Follow

Motivation

- Consider top down parsing
- We are at a node representing nonterminal X
- Need to decide which production to expand X into
- But we also need to know when we're *done* with X

Follow

- Example: How do we know when we are done parsing a given expansion?
- stmt $\rightarrow$ assign | if-stmt
  assign $\rightarrow$ ID = expr
  if-stmt $\rightarrow$ IF expr stmt
  expr $\rightarrow$ NUM expr'
  expr' $\rightarrow$ + NUM expr' | λ
- Suppose we have these tokens:
  if 5 + 3 + 2 a=3
- How will parser know where expr ends and stmt begins?
  - Easy for a human to do, but we want an automated approach

Definition

- Def: follow[X]: Set of all tokens that can appear in input immediately after that part that corresponds to expansion of X.
  - I.e., they are the tokens that can *follow* an expansion of X
  - Note: Terminals have no entries in follow[] because they are never expanded
- Idea: If we are doing expansion for nonterminal N and we need to decide if we're done, see if next token is in follow[N].
- If so: We are done with N.

Example

- $S \rightarrow A\ x \mid B\ y$
  $A \rightarrow a \mid b$
  $B \rightarrow c \mid d$
- Follow[A] = {x}
  - Why? Because any time we have an A, must be followed with an x when expansion done
- Follow[B] = {y}
  - Same reason
- Follow[S] = end of string
  - Usually denoted with $

Example

- $S \rightarrow A\ x \mid B\ y$
  $A \rightarrow A\ a \mid b$
  $B \rightarrow A\ z \mid c$
- Follow[A] = {a,x,z}
- Follow[B] = {y}
  - We only care about what follows B's entire chain of derivation.
  - Ex: $S \rightarrow \underline{B}\ y \rightarrow \underline{A\ z}\ y \rightarrow \underline{A\ a\ z}\ y \rightarrow \underline{A\ a\ a\ z}\ y \rightarrow \underline{b\ a\ a\ z}\ y$
- Follow[S] = {$}

Example

- S $\rightarrow$ y A | x
  A $\rightarrow$ b b | c | A d
- Follow[S] = {$}
- Follow[A] = {d,$}
  - Since A is at end of one of S's derivations, it can be followed by whatever follows S

Example

- $S \rightarrow A\ x \mid A\ B$

  $A \rightarrow a$

  $B \rightarrow b \mid \lambda$
  - Follow[B]={$}
  - Follow[S]={$}
  - Follow[A]={x,b,$}
    - Why $? We can derive:
      $S \rightarrow \underline{A}\ B \rightarrow \underline{A} \rightarrow \underline{a}$
    - So the thing that A expanded into (“a”) is followed by the end of the string

Example

- $S \to A\,x \mid A\,B \mid A\,B\,C\,D\,y \mid A\,B\,C\,z$
  D
  $A \to a$
  $B \to b \mid C \mid \lambda$
  $C \to c \mid \lambda$
  $D \to S\,d \mid w \mid \lambda$

- $A : a\,c\,b\,\$\,w\,y\,x\,z\,d$
- $S : \$\,d$
- $B : a\,c\,\$\,w\,y\,z\,d$
- $C : a\,c\,\$\,w\,y\,z\,d$
- $D : y\,\$\,d$

Computing Follow

- When computing first[x] we only looked at productions with left hand side of x
- When computing follow[x], we must look at all productions
  - If x appears in a production: That production somehow contributes to follow[x]
- Assume that nullable and first are known

Algorithm

```
follow = {}
follow[start_symbol]=set( ["$"] )
do
    for all nonterminals N:
        for all productions P with lhs of N:
            for i in range(len(P)):
                x=P[i]
                if x is nonterminal:
                    for y in P[i+1:]
                        add first[y] to follow[x]
                        if y not in nullable:
                            break
                    if we didn't break out:
                        add follow[N] to follow[x]
until follow stabilizes
```

Example

- Grammar:
  $S \rightarrow a \mid A\ b\ B \mid B\ C\ e$
  $A \rightarrow x \mid C\ y$
  $B \rightarrow C\ C \mid q$
  $C \rightarrow \lambda \mid w$

- Firsts:
  A: w , x , y
  B: q , w
  C: w
  S: a , e , q , w , x , y
- Follows:
  A: b
  B: $, e , w
  C: $, e , w , y
  S: $

Example

- Now we can go back to our example from before:

stmt → assign | cond
assign → ID = expr
cond → IF expr stmt
expr → NUM expr'
expr' → + NUM expr' | λ

Nullable:
expr'

Firsts:
stmt: NUM, IF
assign: ID
cond: IF
expr: NUM
expr': +

Follows:
stmt: $
assign: $
cond: $
expr: $, NUM, IF
expr': $, NUM, IF

Parse

- Input: if 5 + 3 + 2 a=3
- Tokens: IF NUM + NUM + NUM ID = 3
- Parsing:
  - We know we're expanding start ('stmt')
  - Two alternatives
  - Only one begins with "IF" (use first to determine which one to use)
  - So we're working on cond $\rightarrow$ IF expr stmt

Parse

- Match the IF with the input
- Remaining input: NUM + NUM + NUM ID = 3
- We know next thing is expr
- We know that follow[expr] contains ID
- So when we reach the ID, we're done with expr and we can go back to working on the cond

Assignment

- Extend your code to compute the follow set for a grammar
- Your program must work with the test suite

Sources

- Alfred Aho, Monica Lam, Ravi Sethi, Jeffrey D. Ullman. Compilers: Principles,Techniques and Tools (2nd ed).
- K. Louden. Compiler Construction: Principles and Practice.

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org