Lecture 7: General and Bottom-Up Parsing

Administrivia

- Project #1 posted. Due 27 Feb.
- HW #3 posted, due next Monday. HW #4 goes back to Friday schedule.
- Test #1: March 10 (in class).

Fixing Recursive Descent

- Can formulate top-down parsing analogously to NFAs.

```
parse (A, S):
   """Assuming A is a nonterminal and S = c1c2...cn is a string,
      return integer k such that A can derive the string c1...ck."""
   Choose production 'A: α1α2···αm' for A (nondeterministically)
   k = 0
   for x in α1, α2, ···, αm:
       if x is a terminal:
          if x == ck+1:
              k += 1
          else:
              GIVE UP
       else:
          k += parse (x, ck+1···cn)
   return k
```

- Assume that the grammar contains one production for the start symbol: `p:` $\gamma\dashv$.
- We'll say that a call to `parse` returns a value if *some* set of choices for productions (the blue step) would return a value (just like NFA).
- Then if `parse(p, S)` returns a value, `S` must be in the language.

Making a Deterministic Algorithm

- If we had an infinite supply of processors, could just spawn new ones at each "Choose" line.
- Some would *give up*, some loop forever, but on correct programs, at least one process would *get* through.
- To do this for real (say with one processor), need to keep track of all possibilities systematically.
- This is the idea behind Earley's algorithm:
  - Handles any context-free grammar.
  - Finds all parses of any string.
  - Runs in $O(N^3)$ time for ambiguous grammars, $O(N^2)$ time for "nondeterministic grammars", or $O(N)$ time for deterministic grammars (such as accepted by Bison).

Earley's Algorithm: I

- First, reformulate to ditch the loop. Assume the string $S = c_1 \cdots c_n$ is fixed.

```
parse (A: α • β, s, k):
    """Assumes A: αβ is a production in the
       grammar,  0 <= s <= k <= n, and α can produce the string
       c_{s+1} ··· c_k. Returns integer j such that β can
       produce c_{k+1} ··· c_j."""
    if β is empty:
       return k
    Assume β has the form yδ
    if y is a terminal:
       if y == c_{k+1}:
           return parse(A: αy • δ, s, k+1)
       else
           GIVE UP
    else:
       Choose production 'y:  κ' for y (nondeterministically)
       j = parse(y: •κ, k, k)
       return parse (A: αy • δ, s, j)
```

- Now do all possible choices that result in such a way as to avoid redundant work ("nondeterministic memoization").

Chart Parsing

- Idea is to build up a table (known as a *chart* of all calls to `parse` that have been made.
- Only one entry in chart for each distinct triple of arguments (`A:` $\alpha \bullet \beta$, s, k).
- We'll organize table in columns numbered by the k parameter, so that column k represents all calls that are looking at c_{k+1} in the input.
- Each column contains entries with the other two parameters: [`A:` $\alpha \bullet \beta$, s], which is called an *item*.
- The columns, therefore, are *item sets*.

Example

Grammar

```
p : e '⊣'
e : s I  |  e '+' e
s : '-' |
```

Input String

```
- I + I ⊣
```

Chart. Headings are values of k and c_{k+1} (raised symbols).

0 (-)	1 (I)	2 (+)	3 (I)
a. p: ●e '⊣', 0	*e.* s: '-'●, 0	*g.* e: s I●, 0	*i.* e: e '+' ●e, 0
b. e: ●e '+' e, 0	*f.* e: s●I, 0	*h.* e: e ●'+' e, 0	*j.* e: ●s I, 3
c. e: ●s I, 0			*k.* s: ●, 3
d. s: ●'-', 0			*l.* e: s ●I, 3

4 (⊣)	5
m. e: s I●, 3	*p.* p: e '⊣' ●, 0
n. e: e '+' e●, 0	
o. p: e●'⊣', 0	

Example, completed

- Last slide showed only those items that survive and get used. Algorithm actually computes dead ends as well (unlettered, in red).

0	–	*1*	I	*2*	+	*3*	I
a. p: •e '⊣', 0		*e.* s: '-'•, 0		*g.* e: s I•, 0		*i.* e: e '+' •e, 0	
b. e: •e '+' e, 0		*f.* e: s•I, 0		*h.* e: e •'+' e, 0		*j.* e: •s I, 3	
c. e: •s I, 0						*k.* s: •, 3	
d. s: •'-', 0						*l.* e: s •I, 3	
s: •, 0						s: •'-', 3	
						e: •e '+' e, 3	

4	⊣	*5*
m. e: s I•, 3		*p.* p: e '⊣' •, 0
n. e: e '+' e•, 0		
o. p: e•'⊣', 0		
e: e •'+' e, 3		

Adding Semantic Actions

- Pretty much like recursive descent. The call `parse(A:` $\alpha \bullet \beta$`, s, k)` can return, in addition to j, the semantic value of the `A` that matches characters $c_{s+1} \cdots c_j$.
- This value is actually computed during calls of the form parse(`A:` $\alpha' \bullet$, s, k) (i.e., where the β part is empty).
- Assume that we have attached these values to the nonterminals in α, so that they are available when computing the value for A.

Ambiguity

- Ambiguity only important here when computing semantic actions.
- Rather than being satisfied with a single path through the chart, we look at *all* paths.
- And we attach the *set* of possible results of `parse(Y: •`κ`, s, k)` to the nonterminal Y in the algorithm.