Assigned: Feb 14

Due: Mar 10

Assignment

Write a program with 3 modes:

- A generic DPLL solver
- A BNF to CNF converter
- Takes BNF and solves it by running the above two steps

Grading

- out of 100: 35 points for correct DPLL solver behavior
- 55 points for correct BNF to CNF converter behavior
- 10 points for well written code: clear data structures, methods, control flow, etc.
- 10 points of extra-credit if your converter correctly handles parenthesis '(' ')'

Running your program

A program run should look like: (may be slightly different depending on language, note this in your README)

solver [-v] -mode $mode $input-file

- -v is an optional flag for verbose mode (more later)
- -mode is followed by one of: cnf, dpll, solver
- a mode-dependent input file (see next section)
- -mode In mode "cnf" you should expect a BNF input file, convert to CNF and print to console
- In mode "dpll" you should expect a CNF input file, which you solve using the DPLL algorithm printing the solution to console
- In mode "solver" you should expect a BNF input file, run cnf mode, but instead of printing, send the input to dpll mode

Your program should exit gracefully and indicate as best as possible the problem in case of bad arguments or file.

DPLL solver

It should handle arbitrary CNF input and run DPLL as described in class, RN chapter 7.6, and also here. When guessing, use the smallest lexicographic atom and guess True first. The output should be either: an assignment of all atoms that satisfies the clauses, or 'NO VALID ASSIGNMENT'.

- Each line of the input file (when running in dpll mode) represents a sentence, with space separated atoms
- Atoms are only alphanumeric [so no parenthesis or brackets as used in class
- No punctuation is needed as in CNF every sentence is a disjunction
- Except ! the negation operator which may be directly in front of an atom (no spaces)

So, the CNF sentences:

```
P v Q v ! W
! P v Q
W
! P v ! W
```

Would be written in input as simply:

```
P Q !W
!P Q
W
!P  !W
```

And output:

```
P=false
Q=true
W=true
```

In verbose mode you should indicate each step taken, e.g.

```
easy case: unit literal W
easy case: pure literal Q=true
hard case: ...
```

BNF to CNF converter

This portion of the program handles an input file of clauses in BNF, converts to CNF and outputs the clauses in the format understood by the DPLL program.

Run the algorithm as discussed in class and described in RN 7.5 for converting from BNF to CNF.

- Each line of the input file (in cnf or solver mode) represents a BNF sentence to be converted into one or more CNF sentences

- Atoms are only alphanumeric [so no parenthesis or brackets as used in class]
- Operators: !, &, |, =>, <=> are used to represent negate, and, or, implies, if and only if respectively.
- A full grammar also has grouping parenthesis, but you do NOT need to support those unless going for extra-credit

Without parenthesis, the grammar becomes easy to parse without knowledge of compilers and parsing.

Update: You may use yacc/lex or any similar parsing library to read the input file.

Since operators are associated in precedent order: !, &, |, =>, <=>, you can create a parse tree by splitting on the rightmost lowest precedent operator. E.g.

P & ! Q => W <=> A | B & C

If parsed, this would be:

[(P & (!Q)) => (W)] <=> [A | (B & C)]

I will demonstrate this in class. So for a file such as:

```
P & ! Q => W <=> A | B & C
 A => B
 C | B & A
```

Would produce output ready for the DPLL program:

```
!A !P Q W
 !B !C !P Q W
 B A !W
 C A !W
 B A P
 C A P
 B A !Q
 C A !Q
 !A B
 A C
 B C
```

In verbose node you should indicate the progression of each of the 5 conversion steps

More Examples

Your CNF solutions need only produce a set of complete bindings. This is my program in verbose mode showing its work. (your program does not need to do so)

1. BNF produces CNF with verbose conversion + solution.
2. BNF produces CNF with verbose conversion + solution.
3. CNF with verbose solution.
4. CNF with verbose solution.
5. [Extra credit] Needs () and _ handling Australia with verbose solution.

Submission

- On Brightspace, a zip file containing: Code for the two programs
- Makefile/BUILD/pom.xml/etc needed to build your code
- README indicating how to compile and run your code
- In your README include any classmates you consulted with
- As previously indicated, the code should build on department linux machines