NWQL: A declarative antidote to network management headaches

Submitted by: Jehanzeb Khan
UNI: JK3305
Course: COMS 4115 Fall 2015

Contents

1 Introduction 4
1.1 Interface Model vs. Interface Implementation 5
1.2 Correctness, Optimization and Compactness 5
2 NWQL: Network query language 5
2.1 Relational model as a choice for the abstraction layer 5
2.2 Previous work 6
3 Tutorial 6
3.1 Program 6
3.2 Global declarations 6
3.3 Statements 7
3.3.1 Expressions 7
3.3.2 Select statement 8
3.3.3 Insert statement 8
3.3.4 Delete statement 9
3.4 Compiling NWQL programs 9
4 Language Reference Manual 10
4.1 Tokens 10
4.1.1 Comments 10
4.1.2 Whitespaces 10
4.1.3 Identifiers 10
4.1.4 Reserved keywords 10
4.1.5 Punctuation and separators 11
4.1.5.1 Hash 11
4.1.5.2 Semicolons 11
4.1.5.3 Commas 11
4.1.5.4 Parentheses 12
4.1.5.5 Curly brackets 12
4.1.6 Literals 12
4.1.6.1 Integer literals 12
4.1.6.2 String literals 12
4.1.6.3 Boolean literals 12
4.1.6.4 AVP literals 13
4.2 NWQL Program 13
4.2.1 Global declarations 13
4.2.1.1 Syntax of global variable declarations 13
4.2.1.2 Semantic analysis on global variable declarations 14
4.2.2 Statements 15
4.2.2.1 Expressions 15
4.2.2.2 Select statements 17
4.2.2.3 Insert statements 19
4.2.2.4 Delete statements 25
5 NWQL Architecture 26
5.1 NWQL compilation pipeline and relevant files 26
6 Example NWQL programs and the generated code 27
6.1 Expression statements 27
6.2 Examples of useful NWQL programs 28
6.2.1 Program composed of select statements 28
6.2.2 Program composed of insert statements 29
6.2.3 Program composed of delete statements 32
7 Project logistics 33

7.1 Project plan 33
7.1.1 Phase I: Development of features on critical path 33
7.1.2 Phase II: Feature development 34
7.1.3 Phase III: Developing and executing end-to-end testing manually 35
7.1.4 Phase IV: Bug fixes and automated execution 35
7.2 Project milestones 35
7.3 Project log 36
7.4 Development environment 38
7.5 Test suites 39
7.6 Test case execution 39
7.6.1 Step1: Build the NWQL compiler 39
7.6.2 Step2: Execute test suite 40
7.6.3 Step3: Cleaning up the build [optional] 40
8 Lessons learned 41
9 Conclusion and next steps 42
9.1 Conclusion 42
9.2 Next Steps 42
10 Bibliography 43
11 Source code 44
11.1 ast.ml 44
11.2 parser.mly 46
11.3 scanner.mll 48
11.4 translate_env.ml 49
11.5 past.ml 49
11.6 translate.ml 50
11.7 typechecker.ml 56
11.8 compile.ml 64
11.9 71
11.10 nwql.ml 72

1 Introduction

Contemporary packet switched networks are complex and are commonly composed of multivendor pieces of equipment such as routers, firewalls, load balancers and switches. One of the main reasons behind this complexity is the proprietary nature of the software operating inside these devices [1]. Consequently limited opportunities for research and development exist in the field of computer networks. The so called control planes of these devices manipulate the forwarding planes by executing distributed routing protocols such as Open Shortest Path First. Although most of these protocols are standardized by such bodies as Internet Engineering Task Force, the implementation of these protocols in network equipment is usually proprietary with no APIs available for extensions. This lack of extensibility forces stakeholders such as Internet operators and the academic community to rely heavily on vendors to include expected features and protocols in product roadmaps. For e.g. a firewall application that filters network traffic based on administrative rules cannot be extended to support additional packet headers unless implemented by the respective vendor(s).

OpenFlow [2] switch architecture is an attempt to de-ossify and to decouple the control and data planes of packet switched networks. An OpenFlow switch is composed of a control protocol which is used to manipulate flow tables residing inside the OpenFlow compatible switches. Flow tables are used for packet lookups and packet forwarding. Entries in these tables represent packet matching and action rules governing the packets traversing through the respective network elements. As shown in Figure 1 below the OpenFlow control protocol allows such operations on flow tables as addition, modification and deletion of flow entries via the control protocol.

```
    +--+   +--+                      +--+   +--+                      +--+   +--+
    |H1|   |H2|                      |H1|   |H2|                      |H1|   |Hn|
    ++-+   ++-+                      ++-+   +--+                      ++-+   ++-+
+----+------+-----+              +----+------------+              +----+------+-----+
|OpenFlow Switch 1|              |OpenFlow Switch 2|              |OpenFlow Switch N|
+---------+--------+             +---------+--------+             +---------+--------+
 OpenFlow|Protocol                OpenFlow|Protocol                OpenFlow|Protocol
         |                                |                                |
+--------+--------------------------------+--------------------------------+--------+
|                                                                                   |
|                                                                                   |
|SDN Controller SouthBound API towards OpenFlow Switches for Flow tables management |
+-----------------------------------------------------------------------------------+
|                                                                                   |
|                                                                                   |
|SDN Controller NorthBound REST API towards network applications such as Routing    |
+--------+--------------------------------+--------------------------------+--------+
         |                                |                                |
     REST|API                         REST|API                         REST|API
         |                                |                                |
   +-----+-------+                +-------------+                +----------------+
   |ROUTING APP  |                |FIREWALL APP |                |TRAFFIC ENGR APP|
   +-------------+                +-------------+                +----------------+
```

Figure 1 OpenFlow architecture

This so called programmable networks paradigm fosters research and development of innovative applications on one hand but on the other hand it lifts the abstractions that are provided by traditional networking equipment. This trade off introduces new challenges facing the networking community. We will review a few such challenges below.

1.1 Interface Model vs. Interface Implementation

OpenFlow controllers' northbound interfaces expose implementation details such as REST API [3] messages for developing network applications. We argue that network administrators shouldn't be concerned with such questions as how to implement certain networking application, instead they should be equipped with a logical model that abstracts implementation aspects of the northbound interface. Contemporary controllers do not provide any such construct(s) and expect network administrators to take charge of imperative aspects of network applications.

1.2 Correctness, Optimization and Compactness

Northbound interfaces of contemporary controllers do not provide any constructs for verifying correctness and for generating optimal sequences of messages towards the controllers and leave the tasks related to optimization to the consumers of these interfaces, which increases the probability of runtime errors in the operational software. Furthermore the required lines of code as a result of imperative nature of the northbound API increase.

2 NWQL: Network query language

2.1 Relational model as a choice for the abstraction layer

We argue that application of relational model is appropriate for solving the abovementioned challenges due to the following relational model properties [4] observed in the OpenFlow framework

1. A packet switch network, composed of OpenFlow devices, may be viewed as a database of N-ary relations such as flow tables. We call this database Network Information Base [4]

2. All relations in a Network Information Base are defined over types, and values for each of the attributes in a given relation are selected from their respective types [4]

3. All relations in a Network Information Base have attributes qualifying the definition of candidate key [4]

4. All relations in a Network Information demonstrate integrity constraints such as no two tuples with identical values [4]

5. Order of tuples in a given relation doesn't affect the operations supported by the respective relations [4]

NWQL is a declarative programming language based on relational algebra. NWQL is a complied language that translates a program written in NWQL into a python program that sends sequence of OpenDaylight SDN controller NorthBound API messages [3], processes the received responses and presents the output to the user(s). NWQL provides simple yet powerful constructs which network administrator may use to query and modify Network Information Base without having to worry about imperative aspects such as how to process received responses. In short NWQL could be viewed as Network Manipulation Language— similar to Data Manipulation Language— which allows network administrators to query and to manipulate flow tables of the underlying network without having to worry about imperative aspects involved in the process.

2.2 Previous work

Jehanzeb Khan and Elliot Scull proposed and implemented **EZFLow: A declarative Programming Language for Performance Management in Software Defined Networks** as part of COMS-E6998 Fall 2014 course. NWQL differs from EZFlow in the following aspects:

1. EZFlow doesn't impose any semantic analysis —including type checking —as a result of which an incompatible program may translate and result in unknown behavior when executed

2. EZFlow is not a complied language as a result of which limited amount of correctness checks could be performed

3. EZFlow doesn't address the modification constructs such as insert and delete statements

3 Tutorial

3.1 Program

A NWQL program consists of zero or more global declarations and one or more statements.

3.2 Global declarations

A NWQL program begins with a global declaration section in which zero or more variables of valid data types and values may be declared. NWQL has four datatypes including Integers, Strings, Booleans and AVPs or attribute value pairs. NWQL doesn't have any notion of local declaration(s) hence any variable that a NWQL program uses must be declared in the declaration section. The values of these variables however can be changed anywhere in the program. Beginning and end of declaration section is marked by # as shown in Figure 2 below. Furthermore a valid variable declaration in NWQL constitutes identifying one of the four datatypes followed by name of the variable and finally assignment of a consistent value of the variable.

```
#int intvar=5; string stringvar="flow20" ; boolean boolvar=true
;avp avpvar={("name":"place")};#
```

Figure 2: Declaring global variables in the beginning of NWQL program

3.3 Statements

The smallest compilation unit in NWQL is statement. A valid NWQL program must have at least one statement. NWQL has four types of statements including expressions, select, insert and delete.

3.3.1 Expressions

NWQL has six types of expressions including integer literal, string literal, boolean literal, a list of attribute value pairs, an identifier that identifies a globally declared variable in variable declaration section and an assignment expression which allows assigning a different value to a previously declared global variable. Below are examples of each of the valid expression types.

```
#int intvar=5;# /* variable declaration section */

1; /*An expression of type integer literal*/

"hello"; /*An expression of type string literal*/

true; /*An expression of type boolean literal*/

{("name":"place")};/*An expression of type AVP literal*/

intvar;/*An expression of type identifier*/

intvar=20 /*An expression of type assignment*/
```

Figure 3: A sample NWQL program with expressions of different types in NWQL

3.3.2 Select statement

A select statement in NWQL is one of the network manipulation statements similar to the select command in SQL. Select statement allows querying a flow table to retrieve values of a subset of tuples. For e.g. with the help of a select statement a query can be composed that may return all the actions [such as dropping and forwarding a packet] performed by each of the flow entries in the given flow table. As shown in example program in Figure 4 a select statement is written using reserved keyword `select` followed by a comma separated list of one or more tuple names to be shown in output of the query, followed by `from` keyword and finally the name and location of the table being queried.

```
#int intvar=5; string stringvar="hello" ; boolean boolvar=true
;avp avpvar={("name":"place")};#

select actions from
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
```

Figure 4: A sample NWQL program with select statement

3.3.3 Insert statement

An insert statement in NWQL is another network manipulation statement similar to insert command in SQL. Insert statements allow adding flow entries in given flow tables. For e.g. with the help of insert statements different flow entries may be introduced in a flow table that may combine to form sophisticated network applications such as firewalls and routers. An insert statement is composed of `insert` keyword followed by the name of the table being updated, followed by a list of one of more column names between braces where each column name is separated by a comma, followed by the keyword `values` and finally the respective values for each of the columns between braces and separated by a comma.

```
#int intvar=5; string stringvar="flow20" ; boolean boolvar=true
;avp avpvar={("name":"place")};#

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
(stringvar,2,"00:00:00:00:00:00:00:02",8000,"0x800",{("DROP":"DRO
P")})
```

Figure 5: A sample NWQL program with insert statement

3.3.4 Delete statement

A delete statement in NWQL allows deleting a particular flow entry in a given flow table. A delete statement is composed of `delete` keyword followed by a comma separated names of flow entries to be deleted, followed by `in` keyword, followed by name of the particular switch from which the flow entries are being deleted, followed by `from` keyword and finally the name and location of the table.

```
#int intvar=5; string stringvar="flow1";string stringvar2="flow2" ;
boolean boolvar=true ;avp avpvar={("name":"place")};#

delete stringvar,stringvar2 in "00:00:00:00:00:00:00:02" from
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default',
auth=('admin','admin')
```

Figure 6: A sample NWQL program with delete statement

3.4 Compiling NWQL programs

Below is a sample NWQL program.

```
# string stringvar="flow20";#

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default'
,auth=('admin','admin')

(name,ingressPort,node,priority,etherType,actions) values
(stringvar,2,"00:00:00:00:00:00:00:02",8000,"0x800",{("DROP":"DROP
")})
```

Figure 7: A sample NWQL program

This program is based on a global variable declaration of datatype string and a statement of type insert. To compile this sample program the input file with .nwql extension must be specified and an optional output file name with –o command line argument may also be specified for output file name. If optional argument is not specific then the output file name is the same as the input filename.

```
>/nwql insert-man-col-name-check-value-identifier.nwql – o
hello.py
```

Figure 8: Compiling NWQL program with -o argument

```
>/nwql insert-man-col-name-check-value-identifier.nwql
```

Figure 9: Compiling NWQL program without -o argument

Since NWQL compiler compiles a NWQL program in to a python program, executing the compiled program is similar to executing any program written in python.

```
python insert-man-col-name-check-value-identifier
```

Figure 10: Executing a compiled program

4 Language Reference Manual

This section presents details of NWQL syntax including different kinds of tokens and production rules that allow permutation of different tokens to constitute compilation units such as expressions, statements and programs.

4.1 Tokens

4.1.1 Comments

NWQL supports comments in a program. Comments start with `/*` and are terminated by */. As shows in the example below everything between `/*` and `*/` is a comment and is ignored by the NWQL compiler during compilation.

```
1; /*An expression of type integer literal*/
"hello"; /*An expression of type string literal*/
true; /*An expression of type boolean literal*/
{("name":"place")};/*An expression of type AVP literal*/
intvar;/*An expression of type identifier*/
intvar=20 /*An expression of type assignment*/
```

Figure 11: Example of comments in a NWQL program

4.1.2 Whitespaces

Similar to comments, whitespaces including spaces, tabs, carriage return and line feeds are ignored by NWQL compiler.

4.1.3 Identifiers

Identifiers are used to assign values to global variables; to identify column names in such type of statements as select, insert and delete and to re-assign values to global variables in a NWQL program. Identifiers must begin with either uppercase or lowercase alphabetic character and may include additional sequence of characters that may contain a combination of one or more uppercase, lowercase, numeric and underscore characters. Identifiers may not be one of the reserved keywords as explained in the next section.

4.1.4 Reserved keywords

Reserved keywords are tokens that serve special purposes in a NWQL program may not be used as identifiers. Following keywords in NWQL are reserved and may not be used as identifiers in NWQL. Furthermore these keywords are case sensitive and may be used as string literals in any letter case.

avp	boolean	delete	false
from	in	insert	int
into	select	string	true
values	where	http	

Table 1: Reserved keywords in NWQL

4.1.5 Punctuation and separators

Following tokens are used as punctuation and separators.

4.1.5.1 Hash

Hash character is used to begin and to terminate global declaration section. For e.g. `#int intvar=5;#`

4.1.5.2 Semicolons

Semicolon are used as separators for following two constructs in NWQL programs

- ✓ As a separator between variable declarations in the global variable declaration section of a NWQL program
- ✓ As a separator between NWQL statements in a program. A program based on single statement doesn't require a separator

4.1.5.3 Commas

Commas are used as separators for following constructs in NWQL programs

- ✓ For enclosing multiple attribute value pairs separated by commas, in literals of AVP type
- ✓ For providing a list of expressions. Multiple expressions are typically used in
  - o select statements where multiple column names are separated by commas
  - o insert statements where multiple column names are separated by commas
  - o insert statements where multiple values are separated by commas

Below examples shows usage of commas in various NWQL constructs.

```
{("name":"place"),("age":"50")};

select node,name from
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defau
lt',auth=('admin','admin');

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defau
lt',auth=('admin','admin')
(name,node,priority,etherType,actions) values
("flow1","00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DROP")}
)
```

Figure 12: Example of Commas used as separators in different NWQL constructs

4.1.5.4 Parentheses

Parentheses are used as separators for following constructs in NWQL programs as shown in Figure 12.

- ✓ For enclosing multiple comma separated column names in insert statements
- ✓ For enclosing multiple comma separated values in insert statements

4.1.5.5 Curly brackets

Curly brackets are used to enclose avp literals as shown in Figure 12.

4.1.6 Literals

NWQL accepts four types of literals.

4.1.6.1 Integer literals

Integer literals include positive integers.

4.1.6.2 String literals

String literals include any sequence of characters between quotes such as “this is a string”.

4.1.6.3 Boolean literals

Boolean literals include either `true` or `false` as their values.

4.1.6.4 AVP literals

AVP literals are similar to key value pair where first object of AVP identifies a key and the second object identifies a value. AVPs are enclosed in braces and a list of AVP literals may be formulated using comma separated AVPs and curly braces in the beginning and the end of the list. For e.g. `{("name":"john")}` and `{("name":"place"),("age":"50")}` are both valid AVP literals.

4.2 NWQL Program

A NWQL program is composed of a list of zero or more global declarations and one or more statements. All valid NWQL programs are derived from the tokens mentioned in the Tokens section.

4.2.1 Global declarations

Since NWQL is modelled after declarative languages it doesn't support functions, procedures or methods, instead, the fundamental purpose of NWQL is to allow programmers to specify what needs to be accomplished rather than how to accomplish it. Consequently NWQL may not benefit much from allowing variable scoping hence all variables declared in NWQL are global and must be declared in the global declaration section in the beginning on a NWQL program.

4.2.1.1 Syntax of global variable declarations

Global variable declarations in NWQL are based on the following syntax.

1. A NWQL program begins with `#` separator that marks beginning of variable declaration section

2. Zero or more variables are declared in the variable declaration section using the following syntax

    a. One of the four data types including Int, String, Boolean and AVP is specific

    b. Variable name is specified that must adhere to the following identifier rule

        i. A valid identifier must start with an upper or lower case alphabet such as any alphabet between `a-z` or `A-Z` inclusive and at least one or more additional characters from a set of `a-z` or `A-Z` or `0-9` or an underscore character _. Figure 13 show some examples of valid and invalid identifiers. Please note that reserved keywords are not valid identifiers in NWQL. See section Reserved keywords for more information on reserved keywords in NWQL

    c. Assignment symbol = is specified

d. One of the four expression types i.e. integer literal, string literal, AVP literal and Boolean literal is specific as a value for the variable being declared. The value must be consistent with the specified datatype. Please refer to the section Semantic analysis on global variable declarations for more details

e. A semicolon which concludes the respective global variable declaration

3. `#` token that marks end of the variable declaration section

```
Aa  is a valid identifier
aa  is a valid identifier
a9  is a valid identifier
a_  is a valid identifier
abcded_  is a valid identifier
9 is an invalid identifier
@  is an invalid identifier
$  is an invalid identifier
A$  is an invalid identifier
Select  is an invalid identifier
```

Figure 13: Valid and invalid identifiers in NWQL

4.2.1.2 Semantic analysis on global variable declarations

Following semantic analysis is performed on global variable declarations.

1. NWQL compiler checks if a variable being declared already exists in the variable declaration section. If it does then compilation fails with the error **"Variable <name of the variable > already defined"**. Figure 14 shows an example of duplicate variable declaration and the respective compilation error generated

```
#int intvar=5; int intvar=9 ; boolean boolvar=true ;avp
avpvar={("name":"place")};#

Failure "Variable intvar already defined"
```

Figure 14: Example of a program with Duplicate global variable declaration

2. NWQL compiler checks if the value being assigned to the declared variable is consistent with its data type. Table 2 shows data types and the range of acceptable values for each of the data types. Figure 15 shows an example of a program where the declared datatype for a variable is not consistent with the value being assigned which results in compilation failure

Data type	**Acceptable values**
Int	Integer literals only
String	String literals between quote
Boolean	true or false

AVP	{(<string>:<string>)}

Table 2: Valid literals for NWQL data types

```
#int intvar=5; int intvar2="hello" ; boolean boolvar=true ;avp
avpvar={("name":"place")};#

Failure "Incompatible value for intvar2"
```

Figure 15: Example of a program with incompatible value in a variable

4.2.2 Statements

Unlike global variables which are optional in NWQL, a NWQL program must include at least one statement. Multiple statements are allowed in NWQL programs where each statement must be delimited from its subsequent statement using a semicolon. NWQL supports four types of statements as explained in the subsequent sections.

4.2.2.1 Expressions

NWQL supports six types of expressions including integer literals, string literals, Boolean literals, list of AVPs, identifiers and assignments. Expression is the smallest compilation unit in NWQL.

1. An expression of type integer literal is based on an integer value from a range of a set of positive integers

```
#int a9=5;#
1073741825 /*An expression
of type integer literal*/
```

Figure 16: Integer literals in NWQL

2. An expression of type string literal is based on any sequence of characters inserted between quotes

```
#int a9=5;#
“DROP” /*An expression of
type string literal*/
```

Figure 17: String literals in NWQL

3. An expression of type Boolean literal is based on either `true` or `false` keywords

```
#int a9=5;#
true /*An expression of
type boolean literal*/
```

Figure 18: Boolean literals in NWQL

4. An expression of type AVP list literal is based on an opening and closing curly brace and one or more pairs of attribute-value where each of the AVP is enclosed in round brackets. Furthermore multiple AVPs are separated using commas Figure 19 shows the syntax tree of AVP list type literals and

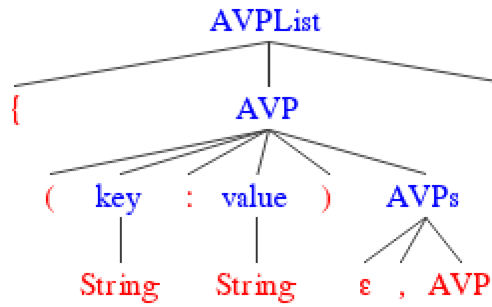

Figure 19: AST of AVP literals in NWQL

```
#int a9=5;#

{("name":"john")};
{("name":"john"),("age":"18")};
{("name":"john"),("age":"18"),("address":"4205 shipp dr,
Mississauga, ON, L4Z 2Y9")}
```

Figure 20: Example of AVP List literals in NWQL

5. An expression of type identifier where identifier must adhere to rules mentioned in the section Syntax of global variable declarations

6. An expression of type assignment allows changing values [not types] of globally declared variables in a NWQL program. Assignment expression must start with a variable name followed by an equal sign and a literal of a type consistent with the datatype of the respective variable

```
#int a9=5;#
a9 = 15
```

Figure 21: Example of assignment expression

4.2.2.1.1 Semantic analysis on expressions

Following semantic analysis is performed on statements of type expression.

1. For expressions of type assignment the value being assigned is checked to ensure that it is compatible with the datatype declared for the respective variable in the global variable declaration section. If the value being assigned is incompatible the program compilation fails with the error shown in Figure 22

```
#int a9=5;#
a9 = "hello"

(Failure "Incompatible value for a9")
```

Figure 22: Program compilation fail due to incompatible value in assignment expression

2. For expression of type identifier [calling a variable] NWQL checks if the variable exists, and if the variable doesn't exist the compilation fails an error shown in Figure 23

```
#int a9=5;#
unknownvar = "hello"

(Failure "Variable unknownvar does not
exist")
```

Figure 23: Program compilation fails due to the called variable not declared

4.2.2.2 Select statements

A valid select statement must start with the `select` keyword followed by a list of one or more column names where multiple column names are separated by commas, followed by the keyword `from` and finally the name of the table being queried. As shown in table below NWQL currently supports fifteen column names, all column names in select statements must be one of the fifteen supported column names, invalid column names in select statement results in compilation failure with the error message printing out the name of the invalid column. Table 3 shows supported column names in select statements.

actions	etherType	hardTimeout	idleTimeout	ingressPort
Name	Node	nwDst	nwSrc	priority
protocol	tpDst	tpSrc	vlanId	vlanPriority

Table 3: Supported column names in select statements

A table name in NWQL represents a hyperlink where the flow table being queried by the select statement, is expected to reside. NWQL doesn't allow an arbitrary collection of characters to be represented as table name instead it enforces programmers to adhere to strict format based on the following rules.

1. All table names in select statement must start with the string `http://`
2. Following `http://` a valid IPv4 address where the table is located must be specified
3. Following IPv4 address a valid transport port number between 0 and 65535 must be specified. The IPv4 address and port number must be separated by a colon

4. Following the transport port number, a path to the folder must be specific. Currently only the default path is support i.e. `/controller/nb/v2/flowprogrammer/default`

5. The authentication credential must be specified in `auth=('username','password')` format. Please note that authentication credentials are separated by the rest of the table name using a comma

An invalid table name results in Illegal table name compilation error. Figure 24 shows examples of illegal table names.

```
'http://192.168.0.14:8080/cont/nb/v2/flowprogrammer/default',auth=('admin','admin')

http://192.168.0.14:800000/controller/nb/v2/flowprogrammer/default',auth=('admin','admin')

http://192.168.0.10.12:8080/controller/nb/v2/flowprogrammer/default',auth=('admin','admin')
```

Figure 24: Examples of illegal table names in select statements

4.2.2.2.1 Semantic analysis on select statements

Following semantic analysis is performed on select statements.

1. Column names supplied as part of the select statements are validated. If one or more of the provided columns are not in the set of valid column names as shown in Table 3 then compilation fails with an error as shown in Figure 25

```
##
select qwerty from
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')

Failure "Invalid Column name qwerty"
```

Figure 25: Example of invalid column name in select statement

2. Table name supplied as part of the select statement in validated. If the supplied table name does not conform to the syntax mentioned above then the compilation fails with an error as shown in Figure 26.

```
##
select name from
'http://192.168.0.14:8080/controller/nb/v/flowprogrammer/def
ault',auth=('admin','admin')

(Failure "Illegal table name
'http://192.168.0.14:8080/controller/nb/v/flowprogrammer/def
ault',auth=('admin','admin')")
```

Figure 26: Example of invalid table name in select statement

4.2.2.3 Insert statements

A valid insert statement allows flow entries to be added to Opendaylight controller flow tables. These flow entries when combined operate as such network applications as routing and packet filtering. An insert statement

1. Starts with the `insert into` keyword

2. Followed by a table name

3. Followed by a braces-enclosed list of columns being added including mandatory and optional columns. Multiple column names must be separated by a comma

4. Followed by `values` keyword

5. Followed by a braces-enclosed list of expressions representing values of corresponding columns. Multiple values must be separated by a comma

Figure 27 shows examples of valid insert statements. Please note that an insert statement must have at least all the mandatory columns and their corresponding values present. Table 4 shows six mandatory column names for all Insert statements. If an insert statement doesn't include all of the six mandatory columns that the compilation will fail with the missing column name identified in the error message. Furthermore in addition to mandatory column names insert statements may include zero or more optional column names as shows in Table 5. Each of the column names—including mandatory and optional—supplied in the insert statement must have their corresponding values supplied following the syntax mentioned above. These values are themselves statements of type expressions however each column type may accept a subset of values typically allowed by the corresponding expression type. For e.g. the mandatory column `priority` and the optional column `tpSrc` both accept expression of types IntLit however the range of values for the `priority` column includes any positive integer whereas the range for values for `tpSrc` is between 0 and 65535 since transport port numbers may not be greater than 65535. This type system is enforced to ensure that NWQL programs when translated to Opendaylight API calls are correct and don't cause any runtime issues in the underlying network. Table 6 presents the range of values for each of the supported mandatory and optional columns.

```
# int intvar=500;#
insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow_123",2,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DROP
")});

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow1",2,"00:00:00:00:00:00:00:01",intvar,"0x800",{("DROP":"DRO
P")})
```

Figure 27: Example of valid insert statements

Name	Node	ingressPort
Priority	etherType	Actions

Table 4: Mandatory column names in Insert statements

hardTimeout	idleTimeout	nwDst
nwSrc	Protocol	tpDst
tpSrc	vlanId	vlanPriority

Table 5: Optional column names in Insert statements

Column name	Expression type	Range of values
name	StrLit or Id	Any string beginning with a lower or upper case alphabet character and include at least one additional uppercase/lowercase/digit/underscore character. Example Aa, A9, A_,
node	StrLit or Id	Any character except for newline
ingressPort	IntLit or Id	Any positive integer
Priority	IntLit or Id	Any positive integer
etherType	StrLit or Id	0x800 or 0x8100
hardTimeout	IntLit or Id	Any positive integer
idleTimeout	IntLit or Id	Any positive integer
nwDst	StrLit or Id	Any valid IPv4 address between 0.0.0.0 and 255.255.255.255 inclusive
nwSrc	StrLit or Id	Any valid IPv4 address between 0.0.0.0 and 255.255.255.255 inclusive
protocol	StrLit or Id	tcp or udp
tpDst	IntLit or Id	Between 0 and 65535 inclusive
tpSrc	IntLit or Id	Between 0 and 65535 inclusive
vlanId	IntLit or Id	Between 0 and 4096 inclusive
vlanPriority	IntLit or Id	Between 0 and 7 inclusive
actions	AVPList or id	One of the following values {("DROP":"DROP")} {("SET_VLAN_ID":”<0-2048>”)} {("SET_VLAN_PCP":"<0-7>")} {("SET_NW_DST":"<valid IPv4 address>")} {("SET_NW_SRC":"<valid IPv4 address>")} {("SET_TP_SRC":"<valid transport port between 0 and 65535>")} {("SET_TP_DST":"<valid transport port between 0 and 65535>")}

Table 6: Range of values for different columns in Insert statement

4.2.2.3.1 Semantic analysis on insert statements

Following semantic analysis is performed on insert statements.

1. Similar to select statements table name supplied as part of the insert statement is validated see Semantic analysis on select statements

2. Column names supplied as part of the insert statements are matched against the mandatory column names as shown in Table 4 if any one of the mandatory column names is missing the compilation of the program fails with the missing column name identified in the error message. Figure 28 shows an example of a program with a mandatory column name missing in the insert statement as well as the compilation error that is generated when compilation of this program fails

```
##

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/def
ault',auth=('admin','admin')
(name,ingressPort,priority,etherType,actions) values
("flow1",2,80,"0x800",{("DROP":"DROP")})


(Failure "Mandatory column node missing")
```

Figure 28: Mandatory column name missing in insert statement

3. Since in addition to mandatory columns an insert statement may include zero or more optional columns, the validity of optional columns is also checked

4. Insert statements don't accept duplicate column names therefore every column name must be unique within insert statements, however same column names may appear in multiple insert statements. If NWQL finds duplicate column names in the same insert statement the complication of the program fails and the generated error message points out the duplicate column name as shown in Figure 29.

```
##

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default',
auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions,node) values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DROP")},"
00:00:00:00:00:00:00:01")

(Failure "Column node appears multiple times, duplicates are not
allowed in NwQL")
```

Figure 29: Duplicate column names in insert statement

5. Insert statement must have equal number of column names and values—every value is an expression of appropriate type as shown in Table 6— on one hand if number of columns supplied is greater than the number of values provided then NWQL compiler assumes that values for some columns are missing and fails the compilation with the missing column name printed in the error message as shown in Figure 30, on the other hand if number columns supplied is less than that of values then NWQL compiler assumes that column names for some of the supplied values are missing and fails the compilation with the value missing its corresponding column name printed in the error message as shown in Figure 31.

```
##
insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions,nwSrc)
values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DR
OP")})

(Failure "No value for column nwSrc")
```

Figure 30: Insert statement containing more column names than values

```
##
insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions,nwSrc)
values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DR
OP")},"10.10.10.10",200,500)

(Failure "The expression 200 does not belong to any column")
```

Figure 31: Insert statement containing more values than column names

6. The final set of checks that NWQL compiler performs on insert statements are related to the range of values that are allowed for each of the mandatory and optional columns and shown in Table 6. If the value for any of the column names is out of range then the compilation fails with the error message identifying the column name with out of the range value. In addition to expressions of type IntLit/StrLit/BoolLit/AVPList identifiers representing globally declared variables may also be used as values for corresponding column(s), consequently all semantic analysis related to the identifier expression is also performed when identifiers are used in Insert statement see Semantic analysis on expressions. Figure 32, Figure 32, Figure 34 and Figure 35 show examples of a subset of type/value checking analysis performed on mandatory and optional columns in insert statements

```
##
insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("SET_DL_SR
C":"DROP")})

(Failure "Value of actions is out of range")
```

Figure 32: Insert statement unrecognized value for actions column

```
##
insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("SET_VLAN_
ID":"4096")})

(Failure "Bad value for SET_VLAN_ID")
```

Figure 33: Insert statement out of range value for action type SET_VLAN_ID

```
#int intvar=5; string stringvar="sctp";string
stringvar2="rdp" ; boolean boolvar=true ;avp
avpvar={("name":"place")};#

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions,protocol)
values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DR
OP")},intvar)

(Failure "Illegal literal type for protocol")
```

Figure 34: Inconsistent data type protocol column

```
#int intvar=5; string stringvar="sctp";string
stringvar2="rdp" ; boolean boolvar=true ;avp
avpvar={("name":"place")};#

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/de
fault',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions,protocol)
values
("flow1",2,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DR
OP")},stringvar)

(Failure "Bad value for protocol type")
```

Figure 35: Compatible datatype but out of range value for a column in insert statement

4.2.2.4 Delete statements

A valid delete statement allows removing flow entries from a given switch provisioned in Opendaylight controller. A valid delete statement

1. Starts with `delete` keyword
2. Followed by an expression of type StrLit or id identifying name of the flow entry being deleted
3. Followed by `in` keyword
4. Followed by one or more expressions of type StrLit or id identifying name of the openflow node where flow entries are currently installed. Multiple expressions are separated by commas
5. Followed by `from` keyword
6. Followed by the table name

4.2.2.4.1 Semantic analysis on delete statements

Following semantic analysis is performed on delete statements.

1. Similar to select and insert statements table name supplied as part of the insert statement is validated see Semantic analysis on select statements
2. Similar to insert statement duplicate column names are not allowed in delete statement see Semantic analysis on insert statements

Please note that the allowed values for flow names and node names are subject to the range of values mention in Table 6

5 NWQL Architecture

NWQL compiler consists of four key components including a scanner that reads tokens from NWQL programs, a parser that matches read tokens against production rules and arranges them in to compilation units, a type checker that perform semantic analysis, and a code generator that generates NWQL code in to executable python program. Figure 36 shows NWQL compilation pipeline highlighting stages that every NWQL program goes through until either the compilation fails with an error or executable code is generated in python, which implies successful compilation.

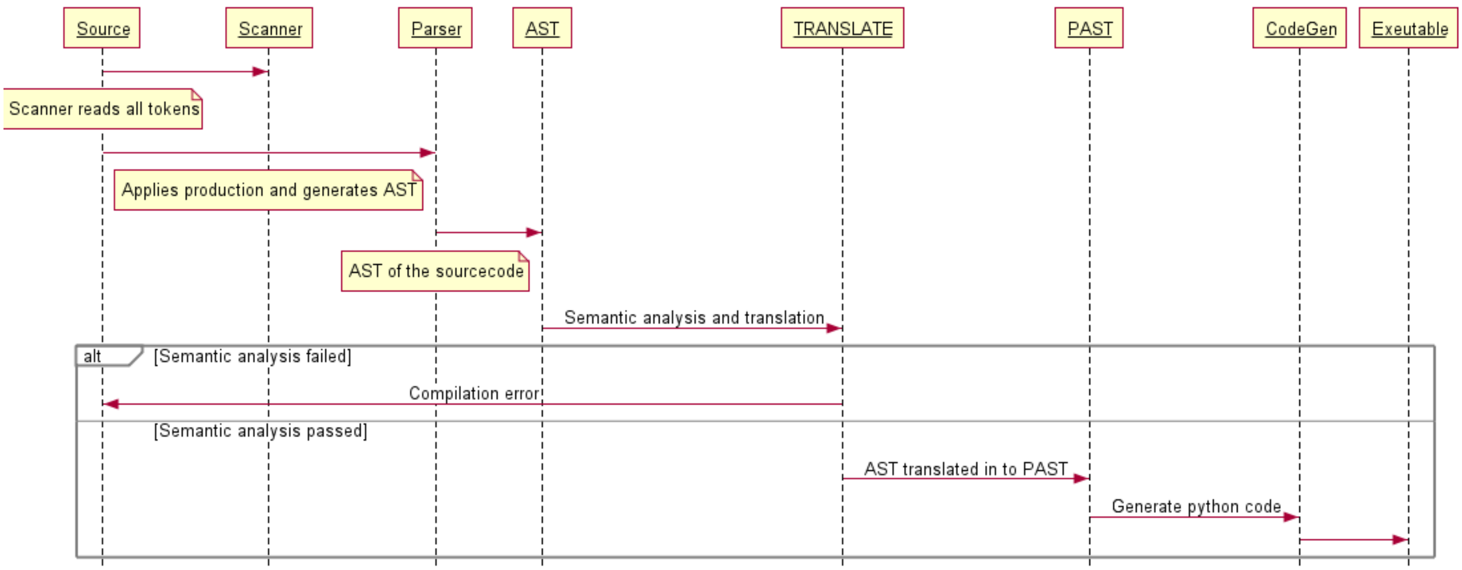

Figure 36: NWQL Compilation pipeline

5.1 NWQL compilation pipeline and relevant files

NWQL filename	Pipeline stage	Input	Output
scanner.mll	Scanner	NWQL source file	Tokens
parser.mly	Parser	Tokens	Program syntax tree
ast.ml	Parser		
translate_env.ml	Translate	Represent default runtime environment	
translate.ml	Translate	A NWQL program syntax tree	Equivalent python syntax tree
typechecker.ml	Translate	Used by translate.ml to perform type checking on different compilation units of a NWQL program such as statements before a python syntax tree for that program is generated	

compile.ml	Code Generation	A python syntax tree	Python executable code

6 Example NWQL programs and the generated code

6.1 Expression statements

Figure 37 shows a simple NWQL program comprised of multiple compilation units including global variable declarations; expressions of type integer, string, Boolean and AVP literals and expressions of type assignment. Figure 38 on the other end shows this NWQL program translated in to a python executable.

```
#int intvar=5; string stringvar="hello" ; boolean boolvar=true
; avp
avpvar={("hello1":"world1"),("hello2":"world2"),("hello3":"wor
ld3"),("hello4":"world4")};#

1;2;3;4;5;"string literal1"; "string literal2";"string
literal3";
true;false;{("hello1":"world1"),("hello2":"world2"),("hello3":
"world3"),("hello4":"world4")};

intvar=1;stringvar="string
literal1";boolvar=true;avpvar={("hello1":"world1"),("hello2":"
world2"),("hello3":"world3"),("hello4":"world4")}
```

Figure 37: A NWQL program with global variable declarations, statements of type expression literals and assignments

```
import requests
import json
intvar=5
stringvar="hello"
boolvar=True
avpvar=["hello1","world1","hello2","world2","hello3","world3","h
ello4","world4"]
1
2
3
4
5
"string literal1"
"string literal2"
"string literal3"
True
False
["hello1","world1","hello2","world2","hello3","world3","hello4",
"world4"]
intvar=1
stringvar="string literal1"
boolvar=True
avpvar=["hello1","world1","hello2","world2","hello3","world3","h
ello4","world4"]
```

Figure 38: A NWQL program translated into a python executable

6.2 Examples of useful NWQL programs

6.2.1 Program composed of select statements

The NWQL program and its translated python executable shown in Figure 39 and Figure 40 respectively demonstrate how select statements in NWQL abstract the underlying processing that would otherwise be required to accomplish the same result including sending HTTP request to OpenDaylight SDN controller, handling responses, filtering out undesired data and handling exceptions.

```
##
select actions,name,protocol,etherType from
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defaul
t',auth=('admin','admin')
```

Figure 39: A NWQL program comprised of select statement

```
import requests
import json
dummy="dummy"
try:

select=requests.get('http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin'))
         if (select.status_code == requests.codes.ok):
                collist=["actions","name","protocol","etherType",]
                select=select.json()
                flow_entries=select['flowConfig']
                for i in range(len(flow_entries)):
                        print('\n\n\n')

print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+')
                        for j in range(len(collist)):
                                if collist[j] in flow_entries[i]:
                                        print('---------------------------------------
-----------------------------------------')
                                        print(collist[j],flow_entries[i][collist[j]])

print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+')
                        print('\n\n\n')
         else:
                 print("HTTP error",select.status_code)
except requests.ConnectionError:
         print("Connection Unsuccessful")
```

Figure 40: A NWQL program based on select statement

6.2.2 Program composed of insert statements

The NWQL program and its translated python executable shown in Figure 41 and Figure 42 demonstrate how select insert statements in NWQL not only abstract the underlying processing but also allow writing multiple statements that combine to from end-to-end network applications such as firewalls, routers and traffic engineering.

```
##

insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow1",1,"00:00:00:00:00:00:00:01",80,"0x800",{("DROP":"DROP")}
);


insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow2",2,"00:00:00:00:00:00:00:01",70,"0x800",{("SET_TP_SRC":"5
060")});


insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow3",2,"00:00:00:00:00:00:00:01",60,"0x800",{("SET_NW_DST":"1
92.168.0.1")});


insert into
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/default
',auth=('admin','admin')
(name,ingressPort,node,priority,etherType,actions) values
("flow4",2,"00:00:00:00:00:00:00:01",50,"0x800",{("SET_VLAN_PCP":
"6")})
```

Figure 41: A NWQL program comprised of insert statements

```
import requests
import json
dummy="dummy"
url="http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defau
lt/node/OF/00:00:00:00:00:00:00:01/staticFlow/flow1"
putdata={
"installInHw":"true","name":"flow1","ingressPort":"1","node":{"id":
"00:00:00:00:00:00:00:01","type":"OF"},"priority":"80","etherType":
"0x800","actions":["DROP"],}
headers={"content-type":"application/json"}
try:

insert=requests.put(url,auth=('admin','admin'),data=json.dumps(putd
ata),headers=headers)
         if (insert.status_code==requests.codes.ok):
                 print("Flow entry successfully updated")
         elif (insert.status_code==requests.codes.created):
                 print("Flow entry successfully created")
         else:
                 print("HTTP error",insert.status_code)
except requests.ConnectionError:
         print("Connection Unsuccessful")
```

```
url="http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defau
lt/node/OF/00:00:00:00:00:00:00:01/staticFlow/flow2"
putdata={
"installInHw":"true","name":"flow2","ingressPort":"2","node":{"id":
"00:00:00:00:00:00:00:01","type":"OF"},"priority":"70","etherType":
"0x800","actions":["SET_TP_SRC=5060"],}
headers={"content-type":"application/json"}
try:

insert=requests.put(url,auth=('admin','admin'),data=json.dumps(putd
ata),headers=headers)
         if (insert.status_code==requests.codes.ok):
                 print("Flow entry successfully updated")
         elif (insert.status_code==requests.codes.created):
                 print("Flow entry successfully created")
         else:
                 print("HTTP error",insert.status_code)
except requests.ConnectionError:
         print("Connection Unsuccessful")
```

```
url="http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defau
lt/node/OF/00:00:00:00:00:00:00:01/staticFlow/flow4"
putdata={
"installInHw":"true","name":"flow4","ingressPort":"2","node":{"id":
"00:00:00:00:00:00:00:01","type":"OF"},"priority":"50","etherType":
"0x800","actions":["SET_VLAN_PCP=6"],}
headers={"content-type":"application/json"}
try:

insert=requests.put(url,auth=('admin','admin'),data=json.dumps(putd
ata),headers=headers)
         if (insert.status_code==requests.codes.ok):
                 print("Flow entry successfully updated")
         elif (insert.status_code==requests.codes.created):
                 print("Flow entry successfully created")
         else:
                 print("HTTP error",insert.status_code)
except requests.ConnectionError:
         print("Connection Unsuccessful")
```

Figure 42: A NWQL program based on multiple insert statements translated in to a python executable

6.2.3 Program composed of delete statements

The NWQL program and its translated python executable shown in Figure 43 and Figure 44 demonstrate how select delete statements in NWQL abstract the underlying processing.

```
##
delete "flow1" in "00:00:00:00:00:00:00:02" from
'http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defau
lt',auth=('admin','admin')
```

Figure 43: A NWQL program comprised of delete statements

```
import requests
import json
dummy="dummy"
url="http://192.168.0.14:8080/controller/nb/v2/flowprogrammer/defaul
t/node/OF/00:00:00:00:00:00:00:02/staticFlow/flow1"
headers={"content-type":"application/json"}
try:

remove=requests.delete(url,auth=('admin','admin'),headers=headers)
        if (remove.status_code==204):
                print("Flow entry successfully removed from the
controller")
        else:
                print("HTTP error",remove.status_code)
except requests.ConnectionError:
        print("Connection Unsuccessful")
```

Figure 44: A NWQL program based on multiple delete statement translated in to a python executable

7 Project logistics

7.1 Project plan

The project plan for designing, developing, testing and finalizing the NWQL compiler was developed based on lesson learned from past projects as well as pieces of advice given by Prof. Stephen during lectures. It was evident that rather than waterfall model having an agile mindset —which favors parallel development and testing of different components—would optimize the end-to-end results, hence only the components on the critical path were developed in the first phase. In the second phase each of the required functionalities was iteratively developed and functionally tested. The third phase involved developing and executing end-to-end testing manually. The fourth phase involved automating the test case execution and fixing identified bugs.

7.1.1 Phase I: Development of features on critical path

As shown in the Table 7 the first phase of the project included developing only those components that are dependents of other components. For e.g. the AST for NWQL is a direct dependent for scanner, python abstract syntax tree and the type checker, and indirectly also impacts parser, translator and the code generator hence the ambition of the first phase was to design and develop the AST as concretely as possible. Please note with the exception of the AST other components were developed with enough bare minimum functionality to lift dependencies and were finalized in later stages. Examples of such components include translator and type checker.

Features	Dependency
AST	None
Scanner	AST
Parser	Scanner
Python Abstract syntax tree	AST
Translator	PAST
Type checker	AST
Code generator	Translator

Table 7: Activities in the critical path

7.1.2 Phase II: Feature development

Phase II of the project involved developing and individually testing features in the following order.

1. Completing the front end of the compiler including AST, scanner and parser
2. Pretty printer [not a feature but the most invaluable tool] for testing the front end
3. Developed symbol table
4. Developed translation and semantic analysis for global variables
5. Code generation and testing global variable feature
6. Developed translation and semantic analysis for expressions
7. Code generation and testing expressions feature
8. Developed translation and semantic analysis for select statements
9. Code generation and testing select statements
10. Developed translation and semantic analysis for insert statements
11. Code generation and testing insert statements
12. Developed translation and semantic analysis for delete statements
13. Code generation and testing delete statements
14. Tested programs with statements of multiple types

7.1.3 Phase III: Developing and executing end-to-end testing manually

Phase III of this project involved creating a test object list and executing each of the test cases manually. All together approximately 135 test cases were written and executed manually.

7.1.4 Phase IV: Bug fixes and automated execution

Phase IV of the project involved fixing anomalies identified such as making cosmetic changes including messages that are printed when compilation fails.

7.2 Project milestones

Description	Date
Front end ready **with AST, parser and scanner J**an 18 2016	1/17/2016
Tested that programs including variables, expressions, select and insert are being generated	1/21/2016
Started working on semantic analysis Jan 22, 2016	1/22/2016
Added semantic analysis methods for global variable declaration and assignment expressions	2/9/2016
Compilation complete for simple expressions, statement of type expression and program Feb 22 2016, errors with assignment	2/22/2016
Select compilation done	3/7/2016
Translated the insert code	3/25/2016
Adjusted and executed manually the test cases for select, insert and delete statements April 30, 2016	4/30/2016
Adjusted and executed manually the test cases for	4/30/2016

select, insert and delete statements April 30, 2016	
executed all test cases automatically July 14, 2016	7/15/2016

7.3 Project log

Description	Date
Added lexer.mll file	1/6/2016
Initial commit with contributors	1/6/2016
Added pretty printer for Literals Jan 11 2016	1/10/2016
Intial ast Jan 11 2015 without the pretty printer functions	1/10/2016
pretty printer for select and selectRestrict	1/10/2016
remove redundant type for literals and merged it with type expr and finished the pretty printer for expressions	1/10/2016
pretty printer added for everything except Program Jan 12 2016	1/11/2016
pretty printer added for everything including Program Jan 12 2016	1/11/2016
Added compiled parser and ast jan 14 2016	1/13/2016
Changed AVP data type to a simple list of (string * string) tuples	1/17/2016
Changed AVP data type to a simple list of (string * string) tuples and verified all compilation Jan 18 2016	1/17/2016
Compiled all the files related ast, parser and scanner jan 14 2016	1/17/2016
Front end ready with ast, parser and scanner jan 18 2016	1/17/2016
Front end ready with ast, parser and scanner jan 18 2016	1/17/2016

Front end ready with ast, parser and scanner jan 18 2016	1/17/2016
verified that a problem with ONLY variables, with variables and statements and with only statements can be generated Jan 21 2016	1/20/2016
Added support for global variables including values and tested all valid programs that could be produced Jan 22 2016	1/21/2016
generated all types of valid programs using scanners including variable declarations, expressions, select statements, insert statements, update statements and delete statements Jan 21 2016	1/21/2016
tested that programs including variables, expressions, select and insert are being generated	1/21/2016
started working on semantic analysis Jan 22, 2016	1/22/2016
Added semantic analysis methods for global variable declaration and assignment expressions	2/9/2016
Started working on Python asbtract syntax tree and translation Feb 10 2016	2/10/2016
Completed translation from ast to pst for expressions and gvdecl, updated consistency check method only to return true/false Feb 11 2016	2/11/2016
Translation complete for expression, statement of type expression and program Feb 15 2016	2/15/2016
Compilation complete for simple expressions, statement of type expression and program Feb 22 2016, errors with assignment	2/22/2016
Added translation for select type statement Mar 5 2016	3/3/2016
Compilation in progress for select Mar 3 2016	3/4/2016

Select compilation done Mar 7 2016	3/4/2016
Insert compilation started Mar 19 2016	3/19/2016
Insert translation almost done Mar 20 2016	3/20/2016
Changed the typechecker.ml for type analysis of Insert statements	3/21/2016
Changed the typechecker.ml for type analysis of Insert statements Mar 21, 2016	3/21/2016
Translated the insert code Mar 25	3/25/2016
Added test cases for Insert, working on translation and compilation of Insert Apr 3 2016	4/3/2016
Executed test cases for insert statements manually on April 19, 2016	4/19/2016
Adjusted and executed manually the test cases for select, insert and delete statements April 30, 2016	4/30/2016
changed nwql.ml	4/30/2016
executed all test cases automatically July 14, 2016	7/15/2016
executed all test cases automatically July 14, 2016 and made changes to folder names	7/15/2016

7.4 Development environment

Compiler language: Ocaml version 4.02.3

Compiling helping tool: Ocamlyacc, Ocamllex, re2 package

Target environment: Python

Testing tools: Mininet and Opendaylight [hydrogen]

7.5 Test suites

Approximately 135 test cases were written and executed. The focus of these test cases was not only to provide coverage to a wide range of valid programs that could be written in NWQL but also to ensure that compilation of invalid programs results in expected errors. As shown in Figure 45, out of 135 programs only 65 programs successfully compile and the rest of the 70 programs are related to sematic analysis of different types of NWQL statements. Furthermore the widest coverage has been given to the statement of type Insert due to the level of semantic analysis required

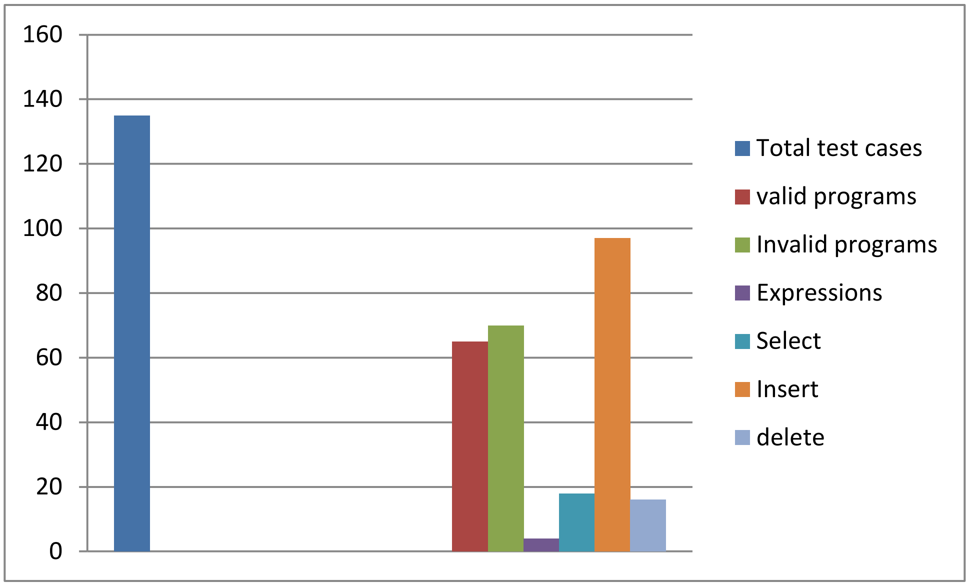

Figure 45: Snapshot of test suite coverage

7.6 Test case execution

7.6.1 Step1: Build the NWQL compiler

1. In the project folder browse to the folder `/finalproject/src`
2. Execute `make make` as shown below

```
caml@ocaml:~/coms4115/finalproject/src$ make make

ocamllex scanner.mll
88 states, 4320 transitions, table size 17808 bytes
1563 additional bytes used for bindings
ocamlyacc parser.mly
ocamlc -c ast.ml
ocamlc -c past.ml
ocamlc -c parser.mli
ocamlc -c scanner.ml
ocamlc -c parser.ml
ocamlc -c translate_env.ml
ocamlfind ocamlc -c -package re2 -thread typechecker.ml
ocamlc -c translate.ml
ocamlc -c compile.ml
ocamlc -c nwql.ml
ocamlfind ocamlc -o nwql -linkpkg -package re2 ast.cmo parser.cmo past.cmo
scanner.cmo translate_env.cmo -thread typechecker.cmo translate.cmo
compile.cmo nwql.cmo
```

7.6.2 Step2: Execute test suite

1. Navigate to the folder `/finalproject/test_cases`

2. Execute `./test.sh` as shown below. Please note that each of the 135 test cases will be executed and compared against the golden logs located at `/finalproject/test_cases/golden_test_suite`. The status of each of the test case printed in green color indicated passed while red indicates failed. All of the test cases should be passed.

```
ocaml@ocaml:~/coms4115/finalproject/test_cases$ ./test.sh
Compiling delete-dup-check-iden-iden.nwql ...
delete-dup-check-iden-iden test case passed.
Compiling delete-dup-check-strlit.nwql ...
delete-dup-check-strlit test case passed.
Compiling delete-invalid-expr-avplit.nwql ...
delete-invalid-expr-avplit test case passed.
Compiling delete-invalid-expr-boollit.nwql ...
delete-invalid-expr-boollit test case passed.
Compiling delete-invalid-expr-iden-avplit.nwql ...
```

7.6.3 Step3: Cleaning up the build [optional]

1. In the project folder browse to the folder `/finalproject/src`

2. Execute `make clean` to remove the built executable and associated files

8 Lessons learned

I learned several lessons throughout the project and in addition to stating the learned lesson I would also share some suggestions pertaining to how I overcome some of the challenges along the way.

Lesson	Suggestion
OCAML is based on an entirely different but exciting programming paradigm that traditional programmers might not be used to of. In spite of its unfamiliarity before enrolling for COMS 4115 course it is the most fun language I have ever programmed in. Some practice and good understanding of key OCAML concepts is crucial to turn this project in to fun	1. ***Developing Applications with Objective Caml*** is a very good book to understand some of the most important concepts that are abstracted from the reader in other books. Chapter 2, 3 and 4 of this book should ideally be finished before the first assignment is out ***2.*** Chapter 5 of the book ***The Functional Approach to Programming*** is also a very good resource to understand applications of algebraic data types. Algebraic data types is one of the most powerful features of OCAML that will be used over and over again throughout this project as well as in the assignments
The front end, specially the grammar of any language designed is only a subset of tasks required to write a compiler, add semantic analysis and code generation atop and the scope of the project increases twofold.	Don't try to design a generic language, rather try to select a specific domain and solve a specific problem by abstracting it through your language. If the abstract syntax tree of your language is more than 100 lines you may want to reassess the scope before it becomes too complicated
Please don't wait until you have a pristine design for your language before starting development since you won't achieve it without developing your language with whatever version of the design you have available	Work on the abstract syntax tree of your programming language first
Experience the abstract syntax tree of your language	Write a pretty printer of AST to print out the tokens as well as pretty printer that takes these tokens and prints out the AST
Print print print!!!!	Writing pretty printer of any component [from AST to code generation] acts as a GPS that

	navigates you to the next steps when you are stuck please write pretty printers as much as possible. One subtle outcome of writing pretty printers that it improves your understanding of what should be the next steps
Test test test!!!!	Test your compiler before your compiler tests your patience
Reuse	Existing projects are an indispensable source of inspiration [especially for CVN students]. pick up one or two projects —from previous semesters— that resemble the overall design of your language such as the code generation [for e.g. if your language and a language from previous project is generating code in python] and reuse these projects NOT FOR COPYING—though allowed but practically twice the effort—BUT FOR TAKING IDEAS. For e.g. I heavily consulted [5] for taking hints

Table 8: Lessons learned and suggestions

9 Conclusion and next steps

9.1 Conclusion

Software defined networking heralds a new approach of designing network applications. Uncovering the layers that once abstract the underpinnings of packet switched networks architecture reveals that packet switched networks are a collection of switching tables that could be arranged following the principles of relational algebra. This arrangement allows extending a set of declarative constructs —as we showed in NWQL— which not only simplify the task of writing network applications but also allow validating correctness of written programs which is one of the most important aspects of network application and is left to the discretion of the programmers in traditional SDN APIs. Furthermore NWQL significantly reduces the number of lines which programmers would have to write otherwise, we observed that programs written in NWQL take 80 to 84 percent less lines of code than those that are written using traditional SDN API such as Python or Java.

9.2 Next Steps

The scope of this project was not to produce a complete framework based on relation algebra for programming packet switched networks but to demonstrate that such a framework could be devised and shall be more powerful and consistent than the current state of the art raw APIs available to program packet switched network. Consequently a lot of powerful relational algebraic constructs are not addressed in this project that I believe should significantly improve the strengths of NWQL. These constructs include:

1. Joins which shall allow performing Cartesian products of multiple tables and run NWQL queries such as select on wider range of tuples

2. Data definition language: Currently NWQL assumes a very specific schema for the tables being queried or modified which severely limits the number of possible operations. Having constructs to first define an explicit schema of database being queried and/or modified before the relational algebraic operations should further enhance the possibilities of applications that could be written in NWQL

3. Logical and comparison operators

4. Subqueries

The author intends to expand NWQL to include these constructs as part on an independent course project if accepted by Prof. Edwards.

10 Bibliography

[1] "http://www.xorp.org/papers/xorp-nsdi.pdf," [Online].

[2] "http://archive.openflow.org/documents/openflow-wp-latest.pdf," [Online].

[3] "https://wiki.opendaylight.org/view/OpenDaylight_Controller:REST_Reference_and_Authentication," [Online].

[4] C. Date, Database in Depth: Relational Theory for Practitioners, 2005.

[5] "http://www.cs.columbia.edu/~sedwards/classes/2015/4115-fall/reports/LFLA.pdf," [Online].

11 Source code

11.1 ast.ml

```
(*A globally declared variable can take one of the four types of expressions as its values*)
type datatype =
         Int|
        String|
        Boolean|
        AVP
(*An expression in NwQL could be literals of one of the four data types, an identifier and an
assignment expression*)
type expr =
        IntLit of int |
        StrLit of string |
        BoolLit of bool |
        AVPList of  (string * string)  list|
        Id of string |
        LitAssign of string * expr
(*global variable declaration is a record of three items i.e. one of the four datatypes, the name
of the variable and its value *)
type gvdecl =
{
         vtype:datatype;
         vname:string;
         value:expr;
}
(*Following types of statements are possible in NWQL, a statement constitute an independently
compilable program in NWQL*)
type stmt=
         Expr of expr
         |
         Select of string list * string (*select bar from tablename*)
         |
         Insert of string * string list * expr list  (*insert into foo (column1,column2,column3)
(value1,value2,value3) *)
         |
         Delete of expr list* expr * string
(*A program is zero of more global variable declarations as well as zero or more statements *)
type program =
         gvdecl list * stmt list
(*************************************************************************
Name=ptydatatype
Description=Print string of the data type
Input= One of the four possible data types for a global variable
Output= String of the data type of global variable
*************************************************************************)
let ptydatatype=function
         Int -> "int"
         |
         String -> "string"
         |
         Boolean -> "boolean"
         |
         AVP -> "avp"
(*************************************************************************
Name=ptyexpr
```

```
Description=Print string of expression types
Input= One of the supported types of expressions
Output= String of the expression types
**************************************************************************)
let rec ptyexpr = function
(*matching pattern for integer type of literals*)
         IntLit(lit) -> string_of_int lit
         |
(*matching pattern for string literals*)
         StrLit(lit) -> lit
         |
(*matching pattern for bollean literals*)
         BoolLit(lit) -> string_of_bool lit
         |
(*matching pattern for identifiers*)
         Id(lit) -> lit
         |
(*matching patterns for the literals of type AVPLit*)
         AVPList(listofavps) ->
                  let rec ptyavp = function
                  []-> "here"
                  |
                  hd::tail -> (fst hd)  ^  (snd hd) ^ (ptyavp tail)
         in ptyavp listofavps
         |
(*matching pattern for the expression of type assignment*)
         LitAssign(name,value) -> name ^ " " ^ (ptyexpr value)
(*************************************************************************
Name=ptygvdecl
Description=Print string of gobal variable declarations
Input= variable declaration
Output= String of the variable declaration
**************************************************************************)
let ptygvdecl  globalvar =
         (ptydatatype globalvar.vtype) ^ (globalvar.vname) ^ (ptyexpr globalvar.value)
(*************************************************************************
Name=ptystmt
Description=Print string of one of the types of statements
Input= one of the statement types
Output= String of the statements
**************************************************************************)
let ptystmt = function
         Expr(expr) -> ptyexpr expr
         |
(*select*)
         Select(columnslist,tablename) ->
                  "select " ^ String.concat "," columnslist ^ " from " ^ tablename ^ "\n"
         |
         Insert(table,columnnamelist,valueslist) ->
         "insert into "^table^" "^ (String.concat "," columnnamelist)^" VALUES "^(String.concat ","
(List.map ptyexpr valueslist))
         |
         Delete(flownames,nodeid,tablename) ->
                           "delete " ^ (String.concat "," (List.map ptyexpr flownames)) ^ "in " ^
(ptyexpr nodeid) ^  " from " ^tablename^ "\n"
(*************************************************************************
Name=ptyprogram
Description= Print string of a program
Input= A nwql program
Output= String of a NWQL program
```

```
*********************************************************************)
let ptyprogram (idvarlist,stmtlist) =
         String.concat "" (List.map ptygvdecl idvarlist) ^
         String.concat "" (List.map ptystmt stmtlist)
```

11.2 parser.mly

```
%{ open Ast %}
/* beginning of declaration section */

%token ASTERISK,COLON,COMMA,LEFTPAREN,RIGHTPAREN,SEMICOLON,LEFTCURLY,RIGHTCURLY,ASSIGN,HASH
/*punctuation and separators*/

%token SELECT,INSERT,DELETE,INTO,FROM,VALUES,IN
/* keywords */

/*data types*/
%token INT,STRING,BOOLEAN,AVP

%token <int> INTLIT
/*integer literal */

%token <string> STRLIT
/*string literals */

%token <bool> BOOLLIT
/* boolean literals */

%token <string> ID
/* identifier */

%token <string> TNAME

%token <string*string>AVPLIT

%token EOF
/*End of file symbol */

%start program
/*start symbol is a program */

%type <Ast.program> program
%%
/*End of declaration section */

/*Beginning of the rules section */

vdecl:
datatype ID ASSIGN expr{{vtype=$1;vname=$2;value=$4}}

vdecllist:
HASH{[]}
|
vdecllist vdecl SEMICOLON {$2::$1}

datatype:
```

```
INT{Int}
|
STRING{String}
|
BOOLEAN{Boolean}
|
AVP{AVP}

program:
vdecllist HASH statements {
          (
          if (List.length $1>0) && (List.length $3>0)
          then
                    List.rev $1, List.rev $3
          else
                    if(List.length $1<1) && (List.length $3>0)
                    then
                              [{vtype=String;vname="dummy";value=StrLit("\"dummy\"")}],List.rev $3


                    else
                              raise(Failure("A program should have at least one statement"))
          )
          }

statements:
/* a single statement */
stmt {[$1]}
|
statements SEMICOLON stmt {($3::$1)}
/* or a list of statements where statements are separated by colons */

stmt:
expr{Expr($1)}
/*An statement is either an expression ,see expr */
|
SELECT columnnames FROM TNAME{Select($2,$4)}
|
INSERT INTO TNAME LEFTPAREN columnnames RIGHTPAREN VALUES LEFTPAREN expressions RIGHTPAREN
{Insert($3,$5,$9)}
|
DELETE expressions IN expr FROM TNAME{Delete($2,$4,$6)}
/* or a delete statement */

columnnames:
ID {[$1]}
|
ID COMMA columnnames {($1::$3)}
/*a list of columnname=value used in insert statement*/

avplist:
AVPLIT {[$1]}
|
AVPLIT COMMA avplist {($1::$3)}

expr:
LEFTCURLY avplist RIGHTCURLY {AVPList($2)}
|
INTLIT{IntLit($1)}
|
```

```
STRLIT{StrLit($1)}
|
BOOLLIT{BoolLit($1)}
|
ID ASSIGN expr{LitAssign($1,$3)}
|
ID{Id($1)}

/* expressions */

expressions:
expr{[$1]}
|
expr COMMA expressions {$1::$3}
```

11.3 scanner.mll

```
{open Parser
}
(*declaring some common regular expressions*)

(*regular expression for table name in select, insert and delete statements*)
let tname=(''''h''t''t''p')[^'\n']*

(*identifier is used in multiple regular expression*)
let identifier =['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']+

(*string literal is used in multiple regular expressions *)
let strlit ='"'(([^ '"'])*)'"'

(*this regular expression matches any of the whitespace characters and the associated semantic
action calls the token rule back. *)
rule token = parse
[' ' '\t' '\r' '\n'] {token lexbuf}
|
"/*" {comment lexbuf}
|
"select"{SELECT} | "from"{FROM} | "insert"{INSERT} | "into"{INTO} | "values"{VALUES}
|"delete"{DELETE}|"in"{IN}
|
'#'{HASH}
|
"int" {INT} | "string" {STRING} | "boolean"{BOOLEAN}|"avp"{AVP}
|
['0'-'9']+ as intlit {INTLIT(int_of_string intlit)}
|
"true"|"false" as boollit {BOOLLIT(bool_of_string boollit)}
|
tname as tablename {TNAME(tablename)}
|
identifier  as ident {ID(ident)}
|
strlit as stringliteral {STRLIT(stringliteral)}
|
'('(strlit as id)':'(strlit as lit)')'{AVPLIT(id,lit)}
(*Regular expression punctuation and separators*)
|'*'{ASTERISK}|','{COMMA}|'('{LEFTPAREN}|')'{RIGHTPAREN}|';'{SEMICOLON}
```

```
|'{'{LEFTCURLY}|'}'{RIGHTCURLY}|'='{ASSIGN}
|
eof {EOF}
(*EOF*)
|
_ as char {raise (Failure("Illegal character " ^ Char.escaped char))}

and
comment= parse
"*/" {token lexbuf}
|
_ {comment lexbuf}
```

11.4 translate_env.ml

```
open Ast

module NameMap=Map.Make(String)

(*The runtime environment is just a map of global variables*)
type env= gvdecl NameMap.t

(*this method will check if a variable already exists in the symbol table*)
let gvar_exist vname env=NameMap.mem vname env

(*find and return variable declaration *)
let gvar_value vname env =NameMap.find vname env
```

11.5 past.ml

```
(*This file is a python abstract syntax tree*)
(*The goal of python compiler/translator will be to translate an ast generated in NwQL to Python*)

(* Similar to a NwQL program the translated python also begins with global variable declarations of
zero or more variables*)
(*There are four data types in NwQL integers, strings, boolean and an aggregate data type called AVP
and the shall be translated in to
int, string, boolean and tuples respectively in NwQL*)

type pdatatype =
        P_int|
        P_string|
        P_boolean|
        P_avp
(*An expression in NwQL could be literals of one of the four data types, an identifier and an
assignment expression*)
```

```
type pexpr =
        P_intLit of int |
        P_strLit of string |
        P_boolLit of bool |
        P_avpList of  (string * string)  list|
        P_id of string |
        P_litAssign of string * pexpr

type pgvdecl =
{
        pvtype:pdatatype;
        pvname:string;
        pvalue:pexpr;
}

(*a NwQL program consist of a list of global declarations and a list of statements*)

type pstmt=
        P_expr of pexpr
         |
         Pselect of string list * string
         |
         Pinsert of string * string list * pexpr list
         |
         Pdelete of pexpr list * pexpr * string

(*A program is zero of more global variable declarations as well as zero or more statements *)
type pprogram =
        pgvdecl list * pstmt list

let rec ptypexpr = function
(*matching pattern for integer type of literals*)
        P_intLit(lit) -> string_of_int lit
        |
(*matching pattern for string literals*)
        P_strLit(lit) -> lit
        |
(*matching pattern for bollean literals*)
        P_boolLit(lit) -> string_of_bool lit
        |
(*matching pattern for identifiers*)
        P_id(lit) -> lit
        |
(*matching patterns for the literals of type AVPLit*)
        P_avpList(listofavps) ->
                let rec ptyavp = function
                []-> ""
                |
                hd::tail -> (fst hd)^(snd hd)^ (ptyavp tail)
        in ptyavp listofavps
        |
(*matching pattern for the expression of type assignment*)
        P_litAssign(name,value) -> name ^ (ptypexpr value)
```

11.6 translate.ml

```
open Ast
open Past
open Translate_env
```

```
open Typechecker

(*This file translates a NwQL syntax tree to a python syntax tree and in the process of this
conversion it also performs semantic analysis*)

(*********************************************************************
Name=translate_datatype

Description= This method is used to translate datatype to pdatatype

Input= A datatype from AST

Output= A past equivalent datatype*)

let translate_datatype= function
         Int -> P_int
         |
         String -> P_string
         |
         Boolean -> P_boolean
         |
         AVP -> P_avp

(*********************************************************************
Name=translate_expr

Description= This method is used to translate Expr to P_expr

Input=Symbol table and an expression of one of the six types

Output=A tuple based on the updated environment and the translated
P_expr. Note the envrionment is only updated when an expression of
type assignment is being matched, otherwise the received environment
as an input is returned as the output
*********************************************************************)

let rec translate_expr env expr= match expr with
(*translating expression of type IntLit to P_intLit*)
IntLit(i) -> (P_intLit(i),env)
|
(*translating expression of type StrLit to P_strLit*)
StrLit(s) -> (P_strLit(s),env)
|
(*translating expression of type BoolLit to P_boolLit*)
BoolLit(b) -> (P_boolLit(b),env)
|
(*translating expression of type AVPList to P_avpList*)
AVPList(l) ->
         (P_avpList(l),env)
|
(*translating expression of type AVPList to P_avpList*)
Id(id) -> (P_id(id),env)
|
(*translating expression of type LitAssign to P_litAssign*)
LitAssign(name,value) as expr->
        (*check if the assignment expression is consistent with its type*)
        (*check assignment will return the updated environment*)
        let updatedenv= (check_assigexpr env expr)
        and
        (*here we call the translate_expr again to translate the literal to p_literal and catch the
```

```
translated object in a variable*)
        (val_to_pval,_) = (translate_expr env value) in
        (*here we return passignment with the updated environment*)
        ((P_litAssign(name, val_to_pval)),updatedenv)

(*************************************************************************
Name=translate_gvdecl

Description= translate global variable declarations
to python global varible declarations

Input= A symbol tablex-which would be typically empty at
at the time of declaration-and a global variable declaration
to be translated

Output= A tuple based on translated global variable and update symbol
table
*************************************************************************)

let translate_gvdecl env gvar=
(*first check if the variable exist*)
        if gvar_exist gvar.vname env then
                (*if the variable exists then raise an exception that the variable is already
defined*)
                raise(Failure("Variable " ^ gvar.vname ^" already defined"))
                  (*if the variable does not exist check if the value of the variable is consistent
with its type*)
        else
                if check_gvdecl_consistency gvar then
                (*if the value is consistent then update the symbol table and call the translate
expression method since the assigned literal is an expression*)
                        let updenv=NameMap.add  gvar.vname gvar env in
                  (*translate expression will return the translated p_expression*)
                            let (pexp,_)=translate_expr updenv gvar.value
                  (*use the returned p_expression as well as the translate_datatype method to
create pvdecl type object*)
                            in let pgvdecl_trans={pvtype=(translate_datatype gvar.vtype);
pvname=gvar.vname;pvalue=pexp}
                  (*return pvdecl type object and the update symbol table*)
                            in (pgvdecl_trans,updenv)
                else
                (*if the value is not consistent with its type then fail with the exception that
the value is incompatible*)
                         raise(Failure("Incompatible value for " ^ gvar.vname))

(*********************************************************************************************
Name=translate_stmt

Description= Translate statements to P_statements

Input = A type of statement

Output= Equivalent type of p_statement
*************************************************************************)

(*translating statement of type expressions*)
let translate_stmt env stmts =
         match stmts  with  Expr(expr) ->
                  (*translate_expr returns P_expr and an update environment, out of which we create
P_expr*)
```

```
                    let expr,env=translate_expr env expr in (P_expr(expr),env)
          |
          Select(col_list,tab_name) ->
                    if ((check_colnames col_list) = true) && ((check_tablename tab_name) = true)
                              then (Pselect(col_list,tab_name),env)
                 else
                         raise(Failure("Illegal table name " ^ tab_name))
          |
          Insert(tab_name,col_list,exprs) ->
                 (*We check if tablename is correct, mandatory columnnames are present, the optional
columnnames are correct and no duplicate columnnames are there*)
                 if ((check_tablename tab_name)=true)&&((check_insert_colnames
col_list)=true)&&((check_colnames col_list)=true)&&((find_dup col_list)=false)
                         then
                                 (*now check if number of columnnames is greater than the number of
expressions*)
                                 if (List.length col_list) > (List.length exprs)
                                         then
                                                 (*if number of columnnames is greater than the
number of expressions then raise an exception*)
                                                 raise((Failure("No value for column " ^ (List.nth
col_list ((List.length exprs))))))
                                 (*check if the number of columnnames are less than the the number
of expressions*)
                                 else if (List.length col_list) < (List.length exprs)
                                         then
                                                 (*if number of columnnames are less than number of
expressions then raise an exception*)
                                                 raise ((Failure("The expression " ^ ptyexpr
(List.nth exprs ((List.length col_list) ))^ " does not belong to any column")))
                                      else
                                         (*If number of columnnames are equal to number of
expressions then*)
                                                         (*we combine the list of columnnames and
list of expressions in to a list of tuples*)
                                         let col_exprs_combine = List.combine col_list exprs in
                                                (*In the combined list we add environment to each
tuples since check_value takes a tuple of env,column and value*)
                                                 let rec env_col_exprs_combine= function [] -> []
                                                 |
                                                 hd::tail -> (env,fst hd, snd hd) ::
env_col_exprs_combine tail in
                                                         let combined_list= env_col_exprs_combine
col_exprs_combine in
                                                                           (*Apply list.map to each
of the elements of the combined list*)
                                                                           (*We can simply ignore
the resulting list since if the semantic check are not passed
                                                                           then an exception will be
raised by check_value*)
                                                                           let _=List.map
check_value combined_list in
                                                                                    (*List.map
concluding successfully means that check_value didnt cause exception*)
                                                                                    (*first we are
going to translate a list of Expr to Pexpr*)
                                                                                    (*since env is
the same we create a partial application of a function*)
                                                                                    let
expr_map=translate_expr env in
```

```
                                                                                            (*this partially
applied function will take an expression*)
                                                                                            let p_expr_list
= List.map expr_map exprs in
                                                                                            (*apply the
partial function to a list of expression that would retrun a list of
                                                                                            p_expr and
environment*)
                                                                                            let
p_exprs=List.map fst p_expr_list in
                                                                                            (*since Pinsert
only requires pexpressions we apply first to be above returned list*)
                                                                                            (*creating and
returning pinsert*)

          (Pinsert(tab_name,col_list,p_exprs),env)
                    (*if tablename is not valid an exception will be raised*)
                    else
                              raise(Failure("Illegal table name " ^ tab_name))
          |
          (*Translating Delete from ast to P_delete in past*)
          Delete(fnamelist,nodename,tab_name) ->
                    (*first we check if table name correct and if there any duplicate expressions*)
                    if((check_tablename tab_name)=true) && ((find_dup_expr fnamelist)=false)
                              then
                                        (*if yes then we serialise flow identifier name, columnname
and environment so that we can pass it to typechecker*)
                                        let join elem1 elem2 elem3 = (elem1,elem2,elem3) in
                                                  let par_join expr=join env "name" expr in
                                                            let env_cname_exp_list=List.map par_join
fnamelist in
                                                                      (*if the exrepssions are not of
type Id or StrLit matching the name regexp then exceptions will be raised*)
                                                                      let _=List.map check_value
env_cname_exp_list and
                                                                      _ = check_value
(env,"node",nodename) in
                                                                      (*If the above List.map is
successful then semantic analysis is successful so far*)
                                                                      (*now translating the expression to
p_expr*)
                                                                                let
expr_map=translate_expr env in
                                                                                          let
p_expr_list=List.map expr_map fnamelist in
                                                                                                    let
p_exprs= List.map fst p_expr_list in
                                                                                                    let
p_node= fst (expr_map nodename) in

                    else
                              raise(Failure("Illegal table name " ^ tab_name))




(************************************************************************
Name=translate_program

Description= Translate a program to p_program
```

```

Input= A tuple based on a list of global variable declarations
and a list of statements

Output= A list of translated p_global variable declarations and
a list of p_statements
***********************************************************************)

let translate_program (gvdecllist,stmtlist) =

         (**********************************************************************
         Name=trav_stmts

         Description=Translate statements to P_statements

         Input= A program environment and a list of statements

         Output = A list of P_statements
         **********************************************************************)
         let rec trav_stmts env stmts=
         match stmts with
         hd::tail ->
                  (*Translate head of the list and store result*)
                  let result = translate_stmt env hd in
                  (*Append the p_statement to a list whose tail calls traverse statements with the
updated environment receved in the result*)
                           (fst result) :: trav_stmts (snd result) tail
         |
         [] -> []

         (**********************************************************************
         Name=get_pgvdecl_updtbl

         Description=Translate gvdecl to pgvdecl

         Input= An environment and a list of global declarations

         Output= A list of pgvdecl
         **********************************************************************)

         and get_pgvdecl_updtbl env gvdecl = match gvdecl with
         (*Translate head of the list and store result*)
         hd::tail ->
                  let result = translate_gvdecl env hd in
         (*Append the result including both pgvdecl and environment to a list whose tail calls
traverse pgvdecl with the updated environment*)
                           result :: get_pgvdecl_updtbl (snd result) tail
         |
         [] -> []

                  in
                  (*Initialise an empty symbol tablr*)
                  let init_symtab = NameMap.empty in
                           (*call the  get_pgvdecl_updtbl method*)
                           let comb_list = get_pgvdecl_updtbl init_symtab gvdecllist in
                           (*separate the pgvdecl into a list, from a list of (pgvdecl,env)*)
                                    let pgvdecl_list =List.map fst comb_list
                           (*from combined list take out the most recent runtime environment*)
                           and upd_sym_tbl =  snd (List.hd (List.rev comb_list))
                           in
```

```
                                        (*Call the trav_stmts methods with the updated runtime and
statement list
                                        and append the resulting list as a second member of a tuple
                                        while first being the pgvdecl list*)
                                        pgvdecl_list,trav_stmts upd_sym_tbl stmtlist
```

11.7 typechecker.ml

```
open Ast
open Translate_env
module Regex = Re2.Regex


(**************************************************************************
Name=check_gvdecl_consistency

Description= Checks if the value of a global variable is consistent
with its type

Input= gvdel record type

Output= true/false
**************************************************************************)

let check_gvdecl_consistency gvdecl= match gvdecl.vtype with
         Int -> let checkvalue value= match value with
                  IntLit(_) -> true
                  |
                  _ -> false
                  in checkvalue gvdecl.value
         |
         String -> let checkvalue value= match value with
                  StrLit(_) -> true
                  |
                  _ -> false
                  in checkvalue gvdecl.value
         |

         Boolean -> let checkvalue value= match value with
                  BoolLit(_) -> true
                  |
                  _ -> false
                  in checkvalue gvdecl.value
         |

         AVP -> let checkvalue value= match value with
                  AVPList(_) -> true
         |
                  _ -> false
                  in checkvalue gvdecl.value

(**************************************************************************
Name = check_assigexpr

Description= This method is used for typechecking assignment expressions

```

```
Input = An environment and an expression

Output = Updated environment
*************************************************************************)
let check_assigexpr env expr =
         (*Match only the Literal assignment*)
         match expr with LitAssign(key,value) ->
                  (*First check if the name exist*)
                  if gvar_exist key env then
                           (*If it exists then retrieve the object then *)
                           let gvdeclobj=NameMap.find key env in
                                    (*copy the type of the value *)
                                    match value with IntLit(_) | StrLit(_) | BoolLit(_) |
AVPList(_) as literal ->
                                    (*Now create a new VDECL object based on the new value for an
existing global variable*)
                                             let gvdeclupdobj =
{vtype=gvdeclobj.vtype;vname=gvdeclobj.vname;value=literal} in
                                                      (*check if the new object is consistent i.e.
the value is of the compatible data type*)
                                                      if check_gvdecl_consistency  gvdeclupdobj
then
                                                               (*if compatability check passes
update and return the updated symbol table*)
                                                               let updenv=NameMap.add key
gvdeclupdobj env in updenv
                                                               (*else clause is just to complete
the if/else statement however the execution will never come to this point
                                                               since if there is no compatability
check_gvdecl_consistency will raise an exception*)
                                                      else
                                                               raise(Failure("Incompatible value
for " ^ gvdeclobj.vname))
                                    |
                                    Id(identifier) ->
                                             (*check now if identifier exist *)
                                             if gvar_exist identifier env then
                                                      (*check if the identifier exist*)
                                                      let  gvdeclobj2=NameMap.find identifier env
in
                                                               (*create a gvdecl object with the
name of existing identifer and the value given in the assignment expression*)
                                                               let
gvdeclupdobj={vtype=gvdeclobj.vtype;vname=gvdeclobj.vname;value=gvdeclobj2.value} in
                                                                        (*check if the new value
will be consistent with the datatype of the existing identifier*)
                                                                        if
check_gvdecl_consistency  gvdeclupdobj then
                                                                                 (*if yes then
update the environment*)
                                                                                 let
updenv=NameMap.add key gvdeclupdobj env in updenv
                                                                        else
                                                                                 (*raise
exception if the value is inconsistent*)

         raise(Failure("Incompatible value for " ^ gvdeclobj.vname))

                                             else     (*raise exception if given identifer does
not exist*)
```

```
                                                raise(Failure("Variable " ^ identifier ^
"does not exist"))

                                |
                                (*Nested assignment not allowed*)
                                LitAssign(key2,value2)  ->
                                        raise(Failure("Invalid assignment " ^key ^"=" ^key2))

                else    (*If the variable does not exist raise an exception*)
                        raise(Failure("Variable " ^ key ^ "does not exist"))

        |

        _  -> raise(Failure("Invalid assignment type"))


(**************************************************************************
Name = check_insert_colnames

Description= This method is used to check if columnnames for insert
statement are valid

Input = A list of columnnames

Output = True if columnnames are valid else exceptions are raised
**************************************************************************)
let check_insert_colnames col_names=
        if List.mem "name" col_names=true then
                if List.mem "node" col_names=true then
                        if List.mem "ingressPort" col_names=true then
                                if List.mem "priority" col_names=true then
                                        if List.mem "etherType" col_names=true then
                                                if List.mem "actions" col_names=true then
                                                        true
                                                else
                                                        raise(Failure("Mandatory column
action missing"))
                                        else
                                                raise(Failure("Mandatory column ether type
missing"))
                                else
                                        raise(Failure("Mandatory column priority missing"))
                        else
                                raise(Failure("Mandatory column ingress port missing"))
                else
                        raise(Failure("Mandatory column node missing"))
        else
                raise(Failure("Mandatory column name missing"))

(**************************************************************************
Name = find_dup

Description= This method is used to check if there are any duplicate
columns

Input = A list of columnnames

Output = False if columnnames are non duplicate else exception is raised
**************************************************************************)

```

```
let rec find_dup= function
[] ->
        false
|
hd::tl ->
        if List.mem hd tl then
                raise(Failure("Column " ^hd ^ " appears multiple times, duplicates are not
allowed in NwQL"))
        else
                find_dup tl


(************************************************************************
Name = find_dup_expr

Description= This method is used to check if there are any duplicate
expressions

Input = A list of expressions

Output = False if expressions are non duplicate else exception is raised
************************************************************************)
let rec find_dup_expr= function
[] ->
       false
|
hd::tl ->
       if List.mem hd tl then
               raise(Failure("Flow name " ^(ptyexpr hd) ^ " appears multiple times, duplicate
flownames are not allowed in DELETE statements"))
       else
               find_dup_expr tl
(************************************************************************
Name = check_colnames

Description= This method is used to check if columnnames are valid

Input = A list of columnnames

Output = True if columnnames are valid else exceptions are raised
************************************************************************)

let check_colnames col_names =

        let val_col_names =
                [
                "actions";"etherType";"hardTimeout";
                "idleTimeout";"ingressPort";"name";"node";
                "nwDst";"nwSrc";"priority";"protocol";
                "tpDst";"tpSrc";"vlanId";"vlanPriority"
                ]
                in
                        let rec cmp_colNames list_of_cnames=
                                match list_of_cnames with
                                (*This is only possible when the
                                entire list is traversed of an empty
                                list is given as an input*)
                                []->true
                                |
                                hd::tl ->
```

```
                                                  if List.mem hd val_col_names then
                                                            cmp_colNames tl
                                                  else
                                                            raise(Failure("Invalid Column name " ^ hd))

                              in cmp_colNames col_names
(*************************************************************************
Name = check_tablename

Description= This method is used to check if provided tablename is valid

Input = Tablename

Output = True if tablename is valid else False
*************************************************************************)
let check_tablename tablename =
          let tableRegex=Regex.create_exn "\\'http:\\/\\/\\b([0-9]{1,3}\\.[0-9]{1,3}\\.[0-
9]{1,3}\\.[0-9]{1,3})\\b:\\b(655[0-3][0-5]|655[0-2][0-9]|6[0-5][0-4][0-9][0-9]|[1-5][0-9][0-9][0-
9][0-9]|[1-9][0-9][0-9][0-9]|[1-9][0-9][0-9]|[1-9][0-9]|[0-
9])\\b\\/controller\\/nb\\/v2\\/flowprogrammer\\/default\\',auth=\\('[_a-zA-Z]+','[_a-zA-Z]+'\\)"
in
          Regex.matches tableRegex tablename
(*************************************************************************
Name = check_identifier

Description= This method is used to check values of columns are
of compatible expression types when values are global variables

Input = variablename,environment,datatype and columnname

Output = True if is consistent else the mutually recursive method will
cause an exception to happen
*************************************************************************)
let rec check_identifier vname env datatype cname =
          if gvar_exist vname env then
                    let literal=gvar_value vname env in
                              match literal with datatype -> check_value (env,cname,literal.value)
          else  raise(Failure("Variable " ^vname^ " does not exist"))

and

(*************************************************************************
Name = check_value

Description= This method is used to check values of columns are
of compatible expression types when values are literals of expression type

Input = environment,columnname and expression

Output = True if is consistent else this method will
cause an exception to happen
*************************************************************************)

check_value (env,colname,expr) =
          (*some regular expressions used throughout*)
         let ipaddrregex =Regex.create_exn "\"\\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\\b\""
          and
        portaddrregex=Regex.create_exn "\\b(655[0-3][0-5]|655[0-2][0-9]|6[0-5][0-4][0-9][0-9]|[1-
```

```
5][0-9][0-9][0-9][0-9]|[1-9][0-9][0-9][0-9]|[1-9][0-9][0-9]|[1-9][0            -9]|[0-9])\\b"
        and
        vlanidregex= Regex.create_exn "\\b(40[0-8][0-9]|409[0-5]|3[0-9][0-9][0-9]|2[0-9][0-9][0-
9]|1[0-9][0-9][0-9]|[1-9][0-9][0-9]|[1-9][0-9]|[0-9])\\b"
        and
        vlanprioregex=Regex.create_exn "0|1|2|3|4|5|6|7"
        and
        identifierregex=Regex.create_exn "[a-zA-Z][a-zA-Z0-9_]+"
        and
        nodeidregex=Regex.create_exn ".+"
        in
         (*matching of different column names start*)
        match colname with "actions" ->
                (*a separate method for performing semantic analysis of column name actions*)
                let check_actions_value envr value =
                        (*Formally actions has literals of expression type AVP*)
                        match value with AVPList(avp) ->
                                (*A subset of AVP literals may qualify for actions column*)
                                let avpmatch=function [("\"DROP\"","\"DROP\"")] -> true
                                        |
                                        [("\"SET_VLAN_ID\"",vlanid)] ->
                                                if Regex.matches vlanidregex vlanid = true then
true
                                                else raise(Failure("Bad value for SET_VLAN_ID"))
                                        |
                                        [("\"SET_VLAN_PCP\"",priority)] ->
                                                if Regex.matches vlanprioregex priority=true then
true
                                                else raise (Failure("Bad value for SET_VLAN_PCP"))
                                        |
                                        [("\"SET_NW_DST\"",ipdst)] ->
                                                if Regex.matches ipaddrregex ipdst=true then true
                                                else raise (Failure("Bad value for SET_NW_DST"))
                                        |
                                        [("\"SET_NW_SRC\"",ipsrc)] ->
                                                if Regex.matches ipaddrregex ipsrc=true then true
                                                        else raise (Failure("Bad value for
SET_NW_SRC"))
                                        |
                                        [("\"SET_TP_SRC\"",portsrc)] ->
                                                if Regex.matches portaddrregex portsrc=true then
true
                                                else raise (Failure("Bad value for SET_TP_SRC"))
                                        |
                                        [("\"SET_TP_DST\"",portdst)] ->
                                                if Regex.matches portaddrregex portdst=true then
true
                                                else raise (Failure("Bad value for SET_TP_DST"))
                                        |
                                        (*Everything else is an exception*)
                                        _  -> raise(Failure("Value of actions is out of range")) in
avpmatch avp
                            |
                        (*besides AVPList the value may also be an identifier representing datatype
of AVP*)
                        Id(identifier) ->
                                check_identifier identifier env "AVP" "actions"
                            |
                            _ -> raise(Failure("Illegal literal type for column actions"))
                  in check_actions_value env expr
```

```
          |

          "name" ->
                  let check_name_value envr value=
                          match value with StrLit(lit) ->
                          if Regex.matches identifierregex lit=true then true
                              else raise(Failure("Bad value for name"))
                          |
                          Id(identifier) ->
                                  check_identifier identifier env "String" "name"
                          |
                          _ -> raise(Failure("Illegal literal type for column name"))
                  in check_name_value env expr
          |
          "node" ->
                let check_node_value envr value=
                       match value with StrLit(lit) ->
                       if Regex.matches nodeidregex lit=true then true
                       else raise(Failure("Bad value for nodeid"))
                           |
                           Id(identifier) ->
                                   check_identifier identifier env "String" "node"
                           |
                           _ -> raise(Failure("Illegal literal type for column node"))
                  in check_node_value env expr
          |
          "ingressPort" ->
                let check_ingressp_value envr value=
                       (*Ingress port could theoritically be any number*)
                       match value with IntLit(lit) -> true
                       |
                       Id(identifier) ->
                                   check_identifier identifier env "Int" "ingressPort"
                           |
                           _ -> raise(Failure("Illegal literal type for column ingressPort"))
                  in check_ingressp_value env expr
          |
          "vlanId" ->
          let check_vlanid_value envr value=
                       match value with IntLit(lit) ->
                               if (lit >=0 && lit<4096)
                                       then true
                                       else raise(Failure("Bad value for vlanId"))
                           |
                       Id(identifier) ->
                                   check_identifier identifier env "Int" "vlanId"
                           |
                           _ -> raise(Failure("Illegal literal type for column VlanId"))
                  in check_vlanid_value env expr
          |
          "priority" ->
                  let check_flow_prio envr value=
                       match value with IntLit(lit) -> true
                       |
                       Id(identifier) ->
                                   check_identifier identifier env "Int" "priority"
                           |
                           _ -> raise(Failure("Illegal literal type for column priority"))
                  in check_flow_prio env expr
          |
```

```
          "idleTimeout" ->
          let check_idle_to envr value=
                         match value with IntLit(lit) -> true
                             |
                         Id(identifier) ->
                                      check_identifier identifier env "Int" "idleTimeout"
                             |
                             _ -> raise(Failure("Illegal literal type for column idle time out"))
                   in check_idle_to env expr
          |
          "hardTimeout" ->
          let check_hard_to envr value=
                         match value with IntLit(lit) -> true
                             |
                         Id(identifier) ->
                                      check_identifier identifier env "Int" "hardTimeout"
                             |
                             _ -> raise(Failure("Illegal literal type for column hard time out"))
                   in check_hard_to env expr
          |
          "vlanPriority" ->
          let check_vlanprio_value envr value=
                   match value with IntLit(lit) ->
                   if (lit >=0 && lit<8)
                             then true
                   else
                             raise(Failure("Bad value for vlan Priority"))
                   |
                         Id(identifier) ->
                                      check_identifier identifier env "Int" "vlanPriority"
                             |
                             _ -> raise(Failure("Illegal literal type for Vlan priority"))
                   in check_vlanprio_value env expr
          |
          "etherType" ->
          let check_ethertype_value envr value=
                   match value with StrLit(lit) ->
                         if (lit="\"0x800\"" || lit= "\"0x8100\"")
                                      then true
                         else
                                  raise(Failure("Bad value for vlan ether type"))
                             |
                         Id(identifier) ->
                                      check_identifier identifier env "String" "etherType"
                             |
                             _ -> raise(Failure("Illegal literal type for etherType"))
                   in check_ethertype_value env expr
          |"protocol" ->
          let check_prototype_value envr value=
                   match value with StrLit(lit) ->
                         if (lit="\"tcp\"" || lit= "\"udp\"")
                                 then true
                         else
                                  raise(Failure("Bad value for protocol type"))
                             |
                         Id(identifier) ->
                                      check_identifier identifier env "String" "protocol"
                             |
                             _ -> raise(Failure("Illegal literal type for protocol"))
                   in check_prototype_value env expr
```

```
            |"tpSrc" ->
            let check_tpsrc_value envr value=
                    match value with IntLit(lit) ->
                         if (lit>=0 && lit< 65536)
                                then true
                         else
                                raise(Failure("Bad value for Source transport port"))
                    |
                         Id(identifier) ->
                                     check_identifier identifier env "Int" "tpSrc"
                             |
                             _ -> raise(Failure("Illegal literal type for tpSrc"))
                    in check_tpsrc_value env expr
            |"tpDst" ->
            let check_tpdst_value envr value=
                    match value with IntLit(lit) ->
                         if (lit>=0 && lit< 65536)
                                then true
                         else
                                raise(Failure("Bad value for Destination transport port"))
                    |

                             Id(identifier) ->
                                     check_identifier identifier env "Int" "tpDst"
                             |
                             _ -> raise(Failure("Illegal literal type for tpDst"))
                    in check_tpdst_value env expr
            |"nwSrc" ->
            let check_nwsrc_value env value=
                    match value with StrLit(lit) ->
                         if Regex.matches ipaddrregex lit= true then true
                         else
                             raise(Failure("Bad value for Source IP address"))
                             |
                             Id(identifier) ->
                                     check_identifier identifier env "String" "nwSrc"
                             |
                             _ -> raise(Failure("Illegal literal type for nwSrc"))
                    in check_nwsrc_value env expr
            |"nwDst" ->
            let check_nwdst_value env value=
                    match value with StrLit(lit) ->
                         if Regex.matches ipaddrregex lit= true then true
                         else
                             raise(Failure("Bad value for Destination transport IP address"))
                             |
                             Id(identifier) ->
                                     check_identifier identifier env "String" "nwDst"
                             |
                             _ -> raise(Failure("Illegal literal type for nwDst"))
                    in check_nwdst_value env expr
            |_ as invalid ->
                    raise(Failure("Invalid column name "^invalid))
```

11.8 compile.ml

```
open Past
(*Code generation module*)

(*Generating code for datatype*)
(*There is no datatype in python so no code will be generated*)
let string_of_pdatatype = function
         P_int -> ""
         |
        P_string -> ""
         |
        P_boolean -> ""
         |
        P_avp-> ""

(************************************************************************
Name=removequotes
Description=This method is used to remove redundant quotes
in string and AVP type of expressions
Input=An expression
Output=An expression with redundant quotes removed if the expression
ins of type string or AVP
************************************************************************)
let removequotes pexpr =
match pexpr with P_strLit(str) ->
         if (String.get str 0) = '"' then
                  P_strLit(String.sub str 1 ((String.length str) -2))
         else
                  P_strLit(str)
|
P_avpList(avplist) ->
         let removeavpquotes=function(key,value) ->
         if (String.get key 0) = '"'
                  then
                           (String.sub key 1 ((String.length key) -2),String.sub value 1
((String.length value) -2))
                  else
                           (key,value)
         in
         let newlist = List.map removeavpquotes avplist
         in
         P_avpList(newlist)
|
_ as other -> other

(************************************************************************
Name=trans_link link
Description=This method is useful for tranforming
the generic hyperlink for the specific hyperlink
required for the statement of type Insert
Input=A string containing hyperlink
Outpput=A string containing hyperlink required for Insert statement
************************************************************************)
let trans_link link =
         (*Find index of , since in a table ',' first apprears to separate authentication
credentials from the hyperlink*)
         let index=String.index link ','
         (*length of the link*)
         and length=String.length link
         in
         (*Return a tuple whose first member is hyperlink and whose second member is
```

```
authentication info*)
         (String.sub link 0 index,String.sub link (index+1) (length-(index+1)))

(*************************************************************************
Name=indexof
Description=This method is useful for getting index of a particular
element in a list
Input=A list and an element
Output=Index of the element
*************************************************************************)
let rec indexof clist elem=
         match clist with [] -> raise(Failure("No columns mentioned"))
         |
         hd::tail ->
                  if hd=elem
                           then 0
                  else
                           1+ indexof tail elem

(*************************************************************************
Name=string_of_pexpr
Description=This method is used to generate python code for expressions
Input= Expression
Output= String representing the respective python code
*************************************************************************)
let rec string_of_pexpr= function
         (*An integer literal in NwQL will simply be compiled in to integer literal in python *)
        P_intLit(lit) -> string_of_int lit
         |
         (*A string literal in NwQL will simply be complied in to string literal in python*)
        P_strLit(lit) ->  lit
         |
         (*A boolean literal in NwQL will simply be compiled in to boolean literal in python with
first letter of the literal capitalised*)
         P_boolLit(lit) -> String.capitalize(string_of_bool lit)
         |
         P_avpList(lit) ->
                  let rec translate_AVPList = function
                  [] -> ""
                  |
                  hd::tail -> (fst hd) ^  "," ^  (snd hd) ^  "," ^  (translate_AVPList tail) in
                           let dict =translate_AVPList lit in
                           let endpos= String.rindex dict ','
                           in "[" ^ (String.sub dict 0 endpos) ^ "]"
         |
         (*An identifier in NwQL is simply compiled in to a python identifier*)
         P_id(id) -> id
         |
         (*Literal assignment*)
         P_litAssign(id,expr) -> id ^ "=" ^ string_of_pexpr expr

(*************************************************************************
Name=string_of_insert_pexpr
Description=This method is used to generate python code for expressions
used as AVPs in insert statements
Input=Expression
Output= String representing the respective python code
*************************************************************************)
let rec string_of_insert_pexpr= function
        P_intLit(lit) -> "\"" ^ (string_of_int lit) ^ "\""
```

```
        |
        (*A string literal in NwQL will simply be complied in to string literal in python*)
        P_strLit(lit) ->   "\"" ^ lit ^ "\""
        |
        (*A boolean literal in NwQL will simply be compiled in to boolean literal in python with
first letter of the literal capitalised*)
        P_boolLit(lit) -> String.capitalize(string_of_bool lit)
        |
        (*An identifier in NwQL is simply compiled in to a python identifier*)
        P_id(id) -> id
        |
         _ -> raise(Failure("Unsupport expression in the statement"))

(************************************************************************
Name=string_of_action
Description=This method is used to generate python code for action column
used in insert statements
Input= Expression of type AVP
Output= String representing the respective python code
************************************************************************)

let string_of_action =function P_avpList(lit) ->
         let rec string_of_avp = function [] -> ""
         |
         hd::tl ->
                  if (fst hd) = "DROP"
                           then "[\"DROP\"],"
                  else  "[\""^(fst hd) ^ "=" ^(snd hd) ^"\"],"

         in string_of_avp lit
         |
         _ as hello-> (string_of_insert_pexpr hello)

let string_of_node=function P_strLit(lit) ->
         "{\"id\":"^"\""^lit^"\",\"type\":\"OF\"},"
         |P_id(id)->"{\"id\":"^id^",\"type\":\"OF\"},"
         |_ ->""


let fold_func data cname expr =
         if cname="actions" then  data ^  "\"" ^ cname ^ "\"" ^  ":" ^(string_of_action expr)
         else
                  if cname="node" then data ^  "\"" ^ cname ^ "\"" ^  ":" ^  (string_of_node
expr)
                  else
         data ^  "\"" ^ cname ^ "\"" ^  ":" ^  (string_of_insert_pexpr expr)  ^ ","


let isflownameID =function P_id(_) -> true
|
_ -> false


let isnodenameID= function P_id(_) -> true
|
_ -> false
(************************************************************************
Name=string_of_pgvdecl
Description= Generating code for global variable declaration
Input= A product type of gvdecl
```

```
Output= A String representing global variable declaration in python
*************************************************************************)
let string_of_pgvdecl  globalvar=
        globalvar.pvname ^"=" ^string_of_pexpr globalvar.pvalue

(************************************************************************
Name=string_of_pstmt
Description=Generating code for statements
Input=A select statement
Output= A String representing statements in python
*************************************************************************)
let string_of_pstmt = function
        P_expr(pexpr) -> string_of_pexpr pexpr
        |
        Pselect(columnname,tablename) ->
                "try:\n"
                        ^
                        "\t select=requests.get("^tablename^")\n"
                        ^
                        "\t if (select.status_code == requests.codes.ok):\n"
                                ^
                                (*Building a list of keys*)
                                let rec col_string= function
                                        [] -> ""
                                        |
                                        hd::tl -> "\"" ^ hd ^ "\"," ^ (col_string tl)
                                        in "\t\tcollist=[" ^ col_string columnname^"]\n"
                                        ^
                                        "\t\tselect=select.json()\n"
                                        ^
                                        "\t\tflow_entries=select[\'flowConfig\']\n"
                                        ^
                                        "\t\tfor i in range(len(flow_entries)):\n"
                                        ^
                                                "\t\t\tprint(\'\\n\\n\\n\')\n"
                                                ^

        "\t\t\tprint(\'+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++\')\n"
                                                ^
                                                "\t\t\tfor j in range(len(collist)):\n"
                                                ^
                                                        "\t\t\t\tif collist[j] in
flow_entries[i]:\n"
                                                        ^
                                                                "\t\t\t\t\tprint(\'----
---------------------------------------------------------------------------\')\n"
                                                        ^

        "\t\t\t\t\tprint(collist[j],flow_entries[i][collist[j]])\n"
                                                        ^

        "\t\t\tprint(\'+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++\')\n"
                                                ^
                                                "\t\t\tprint(\'\\n\\n\\n\')\n"

                        ^
                        "\t else:\n"
                                ^
```

```
                                      "\t\t print(\"HTTP error\",select.status_code)\n"

                   ^
                   "except requests.ConnectionError:\n"
                            ^
                            "\t print(\"Connection Unsuccessful\")\n"
          |
          Pinsert(tablename,cnamelist,exprlists) ->
                   let exprlist=List.map removequotes exprlists
                   in
                   (*first we are going to get a hyperlink and authentication credentials
separate*)
                   let hylink,auth=trans_link tablename in
                   (*now we get the length of the hyperlink*)
                   let hylink_len=String.length hylink in
                   (*now we removed the singles quotes from the hyperlink*)
                   let hylink_sub=String.sub hylink 1 (hylink_len-2)
                   in
                   (*get the index of flowname from the columnnames tables*)
                   let fnameind=indexof cnamelist "name"
                   and
                   (*get the index of nodename from the columnnames tables*)
                   nodenameind=indexof cnamelist "node"
                   in
                   (*got the value of the flowname*)
                   let fname=List.nth exprlist fnameind
                   and
                   (*got the value of the nodename*)
                   nodename=List.nth exprlist nodenameind
                   in
                   (*fname_removed_*)
                   let url=hylink_sub^"/node/OF/"^(ptypexpr nodename)^"/staticFlow/"^(ptypexpr
fname)
                   in
                   let putdata="{ \"installInHw\":\"true\","  ^ (List.fold_left2 fold_func ""
cnamelist exprlist) ^ "}"
                   in
                   (if ((isnodenameID nodename) && (isflownameID fname)) then
"url="^"\""^hylink_sub^"/node/OF/\"+"^(ptypexpr nodename)^"+\"/staticFlow/\""^"+"^(ptypexpr
          fname)^"\n"
                   else if (isnodenameID nodename) then
"url="^"\""^hylink_sub^"/node/OF/\"+"^(ptypexpr nodename)^"+\"/staticFlow/"^(ptypexpr
fname)^"\"\n"
                   else if (isflownameID fname) then  "url="^"\""^hylink_sub^"/node/OF/"^(ptypexpr
nodename)^"/staticFlow/"^"\""^"+"^(ptypexpr fname)^"\n"
                   else
                   "url=\""^url^"\"\n"
                   )
                   ^"putdata="^putdata^"\n"^
                   "headers={\"content-type\":\"application/json\"}\n"^
                   "try:\n"^
                            "\t
insert=requests.put(url,"^auth^",data=json.dumps(putdata),headers=headers)\n"^
                            "\t if (insert.status_code==requests.codes.ok):\n"^
                                      "\t\t print(\"Flow entry successfully updated\")\n"^
                            "\t elif (insert.status_code==requests.codes.created):\n"^
                                      "\t\t print(\"Flow entry successfully created\")\n"^
                            "\t else:\n"^
                                      "\t\t print(\"HTTP error\",insert.status_code)\n"^
                   "except requests.ConnectionError:\n"^
```

```
                   "\t print(\"Connection Unsuccessful\")\n"

          |
          Pdelete(flowlist,nodeid,tname) ->
          let compile_delete (flow,node,table) =
                   let flowname=removequotes flow
                   and
                   nodename=removequotes node in
                            let hylink,auth=trans_link table in
                                     let hylink_len=String.length hylink in
                                              let hylink_sub=String.sub hylink 1 (hylink_len-2)
in
                                                       let url=hylink_sub^"/node/OF/"^(ptypexpr
nodename)^"/staticFlow/"^(ptypexpr flowname) in
                                                       (if ((isnodenameID nodename) &&
(isflownameID flowname))
                                                                then
"url="^"\""^hylink_sub^"/node/OF/\"+"^(ptypexpr nodename)^"+\"/staticFlow/\""^"+"^(ptypexpr
flowname)^"\n"
                                                       else if (isnodenameID nodename)
                                                                then
"url="^"\""^hylink_sub^"/node/OF/\"+"^(ptypexpr nodename)^"+\"/staticFlow/"^(ptypexpr
flowname)^"\"\n"
                                                       else if (isflownameID flowname)
                                                                then
"url="^"\""^hylink_sub^"/node/OF/"^(ptypexpr nodename)^"/staticFlow/"^"\""^"+"^(ptypexpr
flowname)^"\n"
                                                       else
                                                       "url=\""^url^"\"\n"
                                              )
                                                       ^"headers={\"content-
type\":\"application/json\"}\n"^
                                                       "try:\n"^
                                                                "\t
remove=requests.delete(url,"^auth^",headers=headers)\n"^
                                                                         "\t if
(remove.status_code==204):\n"^
                                                                                  "\t\t
print(\"Flow entry successfully removed from the controller\")\n"^
                                                                         "\t else:\n"^
                                                                         "\t\t print(\"HTTP
error\",remove.status_code)\n"^
                                              "except requests.ConnectionError:\n"^
                                                       "\t print(\"Connection Unsuccessful\")\n"
          in
                   let join elem1 elem2 elem3 = (elem1,elem2,elem3) in
                                        let par_join expr=join expr nodeid tname in
                                                       let list_of_deletes= List.map par_join
flowlist in
                                                                let answer=List.map
compile_delete list_of_deletes in
                                                                         String.concat "" answer

(*************************************************************************
Name=string_of_progrm
Description=Generating code for a NWQL program
Input=A NWQL program
Output= An executable python program
*************************************************************************)
let string_of_program (pgvdecllist,pstmtlist)=
```

```
         "import requests \n" ^
         "import json\n"^
        String.concat "\n" (List.map string_of_pgvdecl pgvdecllist) ^ "\n" ^
        String.concat "\n" (List.map string_of_pstmt pstmtlist)
```

11.9

11.10 nwql.ml

```
open Printf

exception Usage of string

(*take the input and the output file name
if output filename is not given by default, make it a.out*)
let _=

let (input_file,output_file) =
         if Array.length Sys.argv ==2 then
                  let output_file_fname=Sys.argv.(1) in
                           let output_file_index=String.index output_file_fname '.' in
                                    (Sys.argv.(1),(String.sub output_file_fname 0
output_file_index)^".out")

         else if Array.length Sys.argv ==4 then
                  (Sys.argv.(1),Sys.argv.(3))

(*if wrong number of inputs are given raise an exception*)
         else
                  raise(Usage("usage: ./nwql [input file name] or ./nwql <input file name> -o
<output file name>"))
(*get the length of the input filename*)
in

         let input_file_length=String.length input_file in
                  (*if a substring created from the position of 5th last character, with a length of
5 does not contain the correct file extension*)
                  if (String.sub input_file (input_file_length -5) 5) <> ".nwql" then
                           raise(Usage("usage: Input file should have an extension .nwql"))

                  else

                           let lexbuf = Lexing.from_channel (open_in input_file) in

                                    let program = Parser.program Scanner.token lexbuf in

                                             let python_program  = Translate.translate_program
program in

                                                      let pyFile = open_out output_file in

                                                               fprintf pyFile "%s\n"
(Compile.string_of_program  python_program);

                                                               close_out pyFile;

                                                               Sys.command ("chmod +x " ^
output_file) ;
```