MACM 300
Formal Languages and Automata

Anoop Sarkar*

`http://www.cs.sfu.ca/~anoop`

**some slides taken from Jason Eisner's course materials*

Applications of Context-free Grammars

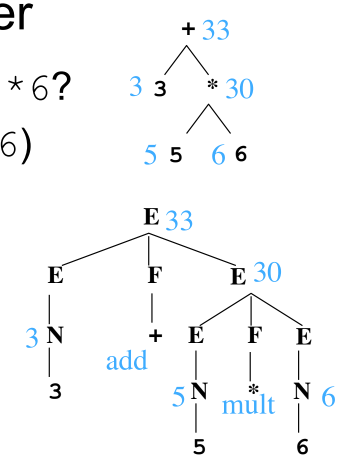
- There are many applications in computer science for context-free grammars
- We will focus here on a few canonical examples from:
  - Structured databases: e.g. XML
  - Compilers and Programming languages
  - Natural language processing
  - Biological sequence analysis

Structured Data: SGML, XML, ...

<DOC><SO> WALL STREET JOURNAL (J), PAGE B5 </SO>
<TXT><p>
New York Times Co. named Russell T. Lewis, 45, president and general manager of its flagship New York Times newspaper, responsible for all business-side activities.
</p><p>
He was executive vice president and deputy general manager. He succeeds Lance R. Primis, who in September was named president and chief operating officer of the parent.
</p></TXT></DOC>

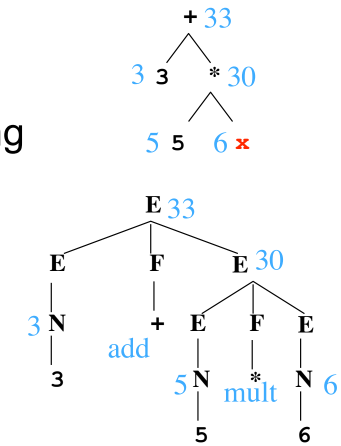
Programming Language Interpreter

- What is meaning of `3+5*6`?
- First parse it into `3+(5*6)`
- Now give a meaning to each node in the tree (bottom-up)



Interpreting in an Environment

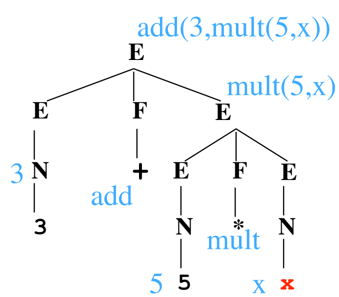
- How about `3+5*x`?
- Same thing: the meaning of `x` is found from the environment (it's 6)



Compiling

- How about `3+5*x`?
- Don't know `x` at compile time
- "Meaning" at a node is a piece of code, not a number

`5*(x+1)-2` is a different expression that produces *equivalent* code
(can be converted to the previous code by optimization)



Logic: Lambda Terms

- Lambda terms:
  - A way of writing "anonymous functions"
    - No function header or function name
    - But defines the key thing: **behavior** of the function
    - Just as we can talk about 3 without naming it "x"
  - Let square = λp p*p
  - Equivalent to int square(p) { return p*p; }
  - But we can talk about λp p*p without naming it
  - Format of a lambda term: λ variable expression

Logic: Lambda Terms

- Lambda terms:
  - Let square = λp p*p
  - Then square(3) = (λp p*p)(3) = 3*3
  - Note: square(x) isn't a function! It's just the value x*x.
  - But **λx** square(x) = λx x*x = λp p*p = square
    (proving that these functions are equal – and indeed they are, as they act the same on all arguments: what is (λx square(x))(y)?)

  - Let even = λp (p mod 2 == 0) a <u>predicate:</u> returns true/false
  - even(x) is true if x is even
  - How about even(square(x))?
  - λx even(square(x)) is true of numbers with even squares
    - Just apply rules to get λx (even(x*x)) = λx (x*x mod 2 == 0)
    - This happens to denote the same predicate as even does

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...

- Suppose we want to write times(5,6)
- Remember: square can be written as λx square(x)
- Similarly, times is equivalent to $\lambda x\ \lambda y$ times(x,y)

Logic: Multiple Arguments

- All lambda terms have one argument
- But we can fake multiple arguments ...

- Claim that times(5)(6) means same as times(5,6)
  - times(5) = ($\lambda x\ \lambda y$ times(x,y)) (5) = λy times(5,y)
    - If this function weren't anonymous, what would we call it?
  - times(5)(6) = (λy times(5,y))(6) = times(5,6)

- So we can always get away with 1-arg functions ...
  - ... which might return a function to take the next argument.

  - We'll still allow times(x,y) as syntactic sugar, though

Logic: Interesting Constants

- Thus, have "constants" that name some of the entities and functions (e.g., times):
  - Gilly - an entity
  - red – a predicate on entities
    - holds of just the red entities: red(x) is true if x is red!
  - loves – a predicate on 2 entities
    - loves(Gilly, Lilly)
    - *Question:* What does loves(Lilly) denote?
- Constants used to define meanings of words
- Meanings of phrases will be built from the constants

Compositional Semantics

- We've discussed briefly what semantic representations should look like.

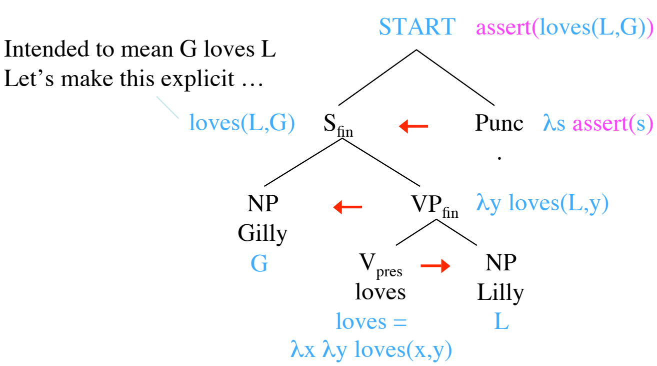
- **But how do we get them from sentences???**

- First - parse to get a syntax tree.
- Second - look up the semantics for each word.
- Third - build the semantics for each constituent
  - Work from the bottom up
  - The syntax tree is a "recipe" for how to do it

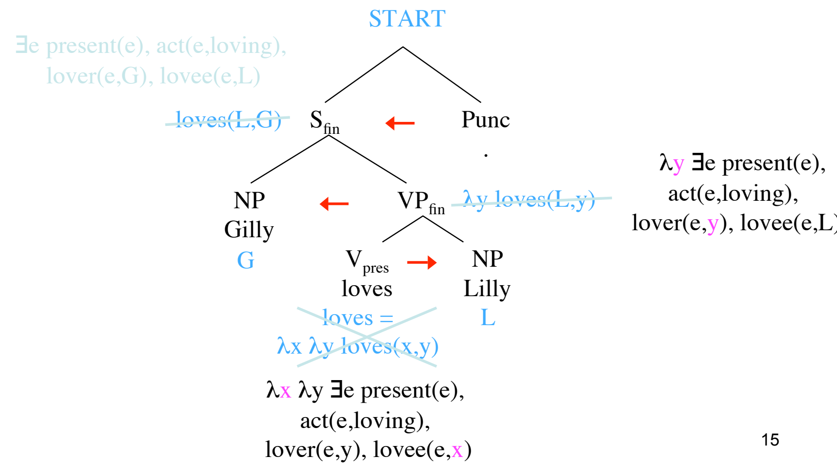
Compositional Semantics

- Now `Gilly loves Lilly` has sem=loves(Lilly)(Gilly)
- In this manner we'll sketch a version where
  - Still compute semantics bottom-up
  - Grammar is in Chomsky Normal Form
  - So each node has 2 children: 1 function & 1 argument
  - To get its semantics, apply function to argument!

Compositional Semantics



Compositional Semantics



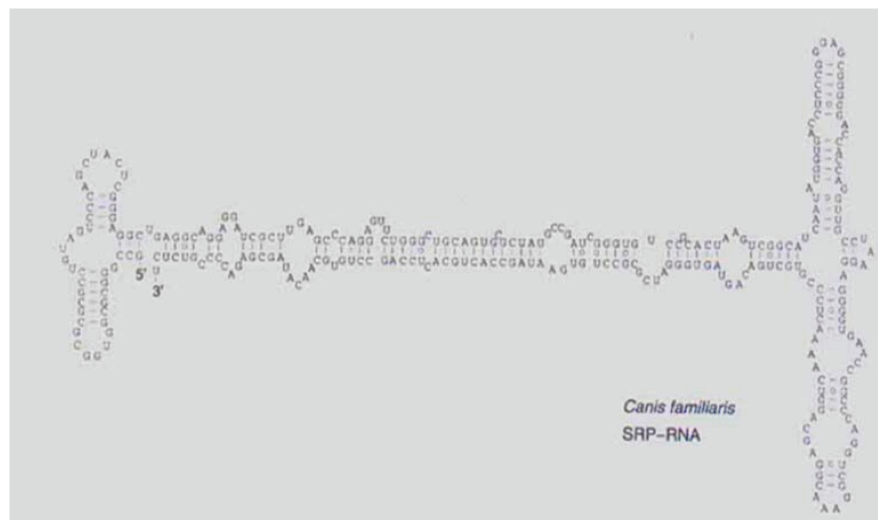


Figure 10.1 *The RNA secondary structure of signal recognition particle (SRP) RNA from the dog,* Canis familiaris.

CFG for RNA secondary structure

$S_0 \rightarrow$ **5' ** S **3'**

S $\rightarrow$ P S | L

P $\rightarrow$ **g** P **c** | **c** P **g** | **a** P **u** | **u** P **a** | L

L $\rightarrow$ **g L | c L | a L | u L | P L | e**