LanguageDesign - Graph Quering Language - DSL

Zhenfei Nie
zhen.fei.nie@usi.ch

30 Nov 2014

1 Usage

1.1 Graph Definition

```
CREATE GRAPH test ([(name, string), (age, int64),
        (nationality, string)], [(duration, int64)]);
INSERT INTO test VERTEX (2, "Bob", 25, "US");
INSERT INTO test VERTEX (5, "Alice", 28, "EN");
INSERT INTO test VERTEX (6, "James", 29, "AU");
INSERT INTO test VERTEX (7, "Robin", 15, "AU");
INSERT INTO test VERTEX (8, "Nami", 27, "JP");

INSERT INTO test EDGE (2, 5, 1, 30);
INSERT INTO test EDGE (2, 6, 12, 360);
INSERT INTO test EDGE (2, 7, 14, 130);
INSERT INTO test EDGE (2, 8, 17, 230);
INSERT INTO test EDGE (5, 8, 29, 220);
INSERT INTO test EDGE (5, 6, 2, 20);
INSERT INTO test EDGE (5, 7, 7, 120);
INSERT INTO test EDGE (6, 8, 27, 1120);
INSERT INTO test EDGE (7, 8, 127, 120);
INSERT INTO test EDGE (8, 2, 2, 10);
INSERT INTO test EDGE (8, 5, 5, 10);
```

1.2 Graph Quering

```
SELECT
        [name="Bob"&&age>20]
        ( [nationality="EN"] | [nationality="AU"&&age>20] )*
        [age<30]
FROM test
WHERE LEN >= 2;
```

The result of above query shoule be *[Bob – Jame –Nami, Bob – Alice – Nami]*. Please see the figure below.

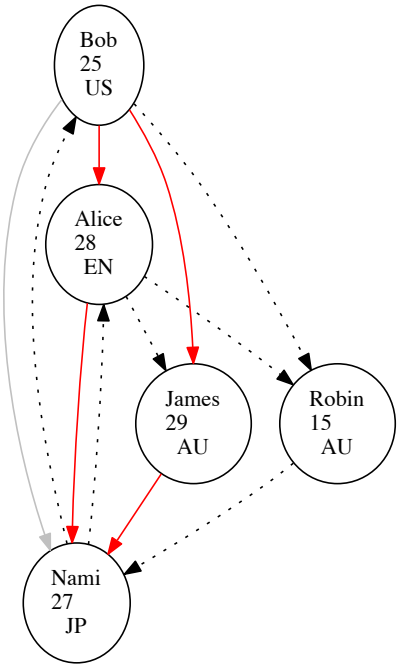

Figure 1: Red lines are paths.

2 Grammer in formal BNF

2.1 Lexer

```
D               [0-9]
L               [a-zA-Z_]
H               [a-fA-F0-9]
E               [Ee][+-]?{D}+
FS              (f|F|l|L)
IS              (u|U|l|L)*

%%
{L}({L}|{D})*               {  return( IDENTIFIER); }

0[xX]{H}+{IS}?              {  return( CONSTANT); }
0{D}+{IS}?                  { return( CONSTANT); }
{D}+{IS}?                   { return( CONSTANT); }
L?'(\\.|[^\\'])+'           { return( CONSTANT); }

{D}+{E}{FS}?                {  return( CONSTANT); }
{D}*"."{D}+({E})?{FS}?      { return( CONSTANT); }
{D}+"."{D}*({E})?{FS}?      { return( CONSTANT); }

L?\"(\\.|[^\\"])*\"         {  return(STRING_LITERAL); }

CREATE GRAPH { return CREATEGRAPH; }
INSERT       { return INSERT; }
INTO         { return INTO; }
VERTEX       { return VERTEX; }
EDGE         { return EDGE; }

SELECT       { return SELECT; }
FROM         { return FROM; }
WHERE        { return WHERE; }
HAVING       { return HAVING; }

'=='  |
'!='  |
'<'   |
'>'   |
'>='  |
'<='         { return COMPARISION; }
```

2.2 Parser

```
program : END | graph END
graph : schema decls
decls : decl | decls decl
decl : vertex | edge
vertex : attributes
edge : attributes

schema : ([(CONSTANT,)],[(CONSTANT,)])
attributes : attribute | attributes attribute
attribute : (CONSTANT, attr)
attr : list | map | int64 | string | double
list : attribute | list attribute
map : {attribute}
```

```
%token NAME
%token ID
%token TIMESTAMP

program :
        END
    |   graph END
    |   query END
    |   graph query END
    ;

graph :
        graph_definition decls
    ;

graph_definition :
        CREATE GRAPH IDENTIFIER schema
    ;

decls :
        decl
    |   decls decl
    ;

decl :
        vertex
    |   edge
    ;
```

```
vertex :
        INSERT INTO IDENTIFIER VERTEX '{' IDENTIFIER ',' attributes '}'
    ;

edge :
        INSERT INTO IDENTIFIER EDGE '{' IDENTIFIER ','
                IDENTIFIER ',' TIMESTAMP ',' attributes '}'
    ;

query :
        SELECT reg_exp FROM IDENTIFIER where_clause
    ;

reg_exp :
        reg_exp reg_exp
    |   reg_exp '|' reg_exp
    |   reg_exp '*' reg_exp
    |   '(' search_exp ')'

search_exp :
        search_exp AND search_exp
    |   search_exp OR search_exp
    |   NOT search_exp
    |   predicate

predicate :
        comprision_predicate
    |   existence_test
    ;

comprision_predicate :
        IDENTIFIER COMPARISION CONSTANT
    ;

existence_test :
        EXISTS '<' IDENTIFIER '>'
    ;

where_clause :
        WHERE search_condition
    ;

opt_having_clause :
        /* empty */
    |   HAVING search_condition
    ;
```

```
searching_condition :
        length_condition
    ;

length_condition :
        LEN COMPARISION CONSTANT
    ;
```