

# CS 460

Programming Languages

Fall 2008

Dr. Watts

(Week 8)



## Prolog Example

```
/* facts about size*/
```

```
size(greatdane,extralarge).  
size(labrador,large).  
size(boxer,large).  
size(tibetan,medium).  
size(lhasaapso,small).  
size(poodle,small).  
size(chihuahua,extrasmall).  
size(maltese,extrasmall).  
size(yorkie,extrasmall).
```

```
/* facts about breed */
```

```
breed(sammy,chihuahua).  
breed(zoe,lhasaapso).  
breed(rocky,boxer).  
breed(shasta,yorkie).  
breed(ginger,lhasaapso).  
breed(marmaduke,greatdane).
```

```
/* rules */
```

```
bigdog(Name) :- breed(Name,Breed),  
                size(Breed,extralarge).  
bigdog(Name) :- breed(Name,Breed),  
                size(Breed,large).
```

```
littledog(Name) :- breed(Name,Breed),  
                  size(Breed,small).  
littledog(Name) :- breed(Name,Breed),  
                  size(Breed,extrasmall).
```



# Prolog Queries



```
[tiawatts@cwolf server ~/cs460f08]$ gprolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- ['dogs.pl'].
compiling /home/faculty/tiawatts/cs460f08/dogs.pl
for byte code...
/home/faculty/tiawatts/cs460f08/dogs.pl compiled,
32 lines read - 2727 bytes written, 28 ms

(1 ms) yes
| ?- breed(sammy,Breed).

Breed = chihuahua

yes
| ?- breed(Who,lhasaapso).

Who = zoe ? ;

Who = ginger ? ;

(1 ms) no
| ?- breed(Who,labrador).

no

| ?- bigdog(rocky).

yes
| ?- littledog(Who).

Who = zoe ? ;

Who = ginger ? ;

Who = sammy ? ;

Who = shasta ? ;

(1 ms) no
| ?- exit.
```

# What is Parsing?



- Process of analyzing syntax
- Determining if the order of the tokens generated for the lexemes of the input are in a legal order according to some grammar
- Creation of a parse tree
  - Explicit or implicit
- Error recovery
  - When an error is detected, the parser must get back to a normal state and continue analysis of the input
- Basis for translation



# LL(1) Grammar for a Small Programming Language



T = {begin, end, ,, =, A, B, C, +, \*}  
N = {<program>, <stmt\_list>, <stmt>, <var>,  
      <stmt\_tail>, <expression>, <expr\_tail>}  
Start = <program>

P =  
{  
1. <program> → **begin** <stmt\_list> **end**  
2. <stmt\_list> → <stmt> <stmt\_tail>  
3. <stmt\_tail> → ; <stmt\_list>  
4. <stmt\_tail> → λ  
5. <stmt> → <var> = <expression>  
6. <var> → **A**  
7. <var> → **B**  
8. <var> → **C**  
9. <expression> → <var> <expr\_tail>  
10. <expr\_tail> → + <var> <expr\_tail>  
11. <expr\_tail> → \* <var> <expr\_tail>  
12. <expr\_tail> → λ  
}

Example 1:

```
begin
  A = B + C;
  C = A * B
end
```

Example 2:

```
begin
  A = B + A * C;
  C = A * B;
end
```

## Parsing of Example 1



From main parsing routine, call <program> function  
  From <program>, match **begin**; call <stmt\_list> function  
    From <stmt\_list>, see **A** call <stmt> function  
      From <stmt>, see **A** call <var> function  
        From <var>, match **A**; return  
      From <stmt>, match =; see **B** call <expression> function  
        From <expression>, see **B** call <var> function  
          From <var>, match **B**; return  
        From <expression>, see + call <expr\_tail> function  
          From <expr\_tail>, match +; see **C** call <var> function  
            From <var>, match **C**; return  
          From <expr\_tail>, see ; call <expr\_tail>  
            From <expr\_tail>, see ; return  
          From <expr\_tail>, return  
        From <expression>, return  
    ...  
    From <program>, match end; return  
From main parsing routine, print error count and terminate

## Recursive Descent Parser for a Small Programming Language



1) `<program>` → `begin <stmt_list> end`

```
program ()
{
    if (current_token == begin)
    { // rule 1
        get_next_token;
        call stmt_list;
        if (current_token == end)
            get_next_token;
        else
            call error_routine;
    }
    else
        call error_routine;
    return;
}
```

## Recursive Descent Parser for a Small Programming Language



2) `<stmt_list>` → `<stmt> <stmt_tail>`

```
stmt_list ()
{ // rule 2
    call stmt;
    call stmt_tail;
    return;
}
```

## Recursive Descent Parser for a Small Programming Language



3)  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$

4)  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$

```
stmt_tail ()
{
    if (current_token == ;)
    { // rule 3
        get_next_token;
        call stmt_list;
    }
    else if (current_token == end)
    { // rule 4
    }
    else
        call error_routine;
    return;
}
```

## Recursive Descent Parser for a Small Programming Language



5)  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$

```
stmt ()
{ // rule 5
    call var;
    if (current_token == =)
    {
        get_next_token;
        call expression;
    }
    else
        call error_routine;
    return;
}
```

## Recursive Descent Parser for a Small Programming Language



6) <var> → A  
7) <var> → B  
8) <var> → C

```
var ()
{ // rule 5
  if (current_token == A)
  { // rule 6
    get_next_token;
  }
  else if (current_token == B)
  { // rule 7
    get_next_token;
  }
  else if (current_token == C)
  { // rule 8
    get_next_token;
  }
  else
    call error_routine;
  return;
}
```

## Recursive Descent Parser for a Small Programming Language



9) <expression> → <var><expr\_tail>

```
expression ()
{ // rule 9
  call var;
  call expr_tail;
  return;
}
```

## Recursive Descent Parser for a Small Programming Language



10)  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$   
11)  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$   
12)  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$

```
expr_tail ()
{
    if (current_token == +)
    { // rule 10
        get_next_token;
        call var;
        call expr_tail;
    }
    else if (current_token == *)
    { // rule 11
        get_next_token;
        call var;
        call expr_tail;
    }
    else if (current_token == ; or current_token == end)
    { // rule 12
    }
    else
        call error_routine;
    return;
}
```

## Context Free Grammar Definition



- Given a Context Free Grammar of the form:
  - Terminals =  $\{T_1, T_2, T_3, \dots\}$
  - Non-terminals =  $\{\langle nt_1 \rangle, \langle nt_2 \rangle, \langle nt_3 \rangle, \dots\}$
  - A Start symbol from the set of non-terminals
  - A set of Production rules of the form  
 $\langle nt_i \rangle \rightarrow \text{string of } T \text{ and } \langle nt \rangle \text{ symbols}$



## First and Follow Sets

- **Firsts**
  - A terminal symbol  $T_i$  is a member of the First Set of non-terminal symbol  $\langle nt_j \rangle$  if  $T_i$  can become the first terminal symbol in a complete expansion of  $\langle nt_j \rangle$ .
- **Follows**
  - A terminal symbol  $T_i$  is a member of the Follow Set of non-terminal symbol  $\langle nt_j \rangle$  if  $T_i$  can become the first terminal symbol immediately following a complete expansion of  $\langle nt_j \rangle$ .



## Calculating First and Follow Sets – Rule 1

1.  $\langle \text{program} \rangle \rightarrow \mathbf{begin} \langle \text{stmt\_list} \rangle \mathbf{end}$
  2.  $\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle \langle \text{stmt\_tail} \rangle$
  3.  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$
  4.  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$
  5.  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$
  6.  $\langle \text{var} \rangle \rightarrow \mathbf{A}$
  7.  $\langle \text{var} \rangle \rightarrow \mathbf{B}$
  8.  $\langle \text{var} \rangle \rightarrow \mathbf{C}$
  9.  $\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  10.  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  11.  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  12.  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$
1. For each rule of the form  $\langle nt_i \rangle \rightarrow T_k \dots$   
 $T_k$  is included in the first set of  $\langle nt_i \rangle$

## Calculating First and Follow Sets – Rule 2



1.  $\langle \text{program} \rangle \rightarrow \text{begin } \langle \text{stmt\_list} \rangle \text{ end}$
  2.  $\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle \langle \text{stmt\_tail} \rangle$
  3.  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$
  4.  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$
  5.  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$
  6.  $\langle \text{var} \rangle \rightarrow \mathbf{A}$
  7.  $\langle \text{var} \rangle \rightarrow \mathbf{B}$
  8.  $\langle \text{var} \rangle \rightarrow \mathbf{C}$
  9.  $\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  10.  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  11.  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  12.  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$
2. For each rule of the form  $\langle \text{nt}_i \rangle \rightarrow \langle \text{nt}_j \rangle \dots$  if  $T_k$  is a member of the first set of  $\langle \text{nt}_j \rangle$  then  $T_k$  is included in the first set of  $\langle \text{nt}_i \rangle$

## Calculating First and Follow Sets – Rule 3



1.  $\langle \text{program} \rangle \rightarrow \text{begin } \langle \text{stmt\_list} \rangle \text{ end}$
  2.  $\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle \langle \text{stmt\_tail} \rangle$
  3.  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$
  4.  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$
  5.  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$
  6.  $\langle \text{var} \rangle \rightarrow \mathbf{A}$
  7.  $\langle \text{var} \rangle \rightarrow \mathbf{B}$
  8.  $\langle \text{var} \rangle \rightarrow \mathbf{C}$
  9.  $\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  10.  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  11.  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  12.  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$
3. For each rule of the form  $\langle \text{nt}_i \rangle \rightarrow \lambda$  if  $T_k$  is a member of the follow set of  $\langle \text{nt}_i \rangle$  then  $T_k$  is included in the first set of  $\langle \text{nt}_i \rangle$

## Calculating First and Follow Sets – Rule 4



1.  $\langle \text{program} \rangle \rightarrow \text{begin } \langle \text{stmt\_list} \rangle \text{ end}$
  2.  $\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle \langle \text{stmt\_tail} \rangle$
  3.  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$
  4.  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$
  5.  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$
  6.  $\langle \text{var} \rangle \rightarrow \mathbf{A}$
  7.  $\langle \text{var} \rangle \rightarrow \mathbf{B}$
  8.  $\langle \text{var} \rangle \rightarrow \mathbf{C}$
  9.  $\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  10.  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  11.  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  12.  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$
4. For each rule of the form  $\langle \rangle \rightarrow \dots \langle \text{nt}_i \rangle T_k \dots$   
 $T_k$  is included in the follow set of  $\langle \text{nt}_i \rangle$

## Calculating First and Follow Sets – Rule 5



1.  $\langle \text{program} \rangle \rightarrow \text{begin } \langle \text{stmt\_list} \rangle \text{ end}$
  2.  $\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle \langle \text{stmt\_tail} \rangle$
  3.  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$
  4.  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$
  5.  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$
  6.  $\langle \text{var} \rangle \rightarrow \mathbf{A}$
  7.  $\langle \text{var} \rangle \rightarrow \mathbf{B}$
  8.  $\langle \text{var} \rangle \rightarrow \mathbf{C}$
  9.  $\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  10.  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  11.  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  12.  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$
5. For each rule of the form  $\langle \rangle \rightarrow \dots \langle \text{nt}_i \rangle \langle \text{nt}_j \rangle \dots$   
if  $T_k$  is a member of the first set of  $\langle \text{nt}_j \rangle$  then  $T_k$  is included in the follow set of  $\langle \text{nt}_i \rangle$

## Calculating First and Follow Sets – Rule 6



1.  $\langle \text{program} \rangle \rightarrow \text{begin } \langle \text{stmt\_list} \rangle \text{ end}$
  2.  $\langle \text{stmt\_list} \rangle \rightarrow \langle \text{stmt} \rangle \langle \text{stmt\_tail} \rangle$
  3.  $\langle \text{stmt\_tail} \rangle \rightarrow ; \langle \text{stmt\_list} \rangle$
  4.  $\langle \text{stmt\_tail} \rangle \rightarrow \lambda$
  5.  $\langle \text{stmt} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$
  6.  $\langle \text{var} \rangle \rightarrow \mathbf{A}$
  7.  $\langle \text{var} \rangle \rightarrow \mathbf{B}$
  8.  $\langle \text{var} \rangle \rightarrow \mathbf{C}$
  9.  $\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  10.  $\langle \text{expr\_tail} \rangle \rightarrow + \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  11.  $\langle \text{expr\_tail} \rangle \rightarrow * \langle \text{var} \rangle \langle \text{expr\_tail} \rangle$
  12.  $\langle \text{expr\_tail} \rangle \rightarrow \lambda$
6. For each rule of the form  $\langle \text{nt}_i \rangle \rightarrow \dots \langle \text{nt}_j \rangle$  if  $T_k$  is a member of the follow set of  $\langle \text{nt}_i \rangle$  then  $T_k$  is included in the follow set of  $\langle \text{nt}_j \rangle$

## Parse Table for Grammar for Small Programming Language



	begin	end	;	=	A	B	C	+	*
program	1								
stmt_list					2	2	2		
stmt					5	5	5		
stmt_tail		4	3						
var					6	7	8		
expression					9	9	9		
expr_tail		12	12					10	11