Bilkent University
Computer Engineering Department
CS 315 - Programming Languages

Spring 2016

Project Part I

Due Apr 1, 2016 23.59

Design a Street Network Definition and Querying Language

You are asked to design a street network definition and querying language. There shall be two parts to your language:

1. The street network definition language: This language would be used to define a street network. It will support specification of the streets and points (connecting corners) of the network.
2. The street network querying language: This language is used to define routes or similar queries on the street network.

There are specific requirements for each part. These are listed below:

1. (45pts) The street network definition language shall support:
   (a) (5pts) Defining a street network with one-way streets.
   (b) (5pts) Defining a street network with only two-way streets.
   (c) (10pts) Defining point properties.
      - A point property is some data value associated with a point.
      - It shall be possible to attach multiple properties to a point.
      - A property should be a *(name, value)* pair. For instance, if a point represents a gas station, then a point property can be `('name', 'Bilkent Petrol')`. Remember that there may be multiple such properties, such as: `('type', 'gas station')`, `('name', 'Bilkent Petrol')`, `('sells', 'LPG')`, `('payment-options', 'Cash')`.
   (d) (10pts) Defining street properties
      - It shall be possible to attach multiple properties to a street.
      - The streets shall have a property to describe the average time to pass the street.

- The streets shall support adding temporary properties. These properties shall support the definition of a delay on the average time or close of the street with a reason and period, such as *"a delay of 30 minutes due to an accident till 12:00"*, *"road closed due to roadwork between 10:00 and 18:00"*.

(e) (15pts) A dynamic type system for the property values (property names should be strings)

- Support `strings`, `integers`, and `floats` as primitive types. For instance, the `'average-time'` property of the streets may have an `integer` value, whereas the `'name'` attribute may have a `string` value.
- Support `lists`, `sets`, and `maps` as collection types. For instance, it should be possible to have a property like: `('sells', {'LPG', 'Diesel'})`. This would be an example of a property, whose value is a `list` of `strings`. Arbitrary nesting should be possible as well. For instance, `('sells', {'gas-types': {'LPG', 'Diesel'}, 'payment-options': {'Cash', 'Visa', 'Mastercard'}}`. This is an example where the value type is a `map` from a `string` to a `list`.

2. (55pts) The street network querying language shall support:

(a) (30pts) Creating route queries. A route query is an expression specifying a route while visiting several points and/or streets. A route is an alternating series of points and streets. Importantly, we are not asking you to evaluate queries. We are asking you to create a language to express them. A route query shall be able to specify:

- (6pts) Concatenation, alternation, and repetition.
- (6pts) Provides filtering constraints as Boolean expressions, which are composed of predicates defined over street properties as well as incident point properties.
  - For instance, one may want to find all routes, from a specific building at Bilkent to a shopping mall while stopping at a gas station, formally, the start point of the first street has a property `name='Bilkent Building 81'`, passes through a street with a starting point that has properties `type='gas station'` and `sells='Diesel'`, and finally the last street end point has a property `type='Shopping Mall'`.
  - Another one may want to find the routes from a train station at Ankara to his home, formally, the start point of the first street has properties `type='train station'`, `city='Ankara'`, the last street has a property of `name='13th St.'` and the end point has a property of `house-number='25'`.
- (6pts) Support for existence predicates as well as arithmetic expressions and functions in predicate expressions. For instance: a point containing or not containing a property with a given name or value; or a street that has a certain property whose value is greater than a constant; or a point that has a certain property whose value is a string that starts with `'A'` (this would require a string indexing function).
- (6pts) Support for sorting based on: "Shortest route in time", "Shortest route in distance" or "Simplest Route" (minimum number of streets on route).
- (6pts) Support to limit the number of routes on the result. For instance, one may want to limit the query to return only the best three routes.

(b) (15pts) Having variables in route queries. For instance, you may want to query all routes where both start and end streets in the route have a property called name

with the same property value, but the value is not known. In this case, that value becomes a variable.

(c) (10pts) Modularity, that is dividing regular route queries into multiple pieces that are specified separately. This would require giving names to each piece and being able to use those names in a higher-level query.

We ask you to:

1. (60%) Design a language (give it a name) to meet the requirements described above.
   (a) (25%) Write a tutorial for the language you have designed. The tutorial's goal is to teach your language to someone who does not know about it.
   (b) (25%) Write a report describing how you addressed each requirement. For each requirement that has points associated with it, provide a small code segment in the language you designed, showcasing your design. If you like, you can also reference parts of your tutorial in the report.
   (c) (10%) Write 2 sample street network definitions (one one-way, one two-way), and 10 queries.
2. (40%) Write a lexer for your language (you might write separate lexers for the definition and querying, it is up to you), using Lex.

Logistics

Once you are done, put your deliverable under a directory named `group<GroupNo>_proj1` and make an archive from that directory. It is important that your code should compile without any issues using the `make` command. Once complete, it should output an executable called `'lexer'` that can be used to lex your sample input. For example, the following Unix commands could be used:

```
mkdir group<GroupNo>_proj1
cd group<GroupNo>_proj1
    mkdir code      # contains Makefile, lex, and C code
    mkdir samples   # contains the program samples
    mkdir documents # contains the tutorial and report
    #(edit and test your files)
    ...
cd ..
tar -cvzf group<GroupNo>_proj1.tar.gz group<GroupNo>_proj1
```

Then e-mail this newly generated file (named `group<GroupNo>_proj1.tar.gz`) to Arda Unal <arda.unal at bilkent.edu.tr>

Reports in formats other than `.pdf` and `.txt` are not accepted.