211hw9

Homework 09: Symbolic Evaluation of Boolean Expressions in Java

Due: 10am Wednesday, March 31, 2010

Overview

Write a Java program `boolSimp.dj1` that reduces boolean expressions (represented in the input and output streams in Scheme-like notation) to simplified form. For the purposes of this assignment, boolean expressions are Scheme expressions constructed from:

- the symbols `T` and `F` denoting the boolean values `true` and `false` ;
- boolean variables (represented by symbols other than `T`, `F`, `!`, `&`, `|`, `>`, and `?` that can be bound to either `true` or `false` .
- the unary function `!` meaning `not` .
- the binary functions `&`, `|`, and `>` denoting `and`, `or`, and `implies`, respectively), and
- the ternary function `?` meaning `if` .

The shorter names `T`, `F`, `!`, `&`, `|`, `>`, and `?` are used instead of `true`, `false`, `not`, `and`, `or`, `implies`, and `if` for notational brevity which matters in very large inputs.

The course staff is providing:

- a Scheme program in the file `boolsimp.ss` equivalent to the Java program that you are required to write;
- a Java "stub" file `boolSimp.dj1` that defines a composite hierarchy of "abstract syntax" tree classes rooted in the class `Form` representing boolean expressions;
- a Java library file `Parser.java` contain a class `Parser` with
    - a `read()` method that reads a boolean expression represented in "Scheme form" and returns the corresponsing Java `Form` abstract syntax tree and
    - a `reduce()` method that composes the visitors you must write in `boolSimp.dj1` to reduce whatever formula the `Parser` instance contains to simplified form.
- a Java "stub" test file `boolSimpTest.java` that includes some rudimentary tests of the code in the `boolSimp.dj1` stub file.

The stub file `BoolSimp.dj1` also includes comments showing you exactly what code you have to write to complete writing your simplifier. Of course, you also need to write corresponding tests and add them to the file `BoolSimpTest.java`.

The file `Parser.java` includes two Parser constructors `Parser(File file)` and `Parser(String form)` for building parsers to parse the boolean expression (in external text form) in the specified `File` or `String`, respectively. To construct a `Parser` for the formula in a file {{}} you must invoke

```
new Parser(new File("<fileName>"));
```

If you omit the `new File(...)` construction in the argument to `Parser` and use `"<fileName>"` instead, you will create a `Parser` for the String `"<file Name>"`. which is interpreted as a simple boolean variable. The `File` input format is important because it enables us to conveniently apply your simplifier to formulas that are thousands of symbols long. As a result, you only have to translate the Scheme code in `boolsimp.ss` into corresponding cleanly-written OO Java code by filling in the gaps in our Java stub file `boolSimp.dj1`. You are expected to appropriately use the composite, interpreter, singleton, and visitor patterns in the code that you write. Since the only stub files that you have to modify are `boolSimp.dj1` and `boolSimpTest.java`, simply submit expanded versions of these files via OwlSpace to submit your assignment. *Warning:* we will run your program on large inputs to determine if you wrote the code correctly. Try using the large test files provided on the course wiki.

We have formatted the test files as a `.java` file rather than a `.dj1` because the Language Levels facility peforms no useful augmentation of JUnit test classes and bypassing the language levels translator avoids some annoying bugs in the implementation of that facility. When using the "Save As" command, please remember to save you file `boolSimpTest.java` as a `.java` file not as a `.dj1` file. The "Save" command always retains the file types of all files.

The Scheme file `boolsimp.ss` includes Scheme functions `parse` and `unparse` to translate Scheme lists into abstract syntax trees and vice-versa. Scheme provides a simple external syntax for lists (in homage to its LISP heritage) but Java does not. Hence the Java `Parser` class works on Java strings instead of lists. The Java visitor class `Print` in the `BoolSimp.java` file performs unparsing of the abstract syntax types `Form` and `IfForm` to type String.

The Scheme parsing functions rely on the following Scheme data definitions.

Given

```
(define-struct ! (arg))
(define-struct & (left right))
(define-struct \| (left right))
(define-struct > (left right))
(define-struct ? (test conseq alt))
```

a `boolExp` is either:

- a boolean constant `true` and `false` ;
- a symbol S representing a boolean variable;
- `(make-Not X)` where X is a `boolExp` ;
- `(make-And X Y) where =`X and Y are `boolExps`;
- `(make-Or X Y) where =`X and Y are `boolExps`;
- `(make-Implies X Y) where {{`X and Y are `boolExps`; or
- `(make-If X Y Z)` where X, Y, and Z are `boolExps`.

Note: The `or` operator must be written as

```
\|
```

in Scheme instead of `|` because `|` is a metasymbol with a special meaning in Scheme.

Description of the Provided Scheme program

Given a parsed input of type `boolExp`, the simplification process consists of following four phases:

- Conversion to `if` form implemented by the function `convert-to-if` .
- Normalization implemented by the function `normalize` .
- Symbolic evaluation implemented by the function `eval` .
- Conversion back to conventional *boolean* form implemented by the function `convert-to-bool` .

These phases are described in detail in HW6.

Hints on Writing Your Java Code

The Java abstract syntax classes include a separate composite hierarchy (called `IfForm` for representing boolean expression as conditionals (the type `if Exp` in `boolsimp.ss`). This representation includes only three concrete variant classes, making it much easier to write the visitors that perform normalization, evaluation, and clean-up.

The visitor pattern is a straightforward but notationally involved alternative to the interpreter pattern.. You can mechanically translate interpreter pattern code to visitor pattern code. (Perhaps IDEs like Eclipse should support such transformations.) The interpreter solution to this assignment is easier to write than the visitor solution described in the preceding program description. If you are still learning Java mechanics, you are encouraged to write an interpreter solution first and translate it (if you can) to visitor form. A perfect interpreter solution will only be penalized 15% versus a perfect visitor solution. If you submit an interpreter solution, your program must conform to class signatures given in the interpreter pattern support code below (just as a visitor solution must conform to the class signatures given in the visitor pattern code below).

The interpreter version of the support code replaces the `ConvertToIf`, `Normalize`, `HeadNormalize`, `Evaluate`, and `Print` visitors by methods named `convertToIf`, `normalize`, `headNormalize`, `eval`, and `print`.

Support Code

Here are the links for the files:

- boolsimp.ss is the reference Scheme program.
- BoolSimp.dj1 is a stub program for a visitor solution.
- BoolSimpTest.java is a stub test file for a visitor solution.
- Parser.java is a parser file for a visitor solution.
- InterpBoolSimp.dj1 is a stub program for an interpreter solution.
- InterpBoolSimpTest.java is a stub test file for an interpreter solution.
- InterpParser.java is a parser file for an interpreter solution.

`InterpParser.java` is distinct from `Parser.java` because the code for the `reduce` method embedded in the parser is different in the two versions.

Sample Input Files

The following files contain large formulas that can be reduced by your simplifier. Only the file named `bigData` require a larger thread stack size than the default on my laptop. I used the JVM argument -Xss64M for the Interactions JVM to get the `bigData` files to run.

- littleData1 -> "T"
- littleData2 -> "T"
- littleData3 -> "(> h (> g (> f (> e (> d (> c (! b)))))))"
- littleData4 -> "(> h (> g (> f (> e (| d (| c (| b a)))))))"
- bigData0 -> "T"
- bigData1 -> "(> j (> i (> h (> g (> f (> e (| d (| c (| b a)))))))))"