s.contents_list.add(contents_str1);
    s.contents_list.add(contents_str2);
}
);

parsePassMessage [SourceHolder s, int indent]
: #(a:PASSMESSAGE
{
    String messageName =
a.getFirstChild().getNextSibling().getText();

    String contents_str = "";
    for (int i=0; i<indent; ++i) { contents_str += " "; };
    contents_str +=
"SendMessage"+(((Integer)all_msgs.get(messageName)).intValue()+"));";
    s.contents_list.add(contents_str);
}
);

parseThrowMessage [SourceHolder s, int indent]
: #(a:THROWMESSAGE
{
    String messageName =
a.getFirstChild().getNextSibling().getText();

    String contents_str = "";
    for (int i=0; i<indent; ++i) { contents_str += " "; };
    contents_str += messageName+"=0";
    s.contents_list.add(contents_str);
}
);

```

```

    }
    );

parseCatchMessage [SourceHolder s, int indent]
: #(a:CATCHMESSAGE
  {
    String messageName =
a.getFirstChild().getNextSibling().getText();

    String contents_str = "";
    for (int i=0; i<indent; ++i) { contents_str += " "; };
    contents_str += messageName+"++; while( "+messageName+" > 0)
{}";
    s.contents_list.add(contents_str);
  }
);

parseNQCFUNCTIONCALL [SourceHolder s, int indent]
: #(a:NQCFUNCTIONCALL
  {
    AST b = a.getFirstChild();
    AST params = b.getNextSibling();

    String call_str = "";
    for (int i=0; i<indent; ++i) { call_str += " "; }
    call_str += b.getText()+" ( ";

    int children = params.getNumberOfChildren();

    AST c = params.getFirstChild();

    // Our nodes all have the comma/pluses in them...

    for (int i=0; i < children; ++i) {
      call_str += " "+c.getText()+" ";
      c = c.getNextSibling();
    }

    call_str += " ) ; ";
    s.contents_list.add(call_str);
  }
);

//*****
// Here begin translations of tasks abstractly into the structs
//*****
readTasks
: #(TASKDECLARATIONLIST
  (#(TASK
    {
      wTask t = new wTask();
    }
    parseTaskDeclaration[t]
    parseTaskBody[t]

```

```

    ))*
);

parseTaskDeclaration [wTask t]
: #(TASKDECLARATION
  a:IDENTIFIER
  {
    t.name = a.getText();
    all_tasks.put(t.name, t);
    //System.out.println("OGELWriter: Found Task named
"+t.name);
  }
  readInList[t]
  readOutList[t]
);

parseTaskBody [wTask t]
: #(a:TASKBODY
  {
    //System.out.println("OGELWriter: Found Task Body for
"+t.name);
    int children = a.getNumberOfChildren();
    if(children > 0) {
      AST b = a.getFirstChild();
      for (int i=0; i < children; ++i){
        String temp = b.getText();
        if(temp.equals("when"))
          parseWhenCondition(b, t);
        else if(temp.equals("while"))
          parseWhileBlock(b, t, INDENT_INC);
        else if(temp.equals("do"))
          parseDo(b, t, INDENT_INC);
        else if(temp.equals("if"))
          parseIf(b, t, INDENT_INC);
        else if(temp.equals("functionCall"))
          parseFunctionCall(b, t, INDENT_INC);
        else if(temp.equals("variableDeclaredAssignment"))
          parseVariableDeclaredAssignment(b, t, INDENT_INC);
        else if(temp.equals("variableAssignment"))
          parseVariableAssignment(b, t, INDENT_INC);
        else if(temp.equals("variableDeclaration"))
          parseVariableDeclaration(b, t, INDENT_INC);
        else if(temp.equals("passMessage"))
          parsePassMessage(b, t, INDENT_INC);
        else if(temp.equals("receiveMessage"))
          parseReceiveMessage(b, t, INDENT_INC);
        else if(temp.equals("nqcBlockStatement"))
          parseNQCBlockStatement(b, t, INDENT_INC);
        else if (temp.equals("nqcFunctionCall"))
          parseNQCFunctionCall(b, t, INDENT_INC);
        //else if (temp.equals("catchMessage"))
        //  parseCatchMessage(b, t, INDENT_INC);
        //else if (temp.equals("throwMessage"))
        //  parseThrowMessage(b, t, INDENT_INC);

        b = b.getNextSibling();
      } //end for each child
    }
  }
);

```

```

        } //end if children exist
    }
);

parseWhenCondition [wTask t]
: #(a:"when"
  b:WHENCONDITION
  {
      AST condition = b.getFirstChild();
      AST whenStatementDo = b.getNextSibling();
      AST function = whenStatementDo.getFirstChild();
      AST funcParamsPass =
function.getFirstChild().getNextSibling();

      String func_name =
checkvar(function.getFirstChild().getText());
      String when_cond = parseCondition(b);

      wFunction when_func = (wFunction)all_funcs.get(func_name);
      t.when_funcs.put(func_name, when_func);

      String call_str = func_name + "( ";
      //parameters that are being passed
      int children = funcParamsPass.getNumberOfChildren();
      if(children != 0) {
          AST c = funcParamsPass.getFirstChild();

          //iterate through each child
          for(int i =0; i < children; ++i){

              //for last guy, don't add the comma
              if (i == (children-1))
                  call_str += c.getText() + " ";
              else
                  call_str += c.getText() + " , ";

              c = c.getNextSibling();
          }
      } //endif children exist

      call_str += " );";

      //now add to map for whens
      t.when_pairs.put(when_cond, call_str);
  }
);

//*****
//*****
// The Main wTask struct will have certain elements which are
// handled in this section
//*****
//*****

readMain
: #(a:"main"
  parseRCXInstantiations

```

```

        parseRCXAddTaskList
    {
        //POST-processing... here we'll write out info about wRCX
instances as a whole

        FileWriter out_pipe = null;
        try {
            out_pipe = new FileWriter(output_file_name, false); //do
NOT append
        } catch (IOException ioe) { /* do nothing - caught later */ }

        if (out_pipe == null) {
            System.err.println("Failed To create output file:
"+output_file_name);
            System.exit(1);
        }

        //For every wRCX instance we've found... write everythign
Iterator rcx_itr = inst_rcxs.entrySet().iterator();
while (rcx_itr.hasNext()) {
    Map.Entry rcx_elm = (Map.Entry)rcx_itr.next();
    String rcx_id = (String)((Map.Entry)rcx_elm).getKey();
    wRCX rcx_obj = (wRCX)((Map.Entry)rcx_elm).getValue();

    try {
        write_rcx_info(rcx_id, rcx_obj, out_pipe);
        write_rcx_subfuncs(rcx_id, rcx_obj, out_pipe);
        write_rcx_tasks(rcx_id, rcx_obj, out_pipe);
    } catch (IOException ioe) {
        System.err.println("Caught IOException on final write");
        ioe.printStackTrace();
    }

} //end for each rcx instance

}
);

```

```

parseRCXInstantiations
: #(a:RCXINSTANTIATIONS
{
    int children = a.getNumberOfChildren();

    String variable = "";
    String className = "";
    AST inst = a.getFirstChild();

    for (int j=0; j < children; ++j) {
        className = inst.getFirstChild().getText();
        variable = inst.getFirstChild().getNextSibling().getText();

        wRCX decl_rcx = (wRCX) all_rcxs.get(className);
        wRCX new_inst_rcx = new wRCX();
        new_inst_rcx.a_port = decl_rcx.a_port;
        new_inst_rcx.b_port = decl_rcx.b_port;
        new_inst_rcx.c_port = decl_rcx.c_port;
    }
}
);

```

```

        new_inst_rcx.one_port = decl_rcx.one_port;
        new_inst_rcx.two_port = decl_rcx.two_port;
        new_inst_rcx.three_port = decl_rcx.three_port;

        new_inst_rcx.update_portmap();
        inst_rcxs.put(variable, new_inst_rcx);

        inst = inst.getNextSibling();
    }
}
);

parseRCXAddTaskList
: # (a:RCXADDTASKLIST
  {
    int children = a.getNumberOfChildren();
    AST b = a.getFirstChild();

    for (int i=0; i < children; ++i) {
        String rcxVar = b.getFirstChild().getText();
        String addedwTask = null;

        AST task_id_list = b.getFirstChild().getNextSibling();
        //System.out.println("OGELwriter: Binding task
"+task_id_list.getFirstChild().getText()+" to RCX "+rcxVar);
        parseTaskIdentifierList(task_id_list, rcxVar,
(wRCX)inst_rcxs.get(rcxVar));

        b = b.getNextSibling();
    } //end for each rcx-add-task-call

  }
);

parseTaskIdentifierList [String rcxVar, wRCX this_rcx]
: # (a:TASKIDENTIFIERLIST
  {
    //get the number of tasks that are trying to be added
    int children = a.getNumberOfChildren();

    //get the first child to start traversing
    AST child = a.getFirstChild();

    for (int i=0; i < children; ++i) {

        String task_name = child.getText();
        wTask this_task = (wTask) all_tasks.get(task_name);

        bind_task(this_rcx, this_task);           //apply appropriate
bindings for whens, task-writing

        find_recursion(this_rcx, this_task);     //add which subfuncs
'are' recursive                               //also adds subfunc
ref counts from 'this' task

```



```

        child = child.getNextSibling();
    } //end for each task being added to an RCX
}

);

```

---

```

#!/usr/bin/perl

# This script parses through a main OGEL file and all its constituent
# library files, and creates one 'master' file which has all the
# necessary
# source code in the correct order
#
# Author: Charles O'Donnell <cwo4@columbia.edu>
#
# This also checks to see if a task/function/rcx/message is being used
# twice
# and throws out an Error (this is bad ;)
#
# usage: "libpacker.pl <mainfile>
#
#####
# Constants

$file_stem = @ARGV[0];

if ($file_stem =~ /(.*?)\.ogel/) { $file_stem = $1; }

$master_file = ">" . $file_stem . ".master";

$rcxid = "rcx";
$messageid = "message";
$taskid = "task";
$funcid = "void";
$mainid = "task main";
#####

@lib_files; # array with all libraries which will be parsed
# run through, trying to detect #includes

open (INFILE, @ARGV[0]);

while (<INFILE>) {

    if ($_ !~ / /) { #ignore lines which have no useful data (ie. check
for spaces)
        next;
    }
    if (/^#include\s+"(.*?)"/) {
        push (@lib_files, $1);
    } else {
        last;
    }
}
}

```

```

push(@lib_files, @ARGV[0]); #push main file last so its eval'd last

#####

%define_sets;

#####

#lists which hold found names to check for duplicates
@rcx_names;
@message_names; #special... if find duplicate just dont print second
version
@task_names;
@func_names;

open (MFILE, $master_file);

# parse out RCXs
foreach $filename (@lib_files) {
    $write_bool = 0;
    $linenum = 0;

    # For Each file, parse defines first...
    open (DEFFILE, $filename);
    while (<DEFFILE>) {

        if ($_ !~ / /) { #ignore lines which have no useful data (ie.
check for spaces)
            next;
        }
        if (/^#define\s+(.+?)\s+(.*)/) {
            $fake = $1;
            $rep = $2;
            chomp($fake);
            chomp($rep);
            $define_sets{$fake} = $rep;
        } else {
            last;
        }
    }
    close DEFFILE;

    open (INFILE, $filename);
    while (<INFILE>) {
        $linenum++;
        chomp;
        if (/^$rcxid\s+(.+?)\s+(\s|\{|\})/) {
            $write_bool = 1;

            foreach $name (@rcx_names) {
                if ($1 eq $name) {
                    print "Error: Duplicate definition of rcx $name in file
$filename \ (line $linenum)\n";
                    exit;
                }
            }
        }
    }
}

```

```

        }
    }
    push @rcx_names, $1;
} elsif (/^(($messageid|$taskid|$funcid|$mainid)/) {
    $write_bool = 0;
}
if ($write_bool) {
    while (my($def_fake, $def_rep) = each(%define_sets) ) {
        $_ =~ s/$def_fake/$def_rep/;
    }

    print MFILE $_ . "\n";
}
} #end check each line of file
close INFILE;
} # end foreach ARGV

# parse out Messages
foreach $filename (@lib_files) {
    $write_bool = 0;
    $linenum = 0;
    open (INFILE, $filename);
    while (<INFILE>) {
        $linenum++;
        chomp;
        if (/^$messageid\s+(.+?) (\s|\;)/) {
            $write_bool = 1;
            foreach $name (@message_names) {
                if ($1 eq $name) {
                    $write_bool = 0; #just dont write second copy
                }
            }
            push @message_names, $1;
        } elsif (/^($rcxid|$taskid|$funcid|$mainid)/) {
            $write_bool = 0;
        }
        if ($write_bool) {
            while (my($def_fake, $def_rep) = each(%define_sets) ) {
                $_ =~ s/$def_fake/$def_rep/;
            }

            print MFILE $_ . "\n";
        }
    } #end check each line of file
    close INFILE;
} # end foreach ARGV

# parse out Funcs
foreach $filename (@lib_files) {
    $write_bool = 0;
    $linenum = 0;
    open (INFILE, $filename);
    while (<INFILE>) {
        $linenum++;

```

```

chomp;
if (/^$funcid\s+(.+?) (\s|\{)/) {
    $write_bool = 1;
    foreach $name (@func_names) {
        if ($1 eq $name) {
            print "Error: Duplicate definition of function $name in
file $filename \ (line $linenum)\ \n";
            exit;
        }
    }
    push @func_names, $1
} elsif (/^($messageid|$taskid|$rcxid|$mainid)/) {
    $write_bool = 0;
}
if ($write_bool) {
    while (my($def_fake, $def_rep) = each(%define_sets) ) {
        $_ =~ s/$def_fake/$def_rep/;
    }

    print MFILE $_ . "\n";
}
} #end check each line of file
close INFILE;
} # end foreach ARGV

# parse out Tasks
foreach $filename (@lib_files) {
    $write_bool = 0;
    $linenum = 0;
    open (INFILE, $filename);
    while (<INFILE>) {
        $linenum++;
        chomp;
        if (/^$taskid\s+(.+?) (\s|\{)/) {
            $write_bool = 1;
            #libraries can't have main

            if ($1 eq "main") {
                if ($filename ne @ARGV[0]) {
                    print "Error: Libraries cannot have a main function\n";
                    exit;
                }
                last;
            }
            foreach $name (@task_names) {
                if ($1 eq $name) {
                    print "Error: Duplicate definition of task $name in file
$filename \ (line $linenum)\ \n";
                    exit;
                }
            }
            push @task_names, $1;
        } elsif (/^($messageid|$rcxid|$funcid|$mainid)/) {
            $write_bool = 0;
        }
    }
    if ($write_bool) {

```

```

        while (my($def_fake, $def_rep) = each(%define_sets) ) {
            $_ =~ s/$def_fake/$def_rep/;
        }

        print MFILE $_ . "\n";
    }
} #end check each line of file
close INFILE;
} # end foreach ARGV

#finally grab main function
$write_bool = 0;
open (INFILE, @ARGV[0]);
while (<INFILE>) {
    chomp;
    if (/^$mainid/) { $write_bool = 1; }
    elsif (/^($messageid|$rcxid|$funcid|$taskid)/) { $write_bool = 0; }
    if ($write_bool) {

        while (my($def_fake, $def_rep) = each(%define_sets) ) {
            $_ =~ s/$def_fake/$def_rep/;
        }

        print MFILE $_ . "\n";
    }
}

close MFILE;

```

---

```
#!/usr/bin/perl
```

```

#
# This is a Translator from UNI-OGEL to NQC designed for the RCX 1.0
#
# Author: Charles O'Donnell <cwo4@columbia.edu>
#
# RCX 1.0 Has the following requirements upon the NQC code...
# 1) no more than 8 subfunctions
# 2) no more than 10 tasks
# 3) functions cannot call other functions since they're inlined
# 4) No recursion (from above as well)
#
#
#

```

```

#####
# General Constants
$ENDLINE = "\n";
$FUNCCALL_TOK = "--FUNCCALL--";
$WHEN_TASK_NAME_PREFIX = "uni_ogel_whens_task";
$WHEN_TOK_SEPARATOR = "\~\~\~";

```

```

#####
# Constants for RCX 1.0 Settings

```

```

$MAX_SUBFUNC_NUMBER = 8;
$MAX_TASK_NUMBER = 10;
$MAX_WHEN_TASKS = 1;

$MAX_GLOBAL_VARS = 32;
$MAX_LOCAL_VARS = 0;

$uniogel_filename = @ARGV[0];
#####

open (INFILE, $uniogel_filename);

    $rcx_id = "<>";      # name of current rcx instance
    $RCX_OUT_FILE;      # handle for output to current RCX

    $rcx_total_tasks = 0; # total tasks the rcx has
    @all_task_names;     # array of ALL the tasks' names (for main
start)
    $current_task_count = 0; #keep track of task count so can add start
taskers?
    $current_task_name;   # name of task currently being parsed

    %subfunctions = (); #hash, key=subfunc name, val=scalar of whole
function
    $current_func_name; # name of the function we're currently parsing
    %subfunc_ref_counts = (); #keeps track of ref-counts, key=subfunc-
name, val=scalar

    %whens_pairs = (); # key = condition-string val=function-call-
string

while (<INFILE>) {
    chomp;

    #first see if we've hit a 'NEW RCX' issue
    if (/^new-rcx:(.+?):/) {
        $rcx_id = $1;

        ### Go through and finish up the LAST RCX, ie. do main and when
task
        # first print out our WHENs task if necessary...
        if (keys(%whens_pairs) > 0) {
            print RCX_OUT_FILE "task ".$WHEN_TASK_NAME_PREFIX."()
{". $ENDLINE;
            print RCX_OUT_FILE " while (1) {". $ENDLINE;
            while( ($when_cond, $when_funccall) = each(%whens_pairs) ) {
                print RCX_OUT_FILE " if ".$when_cond." {
".$when_funccall." }". $ENDLINE;
            }
            print RCX_OUT_FILE " }". $ENDLINE;
            print RCX_OUT_FILE "}". $ENDLINE;
        }

        # Now change 'this' task to the 'main' task and print it out

```

```

print RCX_OUT_FILE "task main() {"$.ENDLINE;
foreach $this_task_name (@all_task_names) {
    print RCX_OUT_FILE "start ".$this_task_name.";".ENDLINE;
}
if (keys(%whens_pairs) > 0) {
    print RCX_OUT_FILE "start $WHEN_TASK_NAME_PREFIX;".ENDLINE;
}
print RCX_OUT_FILE "}".ENDLINE;

# First check to close last RCX file
if (rcx_id eq "<>") {
    close (RCX_OUT_FILE);
}

#clear out old data structures
%subfunctions = ();
%subfunc_ref_counts = ();
%whens_pairs = ();
@all_task_names = ();
$rcx_total_tasks = 0;
$current_task_count = 0;
$current_task_name = "";
$current_func_name = "";

#OK, lets open a new file which will hold the data for this RCX
$rcx_output_file = ">" . $rcx_id . ".nqc";
open (RCX_OUT_FILE, $rcx_output_file);
next;
}

elsif (/^$rcx_id-total-tasks:\s(.+?)/) {
    $rcx_total_tasks = $1;
    # if too many, cut out right away...
    if ($rcx_total_tasks > $MAX_TASK_NUMBER) {
        print "RCX10TRANS: RCX ".$rcx_id." too many Tasks".ENDLINE;
        print "RCX10TRANS:      RCX 1.0 NQC code cannot be
generated".ENDLINE;
        exit(0);
    }
}

elsif (/^$rcx_id-message:\s(.*)/) {
    #just print out global variable...
    print RCX_OUT_FILE "int $1=0;".ENDLINE;
}

elsif (/^$rcx_id-recur-funcs:\s(.*)/) {
    #rcx 1.0 wouldn't handle recursion, die right now...
    if ($1 ne "") {
        print "RCX10TRANS: RCX ".$rcx_id." has recursive subfunction
calls".ENDLINE;
        print "RCX10TRANS:      RCX 1.0 NQC code cannot be
generated".ENDLINE;
        exit(0);
    }
}
}

```

```

elseif (/^$rcx_id-subfunc-ref:(.+?):(.+?)/) {
    # insert the ref count into hash
    $subfunc_ref_counts{$1} = $2;
}

elseif (/^$rcx_id-
when:$WHEN_TOK_SEPARATOR(.+?)$WHEN_TOK_SEPARATOR(.+)/) {
    # Eep!, WHEN!!!
    $whens_pairs{$1} = $2;
}

elseif (/^$rcx_id-subfunc:(.+?):\s\s\s(.+)/) {
    # Save the function to a hash with a LONG scalar... (keep
appending)
    $current_function_name = $1;
    $func_statement = $2;
    chomp($func_statement);

    #if ($func_statement =~ /^$FUNCCALL_TOK(.+)/) {
    # print "RCX10TRANS: RCX ".$rcx_id.", subfunction
".$current_function_name;
    # print " calls another function".$ENDLINE;
    # print "RCX10TRANS:    RCX 1.0 NQC code cannot be
generated".$ENDLINE;
    # exit(0);
    #}

    $func_statement =~ s/$FUNCCALL_TOK//;

    print RCX_OUT_FILE $func_statement.$ENDLINE;
}

elseif (/^$rcx_id-task:(.+?):\s\s\s(.+)/) {
    # Write out Task info...
    if ($current_task_name ne $1) {
        $current_task_name = $1;
        $current_task_count++;
        push @all_task_names, $current_task_name;

        # need to leave space for any WHEN tasks else there's too
many!

        # remember to count the Main statement
        if ($current_task_count > $MAX_TASK_NUMBER - 1 -
$MAX_WHEN_TASKS) {
            print "RCX10TRANS: RCX ".$rcx_id." contains too many
Tasks".$ENDLINE;
            print "RCX10TRANS:    RCX 1.0 NQC code cannot be
generated".$ENDLINE;
            exit(0);
        }
    }
    $task_statement = $2;
    chomp($task_statement);

    $task_statement =~ s/$FUNCCALL_TOK//;
    print RCX_OUT_FILE $task_statement.$ENDLINE;
}

```



```

    }

} # end while every line in uni-ogel file

close RCX_OUT_FILE;

```

---

```

#!/usr/bin/perl

#
# This is a Translator from UNI-OGEL to NQC designed for the RCX 1.0
#   This is an OPTIMIZING COMPILER
#
# Author: Charles O'Donnell <cwo4@columbia.edu>
#
# RCX 1.0 Has the following requirements upon the NQC code...
# 1) no more than 8 subfunctions
# 2) no more than 10 tasks
# 3) functions cannot call other functions since they're inlined
# 4) No recursion (from above as well)
#
# Some Optimizations:
# 1) main task startup folded into other tasks
# 2) FIXME-INC multiple when tasks allowed for quicker when reactance
# 3) FIXME-INC functions with no parameters converted to subfunctions
when possible
# 4) FIXME-INC allow functions to call other functions as long as
there's no recursion
#

#####
# General Constants
$ENDLINE = "\n";
$FUNCCALL_TOK = "--FUNCCALL--";
$WHEN_TASK_NAME_PREFIX = "uni_ogel_whens_task";
$WHEN_TOK_SEPARATOR = "~~~";

#####
# Constants for RCX 1.0 Settings

$MAX_SUBFUNC_NUMBER = 8;
$MAX_TASK_NUMBER = 10;
$MAX_WHEN_TASKS = 1;

$MAX_GLOBAL_VARS = 32;
$MAX_LOCAL_VARS = 0;

$uniogel_filename = @ARGV[0];
#####

open (INFILE, $uniogel_filename);

```

```

$rcx_id = "<>";      # name of current rcx instance
$RCX_OUT_FILE;      # handle for output to current RCX

$rcx_total_tasks = 0; # total tasks the rcx has
@all_task_names;    # array of ALL the tasks' names (for main
start)
$current_task_count = 0; #keep track of task count so can add start
taskers?
$current_task_name;  # name of task currently being parsed

%subfunctions = (); #hash, key=subfunc name, val=sclar of whole
function
$current_func_name; # name of the function we're currently parsing
%subfunc_ref_counts = (); #keeps track of ref-counts, key=subfunc-
name, val=sclar

%whens_pairs = (); # key = condition-string val=function-call-
string

while (<INFILE>) {
  chomp;

  #first see if we've hit a 'NEW RCX' issue
  if (/^new-rcx:(.+?):/) {
    $rcx_id = $1;

    # First check to close last RCX file
    if (rcx_id eq "<>") {
      close (RCX_OUT_FILE);
    }

    #clear out old data structures
    %subfunctions = ();
    %subfunc_ref_counts = ();
    %whens_pairs = ();
    @all_task_names = ();
    $rcx_total_tasks = 0;
    $current_task_count = 0;
    $current_task_name = "";
    $current_func_name = "";

    #OK, lets open a new file which will hold the data for this RCX
    $rcx_output_file = ">" . $rcx_id . ".nqc";
    open (RCX_OUT_FILE, $rcx_output_file);
    next;
  }

  elsif (/^$rcx_id-total-tasks:\s(.+?)/) {
    $rcx_total_tasks = $1;
    # if too many, cut out right away...
    if ($rcx_total_tasks > $MAX_TASK_NUMBER) {
      print "RCX10TRANS: RCX ".$rcx_id." too many Tasks".$ENDLINE;
      print "RCX10TRANS:      RCX 1.0 NQC code cannot be
generated".$ENDLINE;
      exit(0);
    }
  }
}

```

```

elseif (/^$rcx_id-message:\s(.*)/) {
    #just print out global variable...
    print RCX_OUT_FILE "int $1=0;".$ENDLINE;
}

elseif (/^$rcx_id-recur-funcs:\s(.*)/) {
    #rcx 1.0 wouldn't handle recursion, die right now...
    if ($1 ne "") {
        print "RCX10TRANS: RCX ".$rcx_id." has recursive subfunction
calls".$ENDLINE;
        print "RCX10TRANS:      RCX 1.0 NQC code cannot be
generated".$ENDLINE;
        exit(0);
    }
}

elseif (/^$rcx_id-subfunc-ref:(.+?):(.+?)/) {
    # insert the ref count into hash
    $subfunc_ref_counts{$1} = $2;
}

elseif (/^$rcx_id-
when:$WHEN_TOK_SEPARATOR(.+?)$WHEN_TOK_SEPARATOR(.*)/) {
    # Eep!, WHEN!!!
    $whens_pairs{$1} = $2;
}

elseif (/^$rcx_id-subfunc:(.+?):\s\s\s(.+)/) {
    # Save the function to a hash with a LONG scalar... (keep
appending)
    $current_function_name = $1;
    $func_statement = $2;
    chomp($func_statement);

    #if ($func_statement =~ /$FUNCCALL_TOK/) {
        # print "RCX10TRANS: RCX ".$rcx_id.", subfunction
".$current_function_name;
        # print " calls another function".$ENDLINE;
        # print "RCX10TRANS:      RCX 1.0 NQC code cannot be
generated".$ENDLINE;
        # exit(0);
    #}

    # $subfunctions{$current_function_name} =
#
$subfunctions{$current_function_name}.$func_statement.$ENDLINE;

    $func_statement =~ s/$FUNCCALL_TOK//;
    print RCX_OUT_FILE $func_statement.$ENDLINE;
}

elseif (/^$rcx_id-task:(.+?):\s\s\s(.+)/) {
    # Write out Task info...
    if ($current_task_name ne $1) {

```

```

$current_task_name = $1;
$current_task_count++;
push @all_task_names, $current_task_name;

# need to leave space for any WHEN tasks else there's too
many!
if ($current_task_count > $MAX_TASK_NUMBER - $MAX_WHEN_TASKS)
{
    print "RCX10TRANS: RCX ".$rcx_id." contains too many
Tasks".$ENDLINE;
    print "RCX10TRANS:    RCX 1.0 NQC code cannot be
generated".$ENDLINE;
    exit(0);
}

#if this is the Last task in the list, then add the MAIN info
to the beginning of the task
if ($current_task_count == $rcx_total_tasks) {

    # first print out our WHENs task...
    if (keys(%whens_pairs) > 0) {
        print RCX_OUT_FILE "task ".$WHEN_TASK_NAME_PREFIX."()
".$ENDLINE;
        print RCX_OUT_FILE "  while(1) {".$ENDLINE;
        while( my ($when_cond, $when_funccall) =
each(%whens_pairs) ) {
            print RCX_OUT_FILE "    if ".$when_cond." {
".$when_funccall." }".$ENDLINE;
        }
        print RCX_OUT_FILE "  }".$ENDLINE;
        print RCX_OUT_FILE "}".$ENDLINE;
    }

    # Now change 'this' task to the 'main' task and print it
out
    print RCX_OUT_FILE "task main() {".$ENDLINE;
    foreach $this_task_name (@all_task_names) {
        #dont startup the task we changed to main ;)
        if ($current_task_name ne $this_task_name) {
            print RCX_OUT_FILE "start
".$this_task_name.";".$ENDLINE;
        }
    }
    if (keys(%whens_pairs) > 0) {
        print RCX_OUT_FILE "start
$WHEN_TASK_NAME_PREFIX;".$ENDLINE;
    }

    next; #ignore the first line statement b/c thats just the
task name decl anyways
}

}
$task_statement = $2;
chomp($task_statement);

$task_statement =~ s/$FUNCCALL_TOK//;

```

```

        print RCX_OUT_FILE $task_statement.$ENDLINE;
    }
} # end while every line in uni-ogel file
close RCX_OUT_FILE;

```

---

## Supporting Structures for the Back-End

```

// Author: Charles O'Donnell <cwo4@columbia.edu>

import java.util.*;

public class SourceHolder {

    List contents_list = new ArrayList();    //ArrayList full of
Strings or StringBuffers (only call .toString from)
                                           //which hold lines of
'real' source

    //Map to hold the name of a function, and the number of times its
called (ref count)
    Map subfunc_ref_cnt = new TreeMap();    //key=String func-name,
val=Integer obj ref cnt

    //fake port names (unmapped) -- Strings!!
    public List in_ports = new ArrayList();
    public List out_ports = new ArrayList();
}

```

---

```

// Author: Charles O'Donnell <cwo4@columbia.edu>

import java.util.*;

public final class wFunction extends SourceHolder {

    public String name;

    //incoming names (might need rename in later stage)
    Map paramnames = new TreeMap();    //key=String param-name
val=String type

    public wFunction() {
    }
} //end class Function

```

---

```

// Author: Charles O'Donnell <cwo4@columbia.edu>

```

```

import java.util.*;

public final class wRCX {

    Map portmaps = new TreeMap(); //key=String fake-name val=String
real-name
    Map typemaps = new TreeMap(); //key=String fake-name val=String
TYPE_NAME

    public String a_port;
    public String b_port;
    public String c_port;
    public String one_port;
    public String two_port;
    public String three_port;

    public String type_one_port;
    public String type_two_port;
    public String type_three_port;

    //holds links to Tasks this wRCX is connected to...
    Set task_bindings = new HashSet(); //val= wTask Obj

    //holds the names of the recursive functions this rcx contains
    Set recur_func_names = new TreeSet(); //val=String

    //Map to hold the name of a function, and the number of times its
called (ref count)
    Map subfunc_ref_cnt = new TreeMap(); //key=String func-name,
val=Integer obj ref cnt

    //WHEN's two-pair vector (map) with two strings
    // key=if-conditional (with parens) val=f=subfunc call (full)
    // the portmappings have *ALREADY* been applied to these guys...
    Map when_pairs = new TreeMap();

    Map when_funcs = new TreeMap(); //key=String func-name
val=Function object

    public wRCX () {
    }

    public void update_portmap() {
        if (a_port!=null) { portmaps.put(a_port, "OUT_A"); }
        if (b_port!=null) {
            System.out.println("placing B");
portmaps.put(b_port, "OUT_B"); }
        if (c_port!=null) { portmaps.put(c_port, "OUT_C"); }
        if (one_port!=null) { portmaps.put(one_port, "SENSOR_1"); }
        if (two_port!=null) { portmaps.put(two_port, "SENSOR_2"); }
        if (three_port!=null) { portmaps.put(three_port, "SENSOR_3"); }

        if (type_one_port!=null) { typemaps.put(one_port, type_one_port);
    }
        if (type_two_port!=null) { typemaps.put(two_port, type_two_port);
    }
    }
}

```

```

        if (type_three_port!=null) { typemaps.put(three_port,
type_three_port); }
    }

} //end class RCX



---


// Author: Charles O'Donnell <cwo4@columbia.edu>

import java.util.*;

public final class wTask extends SourceHolder {

    public String name;

    //WHEN's two-pair vector (map) with two strings...
    // key=if-conditional (with parens)    val=f=subfunc call (full)
    // both of these might need to be remapped...
    Map when_pairs = new TreeMap();

    Map when_funcs = new TreeMap(); //key=String fun-name
    val=Function obj

    public wTask() {
    }
} //end class Task

```