Pathfinder Report

1) Street Network Definition Language

   a) One-way streets are defined by One-Way keyword and connections are represented using a dash followed by a greater than sign. It should be noted that one-way streets can only be entered, they cannot connect to any other street.

      Example:

      ```
      One-Way street1;
      Two-Way street2;
      Two-Way street3;
      street2->street1; //legal operation
      street1->street3; //illegal operation
      ```

   b) Two-way street networks are similarly defined by Two-Way keyword. Connections are represented by less than, dash, greater than signs. There is no limitation to how many streets a two-way street can be connected. That is, two-way streets can be connected to infinitely many streets.

      Example:

      ```
      Two-Way street1;
      Two-Way street2;
      Two-Way street3;
      street1<->street2;
      street1<->street3;
      ```

   c) Point are defined by Point keyword. Point can have arbitrary properties. These properties can be added to a individually or as a list. Properties are name and value pairs. Names must be string primitives. When a properties is added multiple times with different values, new values will overwrite the old values .

      Example:

      ```
      Point burger_king;
      burger_king.insert('name','Burger King');
      burger_king.insert( { ('type','Restaurant'), ('sells','Hamburger') } );
      ...
      burger_king.insert('sells', 'Fast Food'); //sells property became 'Fast Food'
      ```

   d) Street properties are treated the same way as point properties. Average time to pass a street is a special property called 'avg_pass' which is denoted by an integer. Temporary properties are passed with tmp function and an additional time interval is appended. It also has an optional message parameter that must be a string primitive. It can create new properties or existing properties.

Example:

```
Two-Way tunus;
tunus.insert('name','Tunus');
tunus.insert('avg_pass', 300);
tunus.tmp('avg_pass', 2100, [17:00,19:00], '30 min delay at 5-7 pm.');
//avg_pass property is overwritten for 2 hours, it will return to 300 later
```

e) Language supports strings, integers and floats as primitive types. Strings are captured between single quotation marks. Primitive values must be set as they are declared. For strings of values, language supports lists, sets, and maps. Lists and sets have the same syntax but set members must be unique. Uniqueness is not enforced on list members. In addition lists can contain values of different types whereas sets cannot. Maps contain by <key,value> pairs. Map keys must be unique strings. Map values can be any type. Uniqueness is not enforced for map values.

Example:

```
Integer a = 5;
Float f = 0.07;
String str = 'Hello world';
List people = { 'Erdinç Albayrak', 'Fatih Aktepe'};
Set primes = { 2, 3, 5, 7};
Map map = { <'Greetings',{'Hello','Welcome'}>,
            <'People', people>,
            <'Primes', primes>};

people.insert( a); //this is allowed

primes.insert( str); //whereas this is illegal
```

2) Street Network Query Language

a) Language has several keywords defined for queries. These are REACH, FROM, BY, SORTON, LIMIT. REACH, FROM, BY take point and street parameters. The result of a query is a list of streets and points to be visited in order, as well as estimated time to finish the journey and distance to be travelled to finish the journey.

The keywords' meaning are as follows: REACH specifies the final destination of a journey and FROM specifies the starting point of a journey. BY specifies the points and streets that must be visited throughout the journey. REACH and FROM take exactly one parameter whereas BY can take multiple parameters. Therefore, the general form of a query is as follows:

REACH a
FROM b
BY c_1, c_2, ..., c_n;

- SORTON defines comparison parameter for sorting. It can take ‘time’, ‘distance’, ‘simple’. Sorting on time returns the routes that complete in the lowest time. Sorting on distance returns the routes that complete the journey in shortest way. Sorting with simple parameter returns the routes that visit the lowest number of streets to reach the destination.

Example:

```
REACH Street.name=’Tunus’
FROM Point.name=’Starbucks’ + Point.location=’Bilkent’
BY (Point.sells=’Gas’ | Point.sells=’LPG’), Point.type=’Restaurant’
SORTON ‘time’;
```

- LIMIT keyword is followed by an integer parameter which specifies the number of results to be returned by the query. Using LIMIT without SORTON is allowed. However, when they are both used in a query, sort operation will precede the limit operation.

Example:

```
REACH Street.name=’3235’
FROM Street.name=’5564’
BY Street.name=’4264’
LIMIT 10;
```

- Language also supports alternation by |, concatenation by + and repetition by *. Boolean expressions can be formed using these as well as parantheses. Boolean operands are the standard comparison symbols.

Example:

```
...
BY ( Point.prices[‘Gas’] | Point.prices[‘LPG’] ) <= 4 ;
```

- String functions Integer strlen(), Integer substring(String), String index(Integer) are provided to express complex string constraints.

b) Variables can be declared with VAR keyword followed by type name and symbol.

Example:

```
VAR Integer i
REACH Street.avg_pass= i
FROM ...;
```

c) Assignment operator ‘<-‘ has been added to provide modularity for complex queries that should be divided into smaller parts.

Example:

```
t <- REACH Point.type(‘Restaurant’)
FROM Point.name(‘Home’);

REACH Street.name=’Tunus’
FROM Point.name=’Starbucks’ + Point.location=’Bilkent’
BY t
SORTON ‘time’
LIMIT 5;
```