Grammar Analysis

- Automated construction of syntax analysers
  - simple checks on consistency of a grammar
  - algorithms for constructing lookahead sets (used in recursive descent parsing).

Themes

1. Construction of compilers
2. Use of tools (lex and yacc) to construct compilers — "compiler-compilers"
3. Construction of compiler-compilers

A grammar

Expression ::= Expession Operator primanyExpression
| primaryExpression

primaryExpression ::= Operator primaryExpression

- Check that all symbols (terminal and nonterminal) can be derived from the sentence symbol.
- Check that all nonterminal symbols can generate a terminal string.

Example $G = (\{A,B,C,D,S\}, \{a,b\}, P, S)$ where P is

$S ::= AB \mid C$

$A ::= a \mid aA$

$B ::= bA$

$C ::= bCa$

$D ::= aD \mid b$

Nomenclature

A	nonterminal symbol
α, β, γ	strings of nonterminal and/or terminal symbols

X	terminal or nonterminal symbol
u, v, w	strings of terminal symbols.

directly generates

A grammar G defines a binary relation $\rightarrow$ (read directly generates) on strings. This relation is the smallest relation such that, for all A, α, β, γ

$$\alpha A \beta \rightarrow \alpha \gamma \beta \quad \equiv \quad A ::= \gamma \text{ is a production in } G$$

generates

$\rightarrow^*$ is the reflexive, transitive closure of $\rightarrow$

So

$$\alpha \rightarrow^* \beta$$

$$\equiv \quad \alpha = \beta \ \vee \ \exists(\gamma :: \ \alpha \rightarrow \gamma \ \wedge \ \gamma \rightarrow^* \beta)$$

Definitions

$$\text{nonempty.}X \quad \equiv \quad \exists(u :: \ X \rightarrow^* u)$$

$$\text{derivable.}X \quad \equiv \quad \exists(u, v :: \ S \rightarrow^* uXv)$$

(ne.X abbreviates nonempty.X)

$$\left.\begin{array}{l} ne.a = true \\ ne.b = true \end{array}\right\} \text{ terminal symbols}$$

$$ne.S \Leftarrow (ne.A \wedge ne.B) \vee ne.C \qquad S ::= AB \mid C$$

$$ne.A \Leftarrow ne.a \vee (ne.a \wedge ne.A) \qquad A ::= a \mid aA$$

$$ne.B \Leftarrow ne.b \wedge ne.A \qquad B ::= bA$$

$$ne.C \Leftarrow ne.b \wedge ne.C \wedge ne.a \qquad C ::= bCa$$

$$ne.D \Leftarrow (ne.a \wedge ne.D) \vee ne.b \qquad D ::= aD \mid b$$

Note:

$$ne.C \Leftarrow ne.b \wedge ne.C \wedge ne.a$$

is satisfied by both

$$ne.C = false$$

and $ne.C = true$

The solution $ne.C = false$ is the <u>least</u> solution (ordering false and true by false < true).

Solving the Equations

Least solution is found by successive approximation:
update lhs by substituting best-known values in rhs.

```
for each nonterminal A do ne.A := false;
for each terminal symbol t do ne.t := true;
{ Invariant : ∀(X :: ne.X ⇒ ∃(u :: X →* u))
  Variant :  #nonterminals - n
    where  n = #(nonterminals A s.t. ne.A) }
repeat
    update lhs values
until  no change in #(nonterminals in A s.t. ne.A)
```

update lhs values:

```
for each production p do
    ne.(lhs.p)
      := ne.(lhs.p) ∨ ∀(i: 0 ≤ i < #(rhs.p)
                        : ne.((rhs.p)_i) )
```

Nomenclature

lhs.p	left hand side of production p
rhs.p	right hand side —"—
#(rhs.p)	length of rhs.p
$(rhs.p)_i$	i th symbol (counting from 0) on rhs of p.

Example

S :: AB | C A ::= a | aA
B ::= bA C ::= bCa
D ::= aD | b

Initially: ne.a = true, ne.b = true
otherwise, ne.X = false

1st iteration: ne.A := true, ne.D := true

2nd iteration: ne.B := true

3rd iteration: ne.S := true

4th iteration: no change

(Note: no. of iterations depends on ordering of productions)

Correctness

Termination: #(nonterminals A s.t. ne.A)
never decreases. (Once ne.A = true it remains true)

Conditional correctness:

The invariant property is

$$ne.X \Rightarrow \exists(u :: X \rightarrow^* u)$$

(where u ranges over terminal strings)

It is easy to see that this is initially true and is maintained by the loop body.

Claim is that on termination: $ne.X = \exists(u :: X \rightarrow^* u)$.

Sketch proof of termination claim.

Introduce count k of the number of times the loop has been executed.

Strengthen invariant to:

$$ne.X \quad = \quad \exists(u :: X \to^{\leq k} u)$$

Suppose loop terminates after $k+1$ iterations.

After k iterations: $\forall(X :: ne.X = \exists(u :: X \to^{\leq k} u))$

" $k+1$ " : $\forall(X :: ne.X = \exists(u :: X \to^{\leq k+1} u))$

So, since termination condition is "no change".

$$\forall(X :: \exists(u :: X \to^{\leq k} u) = \exists(u :: X \to^{\leq k+1} u))$$

Prove hence: $\forall(X :: \exists(u :: X \to^{\leq k} u) = \exists(u :: X \to^{*} u))$

Automatic Construction of Recursive Descent Parsers

- Basic structure of a recursive descent parser is identical to the structure of the underlying grammar.

- Only non-trivial task is the construction of the LOOKAHEAD sets needed to choose among productions.

nullable

$$\text{nullable}(\alpha) \equiv \alpha \rightarrow^* \varepsilon$$

Equations

$$\text{nullable}(t) \equiv \text{false}$$

$$\text{nullable}(A) \equiv \exists(p : \text{lhs}.p = A : \text{nullable}(\text{rhs}.p))$$

$$\text{nullable}(\varepsilon) \equiv \text{true}$$

$$\text{nullable}(X\alpha) \equiv \text{nullable}(X) \wedge \text{nullable}(\alpha)$$

FIRST (starters — Watt and Brown)

$$\text{FIRST}(\alpha) = \{ t \mid \exists(u :: \alpha \rightarrow^* tu) \}$$

Equations

$$\text{FIRST}(t) = \{t\}$$

$$\text{FIRST}(A) = \cup(p : \text{lhs}.p = A : \text{FIRST}(\text{rhs}.p))$$

$$\text{FIRST}(\varepsilon) = \emptyset$$

$$\text{FIRST}(X\alpha) = \text{FIRST}(X) \cup (\text{if } X \rightarrow^* \varepsilon \text{ then FIRST}(\alpha) \text{ else } \emptyset)$$

FOLLOW

$$\text{FOLLOW}(A) = \{ t \mid \exists (u, v :: Z \to^* uAtv) \}$$

Equations

An <u>rhs occurrence</u> of A, written $B \to \alpha A . \beta$, consists of a production of the form $B \to \alpha A \beta$ together with information on the position of A on the rhs (indicated by the dot).

$$\text{FOLLOW}(S) = \{\$\} \cup \bigcup (\text{rhs occurrences } B \to \alpha S . \beta :: \text{FIRST}(\beta \cdot \text{FOLLOW}(B)))$$

$$\text{FOLLOW}(A) = \bigcup (\text{rhs occurrences } B \to \alpha A . \beta :: \text{FIRST}(\beta \cdot \text{FOLLOW}(B)))$$

LOOKAHEAD

$$\text{LOOKAHEAD}(A ::= \alpha) = \{ t \mid \exists (u, \beta : Z \to^* uA\beta : t \in \text{FIRST}(\alpha\beta)) \}$$

Equations

$$\text{LOOKAHEAD}(A ::= \alpha) = \text{FIRST}(\alpha \cdot \text{FOLLOW}(A))$$

Solving the Equations

In each case, use successive approximation

```
initialise approximate solution ;
repeat
    update lhs of equation
    using best known approximation to rhs
until  no change in approximations
```

A	nonterminal
t	terminal
X, Y	terminal or non-terminal
p	production
lhs.p	lefthand side of production p
rhs.p	righthand side of production p
u, v	string of terminals
α, β	string of terminals and/or non-terminals

Assume grammar has a production $Z \rightarrow S\$$ where S is the start symbol. "$" marks the right end of the input. Z is not included in the calculation of FOLLOW sets.

Nullable

```
for each X do nullable.X := false ;
{ Invariant:  ∀(X :: nullable.X ⇒ X →* ε)
  Variant:    #(X :: ¬ nullable.X)  }
repeat
    for each p do
        begin let rhs.p = X1, X2, ..., Xn , lhs.p = A ;
          nullable.A :=
              nullable.A ∨ ∀(i: 1≤i≤n : nullable.Xi)
        end
until no change occurs in nullable
```

FIRST

Initialisation

```
for each t do FIRST.t := {t} ;
foreach A do FIRST.A := ∅
```

Loop body

```
for each p do
  begin let lhs.p = A, rhs.p = X1, ..., Xn ;
        for i := 1 to n do
            begin FIRST.A := FIRST.A ∪ FIRST.Xi ;
                  if ¬ nullable.Xi then break
            end
  end
```

FOLLOW

Initialisation: `foreach X do FOLLOW.X := ∅;`
`FOLLOW.S := {$} /* end-of-file marker */`

Loop body:

```
foreach p do
  begin let A = lhs.p, X1 ... Xn = rhs.p ;
  f := FOLLOW.A ; k := n ;
  {Invariant: f = FIRST(Xk+1 ... Xn · FOLLOW.A)}
  while k ⩾ 1 do
    begin FOLLOW.Xk := FOLLOW.Xk ∪ f ;
      if nullable.Xk then f := f ∪ FIRST.Xk
                     else f := FIRST.Xk ;
      k := k+1
    end
  end
```