Context Free Grammars, Parsing, and Amibuity

CS154

Chris Pollett

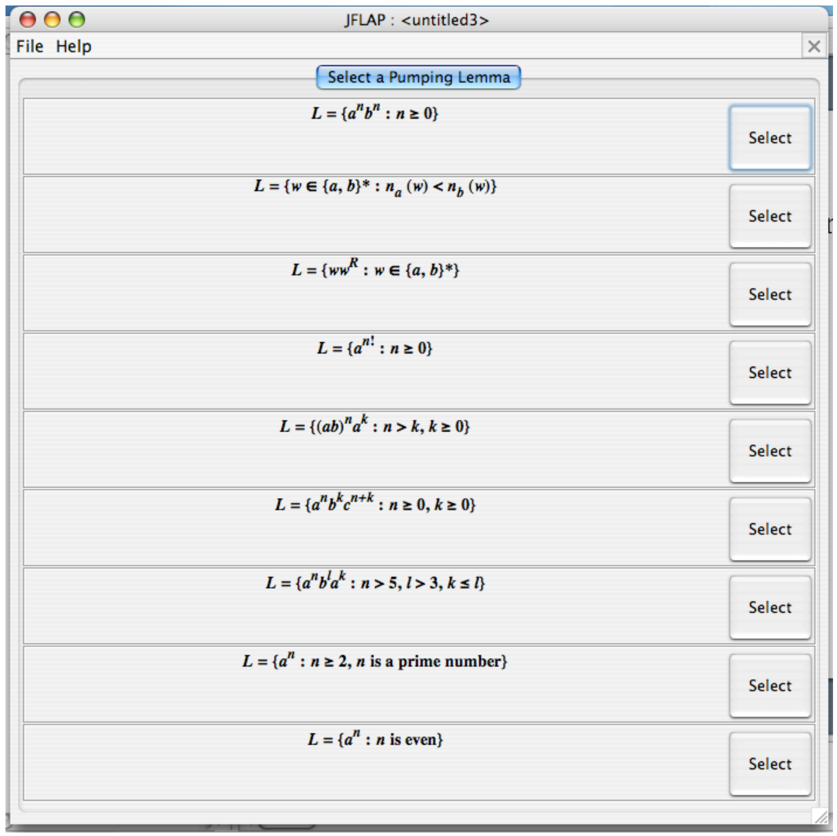
Feb 28, 2007.

Outline

- JFLAP
- Intuitions about Compiler
- Ambiguity
- Left and Rightmost Derivations
- Brute Force Parsing

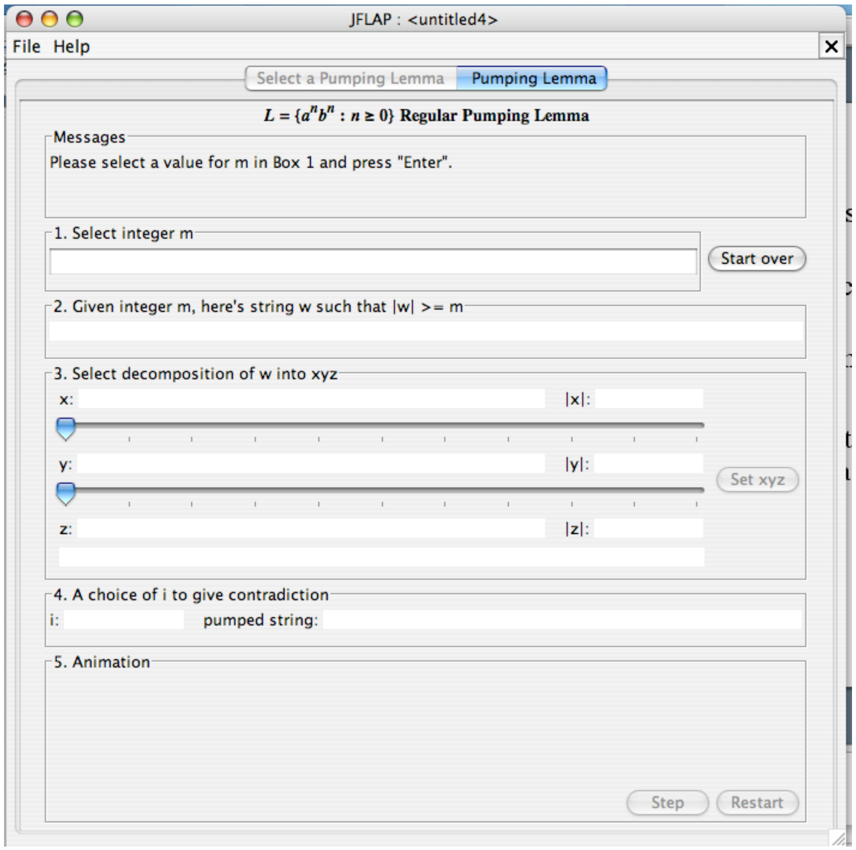
JFLAP and the Pumping Lemma

- JFLAP has a "Regular Pumping Lemma" button.
- Clicking on it gives a window:



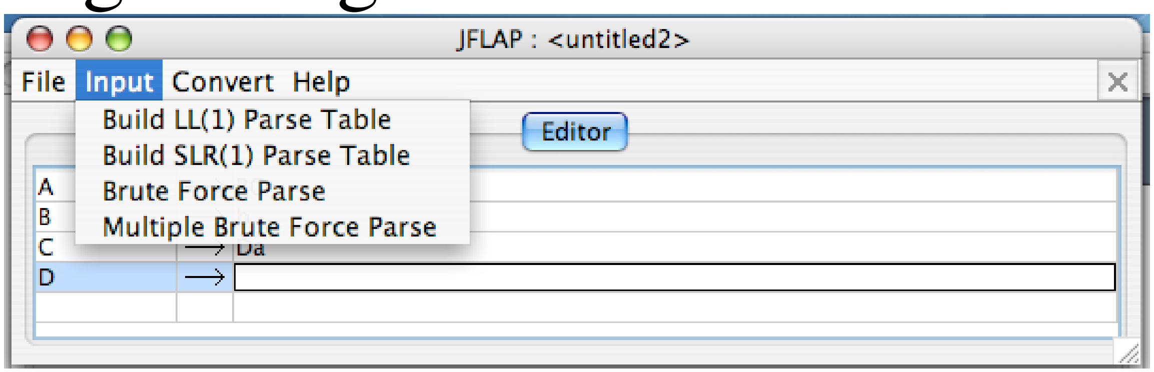
More JFLAP

- Selecting one of these languages lets you play a game versus the computer.
- You choose number of states of machine. i.e., pumping length.
- It selects a string longer than this length.
- You get to choose how it is split
- The computer either tries to find a string that can be pumped but which is not in the language or it loses.



Yet More JFLAP

- JFLAP also has a grammar button.
- If you click on it you can then specify a grammar. If you like, it could be a regular grammar or a CFG.
- JFLAP supports conversions of the grammar to a number of normal forms as well as supports check if a string is in the language of a grammar.



Intuitions about Compilers

- Recall compilers take strings written in a high level language like C or Java and spit out code in machine language.
- So a compilers job is to assign machine code "meaning" to a string written in C.
- The C language might be specified using a CFG.
- For instance, we might have a rule like:

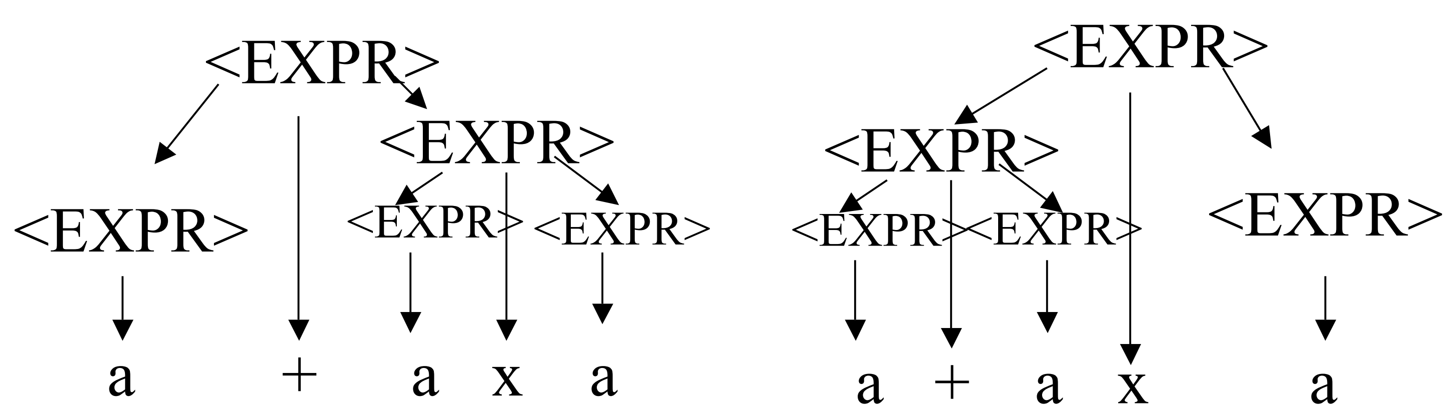
  <while_statement> ::= while <expression> <statement>
- Crudely, we could imagine writing a recursive program to handle this:

```
int parseWhile(String program, int whereParsing, MachineCode whileCode)
{
    MachineCode expressionCode = new MachineCode();
    MachineCode statementCode = new MachineCode();
    whereParsing += 5; //advance passed the keyword "while"
    whereParsing = parseExpression(program, whereParsing, expressionCode);
    whereParsing = parseStatement(program, whereParsing, statementCode);
    // some code to build whileCode from expressionCode and statementCode
    return whereParsing;
}
```

- The underlying assumption for compilation to work is that there can be only one machine code meaning one can give to a given C string.
- Here we have to be careful…

Ambiguity

- Sometime a grammar can generate string in more than one way.
- Such a string will have several different parse trees. As the parse tree is supposed to give us the “meaning” of the string, such a string would have more than one meaning.
- A string with more than one parse tree with respect to a grammar is said to be **ambiguously** derived in that grammar.
- For example, consider <EXPR> --> <EXPR>+ <EXPR>| <EXPR> x <EXPR>|(<EXPR>) |a.
- Then a + a x a can be derived with two different parse trees.



- The left tree probably means a+(a x a); whereas, the right means (a+a) x a

Leftmost and Rightmost Derivations

- We want to formalize the notion of ambiguity in terms of derivations rather than parse trees as derivations are easier to work with syntactically.
- We say that a derivation of a string w in a grammar G is a **leftmost derivation** if at every step the leftmost remaining variable is the one replaced.
- We say that a derivation of a string w in a grammar G is a **rightmost derivation** if at every step the rightmost remaining variable is the one replaced.
- As an example, if our CFG was A-->BAC | λ, B --> b, C-->c. Then A=>BAC=>bAC=>bBACC=>bbACC=>bbCC=>bbcC=>bbcc, is a leftmost derivation of bbcc.
- On the other hand, A=>BAC=>BAc=>BBACc=>BBAcc=>BBcc=>Bbcc=>bbcc would be a rightmost derivation of the same string.
- Intuitively, if we have two ways to expand the leftmost (respectively, rightmost) symbol then the derivation will be ambiguous.

Ambiguity and Leftmost Derivations

- A string w is derived **ambiguously** in G if it has two or more different leftmost derivations (resp. rightmost derivations). A CFG is called **ambiguous** if it generates some string ambiguously.
- In the case of a + a x a, the two left most derivations are:

(1) <EXPR>=> <EXPR> + <EXPR> => a + <EXPR> => a + <EXPR> x <EXPR> => a+ a x <EXPR> => a + a x a;

(2) <EXPR>=> <EXPR> x <EXPR> => <EXPR> + <EXPR> x <EXPR> => a + <EXPR> x <EXPR> => a+ a x <EXPR> => a + a x a;

- There are often many different CFGs for the same language. Even though one of these may be ambiguous some other may be unambiguous. We say a language is **inherently ambiguous** if one can never find an unambiguous CFG for it. The book says without proof that $\{a^ib^jc^k| i=j \text{ or } j=k\}$ is inherently ambiguous.

Brute Force Parsing

- One way to do parsing is by **exhaustive search**.
- We consider each one step derivation from the start variable, then each two step derivation, etc. in turn.
- If we ever see the string we want we accept.
- If all the active derivations involve strings of terminals and variables longer than the string w we are searching for, we halt and reject.
- To handle rules like A->B. Which can give derivations like A=>B=>A, we maintain a list of strings we have already seen. If we repeat, we prune that branch.
- This is an exponential time algorithm; whereas, if we use a normal form for our grammars we can speed things up to be either cubic or in some cases linear time.