Homework Assignment 2

Posted: 2/02/2010
Due: 2/18/2010

The assignment is due in class; please follow the instructions on the homework submission form.

This is a major assignment; start early and visit during the office hours if you run into difficulties. You must solve Parts 1-4 sequentially. You can solve Part 5 even if Parts 1-4 are incomplete.

Objectives

Implement the DFA construction algorithm discussed in class.

Implement a scanner.

Related Files

Homework submission form:
http://www.cs.unc.edu/Courses/comp524-s10/hw/submission-form.pdf

Program skeleton; the starting point for this assignment.

http://www.cs.unc.edu/Courses/comp524-s10/hw/2/HW2-Skeleton.zip

Regular Expressions describing the Java lexical structure:

http://www.cs.unc.edu/Courses/comp524-s10/hw/2/java-lexical-regex.txt

Part 1

This is a "fill in the blank" type assignment. Download the provided Java program skeleton and familiarize yourself with the provided classes and interfaces.

Implement the missing NFA construction operations in the class `NFA`, such that arbitrary regular expressions can be converted into equivalent NFAs.

You can check your progress with the provided `Show` tool, which converts regular expressions to finite automata and outputs them in the *dot graph description language*.

You can download a tool to visualize these graphs at http://www.graphviz.org/. You are **strongly encouraged** to do so and to compare the output with the NFA construction examples in the book and on the slides.

Part 2

Implement the missing NFA-to-DFA conversion algorithm in the class `DFA`.

Again, you can and should check your progress with the provided `Show` tool and visually compare the resulting graphs with the examples in the book and on the slides.

Use `java Show DFA-DETAIL` to show the node labels in the output. Hint: this could be used to show the sets of corresponding NFA states.

Part 3

Implement the DFA optimization algorithm in the class `DFA`. **Note**: the actual partitioning algorithm to find the optimal number of equivalence classes is already provided by the class `DFAOptimizer`; you only have to implement the equivalence-class-to-DFA state conversion.

Again, check your output with the `Show` tool against examples in the book and in the slides.

Part 4

Implement a Java program, named `Tokenize`, that can tokenize a text file (or, alternatively, the standard input stream) based on a provided list of token types.

`Tokenize` must accept one or two command line arguments. The first argument specifies the name of a file containing a list of token types. Each line contains one token type and is formatted as follows:

<token type name>: <regular expression>

In other words, each line consists of the name of the token, a colon, a space, and one regular expression that extends until the end of the line.

Recall that ambiguous (e.g., overlapping) token specifications require precedence rules. For this assignment, assume that tokens are always specified in order of decreasing precedence in the token list file, i.e., assign precedence values based on the input order.

The second argument specifies the name of a text file. If the second argument is omitted, then the program should process the standard input stream (`System.in`).

Your program is to construct an optimized DFA based on the tokens described in the token type list (first argument) using the multi-token-type construction that we discussed in class (a constructor for this purpose exists in the class `NFA`). After constructing the optimized DFA, your program should tokenize the file specified by the second argument (or `System.in`) using the constructed DFA and output each encountered token on a separate line including the line and column number of its first character and its value (see the example output below).

All token types that have names that start with a minus (such as `-WhiteSpace`) should be silently discarded by the tokenizer and not reported to the user, i.e., do not output such tokens.

Any errors (including malformed tokens) should be reported to the user; the program should not crash under any circumstances.

Note: constructing the optimized DFA for the Java lexical structure can require up to a few minutes of runtime and substantial amounts of memory. If you encounter an `OutOfMemoryError`, then you need to increase the heap space available to your program. For example, this can be done by specifying the option `-Xmx1Gb` to the `java` interpreter.

Part 5

The specification of the Java lexical structure provided in the file `java-lexical-regex.txt` (see related files above) is incomplete. Provide short answers to the following questions:

a) Provide a legal Java program that cannot be tokenized with the provided token specification.

b) Explain why this limitation is unavoidable with the implemented algorithms.

c) How would you have to change your solution to support *any* legal Java program?

Hint: one reason is that Java allows escape sequences of the form \uXXXX (where X is a digit) that are preprocessed before the actual tokenization starts. This is not the limitation that this questions asks for, but it is closely related.

Example

Suppose we are given the following Java file, and the list of Java token types (see related files).

Content of `HelloWorld.java`

```java
// a useless import
import java.lang.String;

/* a comment /* within a comment //  */
public class HelloWorld {

    public static void main(String args[]) {
        if (args.length == 0)
            System.out.println("Hello World!");
        else
            System.out.println("Hello " + args[0] + "!");
    }

}
```

Your tool should generate output similar to the following.

Output of `java Tokenize java-lexical-regex.txt HelloWorld.java`

```
002:01 Comment(// a useless import)
003:01 Keyword(import)
003:08 Identifier(java)
003:12 Separator(.)
003:13 Identifier(lang)
003:17 Separator(.)
003:18 Identifier(String)
003:24 Separator(;)
005:01 Comment(/* a comment /* within a comment //  */)
006:01 Keyword(public)
006:08 Keyword(class)
006:14 Identifier(HelloWorld)
006:25 Separator({)
008:05 Keyword(public)
008:12 Keyword(static)
008:19 Keyword(void)
008:24 Identifier(main)
008:28 Separator(()
008:29 Identifier(String)
008:36 Identifier(args)
008:40 Separator([)
008:41 Separator(])
008:42 Separator())
008:44 Separator({)
009:09 Keyword(if)
009:12 Separator(()
009:13 Identifier(args)
009:17 Separator(.)
009:18 Identifier(length)
009:25 Operator(==)
009:28 DecimalIntegerLiteral(0)
009:29 Separator())
010:13 Identifier(System)
010:19 Separator(.)
010:20 Identifier(out)
010:23 Separator(.)
010:24 Identifier(println)
010:31 Separator(()
010:32 StringLiteral("Hello World!")
010:46 Separator())
010:47 Separator(;)
011:09 Keyword(else)
012:13 Identifier(System)
012:19 Separator(.)
012:20 Identifier(out)
012:23 Separator(.)
012:24 Identifier(println)
012:31 Separator(()
012:32 StringLiteral("Hello ")
012:41 Operator(+)
012:43 Identifier(args)
012:47 Separator([)
012:48 DecimalIntegerLiteral(0)
012:49 Separator(])
012:51 Operator(+)
012:53 StringLiteral("!")
012:56 Separator())
012:57 Separator(;)
013:05 Separator(})
015:01 Separator(})
EOF.
```

Guidelines

You cannot discuss Part 5. You may discuss possible approaches to Parts 1-4 with other students, but you **cannot share source code**.

Your solution may use the Java standard library (J2SE 6, see [1]), but **you may not use the regular expression package nor Java-provided tokenizers/scanners to implement Part 4**.

You may use all source code that you have developed for Assignment 1, and also any code from the sample solution provided on the course homepage.

Hints:
- Make use of the Java Collections Framework and in particular the `HashMap`.

Deliverables

The Java source code files for Parts 1-4. Make sure that your code can be compiled manually with `javac`.

A text file (plain text or PDF) that contains the answers to Part 5.

Grading

Your solution will predominantly graded on correctness.

30 points — Part 1.

20 points — Part 2.

10 points — Part 3.

20 points — Part 4.

20 points — Part 5.

Style Guide

Please write clean Java code. See the previous assignment for details. Copy&paste reuse between assignments is ok.

References

[1] http://java.sun.com/javase/6/docs/api/index.html