Ivanna Tinedo
Sebastián Ruiz
Alan Franco

Yacc y Bison como herramientas

Introducción

Yacc es una herramienta ampliamente utilizada para generar analizadores sintácticos en la construcción de compiladores y procesadores de lenguaje. Se utiliza en conjunto con Lex o Flex para construir un analizador léxico y un analizador sintáctico para un lenguaje de programación específico.

Funciones Principales de Yacc

1. Reglas Gramaticales

Una de las funciones más importantes es definir las reglas gramaticales del lenguaje que se está analizando. Esto se hace por medio de reglas de producción que especifican la estructura del lenguaje.

Ejemplo:

```
stmt: IF expr THEN stmt
   | IF expr THEN stmt ELSE stmt
   | WHILE expr DO stmt
   | ...
   ;
```

2. Acciones Semánticas

Yacc permite asociar acciones semánticas con cada regla gramatical y estas se ejecutan cuando se reconoce una regla y se utilizan para construir un árbol de sintaxis abstracta o realizar acciones específicas en respuesta a la entrada.

Ejemplo:

```
stmt: IF expr THEN stmt
   {
      if ($2) {
         // Código para manejar la sentencia IF verdadera
      } else {
         // Código para manejar la sentencia IF falsa
      }
   }
   ;
```

Ivanna Tinedo
Sebastián Ruiz
Alan Franco

3. Integración con Analizador Léxico

Yacc se utiliza en conjunto con un analizador léxico como Lex o Flex para analizar el flujo de tokens en el código fuente.

Ejemplo con lex:

```
%token ID INT
%%
stmt: ID '=' INT
   {
      // Acción semántica para asignar un valor a una variable
   }
   ;
```

4. Generación de Código

Una vez que se han definido las reglas gramaticales y las acciones semánticas, Yacc puede generar código C o C++ para construir el analizador sintáctico.

Uso de Yacc

Este proceso generalmente implica los siguientes pasos:

1. Definir las reglas gramaticales y las acciones semánticas en un archivo “.y” que contiene la especificación de la gramática.

2. Ejecutar Yacc en el archivo “.y” para generar el código C o C++ del analizador sintáctico.

3. Compilar el código generado junto con el código del analizador léxico y otros componentes del compilador.

4. Ejecutar el programa compilado para analizar y procesar el código fuente en el lenguaje de origen.

Ejemplo de Uso de Yacc

Archivo Yacc que define una gramática para expresiones matemáticas simples y utiliza acciones semánticas para evaluar el resultado de las expresiones:

```
%{
```

Ivanna Tinedo
Sebastián Ruiz
Alan Franco

```
#include <stdio.h>
%}

%token NUM
%left '+' '-'
%left '*' '/'

%%

expr: NUM
   {
      $$ = $1;
   }
   | expr '+' expr
   {
      $$ = $1 + $3;
   }
   | expr '-' expr
   {
      $$ = $1 - $3;
   }
   | expr '*' expr
   {
      $$ = $1 * $3;
   }
   | expr '/' expr
   {
      if ($3 != 0) {
         $$ = $1 / $3;
      } else {
         fprintf(stderr, "Error: División por cero\n");
         exit(1);
      }
   }
   ;

%%

int yylex() {
   int c = getchar();
   if (c == EOF) return 0;
   if (c >= '0' && c <= '9') {
```

Ivanna Tinedo
Sebastián Ruiz
Alan Franco

```
 yylval = c - '0';
        return NUM;
    }
    if (c == '+' || c == '-' || c == '*' || c == '/') {
        return c;
    }
    if (c == ' ' || c == '\t' || c == '\n') {
        return yylex();
    }
    fprintf(stderr, "Error: Carácter desconocido '%c'\n", c);
    exit(1);
}

int main() {
    yyparse();
    return 0;
}
```

Biblioteca Yacc

Contiene implementaciones predeterminadas de las funciones yyerror y main. Estas implementaciones predeterminadas normalmente no son útiles, pero POSIX las requiere. Para utilizarla conecte su programa con la opción “-ly”.

Si utiliza la función yyerror de la biblioteca Yacc, debe declarar yyerror de la siguiente manera:

```
int yyerror (char const *);
```

Bison ignora el valor entero devuelto por este yyerror. Si utiliza la función principal de la biblioteca Yacc, su función yyparse debe tener la siguiente firma de tipo:

```
int yyparse (void);
```

Funciones y Parámetros en el Proceso de Análisis Sintáctico

El análisis sintáctico es una etapa de compilación de programas informáticos, donde se valida que una secuencia de tokens cumpla con la gramática del lenguaje de programación y es ahí donde entran Bison y Yacc que son herramientas utilizadas para generar analizadores sintácticos en C/C++. Algunas funciones son:

Función yyparse

Ivanna Tinedo
Sebastián Ruiz
Alan Franco

Su función es leer tokens, ejecutar acciones definidas en la gramática y retornar en distintas situaciones, con valores de retorno específicos, como se mencionó previamente.

Macros YYACCEPT y YYABORT
Estos ayudan a controlar el flujo de ejecución de yyparse de manera inmediata. YYACCEPT retorna con valor 0 para indicar que funciono, mientras que YYABORT retorna con valor 1 para indicar fallo.

Uso de Parámetros Adicionales
Tanto en Bison como en Yacc, es posible pasar información adicional a la función yyparse de manera reentrante utilizando la declaración %parse-param lo que permite mejorar el proceso de análisis sintáctico con datos específicos según las necesidades del compilador o analizador sintáctico.

Conclusión

En esta investigación, aprendimos cómo construir analizadores para compiladores y procesadores de lenguaje usando Yacc y Bison. Las reglas gramaticales, las acciones semánticas y la creación de código de análisis sintáctico dependen de estas herramientas.

Descubrimos que los analizadores léxicos y sintácticos se construyen utilizando Yacc junto con Lex o Flex. Una de sus tareas principales es especificar reglas gramaticales y vincular acciones semánticas con ellas. También analizamos el uso de macros como YYABORT y YYAACCEPT para controlar cómo se ejecuta el código.

Además, aprendimos cómo enriquecer el análisis pasando repetidamente parámetros adicionales. La biblioteca Yacc ofrece implementaciones predeterminadas de funciones compatibles con POSIX como yyerror y main.