Pathfinder Tutorial

Pathfinder is a language designed to represent street networks. Broadly, the language is used to define street and point relationships, and query routes with a number of constraints. The language consists of two sections. First of these is definition language which is used to create the networks. Second section is the query language which is used to generate routes that are subject to certain implications.

Definition Language

Street Networks

Street networks are created using two types of streets: one-way streets and two-way streets. Street types are denoted with One-Way and Two-Way keywords. For a street to have any properties or connections to other streets, it must first be defined in the following way:

```
One-Way x;

Two-Way y;
```

Streets are connected using arrows. Two-way streets connections are denoted using "<->" symbols. When a two-way street is to be connected to a one way street, "->" symbols are to be used with the two-way street as the left hand side and the one-way street as the right hand side. Another important note is that a one-way street can be connected to at most 1 two-way street.

```
One-Way x;

Two-Way y;

Two-Way z;

x -> y; //illegal statement

y->x; //legal statement

z <-> x; //illegal since x is a one-way street

y <-> z ;

z <-> y; //legal but redundant, the line above already implies this relation
```

Points

Points are structures that are located on streets. They are denoted using the Point keyword.

```
Point x;
```

Points become useful with the introduction of properties.

Types

Our language supports strings, integers and floats as primitive types. They are declared using String, Integer, Float keywords respectively.

```
Integer a = 5;

Float b = 0.33;

String c = ‘Hello World’;
```

In addition, lists, sets and maps are supported as basic data structures. Pathfinder’s structures are pretty flexible. They do not enforce definite lengths at declaration. In addition, lists and maps will accept any type of value most of the time. Data structures are declared using braces. Lists will accept any type of value to be inserted. On the other hand, sets can only contain one type at a time. Furthermore, set members have to be unique.

```
List random = { ‘Hello world’ , 5 , ‘Goodbye’ };

Set squares = { 1, 4, 9, 16};

random.insert(squares); //allowed

squares.insert(random); //disallowed since random contains String type values
```

Maps take < key, value > pairs. Keys have to be a unique string literal. When a key is added into the map multiple times, old values will be lost and only new value will be kept. Values can be of any type, including another map.

```
Map games = { < ’city’, {‘Ankara’, ‘Vienna’ } >

              , <’animal’, ‘shark’>

              , <’squares’, squares> };
```

Properties

Properties can be assigned to both points and streets. Generally, properties are defined as (name,value) pairs. Street and point properties are added with a insert function. In addition, properties can be inserted as a list. Streets have a special property called ‘avg_pass’ which specifies the average time to pass that street. It is strongly recommended this value is set for every street that is created in order to get meaningful results from queries.

```
Street a;

a.insert(‘name’, ‘2475. St.’);

a.insert( { (‘avg_pass’ , 300), (‘length’, 2000) } );
```

Pathfinder also supports temporary variables. These variables are inserted with tmp function which takes the name, value and lifetime of the variable. It also takes an optional message parameter where the reason for the temporary variable may be explained. If a temporary variable is created with the same name as an existing variable, the temporary variable’s value will be kept while it is alive. When the variable dies, property will retain its old value.

```
a.tmp( ‘avg_pass’ , -1 , [15:00,20:00], ‘Road closed due to security reasons’ );
```

Query Language

Pathfinder generally makes use of several keywords to create queries. These words are REACH, FROM, BY, SORTON and LIMIT. The general form of a query is:

```
REACH a
FROM b
BY c, d, ...;
```

A query will return the routes starting from the FROM clause, going to REACH clause and visiting other nodes expressed in BY clause. A route is a list of streets that are required to take in order to complete the journey with the given constraints in BY clause.

REACH Clause

REACH clause specifies the destination of a journey. Exactly one point or street expression must follow the REACH keyword in a query.

FROM Clause

FROM clause specifies the starting point of a journey. Similar to REACH, exactly one point or street expression must follow the FROM keyword in a query.

BY Clause

BY clause specifies the points and streets to be visited in a route. It can take multiple expressions. Expressions can make use of | for alternation, + for concatenation, * for repetition. Most boolean expressions such as greater than, equals, etc. are supported as well.

Example: Find the routes from Karanfil street to Bilkent Starbucks coffee, visiting a gas station that sells gas or LPG.

```
REACH Point.name=’Starbucks coffee’ + Point.branch=’Bilkent’
FROM Street.name=’Karanfil’
BY Point.type=’Gas Station’ + ( Point.sells=’Gas’ | Point.sells=’LPG’);
```

SORTON Clause

SORTON clause specifies the order in which routes will be returned. It can take 3 values: ‘time’, ‘distance’, ‘simple’. Sorting on time returns the routes that arrive at the destination quickest. Sorting on distance chooses the routes that minimize the distance to be travelled. Sorting with ‘simple’ parameter returns the routes that visit the minimum number of streets to reach the destination.

LIMIT Clause

LIMIT clause is a very simple one: it takes an integer parameter and limits the number of routes to be returned to that number.

Example: Find the 5 quickest routes from Karanfil street to Bilkent Starbucks coffee, visiting a gas station that sells gas or LPG.

Now our solution becomes:

```
REACH Point.name='Starbucks coffee' + Point.branch='Bilkent'

FROM Street.name='Karanfil'

BY Point.type='Gas Station' + ( Point.sells='Gas' | Point.sells='LPG')

SORTON 'time'

LIMIT 5;
```

Variable Queries

Pathfinder supports declaring variables to be used in queries in order to allow more complex queries. The syntax for this is: VAR <type-name> <variable-name>.

Example:

```
VAR String str

REACH ...
```

Modularity Feature

Another feature that Pathfinder supports is assigning query results to a variable for further use. This will be particularly useful when a query becomes too complex to express in one query so that it is useful to break it up into pieces. We achieve this feature with the '<-' symbol.

Example:

```
results <- REACH ...

FROM ...

BY ...;
```

Now we are able to use the routes in variable results in other queries.