

Tema 10: Optimización de Código Intermedio

1. Introducción
2. Bloque básico y grafo de flujo
3. Optimizaciones locales
 - 3.1. Transformaciones algebraicas
 - 3.2. Transformaciones que preservan la estructura
 - 3.2.1. Introducción al análisis de flujo de datos
 - 3.2.2. Transformaciones
 - 3.3. Grafos dirigidos acíclicos (GDA)
4. Optimizaciones globales
 - 4.1. Técnicas básicas
 - 4.2. Detección de bucles
 - 4.3. Extracción de código invariante
 - 4.4. Reducción de intensidad y eliminación variables inducción.



1. Introducción

- Criterios para optimizar
 - Tiempo
 - Tamaño
- Condiciones de las transformaciones usadas
 - Preservar significado programa
 - Mejora cuantificable
 - Debe merecer la pena (coste/mejora)
- Fuente de la optimización
 - Programa fuente
 - Código intermedio
 - Código objeto



2. Bloque básico y grafo de flujo

Bloque básico

Bloque básico

Secuencia de instrucciones consecutivas en las que el flujo de control **entra al principio** y **sale al final**, sin detenerse y sin posibilidad de saltar, excepto al final.

Llamaremos **líder** de un bloque básico a la primera instrucción del bloque básico.

Grafo de flujo

Grafo dirigido que representa el flujo de control de un programa.

Bloque básico \rightarrow **nodo**

Nodo inicial: Bloque básico que contiene la primera instrucción del programa.

Arista dirigida $X \rightarrow Y$ si Y puede ir inmediatamente después de X en alguna secuencia de ejecución del programa.

Algoritmo división en bloques básicos

Entrada: Una secuencia de instrucciones de tres direcciones.

Salida: Una lista de bloques básicos

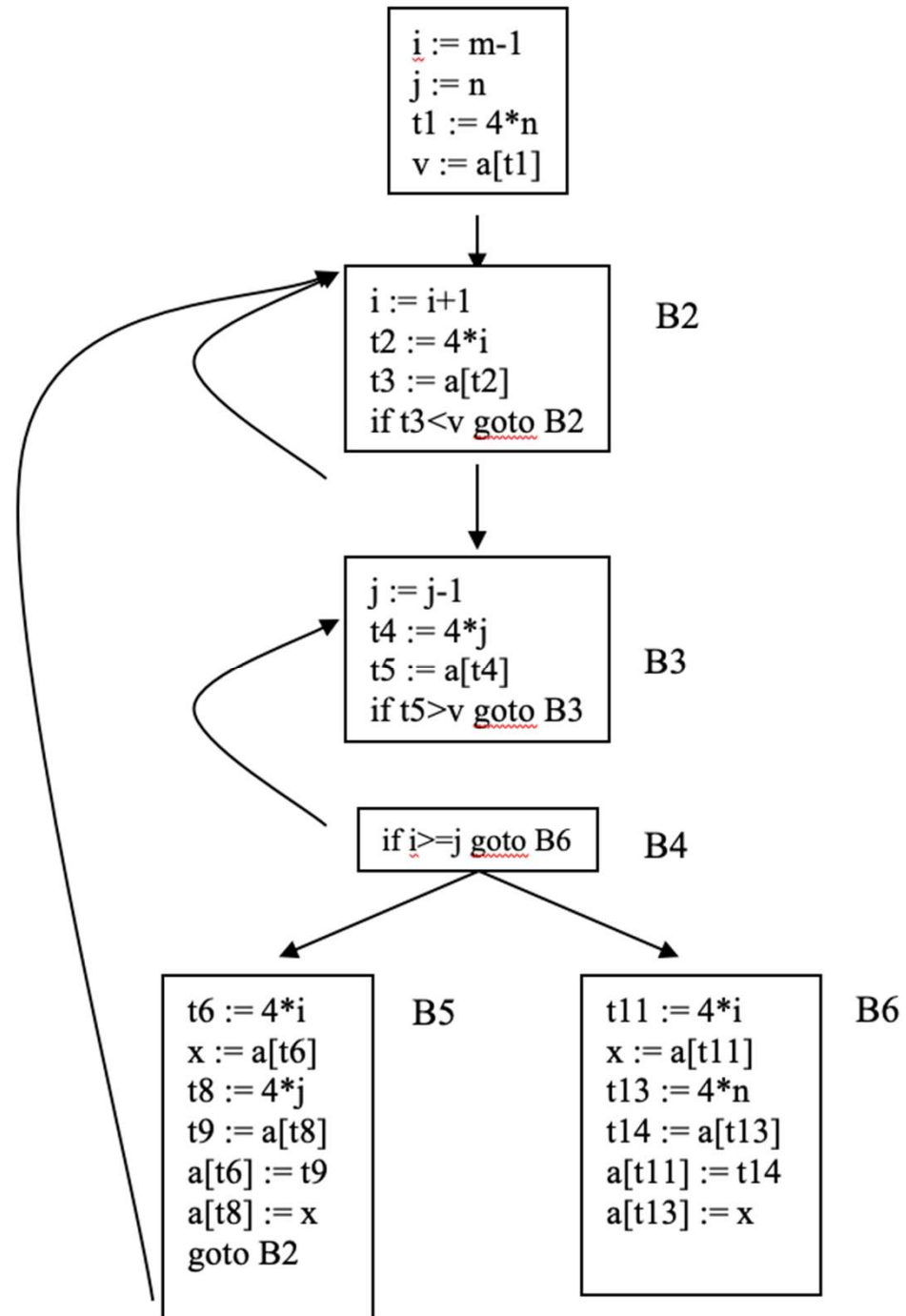
Método:

1. Se determina el conjunto de *líderes* (la primera proposición de cada *bloque básico*).
 1. La **primera** proposición es un líder
 2. Cualquier prop. **destino** de un salto es un líder.
 3. Cualquier prop. que vaya **después de un salto** es un líder.
2. Para cada líder, su bloque básico consta del líder y de todas las proposiciones hasta, sin incluir, el siguiente líder o el fin del programa.

Ejemplo 1

```
131:    i := m-1
132:    j := n
133:    t1 := 4*n
134:    v := a[t1]
135:    i := i+1
136:    t2 := 4*i
137:    t3 := a[t2]
138:    if t3<v goto 135
139:    j := j-1
140:    t4 := 4*j
141:    t5 := a[t4]
142:    if t5>v goto 139
143:    if i>=j goto 153
144:    t6 := 4*i
145:    x := a[t6]
```

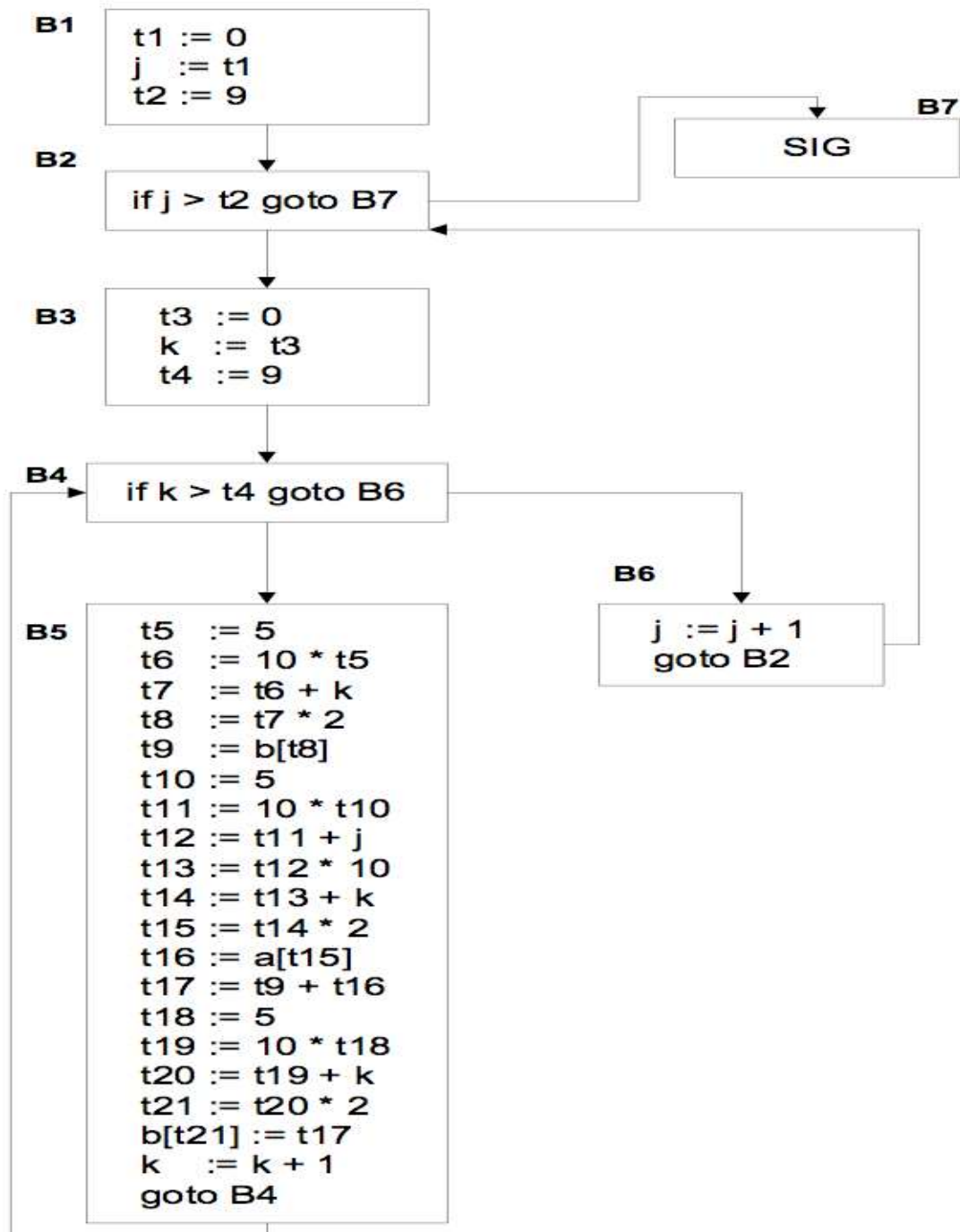
```
146:    t7 := 4*i
147:    t8 := 4*j
148:    t9 := a[t8]
149:    a[t7] := t9
150:    t10 := 4*j
151:    a[t10] := x
152:    goto 135
153:    t11 := 4*i
154:    x := a[t11]
155:    t12 := 4*i
156:    t13 := 4*n
157:    t14 := a[t13]
158:    a[t12] := t14
159:    t15 := 4*n
160:    a[t15] := x
```



Ejemplo 2

```
      t1 := 0
      j := t1
      t2 := 9
L1:    if j > t2 goto L3
      t3 := 0
      k := t3
      t4 := 9
L2:    if k > t4 goto L4
      t5 := 5
      t6 := t5 * 10
      t7 := t6 + k
      t8 := t7 * 2
      t9 := b[t8]
      t10:= 5
      t11:= t10 * 10
      t12:= t11 + j
```

```
      t13:= t12 * 10
      t14:= t13 + k
      t15:= t14 * 2
      t16:= a[t15]
      t17:= t9 + t16
      t18:= 5
      t19:= t18 * 10
      t20:= t19 + k
      t21:= t20 * 2
      b[t21]:= t17
      k := k + 1
      goto L2
L4:    j := j + 1
      goto L1
L3:
```





3. Optimizaciones locales

Optimizaciones locales

- Transformaciones que pueden aplicarse observando únicamente un bloque básico¹.

Optimizaciones globales

- Transformaciones que para aplicarse es necesario conocer el flujo de control entre los distintos bloques básicos del programa.

¹ Sus instrucciones y las variables activas a la entrada y salida del bloque básico. 12



3.1. Transformaciones algebraicas

1. *Simplificaciones algebraicas:*

- Expresiones de identidad: identidad $\theta \alpha = \alpha$
- Propiedad conmutativa: $\alpha \theta \beta = \beta \theta \alpha$
- Propiedad asociativa: $\alpha \theta (\beta \theta \gamma) = (\alpha \theta \beta) \theta \gamma$
- Operador θ_1 distributivo respecto a θ_2 :
$$\alpha \theta_1 (\beta \theta_2 \gamma) = (\alpha \theta_1 \beta) \theta_2 (\alpha \theta_1 \gamma)$$
- Operador unario autoinverso: $\theta \theta \alpha = \alpha$
$$A*(B*C)+(B*A)*D+A*E = A*(B*(C+D)+E)$$

2. Reducción de intensidad.

Sustitución de una operación por otra algebraicamente equivalente pero con menor coste computacional

3. Cálculo previo de constantes (folding)

```
y := 3  
t1 := 2 + y  
t2 := t1 + z  
x := t2 + 5    →    x := 10 + z
```

Precauciones:

- Deben aplicarse localmente
 - (1) $i := 0$
 - (2) $i := i + 1$
 - (3) if $i < N$ goto 2
- Números en coma flotante



3.2. Transformaciones que preservan la estructura

Introducción al análisis de flujo de datos

(i) $a := b + c$

$\text{Def}[i] = \{a\}$ $\text{Usa}[i] = \{b, c\}$

Variable activa en un arco del grafo de flujo, si existe camino dirigido desde ese arco a algún uso de la variable que no pasa por ninguna definición de la variable.

X está activa a la entrada de un nodo n ($x \in \text{ent}[n]$) si está activa en cualquiera de los arcos de entrada del nodo.

x está activa a la salida de un nodo n ($x \in \text{sal}[n]$) si está activa en cualquiera de los arcos de salida del nodo.

$$\text{ent}[i] = \text{usa}[i] \cup (\text{sal}[i] - \text{def}[i])$$

$$\text{sal}[i] = \bigcup_{s \in \text{Suc}[i]} \text{ent}[s]$$

Introducción al análisis de flujo de datos

Cálculo de $ent[]$ y $sal[]$ en un bloque básico $(\{int_i\}_1^n, E, S)$.

para todo $i \in 1..n$ hacer $ent[i] := \{\}; \text{ sal}[i] := \{\};$

Repetir

para $i := n$ hasta 1 hacer

si $i = n$ entonces $sal[i] := \bigcup_{s \in Suc[i]} ent[s] \cup S$

sino $sal[i] := \bigcup_{s \in Suc[i]} ent[s]$

$ent[i] := usa[i] \cup (sal[i] - def[i])$

fin para

hasta ningún $ent[i]$ o $sal[i]$ cambie.


Ejemplo 3

- (1) $x := \dots$
- (2) $x := y + z$
- (3) $a := x + b$
- (4) $y := a + x$
- (5) if $x < 5$ goto 2
- (6) $b := x$


| | | | 1ª iteración | | 2ª iteración | | 3ª iteración | |
|-----|-----|------|--------------|-------|--------------|------------|--------------|------------|
| | def | usa | ent[] | sal[] | ent[] | sal[] | ent[] | sal[] |
| (2) | x | y, z | y, z, b | x, b | y, z, b | x, b, z | y, z, b | x, b, z |
| (3) | a | x, b | x, b | a, x | x, b, z | a, x, z, b | x, b, z | a, x, z, b |
| (4) | y | a, x | a, x | x | a, x, z, b | x, y, z, b | a, x, z, b | x, y, z, b |
| (5) | | x | x | x | x, y, z, b | x, y, z, b | x, y, z, b | x, y, z, b |

Transformaciones

1.- Eliminación de subexpresiones comunes.

| | | |
|------------------|--|--------------|
| (1) $a := b + c$ | | $a := b + c$ |
| (2) $b := a - d$ | | $b := a - d$ |
| (3) $c := b + c$ |  | $c := b + c$ |
| (4) $d := a - d$ | | $d := b$ |

2.- Propagación de copias

| | | |
|-------------------|--|-------------------|
| (1) $x := t3$ | | (1) $x := t3$ |
| (2) $t9 := t5$ | | (2) $t9 := t5$ |
| (3) $a[t2] := t9$ |  | (3) $a[t2] := t5$ |
| (4) $a[t4] := x$ | | (4) $a[t4] := t3$ |

3. Eliminación de código inactivo

$B = (\{ f := a + a; g := f * c; f := a + b; g := a * b \}, \{a, b, c\}, \{f, g\})$

$f := a + a$
 $g := f * c$
 $f := a + b$
 $g := a * b$



$f := a + a$
 $f := a + b$
 $g := a * b$



$f := a + b$
 $g := a * b$

$B' = (\{ f := a + b; g := a * b \}, \{a, b\}, \{f, g\})$

4. Renombrar variables temporales

5. Intercambiar dos instrucciones adyacentes independientes.

$t_1 := b + c$

$t_2 := x + y$

3.3. Grafos dirigidos acíclicos (GDA)

Algoritmo construcción de un GDA

Entrada: Un bloque básico $\beta=(I,E,S)$

Salida: GDA

Usa: Un nodo tendrá

- Una etiqueta: *etiqueta*

- Un hijo *derecho* y un hijo *izquierdo*

- Una *lista de variables* que toman el valor representado por el nodo.

CreaNodo(v, i, d): Crea un nodo etiquetado con v, hijo izquierda i, e hijo derecha d

Ult_nodo(v): Devuelve el último nodo que contiene en su lista_var (o su etiqueta si no está en ninguna lista) a la variable v */

Paratoda variable $x \in \text{usa}[\beta] \cup \text{def}[\beta]$ hacer

Ult_nodo(x) := NULL

Paratoda instrucción $S_i (x := y \text{ op } z) \in I$ hacer {

si ult_nodo(y) = NULL entonces CreaNodo(y, NULL, NULL)

si ult_nodo(z) = NULL entonces CreaNodo(z, NULL, NULL)

si $\neg \exists \text{ nodo} \mid \text{Etiqueta}(\text{nodo}) = \text{op} \wedge \text{HijoIzq} = \text{Ult_nodo}(y) \wedge \text{HijoDer}(\text{nodo}) = \text{Ult_nodo}(z)$

entonces nodo := CreaNodo(op, Ult_nodo(y), Ult_nodo(z))

si Ult_nodo(x) \neq NULL entonces {

Quitar x de la lista de variables de Ult_nodo(x)

Añadir x a la lista de variables del nodo, que pasa a ser el Ult_nodo de x

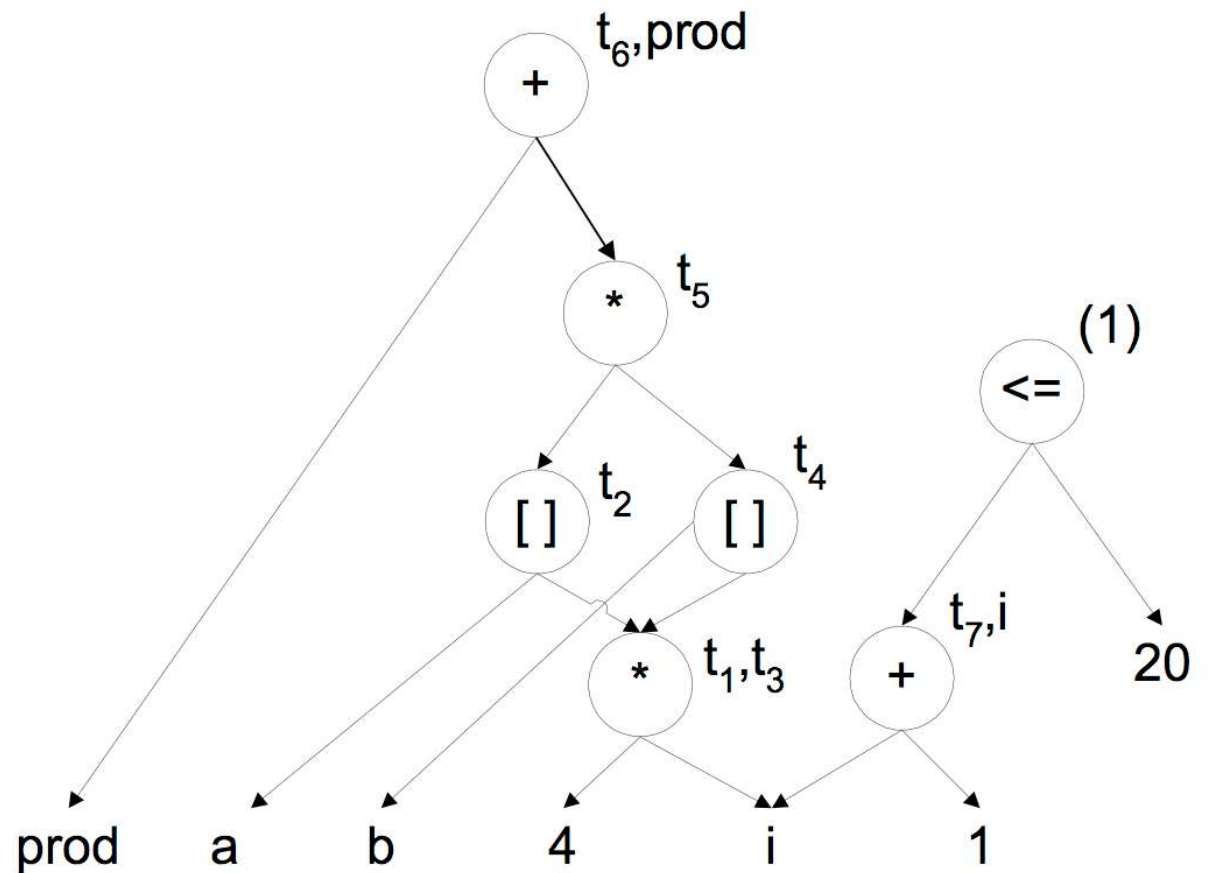
}

}

Ejemplo 4

BB={P, {a,b,i,prod,},{a,b,i,prod}}

- (1) $t_1 := 4 * i$
- (2) $t_2 := a[t_1]$
- (3) $t_3 := 4 * i$
- (4) $t_4 := b[t_3]$
- (5) $t_5 := t_2 * t_4$
- (6) $t_6 := \text{prod} + t_5$
- (7) $\text{prod} := t_6$
- (8) $t_7 := i + 1$
- (9) $i := t_7$
- (10) if $i \leq 20$ goto (1)





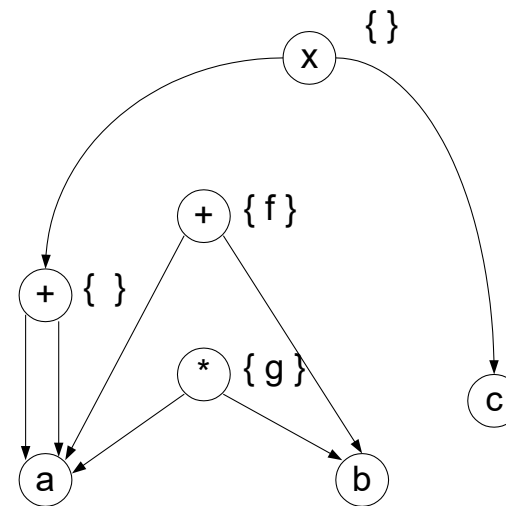
Reglas reconstrucción código

1. Eliminación de código inútil

Nodo sin ascendientes ni variables activas asociadas

BB = (P, {a, b, c}, {f, g})

f := a + a
g := f * c
f := a + b
g := a * b

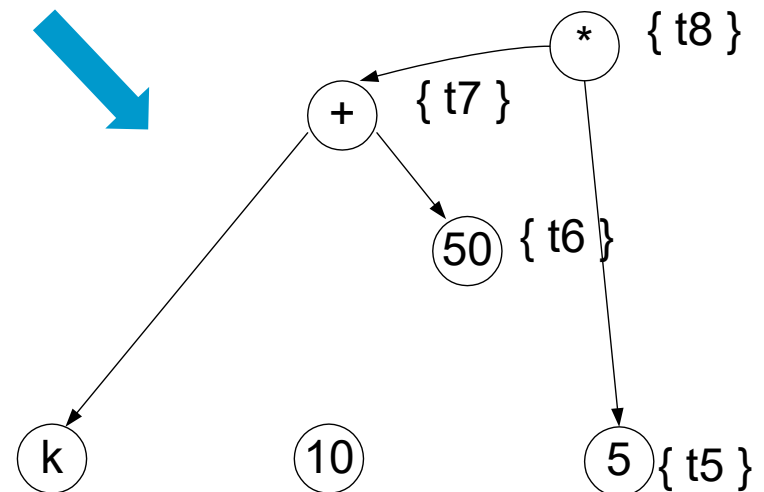
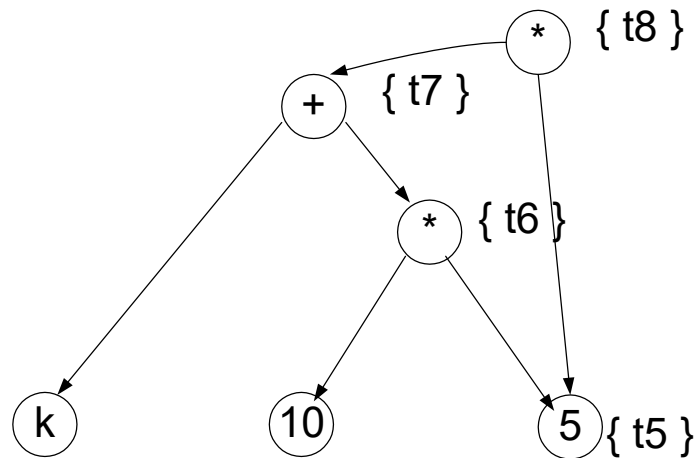


f := a + b
g := a * b

2. Plegado de constantes

BB = (P, {}, {k})

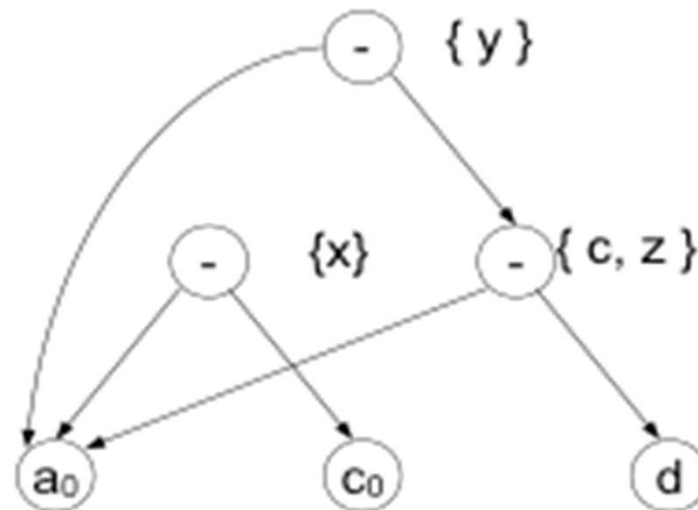
t5 := 5
t6 := 10 * t5
t7 := k + t6
t8 := t7 * t5



3. Usar orden topológico

BB = (P, {a, c, d}, {x, y, z})

$x := a - c$
 $c := a - d$
 $y := a - c$
 $z := a - d$



Órdenes topológicos **correctos**

| | |
|--------------|--------------|
| $x := a - c$ | $z := a - d$ |
| $z := a - d$ | $x := a - c$ |
| $y := a - z$ | $y := a - z$ |

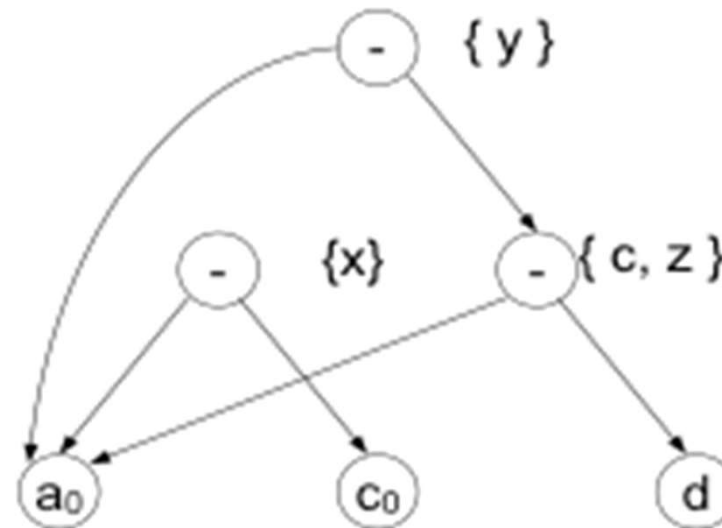
Incorrectos

| | |
|--------------|--------------|
| $y := a - c$ | $c := a - d$ |
| $x := a - c$ | $z := c$ |
| $z := a - d$ | $x := a - c$ |
| | $y := a - z$ |

4. Variable con valores iniciales

No se genera código para definir una variable X de la lista de vbles. de un nodo hasta generar código para todos los usos del valor inicial de X en el bloque (X_0)

$x := a - c$
 $c := a - d$
 $y := a - c$
 $z := a - d$



Secuencia incorrecta:

$c := a - d$
 $x := a - c$
 $y := a - c$
 $z := c$

5. Elección de variable para nodo

Para contener el valor de la expresión de un nodo se elegirá como variable, aquella que esté **activa a la salida** del bloque.

Si **no hubiera** ninguna se elegirá **cualquiera**

Si a la salida hay **más de una activa** se generarán instrucciones de **copia** para cada una

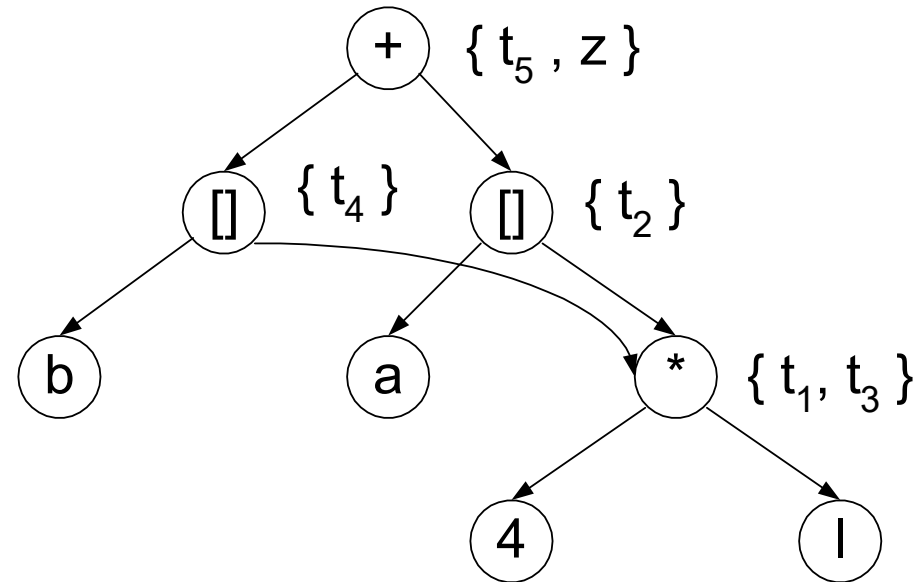
En las hojas:

- Si la etiqueta es una variable y no se modifica dentro del bloque, se considerara está junto a su lista de variables y se aplicará el mismo criterio.
- Si la etiqueta es una constante, y ninguna variable en el campo `lista_vbles` está activa a la salida del bloque, no se generara ninguna instrucción para evaluar esta hoja.

Ejemplo 5

$BB=(P, \{a, i, b\}, \{z\})$

```
t1 := 4 * i  
t2 := a[t1]  
t3 := 4 * i  
t4 := b[t3]  
t5 := t2 + t4  
z := t5
```

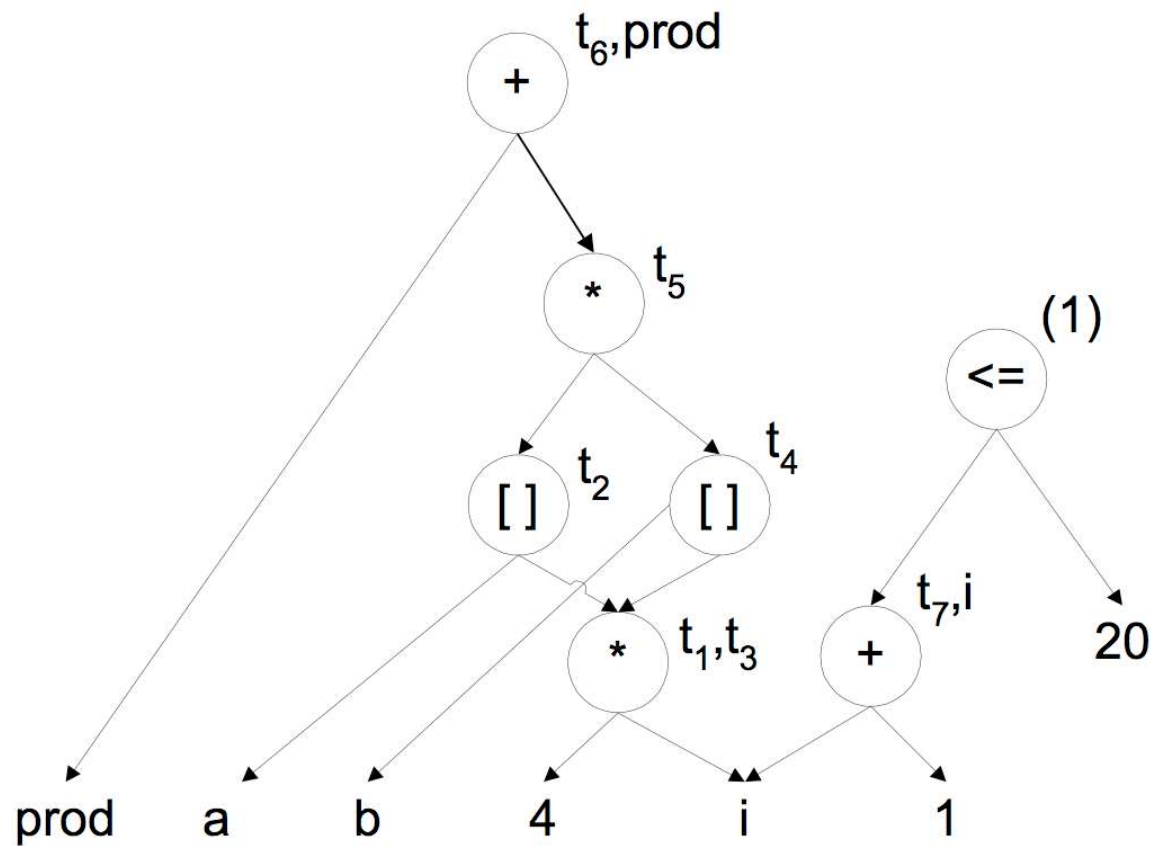


Código reconstruido:

```
t1 := 4 * i  
t2 := a[t1]  
t4 := b[t1]  
z := t2 + t4
```

Ejemplo 4 (cont)

$B = \{P, \{a, b, i, \text{prod}, \}, \{a, b, i, \text{prod}\}\}$



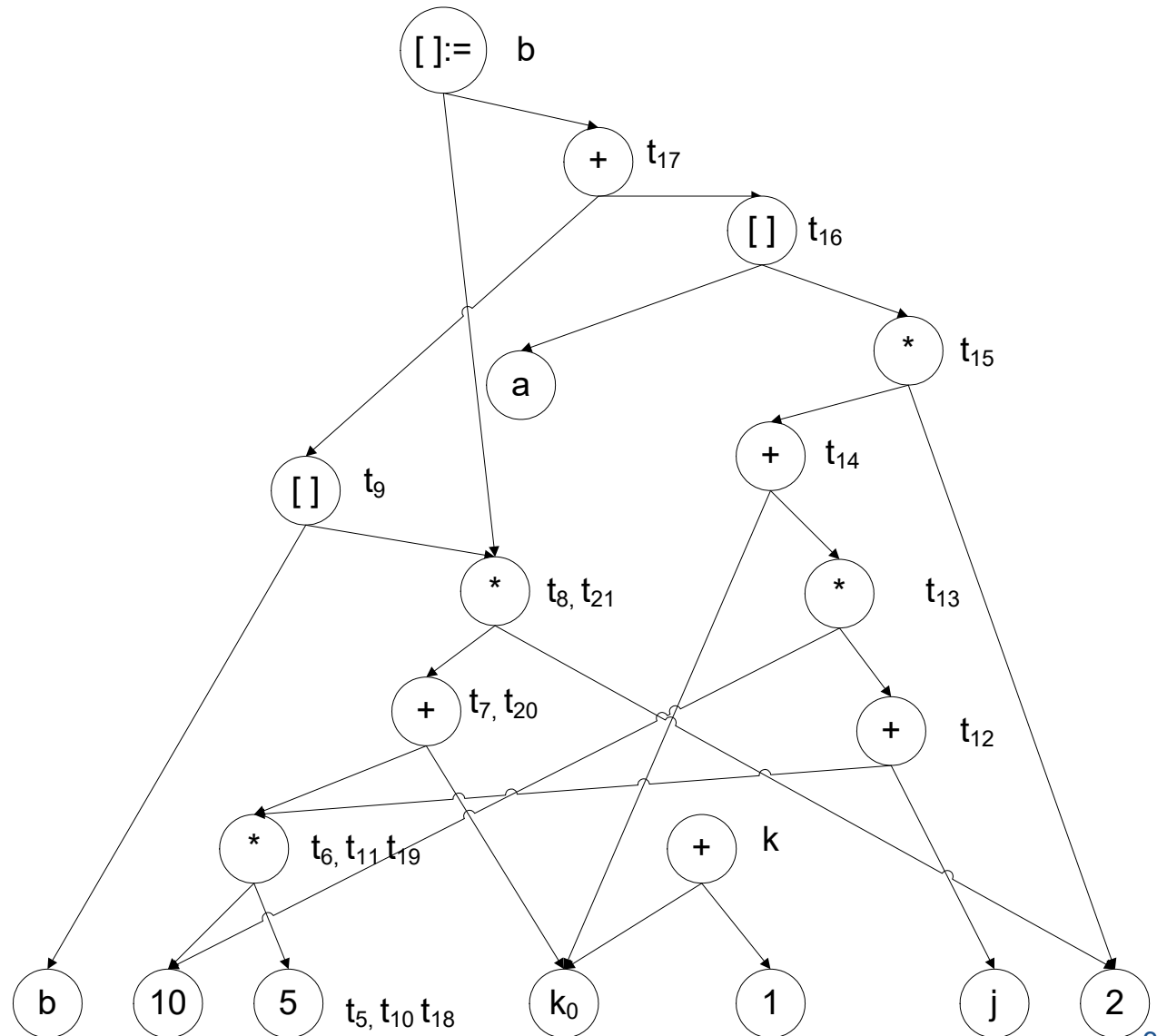
(1) $t_3 := 4 * i$
(2) $t_4 := b[t_3]$
(3) $t_2 := a[t_3]$
(4) $t_5 := t_2 * t_4$
(5) $\text{prod} := \text{prod} + t_5$
(6) $i := i + 1$
(7) if $i \leq 20$ goto (1)

B5 del ejemplo 2

```

t5  := 5
t6  := 10 * t5
t7  := t6 + k
t8  := t7 * 2
t9  := b[t8]
t10 := 5
t11 := 10 * t10
t12 := t11 + j
t13 := t12 * 10
t14 := t13 + k
t15 := t14 * 2
t16 := a[t15]
t17 := t9 + t16
t18 := 5
t19 := 10 * t18
t20 := t19 + k
t21 := t20 * 2
b[t21] := t17
k    := k + 1
goto B4

```

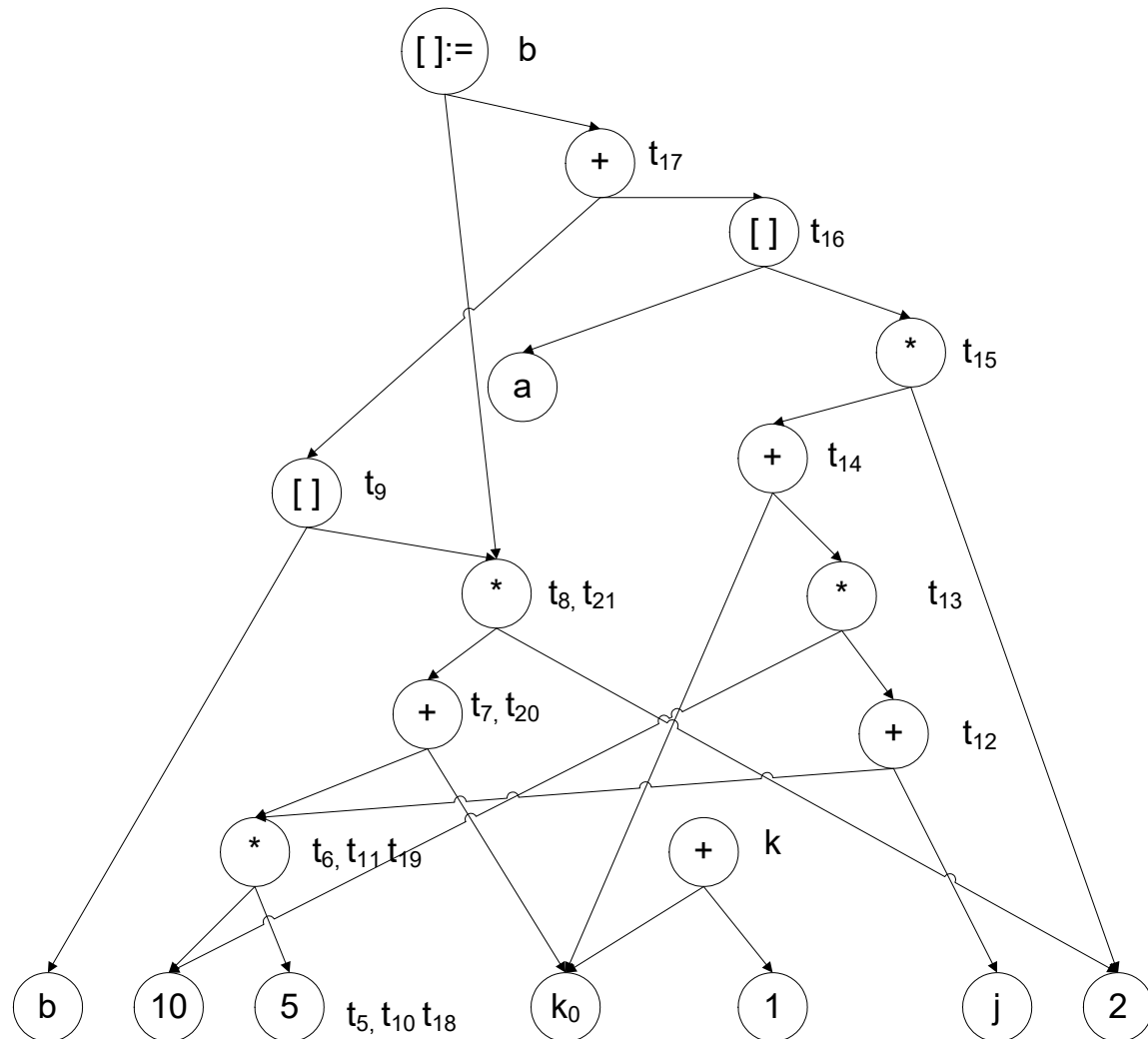


$$BB = \{ P, \{k, j, a, b, t_2, t_4\}, \{k, j, a, b, t_2, t_4\} \}$$

```

t7 := 50 + k
t8 := t7 * 2
t9 := b[t8]
t12 := 50 + j
t13 := t12 * 10
t14 := t13 + k
t15 := t14 * 2
t16 := a[t15]
t17 := t9 + t16
b[t8] := t17
k := k + 1
goto B4

```

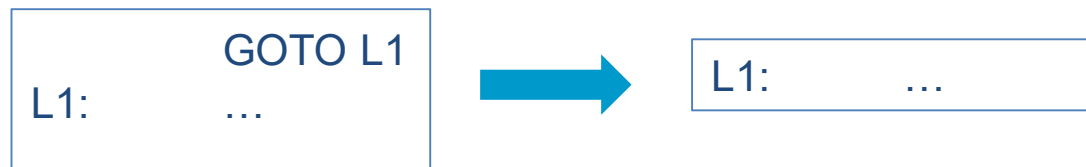
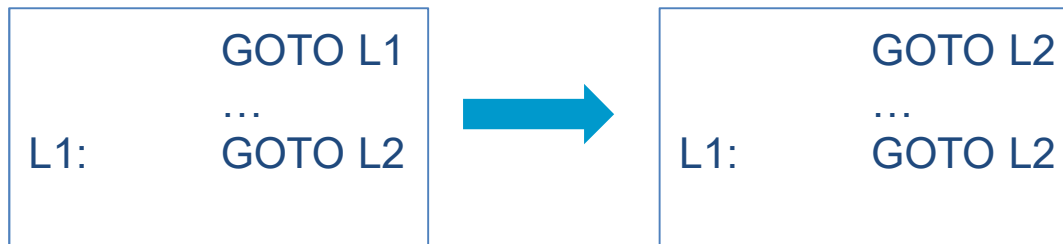




4. Optimizaciones globales

4.1. Técnicas básicas

1. Optimizaciones de saltos



2. Desplegado de bucles

3. Técnica de la mirilla

4.2. Detección de bucles

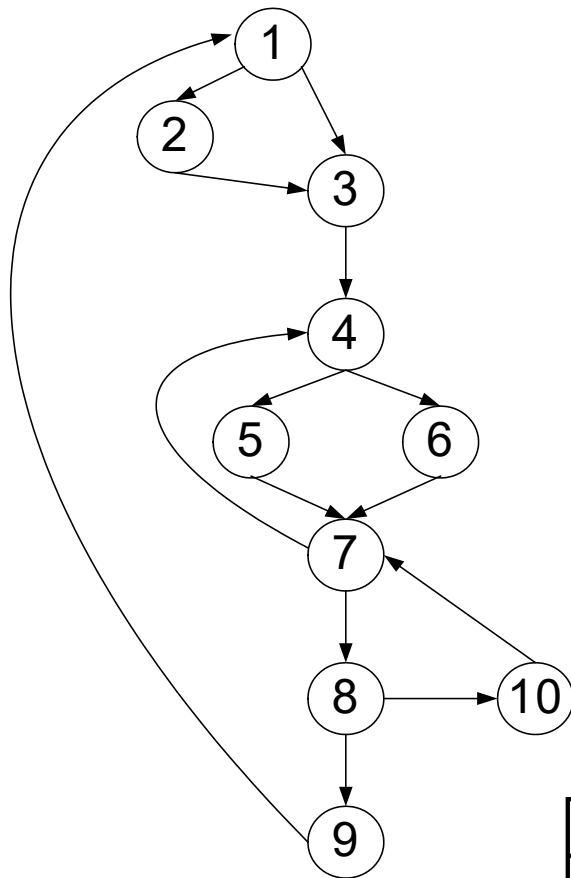
Bucle natural

- Un nodo **d** de un grafo de flujo **domina** a otro nodo **n** ($d \text{ dom } n$) si todo camino desde el nodo inicial del grafo de flujo a **n**, pasa por **d**. La relación de dominación se representa en un **árbol de dominación**: Un nodo domina a sus hijos.
- **Arista de retroceso**: Aquella en la que el nodo de la cabeza domina al nodo de la cola.

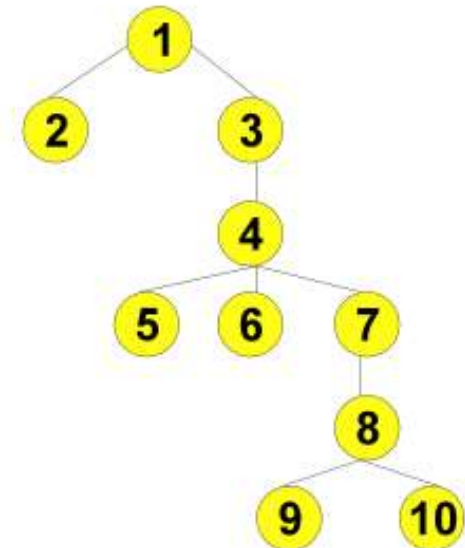
Bucle natural:

- Tiene un solo punto de entrada (*encabezamiento*)
- Tiene una arista (*de retroceso*) que permite iterar el bucle
- Dada una arista de retroceso $n \rightarrow d$, llamamos **bucle natural** asociado a la arista de retroceso, a **d** (encabezamiento del bucle) más el conjunto de nodos que pueden alcanzar **n** sin pasar a través de **d**.

Grafo de flujo



Árbol de dominación



| Arista de retroceso | Nodos del bucle natural |
|---------------------|------------------------------|
| $7 \rightarrow 4$ | $\{ 4,5,6,7,8,10 \}$ |
| $10 \rightarrow 7$ | $\{ 7,8,10 \}$ |
| $9 \rightarrow 1$ | $\{ 1,2,3,4,5,6,7,8,9,10 \}$ |

4.3. Extracción código invariante

Algoritmo marcado código invariante

Repetir

Marcar instrucción si sus operandos:

- Son constantes, O
- No se definen dentro del bucle, O
- Tienen una sola definición dentro del bucle y está marcada

Hasta no se marque ninguna nueva

Ejemplo 6

```
(1) a:= 5 + N  
(2) i := i +1  
(3) b := a * 4  
(4) arr[i] := b  
(5) if a < N goto (1)  
(6) x:= t
```

Marcamos (1) y luego (3)

Ejemplo 7

```
(1) a:= 5  
(2) if b > 2 goto (4)  
(3) a := 4  
(4) b := 2 * b  
(5) if b < N goto (2)  
(6) x := a
```

(3) No puede extraerse

Condiciones extracción código invariante

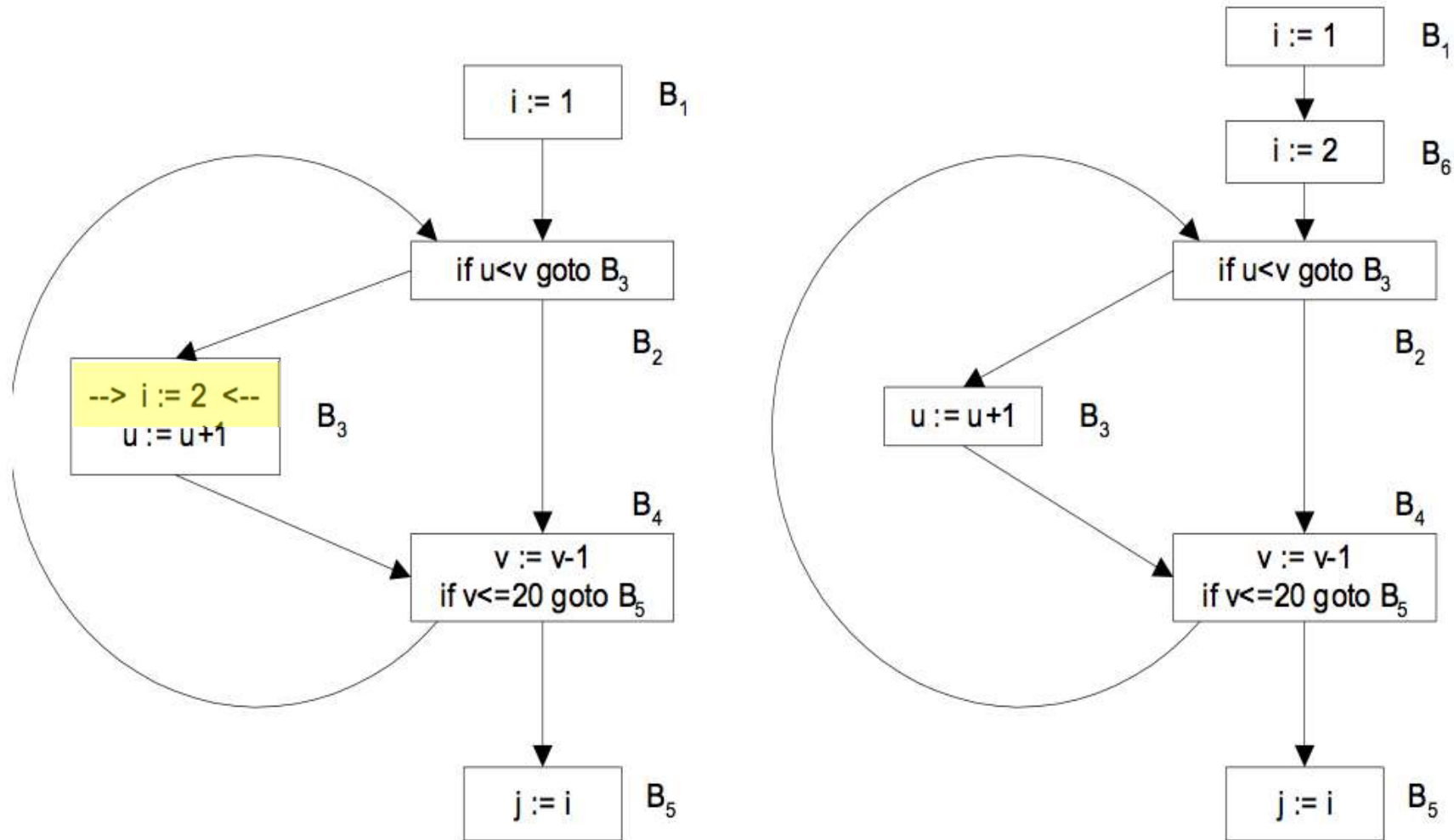
Extraer, en el orden de marcado las instrucciones¹, si cumplen:

1. x está en un bloque que **domina todas las salidas** del bucle, (o x no está activa a la salida del bucle).
2. x **no** se define en **otra parte** del bucle.
3. Todos los **usos de x** en el bucle solo pueden ser **alcanzados** por esa definición de x.
4. Los valores de los operandos son constantes o se definen ya en el pre-encabezamiento.

Se extrae al **preencabezamiento**

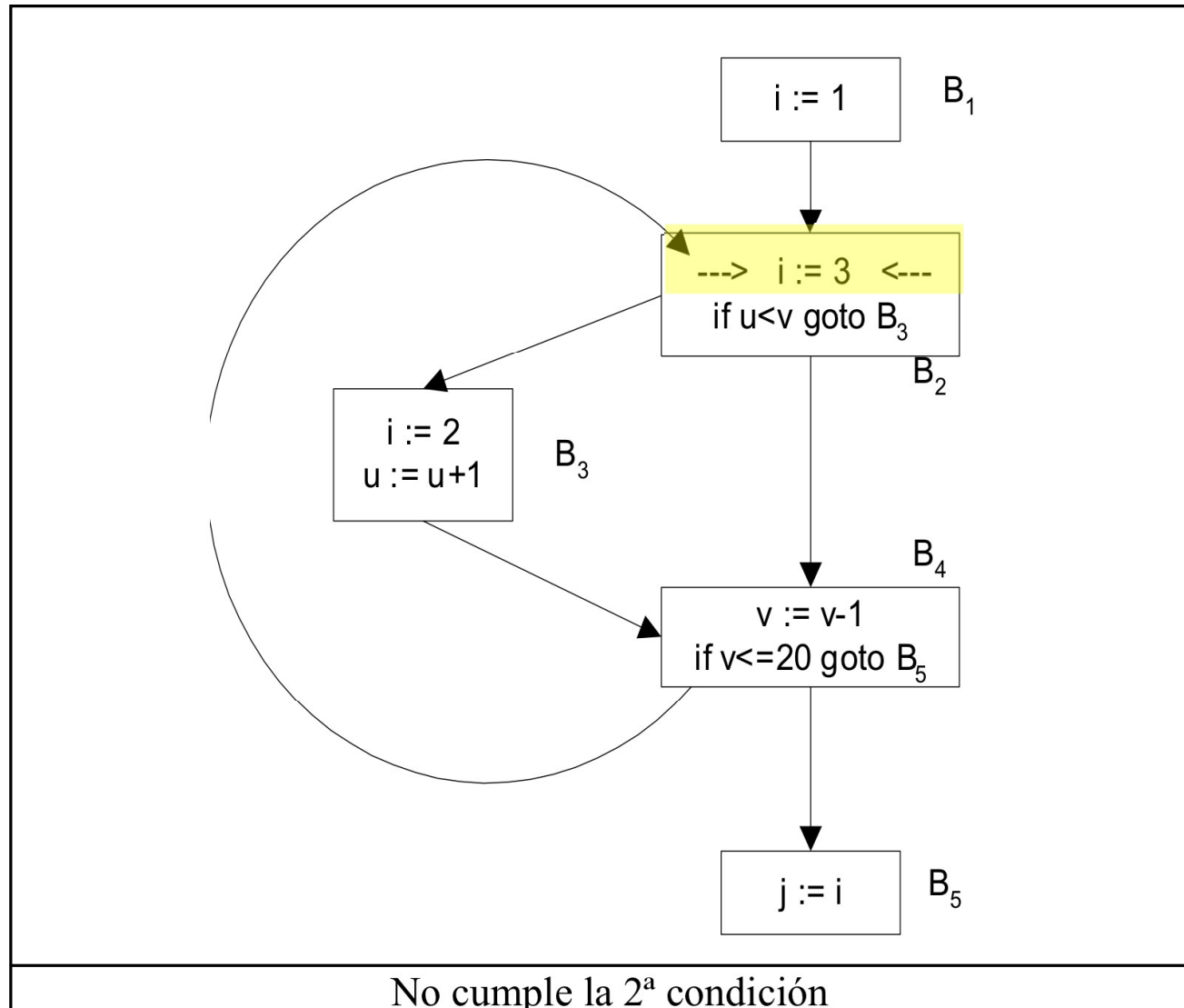
¹Suponemos que la variable definida en la instrucción es x

Ejemplo de extracción ilegal: B_3 no domina a todas las salidas del bucle

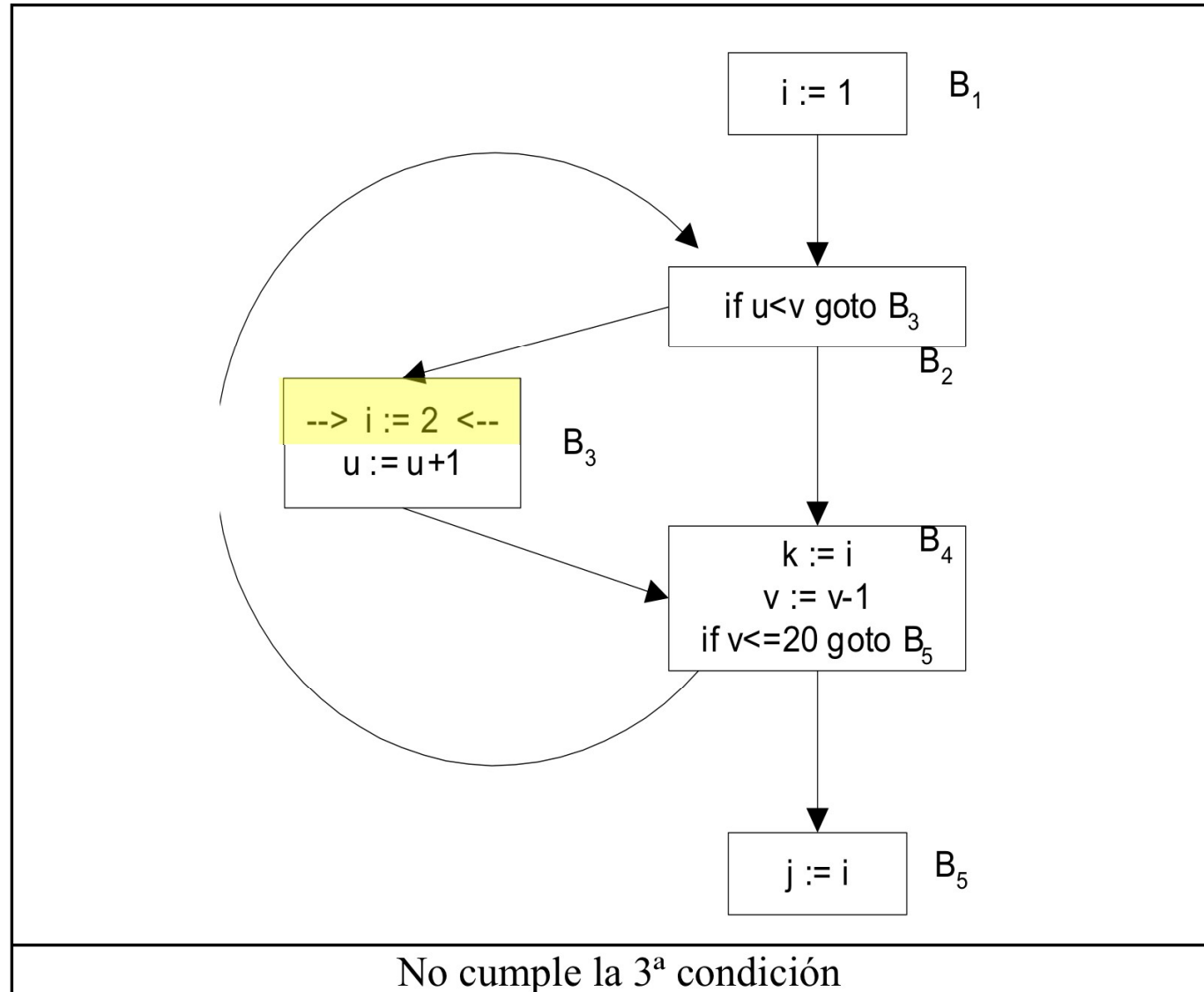


No cumple la 1ª condición

Ejemplo de extracción ilegal: i tiene otra definición en el bucle



Ejemplo de extracción ilegal: Uso de i alcanzado por otra definición



Ejemplo 6 (sol.)

```
(1) a := 5 + N  
(2) i := i + 1  
(3) b := a * 4  
(4) arr[i] := b  
(5) if a < N goto (1)  
(6) x := t
```

Quedaría

```
(0) a := 5 + N  
(1) b := a * 4  
(2) i := i + 1  
(4) arr[i] := b  
(5) if a < N goto (2)  
(6) x := t
```

4.4. Reducción de intensidad y eliminación de variables de inducción

Variable inducción básica (i)

Variable cuya única definición dentro del bucle es de la forma

$$i := \pm c$$

con c constante o invariante del bucle.

Familia de i

Conjunto de variables (j) cuya relación con i en el bucle es de la forma

$$j = i * c + d$$

c y d con invariantes del bucle

se le asocia la terna $j(i, c, d)$

Detección familias vbles. inducción

Algoritmo Detección de variables de inducción

Entrada Bucle con información sobre el alcance y cálculos invariantes

Salida Familias de variables de inducción

Método Encontrar todas las variables **básicas** de inducción (i). Asociar a cada una el triple (i, 1, 0).

Buscar las variables k con una sola definición dentro del bucle de la forma:

$k := j*b,$ $k := b*j,$ $k := j/b,$ $k := j \pm b,$ $k := b \pm j$

donde b es una cte. y j una vble. de inducción

Si j es variable de inducción básica

entonces k está en la familia de j

sino /* j pertenece a familia(i) */

si $(\neg \exists$ definición de i entre la definición de j y la de $k) \wedge$
 $(\neg \exists$ ninguna definición de j fuera del bucle alcanza la definición de $k)$

entonces k está en la familia de i .

Asociar a cada una el triple correspondiente.

Ejemplo 8

```
(0)    i := i + 3
(1)    t1 := 2 * i
(2)    j := t1 + 5
(3)    t2 := 3 * j
(4)    k := t2 + 6
(5)    m := a[j]
(6)    a[k] := m
(7)    if i < N goto (0)
```

Variables de inducción

```
i    (i, 1, 0)
t1   (i, 2, 0)
j    (i, 2, 5)
t2   (i, 6, 15)
k    (i, 6, 21)
```

Reducción intensidad sobre vbles. inducción

Algoritmo Reducción de intensidad aplicada a variables de inducción.

Método

Para toda variable de inducción básica i hacer

Para todo triple (i, c, d) asociado a las variables de inducción j_1, j_2, \dots, j_n , de la Familia(i) hacer

1. Crear una variable temporal s
2. Sustituir las asignaciones a j_i por $j_i := s \quad \forall i \in [1..n]$
3. Para toda asignación $i := i + n$ en el bucle hacer
 añadir a continuación $s := s + c * n$ /* $c*n$ es cte.*/
 poner s en Familia(i) /* triple (i,c,d) */
4. Poner la inicialización de s al final del preencabezamiento:
 $s := c * i$ /* solo $s:=i$ si c es 1 */
 $s := s + d$ /* omitir si d es 0 */

Ejemplo 8 (cont)

```
(0)    i := i + 3
(1)    t1 := 2 * i
(2)    j := t1 + 5
(3)    t2 := 3 * j
(4)    k := t2 + 6
(5)    m := a[j]
(6)    a[k] := m
(7)    if i < N goto (0)
```

Variables de inducción

```
i   (i, 1, 0)
t1  (i, 2, 0)
j   (i, 2, 5)
t2  (i, 6, 15)
k   (i, 6, 21)
```

```
st1 := 2 * i
sj  := 2 * i
sj  := sj + 5
st2 := 6 * i
st2 := st2 + 15
sk  := 6 * i
sk  := sk + 21
```



```
i := i + 3
sk := sk + 18
st2 := st2 + 18
sj := sj + 6
st1 := st1 + 6
t1 := st1
j := sj
t2 := st2
k := sk
m := a[j]
a[k] := m
if i < N goto (0)
```


Eliminación vbles. inducción

Algoritmo Eliminación de variables de inducción.

Entrada Bucle L con información sobre definiciones de alcance, cálculos invariantes y variables activas.

Salida Bucle revisado

Método

1. Eliminar variables de inducción sin usos en el bucle y fuera de él (sólo su propia definición o instrucción de copia)

2. para toda variable de inducción *i* que **se usa sólo** para calcular vbles. de inducción en su familia y saltos condicionales

hacer Elegir una *j* ∈ Familia(*i*) /* con triple (*i*,*c*,*d*) lo más simple */
Modificar cada comprobación en que aparezca *i* para utilizar *j*

finpara

EJEMPLO: **if i oprel x goto B**
 r := *c***x*
 r := *r*+*d*
 if j oprel r goto B

La vble. de inducción *x* se sustituye por *r* (*x*, *c*, *d*)
 /* *r* := *x* si *c* es 1 */
 /* se omite si *d* es 0 */
 /* *r* temporal */

3. para toda vble de inducción eliminada del bucle

hacer borrar todas sus asignaciones

finpara

4. para toda vble de inducc. j para la que el alg. de reducción de intensidad en v.i. introdujo una proposición $j := s$

hacer

Comprobar que no se define s entre $j := s$ y cualquier uso de j

Sustituir usos de j por usos de s

Borrar $j := s$

finpara

Ejemplo 8 (cont)

```
st1 := 2 * i  
sj := 2 * i  
sj := sj + 5  
st2 := 6 * i  
st2 := st2 + 15  
sk := 6 * i  
sk := sk + 21
```



```
i := i + 3  
sk := sk + 18  
st2 := st2 + 18  
sj := sj + 6  
st1 := st1 + 6  
t1 := st1  
j := sj  
t2 := st2  
k := sk  
m := a[j]  
a[k] := m  
if i < N goto (0)
```



```
sj := 2 * i  
sj := sj + 5  
sk := 6 * i  
sk := sk + 21
```



```
i := i + 3  
sk := sk + 18  
sj := sj + 6  
j := sj  
k := sk  
m := a[j]  
a[k] := m  
if i < N goto (0)
```

Ejemplo 8 (cont)

```
sj := 2 * i  
sj := sj + 5  
sk := 6 * i  
sk := sk + 21
```



```
i := i + 3  
sk := sk + 18  
sj := sj + 6  
j := sj  
k := sk  
m := a[j]  
a[k] := m  
if i < N goto (0)
```



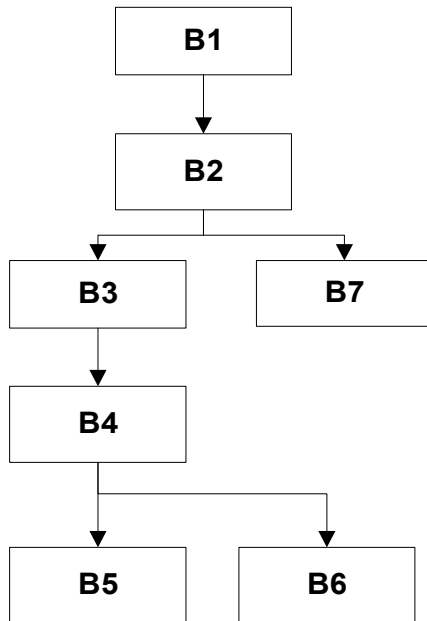
```
sj := 2 * i  
sj := sj + 5  
sk := 6 * i  
sk := sk + 21  
t := 2 * N  
t := t + 5
```



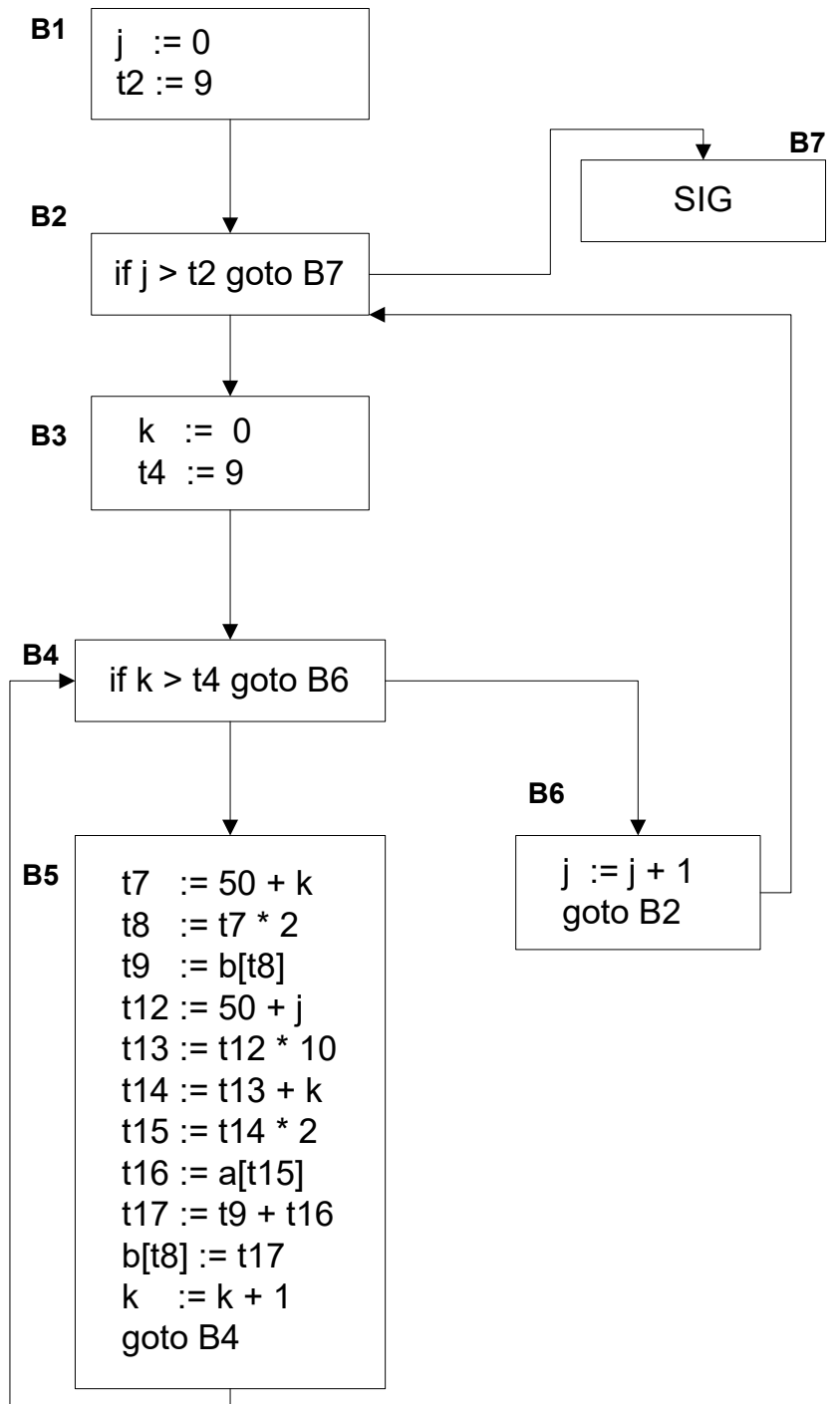
```
sk := sk + 18  
sj := sj + 6  
m := a[sj]  
a[sk] := m  
if sj < t goto (0)
```

Ejemplo 2: Detección bucles

Árbol de dominación



| Ar. Retroceso | Bucle Natural |
|---------------|--------------------|
| B5->B4 | B5, B4 |
| B6 -> B2 | B6, B4, B5, B3, B2 |



Ejemplo 2: Extracción código invariante

Bucle interno (B5-B4):

Instrucciones invariantes

$t_{12} := 50 + j$

$t_{13} := t_{12} * 10$

t_{12} y t_{13} no están activas a la salida del bucle.

Variables de inducción Bucle interno (B5-B4):

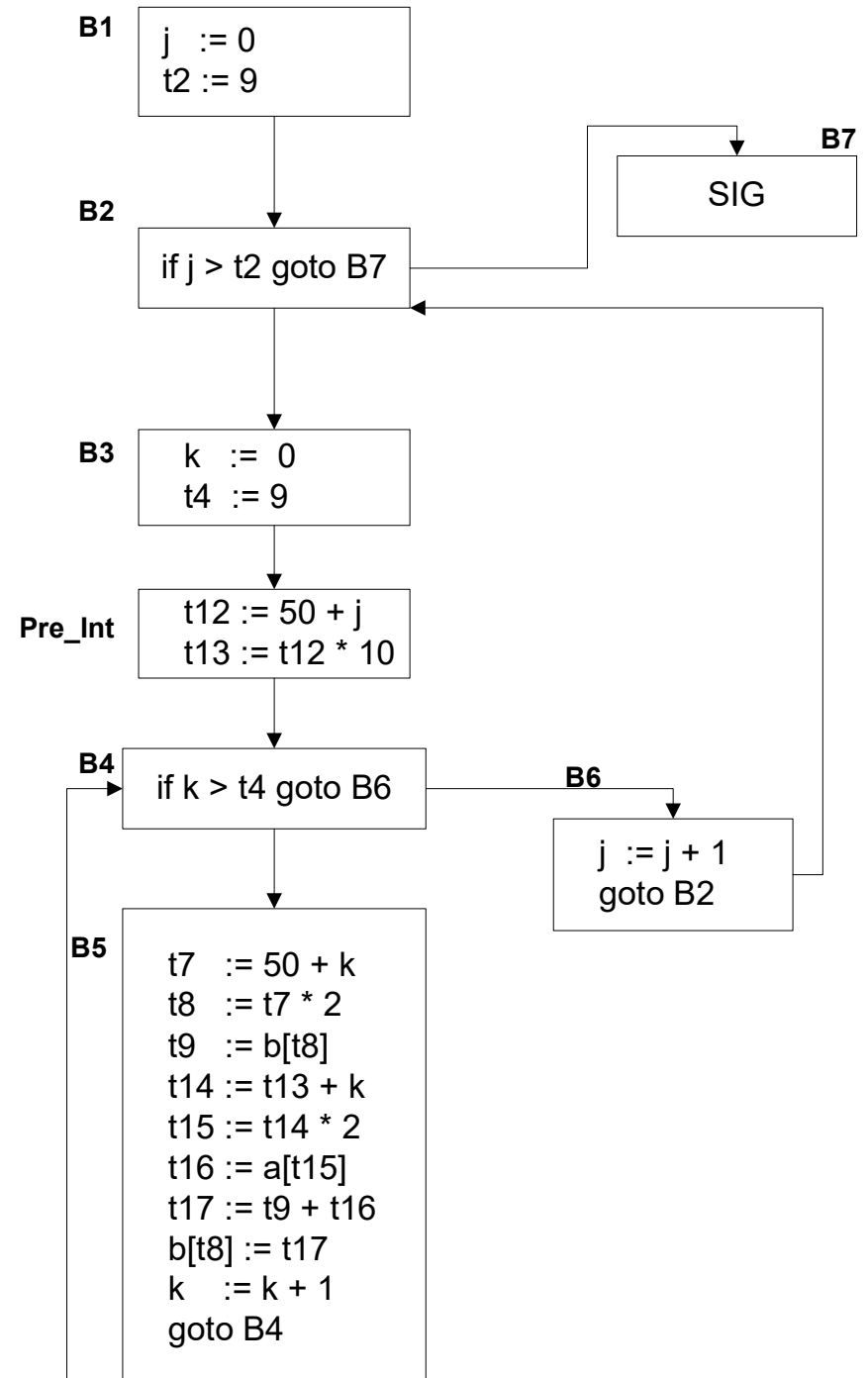
$k(k, 1, 0)$

$t_7(k, 1, 50)$

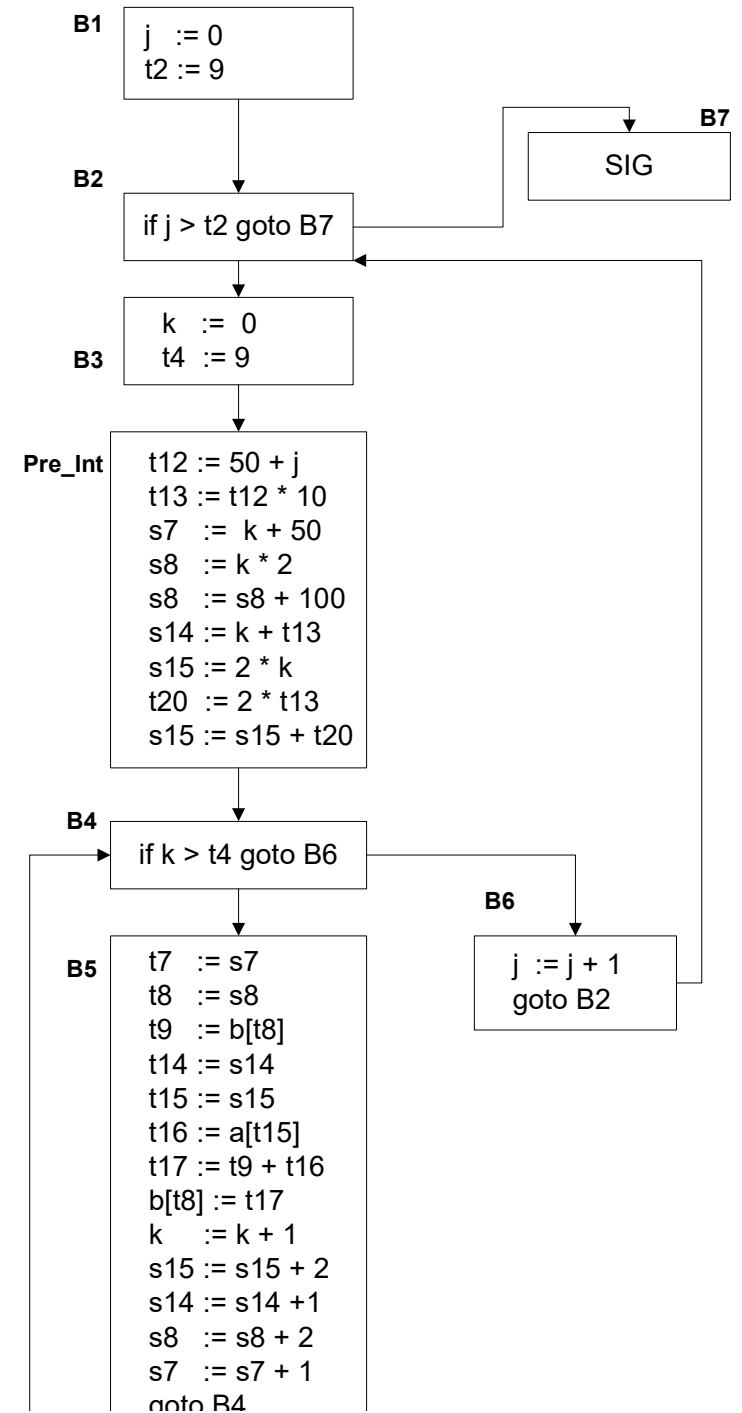
$t_8(k, 2, 100)$

$t_{14}(k, 1, t_{13})$

$t_{15}(k, 2, 2 * t_{13})$



Ejemplo 2. Bucle interno: Reducción intensidad



Ejemplo 2. Bucle interno: Eliminación vbles. inducción

