Piconot

Due: Monday, October 8, 2012 at 11:59pm Pacific
Complete this assignment with your group from class — 210 total points..

Overview

This assignment asks you to design and implement your own syntax. The goals of this assignment are to: (a) get practice formally specifying the syntax of a language, (b) design a brand-new syntax, and (c) get some experience implementing a language the **wrong way**. You'll probably design a syntax that is either difficult to implement internally or that is so different from the semantics of the language that it requires significant implementation effort on your part. This is good! I want you to have a chance to experience the limitations of the tools we've been discussing, so you'll develop an instinct for when they are and are not appropriate. I also want you to have a chance to experience how different a language's syntax can be from its semantics.

In short: get in there and play. Push everything we've learned so far to the limit. When we've done that, we'll have earned the right to set these topics aside and move on to something new.

1. Warmup: Formal Syntax *(20 total points)*

Formally specify the syntax for Picobot, as it's defined in CS 5. Place your definition in `grammar-orig.txt`. You must use a well-known standard (e.g., regular expressions or Extended Backus-Naur Form [EBNF]) to formally specify the syntax.

The "Materials" section contains information on how to get the files you need.

Note: If you took CS 81, these standards should be either well-known or easily decipherable. If it turns out that no one in your group is familiar with these standards, that's OK—they're easy to learn. You can give it a go on your own or (more preferably) contact `dslhelp` to set up a brief tutorial.

Do not *use syntax diagrams, I beg you.*

2. Syntax Design *(60 total points)*

Design your own syntax for PicoBot. Try to come up with a design that is faithful to the domain (of maze-navigation), that does *not* add any new features (e.g., non-determinism), but that is also not restricted to students in CS 5. The more novel your syntax, the better, assuming that it's faithful to the domain. Design your syntax as an ideal, i.e., without worrying about how you'll have to implement it.

Note: If you're in a group of three, you should come up with *two* designs. You must describe and justify both designs, but you need to implement only one of them.

(a) Formally specify the syntax for your design. Place your definition in the file `grammar-ideal.txt`. *(10 points)*

(b) In the file `design.txt`, describe and justify your design. *(50 points)*

If you're in the group of three, you should specify both of your designs in `grammar-ideal.txt`

Your description should be high-level, clear, and no more than 250 words. It should address the following questions:

> Who is the target for this design, e.g., are you assuming any knowledge on the part of the language users? Why did you choose this design, i.e., why did you think it would be a good idea for users to express the maze-searching computation using this syntax? What behaviors are easier to express in your design than in PicoBot's original design? If there are no such behaviors, why not? What behaviors are more difficult to express in your design than in PicoBot's original design? If there are no such behaviors, why not? On a scale of 1–10 (where 10 is "very different"), how different is your syntax from PicoBot's original design? Is there anything you don't like about your design?

If you're in the group of three, you should describe both of your designs in `design.txt` (using at most 250 words to describe each design). Also, you should indicate which of these designs you've chosen to implement and justify your choice.

3. Syntax Implementation *(60 total points)*

This part might take more time than you think. Budget accordingly.

(a) Using only the techniques described in the the Internal Techniques section, implement your syntax in Scala. Replace the program in the `main` function of `PicobotSyntax.scala` with a program in your syntax. You may any other files that you need to implement your syntax. *(55 points)*

As you proceed with your implementation, you should simultaneously answer the questions in the next section.

The library has auto-generated documentation.

I've provided a library that implements the semantics, you'll need to transform statements in your internal language into calls to the provided library. You can compile your code against the library with the command:

```
scalac -cp /path/to/library/picolib.jar PicobotSyntax.scala
```

and execute the program with the command:

```
scala -cp /path/to/library/picolib.jar PicobotSyntax
```

You are not allowed to change the library code. You *are* allowed to change your syntax.

(b) Formally specify the syntax you ultimately implemented in the file `grammar-actual.txt`. *(5 points)*

4. Evaluation *(60 total points)*

So... how did it go? Did you have to modify your syntax very much? Why or why not?

Document your experience in the file `evaluation.txt`.

In particular, for each change to the syntax, you should state why you made that change (e.g., so that you could implement it internally, or so that it would match the library calls better). You should also answer the following questions: On a scale of 1–10 (where 10 is "a lot"), how much did you have to change your syntax? On a scale of 1–10 (where 10 is "very difficult"), how difficult was it to map your syntax to the semantics?

5. Feedback *(10 points)*

Answer the questions in the file `feedback.txt`.

Internal Techniques

You may use only these techniques to implement your syntax. If you discover that these techniques are not sufficient, you should change your syntax so that you can use these techniques.

Fluency: Function chaining, sequencing, or nesting. Omitting syntax (e.g., the dot on a method call, the parentheses on a function or method call, or `new` on object instantiation) and changing syntax (e.g., braces on a single-argument function or method).

Closures: Higher-order functions and blocks (including partial and curried functions and by-name parameters).

Literal extension: Implicit functions that add behavior to data.

Mixin composition: Traits that add behavior to data.

Materials

The materials are available in the Subversion repository. Because you're working in groups, the instructions are a little different.

First, one partner should create a directory for your pair:

```
svn mkdir -m "creating group" https://svn.cs.hmc.edu/dsl/fall12/ours/group
```

where `group` is your group name. Your group name is formed by joining together each of your knuth userids, in alphabetical order, with a hyphen. Thus, if Mary Jones (`mjones`) and Todd Smith (`tsmith`) were working together, their group name would be `mjones-tsmith`.

Both partners should checkout the group's directory into their own working copy (which is located in local directory `dir`):

```
cd dir
svn co https://svn.cs.hmc.edu/dsl/fall12/ours/group
```

One partner should now copy the assignment files to their working copy:

```
cd group
svn copy https://svn.cs.hmc.edu/dsl/fall12/hw/6 hw6
```

This partner should immediately commit their files:

```
svn commit -m "initial import" hw6
```

The other partner should now be able to check out the files, in their working copy by running

```
cd dir/group
svn update
```

Both team members should now be able to collaborate, using their working copies. Modify the materials to supply your answers and use the instructions below to submit.

Submitting Your Solution

Make sure you're in the `hw6` directory of your working copy and execute:

```
svn commit
```

Committing is like voting in Chicago: you should do it early and often.

Collaboration and Honor Code

I expect you to abide by the Harvey Mudd Honor code. You and your teammates must divide the work equally among yourselves, and you must work together (at the same time and in the same location). Your solution to this assignment should be produced only by the members of your team. You may discuss concepts and language design at a high level with any student in the class — I encourage you to do so! However, you may not copy solutions from anyone. If someone strongly influences your language design, be sure to cite that person.

If you have any questions about what behavior is acceptable, it is your responsibility to come see me before you engage in this behavior. I will be happy to answer any of your questions.