Patternizer

Yianni Alexander
Marinos Constantinides
Boriana Ditcheva
Yavor Tchakalov
Adam Vartanian

December 21, 2005

Contents

Chapter 1

Introduction

1.1 Overview

1.1.1 Description of the Language

The language will be used to create geometrical patterns, which would otherwise be difficult and time-consuming to create, unless one has knowledge of graphic design tools. The language works by combining points and lines into one object or entity (called a *pattern* in our language). This pattern can be a point, a line, a pentagon, or simply a collection of lines. The user can then manipulate this pattern further by scaling, translating, and rotating. Patterns can also be defined to include other patterns within them; thus, arbitrarily complex geometric patterns can be created recursively.

1.1.2 Fundamentals: Primitives

At the core of our language lies the concept of a pattern. A pattern is essentially a template of how to draw something. An actual copy of a pattern we will call an instance. Patterns feature three intrinsic properties: origin, angle, and scale. The values of these properties are not absolute. They are calculated in relation to the parent pattern, the pattern which was used to create the current one. Each pattern comes equipped with a default constructors that defaults its intrinsic properties to $(0, 0), 0\,^{\circ}, 1.0$ respectively. The language also provides default accessors and mutators for its intrinsic properties. A pattern can also have custom properties defined by the user to avoid redundant coding of new patterns.

There exists the concept of a *native* pattern in Patternizer. That is the *Point* pattern, which represents a single point. A Point will be viewed as a vector with regard to its intrinsic properties. Thus scaling an instance of a Point (e.g. modifying its scale from 1.0 to 2.0) will apply vector scaling with respect to the origin of the Point instance.

The remaining native data types or primitives will be floating point numbers.

1.1.3 Fundamentals: Operators

Real numbers will have all basic arithmetic operators defined on them: addition, subtraction, multiplication, and division.

Points are expressed as a pair: (x, y), and the '$->$' operator will be used to create lines between points, e.g. $(x_1, y_1) -> (x_2, y_2)$

Scaling, rotating, and translating can be applied globally to the entire pattern file, or within a pattern definition.

1.1.4 Miscellaneous

Standard control structures like for, while, if, else would also be included, with normal logical operators. == and != would be the only valid operators on objects. Printing would be allowed for debugging purposes, and there are also options to include other defined pattern files, and set background and foreground color.

1.2 Advantages

1.2.1 Why Learn Patternizer?

A user will be able to quickly learn this simple and concise language and be able to create arbitrarily complex patterns from simple geometric primitives like lines, points, and squares. One of the strengths of the language is its recursive nature. This feature not only adds power to what a user can do with the language, but also encourages the user to think in a different way about patterns. The user thinks about new patterns in terms of combining already existing ones, thus simplifying the creative process.

Chapter 2

Tutorial

Pattern files are definitions of graphical patterns to be drawn. They are simply a collection and organization of points and lines, of which other patterns are constructed, and all assembled together to form the parent pattern. A point is represented as a pair of reals: `(25, 50)`. A line is constructed by connecting points using the line operator: `->`. For example then, two points can be connected to form a line as such: `(25,50)->(50,50)`. These points, lines, and patterns can be manipulated using methods such as `move, scale,` and `rotate`. Control flow of the program is dictated using loops, conditionals, declarations, and operations.

Patterns are declared by the keyword `pattern`, followed by an identifier for the pattern, a list of parameters, and a set of statements declaring how to draw the pattern.

2.1 Sample Programs

2.1.1 Sample Program 1: Line

The simplest nontrivial pattern created by Patternizer is a straight line.

```
pattern line(){
    (0,0)->(40,0);
}
```

2.1.2 Sample Program 2: Rectangle

Patterns are declared by the keyword pattern, followed by an identifier for the pattern, a list of parameters (somehow), and a set of statements declaring how to draw the object. Thus, for example, if Line() produced a vertical line from $(0, 0)$ to $(0, 1)$, we could have:

```
pattern Line() {
```

```
    (0,0)->(1,0);
    move(1,0); //move cursor to end of line
}
pattern Rectangle() {
    for (x=1;x<5;x=x+1;) {
        Line();
        rotate(3.14/2);
    }
}
```

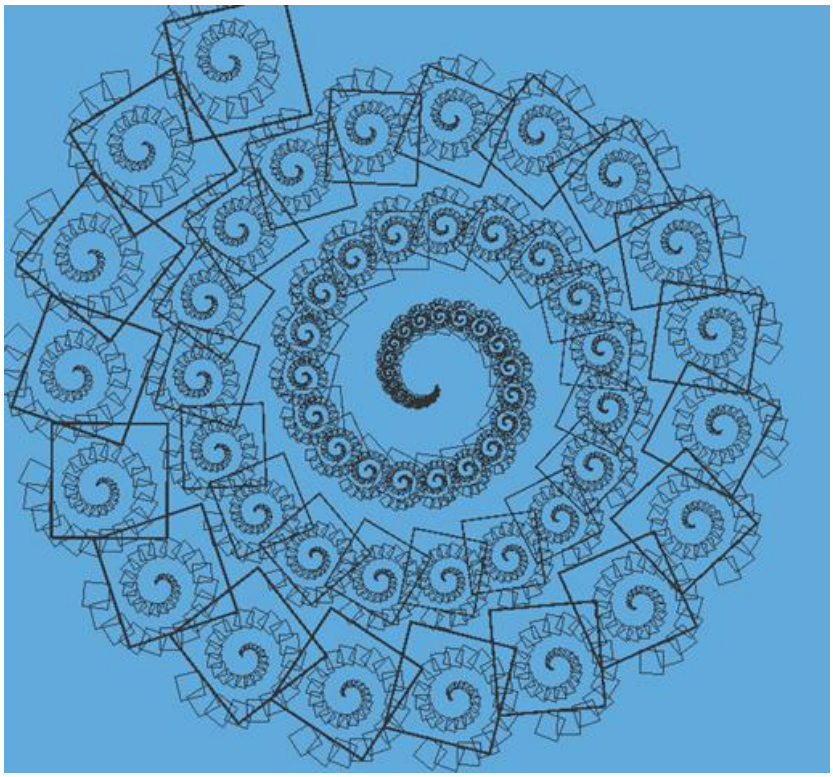
2.2 Getting an Image From Your Pattern File

In the console type paste the Patternizer code and then hit 'Ctrl-Z'.

2.3 Even Cooler Sample Program

Yeah, look at what this sample program makes.

```
set_background_color(97, 170, 220);
move(375, 330);
pattern spiral(){
    (0,0)->(40,0)->(40,40)->(0,40)->(0,0);
    move(24,18);
    scale(0.15);
    for(x=1; x<40; x=x+1;){
        scale(x/26.5);
        rotate(x/3);
        z=1+x/100;
        move(15*z,15*z)
        (0,0)->(40,0)->(40,40)->(0,40)->(0,0);
        scale(26.5/x);
        rotate(x/3-2*x/3);
    }
}
for(x=1; x<71; x=x+1;){
    scale(x/26.5);
    rotate(x/3);
    z=1+x/100;
    move(15*z,15*z)
    spiral();
    scale(26.5/x);
    rotate(x/3-2*x/3);
}
```



Chapter 3

Language Reference Manual

3.1 Lexical Conventions and Tokens

Tokens are punctuation, comments, identifiers, number constants, strings, reserved keywords, and operators.

3.1.1 Comments

You can place comments in the text by either using /* ... */ or for a single line //.

3.1.2 Identifiers

Identifiers must begin with a letter or an underscore symbol '_', but then can consist of any sequence of digits, letters, and the underscore symbol. Upper and lower case letters are different. Identifiers can be used to identify the 4 types in our language: real numbers, points, patterns, and strings.

3.1.3 Numbers Constants

Numbers consists of digits, optional decimal point, and optional 'e' followed by a signed integer exponent. We will only be using real numbers.

3.1.4 Strings

Strings are text that begin with “ and end with ”.

3.1.5 Keywords

The following are reserved for use as keywords, and may not be used otherwise:

```
pattern  if     else   for
while    move   scale  rotate
setFG    setBG  print  include
```

3.1.6 Operators and Other Tokens

Characters or sequences of characters used in the language:

```
{    }    (    )    +    -
*    /    ;    ,    =    ->
==   !=   +=   -=   *=   /=
<    >    <=   >=   &&   ||
!    "    //   /*   */
```

3.2 Types

We have only 3 types which can be represented using identifiers. There are real numbers, points, and strings. There are no type identifiers, except for the declaration of patterns, thus our language is not statically-typed.

3.2.1 Reals

The real numbers can be maintained as 64-bit floating point numbers.

3.2.2 Points

Points are a pair of two real numbers, represented by an open parentheses, a real number corresponding to the horizontal coordinate, a comma, another real number corresponding to the vertical coordinate, then finally following a closing parenthesis. An example of a point would be:
`(0,1)`.

3.2.3 Strings

String identifiers and variables are allowed in our language. Strings will be used to print statements, mostly for debugging purposes. Strings begin with " and with ".

3.3 Expressions

3.3.1 Primary Expressions

Primary expressions are the basic element expressions of our language. These can include any of our types, either as identifiers or constants, as well as the representation of a point.

3.3.2 Arithmetic

An arithmetic expression is to be evaluated using obvious methods. Sometimes the pattern parameters will be determined using some sort of function and those functions are evaluated as arithmetic expressions. Additional manipulation will be available as stand-alone expressions. Arithmetic expressions take primary expressions as operands. Any mathematical operation on real numbers is allowed, such as +, -, *, and /. Arithmetic expressions for points is limited to translation (+) and scaling (*). These will be performed on the points as they would be on vectors.

3.3.3 Relational

Similar to arithmetic, relational expressions will be helpful in pattern and control flow. Boundary conditions will be necessary when constructing patterns and comparisons will be allowed using standard notation(these operations will be limited to reals): <, >, <=, >=. Relational expressions can also involve patterns and points, but are limited to the operations of != and ==. Of course these operations can also be performed on reals.

3.3.4 Logical

Logical expressions will be used in conditional and looping statements. They will essentially evaluate to either 0 or 1, which will determine program flow on conditionals and loops. Operators associated with logical expressions are: &&, ||, and !, which denote “and”, “or”, and “not”, respectively.

3.3.5 Operational

The arrow operator ’->’ connects two points or two lines (or a point to a line). It is automatically evaluated to draw the resulting segment on the screen. Therefore, the simplest form of a line would involve two points:
`(0,0)->(1,1);`
It is also possible to connect points to already existing lines, by drawing a line from the last point in the sequence, to the either the new point, or the first point of the new line.
Thus,
`(0,0)->(1,1)->(2,2);`
is also acceptable syntax.

3.3.6 Precedence of Operators

Below is a table of precedence for our operators.

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

3.4 Statements

Statements are used primarily for control flow as the major usage of our language is in displaying patterns which are realized as expressions. Thus our statements are rudimentary and necessary.

3.4.1 Transformation Statements

Patterns can be transformed through transformation statements. These statements are limited to: `scale`, `move`, and `rotate`. These statements can be applied either locally or globally. If they are made outside of a pattern, they apply to the global environment. If they are declared within the pattern, they only apply locally to the current pattern. All patterns that stem from an original pattern inherit the transformation properties.

3.4.2 Assignments

Assignments are made using the '=' operator. Point identifiers can be assigned the values of other points using other point identifiers or explicit definition of points. Patterns cannot be assigned explicit patterns, but rather are defined using explicit patterns.

3.4.3 Loops

There are two kinds of loops used in the language: `for` and `while`. Standard C-style syntax will be used in construction of loop statements.

3.4.4 Conditional

Conditional statements will be allowed using the standard if-else paradigm, with a few specific tailorings. First of all, the only keywords associated with conditional statements are `if` and `else`. A conditional statement is in the form:

```
if ( <logical expression> )
    <statement>
```

or

```
if ( <logical expression> )
```

```
    <statement>
else
    <statement>
```

3.4.5 Include

Other pattern files can be included by specifying which pattern files to include before any explicitly patterns in the file are defined. To include a pattern file, the line
`include "filename";`
must appear before any explicit pattern definitions. Multiple inclusions are allowed.

3.5 Pattern Definition

Patterns are definitions of how to draw things. They can contain points connected to form lines and other patterns. Only line segments can be drawn. If one wants to draw a point, she must draw a small line centered at the point. Patterns are defined by newly created identifiers followed by an open curly brace '{'. Thus, to make a rectangle pattern, the syntax would appear as:

```
pattern my_rectangle {
    (0,0)->(2,0)->(2,2)->(0,2)->(2,2);
}
```

3.6 Internal Functions

3.6.1 Print Function

Printing strings out to console is allowed in our language, mostly for debugging purposes and printing pattern parameters. It would be possible to print identifiers as well as strings. A print would appear like this:
`print ("x = "x", y = "y);`

3.6.2 Color Settings

You can set the background color using an internal function
`set_BG{real red, real blue, real green}`
as well as the foreground color using
`set_FG{real red, real blue, real green}`
where the values of `red`, `blue`, and `green` must all be between 0 and 255.

Chapter 4

Project Plan

4.1 Project Timeline

4.1.1 Brainstorming

Our first idea of Patternizer was envisioned slightly different than what we expected, but essentially turned out to be the same thing. We wanted a way of making neat little geometric patterns and replicating them to form more decorated patterns. The now recursive nature of Patternizer, eluded us in our initial thoughts. Once we decided on creating a language that abstracts and simplifies the design of repetitive geometric figures, we brainstormed as to what features would be implemented. We also were critically analyzing the structure and design of our ideas as proposing them.

4.1.2 Proposal

After careful thought and discussion with Professor, we trimmed our ideas into a solid coherent vision of what Patternizer would do. We wanted to keep it simple, but yet allow building on and up of those simple structures to create complex ones. Therefore, there are no "complex" functions in our language; but maintaining a geometrically rudimentary design as possible to avoid conflict within language functionality.

4.1.3 Language Reference Manual

The Language Reference Manual was the first step towards coming to a formal definition of our language. In it we carefully define the syntax and semantics of our language. Some things have changed since our initial version, but nothing affecting the central core of our language.

4.1.4 Development

The first thing we decided on was a working environment that allowed for ease of merging all our tools together. By suggestion of Yavor, we decided to use Eclipse as our developing environment. Eclipse packs a lot functionality so its easy assimilate the various technical aspects of our developing environment, such as ANTLR. There is also built-in team project synchronization and SVN repository support. The first step was coming up with the grammar, and from there, we easily generated the lexer, parser, and tree walker. Afterwards, there were two aspects of back-end needing implementation: the graphical, and everything else. During the entire process, constant "local" testing was done by each individual for their respective responsibility.

4.1.5 Testing

Testing was very important to the success of our language, as mistakes would be easily noticed as our language was utilized to display graphical patterns, and its pretty easy to see mistakes in images. Testing increased in complexity as time went on. The first steps were to test the basic functionality and logic of our language. Make sure points could be positioned and lines drawn, as well as simple control flow structures like loops and conditionals. Final stages of testing involved testing increasingly complex patterns to push the limits of our language.

4.1.6 Documentation

Documentation began before the first line of code was written and ended after the last line of code was written. Documentation was a fairly smooth due to the very effective level of communication between each of us. We did a great job of keeping each other up-to-date on progresses and failures, as well as speculative issues.

4.2 Team Responsibilities

4.2.1 Front-End

Our super wonderful and uber gorgeous Boriana was responsible for creating the ANTLR code which generated the lexer and parser.

4.2.2 Back-End

Our Resident Genius Adam coded all of the back-end, except for the actual code responsible for drawing the graphical images. He also used ANTLR in the construction of the tree walker.

4.2.3 Graphical Back-End

Always Alert Yavor was very eager to wrap up our language and the first to put its power to the test. Yavor, heavily experience in graphics, was responsible for implementing all of the Java AWT code which painted the graphical patterns on the screen.

4.2.4 Testing

Marinos was responsible for testing the limits of our language. He first started out making sure all the basic and necessary parts worked. He then moved on to making sure non-trivial graphical patterns could be drawn. As more and more improvements and fixes were made to the language, Marinos' test program became increasingly complex to test the graphical power of Patternizer.

4.2.5 Documentation

Yianni was responsible for final documentation. This involved consulting each of the group members and critically questioning their responsibilites and experiences with the project.

Chapter 5

Architectural Design

5.1 Architecture

The Patternizer compiler consists of a few essential blocks which are common in most compilers: the lexer, parser, symbol table, the runtime environment, the code generator, and our graphics module. Our lexer is fed a patternizer file, which breaks the sequence of characters into a list of tokens. These tokens are checked for syntactical correctness by the parser, which builds an Abstract Syntax Tree which is walked by our tree walker, checking semantics. The parser and tree walker interact with the symbol table and error handlers to ensure a syntactically and semantically proper Abstract Syntax Tree. Finally, our tree walker walks our code and calls the Java functions to create the graphics on the screen. The final output will be an image generated by Java. ANTLR was used to generate the code for the lexer, parser, and tree walker. The runtime system is a collection of classes already defined. These classes provide the basic computation necessary for creation of a Patternizer pattern.

The entry point of the compiler is the class `test.java`. The `main()` method parses the patternizer source code sent to the console. The lexer is contained as an object in the parser which allows the two to interface. The parser's task is to direct the code generator and back-end on how to build to target Java code. Semantic actions on parser productions invoke a set of builder methods, which construct our AST from the source to be able to generate our Patternizer code. If a syntax error occurs, the parser will invoke its error handler. If an error occurs within semantics, the tree walker will invoke its error handler. All appropriate information is reported when an error is encountered. When parsing is finished, the tree walker is invoked and begins generating Java code. The tree walker invokes the Environment class which is responsible for handling the symbol table, scoping, coloring, graphical functionality, including transformations.

5.2 The Runtime Environment

The runtime environment provides the structure necessary to operate and organize our generated Java code in order to create the Java code necessary to create our patterns. The runtime environment is contained within our Environment.java class. The responsibilities of our runtime environment include handling scoping issues, setting background and foreground colors, maintaining and querying the symbol table, and propogating graphical functionality, as well as dictating graphical statements to allow for eventual creation of the pattern images.

5.3 Error Handling

ANTLR allowed for error handling within each appropriate block. Thus, the parser would contain its own error handler for dealing with syntax, whereas the tree walker would contain its own error handling mechanisms interfacing with the symbol table as well, which would allow for dealing with semantics. Each respective part will halt and report the type of error. The fact that the error handling was divided into each part made categorization of errors much easier. At each detection of an error, the location of logic and control in the source will determine all the specifics of the error and allow for sufficient error reporting.

Chapter 6

Test Plan

6.1 Goals

The goals of the testing project is not to detect and fix every possible bug that exists in Patternizer, but instead, to try and find possible inconsistencies that may exist in the language. Through planned implementation, every aspect of the language is tested thus providing the basis of a successful development process.

6.2 Testing Stages

For the purposes of testing, several sample programs were created, some of which were meant to be faulty, and we made sure that Patternizer would make the right decision in either compiling or identifying the possible errors. A preliminary form of testing took place throughout the three design phases of the project: frontend design, back-end design, and animation. The code was also tested while being implemented amonst the individual group members.

6.2.1 Stage 1 - Basics

Initially, a general error checking stage took place, in the sense that it was not associated directly to Patternizer. Some examples include checking that the number parameter values behave properly or that expressions can be passed as parameters. One bug that was fixed by repetitive checking of these values was the setting of foreground color. Initially, if you tried to change the foreground color of a pattern, only the final value set by `setFG()` will be applied to the patern, and it will be applied to the entire pattern (and all intermediate values the foreground color is set to are lost).

6.2.2 Stages 2 - Patternizer Graphics

The second stage involved testing the actual constructs specific to our the language(specifically move, rotate, and scale). Sample programs were created to test every possible command in the language. Through thorough testing and implementation we were able to identify and correct several bugs of the language. The initial version of the language could not accept negative numbers. For example, it was not possible to perform the command
$(0,0) -> (0-100,0)$
which draws a line connecting the points $(0,0)$ and $(-100,0)$.

6.3 Test Programs That Fully Utilize the Potential of Patternizer

Several test programs were created that fully utilize the capabilities of Patternizer. Such programs include forming series of spirals and creating beautiful wallpapers by simply repeating patterns. Some of the cool programs were very intricate and complicated designs of spirals and snowflakes.

Chapter 7

Lessons Learned

7.1 Weekly Meeting Times

We should have set up a consistent weekly meeting time. Most of the time, we planned on meeting after class, but usually schedules during the day are tight, so people couldn't dedicate a determinant amount of time. It would have been nice to have a more structured time to allot many hours and proper preparation towards more efficient project development.

7.2 Technical Tools

We should have ensured everybody's computers are set up with all tools needed to run programs (correct Java SDK version, eclipse, ANTLR, SVN). The fact that we had to use all this software to develop our language mean we had to each configure them to our unique machine. This did sometimes provide difficulty for portability and communication.

7.3 Prototyped Design

We should have prototyped our design to make sure our language will be implementable. Our project plan developing the project in entire pieces. This means we first made the lexer, and then the parser, then the tree walker, then the graphics and rest of back-end. For more efficient testing it would have helped had we developed a fully working compiler, but with very basic and necessary functionality to test as we build it up.

Chapter 8

Complete Listings

8.1 patternGrammar.g

```
/**
 * patternGrammar.g: the lexer, parser, and walker in ANTLR grammar
 *
 * authors: Boriana Ditcheva && Adam Vartanian  October 2005
 *
**/


//
//    ====================================================
//                     THE LEXER
//  ====================================================
//
class patternLexer extends Lexer;

options{
    k = 2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
}


// Error handling
{
  // Error stuff
}

// Protected rules can only be called by other rules
```

```
protected
ALPHA   : 'a'..'z' | 'A'..'Z' | '_' ;
protected
DIGIT   :   '0'..'9';


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

// The java newline() routine increments the program newline count
NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
            { $setType(Token.SKIP); newline(); }
        ;

// C-style comments
COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NL)
                    | ~( '\n' | '\r' )
                )* "*/"
            | "//" (~( '\n' | '\r' ))* (NL)
          )
        { $setType(Token.SKIP); }            // Ignore these
        ;


// Important tokens
LPAREN  : '(';
RPAREN  : ')';
LBRACE  : '{';
RBRACE  : '}';
PLUS    : '+';
MINUS   : '-';
MULT    : '*';
DIV     : '/';
SEMI    : ';';
COMMA   : ',';
ASSIGN  : '=';
ARROW   : "->";
EQ      : "==";
NEQ     : "!=";
PLUSEQ  : "+=";
MINUSEQ : "-=";
MULTEQ  : "*=";
DIVEQ   : "/=";
```

```
GT      : '>';
LT      : '<';
GE      : ">=";
LE      : "<=";
AND     : "&&";
OR      : "||";
NOT     : '!';
QOUTE   : '"';


ID  options { testLiterals = true; }
        : ALPHA (ALPHA|DIGIT)*
        ;

/* NUMBER example:
 * 1, 1e, 1., 1.e10, 1.1, 1.1e10
 */
NUMBER  : (DIGIT)+ ('.' (DIGIT)*)? (('E'|'e') ('+'|'-')? (DIGIT)+)?
        ;


STRING  : '"'!
              (  ~('"' | '\n')
               | ('"'!'"')
              )*
          '"'!
        ;







//
//    ======================================================
//                     THE PARSER
//  ======================================================
//
class patternParser extends Parser;

options{
    k = 2;
    buildAST = true;
}
```

```
tokens{
    STATEMENT;
    POINT;
    FUNC_CALL;
    ARGS;
    UNARY_MINUS;
    PAREN_EXPR;
}


// Error handling
{

}

// Tree parser Rules
program
        : ( outer_statement | pattern_def )* EOF!
            { #program = #([STATEMENT,"PROG"], program); }
        ;

outer_statement
    : bckgrnd_color_stmt
    | include_stmt
    | inner_statement
    ;

inner_statement
    : foregrnd_color_stmt
    | move_stmt
    | scale_stmt
    | rotate_stmt
    | for_stmt
    | if_stmt
    | while_stmt
    | break_stmt
    | assign_stmt
    | draw_line_stmt
    | call_stmt
    | print_stmt
    | LBRACE! (inner_statement)* RBRACE!
      { #inner_statement = #([STATEMENT], inner_statement); }
        ;
```

```
pattern_def
        : "pattern"^ ID LPAREN! pattern_args RPAREN! LBRACE! pattern_body RBRACE!
            { System.out.println("Found a pattern"); }
        ;

pattern_args
        : ID (COMMA! ID)*
            {#pattern_args = #([ARGS,"ARGS"], pattern_args); }
        |
            {#pattern_args = #([ARGS,"ARGS"], pattern_args); }
        ;

pattern_body
        : (inner_statement)+
            { #pattern_body = #([STATEMENT], pattern_body); }
        ;

foregrnd_color_stmt
        : "set_FG"^ LPAREN! expression COMMA! expression COMMA! expression RPAREN! SEMI!
            { System.out.println("Found set_foreground");}
        ;

bckgrnd_color_stmt
        : "set_BG"^ LPAREN! expression COMMA! expression COMMA! expression RPAREN! SEMI!
            { System.out.println("Found set_background");}
        ;

move_stmt
        : "move"^ LPAREN! expression COMMA! expression RPAREN! SEMI!
            { System.out.println("Found a move");}
        ;

scale_stmt
        : "scale"^ LPAREN! expression RPAREN! SEMI!
        { System.out.println("Found a scale");}
        ;

rotate_stmt
        : "rotate"^ LPAREN! expression RPAREN! SEMI!
        { System.out.println("Found a rotate");}
        ;

assign_stmt
        : ID ASSIGN^ (expression) SEMI!
        { System.out.println("Found an assignment"); }
```

```
        ;

for_stmt
        : "for"^ LPAREN! inner_statement expression SEMI! inner_statement RPAREN!
inner_statement
        {System.out.println("Found a for statement");}
        ;

while_stmt
        : "while"^ LPAREN! expression RPAREN! inner_statement
        { System.out.println("Found a while");}
        ;

if_stmt
        : "if"^ LPAREN! expression RPAREN! inner_statement
            (options {greedy = true;}: "else"! inner_statement )?
        { System.out.println("Found an if");}
        ;

draw_line_stmt
        : point (ARROW^ point)+ SEMI!
        {System.out.println("Found a draw line sequence.");}
        ;

call_stmt
        : pattern_call SEMI!
        ;

pattern_call
: ID LPAREN! call_args RPAREN!
            { #pattern_call = #([FUNC_CALL], pattern_call); }
        ;

call_args
        : expression (COMMA! expression)*
            {#call_args = #([ARGS,"ARGS"], call_args); }
        |
            {#call_args = #([ARGS,"ARGS"], call_args); }
        ;

include_stmt
        : "include"^ STRING SEMI!
        { System.out.println("Found an include");}
        ;

break_stmt
```

```
        : "break"^ SEMI!
        { System.out.println("Found a break");}
        ;

print_stmt
        : "print"^ LPAREN! expression (COMMA! expression)* RPAREN! SEMI!
        { System.out.println("Found a print");}
        ;

expression
        : logic_term ( "or"^ logic_term )*
        ;

logic_term
        : logic_factor ( "and"^ logic_factor )*
        ;

logic_factor
        : ("not"^)? comparison_expr
        ;

comparison_expr
        : arith_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) arith_expr )?
        ;

arith_expr
        : arith_term ( (PLUS^ | MINUS^) arith_term )*
        ;

arith_term
        : unary_minus ( (MULT^ | DIV^) unary_minus )*
        ;

unary_minus
: MINUS! arith_factor
{ #unary_minus = #([UNARY_MINUS], unary_minus);}
|
arith_factor
;

arith_factor
        : (ID | NUMBER | STRING | paren_expr | pattern_call)
        ;

paren_expr
: LPAREN! expression (COMMA! expression { #paren_expr = #([POINT], paren_expr); }
```

```
                    | {#paren_expr = #([PAREN_EXPR], paren_expr);})
  RPAREN!
;

point
        : LPAREN! (expression) COMMA! (expression) RPAREN!
            { #point = #([POINT], point); }
        ;



//
//    ======================================================
//                     THE TREE WALKER
//  ======================================================
//

{
import java.util.ArrayList;
}

class pWalker extends TreeParser;

{
    Environment e = new Environment();
    public void display(){ e.display();}
}

expr returns [Variable r]
    {
        r = null;
        Variable a, b, c;
        Variable[] args;
        String[] def_args;
    }

    : #(STATEMENT (stmt:. { r=expr(#stmt); } )* )
    | #(FUNC_CALL func:ID args=arg_list) { r = e.callPattern(this, func.getText(), args); }
    | #("set_BG" a=expr b=expr c=expr) { e.setBackground(a, b, c); }
    | #("set_FG" a=expr b=expr c=expr) { e.setForeground(a, b, c); }
    | #("include" a=expr) { e.include(a); }
    | #("pattern" fname:ID def_args=def_arg_list body:.) { e.definePattern(fname.getText(),
    def_args, body); }
    | #("print"
        (a=expr { System.out.print(a.toString() + " "); })+
      ) { System.out.println(); }
```

```
| #("for" fbefore:. fexpr:. fafter:. fbody:.)
  {
    expr(#fbefore);
    while (expr(#fexpr).isTrue()) {
      expr(#fbody);
      expr(#fafter);
    }
  }
| #("while" wexpr:. wbody:.)
  {
    while (expr(#wexpr).isTrue()) {
      expr(#wbody);
    }
  }
| #("if" a=expr ithen:. (ielse:.)?)
  {
    if (a.isTrue()) {
      expr(#ithen);
    } else if (ielse != null) {
      expr(#ielse);
    }
  }
| #("scale" a=expr) { e.scale(a); }
| #("move" a=expr b=expr) { e.move(a, b); }
| #("rotate" a=expr) { e.rotate(a); }
| #(PLUS a=expr b=expr) { r = a.plus(b); }
| #(MINUS a=expr b=expr) { r = a.minus(b); }
| #(MULT a=expr b=expr) { r = a.times(b); }
| #(DIV a=expr b=expr) { r = a.divide(b); }
| #(GT a=expr b=expr) { r = a.greaterthan(b); }
| #(LT a=expr b=expr) { r = a.lessthan(b); }
| #(GE a=expr b=expr) { r = a.greaterorequal(b); }
| #(LE a=expr b=expr) { r = a.lessorequal(b); }
| #(EQ a=expr b=expr) { r = a.equals(b); }
| #(NEQ a=expr b=expr) { r = a.notequals(b); }
| #("not" a=expr) { if (!(a instanceof DoubleVariable)) a.typeError("number"); else
r = ((DoubleVariable)a).not(); }
| #("or" a=expr rhs_or:.) { if (!(a instanceof DoubleVariable)) a.typeError("number");
else r = ((DoubleVariable)a).or(expr(#rhs_or)); }
| #("and" a=expr rhs_and:.) { if (!(a instanceof DoubleVariable))
a.typeError("number"); else r = ((DoubleVariable)a).and(expr(#rhs_and)); }
| #(UNARY_MINUS a=expr) { r = a.negative(); }
| #(PAREN_EXPR a=expr) { r = a; }
| #(ASSIGN name:ID a=expr) { e.assign(name.getText(), a); }
| ID { r = e.getSymbol(#ID.getText()); }
| NUMBER { r = new DoubleVariable(Double.parseDouble( #NUMBER.getText() )); }
```

```
    | STRING { r = new StringVariable(#STRING.getText()); }
    | #(POINT a=expr b=expr) { r = new PointVariable(a, b); }
    | #(ARROW a=expr b=expr) { e.drawLine(a, b); r = b; }
    ;

arg_list returns [Variable[] r]
    {
        r = null;
        Variable a;
        ArrayList<Variable> list;
    }

    : #(ARGS { list = new ArrayList<Variable>(); }
        ( a=expr { list.add(a); } )*
       ) { r = list.toArray(new Variable[list.size()]); }
    ;

def_arg_list returns [String[] r]
    {
        r = null;
        ArrayList<String> list;
    }

    : #(ARGS { list = new ArrayList<String>(); }
        ( id:ID { list.add(id.getText()); } )*
       ) { r = list.toArray(new String[list.size()]); }
    ;
```

8.2 patternLexer.java

```
// $ANTLR 2.7.5 (20050128): "patternGrammar.g" -> "patternLexer.java"$

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
```

```
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

/**
 * patternGrammar.g: the lexer, parser, and walker in ANTLR grammar
 *
 * authors: Boriana Ditcheva && Adam Vartanian  October 2005
 *
**/
public class patternLexer extends antlr.CharScanner implements patternLexerTokenTypes,
                                                               TokenStream
 {

  // Error stuff
public patternLexer(InputStream in) {
this(new ByteBuffer(in));
}
public patternLexer(Reader in) {
this(new CharBuffer(in));
}
public patternLexer(InputBuffer ib) {
this(new LexerSharedInputState(ib));
}
public patternLexer(LexerSharedInputState state) {
super(state);
caseSensitiveLiterals = true;
setCaseSensitive(true);
literals = new Hashtable();
literals.put(new ANTLRHashString("if", this), new Integer(52));
literals.put(new ANTLRHashString("for", this), new Integer(50));
literals.put(new ANTLRHashString("set_BG", this), new Integer(46));
literals.put(new ANTLRHashString("move", this), new Integer(47));
literals.put(new ANTLRHashString("while", this), new Integer(51));
literals.put(new ANTLRHashString("scale", this), new Integer(48));
literals.put(new ANTLRHashString("break", this), new Integer(55));
literals.put(new ANTLRHashString("or", this), new Integer(57));
literals.put(new ANTLRHashString("rotate", this), new Integer(49));
literals.put(new ANTLRHashString("else", this), new Integer(53));
```

```
literals.put(new ANTLRHashString("pattern", this), new Integer(44));
literals.put(new ANTLRHashString("include", this), new Integer(54));
literals.put(new ANTLRHashString("set_FG", this), new Integer(45));
literals.put(new ANTLRHashString("not", this), new Integer(59));
literals.put(new ANTLRHashString("and", this), new Integer(58));
literals.put(new ANTLRHashString("print", this), new Integer(56));
}

public Token nextToken() throws TokenStreamException {
Token theRetToken=null;
tryAgain:
for (;;) {
Token _token = null;
int _ttype = Token.INVALID_TYPE;
resetText();
try {   // for char stream error handling
try {   // for lexical error handling
switch ( LA(1)) {
case '\t':  case ' ':
{
mWS(true);
theRetToken=_returnToken;
break;
}
case '\n':  case '\r':
{
mNL(true);
theRetToken=_returnToken;
break;
}
case '(':
{
mLPAREN(true);
theRetToken=_returnToken;
break;
}
case ')':
{
mRPAREN(true);
theRetToken=_returnToken;
break;
}
case '{':
{
mLBRACE(true);
theRetToken=_returnToken;
```

```
break;
}
case '}':
{
mRBRACE(true);
theRetToken=_returnToken;
break;
}
case ';':
{
mSEMI(true);
theRetToken=_returnToken;
break;
}
case ',':
{
mCOMMA(true);
theRetToken=_returnToken;
break;
}
case '&':
{
mAND(true);
theRetToken=_returnToken;
break;
}
case '|':
{
mOR(true);
theRetToken=_returnToken;
break;
}
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case '_':  case 'a':
case 'b':  case 'c':  case 'd':  case 'e':
case 'f':  case 'g':  case 'h':  case 'i':
case 'j':  case 'k':  case 'l':  case 'm':
case 'n':  case 'o':  case 'p':  case 'q':
case 'r':  case 's':  case 't':  case 'u':
case 'v':  case 'w':  case 'x':  case 'y':
case 'z':
```

```
{
mID(true);
theRetToken=_returnToken;
break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
mNUMBER(true);
theRetToken=_returnToken;
break;
}
default:
if ((LA(1)=='/') && (LA(2)=='*'||LA(2)=='/')) {
mCOMMENT(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='>')) {
mARROW(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='=') && (LA(2)=='=')) {
mEQ(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='!') && (LA(2)=='=')) {
mNEQ(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='+') && (LA(2)=='=')) {
mPLUSEQ(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='-') && (LA(2)=='=')) {
mMINUSEQ(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='*') && (LA(2)=='=')) {
mMULTEQ(true);
theRetToken=_returnToken;
}
else if ((LA(1)=='/') && (LA(2)=='=')) {
mDIVEQ(true);
theRetToken=_returnToken;
}
```

```
else if ((LA(1)==’>’) && (LA(2)==’=’)) {
mGE(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’<’) && (LA(2)==’=’)) {
mLE(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’"’) && (_tokenSet_0.member(LA(2)))) {
mSTRING(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’+’) && (true)) {
mPLUS(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’-’) && (true)) {
mMINUS(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’*’) && (true)) {
mMULT(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’/’) && (true)) {
mDIV(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’=’) && (true)) {
mASSIGN(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’>’) && (true)) {
mGT(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’<’) && (true)) {
mLT(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’!’) && (true)) {
mNOT(true);
theRetToken=_returnToken;
}
else if ((LA(1)==’"’) && (true)) {
mQOUTE(true);
```

```
theRetToken=_returnToken;
}
else {
if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken = makeToken(Token.EOF_TYPE);}
else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(),
 getColumn());}
}
}
if ( _returnToken==null ) continue tryAgain; // found SKIP token
_ttype = _returnToken.getType();
_returnToken.setType(_ttype);
return _returnToken;
}
catch (RecognitionException e) {
throw new TokenStreamRecognitionException(e);
}
}
catch (CharStreamException cse) {
if ( cse instanceof CharStreamIOException ) {
throw new TokenStreamIOException(((CharStreamIOException)cse).io);
}
else {
throw new TokenStreamException(cse.getMessage());
}
}
}
}

protected final void mALPHA(boolean _createToken) throws RecognitionException,
                    CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ALPHA;
int _saveIndex;

switch ( LA(1)) {
case 'a':  case 'b':  case 'c':  case 'd':
case 'e':  case 'f':  case 'g':  case 'h':
case 'i':  case 'j':  case 'k':  case 'l':
case 'm':  case 'n':  case 'o':  case 'p':
case 'q':  case 'r':  case 's':  case 't':
case 'u':  case 'v':  case 'w':  case 'x':
case 'y':  case 'z':
{
matchRange('a','z');
break;
}
```

```
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':
{
matchRange('A','Z');
break;
}
case '_':
{
match('_');
break;
}
default:
{
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());
}
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mDIGIT(boolean _createToken) throws RecognitionException,
                    CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = DIGIT;
int _saveIndex;

matchRange('0','9');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mWS(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = WS;
```

```
int _saveIndex;

{
int _cnt5=0;
_loop5:
do {
switch ( LA(1)) {
case ' ':
{
match(' ');
break;
}
case '\t':
{
match('\t');
break;
}
default:
{
if ( _cnt5>=1 ) { break _loop5; } else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
}
}
_cnt5++;
} while (true);
}
if ( inputState.guessing==0 ) {
_ttype = Token.SKIP;
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mNL(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = NL;
int _saveIndex;

{
boolean synPredMatched9 = false;
if (((LA(1)=='\r') && (LA(2)=='\n'))) {
int _m9 = mark();
```

```
synPredMatched9 = true;
inputState.guessing++;
try {
{
match('\r');
match('\n');
}
}
catch (RecognitionException pe) {
synPredMatched9 = false;
}
rewind(_m9);
inputState.guessing--;
}
if ( synPredMatched9 ) {
match('\r');
match('\n');
}
else if ((LA(1)=='\n')) {
match('\n');
}
else if ((LA(1)=='\r') && (true)) {
match('\r');
}
else {
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());
}

}
if ( inputState.guessing==0 ) {
_ttype = Token.SKIP; newline();
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mCOMMENT(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = COMMENT;
int _saveIndex;

{
```

```
if ((LA(1)=='/') && (LA(2)=='*')) {
match("/*");
{
_loop15:
do {
// nongreedy exit test
if ((LA(1)=='*') && (LA(2)=='/')) break _loop15;
if ((_tokenSet_1.member(LA(1))) && ((LA(2) >= '\u0003' && LA(2) <= '\u00ff'))) {
{
match(_tokenSet_1);
}
}
else if ((LA(1)=='\n'||LA(1)=='\r')) {
{
mNL(false);
}
}
else {
break _loop15;
}

} while (true);
}
match("*/");
}
else if ((LA(1)=='/') && (LA(2)=='/')) {
match("//");
{
_loop18:
do {
if ((_tokenSet_1.member(LA(1)))) {
{
match(_tokenSet_1);
}
}
else {
break _loop18;
}

} while (true);
}
{
mNL(false);
}
}
else {
```

```
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());
}

}
if ( inputState.guessing==0 ) {
_ttype = Token.SKIP;
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLPAREN(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LPAREN;
int _saveIndex;

match('(');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mRPAREN(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = RPAREN;
int _saveIndex;

match(')');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLBRACE(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LBRACE;
```

```
int _saveIndex;

match('{');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mRBRACE(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = RBRACE;
int _saveIndex;

match('}');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mPLUS(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = PLUS;
int _saveIndex;

match('+');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mMINUS(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = MINUS;
int _saveIndex;

match('-');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
```

```
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mMULT(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = MULT;
int _saveIndex;

match('*');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mDIV(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = DIV;
int _saveIndex;

match('/');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSEMI(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = SEMI;
int _saveIndex;

match(';');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
```

```
}

public final void mCOMMA(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = COMMA;
int _saveIndex;

match(',');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mASSIGN(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ASSIGN;
int _saveIndex;

match('=');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mARROW(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ARROW;
int _saveIndex;

match("->");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mEQ(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
```

```
int _ttype; Token _token=null; int _begin=text.length();
_ttype = EQ;
int _saveIndex;

match("==");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mNEQ(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = NEQ;
int _saveIndex;

match("!=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mPLUSEQ(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = PLUSEQ;
int _saveIndex;

match("+=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mMINUSEQ(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = MINUSEQ;
int _saveIndex;
```

```
match("-=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mMULTEQ(boolean _createToken) throws RecognitionException,
                  CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = MULTEQ;
int _saveIndex;

match("*=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mDIVEQ(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = DIVEQ;
int _saveIndex;

match("/=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mGT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = GT;
int _saveIndex;

match('>');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
```

```
}
_returnToken = _token;
}

public final void mLT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LT;
int _saveIndex;

match('<');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mGE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = GE;
int _saveIndex;

match(">=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mLE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = LE;
int _saveIndex;

match("<=");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}
```

```
public final void mAND(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = AND;
int _saveIndex;

match("&&");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mOR(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = OR;
int _saveIndex;

match("||");
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mNOT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = NOT;
int _saveIndex;

match('!');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mQOUTE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = QOUTE;
```

```
int _saveIndex;

match('"');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mID(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ID;
int _saveIndex;

mALPHA(false);
{
_loop48:
do {
switch ( LA(1)) {
case 'A':  case 'B':  case 'C':  case 'D':
case 'E':  case 'F':  case 'G':  case 'H':
case 'I':  case 'J':  case 'K':  case 'L':
case 'M':  case 'N':  case 'O':  case 'P':
case 'Q':  case 'R':  case 'S':  case 'T':
case 'U':  case 'V':  case 'W':  case 'X':
case 'Y':  case 'Z':  case '_':  case 'a':
case 'b':  case 'c':  case 'd':  case 'e':
case 'f':  case 'g':  case 'h':  case 'i':
case 'j':  case 'k':  case 'l':  case 'm':
case 'n':  case 'o':  case 'p':  case 'q':
case 'r':  case 's':  case 't':  case 'u':
case 'v':  case 'w':  case 'x':  case 'y':
case 'z':
{
mALPHA(false);
break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
mDIGIT(false);
break;
}
```

```
default:
{
break _loop48;
}
}
} while (true);
}
_ttype = testLiteralsTable(_ttype);
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mNUMBER(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = NUMBER;
int _saveIndex;

{
int _cnt51=0;
_loop51:
do {
if (((LA(1) >= '0' && LA(1) <= '9'))) {
mDIGIT(false);
}
else {
if ( _cnt51>=1 ) { break _loop51; } else {throw new
NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());}
}

_cnt51++;
} while (true);
}
{
if ((LA(1)=='.')) {
match('.');
{
_loop54:
do {
if (((LA(1) >= '0' && LA(1) <= '9'))) {
mDIGIT(false);
}
else {
```

```
break _loop54;
}

} while (true);
}
}
else {
}

}
{
if ((LA(1)=='E'||LA(1)=='e')) {
{
switch ( LA(1)) {
case 'E':
{
match('E');
break;
}
case 'e':
{
match('e');
break;
}
default:
{
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());
}
}
}
{
switch ( LA(1)) {
case '+':
{
match('+');
break;
}
case '-':
{
match('-');
break;
}
case '0':  case '1':  case '2':  case '3':
case '4':  case '5':  case '6':  case '7':
case '8':  case '9':
{
```

```
break;
}
default:
{
throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine(), getColumn());
}
}
}
{
int _cnt59=0;
_loop59:
do {
if (((LA(1) >= '0' && LA(1) <= '9'))) {
mDIGIT(false);
}
else {
if ( _cnt59>=1 ) { break _loop59; }
else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine(), getColumn());}
}

_cnt59++;
} while (true);
}
}
else {
}

}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSTRING(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = STRING;
int _saveIndex;

_saveIndex=text.length();
match('"');
text.setLength(_saveIndex);
{
```

```
_loop64:
do {
if ((LA(1)=='"') && (LA(2)=='")) {
{
_saveIndex=text.length();
match('"');
text.setLength(_saveIndex);
match('"');
}
}
else if ((_tokenSet_2.member(LA(1)))) {
{
match(_tokenSet_2);
}
}
else {
break _loop64;
}

} while (true);
}
_saveIndex=text.length();
match('"');
text.setLength(_saveIndex);
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}


private static final long[] mk_tokenSet_0() {
long[] data = new long[8];
data[0]=-1032L;
for (int i = 1; i<=3; i++) { data[i]=-1L; }
return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
long[] data = new long[8];
data[0]=-9224L;
for (int i = 1; i<=3; i++) { data[i]=-1L; }
return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
```

```
private static final long[] mk_tokenSet_2() {
long[] data = new long[8];
data[0]=-17179870216L;
for (int i = 1; i<=3; i++) { data[i]=-1L; }
return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());

}
```

8.3 patternParser.java

```
// $ANTLR 2.7.5 (20050128): "patternGrammar.g" -> "patternParser.java"$

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import java.util.Hashtable;
import antlr.ASTFactory;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

public class patternParser extends antlr.LLkParser       implements patternLexerTokenTypes
 {



protected patternParser(TokenBuffer tokenBuf, int k) {
  super(tokenBuf,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}
```

```
public patternParser(TokenBuffer tokenBuf) {
  this(tokenBuf,2);
}

protected patternParser(TokenStream lexer, int k) {
  super(lexer,k);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public patternParser(TokenStream lexer) {
  this(lexer,2);
}

public patternParser(ParserSharedInputState state) {
  super(state,2);
  tokenNames = _tokenNames;
  buildTokenTypeASTClassMap();
  astFactory = new ASTFactory(getTokenTypeToASTClassMap());
}

public final void program() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST program_AST = null;

try {      // for error handling
{
_loop67:
do {
switch ( LA(1)) {
case LPAREN:
case LBRACE:
case ID:
case LITERAL_set_FG:
case LITERAL_set_BG:
case LITERAL_move:
case LITERAL_scale:
case LITERAL_rotate:
case LITERAL_for:
case LITERAL_while:
case LITERAL_if:
case LITERAL_include:
case LITERAL_break:
```

```
case LITERAL_print:
{
outer_statement();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case LITERAL_pattern:
{
pattern_def();
astFactory.addASTChild(currentAST, returnAST);
break;
}
default:
{
break _loop67;
}
}
} while (true);
}
match(Token.EOF_TYPE);
program_AST = (AST)currentAST.root;
program_AST = (AST)astFactory.make( (new
              ASTArray(2)).add(astFactory.create(STATEMENT,"PROG")).add(program_AST));
currentAST.root = program_AST;
currentAST.child = program_AST!=null &&program_AST.getFirstChild()!=null ?
program_AST.getFirstChild() : program_AST;
currentAST.advanceChildToEnd();
program_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_0);
}
returnAST = program_AST;
}

public final void outer_statement() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST outer_statement_AST = null;

try {      // for error handling
switch ( LA(1)) {
case LITERAL_set_BG:
{
```

```
bckgrnd_color_stmt();
astFactory.addASTChild(currentAST, returnAST);
outer_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_include:
{
include_stmt();
astFactory.addASTChild(currentAST, returnAST);
outer_statement_AST = (AST)currentAST.root;
break;
}
case LPAREN:
case LBRACE:
case ID:
case LITERAL_set_FG:
case LITERAL_move:
case LITERAL_scale:
case LITERAL_rotate:
case LITERAL_for:
case LITERAL_while:
case LITERAL_if:
case LITERAL_break:
case LITERAL_print:
{
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
outer_statement_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = outer_statement_AST;
}

public final void pattern_def() throws RecognitionException, TokenStreamException {

returnAST = null;
```

```
ASTPair currentAST = new ASTPair();
AST pattern_def_AST = null;

try {      // for error handling
AST tmp2_AST = null;
tmp2_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp2_AST);
match(LITERAL_pattern);
AST tmp3_AST = null;
tmp3_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp3_AST);
match(ID);
match(LPAREN);
pattern_args();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
match(LBRACE);
pattern_body();
astFactory.addASTChild(currentAST, returnAST);
match(RBRACE);
System.out.println("Found a pattern");
pattern_def_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = pattern_def_AST;
}

public final void bckgrnd_color_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST bckgrnd_color_stmt_AST = null;

try {      // for error handling
AST tmp8_AST = null;
tmp8_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp8_AST);
match(LITERAL_set_BG);
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(COMMA);
expression();
```

```
astFactory.addASTChild(currentAST, returnAST);
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
match(SEMI);
System.out.println("Found set_background");
bckgrnd_color_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = bckgrnd_color_stmt_AST;
}

public final void include_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST include_stmt_AST = null;

try {      // for error handling
AST tmp14_AST = null;
tmp14_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp14_AST);
match(LITERAL_include);
AST tmp15_AST = null;
tmp15_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp15_AST);
match(STRING);
match(SEMI);
System.out.println("Found an include");
include_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_1);
}
returnAST = include_stmt_AST;
}

public final void inner_statement() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
```

```
AST inner_statement_AST = null;

try {      // for error handling
switch ( LA(1)) {
case LITERAL_set_FG:
{
foregrnd_color_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_move:
{
move_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_scale:
{
scale_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_rotate:
{
rotate_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_for:
{
for_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_if:
{
if_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
```

```
case LITERAL_while:
{
while_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_break:
{
break_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LPAREN:
{
draw_line_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LITERAL_print:
{
print_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
break;
}
case LBRACE:
{
match(LBRACE);
{
_loop71:
do {
if ((_tokenSet_2.member(LA(1)))) {
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop71;
}

} while (true);
}
match(RBRACE);
inner_statement_AST = (AST)currentAST.root;
```

```
inner_statement_AST = (AST)astFactory.make( (new
                          ASTArray(2)).add(astFactory.create(STATEMENT)).add(inner_statement_
currentAST.root = inner_statement_AST;
currentAST.child = inner_statement_AST!=null &&inner_statement_AST.getFirstChild()!=null ?
inner_statement_AST.getFirstChild() : inner_statement_AST;
currentAST.advanceChildToEnd();
inner_statement_AST = (AST)currentAST.root;
break;
}
default:
if ((LA(1)==ID) && (LA(2)==ASSIGN)) {
assign_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
}
else if ((LA(1)==ID) && (LA(2)==LPAREN)) {
call_stmt();
astFactory.addASTChild(currentAST, returnAST);
inner_statement_AST = (AST)currentAST.root;
}
else {
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = inner_statement_AST;
}

public final void foregrnd_color_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST foregrnd_color_stmt_AST = null;

try {      // for error handling
AST tmp19_AST = null;
tmp19_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp19_AST);
match(LITERAL_set_FG);
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
```

```
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
match(SEMI);
System.out.println("Found set_foreground");
foregrnd_color_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = foregrnd_color_stmt_AST;
}

public final void move_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST move_stmt_AST = null;

try {      // for error handling
AST tmp25_AST = null;
tmp25_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp25_AST);
match(LITERAL_move);
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
match(SEMI);
System.out.println("Found a move");
move_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = move_stmt_AST;
}
```

```
public final void scale_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST scale_stmt_AST = null;

try {      // for error handling
AST tmp30_AST = null;
tmp30_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp30_AST);
match(LITERAL_scale);
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
match(SEMI);
System.out.println("Found a scale");
scale_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = scale_stmt_AST;
}

public final void rotate_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST rotate_stmt_AST = null;

try {      // for error handling
AST tmp34_AST = null;
tmp34_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp34_AST);
match(LITERAL_rotate);
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
match(SEMI);
System.out.println("Found a rotate");
rotate_stmt_AST = (AST)currentAST.root;
}
```

```
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = rotate_stmt_AST;
}

public final void for_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST for_stmt_AST = null;

try {      // for error handling
AST tmp38_AST = null;
tmp38_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp38_AST);
match(LITERAL_for);
match(LPAREN);
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(SEMI);
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
System.out.println("Found a for statement");
for_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = for_stmt_AST;
}

public final void if_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST if_stmt_AST = null;

try {      // for error handling
```

```
AST tmp42_AST = null;
tmp42_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp42_AST);
match(LITERAL_if);
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
{
if ((LA(1)==LITERAL_else) && (_tokenSet_2.member(LA(2)))) {
match(LITERAL_else);
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
}
else if ((_tokenSet_3.member(LA(1))) && (_tokenSet_4.member(LA(2)))) {
}
else {
throw new NoViableAltException(LT(1), getFilename());
}

}
System.out.println("Found an if");
if_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = if_stmt_AST;
}

public final void while_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST while_stmt_AST = null;

try {      // for error handling
AST tmp46_AST = null;
tmp46_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp46_AST);
match(LITERAL_while);
match(LPAREN);
expression();
```

```
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
System.out.println("Found a while");
while_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = while_stmt_AST;
}

public final void break_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST break_stmt_AST = null;

try {      // for error handling
AST tmp49_AST = null;
tmp49_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp49_AST);
match(LITERAL_break);
match(SEMI);
System.out.println("Found a break");
break_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = break_stmt_AST;
}

public final void assign_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST assign_stmt_AST = null;

try {      // for error handling
AST tmp51_AST = null;
tmp51_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp51_AST);
```

```
match(ID);
AST tmp52_AST = null;
tmp52_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp52_AST);
match(ASSIGN);
{
expression();
astFactory.addASTChild(currentAST, returnAST);
}
match(SEMI);
System.out.println("Found an assignment");
assign_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = assign_stmt_AST;
}

public final void draw_line_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST draw_line_stmt_AST = null;

try {      // for error handling
point();
astFactory.addASTChild(currentAST, returnAST);
{
int _cnt92=0;
_loop92:
do {
if ((LA(1)==ARROW)) {
AST tmp54_AST = null;
tmp54_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp54_AST);
match(ARROW);
point();
astFactory.addASTChild(currentAST, returnAST);
}
else {
if ( _cnt92>=1 ) { break _loop92; }
else {throw new NoViableAltException(LT(1), getFilename());}
}
```

```
_cnt92++;
} while (true);
}
match(SEMI);
System.out.println("Found a draw line sequence.");
draw_line_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = draw_line_stmt_AST;
}

public final void call_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST call_stmt_AST = null;

try {      // for error handling
pattern_call();
astFactory.addASTChild(currentAST, returnAST);
match(SEMI);
call_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = call_stmt_AST;
}

public final void print_stmt() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST print_stmt_AST = null;

try {      // for error handling
AST tmp57_AST = null;
tmp57_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp57_AST);
match(LITERAL_print);
match(LPAREN);
expression();
```

```
astFactory.addASTChild(currentAST, returnAST);
{
_loop102:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop102;
}

} while (true);
}
match(RPAREN);
match(SEMI);
System.out.println("Found a print");
print_stmt_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_3);
}
returnAST = print_stmt_AST;
}

public final void pattern_args() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST pattern_args_AST = null;

try {      // for error handling
switch ( LA(1)) {
case ID:
{
AST tmp62_AST = null;
tmp62_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp62_AST);
match(ID);
{
_loop75:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
```

```
AST tmp64_AST = null;
tmp64_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp64_AST);
match(ID);
}
else {
break _loop75;
}

} while (true);
}
pattern_args_AST = (AST)currentAST.root;
pattern_args_AST = (AST)astFactory.make( (new
             ASTArray(2)).add(astFactory.create(ARGS,"ARGS")).add(pattern_args_AST));
currentAST.root = pattern_args_AST;
currentAST.child = pattern_args_AST!=null &&pattern_args_AST.getFirstChild()!=null ?
pattern_args_AST.getFirstChild() : pattern_args_AST;
currentAST.advanceChildToEnd();
pattern_args_AST = (AST)currentAST.root;
break;
}
case RPAREN:
{
pattern_args_AST = (AST)currentAST.root;
pattern_args_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(ARGS,"ARGS")).add(pattern_args_AST));
currentAST.root = pattern_args_AST;
currentAST.child = pattern_args_AST!=null &&pattern_args_AST.getFirstChild()!=null ?
pattern_args_AST.getFirstChild() : pattern_args_AST;
currentAST.advanceChildToEnd();
pattern_args_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_5);
}
returnAST = pattern_args_AST;
}
```

```
public final void pattern_body() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST pattern_body_AST = null;

try {      // for error handling
{
int _cnt78=0;
_loop78:
do {
if ((_tokenSet_2.member(LA(1)))) {
inner_statement();
astFactory.addASTChild(currentAST, returnAST);
}
else {
if ( _cnt78>=1 ) { break _loop78; }
else {throw new NoViableAltException(LT(1), getFilename());}
}

_cnt78++;
} while (true);
}
pattern_body_AST = (AST)currentAST.root;
pattern_body_AST = (AST)astFactory.make( (new
           ASTArray(2)).add(astFactory.create(STATEMENT)).add(pattern_body_AST));
currentAST.root = pattern_body_AST;
currentAST.child = pattern_body_AST!=null &&pattern_body_AST.getFirstChild()!=null ?
pattern_body_AST.getFirstChild() : pattern_body_AST;
currentAST.advanceChildToEnd();
pattern_body_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_6);
}
returnAST = pattern_body_AST;
}

public final void expression() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST expression_AST = null;

try {      // for error handling
```

```
logic_term();
astFactory.addASTChild(currentAST, returnAST);
{
_loop105:
do {
if ((LA(1)==LITERAL_or)) {
AST tmp65_AST = null;
tmp65_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp65_AST);
match(LITERAL_or);
logic_term();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop105;
}

} while (true);
}
expression_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_7);
}
returnAST = expression_AST;
}

public final void point() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST point_AST = null;

try {      // for error handling
match(LPAREN);
{
expression();
astFactory.addASTChild(currentAST, returnAST);
}
match(COMMA);
{
expression();
astFactory.addASTChild(currentAST, returnAST);
}
match(RPAREN);
```

```
point_AST = (AST)currentAST.root;
point_AST = (AST)astFactory.make( (new
             ASTArray(2)).add(astFactory.create(POINT)).add(point_AST));
currentAST.root = point_AST;
currentAST.child = point_AST!=null &&point_AST.getFirstChild()!=null ?
point_AST.getFirstChild() : point_AST;
currentAST.advanceChildToEnd();
point_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_8);
}
returnAST = point_AST;
}

public final void pattern_call() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST pattern_call_AST = null;

try {      // for error handling
AST tmp69_AST = null;
tmp69_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp69_AST);
match(ID);
match(LPAREN);
call_args();
astFactory.addASTChild(currentAST, returnAST);
match(RPAREN);
pattern_call_AST = (AST)currentAST.root;
pattern_call_AST = (AST)astFactory.make( (new
                   ASTArray(2)).add(astFactory.create(FUNC_CALL)).add(pattern_call_AST));
currentAST.root = pattern_call_AST;
currentAST.child = pattern_call_AST!=null &&pattern_call_AST.getFirstChild()!=null ?
pattern_call_AST.getFirstChild() : pattern_call_AST;
currentAST.advanceChildToEnd();
pattern_call_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_9);
}
returnAST = pattern_call_AST;
}
```

```
public final void call_args() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST call_args_AST = null;

try {      // for error handling
switch ( LA(1)) {
case LPAREN:
case MINUS:
case ID:
case NUMBER:
case STRING:
case LITERAL_not:
{
expression();
astFactory.addASTChild(currentAST, returnAST);
{
_loop97:
do {
if ((LA(1)==COMMA)) {
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop97;
}

} while (true);
}
call_args_AST = (AST)currentAST.root;
call_args_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(ARGS,"ARGS")).add(call_args_AST));
currentAST.root = call_args_AST;
currentAST.child = call_args_AST!=null &&call_args_AST.getFirstChild()!=null ?
call_args_AST.getFirstChild() : call_args_AST;
currentAST.advanceChildToEnd();
call_args_AST = (AST)currentAST.root;
break;
}
case RPAREN:
{
call_args_AST = (AST)currentAST.root;
call_args_AST = (AST)astFactory.make( (new
```

```
ASTArray(2)).add(astFactory.create(ARGS,"ARGS")).add(call_args_AST));
currentAST.root = call_args_AST;
currentAST.child = call_args_AST!=null &&call_args_AST.getFirstChild()!=null ?
call_args_AST.getFirstChild() : call_args_AST;
currentAST.advanceChildToEnd();
call_args_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_5);
}
returnAST = call_args_AST;
}

public final void logic_term() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST logic_term_AST = null;

try {      // for error handling
logic_factor();
astFactory.addASTChild(currentAST, returnAST);
{
_loop108:
do {
if ((LA(1)==LITERAL_and)) {
AST tmp73_AST = null;
tmp73_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp73_AST);
match(LITERAL_and);
logic_factor();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop108;
}

} while (true);
```

```
}
logic_term_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_10);
}
returnAST = logic_term_AST;
}

public final void logic_factor() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST logic_factor_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case LITERAL_not:
{
AST tmp74_AST = null;
tmp74_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp74_AST);
match(LITERAL_not);
break;
}
case LPAREN:
case MINUS:
case ID:
case NUMBER:
case STRING:
{
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
comparison_expr();
astFactory.addASTChild(currentAST, returnAST);
logic_factor_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
```

```
reportError(ex);
recover(ex,_tokenSet_11);
}
returnAST = logic_factor_AST;
}

public final void comparison_expr() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST comparison_expr_AST = null;

try {      // for error handling
arith_expr();
astFactory.addASTChild(currentAST, returnAST);
{
switch ( LA(1)) {
case EQ:
case NEQ:
case GT:
case LT:
case GE:
case LE:
{
{
switch ( LA(1)) {
case GE:
{
AST tmp75_AST = null;
tmp75_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp75_AST);
match(GE);
break;
}
case LE:
{
AST tmp76_AST = null;
tmp76_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp76_AST);
match(LE);
break;
}
case GT:
{
AST tmp77_AST = null;
tmp77_AST = astFactory.create(LT(1));
```

```
astFactory.makeASTRoot(currentAST, tmp77_AST);
match(GT);
break;
}
case LT:
{
AST tmp78_AST = null;
tmp78_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp78_AST);
match(LT);
break;
}
case EQ:
{
AST tmp79_AST = null;
tmp79_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp79_AST);
match(EQ);
break;
}
case NEQ:
{
AST tmp80_AST = null;
tmp80_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp80_AST);
match(NEQ);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
arith_expr();
astFactory.addASTChild(currentAST, returnAST);
break;
}
case RPAREN:
case SEMI:
case COMMA:
case LITERAL_or:
case LITERAL_and:
{
break;
}
```

```
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
comparison_expr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_11);
}
returnAST = comparison_expr_AST;
}

public final void arith_expr() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST arith_expr_AST = null;

try {      // for error handling
arith_term();
astFactory.addASTChild(currentAST, returnAST);
{
_loop117:
do {
if ((LA(1)==PLUS||LA(1)==MINUS)) {
{
switch ( LA(1)) {
case PLUS:
{
AST tmp81_AST = null;
tmp81_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp81_AST);
match(PLUS);
break;
}
case MINUS:
{
AST tmp82_AST = null;
tmp82_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp82_AST);
match(MINUS);
break;
}
```

```
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
arith_term();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop117;
}

} while (true);
}
arith_expr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_12);
}
returnAST = arith_expr_AST;
}

public final void arith_term() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST arith_term_AST = null;

try {      // for error handling
unary_minus();
astFactory.addASTChild(currentAST, returnAST);
{
_loop121:
do {
if ((LA(1)==MULT||LA(1)==DIV)) {
{
switch ( LA(1)) {
case MULT:
{
AST tmp83_AST = null;
tmp83_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp83_AST);
match(MULT);
break;
```

```
}
case DIV:
{
AST tmp84_AST = null;
tmp84_AST = astFactory.create(LT(1));
astFactory.makeASTRoot(currentAST, tmp84_AST);
match(DIV);
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
unary_minus();
astFactory.addASTChild(currentAST, returnAST);
}
else {
break _loop121;
}

} while (true);
}
arith_term_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_13);
}
returnAST = arith_term_AST;
}

public final void unary_minus() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST unary_minus_AST = null;

try {      // for error handling
switch ( LA(1)) {
case MINUS:
{
match(MINUS);
arith_factor();
astFactory.addASTChild(currentAST, returnAST);
```

```
unary_minus_AST = (AST)currentAST.root;
unary_minus_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(UNARY_MINUS)).add(unary_minus_AST));
currentAST.root = unary_minus_AST;
currentAST.child = unary_minus_AST!=null &&unary_minus_AST.getFirstChild()!=null ?
unary_minus_AST.getFirstChild() : unary_minus_AST;
currentAST.advanceChildToEnd();
unary_minus_AST = (AST)currentAST.root;
break;
}
case LPAREN:
case ID:
case NUMBER:
case STRING:
{
arith_factor();
astFactory.addASTChild(currentAST, returnAST);
unary_minus_AST = (AST)currentAST.root;
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_9);
}
returnAST = unary_minus_AST;
}

public final void arith_factor() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
AST arith_factor_AST = null;

try {      // for error handling
{
switch ( LA(1)) {
case NUMBER:
{
AST tmp86_AST = null;
tmp86_AST = astFactory.create(LT(1));
```

```
astFactory.addASTChild(currentAST, tmp86_AST);
match(NUMBER);
break;
}
case STRING:
{
AST tmp87_AST = null;
tmp87_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp87_AST);
match(STRING);
break;
}
case LPAREN:
{
paren_expr();
astFactory.addASTChild(currentAST, returnAST);
break;
}
default:
if ((LA(1)==ID) && (_tokenSet_9.member(LA(2)))) {
AST tmp88_AST = null;
tmp88_AST = astFactory.create(LT(1));
astFactory.addASTChild(currentAST, tmp88_AST);
match(ID);
}
else if ((LA(1)==ID) && (LA(2)==LPAREN)) {
pattern_call();
astFactory.addASTChild(currentAST, returnAST);
}
else {
throw new NoViableAltException(LT(1), getFilename());
}
}
}
arith_factor_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_9);
}
returnAST = arith_factor_AST;
}

public final void paren_expr() throws RecognitionException, TokenStreamException {

returnAST = null;
```

```
ASTPair currentAST = new ASTPair();
AST paren_expr_AST = null;

try {      // for error handling
match(LPAREN);
expression();
astFactory.addASTChild(currentAST, returnAST);
{
switch ( LA(1)) {
case COMMA:
{
match(COMMA);
expression();
astFactory.addASTChild(currentAST, returnAST);
paren_expr_AST = (AST)currentAST.root;
paren_expr_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(POINT)).add(paren_expr_AST));
currentAST.root = paren_expr_AST;
currentAST.child = paren_expr_AST!=null &&paren_expr_AST.getFirstChild()!=null ?
paren_expr_AST.getFirstChild() : paren_expr_AST;
currentAST.advanceChildToEnd();
break;
}
case RPAREN:
{
paren_expr_AST = (AST)currentAST.root;
paren_expr_AST = (AST)astFactory.make( (new
ASTArray(2)).add(astFactory.create(PAREN_EXPR)).add(paren_expr_AST));
currentAST.root = paren_expr_AST;
currentAST.child = paren_expr_AST!=null &&paren_expr_AST.getFirstChild()!=null ?
paren_expr_AST.getFirstChild() : paren_expr_AST;
currentAST.advanceChildToEnd();
break;
}
default:
{
throw new NoViableAltException(LT(1), getFilename());
}
}
}
match(RPAREN);
paren_expr_AST = (AST)currentAST.root;
}
catch (RecognitionException ex) {
reportError(ex);
recover(ex,_tokenSet_9);
```

```
}
returnAST = paren_expr_AST;
}


public static final String[] _tokenNames = {
"<0>",
"EOF",
"<2>",
"NULL_TREE_LOOKAHEAD",
"ALPHA",
"DIGIT",
"WS",
"NL",
"COMMENT",
"LPAREN",
"RPAREN",
"LBRACE",
"RBRACE",
"PLUS",
"MINUS",
"MULT",
"DIV",
"SEMI",
"COMMA",
"ASSIGN",
"ARROW",
"EQ",
"NEQ",
"PLUSEQ",
"MINUSEQ",
"MULTEQ",
"DIVEQ",
"GT",
"LT",
"GE",
"LE",
"AND",
"OR",
"NOT",
"QOUTE",
"ID",
"NUMBER",
"STRING",
"STATEMENT",
"POINT",
```

```
"FUNC_CALL",
"ARGS",
"UNARY_MINUS",
"PAREN_EXPR",
"\"pattern\"",
"\"set_FG\"",
"\"set_BG\"",
"\"move\"",
"\"scale\"",
"\"rotate\"",
"\"for\"",
"\"while\"",
"\"if\"",
"\"else\"",
"\"include\"",
"\"break\"",
"\"print\"",
"\"or\"",
"\"and\"",
"\"not\""
};

protected void buildTokenTypeASTClassMap() {
tokenTypeToASTClassMap=null;
};

private static final long[] mk_tokenSet_0() {
long[] data = { 2L, 0L};
return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
private static final long[] mk_tokenSet_1() {
long[] data = { 135090430994811394L, 0L};
return data;
}
public static final BitSet _tokenSet_1 = new BitSet(mk_tokenSet_1());
private static final long[] mk_tokenSet_2() {
long[] data = { 116988071555107328L, 0L};
return data;
}
public static final BitSet _tokenSet_2 = new BitSet(mk_tokenSet_2());
private static final long[] mk_tokenSet_3() {
long[] data = { 720558588711427586L, 0L};
return data;
}
public static final BitSet _tokenSet_3 = new BitSet(mk_tokenSet_3());
```

```
private static final long[] mk_tokenSet_4() {
long[] data = { 1152904154959314434L, 0L};
return data;
}
public static final BitSet _tokenSet_4 = new BitSet(mk_tokenSet_4());
private static final long[] mk_tokenSet_5() {
long[] data = { 1024L, 0L};
return data;
}
public static final BitSet _tokenSet_5 = new BitSet(mk_tokenSet_5());
private static final long[] mk_tokenSet_6() {
long[] data = { 4096L, 0L};
return data;
}
public static final BitSet _tokenSet_6 = new BitSet(mk_tokenSet_6());
private static final long[] mk_tokenSet_7() {
long[] data = { 394240L, 0L};
return data;
}
public static final BitSet _tokenSet_7 = new BitSet(mk_tokenSet_7());
private static final long[] mk_tokenSet_8() {
long[] data = { 1179648L, 0L};
return data;
}
public static final BitSet _tokenSet_8 = new BitSet(mk_tokenSet_8());
private static final long[] mk_tokenSet_9() {
long[] data = { 432345566247642112L, 0L};
return data;
}
public static final BitSet _tokenSet_9 = new BitSet(mk_tokenSet_9());
private static final long[] mk_tokenSet_10() {
long[] data = { 144115188076250112L, 0L};
return data;
}
public static final BitSet _tokenSet_10 = new BitSet(mk_tokenSet_10());
private static final long[] mk_tokenSet_11() {
long[] data = { 432345564227961856L, 0L};
return data;
}
public static final BitSet _tokenSet_11 = new BitSet(mk_tokenSet_11());
private static final long[] mk_tokenSet_12() {
long[] data = { 432345566247519232L, 0L};
return data;
}
public static final BitSet _tokenSet_12 = new BitSet(mk_tokenSet_12());
private static final long[] mk_tokenSet_13() {
```

```
long[] data = { 432345566247543808L, 0L};
return data;
}
public static final BitSet _tokenSet_13 = new BitSet(mk_tokenSet_13());

}
```

8.4 pWalker.java

```
// $ANTLR 2.7.5 (20050128): "patternGrammar.g" -> "pWalker.java"$

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

import java.util.ArrayList;


public class pWalker extends antlr.TreeParser       implements patternLexerTokenTypes
 {

    Environment e = new Environment();
    public void display(){ e.display();}
public pWalker() {
tokenNames = _tokenNames;
}

public final Variable  expr(AST _t) throws RecognitionException {
Variable r;

AST expr_AST_in = (_t == ASTNULL) ? null : (AST)_t;
AST stmt = null;
AST func = null;
AST fname = null;
AST body = null;
AST fbefore = null;
AST fexpr = null;
```

```
AST fafter = null;
AST fbody = null;
AST wexpr = null;
AST wbody = null;
AST ithen = null;
AST ielse = null;
AST rhs_or = null;
AST rhs_and = null;
AST name = null;

r = null;
Variable a, b, c;
Variable[] args;
String[] def_args;


try {      // for error handling
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case STATEMENT:
{
AST __t131 = _t;
AST tmp92_AST_in = (AST)_t;
match(_t,STATEMENT);
_t = _t.getFirstChild();
{
_loop133:
do {
if (_t==null) _t=ASTNULL;
if (((_t.getType() >= ALPHA && _t.getType() <= LITERAL_not))) {
stmt = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
r=expr(stmt);
}
else {
break _loop133;
}

} while (true);
}
_t = __t131;
_t = _t.getNextSibling();
break;
}
case FUNC_CALL:
```

```
{
AST __t134 = _t;
AST tmp93_AST_in = (AST)_t;
match(_t,FUNC_CALL);
_t = _t.getFirstChild();
func = (AST)_t;
match(_t,ID);
_t = _t.getNextSibling();
args=arg_list(_t);
_t = _retTree;
_t = __t134;
_t = _t.getNextSibling();
r = e.callPattern(this, func.getText(), args);
break;
}
case LITERAL_set_BG:
{
AST __t135 = _t;
AST tmp94_AST_in = (AST)_t;
match(_t,LITERAL_set_BG);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
c=expr(_t);
_t = _retTree;
_t = __t135;
_t = _t.getNextSibling();
e.setBackground(a, b, c);
break;
}
case LITERAL_set_FG:
{
AST __t136 = _t;
AST tmp95_AST_in = (AST)_t;
match(_t,LITERAL_set_FG);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
c=expr(_t);
_t = _retTree;
_t = __t136;
_t = _t.getNextSibling();
```

```
e.setForeground(a, b, c);
break;
}
case LITERAL_include:
{
AST __t137 = _t;
AST tmp96_AST_in = (AST)_t;
match(_t,LITERAL_include);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
_t = __t137;
_t = _t.getNextSibling();
e.include(a);
break;
}
case LITERAL_pattern:
{
AST __t138 = _t;
AST tmp97_AST_in = (AST)_t;
match(_t,LITERAL_pattern);
_t = _t.getFirstChild();
fname = (AST)_t;
match(_t,ID);
_t = _t.getNextSibling();
def_args=def_arg_list(_t);
_t = _retTree;
body = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
_t = __t138;
_t = _t.getNextSibling();
e.definePattern(fname.getText(), def_args, body);
break;
}
case LITERAL_print:
{
AST __t139 = _t;
AST tmp98_AST_in = (AST)_t;
match(_t,LITERAL_print);
_t = _t.getFirstChild();
{
int _cnt141=0;
_loop141:
do {
if (_t==null) _t=ASTNULL;
```

```
if ((_tokenSet_0.member(_t.getType()))) {
a=expr(_t);
_t = _retTree;
System.out.print(a.toString() + " ");
}
else {
if ( _cnt141>=1 ) { break _loop141; } else {throw new NoViableAltException(_t);}
}

_cnt141++;
} while (true);
}
_t = __t139;
_t = _t.getNextSibling();
System.out.println();
break;
}
case LITERAL_for:
{
AST __t142 = _t;
AST tmp99_AST_in = (AST)_t;
match(_t,LITERAL_for);
_t = _t.getFirstChild();
fbefore = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
fexpr = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
fafter = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
fbody = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
_t = __t142;
_t = _t.getNextSibling();

expr(fbefore);
while (expr(fexpr).isTrue()) {
expr(fbody);
expr(fafter);
}

break;
}
```

```
case LITERAL_while:
{
AST __t143 = _t;
AST tmp100_AST_in = (AST)_t;
match(_t,LITERAL_while);
_t = _t.getFirstChild();
wexpr = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
wbody = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
_t = __t143;
_t = _t.getNextSibling();

while (expr(wexpr).isTrue()) {
expr(wbody);
}

break;
}
case LITERAL_if:
{
AST __t144 = _t;
AST tmp101_AST_in = (AST)_t;
match(_t,LITERAL_if);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
ithen = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case ALPHA:
case DIGIT:
case WS:
case NL:
case COMMENT:
case LPAREN:
case RPAREN:
case LBRACE:
case RBRACE:
case PLUS:
case MINUS:
```

```
case MULT:
case DIV:
case SEMI:
case COMMA:
case ASSIGN:
case ARROW:
case EQ:
case NEQ:
case PLUSEQ:
case MINUSEQ:
case MULTEQ:
case DIVEQ:
case GT:
case LT:
case GE:
case LE:
case AND:
case OR:
case NOT:
case QOUTE:
case ID:
case NUMBER:
case STRING:
case STATEMENT:
case POINT:
case FUNC_CALL:
case ARGS:
case UNARY_MINUS:
case PAREN_EXPR:
case LITERAL_pattern:
case LITERAL_set_FG:
case LITERAL_set_BG:
case LITERAL_move:
case LITERAL_scale:
case LITERAL_rotate:
case LITERAL_for:
case LITERAL_while:
case LITERAL_if:
case LITERAL_else:
case LITERAL_include:
case LITERAL_break:
case LITERAL_print:
case LITERAL_or:
case LITERAL_and:
case LITERAL_not:
{
```

```
ielse = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
break;
}
case 3:
{
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
_t = __t144;
_t = _t.getNextSibling();

if (a.isTrue()) {
expr(ithen);
} else if (ielse != null) {
expr(ielse);
}

break;
}
case LITERAL_scale:
{
AST __t146 = _t;
AST tmp102_AST_in = (AST)_t;
match(_t,LITERAL_scale);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
_t = __t146;
_t = _t.getNextSibling();
e.scale(a);
break;
}
case LITERAL_move:
{
AST __t147 = _t;
AST tmp103_AST_in = (AST)_t;
match(_t,LITERAL_move);
_t = _t.getFirstChild();
a=expr(_t);
```

```
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t147;
_t = _t.getNextSibling();
e.move(a, b);
break;
}
case LITERAL_rotate:
{
AST __t148 = _t;
AST tmp104_AST_in = (AST)_t;
match(_t,LITERAL_rotate);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
_t = __t148;
_t = _t.getNextSibling();
e.rotate(a);
break;
}
case PLUS:
{
AST __t149 = _t;
AST tmp105_AST_in = (AST)_t;
match(_t,PLUS);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t149;
_t = _t.getNextSibling();
r = a.plus(b);
break;
}
case MINUS:
{
AST __t150 = _t;
AST tmp106_AST_in = (AST)_t;
match(_t,MINUS);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
```

```
_t = __t150;
_t = _t.getNextSibling();
r = a.minus(b);
break;
}
case MULT:
{
AST __t151 = _t;
AST tmp107_AST_in = (AST)_t;
match(_t,MULT);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t151;
_t = _t.getNextSibling();
r = a.times(b);
break;
}
case DIV:
{
AST __t152 = _t;
AST tmp108_AST_in = (AST)_t;
match(_t,DIV);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t152;
_t = _t.getNextSibling();
r = a.divide(b);
break;
}
case GT:
{
AST __t153 = _t;
AST tmp109_AST_in = (AST)_t;
match(_t,GT);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t153;
```

```
_t = _t.getNextSibling();
r = a.greaterthan(b);
break;
}
case LT:
{
AST __t154 = _t;
AST tmp110_AST_in = (AST)_t;
match(_t,LT);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t154;
_t = _t.getNextSibling();
r = a.lessthan(b);
break;
}
case GE:
{
AST __t155 = _t;
AST tmp111_AST_in = (AST)_t;
match(_t,GE);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t155;
_t = _t.getNextSibling();
r = a.greaterorequal(b);
break;
}
case LE:
{
AST __t156 = _t;
AST tmp112_AST_in = (AST)_t;
match(_t,LE);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t156;
_t = _t.getNextSibling();
```

```
r = a.lessorequal(b);
break;
}
case EQ:
{
AST __t157 = _t;
AST tmp113_AST_in = (AST)_t;
match(_t,EQ);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t157;
_t = _t.getNextSibling();
r = a.equals(b);
break;
}
case NEQ:
{
AST __t158 = _t;
AST tmp114_AST_in = (AST)_t;
match(_t,NEQ);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t158;
_t = _t.getNextSibling();
r = a.notequals(b);
break;
}
case LITERAL_not:
{
AST __t159 = _t;
AST tmp115_AST_in = (AST)_t;
match(_t,LITERAL_not);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
_t = __t159;
_t = _t.getNextSibling();
if (!(a instanceof DoubleVariable)) a.typeError("number");
else r = ((DoubleVariable)a).not();
break;
```

```
}
case LITERAL_or:
{
AST __t160 = _t;
AST tmp116_AST_in = (AST)_t;
match(_t,LITERAL_or);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
rhs_or = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
_t = __t160;
_t = _t.getNextSibling();
if (!(a instanceof DoubleVariable)) a.typeError("number");
else r = ((DoubleVariable)a).or(expr(rhs_or));
break;
}
case LITERAL_and:
{
AST __t161 = _t;
AST tmp117_AST_in = (AST)_t;
match(_t,LITERAL_and);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
rhs_and = (AST)_t;
if ( _t==null ) throw new MismatchedTokenException();
_t = _t.getNextSibling();
_t = __t161;
_t = _t.getNextSibling();
if (!(a instanceof DoubleVariable)) a.typeError("number");
else r = ((DoubleVariable)a).and(expr(rhs_and));
break;
}
case UNARY_MINUS:
{
AST __t162 = _t;
AST tmp118_AST_in = (AST)_t;
match(_t,UNARY_MINUS);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
_t = __t162;
_t = _t.getNextSibling();
r = a.negative();
```

```
break;
}
case PAREN_EXPR:
{
AST __t163 = _t;
AST tmp119_AST_in = (AST)_t;
match(_t,PAREN_EXPR);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
_t = __t163;
_t = _t.getNextSibling();
r = a;
break;
}
case ASSIGN:
{
AST __t164 = _t;
AST tmp120_AST_in = (AST)_t;
match(_t,ASSIGN);
_t = _t.getFirstChild();
name = (AST)_t;
match(_t,ID);
_t = _t.getNextSibling();
a=expr(_t);
_t = _retTree;
_t = __t164;
_t = _t.getNextSibling();
e.assign(name.getText(), a);
break;
}
case ID:
{
AST tmp121_AST_in = (AST)_t;
match(_t,ID);
_t = _t.getNextSibling();
r = e.getSymbol(tmp121_AST_in.getText());
break;
}
case NUMBER:
{
AST tmp122_AST_in = (AST)_t;
match(_t,NUMBER);
_t = _t.getNextSibling();
r = new DoubleVariable(Double.parseDouble( tmp122_AST_in.getText() ));
break;
```

```
}
case STRING:
{
AST tmp123_AST_in = (AST)_t;
match(_t,STRING);
_t = _t.getNextSibling();
r = new StringVariable(tmp123_AST_in.getText());
break;
}
case POINT:
{
AST __t165 = _t;
AST tmp124_AST_in = (AST)_t;
match(_t,POINT);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t165;
_t = _t.getNextSibling();
r = new PointVariable(a, b);
break;
}
case ARROW:
{
AST __t166 = _t;
AST tmp125_AST_in = (AST)_t;
match(_t,ARROW);
_t = _t.getFirstChild();
a=expr(_t);
_t = _retTree;
b=expr(_t);
_t = _retTree;
_t = __t166;
_t = _t.getNextSibling();
e.drawLine(a, b); r = b;
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
catch (RecognitionException ex) {
```

```
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return r;
}

public final Variable[]  arg_list(AST _t) throws RecognitionException {
Variable[] r;

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

r = null;
Variable a;
ArrayList<Variable> list;


try {      // for error handling
AST __t168 = _t;
AST tmp126_AST_in = (AST)_t;
match(_t,ARGS);
_t = _t.getFirstChild();
list = new ArrayList<Variable>();
{
_loop170:
do {
if (_t==null) _t=ASTNULL;
if ((_tokenSet_0.member(_t.getType()))) {
a=expr(_t);
_t = _retTree;
list.add(a);
}
else {
break _loop170;
}

} while (true);
}
_t = __t168;
_t = _t.getNextSibling();
r = list.toArray(new Variable[list.size()]);
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
```

```
_retTree = _t;
return r;
}

public final String[]  def_arg_list(AST _t) throws RecognitionException {
String[] r;

AST def_arg_list_AST_in = (_t == ASTNULL) ? null : (AST)_t;
AST id = null;

r = null;
ArrayList<String> list;


try {      // for error handling
AST __t172 = _t;
AST tmp127_AST_in = (AST)_t;
match(_t,ARGS);
_t = _t.getFirstChild();
list = new ArrayList<String>();
{
_loop174:
do {
if (_t==null) _t=ASTNULL;
if ((_t.getType()==ID)) {
id = (AST)_t;
match(_t,ID);
_t = _t.getNextSibling();
list.add(id.getText());
}
else {
break _loop174;
}

} while (true);
}
_t = __t172;
_t = _t.getNextSibling();
r = list.toArray(new String[list.size()]);
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return r;
```

```
}


public static final String[] _tokenNames = {
"<0>",
"EOF",
"<2>",
"NULL_TREE_LOOKAHEAD",
"ALPHA",
"DIGIT",
"WS",
"NL",
"COMMENT",
"LPAREN",
"RPAREN",
"LBRACE",
"RBRACE",
"PLUS",
"MINUS",
"MULT",
"DIV",
"SEMI",
"COMMA",
"ASSIGN",
"ARROW",
"EQ",
"NEQ",
"PLUSEQ",
"MINUSEQ",
"MULTEQ",
"DIVEQ",
"GT",
"LT",
"GE",
"LE",
"AND",
"OR",
"NOT",
"QOUTE",
"ID",
"NUMBER",
"STRING",
"STATEMENT",
"POINT",
"FUNC_CALL",
"ARGS",
```

```
"UNARY_MINUS",
"PAREN_EXPR",
"\"pattern\"",
"\"set_FG\"",
"\"set_BG\"",
"\"move\"",
"\"scale\"",
"\"rotate\"",
"\"for\"",
"\"while\"",
"\"if\"",
"\"else\"",
"\"include\"",
"\"break\"",
"\"print\"",
"\"or\"",
"\"and\"",
"\"not\""
};

private static final long[] mk_tokenSet_0() {
long[] data = { 1107883276971401216L, 0L};
return data;
}
public static final BitSet _tokenSet_0 = new BitSet(mk_tokenSet_0());
}
```

8.5 Environment.java

```
/*
 * Created on Nov 8, 2005
 */

import guiModule.MainFrame;

import java.awt.Color;
import java.awt.Point;
import java.awt.geom.AffineTransform;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

import javax.swing.JFrame;

import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.collections.AST;

/**
 * @author Adam Vartanian
 */
public class Environment {

private Stack<AffineTransform> transforms = new Stack<AffineTransform>();
private Stack<Map<String, Variable>> symboltables = new Stack<Map<String, Variable>>();
private Map<String, Variable> currentTable = new HashMap<String, Variable>();
private Map<String, Variable> globalTable = currentTable;
private AffineTransform currentTransform = new AffineTransform();
private Map<String, Pattern> patterns = new HashMap<String, Pattern>();

private MainFrame frame;

public Environment(){
frame = new MainFrame();
patterns.put("sin", new Pattern.SinPattern());
patterns.put("cos", new Pattern.CosPattern());
patterns.put("tan", new Pattern.TanPattern());
patterns.put("sqrt", new Pattern.SqrtPattern());
patterns.put("ln", new Pattern.LnPattern());
}

public Variable getSymbol(String name) {
if (currentTable.containsKey(name)) {
return currentTable.get(name);
}
if (globalTable != currentTable) {
if (globalTable.containsKey(name)) {
return globalTable.get(name);
}
}
throw new RuntimeException("Reference to undefined variable.");
}

public void assign(String name, Variable value) {
```

```
currentTable.put(name, value);
}

public void scale(Variable a) {
//System.out.println("Scaling by " + a);

if (a instanceof DoubleVariable) {
currentTransform.scale(((DoubleVariable) a).val, ((DoubleVariable) a).val);
}else if(a instanceof PointVariable) {
currentTransform.scale(((PointVariable) a).x, ((PointVariable) a).y);
}else {
throw new RuntimeException("Attempted scaling by invalid type. Cannot scale by "
+ a.getClass().getName());
}
}

public void rotate(Variable a) {

if (!(a instanceof DoubleVariable)) {
throw new RuntimeException("Attempted rotation by invalid type. Cannot scale by "
+ a.getClass().getName());
}
currentTransform.rotate(((DoubleVariable) a).val);
//System.out.println("Rotating by " + a);
}

public void move(Variable a, Variable b) {
if(!(a instanceof DoubleVariable) || !(b instanceof DoubleVariable)){
throw new RuntimeException("Attempted translation by invalid type.");
}
currentTransform.translate( ((DoubleVariable) a).val, ((DoubleVariable) b).val);
//System.out.println("Moving by " + a + ", " + b);
}

public void drawLine(Variable a, Variable b) {
if(!(a instanceof PointVariable) || !(b instanceof PointVariable)){
throw new RuntimeException("Line coordinates are of invalid type.");
}
frame.drawLine(new Point((int)((PointVariable) a).x, (int)((PointVariable) a).y),
   new Point((int)((PointVariable) b).x, (int)((PointVariable) b).y), currentTransform);
//System.out.println("Drawing line from " + a + " to " + b + " using " + currentTransform);

}

public void setBackground(Variable a, Variable b, Variable c) {
if (!(a instanceof DoubleVariable) || !(b instanceof DoubleVariable) || !(c instanceof Doubl
```

```
throw new RuntimeException("Attempt to set background to non-numerical value.");
}
frame.setBG(new Color( (int)((DoubleVariable) a).val, (int)((DoubleVariable) b).val,
                       (int)((DoubleVariable) c).val ) );
//System.out.println("Set background: " + ((DoubleVariable) a).val + ", " + ((DoubleVariable
//" + ((DoubleVariable) c).val);
}

public void setForeground(Variable a, Variable b, Variable c) {
if (!(a instanceof DoubleVariable) || !(b instanceof DoubleVariable) || !(c instanceof Doubl
throw new RuntimeException("Attempt to set foreground to non-numerical value.");
}
frame.setFG(new Color((int)((DoubleVariable) a).val, (int)((DoubleVariable) b).val,
                      (int)((DoubleVariable) a).val));
//System.out.println("Set foreground: " + ((DoubleVariable) a).val + ", " + ((DoubleVariable
//+ ((DoubleVariable) c).val);
}

public void definePattern(String name, String[] args, AST c) {
if (patterns.containsKey(name)) {
throw new RuntimeException("Attempt to redefine pattern " + name);
}
//System.out.println("Defining pattern " + name + ", " + args.length + " args");
patterns.put(name, new Pattern(name, args, c));
}

public Variable callPattern(pWalker walker, String name, Variable[] args) throws Recognition
//System.out.println("Calling pattern " + name);

//save state
transforms.push(currentTransform);
symboltables.push(currentTable);

//create new state
currentTransform = new AffineTransform(currentTransform);
currentTable = new HashMap<String, Variable>();
Pattern f = patterns.get(name);
if (f == null) {
throw new RuntimeException("Attempt to call undefined pattern " + name);
}
if (args.length != f.args.length) {
throw new RuntimeException("Calling pattern " + name + " with wrong number of arguments (" +
" instead of " + f.args.length + ").");
}
for (int i = 0; i < args.length; i++) {
currentTable.put(f.args[i], args[i]);
```

```
}
Variable r = patterns.get(name).execute(walker, this);

// restore state
currentTable = symboltables.pop();
currentTransform = transforms.pop();
return r;
}

public void include(Variable s) {
if (!(s instanceof StringVariable)) {
throw new RuntimeException("Attempt to include with non-string value.");
}
File f = new File(((StringVariable) s).val);
if (!f.exists()) {
throw new PatternizerException("File " + s + " does not exist.");
}
InputStream in;
try {
in = new FileInputStream(f);
} catch (IOException e) {
throw new PatternizerException("Error trying to read file " + s + ".", e);
}
patternLexer pLexer = new patternLexer(in);
patternParser pParser = new patternParser(pLexer);
        try {
         pParser.program();
        } catch (Exception e) {
         throw new PatternizerException("Exception during parsing file " + s + ".", e);
        }

        CommonAST tree = (CommonAST) pParser.getAST();
        pWalker walker = new pWalker();
        walker.e = this;
        try {
            walker.expr( tree );
        } catch (RecognitionException e) {
         throw new PatternizerException("Recognition exception during parsing file " + s + '
        }
}

void display(){
frame.display();
}
}
```

8.6 PatternizerException.java

```
/*
 * Created on Nov 8, 2005
 */

/**
 * @author Adam Vartanian
 */
public class PatternizerException extends RuntimeException {

public PatternizerException() {
super();
}

public PatternizerException(String arg0) {
super(arg0);
}

public PatternizerException(String arg0, Throwable arg1) {
super(arg0, arg1);
}

public PatternizerException(Throwable arg0) {
super(arg0);
}

}
```

8.7 Pattern.java

```
/*
 * Created on Nov 8, 2005
 */

import antlr.RecognitionException;
import antlr.collections.AST;

/**
 * @author Adam Vartanian
 */
public class Pattern {

public static class SinPattern extends Pattern {
public SinPattern() { super("sin", new String[]{"x"}, null); }
```

```
public Variable execute(pWalker walker, Environment e) {
return new DoubleVariable(Math.sin(((DoubleVariable) e.getSymbol("x")).val));
}
}

public static class CosPattern extends Pattern {
public CosPattern() { super("sin", new String[]{"x"}, null); }
public Variable execute(pWalker walker, Environment e) {
return new DoubleVariable(Math.cos(((DoubleVariable) e.getSymbol("x")).val));
}
}

public static class TanPattern extends Pattern {
public TanPattern() { super("sin", new String[]{"x"}, null); }
public Variable execute(pWalker walker, Environment e) {
return new DoubleVariable(Math.tan(((DoubleVariable) e.getSymbol("x")).val));
}
}

public static class SqrtPattern extends Pattern {
public SqrtPattern() { super("sin", new String[]{"x"}, null); }
public Variable execute(pWalker walker, Environment e) {
return new DoubleVariable(Math.sqrt(((DoubleVariable) e.getSymbol("x")).val));
}
}

public static class LnPattern extends Pattern {
public LnPattern() { super("sin", new String[]{"x"}, null); }
public Variable execute(pWalker walker, Environment e) {
return new DoubleVariable(Math.log(((DoubleVariable) e.getSymbol("x")).val));
}
}

public String name;
public String[] args;
public AST body;

public Pattern(String n, String[] a, AST b) {
name = n;
args = a;
body = b;
}

public Variable execute(pWalker walker, Environment e) throws RecognitionException {
return walker.expr(body);
}
```

```
}
```

8.8 patternLexerTokenTypes.java

```
// $ANTLR 2.7.5 (20050128): "patternGrammar.g" -> "patternLexer.java"$

public interface patternLexerTokenTypes {
int EOF = 1;
int NULL_TREE_LOOKAHEAD = 3;
int ALPHA = 4;
int DIGIT = 5;
int WS = 6;
int NL = 7;
int COMMENT = 8;
int LPAREN = 9;
int RPAREN = 10;
int LBRACE = 11;
int RBRACE = 12;
int PLUS = 13;
int MINUS = 14;
int MULT = 15;
int DIV = 16;
int SEMI = 17;
int COMMA = 18;
int ASSIGN = 19;
int ARROW = 20;
int EQ = 21;
int NEQ = 22;
int PLUSEQ = 23;
int MINUSEQ = 24;
int MULTEQ = 25;
int DIVEQ = 26;
int GT = 27;
int LT = 28;
int GE = 29;
int LE = 30;
int AND = 31;
int OR = 32;
int NOT = 33;
int QOUTE = 34;
int ID = 35;
int NUMBER = 36;
int STRING = 37;
int STATEMENT = 38;
```

```
int POINT = 39;
int FUNC_CALL = 40;
int ARGS = 41;
int UNARY_MINUS = 42;
int PAREN_EXPR = 43;
int LITERAL_pattern = 44;
int LITERAL_set_FG = 45;
int LITERAL_set_BG = 46;
int LITERAL_move = 47;
int LITERAL_scale = 48;
int LITERAL_rotate = 49;
int LITERAL_for = 50;
int LITERAL_while = 51;
int LITERAL_if = 52;
int LITERAL_else = 53;
int LITERAL_include = 54;
int LITERAL_break = 55;
int LITERAL_print = 56;
int LITERAL_or = 57;
int LITERAL_and = 58;
int LITERAL_not = 59;
}
```

8.9 test.java

```
/**
 * test.java - test program for patternParser.
 *
 * author: Boriana Ditcheva October 2005
**/


import java.io.*;
import antlr.CommonAST;
import antlr.RecognitionException;


public class test
{
static boolean printTree = false;

public static void main(String[] args)
{
printTree = args.length >= 1 && args[0].equals( "-p" );
```

```
DataInputStream input = new DataInputStream(System.in);
patternLexer pLexer = new patternLexer(input);
patternParser pParser = new patternParser(pLexer);

//Parse the input program
try{
pParser.program();
} catch(Exception e){}


//  Build the parse tree
CommonAST pTree = (CommonAST)pParser.getAST();

if ( printTree )
{
//  Print the resulting tree out in LISP notation
System.out.println();
System.out.println( "================ PARSE TREE ======================" );
System.out.println( pTree.toStringList() );
}

try{
// Traverse the tree created by the parser
pWalker walker = new pWalker();
Variable r = walker.expr( pTree );
walker.display();
}
catch( RecognitionException e ) {
System.err.println( "Recognition exception: " + e );
}
}
}
```