Autómatas, Teoría de Lenguajes y Compiladores
TPE

Lenguaje “Bimbo”

Instituto Tecnológico de Buenos Aires
Guido Gordyn Biello
59099

Objetivo

El lenguaje Bimbo es un lenguaje de programación creado con el objetivo de enseñar las nociones básicas de programación a aquellos que hayan tenido poca o nula experiencia. Para eso, plantea una sintaxis simple, evitando símbolos que puedan generar confusiones o nombres de funciones cuya utilidad no sea clara.

Consideraciones realizadas

Como se volverá a mencionar posteriormente, el lenguaje Bimbo genera un output en C. Para evitar errores que pudiese retornar gcc, se intentó “atrapar” la mayor cantidad posibles equivocaciones al escribir en el lenguaje, haciendo uso de una tabla de símbolos para revisar la existencia de variables y sus tipos.

Desarrollo del lenguaje

Se empleó LEX (analizador léxico) para reconocer los lexemas del lenguaje, definidos por la gramática. A continuación utilizando YACC se generaron las producciones de la gramática que darían lugar al parser.

Al finalizar la gramática se crearon cinco programas de ejemplo para exhibir las características del lenguaje. Estos programas son:

- **HelloWorld**: el típico Hello World que imprime la frase clásica, pero además lee de entrada estándar un string y un número y los imprime, para demostrar la capacidad de lectura del lenguaje (sentencia READ INTO).
- **multiples:** un programa que recibe un número y calcula e imprime sus 10 primeros múltiplos, utilizando una sentencia de control do-while (REPEAT WHILE en el caso de Bimbo).
- **gcd:** programa que recibe dos números y calcula su máximo común divisor, utilizando sentencias do-while e if.
- **textList:** programa que escribe junto al usuario una lista de shopping, empleando la capacidad del lenguaje de crear listas de strings e imprimirlas.

- **numberList:** programa que escribe junto al usuario una lista de números, empleando la capacidad del lenguaje de crear listas de enteros e imprimirlas.

Para darle más poder al lenguaje, se decidió crear una tabla de símbolos empleando una lista encadenada. No se utilizó un árbol u otra estructura ya que no se pretende que el usuario que está aprendiendo a programar cree una cantidad inmensa de variables. Además, se reutiliza la definición de esta lista como tipo de dato del lenguaje Bimbo.

Con esta tabla de símbolos que alberga el nombre y tipo de las variables, se puede controlar la compilación y abortar cuando se esté cometiendo un error. Si bien la gramática misma previene errores de sintaxis antes de llegar al compilador de C, se quería lograr un mayor control de errores. Por eso, utilizando la tabla de símbolos se controla la sintaxis para que no se redefinan variables, ni se utilicen variables que nunca fueron definidas. Además, en el caso de las listas, se asegura que la variable que se esté utilizando sea efectivamente una lista ya creada previamente.

Si se encuentra uno de estos errores, el compilador aborta la compilación y elimina el output de c que está creando. Además, imprime un mensaje de error que explica claramente cuál es el problema, y luego llama a yyerror() donde se aclara en qué línea ocurre el error.

Esta característica se añadió considerando que los usuarios target del lenguaje son personas con muy poca experiencia en programación, por lo cual mensajes de error crípticos, más aún si vienen de gcc, pueden desconcertarlos.

Otra consideración que se tomó al desarrollar el lenguaje fue intentar simplificar las sentencias para facilitar la comprensión a los potenciales programadores. Esto permite aprender más fácilmente, además de alejarse de la sintaxis a veces complicada de C. Un ejemplo claro de esto es la manera en la que se permite imprimir variables en Bimbo. Se presenta el siguiente extracto de código:

```
CREATE NUMBER num IS 3,
PRINT NUMBER: num,
CREATE TEXT salute IS “hello world”,
PRINT TEXT: salute,
```

En comparación con el equivalente en C:

```
int num = 3;
printf(“%d”, num);
char * salute = “hello world”;
printf(“%s”, salute);
```

En el ejemplo de C, las llamadas a printf pueden ser muy confusas para alguien sin ningún tipo de experiencia. No le sería fácil comprender que debe indicar con %d que lo que busca imprimir es un entero, ni por qué esto funciona así. En Bimbo se buscó simplificar esto aclarando el tipo de dato que se quiere imprimir en la sentencia PRINT, y luego de los dos puntos el nombre de la variable: una sintaxis mucho más agradable.

Además, como adición extra al lenguaje, se incorporaron listas de enteros y de texto. Su implementación es clásica y simple, y permite agregar y eliminar elementos e imprimir la lista. Con esta característica se busca que los programadores se introduzcan a la posibilidad de tener arreglos ordenados de sus tipos de datos y poder visualizar esta información.

Gramática empleada

La gramática empleada se basó en la que había que definir en el ejercicio 6 del Trabajo Práctico 7 de la materia, pero con adiciones de nuevas declaraciones e instrucciones posibles. Sus características son las siguientes:

1. Exige que se inicie el programa con la sentencia BIMBO: y se termine con un punto final (.).
2. Dentro de los delimitadores del programa, se puede realizar cualquier combinación de instrucciones y sentencias de control, permitiendo la anidación.
3. Las instrucciones pueden ser declaraciones, asignaciones , impresiones y lectura desde línea de comandos; todas deben finalizar con ‘,’ . Además, las operaciones sobre listas. Entonces:

    a. Declaración: CREATE [DATATYPE] [var_name], (siendo DATATYPE: NUMBER o TEXT. Se utilizó la palabra text para facilitar la comprensión). Además, se puede crear una lista con CREATE [LIST_TYPE] [list_name] STARTING ON [start_value],
    b. Asignación: [var_name] IS [constant / expression], (obviamente, se puede combinar con la declaración).
    c. Impresión: PRINT [DATATYPE]: [variable / constante],
    d. Lectura: READ [DATATYPE] INTO: [variable],
    e. Operaciones de listas: ADD [variable / constante] TO [LIST_TYPE] [list_name], REMOVE [variable / constante] FROM [LIST_TYPE] [list_name],
4. Las sentencias de control pueden ser:
    a. La sentencia IF, con ELSE opcional: IF [expression] THEN DO: [instructions…] ELSE [instructions…] END IF
    b. La sentencia REPEAT-WHILE: REPEAT: [instructions] WHILE [expression],
5. Las expresiones pueden contener operadores aritméticos, lógicos o de comparación. Se utilizaron palabras en vez de símbolos para facilitar la comprensión a los principiantes. Estos operadores son:
    a. [exp] GREATER THAN [exp] (representa el “>”).
    b. [exp] LESSER THAN [exp] (representa el “<”).
    c. [exp] EQUAL TO [exp] (representa el “==”).
    d. [exp] NOT EQUAL TO [exp] (representa el “!=”);
    e. [exp] PLUS [exp] (representa el “+”);
    f. [exp] MINUS [exp] (representa el “-”);
    g. [exp] DIVIDED BY [exp] (representa el “/);
    h. [exp] MODULO [exp] (representa el “%”);
    i. [exp] TIMES [exp] (representa el “*”);
    j. [exp] AND [exp] (representa el “&&”);
    k. [exp] OR [exp] (representa el “||”);

Dificultades encontradas

Se presentaron varios problemas al desarrollar el compilador de Bimbo. Estos fueron:

1. La traducción a C. Hubo que crear un archivo bash que recibiera como input el archivo .bimbo, lo parseara, tradujera a un output en c, y a ese output lo compilara en gcc para crear el archivo ejecutable.
2. El problema de la impresión. Como se mencionó anteriormente, se quería una manera simple de imprimir, por lo cual hubo que usar los atributos de la gramática estratégicamente para poder traducir la impresión a un printf de c.
3. El problema de la lectura desde entrada estándar. Surgió de la misma manera que la impresión y se resolvió análogamente.
4. La tabla de símbolos. Sin duda el problema más grande, ya que en el scope del trabajo se planeó incluir esta tabla para poder evitar errores, además de usarla en el futuro para otras características. Primero se empleó un array, pero este tenía tamaño limitado por lo cual solamente permitía una cantidad de variables limitadas; esto no tenía sentido en un lenguaje de programación.
5. Los conflictos. Se encontraron algunos conflictos en la gramática que generaron que ciertas producciones fueran inalcanzables, por lo que hubo que resolverlos.
6. Los programas de ejemplo. Se necesitó buscar una manera de poder demostrar todas las utilidades del lenguaje en los programas.

Futuras extensiones

Entre las posibles características para extender el lenguaje, se podría considerar:

1. Emplear la tabla de símbolos para “atrapar” otro tipo de errores, por ejemplo, asignaciones inválidas que son más difíciles de contrarrestar y que llevan a errores en tiempo de ejecución.

2. Adiciones a la biblioteca de listas, si bien complicaría su uso para principiantes podría dar más utilidad, como un iterador.
3. Extensión al tipo de dato NUMBER para incluir punto flotante. Es una idea que le daría más libertad al usuario permitiendo que utilice números con coma, pero tendría que validar cuál de los dos tipos estoy utilizando cada vez que uso un NUMBER. La idea sería que si se reconociera un "." en el número, automáticamente se asumiera que es un número con decimales.
4. Adiciones generales a la biblioteca de funciones. Podría agregarse, además de una lista, un stack o un árbol, con el de que los programadores que estén aprendiendo aprendan sobre estas estructuras y su uso.

Referencias

Para la creación de Bimbo se tomó como base la idea de un lenguaje de enseñanza como Gobstones pero con mayor libertad. Además, se incluyó una estructura de datos de lista que manejan lenguajes como Java, que C no posee. Por último se tomó como gran referencia el lenguaje Python por su sintaxis más agradable.

Bibliografía y referencias:

- Clases de yacc y lex dictadas en campus.
- Lex & Yacc - Levine.
- Serie "Let's make a Programming Language" de Code Emporium.
- StackOverflow.