COMP2212: Glue User Manual

Harry Everett [hce1g16@soton.ac.uk, ID: 28270606]
David Jones [dsj1n15@soton.ac.uk, ID: 27549836]

1 What is Glue?

Glue is a concise and powerful language for performing conjunctive queries on columns of data. Put simply, this language can take multiple sets of data as an input and produce a filtered subset and/or a combined *(glued)* set of this data as an output.

This guide will take you through the overall structure of a Glue program and explain the individual functionality of its component `statements`, `operators` and `variables`.

2 Executing Glue Programs

A Glue program can be executed on your system command line by calling the Glue interpreter with your program path as an argument. For example: `.\myinterpreter program.cql`

3 Program Structure

3.1 Overview

A Glue program is structured into a set of sequential `statements`. The valid statement types are: `import`, `query` and `print`. The only structural rule applied is that `imports` must be declared before any other statement type, after which any combination of `query` and `print` statements can be defined. Statements can be split over multiple lines and can optionally be separated by a semicolon (`;`) to indicate clearly where statements finish.

Statements are executed in the order they are written (sequentially), this means that any print output will occur in this order and any variable assignments will only be available after the respective statement.

Listing 1: Structure Example

```
import "P.csv"  as P;
import "Q.csv"  as Q        Both of these line endings are okay!

A :: x1,x2 <- P(x1) ^ Q(x2)
                    ^ x1 == x2;      This is part of the query above!
print A
```

3.2 Variable Types

Table Names: These are used as a reference to a table. They must start with an uppercase letter and may be followed be any combination of upper case letters, numbers or underscores. For example, `A`, `A1`, `A_B`, but not `aB` or `1B`.

Column (Variables): These are used to reference specific columns of a table. They must start with a lowercase letter and may be followed by any combination of letters (lowercase and uppercase), numbers or underscores. For example, `a`, `a1`, `a_B`, but not `Ab` or `1a`.

3.3 Statement Types

3.3.1 Imports

Imports are used to assign the contents of a file to a table name. This allows the data to be referenced in `queries` and `print` statements with this provided table name. Data is expected to be formatted as comma separated values with an equal number of values in each row. If an import is unsuccessful a suitable error message will show (see Section 7).

3.3.2 Queries

Queries handle the main functionality of the language. They can be considered in three separate parts:

```
[<TABLE> ::]  <OUTPUT VARS> <- <OPERATIONS>
```

Operations: Operations are responsible for getting the data required for the outputs. The combination methods and functionality depends on what operator types are used. See Section 4.

Outputs: The variables placed here form the output table. The output tables columns will be ordered in respect to this list. It is a requirement that all free variables (these are variables that are not existentially quantified) are in the output list. See the Existential Quantification section for detailed functionality (Section 4.2). This may also dictate row output order as rows will be in alphanumeric order.

Listing 2: Query Storage Example

```
MYTABLE :: x3, x2, x1 <- S(x1,x2,x3);
x1, x3 <- MYTABLE(x1, x1, x3);
print MYTABLE;
```

The first query is used to print an output before its own result is printed.

Table Storage [Optional]: By prefixing a query with a table name and two colons the result of a query is stored to the given table name (provided the table does not already exist); this can later be referenced in the same way as an imported table. Without this the output is automatically printed for the user with the system discarding the result.

3.3.3 Print Statements

As the result of a query is not automatically printed when its result is saved to a table, there is functionality for printing a loaded table (this is not limited to user made tables). This is achieved with the statement `print <Table Name>`. Any string can also be printed by using `print "<String>"`, this may be helpful for annotations of output data or titling. The backslash symbol can be used as an escape character to print quotation marks or new lines within a string (e.g. "`\n and \"`").

4 Operator Types

4.0.1 Variable Assignment

Data can be assigned to variables using previously imported or created tables (see Section 3.3.1) using the syntax `<TABLE>(<VARS>)` (where variables are comma separated). If the number of variables assigned is not equal to the tables arity an error is displayed as in Listing 17. Data is fetched by value from tables meaning that each reference to the same table will get a fresh copy of the data. If there are repeated variables inside the table declaration, such as in `A(x1, x2, x2)`, the table will only be populated with rows where the values in the repeated columns match.

4.1 Table Conjunction

Table conjunction is used to combine two tables (see Section 4.3 for equality uses). The combined table will have every valid match of rows from the left and right table. A valid match is one where if a variable is assigned to both tables, the value of that variable is the same in the row from each table. The resultant table will only have one column for each of the repeated variables. Conjugating any table with the empty table will result in the empty table. An example of table conjunction is `A(x1, x2, x3) ^ B(x4, x2, x5)`.

4.2 Existential Quantification

The existential quantification syntax in Glue is `$<VARS>.<OPERATIONS>`. An existential quantification is used to define bound (non-output) variables. The scope of bound variables is explained in Section 5. Multiple bound variables can be chained together in shorthand of `$x1,x2,x3.` or more bound variables can be defined inside of a existential quantification for different levels of scope.

4.3 Equality/Inequality

Equalities and Inequalities use the syntax `<Var> == <Var>` and `<Var> != <Var>` respectively. The equality `x1==x2` will filter the table for its scope such that only rows which satisfy the predicate: `x1` equals `x2` will remain. This must be used inline with a conjunction such as: `A(x1, x2, x3) ^ x1 == x2`. The equality operator is applied scope-wide, this means that any equality that is done with no existential quantification to the left of it will apply to the whole statement. Any equalities that are within an existential quantification's scope, will be applied before the scope is left during evaluation.
Inequality works similarly to Equality, however instead of filtering out non equal rows, it filters out any rows where the two variables are equal, leaving only rows where the variables do not equal each other.

5 Scope

Scope in Glue is simple:

- Any operators that are not inside of an existential quantification are applied query-wide.
- Any operators inside of one or more existential quantification's are applied to all other operators within that level of existential quantification.
- Any bound variables only exist within the Existential Quantification that defines them.

6 Comments

The language features both single line comments and multiline comments. Usage is similar to that of many popular languages such as C++ or Java with `//` starting a single line comment and `/*` starting a multiline comment (closed with `*/`). One major difference for added convenience is that multiline comments have depth; this means that if you use a multiline comment over a section of code which already has a multiline comment in, you do not need to worry about removing the endmarker of the inside multiline comment. Single line comment symbols inside of multiline comments will be ignored.

Listing 3: Commenting Example

```
import "S.csv" as S;/*
/* // MYTABLE :: */
x1, x2, x3 <- S(x1, x2, x3) */
print S
// test
```

This multiline comment does not interfere with the parent comment!

This is after both multiline comments are terminated so will execute!

7 Error Handling

The language has advanced error handling built in to make debugging as simple as possible. Every error message provides a location of exactly where the issue is, be this a syntax error or an error during interpretation. During interpretation a full stack is displayed so the expression hierarchy can be assessed. See Appendix B.1 for example error messages.

A Problem Solutions

A.1 pr1.cql

Listing 4: Problem 1

```
import "A.csv" as A
import "B.csv" as B

x1,x3,x2,x4 <- A(x1, x2) ^ B(x3, x4)
```

A.2 pr2.cql

Listing 5: Problem 2

```
import "A.csv" as A
import "B.csv" as B

x1,x2,x3 <- A(x1,x2) ^ B(x2,x3)
```

A.3 pr3.cql

Listing 6: Problem 3

```
import "P.csv" as P
import "Q.csv" as Q

x1,x2 <- P(x1) ^ Q(x2) ^ x1 == x2
```

A.4 pr4.cql

Listing 7: Problem 4

```
import "R.csv" as R

x1 <- $z . R(x1, z)
```

A.5 pr5.cql

Listing 8: Problem 5

```
import "A.csv" as A
import "B.csv" as B

x1,x2 <- $z . A(x1,z) ^ B(z,x2)
```

A.6 pr6.cql

Listing 9: Problem 6

```
import "R.csv" as R; // arity 1
import "S.csv" as S; // arity 1
x1 <- $z . R(x1)
        ^ S(x2, z);
```

A.7 pr7.cql

Listing 10: Problem 7

```
import "R.csv" as R;
x1, x2 <- $z1, z2 . R(x1,z1) ^ R(z1,z2) ^ R(z2,x2);
```

A.8 pr8.cql

Listing 11: Problem 8

```
import "R.csv" as R;
x1,x2,x3,x4,x5 <- R(x1,x2) ^ R(x2,x3) ^
                  R(x3,x4) ^ R(x4,x5) ^ x1==x5;
```

A.9 pr9.cql

Listing 12: Problem 9

```
import "A.csv" as A;
import "B.csv" as B;
import "C.csv" as C;
x1,x2 <- $z1, z2 . A(x1,z1) ^ B(z1,z2) ^ C(z2,x2);
```

A.10 pr10.cql

Listing 13: Problem 10

```
import "S.csv" as S;
x1,x2,x3 <- $z1, z2, z3, z4 . S(x1,x2,x3) ^
                              S(z1,z1,z2) ^
                              S(z3,z4,z4)
```

B Extra Features

B.0.1 Overview

- The lexer is fully monadic to allow for better error messages and allows functionality such as nested comment depth and strings with escape characters.
- The parser is fully monadic to allow for better error messages and to allow position data to be passed to the interpreter.
- The interpreter is fully monadic with position data routed to the lowest levels so that on an exception a helpful error stack can be displayed.
- Inequality is added as an operation alongside equality. See Section 4.3.
- Tables can be stored to table names so that long queries can be chopped up into smaller queries or the point data is printed can be manipulated.
- String printing support alongside any table.
- Syntax highlighting in Notepad++ using the XML file in Appendix B.2.

B.1 Error Messages

B.1.1 Syntax Errors

B.1.1.1 Non-Terminated String (Example)

Listing 14: Non-Terminated String Error

```
import "R.csv as R;
```

```
Lexing Error : String not terminated before EOF
```

B.1.1.2 Missing Variable Separator (Example)

Listing 15: Missing Variable Separator

```
import "P.csv" as P;
x1, x2 <- P( x1 x2 );
```

There is a missing comma separating the variables!

```
Parse Error (2,18): Incorrect placement of Variable 'x2'
```

B.1.1.3 Other Syntax Errors

- `Lexing Error : Multi-line comment not terminated before EOF (Depth <Depth>)`
- `Lexing Error (<Position>) : Unrecognised token`
- `Parse Error (<Position>) : Incorrect placement of Table <Table>`
- `Parse Error (<Position>) : Incorrect use of '::' Correct use is TABLE_NAME :: VARIABLES <- QUERY`
- `Parse Error (<Position) : Incorrect use of 'import' Correct use is import "FILEPATH" as TABLE_NAME`
- `And many more...`

B.1.2 Interpretation Errors

B.1.2.1 Import Error (Example Stack)

Listing 16: Example Import Error

```
import "C.csv" as C; // does not exist
```

```
Error during interpretation:
  In import at (1,1)
    CSV Load Error: C.csv: openFile: does not exist (No such file or
    directory)
```

B.1.2.2 Query Error (Example Stack)

Listing 17: Example Arity Error

```
import "R.csv" as R; // arity 1
import "S.csv" as S; // arity 1
x1 <- $z . R(x1)
          ^ S(x1, z);
```

This will cause an error as Table S has arity 1, but two variables are provided!

```
Error during interpretation:
  In query at (3,1)
    In expression at (4,12)
      Too many variables are assigned for Table 'S'
```

B.1.2.3 Other Interpretation Errors

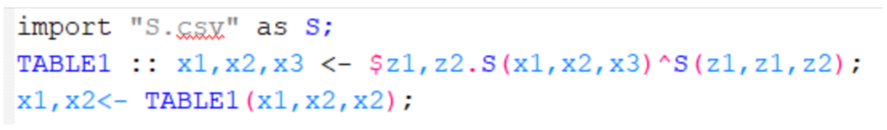
- `Imported CSV <Path> does not have the same number of rows in each column`
- `<Table> has not been assigned`
- `<Table> is already assigned`
- `Not enough variables are assigned for <Table>`
- `<Var> could not be found`
- `<Var> is already bound`
- `<Var> is already assigned and not bound`
- `<Var> is assigned but is not bound or in output`
- `<Var> is bound but is in output`
- `<Var> is not assigned but is in output`

B.2 Notepad++ Syntax Highlighting

B.2.1 XML File

```xml
<NotepadPlus>
    <UserLang name="Glue" ext="" udlVersion="2.1">
        <Settings>
            <Global caseIgnored="no" allowFoldOfComments="no" foldCompact="no" forcePureLC="0" decimalSeparator="
                 0" />
            <Prefix Keywords1="no" Keywords2="yes" Keywords3="no" Keywords4="yes" Keywords5="no" Keywords6="no"
                 Keywords7="no" Keywords8="no" />
        </Settings>
        <KeywordLists>
            <Keywords name="Comments">00// 01 02((EOL)) 03/* 04*/</Keywords>
            <Keywords name="Numbers, prefix1"></Keywords>
            <Keywords name="Numbers, prefix2"></Keywords>
            <Keywords name="Numbers, extras1"></Keywords>
            <Keywords name="Numbers, extras2"></Keywords>
            <Keywords name="Numbers, suffix1"></Keywords>
            <Keywords name="Numbers, suffix2"></Keywords>
            <Keywords name="Numbers, range"></Keywords>
            <Keywords name="Operators1">== != ^  $ . ( )</Keywords>
            <Keywords name="Operators2">(</Keywords>
            <Keywords name="Folders in code1, open"></Keywords>
            <Keywords name="Folders in code1, middle"></Keywords>
            <Keywords name="Folders in code1, close"></Keywords>
            <Keywords name="Folders in code2, open"></Keywords>
            <Keywords name="Folders in code2, middle"></Keywords>
            <Keywords name="Folders in code2, close"></Keywords>
            <Keywords name="Folders in comment, open"></Keywords>
            <Keywords name="Folders in comment, middle"></Keywords>
            <Keywords name="Folders in comment, close"></Keywords>
            <Keywords name="Keywords1">import&#x000D;&#x000A;as&#x000D;&#x000A;print</Keywords>
            <Keywords name="Keywords2">A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</Keywords>
            <Keywords name="Keywords3">&lt;- ::</Keywords>
            <Keywords name="Keywords4">a b c d e f g h i j k l m n o p q r s t u v w x y z</Keywords>
            <Keywords name="Keywords5"></Keywords>
            <Keywords name="Keywords6"></Keywords>
            <Keywords name="Keywords7"></Keywords>
            <Keywords name="Keywords8"></Keywords>
            <Keywords name="Delimiters">00&quot; 01 02&quot; 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19
                 20 21 22 23</Keywords>
        </KeywordLists>
        <Styles>
            <WordsStyle name="DEFAULT" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0" />
            <WordsStyle name="COMMENTS" fgColor="008000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="256"
                  />
            <WordsStyle name="LINE COMMENTS" fgColor="008000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting=
                 "0" />
            <WordsStyle name="NUMBERS" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0" />
            <WordsStyle name="KEYWORDS1" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS2" fgColor="0000FF" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS3" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS4" fgColor="0080FF" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS5" fgColor="0000FF" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS6" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS7" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="KEYWORDS8" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="OPERATORS" fgColor="FF0080" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0"
                 />
            <WordsStyle name="FOLDER IN CODE1" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0"
                 nesting="0" />
            <WordsStyle name="FOLDER IN CODE2" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0"
                 nesting="0" />
            <WordsStyle name="FOLDER IN COMMENT" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0"
                 nesting="0" />
            <WordsStyle name="DELIMITERS1" fgColor="808080" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
            <WordsStyle name="DELIMITERS2" fgColor="FF0000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
            <WordsStyle name="DELIMITERS3" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="
                 768" />
            <WordsStyle name="DELIMITERS4" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
            <WordsStyle name="DELIMITERS5" fgColor="0080FF" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
            <WordsStyle name="DELIMITERS6" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
            <WordsStyle name="DELIMITERS7" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
            <WordsStyle name="DELIMITERS8" fgColor="000000" bgColor="FFFFFF" fontName="" fontStyle="0" nesting="0
                 " />
        </Styles>
    </UserLang>
</NotepadPlus>
```

B.2.2 Image



```
import "S.csv" as S;
TABLE1 :: x1,x2,x3 <- $z1,z2.S(x1,x2,x3)^S(z1,z1,z2);
x1,x2<- TABLE1(x1,x2,x2);
```