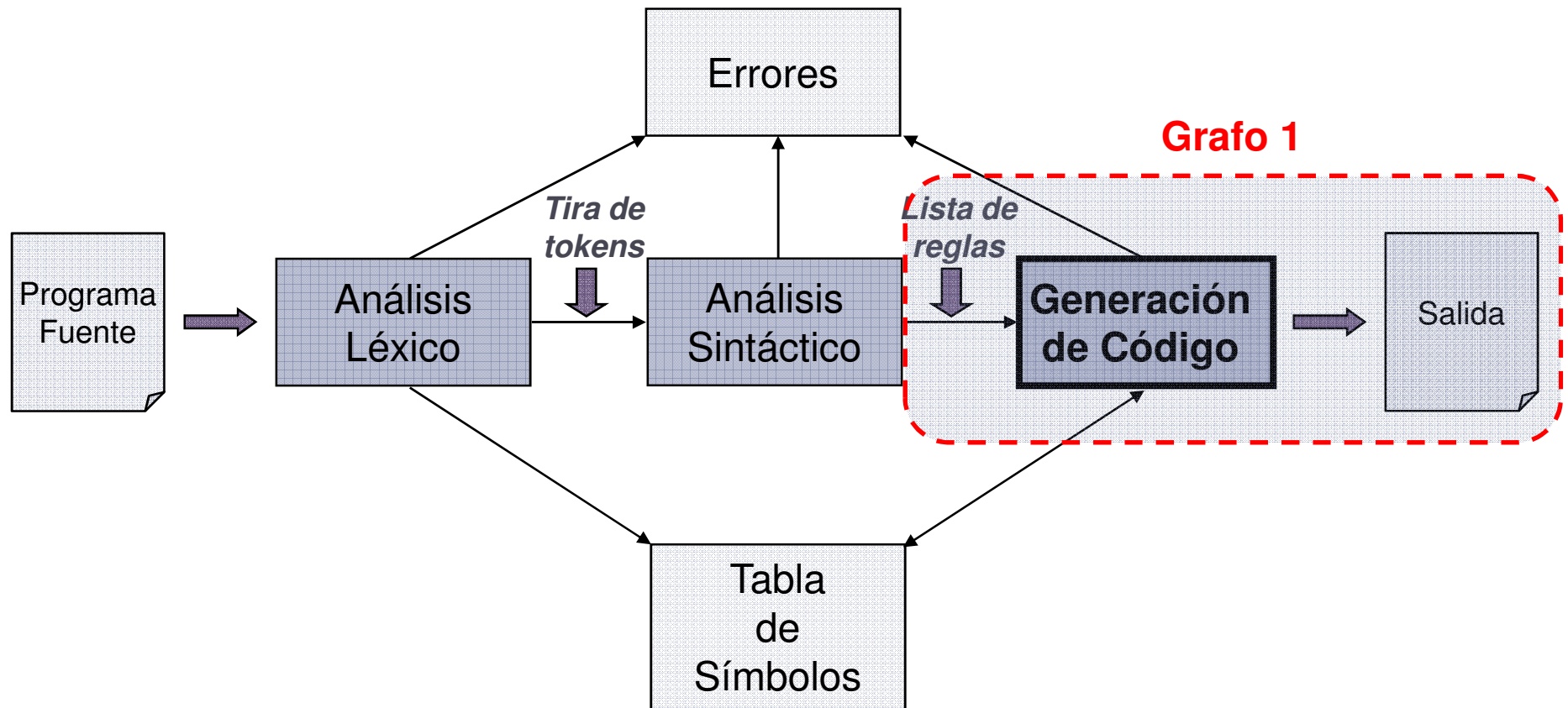


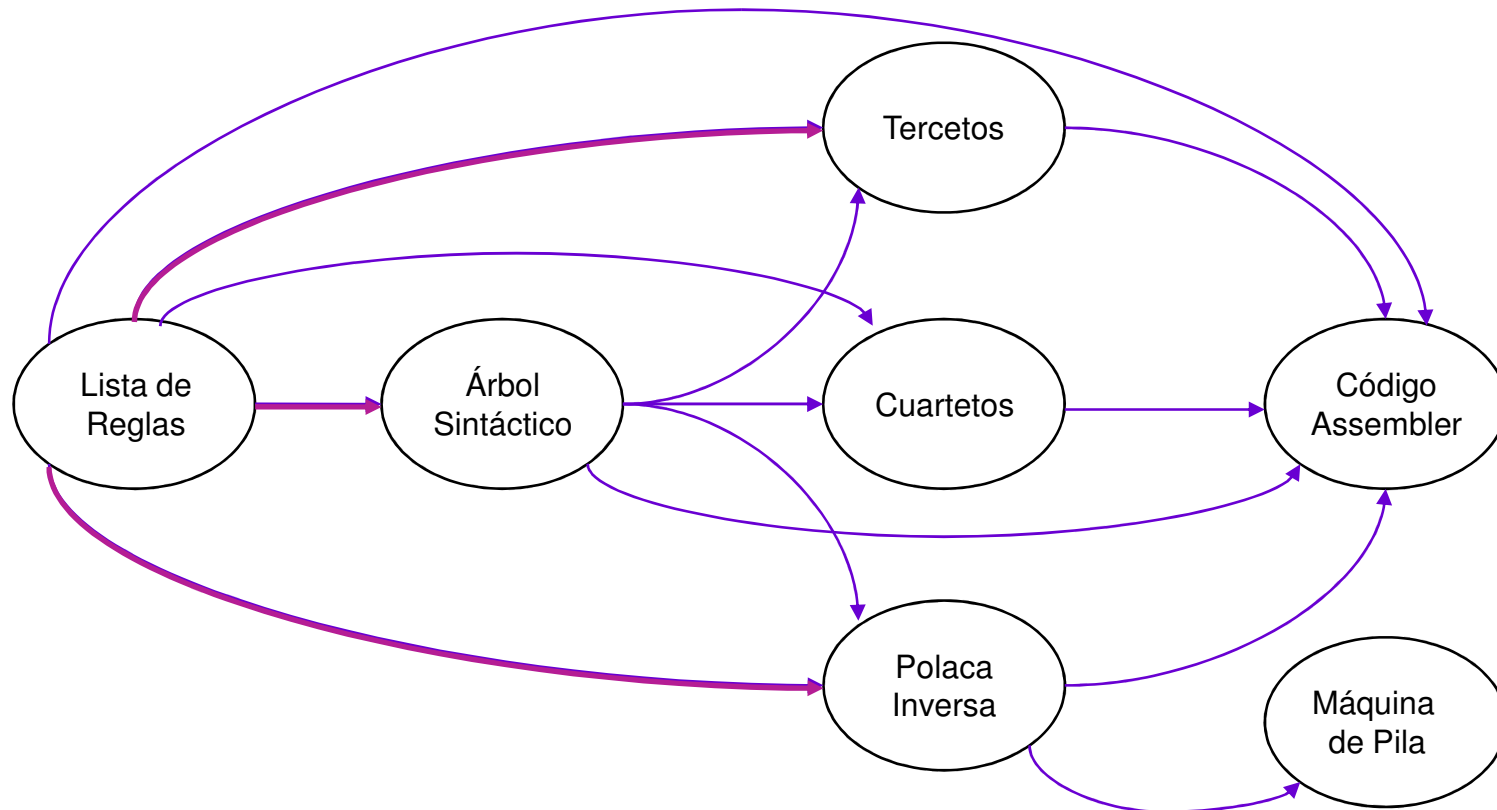
Diseño de Compiladores I

Generación de Código

Fases de la Compilación



Generación de Código



Generación de código intermedio para expresiones / asignaciones

Lista de Reglas → Árbol Sintáctico

Lista de Reglas → Tercetos

Lista de Reglas → Polaca Inversa

(Repaso)

Lista de Reglas \rightarrow Árbol Sintáctico

Reglas de la Gramática	Acciones Semánticas
1. $A \rightarrow id := E$	$A.ptr = crear_nodo(':='; crear_hoja(id.ptr); E.ptr)$ (*)
2. $E \rightarrow E + T$	$E.ptr = crear_nodo('+; E.ptr; T.ptr)$
3. $E \rightarrow T$	$E.ptr = T.ptr;$
4. $T \rightarrow T * F$	$T.ptr = crear_nodo('*; T.ptr; F.ptr)$
5. $T \rightarrow F$	$T.ptr = F.ptr;$
6. $F \rightarrow id$	$F.ptr = crear_hoja(id.ptr)$ (*)
7. $F \rightarrow cte$	$F.ptr = crear_hoja(cte.ptr)$ (**)

(*) $id.ptr$ = Ref a la entrada del identificador en la Tabla de Símbolos

(**) $cte.ptr$ = Ref a la entrada de la constante en la Tabla de Símbolos



Lista de Reglas \rightarrow Árbol Sintáctico

W := X + y * Z

w de tipo double

x de tipo integer

y de tipo float

z de tipo long

- a) Árbol Sintáctico sin información de tipos ni incorporación de conversiones (implícitas)
- b) Árbol Sintáctico con información de tipos
- c) Árbol Sintáctico con información de tipos e incorporación de conversiones (implícitas)



Lista de Reglas \rightarrow Árbol Sintáctico (b) con información de tipos

Reglas de la Gramática	Acciones Semánticas
1. $A \rightarrow id := E$	A.ptr = crear_nodo(':='; crear_hoja(id.ptr); E.ptr); (*) A.Tipo = id.tipo ;
2. $E \rightarrow E + T$	E.ptr = crear_nodo('+' ; E.ptr ; T.ptr) ; E.Tipo = f ₊ (E.tipo , T.tipo)
3. $E \rightarrow T$	E.ptr = T.ptr ; E.tipo = T.tipo ;
4. $T \rightarrow T * F$	T.ptr = crear_nodo('*' ; T.ptr ; F.ptr) ; T.tipo = f _* (T.tipo , F.tipo) ;
5. $T \rightarrow F$	T.ptr = F.ptr ; T.tipo = F.tipo ;
6. $F \rightarrow id$	F.ptr = crear_hoja(id.ptr) ; F.tipo = id.tipo ; (*)
7. $F \rightarrow cte$	F.ptr = crear_hoja(cte.ptr) ; F.tipo = cte.tipo ; (**)

Lista de Reglas \rightarrow Árbol Sintáctico (C)

con información de tipos e incorporación de conversiones

Reglas de la Gramática	Acciones Semánticas
1. $A \rightarrow id := E$	$E'.ptr = agregar_nodo_conversion(f:= [3] ; id.tipo ; E.tipo);$ $A.ptr = crear_nodo(':=' ; crear_hoja(id.ptr) ; E'.ptr);$ (*) $A.Tipo = id.tipo ;$
2. $E \rightarrow E + T$	$E'.ptr = agregar_nodo_conversion(f+[1] ; E.tipo ; T.tipo);$ $T'.ptr = agregar_nodo_conversion(f+[3] ; E.tipo ; T.tipo);$ $E.ptr = crear_nodo('+' ; E'.ptr ; T'.ptr);$ $E.Tipo = f_{+}[2](E.tipo , T.tipo);$
3. $E \rightarrow T$	$E.ptr = T.ptr;$ $E.tipo = T.tipo;$
4. $T \rightarrow T * F$	$T'.ptr = agregar_nodo_conversion(f*[1] ; T.tipo ; F.tipo);$ $F'.ptr = agregar_nodo_conversion(f*[3] ; T.tipo ; F.tipo);$ $T.ptr = crear_nodo('*' ; T'.ptr ; F'.ptr);$ $T.tipo = f_{*}[2](T.tipo , F.tipo);$
5. $T \rightarrow F$	$T.ptr = F.ptr ;$ $T.tipo = F.tipo ;$
6. $F \rightarrow id$	$F.ptr = crear_hoja(id.ptr) ;$ $F.tipo = id.tipo; (*)$
7. $F \rightarrow cte$	$F.ptr = crear_hoja(cte.ptr) ;$ $F.tipo = cte.tipo; (**)$

(*) id.ptr = Ref a la entrada del identificador en la Tabla de Símbolos (**) cte.ptr = Ref a la entrada de la constante en la Tabla de Símbolos

Lista de Reglas \rightarrow Tercetos

Reglas de la Gramática	Acciones Semánticas
1. $A \rightarrow id := E$	$A.ptr = crear_terceto(':='; id.ptr ; E.ptr) (*)$
2. $E \rightarrow E + T$	$E.ptr = crear_terceto('+'; E.ptr ; T.ptr)$
3. $E \rightarrow T$	$E.ptr = T.ptr;$
4. $T \rightarrow T * F$	$T.ptr = crear_terceto('*'; T.ptr ; F.ptr)$
5. $T \rightarrow F$	$T.ptr = F.ptr;$
6. $F \rightarrow id$	$F.ptr = id.ptr (*)$
7. $F \rightarrow cte$	$F.ptr = cte.ptr (**)$

(*) $id.ptr$ = Ref a la entrada del identificador en la Tabla de Símbolos

(**) $cte.ptr$ = Ref a la entrada de la constante en la Tabla de Símbolos



Lista de Reglas \rightarrow Tercetos

W := X + y * Z

w de tipo double

x de tipo integer

y de tipo float

z de tipo long

- a) Tercetos sin información de tipos ni incorporación de conversiones (implícitas)
- b) Tercetos con información de tipos
- c) Tercetos con información de tipos e incorporación de conversiones (implícitas)



Lista de Reglas \rightarrow Polaca Inversa

Reglas de la Gramática	Acciones Semánticas
1. $A \rightarrow id := E$	Agregar(id.ptr);Agregar(':= ') (*)
2. $E \rightarrow E + T$	Agregar('+')
3. $E \rightarrow T$	-
4. $T \rightarrow T * F$	Agregar('*')
5. $T \rightarrow F$	-
6. $F \rightarrow id$	Agregar(id.ptr) (*)
7. $F \rightarrow cte$	Agregar(cte.ptr) (**)

(*) id.ptr = Ref a la entrada del identificador en la Tabla de Símbolos

(**) cte.ptr = Ref a la entrada de la constante en la Tabla de Símbolos



Lista de Reglas \rightarrow Polaca Inversa

W := X + y * Z

w de tipo double

x de tipo integer

y de tipo float

z de tipo long

(a) Polaca Inversa sin información de tipos ni
incorporación de conversiones (implícitas)



Sentencias de Control

LISTA DE REGLAS → Árbol Sintáctico

LISTA DE REGLAS → Polaca Inversa

LISTA DE REGLAS → Tercetos

Generación de código para sentencias de control

- ▶ Lista de Reglas → Árbol Sintáctico
- ▶ Lista de Reglas → Polaca Inversa
- ▶ Lista de Reglas → Tercetos



Lista de Reglas → Árbol Sintáctico

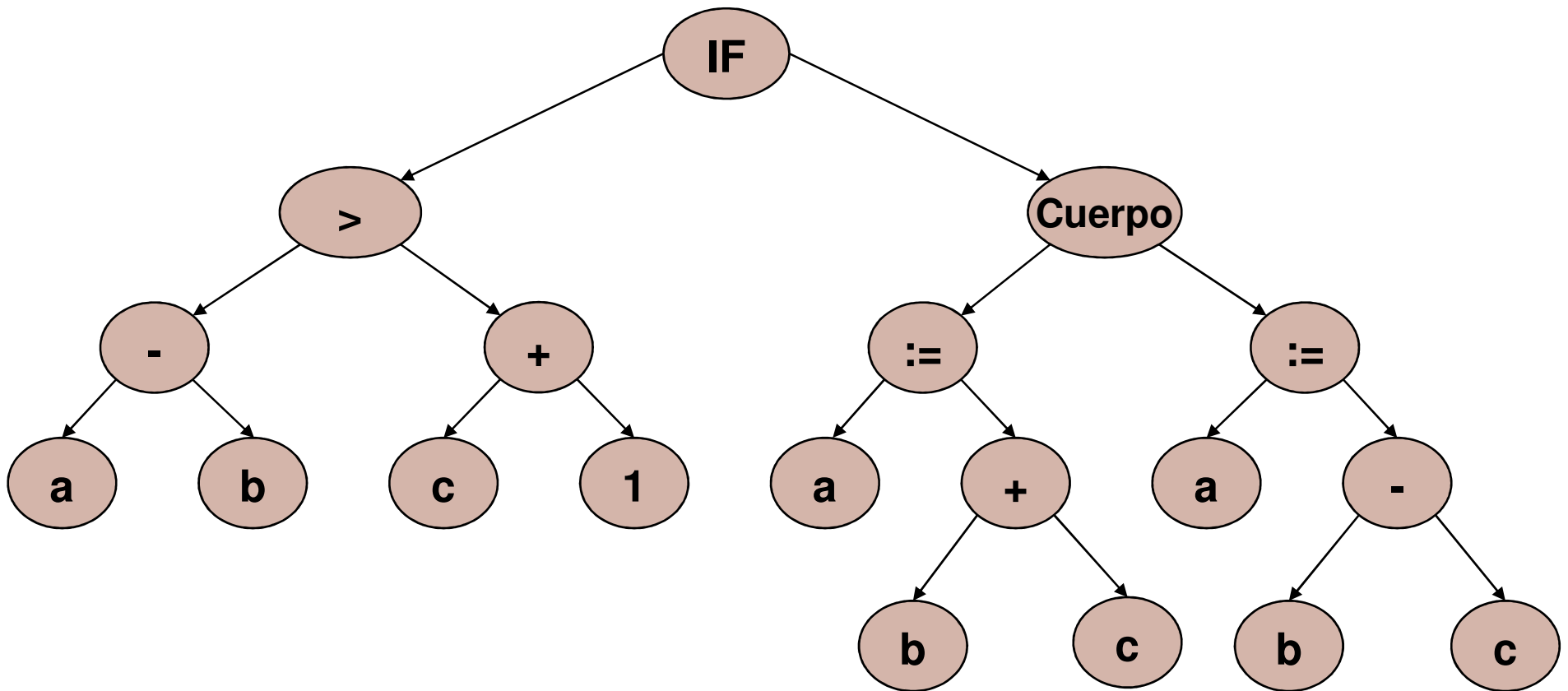
Sentencia IF

Generación de Código

Lista de Reglas \rightarrow Árbol Sintáctico

Ejemplo:

IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;



Generación de Código

Lista de Reglas → Árbol Sintáctico

<seleccion> → IF <condicon> THEN <cpo_then> ELSE <cpo_else>

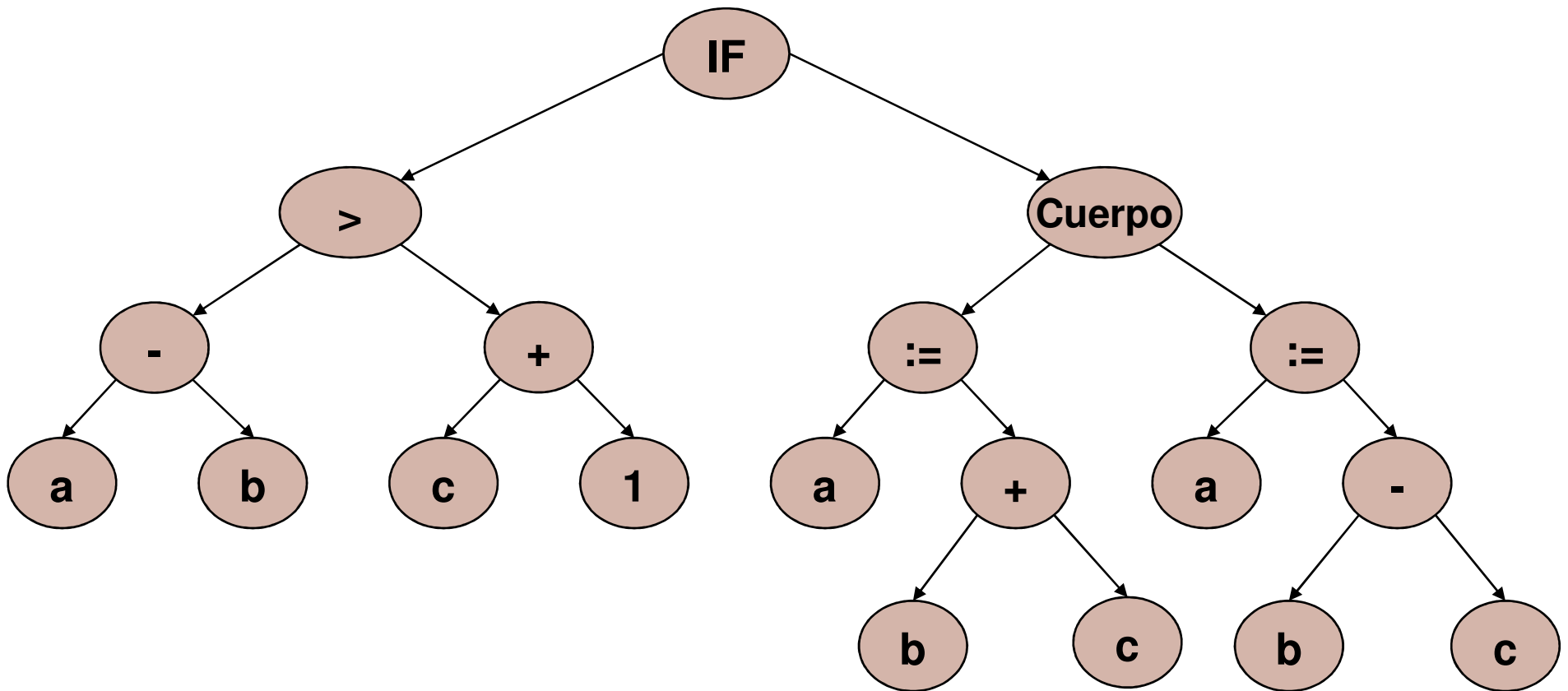
Reglas de la Gramática	Acciones Semánticas
<sel> → IF <cond> <cpo_if>	IF.ptr = crear_nodo("IF", C.ptr, C_l.ptr);
<cond> → '(' <exp_log> ')'	C.ptr = E_L.ptr;
<cpo_if> → <cpo_then> <cpo_else>	C_l.ptr = crear_nodo("cuerpo", C_T.ptr, C_E.ptr);
<cpo_if> → <cpo_then>	C_l.ptr = crear_nodo("cuerpo", C_T.ptr, null);
<cpo_then> → THEN <lista_sent>	C_T.ptr = L_S.ptr;
<cpo_else> → ELSE <lista_sent>	C_E.ptr = L_S.ptr;
<lista_sent> → ...	L_S.ptr = ...
<exp_log> → ...	E_L.ptr = ...



Lista de Reglas \rightarrow Árbol Sintáctico

Ejemplo:

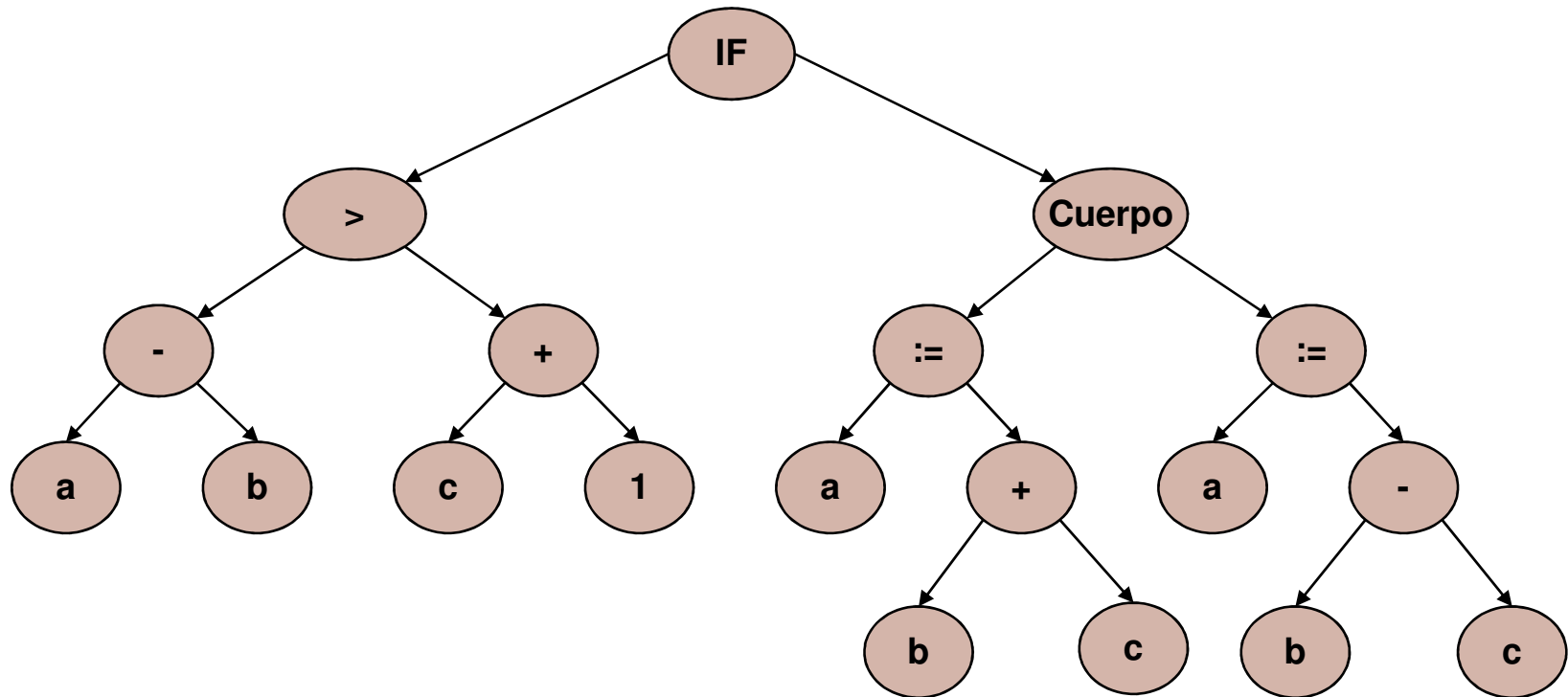
IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;



Generación de Código

Árbol Sintáctico → Polaca Inversa

~~IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;~~



a	b	-	c		+	>	a	b	c	+	:=	a	b	c	-	:=	Cpo	IF
---	---	---	---	--	---	---	---	---	---	---	----	---	---	---	---	----	-----	----

¿Problemas?

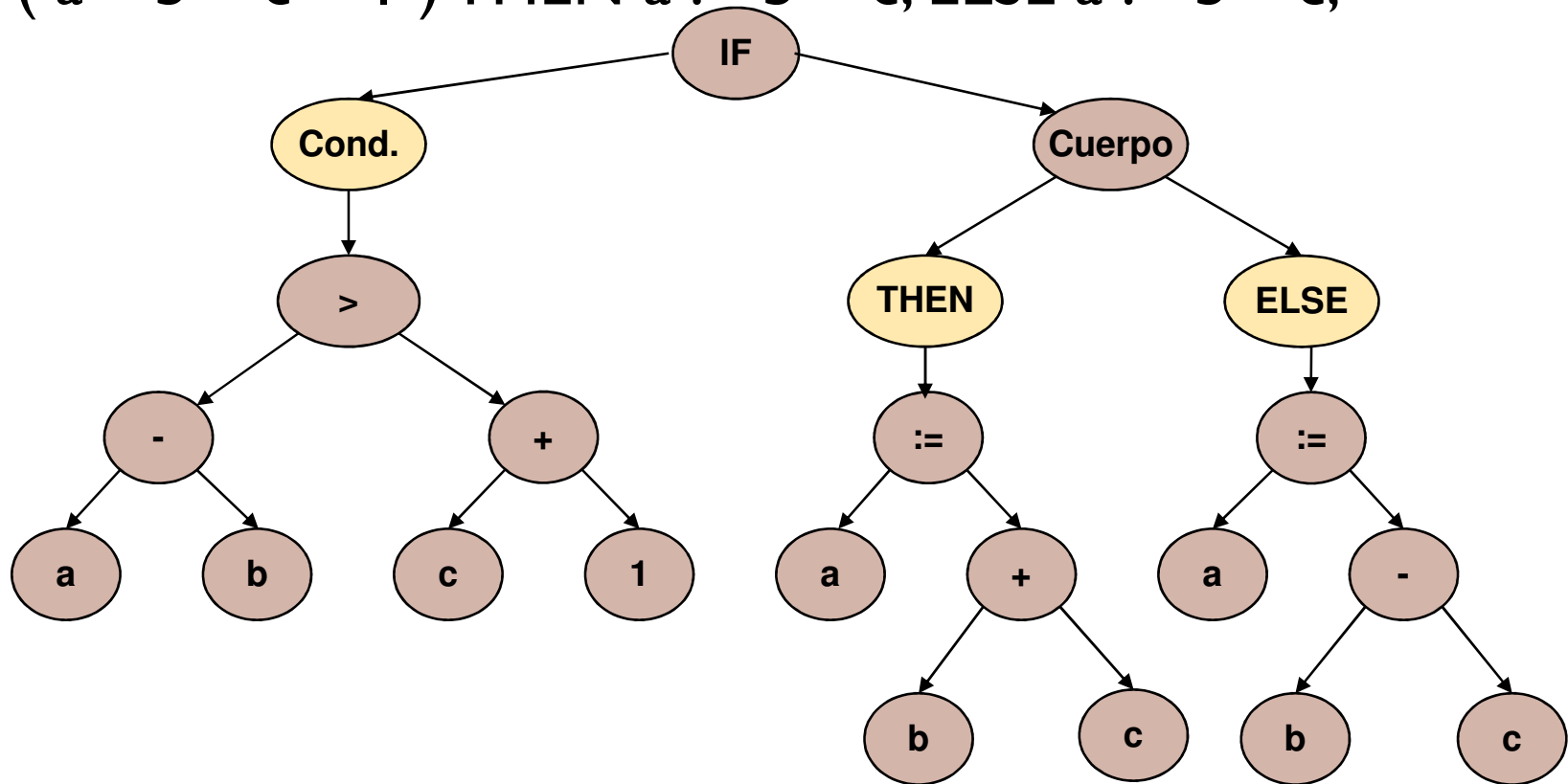
No es posible generar bifurcaciones en Assembler



Generación de Código

Árbol Sintáctico → Polaca Inversa

IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;



Se agregan nodos de control para el algoritmo de generación de bifurcaciones

a	b	-	c		+	>	cond	a	b	c	+	:=	then	a	b	c	-	:=	else	cpo	IF
---	---	---	---	--	---	---	------	---	---	---	---	----	------	---	---	---	---	----	------	-----	----



Generación de Código

Lista de Reglas → Árbol Sintáctico

<seleccion> → IF <condicion> THEN <cpo_then> ELSE <cpo_else>

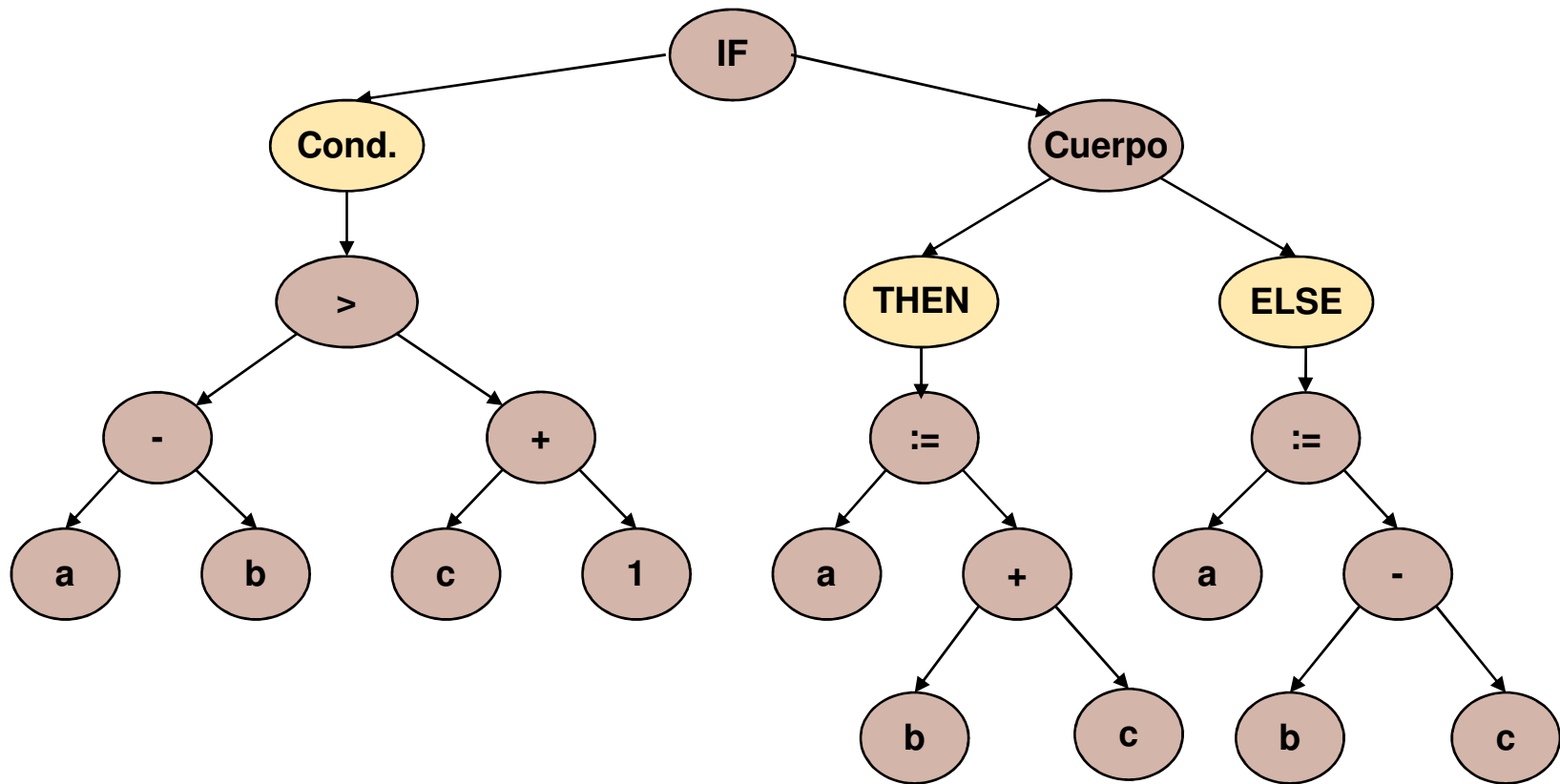


Reglas de la Gramática	Acciones Semánticas
<sel> → IF <cond> <cpo_if>	IF.ptr = crear_nodo("IF", C.ptr, C_l.ptr);
<cond> → '(' <exp_log> ')'	// Agregar acciones para: // Crear el nodo de control "cond"
<cpo_if> → <cpo_then> <cpo_else>	C_l.ptr = crear_nodo("cuerpo", C_T.ptr, C_E.ptr);
<cpo_if> → <cpo_then>	C_l.ptr = crear_nodo("cuerpo", C_T.ptr, null);
<cpo_then> → THEN <lista_sent>	// Agregar acciones para: // Crear el nodo de control "then"
<cpo_else> → ELSE <lista_sent>	// Agregar acciones para: // Crear el nodo de control "else"
<lista_sent> → ...	L_S.ptr = ...
<exp_log> → ...	E_L.ptr = ...



Lista de Reglas \rightarrow Árbol Sintáctico

IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;



El nuevo árbol tiene nodos de control para el algoritmo de generación de bifurcaciones

Lista de Reglas → Árbol Sintáctico

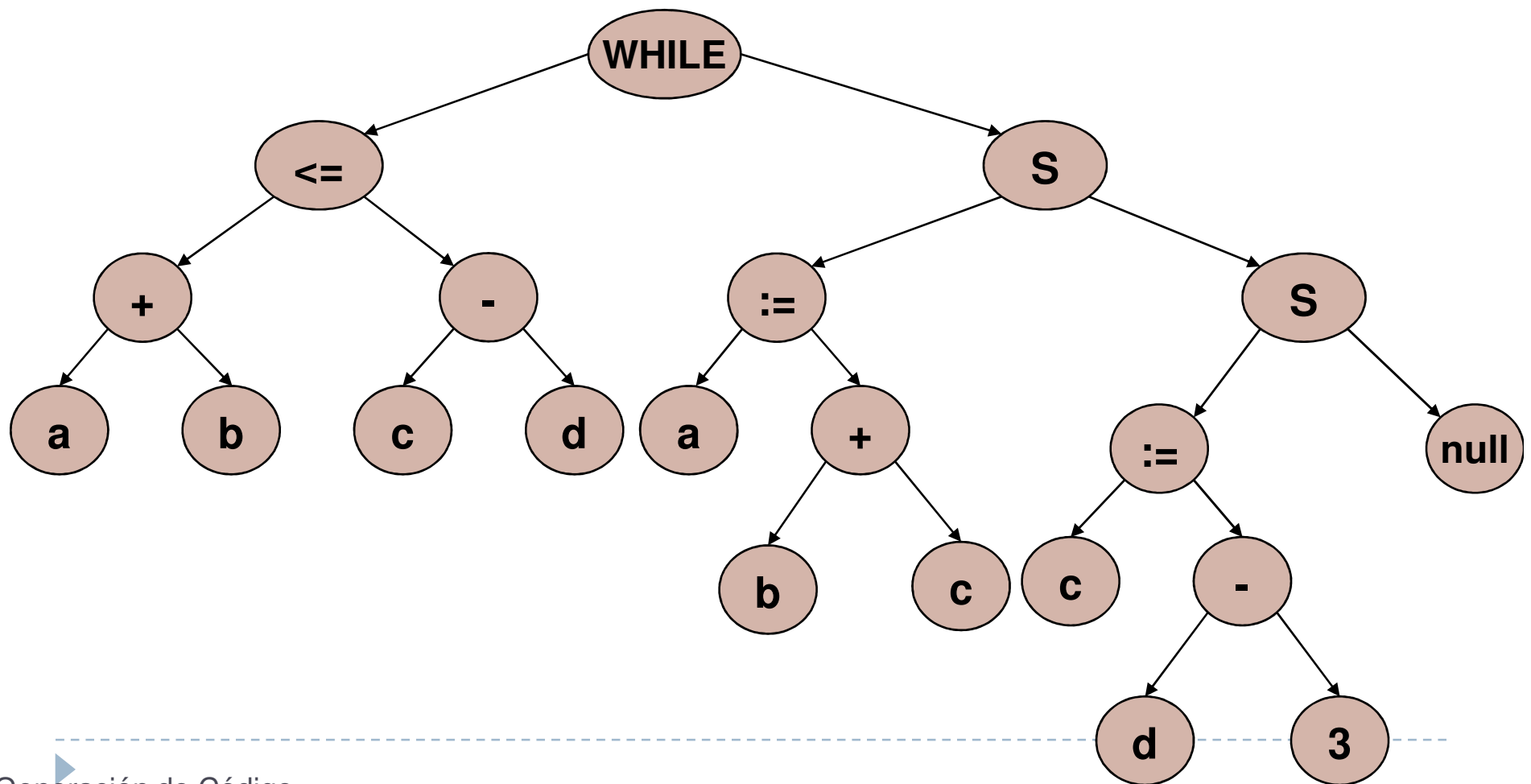
Sentencia WHILE

Generación de Código

Lista de Reglas \rightarrow Árbol Sintáctico

Ejemplo:

`WHILE (a + b <= c - d) { a := b + c; c := d - 3; }`



Lista de Reglas → Árbol Sintáctico

<iteracion> → WHILE <condicion> <cpo_while>

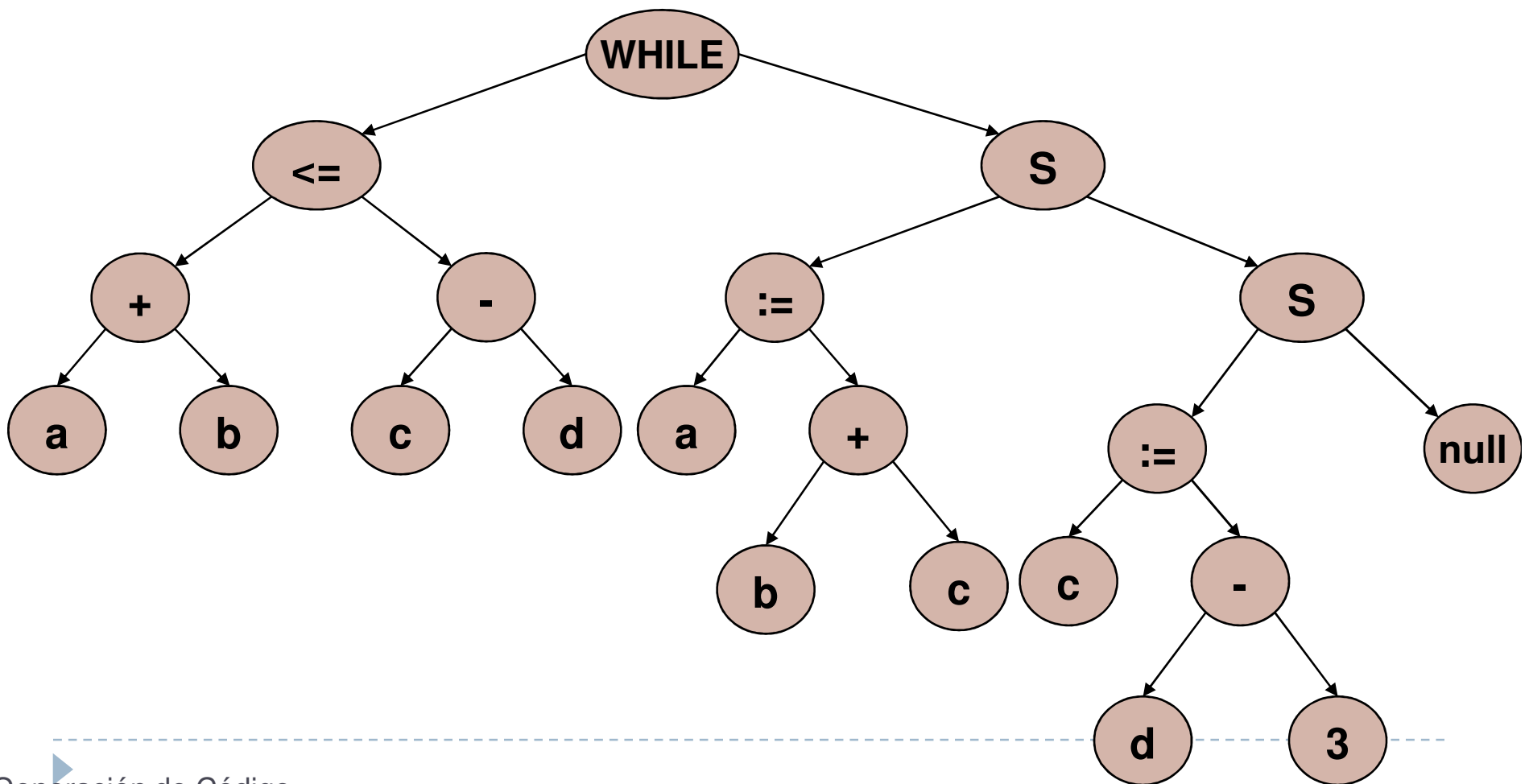
Reglas de la Gramática	Acciones Semánticas
<iter> → while <cond> <cpo_while>	l.ptr = crear_nodo("WHILE",C.ptr,C_W.ptr);
<cond> → '(' <exp_log> ''	C.ptr = E_L.ptr;
<cpo_while> → <lista_sent>	C_W.ptr = L_S.ptr;
<lista_sent> → ...	L_S.ptr = ...
<exp_log> → ...	E_L.ptr = ...



Lista de Reglas \rightarrow Árbol Sintáctico

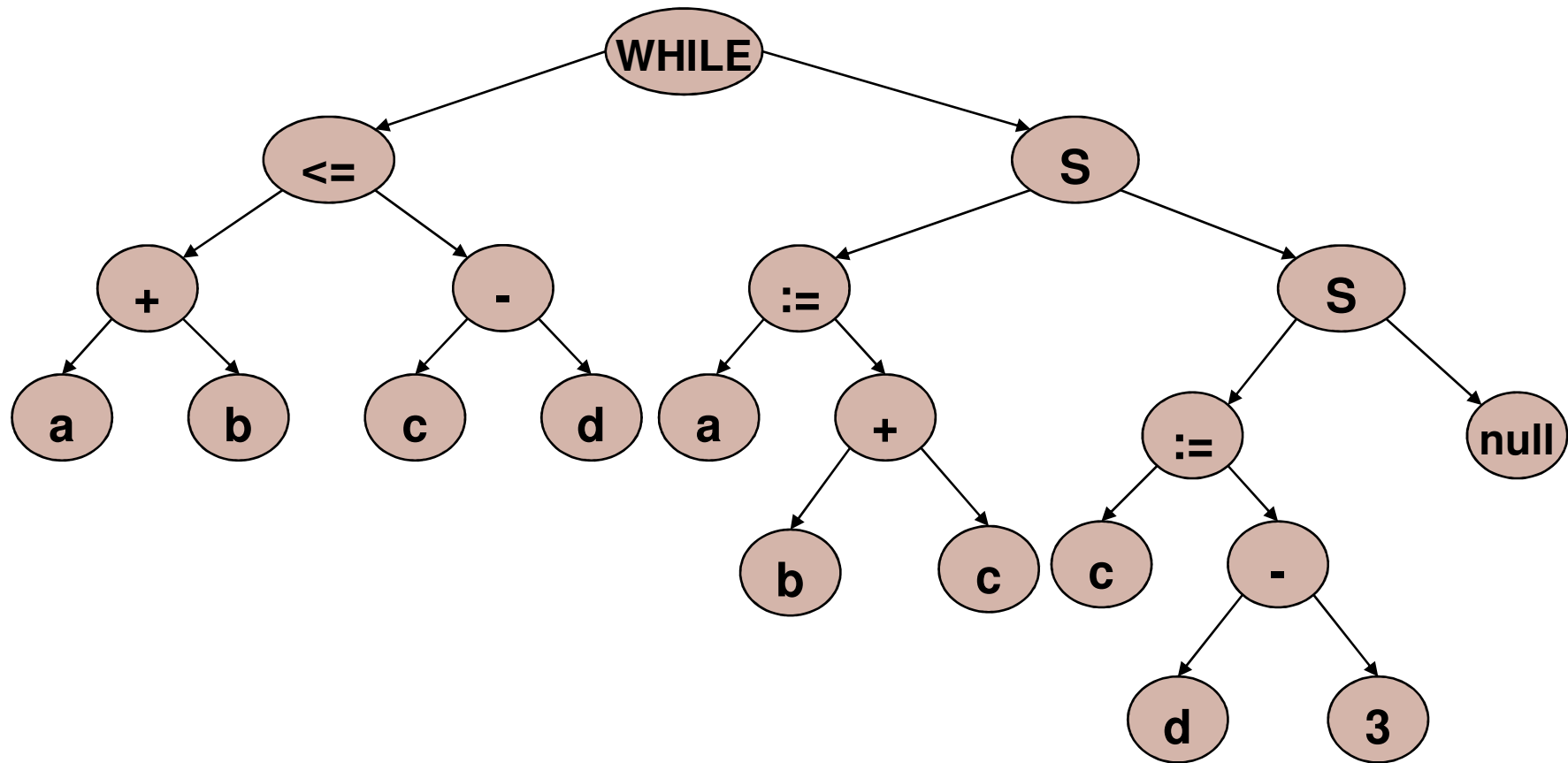
Ejemplo:

`WHILE (a + b <= c - d) { a := b + c; c := d - 3; }`



Árbol Sintáctico → Polaca Inversa

~~WHILE (a + b <= c - d) { a := b + c ; c := d - 3 ; }~~



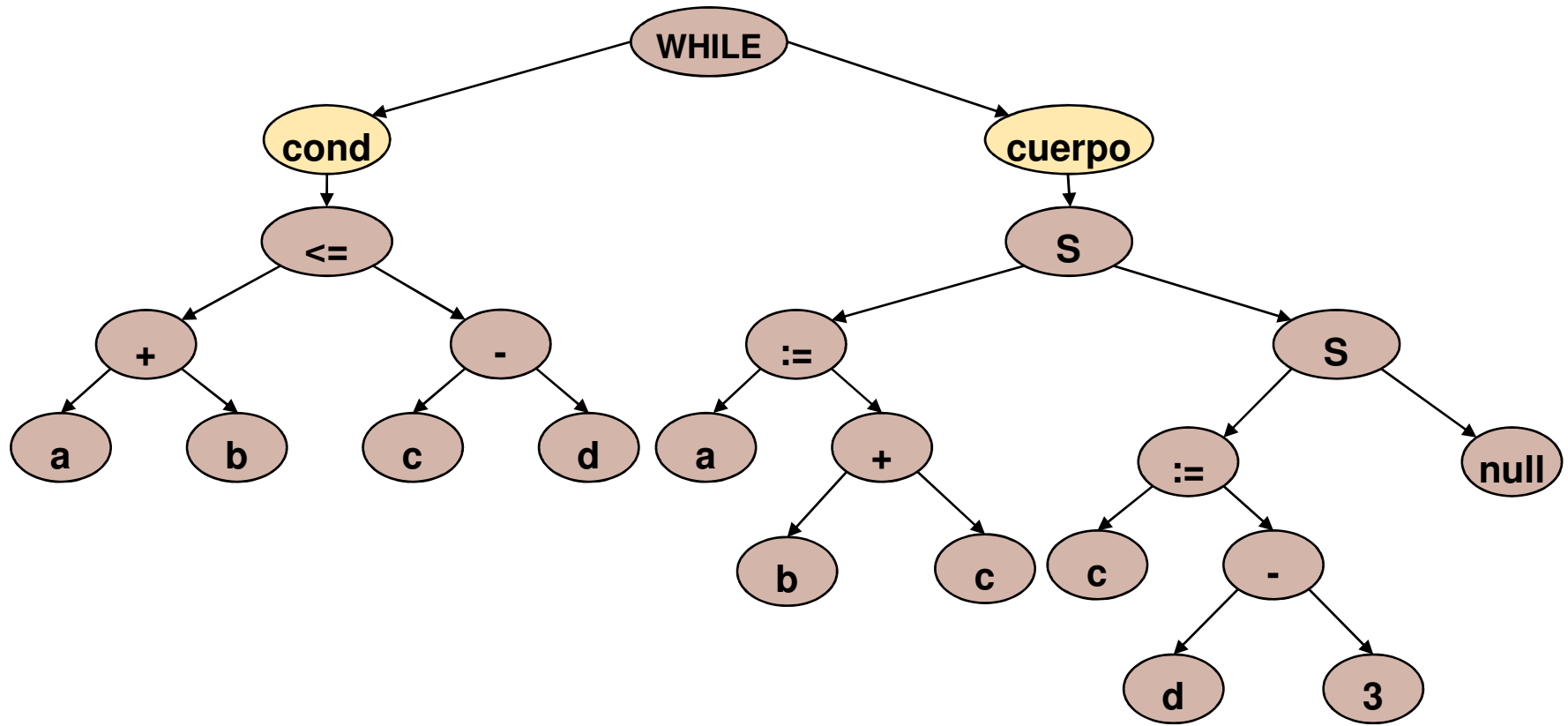
a	b	+	c	d	-	<=	a	b	c	+	:=	c	d	3	-	:=
---	---	---	---	---	---	----	---	---	---	---	----	---	---	---	---	----

~~No es posible detectar los puntos de inserción para generar bifurcaciones~~



Árbol Sintáctico → Polaca Inversa

WHILE (a + b <= c - d) { a := b + c; c := d - 3; }



Se agregan nodos de control para el algoritmo de generación de bifurcaciones

a	b	+	c	d	-	<=	cond	a	b	c	+	:=	c	d	3	-	:=	cpo
---	---	---	---	---	---	----	------	---	---	---	---	----	---	---	---	---	----	-----



Lista de Reglas → Árbol Sintáctico

<iteracion> → WHILE <condicion> <cpo_while>

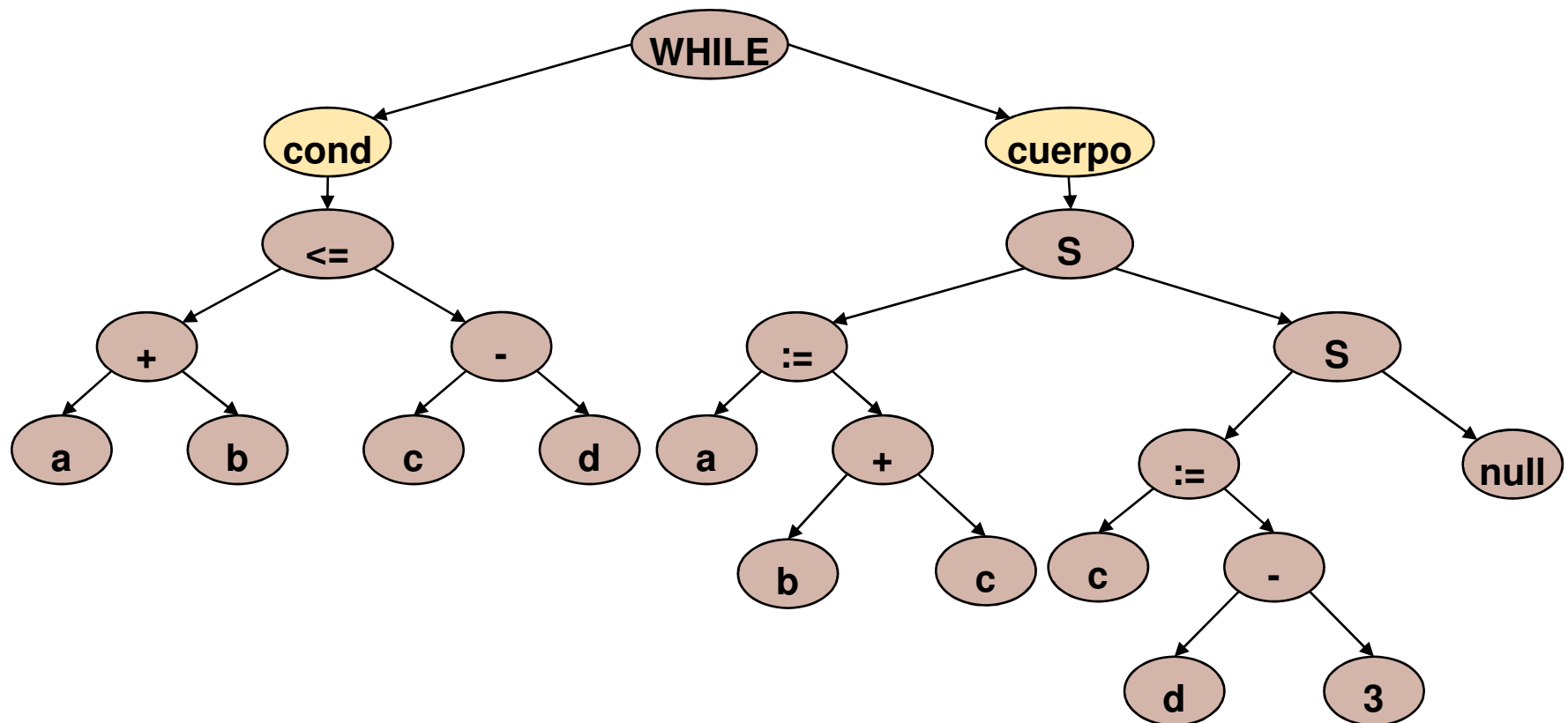
Reglas de la Gramática	Acciones Semánticas
<iter> → while <cond> <cpo_while>	l.ptr = crear_nodo("WHILE",C.ptr,C_W.ptr);
<cond> → '(' <exp_log> ')'	// Agregar acciones para: // Crear el nodo de control "cond"
<cpo_while> → <lista_sent>	// Agregar acciones para: // Crear el nodo de control "cuerpo"
<lista_sent> → ...	L_S.ptr = ...
<exp_log> → ...	E_L.ptr = ...



Lista de Reglas \rightarrow Árbol Sintáctico

Ejemplo:

WHILE (a + b <= c - d) { a := b + c; c := d - 3; }



El nuevo árbol tiene nodos de control para el algoritmo de generación de bifurcaciones

Lista de Reglas → Polaca Inversa

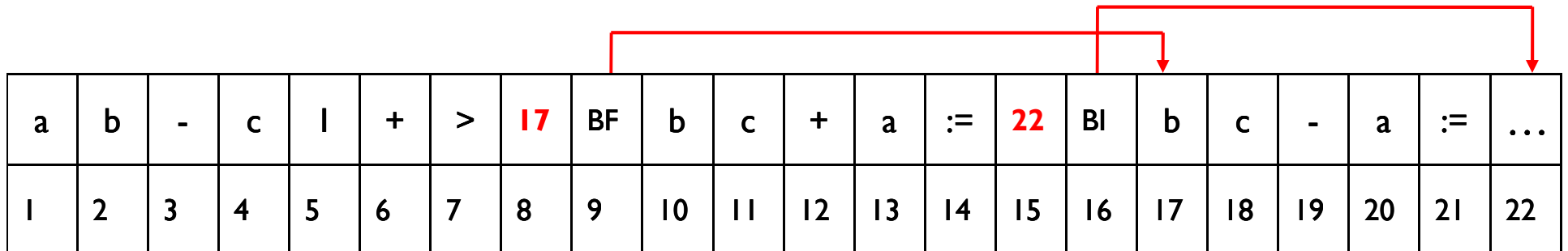
Sentencia IF

Generación de Código

Lista de Reglas \rightarrow Polaca Inversa

Ejemplo:

IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;



a	b	-	c	1	+	>	17	BF	b	c	+	a	:=	22	BI	b	c	-	a	:=	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Backpatching

Generación de Código

Lista de Reglas → Polaca Inversa

IF (a – b > c + l) THEN a := b + c; ELSE a := b – c;

•Generar BF incompleta
•Apilar paso incompleto

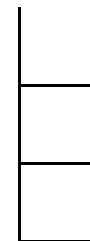
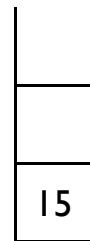
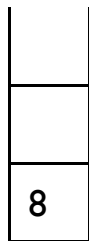
•Desapilar + Completar paso incompleto
•Generar BI con destino vacío
•Apilar paso incompleto

<seleccion> → IF <condicon> THEN <cpo_then> ELSE <cpo_else>

•Desapilar + Completar paso incompleto

○ Puntos de control para el algoritmo de generación de bifurcaciones

a	b	-	c	l	+	>	17	BF	b	c	+	a	:=	22	BI	b	c	-	a	:=	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22



Lista de Reglas → Polaca Inversa

<seleccion> → IF <condicon> THEN <cpo_then> ELSE <cpo_else>

Reglas de la Gramática	Acciones Semánticas
<sel> → IF <cond> <cpo_if>	// Agregar acciones para: // Desapilar // Completar el paso incompleto con el destino de la BI
<cond> → '(' <exp_log> ')'	// Agregar acciones para: // Generar los pasos de la BF incompleta // Apilar el número del paso incompleto
<cpo_if> → THEN <cpo_then> ELSE <cpo_else>	-
<cpo_if> → <cpo_then>	-
<cpo_then> → <lista_sent>	// Agregar acciones para: // Desapilar // Completar el paso incompleto con el destino de la BF // Generar los pasos de la BI incompleta // Apilar el número del paso incompleto
<cpo_else> → <lista_sent>	-
<lista_sent> → <i>// Se crean los pasos para las sentencias de la lista</i>
<exp_log> → ...	E_L.ptr = ... <i>// Se crean los pasos de la expresión</i>

Lista de Reglas \rightarrow Polaca Inversa

Ejemplo:

IF (a - b > c + 1) THEN a := b + c; ELSE a := b - c;

a	b	-	c		+	>		BF	b	c	+	a	:=		BI	b	c	-	a	:=	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22



Lista de Reglas → Polaca Inversa

Sentencia WHILE

Generación de Código

Lista de Reglas \rightarrow Polaca Inversa

Ejemplo:

WHILE (a + b <= c - d) { a := b + c; c := d - 3;};

a	b	+	c	d	-	<=	22	BF	b	c	+	a	:=	d	3	-	c	:=	I	BI	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

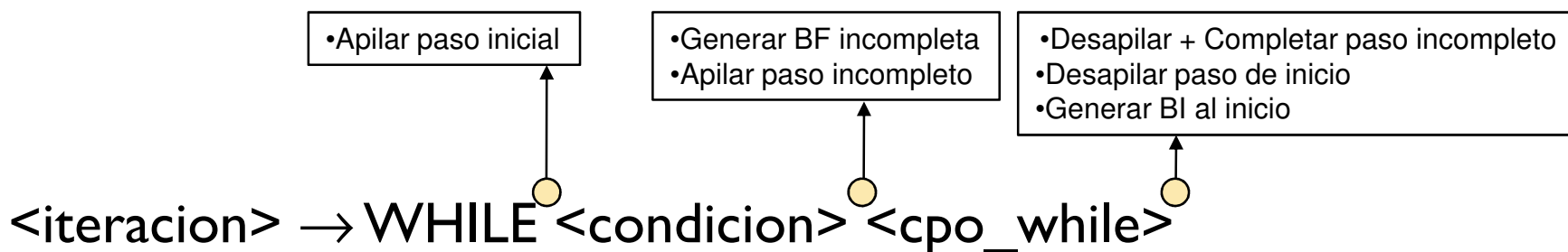
Backpatching



Generación de Código

Lista de Reglas → Polaca Inversa

WHILE (a + b <= c - d) { a := b + c; c := d - 3; }



○ Puntos de control para el algoritmo de generación de bifurcaciones

a	b	+	c	d	-	<=	22	BF	b	c	+	a	:=	d	3	-	c	:=	 	BI	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

8

	-

Lista de Reglas → Polaca Inversa

<iteracion> → WHILE <condicion> <cpo_while>

Reglas de la Gramática	Acciones Semánticas
<iter> → <inicio_while> <cond> <cpo_while>	// Agregar acciones para: // Desapilar // Completar el paso incompleto con el destino de la BF // Desapilar paso de inicio // Generar los pasos de la BI
<inicio_while> → WHILE	// Agregar acciones para: // Apilar el número del paso donde comienza la condición
<cond> → '(' <exp_log> ')'	// Agregar acciones para: // Generar los pasos de la BF incompleta // Apilar el número del paso incompleto
<cpo_while> → <lista_sent>	-
<lista_sent> → <i>// Se crean los pasos para las sentencias de la lista</i>
<exp_log> → ...	E_L.ptr = ... <i>// Se crean los pasos de la expresión</i>



Lista de Reglas \rightarrow Polaca Inversa

Ejemplo:

WHILE (a + b <= c - d) { a := b + c; c := d - 3;};

a	b	+	c	d	-	<=	22	BF	b	c	+	a	:=	d	3	-	c	:=		BI	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Generación de Código

Lista de Reglas → Tercetos

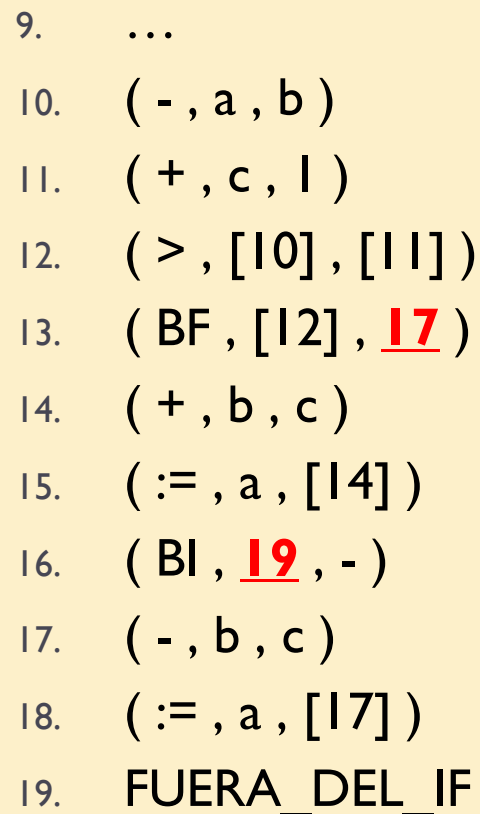
Sentencia IF

Generación de Código

Lista de Reglas \rightarrow Tercetos

Ejemplo:

IF ($a - b > c + 1$) THEN $a := b + c$; ELSE $a := b - c$;



```
9.  ...
10. ( -, a , b )
11. ( + , c , 1 )
12. ( > , [10] , [11] )
13. ( BF , [12] , 17 )
14. ( + , b , c )
15. ( := , a , [14] )
16. ( BI , 19 , - )
17. ( - , b , c )
18. ( := , a , [17] )
19. FUERA_DEL_IF
```

Backpatching


Generación de Código

Lista de Reglas \rightarrow Tercetos

Ejemplo:

IF ($a - b > c + 1$) THEN $a := b + c$; ELSE $a := b - c$;

```
9.    ...
10.   ( - , a , b )
11.   ( + , c , 1 )
12.   ( > , [10] , [11] )
13.   ( BF , [12] , 17 )
14.   ( + , b , c )
15.   ( := , a , [14] )
16.   ( BI , 19 , - )
17.   ( - , b , c )
18.   ( := , a , [17] )
19.   FUERA_DEL_IF
```



Lista de Reglas → Tercetos

<seleccion> → IF <condicon> THEN <cpo_then> ELSE <cpo_else>



Reglas de la Gramática	Acciones Semánticas
<sel> → IF <cond> <cpo_if>	// Agregar acciones para: // Desapilar // Completar el terceto incompleto con el destino de la BI
<cond> → '(' <exp_log> ')'	// Agregar acciones para: // Crear terceto incompleto para la BF // Apilar el número del terceto incompleto
<cpo_if> → <cpo_then> ELSE <cpo_else>	-
<cpo_if> → <cpo_then>	-
<cpo_then> → <lista_sent>	// Agregar acciones para: // Desapilar // Completar el terceto incompleto con el destino de la BF // Crear terceto incompleto para la BI // Apilar el número del terceto incompleto
<cpo_else> → <lista_sent>	-
<lista_sent> → <i>// Se crean los tercetos para las sentencias de la lista</i>
<exp_log> → ...	E_L.ptr = ... <i>// Se crean los tercetos de la expresión</i>

Lista de Reglas \rightarrow Tercetos

Ejemplo:

IF ($a - b > c + 1$) THEN $a := b + c$; ELSE $a := b - c$;

```
9.    ...
10.   ( -, a , b )
11.   ( + , c , 1 )
12.   ( > , [10] , [11] )
13.   ( BF , [12] , 17 )
14.   ( + , b , c )
15.   ( := , a , [14] )
16.   ( BI , 19 , - )
17.   ( - , b , c )
18.   ( := , a , [17] )
19.   FUERA_DEL_IF
```

Lista de Reglas → Tercetos

Sentencia WHILE

Generación de Código

Lista de Reglas \rightarrow Tercetos

Ejemplo:

WHILE (a + b <= c - d) { a := b + c; c := d - 3;};

```
24. ...
25. ( + , a , b )
26. ( - , c , d )
27. ( <= , [25] , [26] )
28. ( BF , [27] , 34 )
29. ( + , b , c )
30. ( := , a , [29] )
31. ( - , d , 3 )
32. ( := , c , [31] )
33. ( BI , 25 , - )
34. FUERA_DEL_WHILE
```

Backpatching


Generación de Código

Lista de Reglas \rightarrow Tercetos

Ejemplo:

WHILE (a + b <= c - d) { a := b + c; c := d - 3;};

```
24. ...
25. ( + , a , b )
26. ( - , c , d )
27. ( <= , [25] , [26] )
28. ( BF , [27] , 34 )
29. ( + , b , c )
30. ( := , a , [29] )
31. ( - , d , 3 )
32. ( := , c , [31] )
33. ( BI , 25 , - )
34. FUERA_DEL_WHILE
```



Generación de Código

Lista de Reglas → Tercetos

<iteracion> → WHILE  <condicion>  <cpo_while> 

Reglas de la Gramática	Acciones Semánticas
<iter> → <inicio_while> <cond> <cpo_while>	// Agregar acciones para: // Desapilar // Completar el terceto incompleto con el destino de la BF // Desapilar número de terceto de inicio // Crear el terceto para la BI
<inicio_while> → WHILE	// Agregar acciones para: // Apilar el número del terceto donde comienza la condición
<cond> → '(' <exp_log> ')'	// Agregar acciones para: // Crear terceto incompleto para la BF // Apilar el número del terceto incompleto
<cpo_while> → <lista_sent>	-
<lista_sent> → <i>// Se crean los tercetos para las sentencias de la lista</i>
<exp_log> → ...	E_L.ptr = ... <i>// Se crean los tercetos de la expresión</i>



Lista de Reglas \rightarrow Tercetos

Ejemplo:

WHILE (a + b <= c - d) { a := b + c; c := d - 3;};

```
24.  ...
25.  ( + , a , b )
26.  ( - , c , d )
27.  ( <= , [25] , [26] )
28.  ( BF , [27] , 34 )
29.  ( + , b , c )
30.  ( := , a , [29] )
31.  ( - , d , 3 )
32.  ( := , c , [31] )
33.  ( BI , 25 , - )
34.  FUERA_DEL_WHILE
```

Ejercicios

- ▶ Para un lenguaje que permite la sentencia DO UNTIL
- ▶ Considerando el camino Lista de Reglas → Polaca Inversa

1. ¿Cuál sería la representación intermedia para la siguiente sentencia?

```
DO
BEGIN
    a := a * 2;
    x := a;
END
UNTIL ( x > 10)
```

2. ¿Cuáles serían las acciones semánticas para poder generar esta representación?



Ejercicios

- ▶ Para un lenguaje que permite la sentencia FOR
- ▶ Considerando el camino Lista de Reglas → Tercetos

1. ¿Cuál sería la representación intermedia para la siguiente sentencia?

```
FOR (i := 0; i < 10; i++)  
BEGIN  
    a := a * 2;  
    x := a;  
END
```

2. ¿Cuáles serían las acciones semánticas para poder generar esta representación?



Ejercicio

- ▶ Dada la siguiente gramática de expresiones:

$A \rightarrow id = E$

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow id \mid cte$

Para un lenguaje que admite datos de tipo int, y de tipo float, y conversiones implícitas.

- ▶ Si el camino de generación de código es:

Lista de Reglas \rightarrow Polaca Inversa \rightarrow Assembler

¿Cómo incorpora las conversiones el compilador?

