Chapter 1

Language Reference Manual

Patternizer

Alexander, Yianni
Constantinides, Marinos
Ditcheva, Boriana
Tchakalov, Yavor
Vartanian, Adam

1.1 Lexical Conventions and Tokens

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

1.1.1 Comments

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

1.1.2 Identifiers

Identifiers must begin with a letter or an underscore symbol '_', but then can consist of any sequence of digits, letters, and the uderscore 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.

1.1.3 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.

1.1.4 Strings

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

1.1.5 Keywords

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

```
for    while    if        else
break  pattern  #include  print
scale  move     rotate
```

1.1.6 Operators and Other Tokens

Characters or sequences of characters used in the language:

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

1.2 Types

We have only 4 types which can be represented using identifiers. There are real numbers, points, patterns, and strings. There are no type identifiers, except for the declaration of patterns, thus our language is not statically-typed.

1.2.1 Reals

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

1.2.2 Points

Points are a pair of two real numbers, represented by an open paranthesis, 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)`.

1.2.3 Patterns

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.

1.2.4 Strings

String identifiers and variables are allowed in our language. Strings will be used to print statements, mostly for debugging purposes, and to specify filenames to include.

1.3 Expressions

1.3.1 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.

1.3.2 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 +, -, *, /, +=, -=, *=, /=. Arithmetic expressions for points is limited to translation +,+= and scaling *,*=. These will be performed on the points as they would be on vectors.

1.3.3 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.

1.3.4 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.

1.3.5 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.

1.3.6 Precedence of Operators

Below is a table of precedence for our operators.

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

1.4 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.

1.4.1 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.

1.4.2 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 can also be assigned using the '=' operator, and can be assigned other pattern identifiers or point identifiers. Patterns cannot be assigned explicit patterns, but rather are defined using explicit patterns.

1.4.3 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.

1.4.4 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 statments are `if` and `else`. A conditional statement is in the form:

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

or

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

1.4.5 Include

Other pattern files can be included by specifying which pattern files to include before any explicity 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.

1.4.6 Break

The language will support a `break` statement which operates as follows. If `break` is called within a loop, it will exit out of the most immediate loop. If `break` is called within a pattern definition, it will exit out of the current pattern and proceed to the next pattern definition. If called in a conditional it will exit out of both branchings.

1.5 Pattern Definition

Patterns are definid 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);
}
```

1.6 Internal Functions

1.6.1 Print Function

Printing strings out to console is allowed in our language, mostly for debugging purposes and printing pattern paramaters. It would be possible to print identifiers as well as strings. A print would appear like this:

```
print ("x = "x", y = "y);
```