Notación

- En este documento, la representación de las producciones de la gramática del lenguaje de programación ALFA se ajusta a la notación propia de Bison con las siguientes particularidades:
  - Los símbolos no terminales se representan en minúsculas
  - Los símbolos terminales se representan con TOK_ …
  - Los símbolos terminales de un carácter se representan encerrados entre comillas simples

Objetivo del análisis semántico

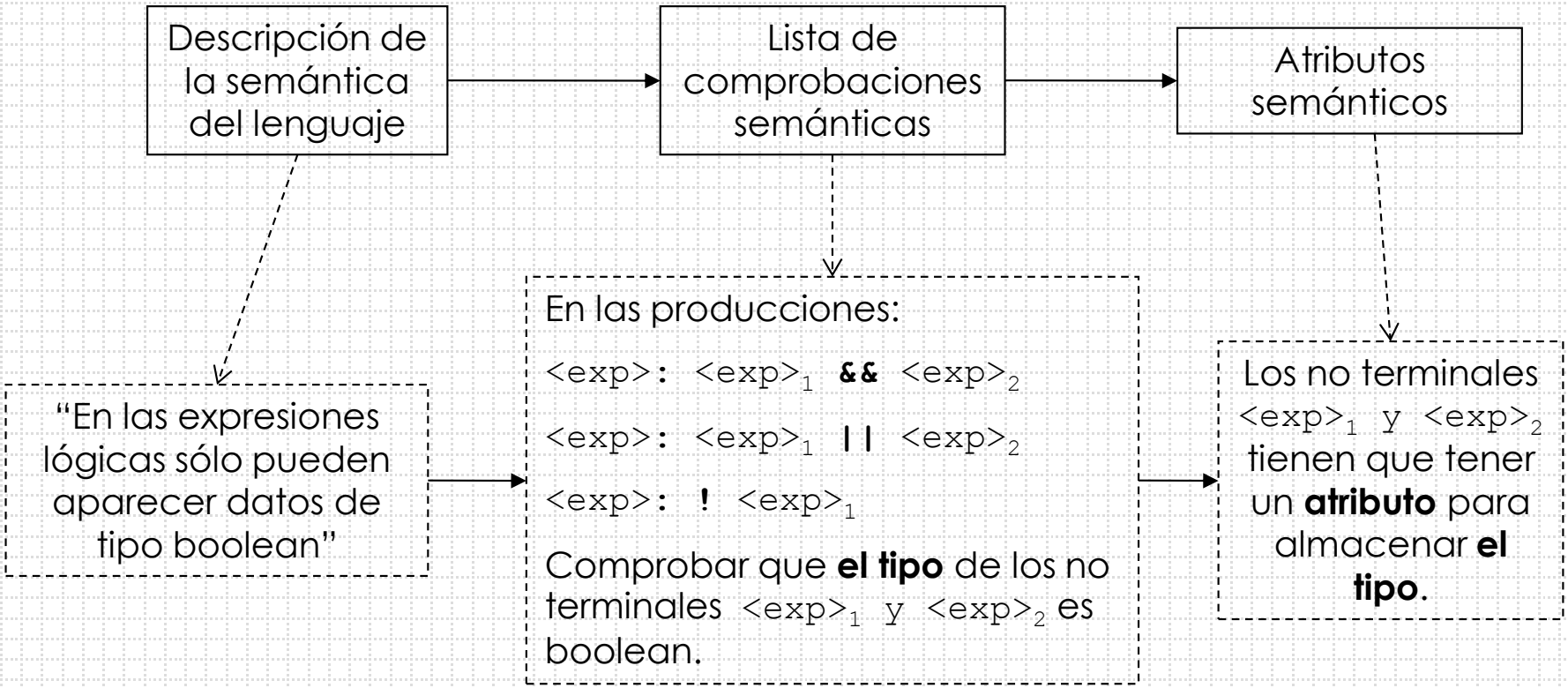
- La semántica del lenguaje forma parte de la especificación del mismo. Normalmente **la semántica de un lenguaje se describe de manera informal**.

- El objetivo del análisis semántico es **comprobar si la semántica del programa que se está compilando cumple las especificaciones de la semántica del lenguaje fuente**. Algunas de estas comprobaciones son:
  - comprobación de tipos en sentencias de asignación
  - comprobación de tipos en operaciones aritmético-lógicas
  - comprobación de tipos en las sentencias condicionales
  - comprobación de la declaración de las variables antes de su uso
  - comprobación del número de parámetros en la llamada a una función

- Es imprescindible conocer las restricciones semánticas del lenguaje antes de iniciar su desarrollo (leer el enunciado con detenimiento)

Pasos para la construcción de un analizador semántico

- Para realizar el análisis semántico se utilizan **gramáticas de atributos.** (Gramática independiente del contexto + sistema de atributos)
- Hay que **definir los atributos** (los necesarios para realizar las comprobaciones semánticas y también la generación de código)



Pasos para la construcción de un analizador semántico

- Además de definir los atributos hay que definir el modo en que se **calculan los valores de los atributos**.

- Por ejemplo, el atributo tipo del no terminal <exp> se calcula/evalúa en muchas producciones. Algunas de ellas son:

  $$ $1 $2 $3

  - En la producción `<exp>` : `<exp>`$_1$ **&&** `<exp>`$_2$ al atributo **tipo** del no terminal <exp> se le asigna el valor boolean.
  - En la producción `<exp>` : `<exp>`$_1$ **||** `<exp>`$_2$ al atributo **tipo** del no terminal <exp> se le asigna el valor boolean.
  - En la producción `<exp>` : **!** `<exp>`$_1$ al atributo **tipo** del no terminal <exp> se le asigna el valor boolean.

Análisis semántico con Bison

- Los pasos para la construcción del analizador semántico con Bison son:
  - De la especificación del lenguaje se deduce la lista de comprobaciones semánticas.
  - Los **atributos necesarios** se deducen a partir de las comprobaciones semánticas. El tipo de los atributos se define mediante la directiva **%union** dentro del fichero de especificación de Bison.
  - Los atributos son **sintetizados**.
    Esta restricción viene impuesta porque el tipo de analizador sintáctico que genera Bison es ascendente LALR(1).
    Si la gramática de atributos que se diseña para implementar el análisis semántico requiere atributos heredados, hay que definir un mecanismo para incorporar el cálculo de dichos atributos en el analizador ascendente generado por Bison.
  - El **cálculo de los atributos** se realiza en las acciones asociadas a las reglas.
  - El **acceso a los atributos** se realiza a través de las pseudovariables $$, $1, $2, ...
- En resumen, el analizador semántico se **incorpora al analizador sintáctico** construido con Bison **insertando en las acciones de las reglas las comprobaciones propias del análisis semántico** (posteriormente también se incorporará la generación de código).

Definición de los atributos semánticos

- Para la implementación del compilador de ALFA se *propone*[1] el siguiente conjunto de atributos:
  - **char* nombre** guarda el nombre de los identificadores.
  - **int tipo:** guarda el tipo de una expresión (TIPO_ENTERO, TIPO_LOGICO) Definir "alfa.h".
  - **int valor_entero**: guarda el valor de una constante entera.
  - **int es_direccion**: atributo que indica si un símbolo representa una dirección de memoria o es un valor constante.
  - **int etiqueta**: atributo necesario para gestión de sentencias condicionales e iterativas. Es un atributo definido exclusivamente para la generación de código.

- IMPORTANTE: no confundir la información guardada en la tabla de símbolos con la información que se guarda en el sistema de atributos.

[1] La elección de los atributos no es única, existen distintas alternativas.

Definición de los atributos semánticos

- Los **atributos semánticos** de los **terminales** lo proporciona (calcula) el analizador léxico de la siguiente manera:
  - los **identificadores** tienen como valor semántico su **nombre**
  - las **constantes enteras** tienen como valor semántico su **valor numérico**.

- Los **atributos semánticos** de los **no terminales** se calculan en el proceso de análisis sintáctico/semántico. Cada vez que se reduce una producción, se pueden calcular los atributos semánticos del símbolo no terminal de la parte izquierda de la producción.

Definición de los atributos semánticos

- En Bison, los atributos semánticos se definen con la declaración **%union** en el fichero de especificación.

- La declaración %union de Bison se transforma en una definición de tipo union de C. Este tipo de datos de C sólo permite el uso de uno de sus campos, y no es válido para símbolos con múltiples atributos. Para permitir la **multiplicidad de atributos**, se puede definir en la declaración %union un sólo campo (atributos) cuyo tipo (tipo_atributos) sea un tipo definido como struct con tantos campos como sea necesario para contemplar todos los casos posibles de valores semánticos de cualquier símbolo del lenguaje (terminal o no terminal).

  NOTA: El tipo "tipo_atributos" se puede definir en un fichero de cabecera aparte.

Definición de los atributos semánticos

- En la declaración **%union** del fichero **alfa.y**, se define un sólo campo, **atributos**, cuyo tipo sea **tipo_atributos** (definido en el fichero alfa.h).
- Se cualifican con **<atributos>** las declaraciones **%token** de los símbolos terminales que tienen valor semántico.
- Se añaden las correspondientes declaraciones **%type** para los **no terminales** que tengan **atributos semánticos.**

alfa.y

```
%{
#include "alfa.h"
...
%}
%union
{
  tipo_atributos atributos;
}

%token <atributos> TOK_IDENTIFICADOR
%token <atributos> TOK_CONSTANTE_ENTERA


/* resto de los tokens sin valor */
/* semántico                     */

%type <atributos> exp
%type <atributos> comparacion

/* resto de los no terminales */
/* con atributos semánticos   */

...

%%
...
%%
...
```

Definición de los atributos semánticos

- En el fichero **alfa.h** se define el tipo **tipo_atributos** como un struct con los siguientes campos :
  - **nombre** : para identificadores.
  - **tipo**: para comprobación de tipos básicos.
  - **valor_entero**: para constantes enteras.
  - **es_direccion**: para controlar lo que representa cada símbolo (una dirección de memoria o un valor constante).
  - **etiqueta**: para la generación de código de sentencias condicionales e iterativas.

alfa.h

```
#ifndef _ALFA_H
#define _ALFA_H

#define MAX_LONG_ID 100
#define MAX_TAMANIO_VECTOR 64
...

/* otros defines */


typedef struct
{
    char nombre[MAX_LONG_+1];
    int tipo;
    int valor_entero;
    int es_direccion;
    int etiqueta;
}tipo_atributos;

#endif
```

- El fichero **alfa.l** se modifica para que el analizador léxico actualice los atributos semánticos de los terminales.

Declaración y uso de identificadores

Inserción en la tabla de símbolos

- Independientemente del punto seleccionado para la inserción de un identificador en la tabla de símbolos, es necesario **disponer** en dicho punto de **toda la información asociada al identificador**, como por ejemplo:
  - **nombre**
  - **categoría** (variable, parámetro o función)
  - **clase** (escalar o vector)
  - **tipo** (entero o booleano)
  - **tamaño** (número de elementos de un vector)
  - **número de variables locales**
  - **posición de variable local**
  - **número de parámetros**
  - **posición de parámetro**

Comprobaciones semánticas en las expresiones aritméticas

Implementación

- En las producciones correspondientes a las demás operaciones aritméticas se realizan las mismas comprobaciones y cálculo de atributos:
  - (73) exp: exp '-' exp
  - (74) exp: exp '/' exp
  - (75) exp: exp '*' exp
  - (76) exp: '-' exp

Comprobaciones semánticas en las expresiones lógicas

Comprobación de tipos

- **Descripción de la semántica:** En las expresiones lógicas sólo pueden aparecer datos de tipo boolean. El tipo de una expresión lógica boolean.

- **Comprobaciones semánticas:** hay que comprobar que los operandos son de tipo boolean en las siguientes producciones y asignar el tipo correspondiente a la expresión resultante:

  - (77) exp : exp TOK_AND exp
  - (78) exp : exp TOK_OR exp
  - (79) exp : '!' exp

Comprobaciones semánticas en las expresiones lógicas

Implementación

- En las producciones correspondientes a expresiones lógicas, las tareas que realiza el analizador semántico son:
  - Realizar las comprobaciones adecuadas en cada producción. Si alguna comprobación no es correcta se genera un error semántico y se termina el proceso de compilación con error.
  - Calcular (propagar) los atributos correspondientes.

- En la producciones correspondientes a las operaciones de conjunción y disyunción de expresiones:

  (77) exp : exp TOK_AND exp
  (78) exp : exp TOK_OR exp

  Se comprueba que:
  - Las dos expresiones son de tipo lógico, accediendo a través de $1.tipo y $3.tipo.

  Se calculan los atributos:
  - El tipo del símbolo de la parte izquierda es lógico ($$.tipo = BOOLEAN).
  - El símbolo de la parte izquierda no es una dirección de memoria, es un valor ($$.es_direccion =0).

Comprobaciones semánticas en las expresiones lógicas

Implementación

- En la producción correspondiente a la negación lógica:

  (79) exp : '!' exp

  Sólo es necesario comprobar que el tipo de la expresión de la parte derecha es boolean.

  Se calculan los atributos:

  - El tipo del símbolo de la parte izquierda es lógico ($$.tipo = BOOLEAN).
  - El símbolo de la parte izquierda no es una dirección de memoria, es un valor ($$.es_direccion =0).

Comprobaciones semánticas en las expresiones de comparación

Comprobación de tipos

- **Descripción de la semántica:** Las comparaciones sólo pueden operar con datos de tipo numérico, siendo el resultado de la comparación de tipo boolean.

- **Comprobaciones semánticas:** hay que comprobar que los elementos comparados son de tipo numérico en las siguientes producciones y asignar el tipo correspondiente (boolean) a la expresión resultante:

  - (93) comparacion : exp TOK_IGUAL exp
  - (94) comparacion : exp TOK_DISTINTO exp
  - (95) comparacion : exp TOK_MENORIGUAL exp
  - (96) comparacion : exp TOK_MAYORIGUAL exp
  - (97) comparacion : exp '<' exp
  - (98) comparacion : exp '>' exp

Comprobaciones semánticas en las expresiones de comparación

Implementación

- En todas las producciones correspondientes a las comparaciones de expresiones, las únicas tareas que realiza el analizador semántico son:

  Comprobar que,

  - Las dos expresiones son de tipo numérico, accediendo a través de $1.tipo y $3.tipo.

  Calcular los atributos:

  - El tipo del símbolo de la parte izquierda es lógico ($$.tipo = BOOLEAN).
  - El símbolo de la parte izquierda no es una dirección de memoria, es un valor ($$.es_direccion =0).

- Las producciones en las que se implementan las comprobaciones semánticas son:
  - (93) comparacion : exp TOK_IGUAL exp
  - (94) comparacion : exp TOK_DISTINTO exp
  - (95) comparacion : exp TOK_MENORIGUAL exp
  - (96) comparacion : exp TOK_MAYORIGUAL exp
  - (97) comparacion : exp ‘<‘ exp
  - (98) comparacion : exp ‘>’ exp

Comprobaciones semánticas para las constantes

Introducción

- Todas las producciones relacionadas con las constantes sean del tipo que sean únicamente incorporan cálculo (propagación) de atributos. La semántica de ALFA no exige la realización de comprobaciones semánticas para las constantes.

- Las producciones relacionadas con las constantes son las siguientes:
  - (81) exp: constante
  - (99) constante : constante_logica
  - (100) constante : constante_entera
  - (102) constante_logica : TOK_TRUE
  - (103) constante_logica : TOK_FALSE
  - (104) constante_entera : TOK_CONSTANTE_ENTERA

Comprobaciones semánticas para las constantes

Implementación

- En la producción que se reduce cuando el analizador morfológico identifica en la entrada una constante entera, se realiza la siguiente propagación de atributos:

```
(104) constante_entera:  TOK_CONSTANTE_ENTERA
                         {
                                 $$.tipo = INT;
                                 $$.es_direccion = 0;
                         }
                         ;
```

Comprobaciones semánticas para las constantes

Implementación

- Las producciones que se reducen cuando el analizador morfológico identifica en la entrada una constante booleana, contienen la siguiente propagación de atributos:

```
(102) constante_logica:  TOK_TRUE
                         {
                                 $$.tipo = BOOLEAN;
                                 $$.es_direccion = 0;
                         }
                         ;
(103) constante_logica:  TOK_FALSE
                         {
                                 $$.tipo = BOOLEAN;
                                 $$.es_direccion = 0;
                         }
                         ;
```

Comprobaciones semánticas para las constantes

Implementación

- Por último, las producciones intermedias que representan que una constante es entera o lógica, tienen el siguiente cálculo de atributos:

```
(99)   constante : constante_logica
                   {
                            $$.tipo = $1.tipo;
                            $$.es_direccion = $1.es_direccion;
                   }
(100)  constante : constante_entera
                   {
                            $$.tipo = $1.tipo;
                            $$.es_direccion = $1.es_direccion;
                   }
                   ;
```

Comprobaciones semánticas para otras expresiones

Introducción

- El no terminal **exp** tiene varias producciones además de las relativas a expresiones aritméticas y lógicas cuyas comprobaciones semánticas ya han sido descritas. Estas producciones son:
  - (80) exp: TOK_IDENTIFICADOR
  - (81) exp: constante
  - (82) exp: '(' exp ')'
  - (83) exp: '(' comparacion ')'
  - (85) exp: elemento_vector
  - (88) exp: TOK_IDENTIFICADOR '(' lista_expresiones ')'
- Las comprobaciones semánticas de cada una de las producciones se describe a continuación.

Comprobaciones semánticas para otras expresiones

Implementación

- En la producción siguiente:

  (80) exp : TOK_IDENTIFICADOR

  Se comprueba que:

  - El identificador (`$1.nombre`) existe en la tabla de símbolos (en cualquiera de los ámbitos abiertos). Si el identificador no existe, se genera un error semántico y se termina el proceso de compilación con error.
  - El identificador no es ni de categoría función (es decir, es una variable o un parámetro), ni de clase vector (es decir, es un escalar).

  Se calculan los atributos:

  - El tipo del símbolo exp de la parte izquierda ($$.tipo) se establece haciendo uso de la información almacenada en la tabla de símbolos para el identificador de la parte derecha de la producción. (IMPORTANTE, $1.tipo no almacena el tipo del identificador, el tipo está en la tabla de símbolos).
  - El símbolo de la parte izquierda representa una dirección de memoria ($$.es_direccion=1).

Comprobaciones semánticas para otras expresiones

Implementación

- Las producciones (81) a (85) únicamente contienen en su acción semántica las siguientes instrucciones para propagar atributos:
  - $$.tipo = $1.tipo en las producciones 81 y 85.
  - $$.tipo = $2.tipo en las producciones 82 y 83.
  - $$.es_direccion = $1.es_direccion en las producciones 81 y 85.
  - $$.es_direccion = $2.es_direccion en las producciones 82 y 83.

Comprobaciones semánticas para otras expresiones

Implementación

- La producción 88 define la sintaxis de la llamada a una función:

  (88) exp: TOK_IDENTIFICADOR '(' lista_expresiones ')'

- La descripción de la semántica es la siguiente:
  - En las sentencias de llamadas a funciones, sólo es necesario comprobar la corrección del número de argumentos. No es necesario realizar ninguna comprobación de la correspondencia de tipos.
  - En las llamadas a funciones, no se puede escribir, en un parámetro, una llamada a otra función.

- En primer lugar, se comprueba que el identificador existe en la tabla de símbolos y además tiene categoría de función.