CSCI: 4500/6500 Programming Languages

Natural and Programming Languages

Syntactic Structures

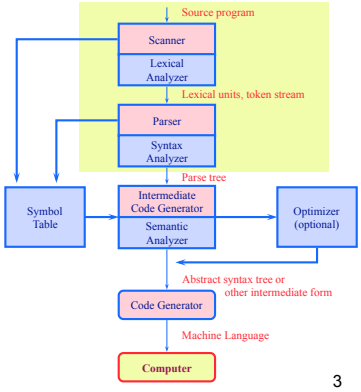
Contributors: Portions of this lecture thanks to: Prof David Evans, U Virginia and Prof Spencer Rugaber, GTech

Review Last Time: Programming Language History

- 50s, 60s: Exciting Time
  - » Invention of: assemblers, compilers, interpreters, first high-level languages, *structured programming*, abstraction, formal syntax, object-oriented programming, LISP, program verification
- 70s, 80s, 90s: Boring Time
  - » Refinement of earlier ideas, better implementations, making theory more practical
  - » A few new/refined ideas: functional languages, data abstraction, concurrent languages, data flow, type theory, etc.
- 00+s: Party Time
  - » A New Environment: Internet, large scale distributed computing, the grid, Java, C#, Maria at UGA
- Alan Kay: "The best way to predict the future is to invent it."

This Week: Programming Language Implementation

- This week and next we will talk about the first two phases of compilation, namely:
  - » Scanning and
  - » Parsing.
- Today the basic concepts next week we talk about parse trees & discuss practicalities



Formal System & Language

Formal System:

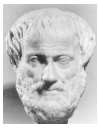
- Set of symbols:
  - » the primitives
- Set of rules for manipulating symbols
  - » Rules of production

What is a *Language* (theoretically)?:

- Formal System + (mapping of sequence of symbols and their meaning)

Linguist's Language

- Description of pairs (S, M)
  - » S is the "sound", or any kind of surface forms, and
  - » M is the meaning.
- Language specifies properties of *sound* and *meaning* and how they relate (Aristotle characterize language as a system than links sound and meaning)
  - Aristotle: 384-322 B.C. Greek philosopher, father of deductive logic, Meta physics, "Physics", teacher of Alexander the Great.



What are languages made of?

- Primitives
  - » The smallest units of meaning, or the *simplest 'surface forms'* (pronunciation).
- Means of Combination (all languages have these)
  - » Like Rules of Production for Formal Systems
  - » Creates 'new' surface forms from the ones you have
- Means of Abstraction (all *powerful* languages have these)
  - » Ways to use simple surface forms to represent more complicated ones

What is *longest* word in the English language?

- *Supercalifragilisticexpialidocious*
  - Popularized by Mary Poppins
  - Oxford English Dictionary, 34 letters
  - Nonsense word meaning fantastic
- Pneumonoultramicroscopicsilicovolcanoconiosis
  - 'a lung disease caused by the inhalation of very fine silica dust', 45 letters (miner's lungs).
  - 207,000+ mitochondrial DNA
- Floccinaucinihilipilification
  - The estimation *of something as worthless* (usage dated since 1741) -- four 'worthless' words with a verb ending.
  - 27 letters, longest non-technical word according first edition of Oxford English Dictionary (floccus - I don't care, I don't make wool, naucum - little value, nihilum - nothing, pilus - a hair, a bit or whit, something small and insignificant, *facio, facere, feci, factus* make or do

Creating longer words

- Floccinaucinihilipilification (previous slide)
  - The estimation of something as worthless, the act of estimating something as useless
- Anti-floccinaucinihilipilification
  - The estimation of something as not worthless
- Antifloccinaucinihilipilification-or
  - The one who does the act of not rendering useless
- Anti- antifloccinaucinihilipilification

Natural Languages

- Are there any non recursive languages?
  - No, we would run out of things to say
- So, we only need to start with a few building blocks and from there we can create infinite things

What are languages made of?

- Primitives
  - The smallest units of meaning, the "*simplest" surface forms.* Lexemes lowest level of meaning.
- Means of Combination (all languages have these)
  - Creates new surface forms from the ones you have
  - Sentences and works on word parts too!
- Means of Abstraction (all powerful languages have these)
  - Ways to use simple surface forms to represent more complicated ones
  - Example: pronouns: "I in English; or Phom, Dichan is the polite way of saying I in Thai depending on gender (Dichan for females).

Primitives/Tokens

- Tokens: Described by regular expressions
  - First phase of compilation process converts strings/lexemes of the programming language to tokens (a representation of the lexeme in the computer)
    - Example: `letter ( letter | digit )*`
  - Can be generated from just three rules/operations:
    - Concatenation
    - Repetition (arbitrary number of times - Kleene closure)
    - Alternation (Choice from a finite set)
  - Corresponds to type-3 grammars in Chomsky hierarchy and is the most restrictive A -> a, A-> aB or A -> Ba
- Many utilities exist that use regular expressions
  - grep (global regular expression print)
    - `grep ^root /etc/passwd`
  - Lex/flex, turn a regular expression of tokens into a scanner, so they are generators (next week)

Means of Combination

- Allow us to say infinitely many things with a finite set of primitives
- We can create *sentences* using primitives
  - But really, in English "words" are really not the 'primitives' since we can create longer words
- How can we *describe* "means of combinations" in the syntax of a language?
  - Computer Scientists:
    - Backus-Normal-Form -> Backus-*Naur*-Form (BNF)

BNF Example

```
Sentence ::= Noun-Phrase  Verb-Phrase
Noun-Phrase ::= Maria | Microsoft
Verb-Phrase := Rocks |  Jumps
```

- What are the terminals?
  - » Maria, Microsoft, Rocks, Jumps
- How many different things can we express with this language?
  - » 4
  - » … but only 1 is true

BNF Example

Sentence ::= Noun-Phrase Verb-Phrase Noun-Phrase
Noun-Phrase ::= Noun | Adjective Noun-Phrase
Noun := Maria | Microsoft | Home | Feet
Adjective := Yellow | Smelly
Verb-Phrase := Skips | Runs | Rocks

- Now we can express infinitely many things with this little language…

Definition of Languages

- Recognizers
  - » Reads input string and accepts or rejects if the string is in the language
  - » Example: Parsers -- the syntax analyzer of a compiler (yacc- yet another compiler compiler)
- Generators
  - » Generate sentences of a language
  - » Example: Grammars are language generators

BNF and Context Free Grammars

- Context Free Grammars
  - » Developed by Noam Chomsky in the 1950s
  - » Define a class of languages called context-free languages (type 2)
- Backus Naur Form (BNF)
  - » A meta-language used to describe another language
  - » Equivalent to context-free grammars

BNF Basics

A BNF grammar consists of four parts:

- *Tokens:* tokens of the language, the terminals
- *Non-terminal symbols:* BNF abstractions in <> brackets
- A *start symbol*
- Grammar: The set of *productions or rules*

BNF details

- The tokens are the smallest units of syntax
  - » Strings of one or more characters of program text
  - » They are atomic: not treated as being composed from smaller parts
- The non-terminal symbols stand for larger pieces of syntax
  - » They are strings enclosed in angle brackets, as in <NP>
  - » They are not strings that occur literally in program text
  - » The grammar says how they can be expanded into strings of tokens
- The start symbol is the particular non-terminal that forms the root of any parse tree for the grammar

BNF Productions (Grammar)

- The productions are the tree-building rules
- Each one has a left-hand side, the separator ::=, and a right-hand side
  - » The left-hand side is a single non-terminal
  - » The right-hand side is a sequence of one or more things, each of which can be either a token or a non-terminal
- A production gives one possible way of building a parse tree: it permits the non-terminal symbol on the left-hand side to have the things on the right-hand side, in order, as its children in a parse tree

Alternatives

- The BNF grammar can give the left-hand side, the separator ::=, and then a list of possible right-hand sides separated by the special symbol |

Example

<exp> ::= <exp> + <exp> | <exp> * <exp> | (<exp>)
| a | b | c

- Equivalent to six productions:

<exp> ::= <exp> + <exp>
<exp> ::= <exp> * <exp>
<exp> ::= (<exp>)
<exp> ::= a
<exp> ::= b
<exp> ::= c

Extensions to BNF - EBNF

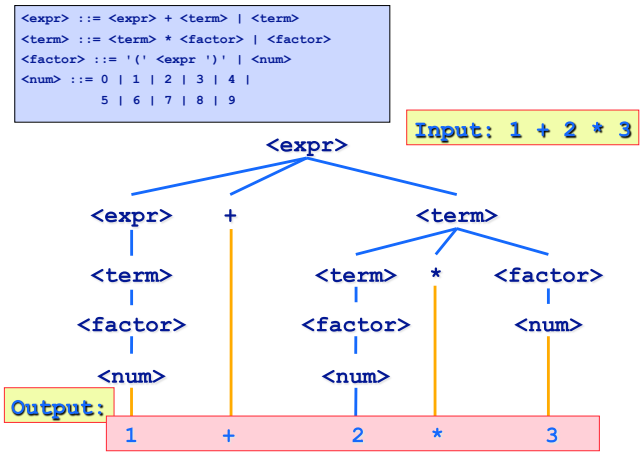
- BNF is sufficient to describe context free languages
- Various extensions and modifications have been made to ease the expression of programming language grammars
  - » The extensions can be bee describe in the original BNF
  - » Collectively these are called EBNF extended BNF

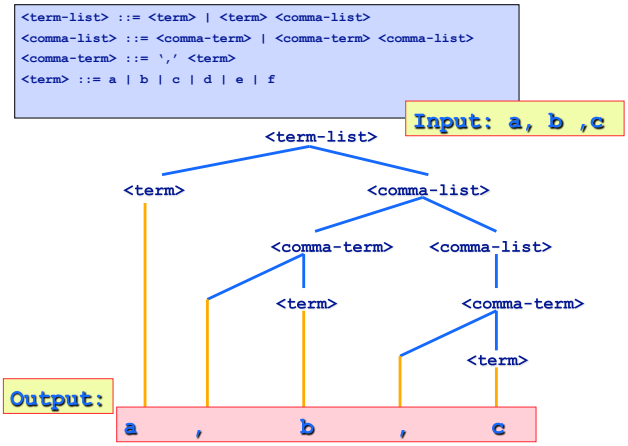
Example EBNF extensions

- Remove brackets for non-terminal
- Replace ::= with →
- Replace vertical bars with spaces
- + for one or more occurrences
  - » EBNF: A → X $(Y)^+$
  - » BNF: A := XB
    - – B := Y | YB
- * for zero or more occurrences

Parse Trees

- Grammars describes '*hierarchical syntactic structures*' so these can be *represented by parse trees* (e.g., a parser generates parse trees).
- Idea:
  - » To build a parse tree, put the start symbol at the root
  - » Add children to every non-terminal, following any one of the productions for that non-terminal in the grammar
  - » Done when all the leaves are tokens
  - » Read off leaves from left to right—that is the string derived by the tree

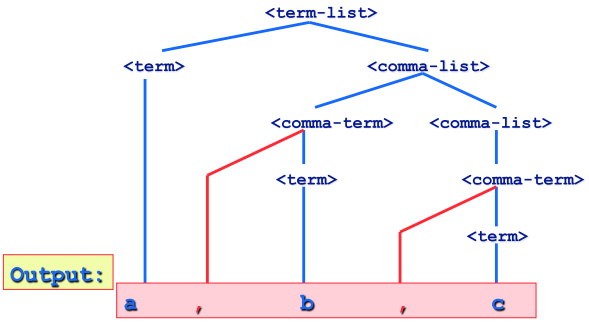




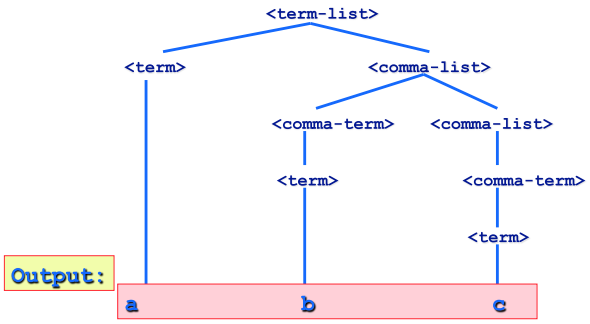
Abstract Syntax Tree

- An abstract syntax tree (AST) describes the elements of a program stripped down to the essentials.
  - » Remove unnecessary components
  - » Some symbols are there not to be interpreted, e.g. punctuations with really no meaning
    - – Example: ",” are there only to tell parser how to build tree
  - » Convert tree from a narrow tree to flat tree
  - » Remove non-essential intermediate non-terminals

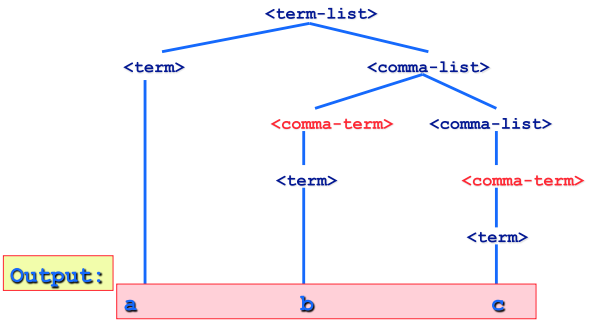
Remove Commas



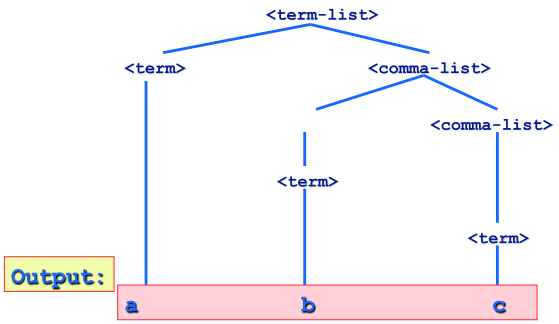
Remove Commas



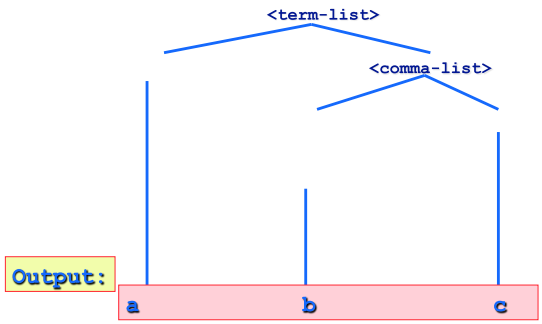
Remove intermediate non-terminals



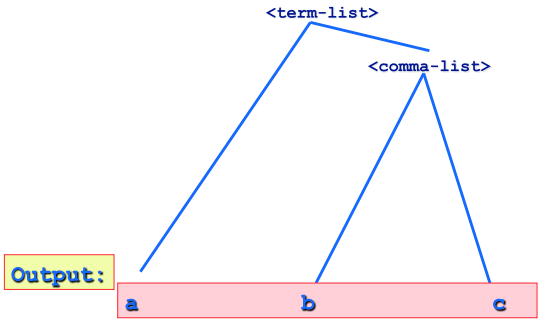
Remove intermediate non-terminals



Remove intermediate non-terminals



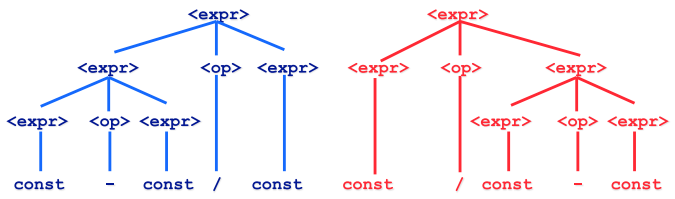
Remove intermediate non-terminals



Ambiguity in Grammars

- Some grammars have more than 1 parse tree for a given string
- Example:

```
<expr> ::= <expr> <op> <expr> | const
<op> ::=  / | -
```



Ambiguity

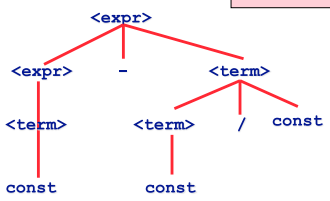
- Compiler often base the semantic on a phrase's parse tree
  - More than one cannot determine the meaning
  - Unless there are some additional non-grammatical information
- Precedence and associatively can be defined outside the grammar.
- Can include it in the grammar to facilitate the compiler to evaluate from the parse tree

Unambiguous Expression Grammar

- If we use the parse tree to indicate precedence levels of operators we cannot have ambiguity

```
<expr> ::= <expr> <op> <expr> | const
<op> ::=  / | -
```

```
<expr> ::= <expr> - <term> | <term>
<term> ::=  <term> / const | const
```

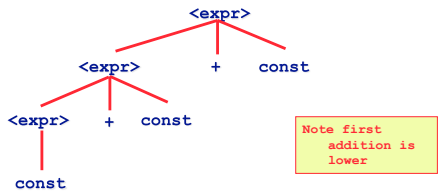


Hint: Higher precedence operators are lower in tree, here "/" has higher precedence than "-"

Associativity

- Operator associativity can also be indicated by a grammar
- Left Associative: 9+5+2 is equivalent to (9 +5) + 2

```
<expr> -> <expr> + <expr> |  const  (ambiguous)
<expr> -> <expr> + const  |  const  (unambiguous)
```



Note first addition is lower

- Project 1 will be posted later tonight - two parts due 1 week and 2 weeks from today
- No floccipoccinihilipilification please!