LOGIRAY
White Paper

Team 5

Contents

Introduction to Logiray

From the vast ever-expanding universe of programming, our sights were focussed on imperative and logical programming. We have moulded a compiler capable of combining the best of both worlds. This compiler is basically extracting necessary parts from C/C++ and Prolog. We will make use of the Prolog features present within our language to implement solutions for symbolic AI. The features that have been borrowed from C++ will help us to implement solutions for deep learning with the use of arrays.

We would like to take a step back and briefly discuss the profuse pros Prolog provides. Broadly speaking it is a language that surrounds simple principles such as automatic backtracking, pattern matching, and tree-based data structures. This set of tools presents us with a flexible and powerful framework. There are multiple cases in which the implementations of symbolic computations will take several pages of indigestible code, whereas the Prolog implementation would take a single page and present it in a clear manner. Our language is gonna be a compiled language and its type-checking is static. Like all other statically typed programming languages our language is also compiler-based, this will cause the type checking to execute while the code is being compiled.

Logiray and it's environment has been designed to execute operations on arrays required for deep learning and the logical operations required for symbolic AI. Taken individually the above specifics can be found in multiple environments, the novelty here is combining them in a compile-time system and providing optimal results.

Language Specifications

Logiray walks on the footpath of C to a certain degree, this provides an extra edge to programmers as they will see familiar terms.

2.1 Data_Types

Primitive Data Types

Primitive data types are inbuilt/predefined data types that a user can directly access to declare desired variables.
First and foremost let's go through all the primitive data types that have been included in Logiray.

- **int:** The integer data type is declared by using the keyword int.
- **float:** Floating Point literals are stored with the help of the floating-point data type. It is declared using the keyword float.
- **char:** The character data type is used for the storage of characters. The keyword used is char. It generally takes up 1 byte of memory space.
- **bool:** This data type is typically used for the storage of boolean or logical values. It can either store true or false, the keyword used is bool. It generally takes up 1 byte of memory space.

Derived Data Types

The data types which are generally derived from the previously defined primitive data types are called derived data types. In Logiray we have both array and string as derived data types.

The struct is going to be our **user-defined data type**. We have decided to make use of the similarities between classes and structs and use structs for the deficiency of classes.

Array:

We can define an array as a collection of data objects that have been arranged sequentially, generally of the same data type.

Syntax: int array_name[array_size];
Here array_name is the name of the array, and array_size is the length.

String:

Just like the traditional definition in computer programming, it is basically a sequence of characters. It can be a variable or a literal constant that has been defined.

2.2 Comments

Logiray supports single line as well as multi-line comments.

- Single line comment : Starts with a hash symbol (#) and ends when a newline or EOF encounters.
  - Example :

```
int v;                        #This is a comment
```

- Multi line comment : Starts with a hash followed by a star (#*) and ends with star followed by hash(*#).
  - Example :

```
int v;                        #*Here starts a multi-line comment
It continues untill closing characters are encountered *#
```

2.3 White Space

White space consists of any sequence or combination of the following characters:

- Blank (ASCII 32)
- ‘\t’ (tab, ASCII 9)
- ‘\n’ (newline, ASCII 10)
- ‘\f’ (form feed, ASCII 12)
- ‘\r’ (carriage return, ASCII 13)
- ‘\v’ (vertical tab, ASCII 11)

2.4 Loops

If we want to execute a certain piece of code multiple times, loops provide an effective way of doing it. Logiray supports two kinds of loops. They are FOR and WHILE loop.

The main difference being increment statement execution. Increment is done after execution of the block in FOR loop whereas increment can be done anywhere in the block of WHILE loop.

Logiray also supports variable declaration in the initialization statement of FOR loop. Below are the example codes for loops :

- FOR loop :

```
for(i=0;i<10;i++)
{
    s+=i;
}
```

```
for(int i=0;i<10;i++)
{
    s+=i;
}
```

- WHILE loop :

```
i = 0;
while(i<10)
{
    s+=i;
    i++;
}
```

2.5 Conditional Statements

Checking a condition and changing / adjusting the behaviour of the code according to that specification, is necessary in almost every language to design practical programs. One of the simplest ways to do this is to use the if statement. Its syntax is as follows,

```
if(condition)
{
    expression
}
```

The program checks the condition within the brackets following the “if”, and returns a boolean value. If the boolean value is true the expression with the curly braces is executed if it is false the expression within is omitted.

Another possibility for conditional statements is shown below,

```
if(condition)
{
    expression
}
else if(condition)
{
    expression
}
else
{
    expression
}
```

What happens here is that after checking the if statement and failing to enter the loop, the condition within the else if loop is checked and if true the expression within it is carried. It is possible for this else if loop statement to be trailed by many others. We can also have an else statement which executes the expression(s) within it if all the conditional statements above it fail to execute.

2.6 Operators

Logiray supports various operators supported in C. Below is the table of operators with their precedence :

Precedence	**Operator**	**Description**	**Associativity**
1	++	Postfix increment	Left-to-right
	--	Postfix decrement	
2	++	Prefix increment	Right-to-left
	--	Prefix increment	
	!	Logical not	
	~	Bitwise not	
3	*	Multiplication	Left-to-right
	/	Floating point division	
	// or \	Integer division	
	%	Modulo (remainder)	
4	+	Addition	Left-to-right
	-	Subtraction	
5	<<	Bitwise left shift	Left-to-right
	>>	Bitwise right shift	
6	<	Less than	Left-to-right
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
7	==	Equal to check	Left-to-right

	is		
	!=	Not equal to check	
	isnot		
8	&	Bitwise and	Left-to-right
9	^	Bitwise xor	Left-to-right
10	|	Bitwise or	Left-to-right
11	&&	Logical and	Left-to-right
12	||	Logical or	Left-to-right
13	=	Assignment	Right-to-left
	+=	Assignment by sum	
	-=	Assignment by difference	
	*=	Assignment by product	
	/=	Assignment by quotient	
14	,	comma	Left-to-right

Operations on matrix :

Matrices are an important part in many algorithms and having an ease to work with them will make programming a lot easier and fun. Logiray aims to provide a simple and effective way to operate on matrices. Following are the operations which could be applied on matrices, here A, B, C are matrices :

- Addition:

  ```
  C = A + B
  ```

  The result of matrix addition on A and B will be stored in C. Dimensions of all three matrices must match.

- Subtraction:

  ```
  C = A - B
  ```

  The result of matrix subtraction on A and B will be stored in C. Dimensions of all three matrices must match.

- Multiplication:
  Logiray provides 2 ways for multiplication on matrices. One is normal matrix multiplication (this is achieved by operator @). Another is element-wise multiplication of both matrices.

  ```
  C = A @ B
  ```

  The result of matrix multiplication on A and B will be stored in C. The dimensions of A and B should match as is required for matrix multiplication.

  ```
  C = A * B
  ```

  The result of element-wise multiplication on A and B will be stored in C. Dimensions of all three matrices must match.

Features Removed from C and C++

There are too many features in C and C++ that are not necessary or are least used. This makes it difficult for programmers to know about functionality of every feature and its implementation.
Logiray aims to make programming easy but providing all the essential features at the same time. Hence, following are the features of C and C++ that are dropped from Logiray.

- Pointers
- Enumerator
- Union
- Namespaces
- Preprocessors
- Typedef
- Goto statements
- Static and constant

A Glimpse into the Grammar

Let’s start by taking a sneak peek into the grammar of Logiray. We have included facts from Prolog. By adding facts we have given the programmer the power to declare facts and rules to the problem, and then make queries so that Logiray can return correct solutions. The level of abstraction that is provided to the programmer will make Logiray excellent for AI implementations because the programmer will be able to give prominence to the problem rather than individualistic commands that should be imposed on the computer or system.

The data structure namely list which Logiray borrows from Prolog is very crucial for problems related to AI. The list data structure is built in Prolog in contrast to many other languages. We have borrowed that feature, so now this allows us to write programs that require list handling faster and easier. The recursive nature of the list will allow us to use recursion extensively which will provide us an edge while computing AI problems in Logiray. To understand the grammar a bit better let us take a deeper dive into how the grammar has been defined and their respective functions.

First of all, let's start with the declaration of facts. Let Prolog-expr denote a random prolog expression that is being used in Logiray. The syntax for declaring a rule is to write down the expression and terminate it with a period **(.).**

Example:

Prolog-expr .

Up next we have the list data structure. For this example, we will use prolog-expr-list to denote a list that will be used in Logiray. Similar to the above example prolog-expr will be used to represent an expression that will be used in Logiray. The names conditional-expr and unary-opt used in the example are self explanatory, they represent any random conditional expression and unary operator used in Logiray respectively. Time to take a look at the different ways in which a list in Logiray can be declared.

prolog-expr-list:

```
  prolog-expr
| unary-opt prolog-expr
| conditional-expression
```

| prolog-expr prolog-expr-list

From the above points we can infer that an expression list in Logiray can be any of the following, a prolog expression, a unary operator trailed by a prolog expression, a conditional expression. The last one provides us the power to use recursion extensively.

To declare or to define statements in Logiray let us use the name prolog-def. The syntax for this would be to start with a prolog expression followed by a colon and a hyphen, which is later trailed by the list we have used. In the end we have a period.

Example:

prolog-expr :- prolog-expr-list.