Programming Language Concepts Coursework

COMP2212

Programming Language Report

Kristian Ivanov | kii1u20@soton.ac.uk

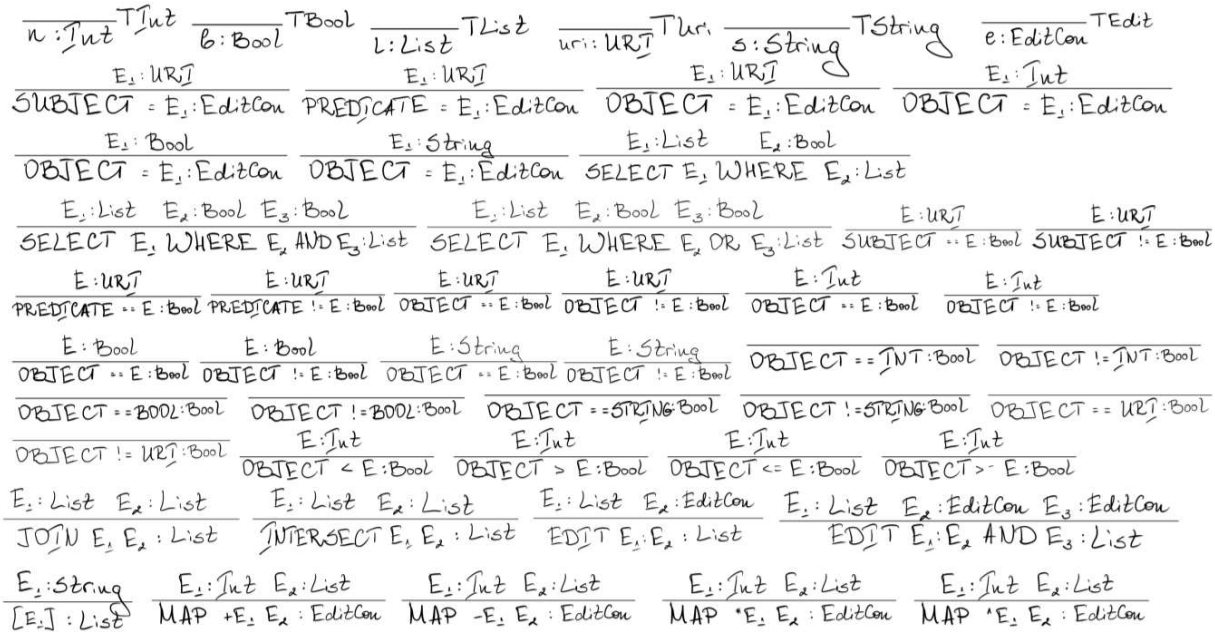
Lyubomira Kaynakchieva | lvk1u20@soton.ac.uk

Ivan Entchev | ie1g20@soton.ac.uk

04/05/2022

Types

The types supported by our language are INT for any number in Z (the set of integers); BOOL for true/false assertions; STRING for any sequence of characters surrounded in quotes; URI, which starts with http://; LIST for lists of objects and EditCon for representing the edit conditions in the EDIT statement. Type checking occurs before runtime, as Happy and Alex check for the correct usage of types before the resulting Abstract Syntax Tree is passed to the interpreter. Furthermore, only direct manipulation is permitted for data operations, resulting in a strong static typing in our language.



Error messages

In case of an incorrect user input, an error message displaying information about the type of the error is displayed.

Example 1: If a lexical error occurs, the user will receive an error message indicating the line and column where the spelling error occurred. The notification is written in the following format: "lexical error at line x column y."

Example 2: A user produces a correct program in terms of keywords and spelling. However, the user does not adhere to the grammar of the language. For example, the user may specify that a subject be equal to the number 3.

In that situation, an error message appears, stating " Grammar error at: line x, char y. This is not allowed in the syntax! " This error shows that the keywords were accurately entered, but the grammar rules of the language were violated. The error also indicates which line the problem is on and the position of the incorrect character.

Syntax

We adopted the query languages model for simplicity because most users are already familiar with its syntax. As keywords for operations, we have chosen SELECT, JOIN, WHERE, EDIT, and INTERSECT. It is important to note that the words must be capitalised.

The user can enter a list of triples using "[" and "]" with the various triples listed as they would in a standard .ttl file.

```
[@base <http://test.com/> .
<testSub><TestPred><TestObj> . ]
```

SELECT is used to specify which files or lists the user wants to work with. It can accept several files or lists separated by white space, as well as two files separated by a comma. Files and lists separated by white space are effectively JOINed and then other operations are performed. This is syntax sugar for selecting the conjunction of files or lists. The user has the option of using only files, only lists, or both.

```
SELECT foo.ttl bar.ttl WHERE OBJECT == 3
```

Files separated by a comma, on the other hand, are handled as separate lists and used to compare the contents of the two files. A SELECT statement produces a list of triples. The WHERE clause is always included after a SELECT statement.

```
SELECT foo.ttl, bar.ttl WHERE PREDICATES CONTAINS OBJECTS
```

WHERE is a conditional expression. It can accept one or more Boolean conditions. When numerous conditions are used in a WHERE statement, they are separated by AND or OR.

```
SELECT foo.ttl bar.ttl WHERE OBJECT == 3 AND SUBJECT == http://test.com/testSub
SELECT foo.ttl bar.ttl WHERE OBJECT == 3 OR SUBJECT == http://test.com/testSub
```

When two or more requirements must be met, the keyword AND is used. When only one of multiple conditions must be met, OR is used. It is important to keep in mind that OR binds more tightly than AND.

Users can also select triples based on the type of the OBJECT using the keywords INT, BOOL, URI, STRING.

```
SELECT foo.ttl bar.ttl WHERE OBJECT == INT
```

The language supports all the basic logical operators: ==(equals), < (less than), > (greater than), <= (less than or equal), >= (greater than or equal), != (not equal). == and != work with subjects, predicates, and objects to determine whether a subject/predicate/object is equal to a given URI. Objects can also be compared to Integers, Boolean values, or Strings by using the operators >, =, >=, !=, ==. When the SELECT clause takes two files separated by a comma, the user can use the CONTAINS keyword to determine whether the subjects/predicates/objects of the second file are contained in the subjects/predicates/objects of the first file. The keywords SUBJECTS, PREDICATES, and OBJECTS produce a list of a file's subjects, predicates, and objects. This procedure yields a list of triples from the second file that fulfil the CONTAINS condition.

```
SELECT foo.ttl bar.ttl WHERE OBJECT < 3
```

```
SELECT foo.ttl, bar.ttl WHERE PREDICATES CONTAINS OBJECTS
```

JOIN accepts two files or lists as parameters and joins them using conjunction. We have chosen that a JOIN statement does not require SELECT to select the files to work with for the sake of simplicity.

```
JOIN foo.ttl bar.ttl
```

EDIT is used to make direct changes to files or lists. To change a list, use SELECT, JOIN, or enter a list of triples using "[" "]", as these operators all return a list of triples. It is also capable of accepting a single file. To denote the changes, use the sign ":". Using SUBJECT = URI, PREDICATE = URI, or OBJECT = URI/int/bool/string, the user can alter all subjects, predicates, or objects in a list. The MAP function, which maps a mathematical function to all objects with integer values, is also available to the user. The mathematical functions can be + (addition), - (subtraction), * (multiplication), ^ (to the power of), and they take an integer value.

```
EDIT

SELECT foo.ttl WHERE OBJECT >= 0 AND OBJECT <= 99 :

PREDICATE = http://www.cw.org/problem5/#inRange

AND

MAP +1 OBJECTS
```

The INTERSECT command is used to find the intersection of two files or lists. It accepts two input parameters: two files, two lists, or a list and a file.

```
INTERSECT foo.ttl bar.ttl
```

All commands in the language can be nested within one another, for example.

```
SELECT SELECT foo.ttl bar.ttl WHERE OBJECT < 3

WHERE OBJECT > 0
```

The AND and OR keywords can be used to do most of these operations, although nesting is also an option.

Execution model

A list of lists of triples is the fundamental data structure that the interpreter manipulates (e.g. [[Triple Subject Predicate Object], [Triple ...]]). When you execute the interpreter, the Alex utility scans the program and divides it into a list of Tokens. The Tokens list is then sent to the Happy utility, which validates it to the language's grammar. This step examines and enforces the use of Boolean and mathematical operators (e.g., check if the user is trying to set the subject to integer, which is not allowed and throws an error). If everything is correct, the Tokens input list is transformed into an Abstract Syntax Tree. The interpreter understands the tree, executing each instruction as described in the Syntax section of this report. Each file is processed by a TurtleParser, which produces a list of lists of triples. The "[" "]" operation is also supplied to the TurtleParser to generate a list of lists of triples. The SELECT statement goes through the list of triples, checking them against the conditions in the WHERE clause. The EDIT statement runs through all of the triples in the list and modifies the subject/predicate/object based on the edit statements following the ":". Using common Haskell functions, the JOIN command combines two lists. Using common Haskell functions, the INTERSECT statement intersects two lists.