Clase #23 de 25
Gramáticas, P.A.S., y Yacc

Noviembre 25, Lunes

Gramáticas

La Teoría que originó a BNF

Introducción a Gramáticas

- Modelo matemático para generar LF
  - Un LF puede ser generado por más de una G
  - Una G genera solo un LF
  - Dos G son equivalentes si generan el mismo LF
- Conjuntos de reglas de reescritura
- Derivación
- Vol 1, Página 19, Ejemplo 1
  - L = {ab, ac}
    - S → ab
    - S → ac
- Forma de la regla
  - Lado izquierdo y derecho
    - Par ordenado.

Más Ejemplos

- Vol 1, Página 19, Ejemplo 2
  - L = {a}
  - S → a
- Ejemplo 3
  - L = {aa, ab}
  - Una gramática
    - S → aa
    - S → ab
  - Otra
    - S → aT
    - T → a
    - T → b
- Ejemplo 5
  - L = {aa, ε}
  - S → aa
  - S → ε
- Ejemplo 6
  - L = {aa, aab}
  - S → aaT
  - T → ε
  - T → b
- Página 20, Ejercicio 1b
  - En el ejemplo 6, ¿ T → ε genera la palabra vacía?.

Derivación

- Derivación
- Vol 1, página 27
- Derivación en Forma de Árbol
- Derivación Horizontal
- Derivación Vertical
- Derivación Vertical a Izquierda
- Derivación Vertical a Derecha
- Cadena de Derivación.

Ejemplos y Ejercicios

- ¿Como generar LF infinitos?
  - Recursividad
- Vol1, Página 24, Ejemplo 14
  - $L = \{a^n b / n \geq 1\}$
  - Una GF
    - $S \rightarrow aS$
    - $S \rightarrow aT$
    - $T \rightarrow b$
- Página 24, Ejemplo 20
  - $S \rightarrow aSb$
  - $S \rightarrow a$
- Ejercicio 12
  - a) ¿Cuál es la mínima palabra generada por la G del ejemplo anterior? Justifique
  - b) ¿Cuál es la palabra que le sigue en longitud? Justifique
- Ejercicio 13
  - Describa, por comprensión, el LIC generado por la G del Ejemplo 20
- Ejercicio 14
  - Sea el $L9=\{a^n b^{n+1}/n\geq 0\}$. Escriba las producciones de una GIC que genere L9.

Formalización de Gramáticas

- 4-upla (V_N, V_T, P, S)
  - "Una Gramática Formal genera un Lenguaje Formal" implica que
    - Es capaz de generar todas las palabras del Lenguaje Formal
    - No genera cadenas que están fuera del lenguaje
- Ejemplo 9, página 21
  - G = ({S, T}, {a, b}, {S→aT, T→a, T→b}, S)
  - G = ({S, T}, {a, b}, {(S, aT), (T, a), (T, b)}, S)
  - Con convenciones
    - S → aT
    - T → a | b
  - Metasímbolos
    - →
    - |

Gramática

- Reglas
- Derivación
- Forma de la regla
- Lado izquierdo y derecho
- Par

Notación Matemática

- $A \rightarrow Aa$
- $A \rightarrow a$
- Pares

BNF (ALGOL)

- <exp> ::= <exp-asig> | <exp> , <exp-asig>
- Matemática
  - E → A
  - E → E,A

EBNF (Wirth)

- ISO/IEC 14977
- expresión =
  expresión asignación |
  expresión, ",", expresión asignación
- expresiones asignación =
  expresión asignación , {"," , expresión asignación}

Estilo K&R

- *sentencia*:
  sentencia-expresión
  sentencia-compuesta
- *sentencia-expresión*:
  sentencia$_{opt}$ **;**
- *operador-unario*: uno de
  & * + - ~ !
- *expresión*:
  expresión-asignación
  expresión **,** *expresión-asignación*

Abstracción de las notaciones

- Todas están formadas por 4 partes
  - Noterminales ó Productores o Variables
  - Terminales ó Símbolos
  - Metasímbolos
  - Producciones

Ejemplo de Parser Recursivo Descendente

K&R1988 5.12 Declaraciones Complicadas
MUCH2012 v2s3.2.5 UN PARSER PARA MICRO

Declaraciones – versiones simplificadas

declaration:
    name dcl

dcl:
    optional ***** **'** *s direct-dcl*

direct-dcl:
    name
    (*dcl***)**
    direct-dcl **()**
    direct-dcl **[***optional size***]**

Implementación en C

- Funciones para cada variable
- Analizador Sintáctico Descendente Recursivo (ASDR)

Ejemplos de árboles de derivación

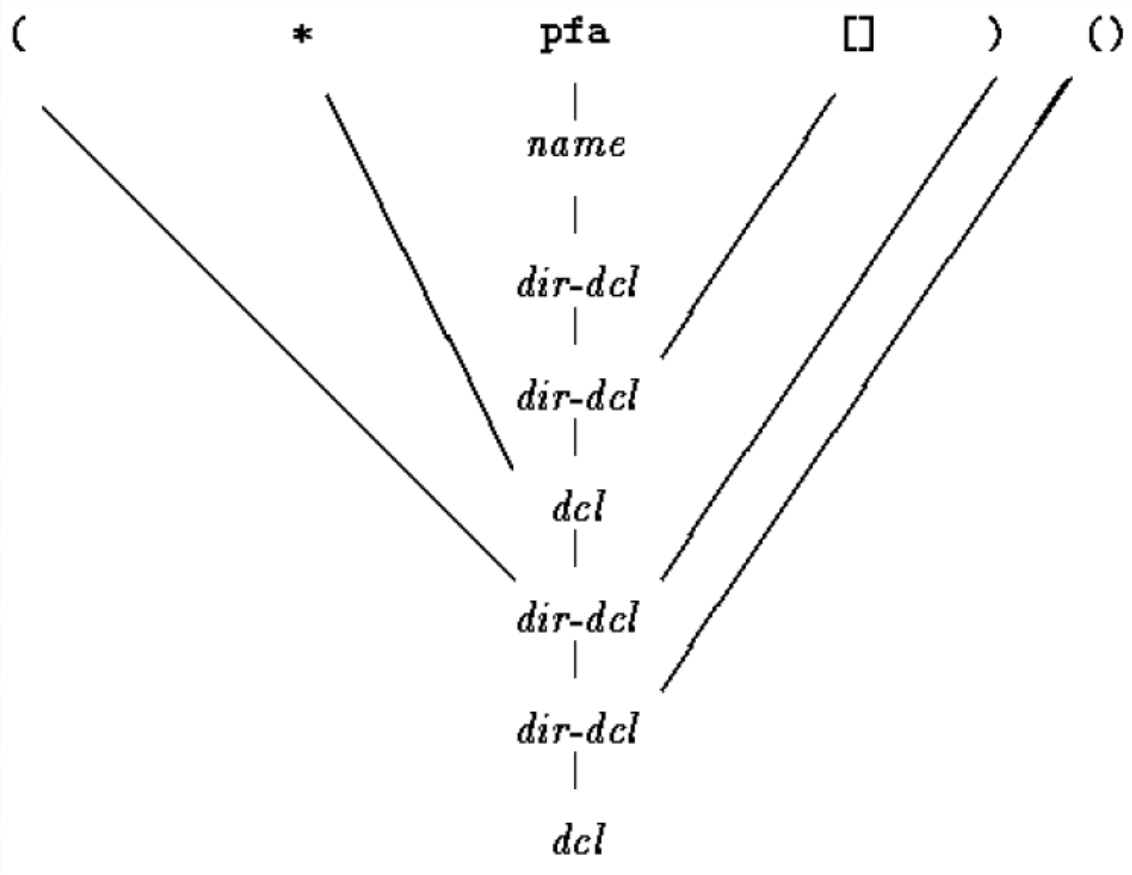
```
int x;

int * y;

int f();

char a[];

float b[7];

long (*p)[9];
```

declaration:
name dcl

dcl:
optional ***'s** *direct-dcl*

direct-dcl:
name
(*dcl***)**
direct-dcl **()**
direct-dcl **[***optional size***]**

Ejemplo: (* pfa []) ()



declaration:
name dcl

dcl:
optional ***** **'s** *direct-dcl*

direct-dcl:
name
(*dcl***)**
direct-dcl **()**
direct-dcl **[***optional size***]**

Dcl

```
void dcl(void){


  for (; gettoken() == '*'; )
    ;

  dirdcl();
}
```

declaration:
    name dcl

dcl:
    optional ***'s** *direct-dcl*

direct-dcl:
    name
    (*dcl***)**
    direct-dcl **()**
    direct-dcl **[***optional size***]**

DirDcl

```
void dirdcl(void){
  int type;

  if (tokentype == '(') {
    dcl();
    if (tokentype != ')')
      printf("error: missing )\n");
  } else if (tokentype == NAME)
    ;
  else
    printf("error: expected name or (dcl)\n");

  while ((type=gettoken()) == PARENS || type == BRACKETS)
    if (type == PARENS)
      ;
    else
      ;
}
```

declaration:
name dcl

dcl:
optional ***** *'s direct-dcl*

direct-dcl:
name
(*dcl***)**
direct-dcl **()**
direct-dcl **[***optional size***]**

Programa completo - Encabezado

```
#include <stdio.h> /* ungetc */
#include <string.h>
#include <ctype.h>
#define MAXTOKEN 100

enum { NAME, PARENS, BRACKETS };

void dcl(void);
void dirdcl(void);
int gettoken(void);

int tokentype; /* type of last token */
char token[MAXTOKEN]; /* last token string */
char name[MAXTOKEN]; /* identifier name */
char datatype[MAXTOKEN]; /* data type */
char out[1000];
```

Programa completo - Main

```
int main(void){ /* Declaration to words */
  while(gettoken()!=EOF){ /*1st token*/
    strcpy(datatype, token); /*datatype*/
    out[0] = '\0';
    dcl(); /* parse rest of line */
    if (tokentype != '\n')
      printf("syntax error\n");
    printf("%s: %s %s\n",
      name, out, datatype);
  }
  return 0;
}
```

declaration:
name dcl

dcl:
optional ***** *'s direct-dcl*

direct-dcl:
name
(*dcl***)**
direct-dcl **()**
direct-dcl **[***optional size***]**

Programa completo – GetToken

```
int gettoken(void) {
   int c;
   char *p = token;

   while ( (c = getchar()) == ' ' || c == '\t' )
      ;

   if (c == '(') {
      if ((c = getchar()) == ')') {
         strcpy(token, "()");
         return tokentype = PARENS;
      } else {
         ungetc(c);
         return tokentype = '(';
      }
   } else if (c == '[') {
      for (*p++ = c; (*p++ = getchar()) != ']'; )
         ;
      *p = '\0';
      return tokentype = BRACKETS;
   } else if (isalpha(c)) {
      for (*p++ = c; isalnum(c = getchar()); )
         *p++ = c;
      *p = '\0';
      ungetch(c);
      return tokentype = NAME;
   } else
      return tokentype = c;
}
```

Dcl con acciones

```
void dcl(void){
  int ns;

  for (ns = 0; gettoken() == '*'; )
    ns++; /* count *'s */
  dirdcl();
  while (ns-- > 0)
    strcat(out, " pointer to");
}
```

declaration:
    name dcl

dcl:
    optional ***'s** *direct-dcl*

direct-dcl:
    name
    (*dcl***)**
    direct-dcl **()**
    direct-dcl **[***optional size***]**

DirDcl con acciones

direct-dcl:

name

(*dcl***)**

*direct-dcl***()**

*direct-dcl***[***optional size***]**

```
void dirdcl(void){
 int type;

 if (tokentype == '(') { /* ( dcl ) */
   dcl();
   if (tokentype != ')')
     printf("error: missing )\n");
 } else if (tokentype == NAME) /* variable name */
   strcpy(name, token);
 else
   printf("error: expected name or (dcl)\n");
 while ((type=gettoken()) == PARENS || type == BRACKETS)
   if (type == PARENS)
     strcat(out, " function returning");
   else {
     strcat(out, " array");
     strcat(out, token);
     strcat(out, " of");
   }
}
```

Undcl

```
x () * [] * () char

char (*(*x())[])()
```

```
int main(void){
  int type;
  char temp[MAXTOKEN];
  while (gettoken() != EOF) {
    strcpy(out, token);
    while ((type = gettoken()) != '\n')
      if(type==PARENS || type==BRACKETS)
        strcat(out, token);
      else if (type == '*') {
        sprintf(temp, "(*%s)", out);
        strcpy(out, temp);
      } else if (type == NAME) {
        sprintf(temp, "%s %s",token,out);
        strcpy(out, temp);
      } else
        printf("invalid %s\n", token);
    puts(out);
  }
  return 0;
}
```

Ejercicios

- 5-18. Que dcl se recupere de errores
- 5-19. Modifique undcl para que no agregue paréntesis redundantes a las declaraciones
- 5-20. Expanda dcl para manejar declaraciones con tipos de argumentos, calificadores como const, etc.

Introducción a Yacc

MUCH2012 vol 2 Sección 4.3.7.4 Ejemplo 21

```
%{
/* S -> aTc
   T -> aTc | b  */
#include <stdio.h>
int yylex(void);
void yyerror(const char *);
%}
%%
S
    : 'a' T 'c'
    ;
T
    : 'a' T 'c'
    | 'b'
    ;
%%
int main(void){
   switch( yyparse() ){
      case 0: puts("Pertenece al LIC"); return 0;
      case 1: puts("No pertenece al LIC"); return 1;
      default: puts("Memoria insuficiente"); return 2;
   }
}

// Scanner: retorna el siguiente token, si no hay más, retorna 0.
int yylex(void){
   int c = getchar();
   if(c == EOF) return 0;
   return c;
 }

// Informa la ocurrencia de un error.
void yyerror(const char *s){puts(s);}
```

¿Consultas?

Fin de la clase