Università degli Studi di Pisa

Laurea Specialistica in Informatica

Laurea Specialistica in Informatica e Networking

Advanced Programming

Final Term Paper

Start Date: 3/09/2013
Submission deadline: 23/09/2013 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone's code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary. **Do not include in an exercise code only needed for a later exercise**.
4. cite references to literature or Web pages from where information was taken.

Introduction

In this project, you will develop a DSL for writing data flow programs. The language provides a construct `Flow` that allows operating on the data in a container. Here is an example of a Flow program:

```
p = Flow("item");
p1 = p.fold("result", "0", "result += item;");
func = p1.generate();
```

The first line creates a Flow object, within which the variable named `item` can be used to refer to the individual elements of the container. The second line defines a folding operation, where `result` is the name of the variable where the result is accumulated, the second parameter is the initial value for the result, and the third expression is the instructions performed at each iteration. The third line generates an executable function form a Flow. The code generated for func can be used as follows:

```
a = [2, -3, 4, 5];
total = func(a);
```

The last line applies the generated function to an array. The result produced will be 8.

There are other Flow operations, which can be combined. For example:

```
p2 = p.filter("item > 0");
p3 = p2.map("item * item").fold("result", "0", "result += item;");
func2 = p3.generate();
total2 = func2(a);
```

The result in this case will be 37.

Exercise 1

Design a set of classes to represent Flow expressions and statements.

Exercise 2

Implement a recursive descent parser for Flow, without using a parser generator. Ensure that each parser method returns a parse tree for the input it analyzes.

Exercise 3

Write an interpreter for Flow expressions. In particular, the `generate()` method should produce code in a scripting language like JavaScript or Python. Ensure to use polymorphism in the implementation of the interpreter. Provide the code generated for the examples in the introduction.

Exercise 4

Extend the language to include additional operators: operator `some` that will return a boolean according to whether a predicate expression holds for some of the items; operator `scan` that collects accumulated results, for example

```
p2.scan("result", "0", "result += item;");
```

when applied to array `a`, will create an array `[0, 2, -1, 3, 8]`.
Provide an implementation by extending the classes developed in earlier exercises and show the code generated for an example of your choice.

Exercise 5

Extend the generator, so that it will perform code optimization by merging consecutive loops. Provide the generated code for the example in the introduction. Perform some benchmarking to show the difference between the normal and optimized versions.

Exercise 6

Describe the dataflow programming model and in particular what kind of object abstractions can be used in an Object Oriented language to emulate dataflow programs. What opportunities does dataflow programming provide to exploit parallelism?