Análisis de Flujo

CI4722 – Lenguajes de Programación III

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Mejoras sobre los ciclos

- Una vez identificados los ciclos, podemos aplicar mejoras locales.
  - Detectar y mover cómputos invariantes.
  - Eliminar variables de inducción.
  - Reducir la fuerza de cómputo en las variables de inducción.
- Aplicar a cada ciclo – desde los más internos hacia los más externos.

UNIVERSIDAD SIMÓN BOLÍVAR

Detección de Cómputos Invariantes

- *Loop invariants* – valores que no cambian mientras el control se encuentre ejecutando dentro del ciclo.
- Combinar Definiciones Activas con Detección de Ciclos Naturales.

UNIVERSIDAD SIMÓN BOLÍVAR

Detección de Cómputos Invariantes

- *Loop invariants* – valores que no cambian mientras el control se encuentre ejecutando dentro del ciclo.
- Combinar Definiciones Activas con Detección de Ciclos Naturales.
- Invariante Directa
  - `x := y + z` está en el Ciclo Natural.
  - Todas las definiciones activas de `y` y `z` están *fuera* del ciclo.

UNIVERSIDAD SIMÓN BOLÍVAR

Detección de Cómputos Invariantes

- *Loop invariants* – valores que no cambian mientras el control se encuentre ejecutando dentro del ciclo.
- Combinar Definiciones Activas con Detección de Ciclos Naturales.
- Invariante Directa
  - `x := y + z` está en el Ciclo Natural.
  - Todas las definiciones activas de `y` y `z` están *fuera* del ciclo.
- Invariante Indirecta
  - Supongamos que `x` es invariante.
  - `v := x + w` está en el Ciclo Natural.
  - Todas las definiciones activas de `w` están *fuera* del lazo.
  - Concluimos que `v` es Invariante Indirecta.

Sugiere un algoritmo de punto fijo...

UNIVERSIDAD SIMÓN BOLÍVAR

Detección de Cómputos Invariantes

Algoritmo

Sea L el conjunto de bloques de un Ciclo Natural.
Cada instrucción dispone de información de Definiciones Activas.

Detección de Cómputos Invariantes

Algoritmo

Sea L el conjunto de bloques de un Ciclo Natural.
Cada instrucción dispone de información de Definiciones Activas.

1. Marcar como invariantes todas aquellas instrucciones tales que:
   - Sus operandos sean constantes.
   - Sus operandos tengan **todas** sus definiciones activas **fuera** de L.

UNIVERSIDAD SIMÓN BOLÍVAR

Detección de Cómputos Invariantes

Algoritmo

Sea L el conjunto de bloques de un Ciclo Natural.
Cada instrucción dispone de información de Definiciones Activas.

1. Marcar como invariantes todas aquellas instrucciones tales que:
   - Sus operandos sean constantes.
   - Sus operandos tengan **todas** sus definiciones activas **fuera** de L.
2. Marcar como invariantes todas aquellas instrucciones tales que:
   - Sus operandos sean constantes.
   - Sus operandos tengan **todas** sus definiciones activas **fuera** de L.
   - Sus operandos tienen **exactamente una** definición, que está **dentro** de L y en una instrucción invariante.
3. Repetir el paso 2 hasta que dejen de marcarse instrucciones.

UNIVERSIDAD SIMÓN BOLÍVAR

Movimiento del Cómputo Invariante

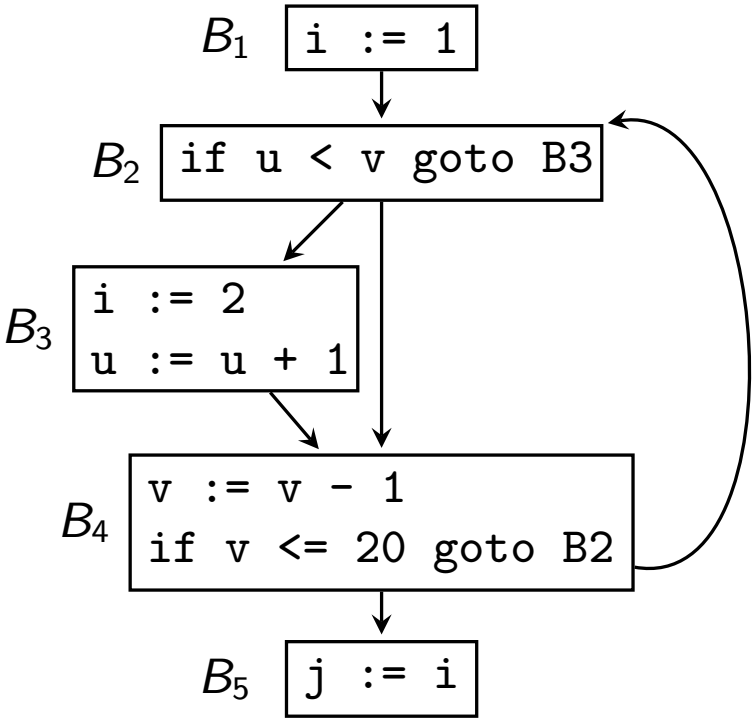
- Un Cómputo Invariante en el ciclo L puede ser movido de manera que se ejecute una sola vez *antes* de entrar al ciclo.
- Agregar un nuevo bloque previo al *header* de L – *pre-header*
  - Tiene como predecesores a todos los predecesores del *header*.
  - Tiene como único sucesor al *header*.
- La instrucción `x := y + z` puede moverse siempre y cuando:
  1. El bloque que la contiene domina a **todas** las salidas del ciclo L.
  2. No hay **ninguna** otra instrucción en L que asigne a `x`.
  3. **Todos** los usos de x en L dependen de esa definición.

Respetar las tres reglas es crucial.

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones para el movimiento

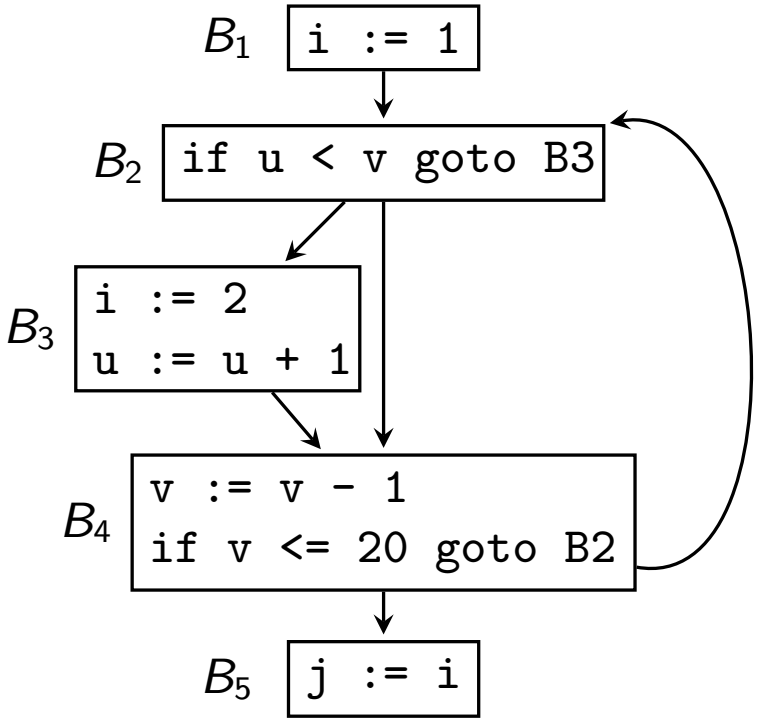
Respetar la Condición de Dominación

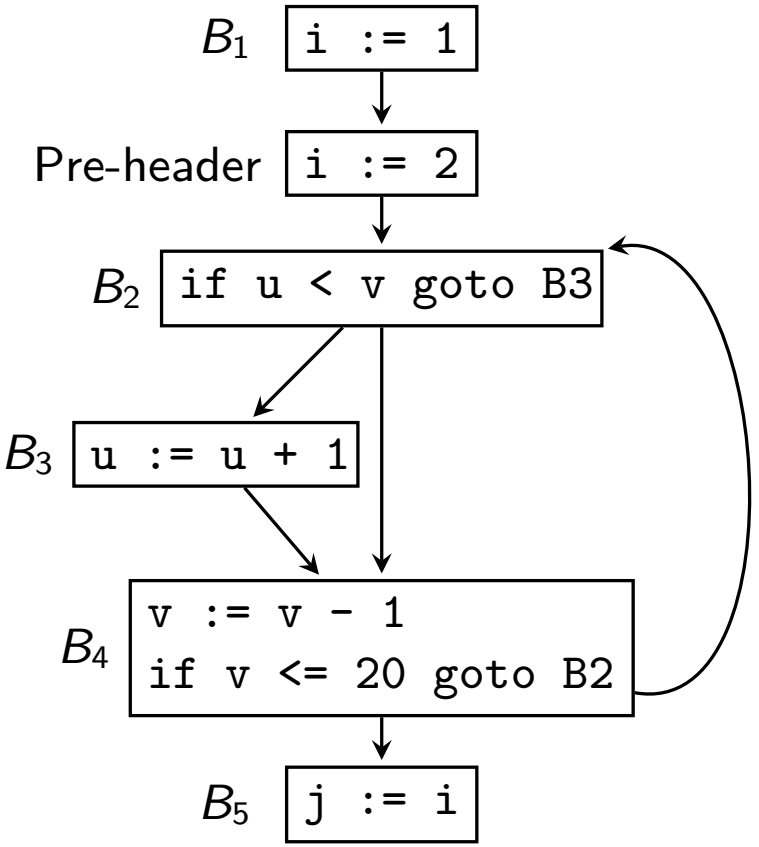


- `i := 2` – invariante en el ciclo B_2, B_3, B_4

Condiciones para el movimiento

Respetar la Condición de Dominación



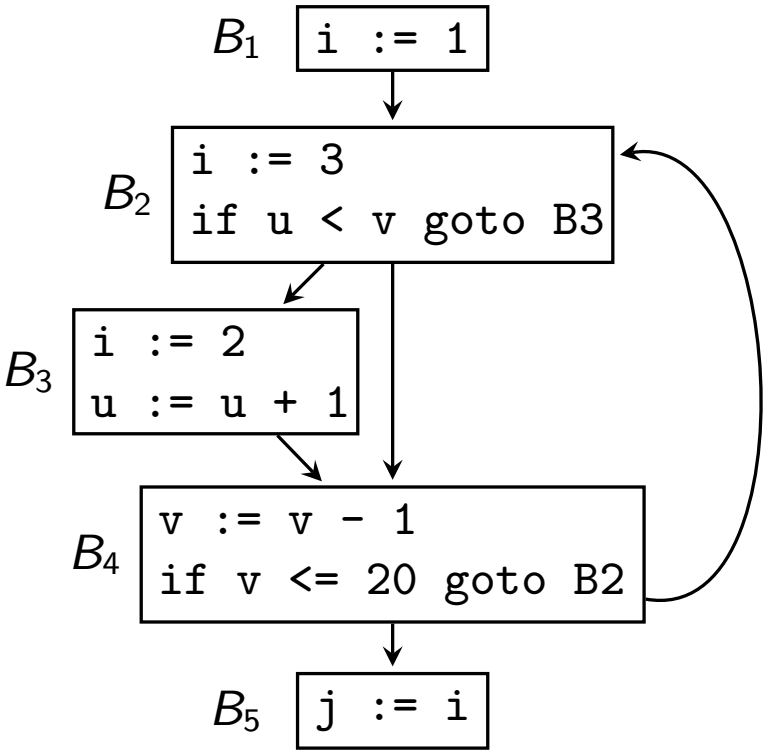


- `i := 2` – invariante en el ciclo B_2, B_3, B_4
- B_3 no domina a B_4 – ¡mover introduce un *bug*!

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones para el movimiento

Respetar la Condición de Asignación Única

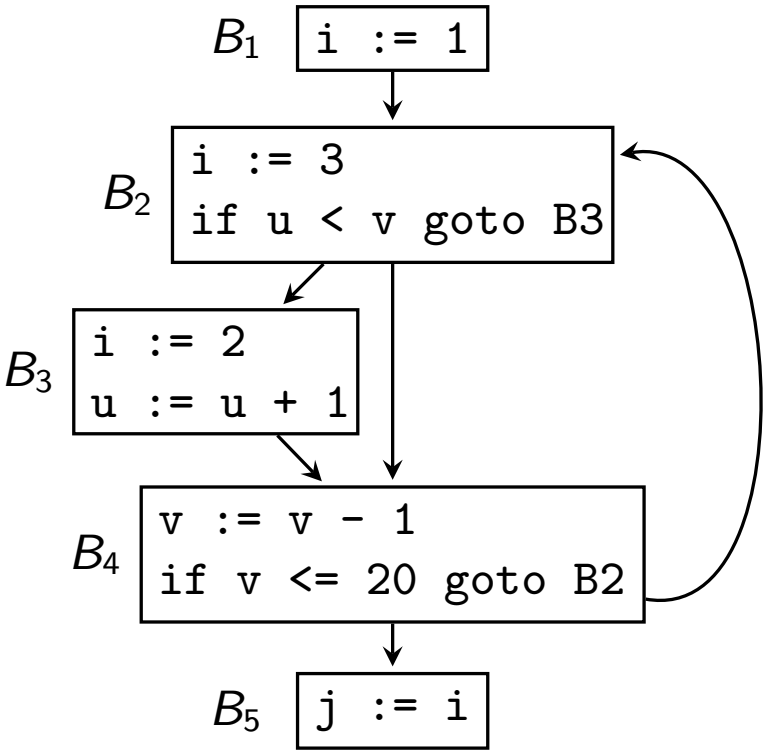


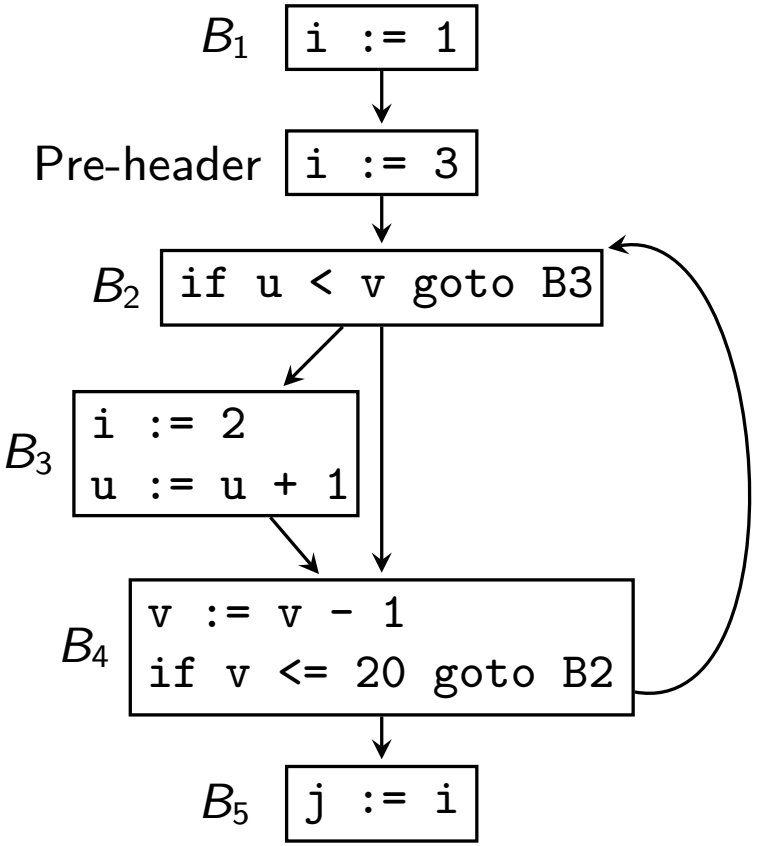
- `i := 3` – invariante en el ciclo B_2, B_3, B_4

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones para el movimiento

Respetar la Condición de Asignación Única



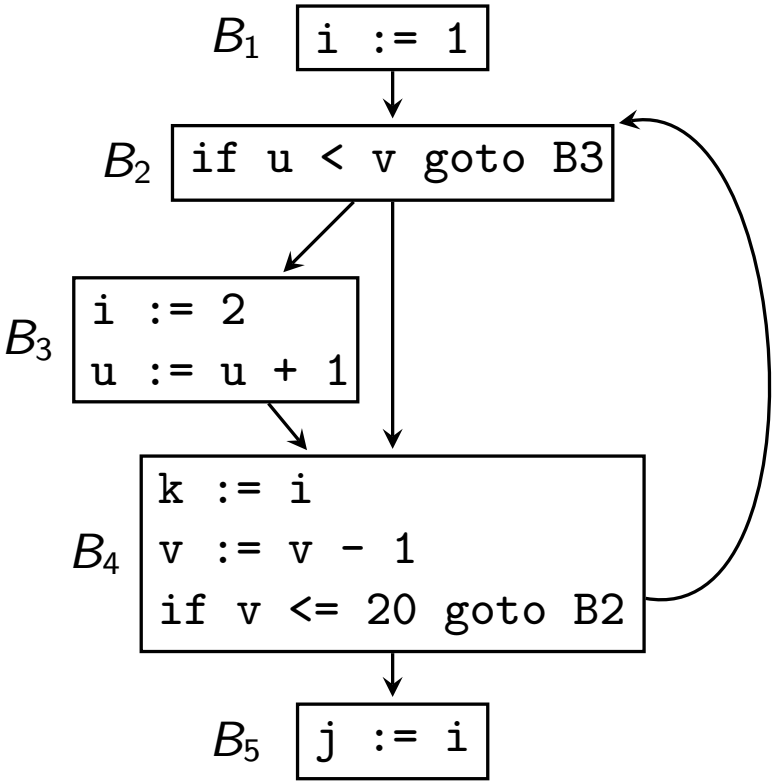


- `i := 3` – invariante en el ciclo B_2, B_3, B_4
- Hay otra asignación – ¡mover introduce un *bug*!

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones para el movimiento

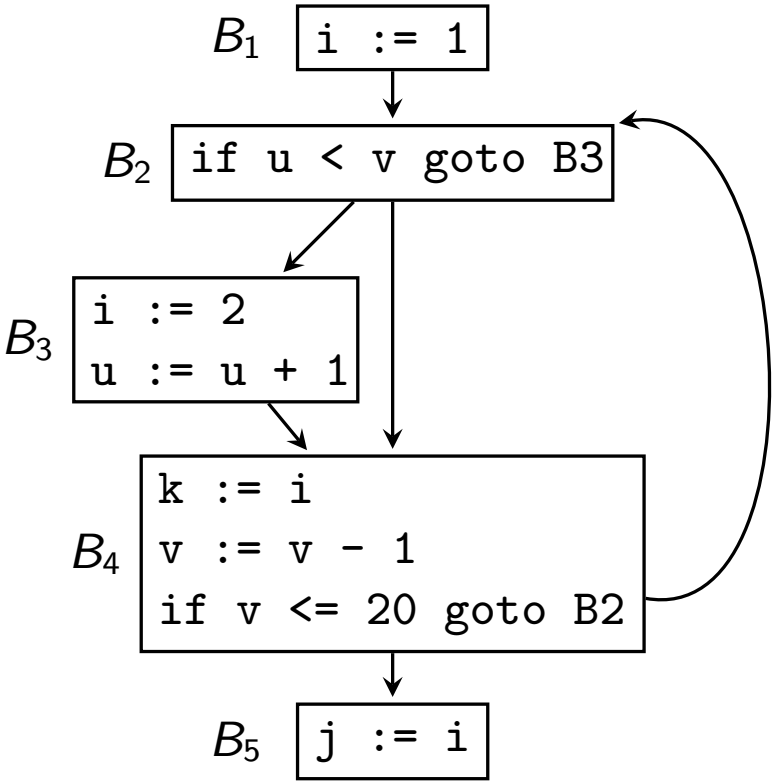
Respetar la Condición de Uso Propio

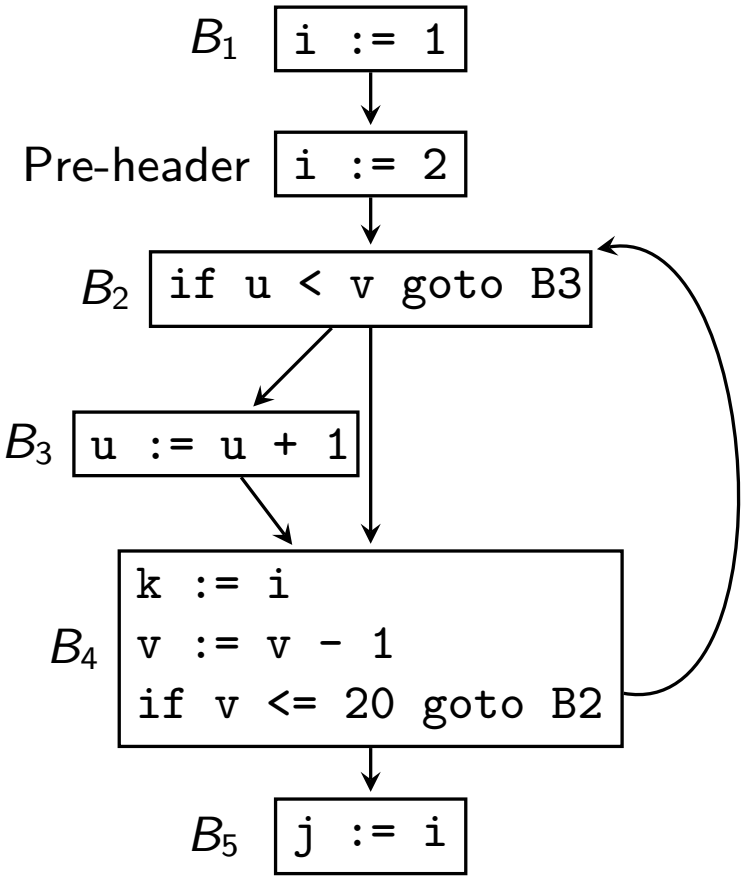


- `i := 2` – invariante en el ciclo B_2, B_3, B_4

Condiciones para el movimiento

Respetar la Condición de Uso Propio





- `i := 2` – invariante en el ciclo B_2, B_3, B_4
- `k := i` usaba a B_1 y B_3 – ¡mover introduce un *bug*!

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Movimiento de Código

Sea L el conjunto de bloques de un Ciclo Natural.
Cada bloque dispone de información de Dominadores.
Cada instrucción dispone de información de Definiciones Activas.

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Movimiento de Código

Sea L el conjunto de bloques de un Ciclo Natural.
Cada bloque dispone de información de Dominadores.
Cada instrucción dispone de información de Definiciones Activas.

1. Encontrar las instrucciones con Cómputos Invariantes.

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Movimiento de Código

Sea L el conjunto de bloques de un Ciclo Natural.
Cada bloque dispone de información de Dominadores.
Cada instrucción dispone de información de Definiciones Activas.

1. Encontrar las instrucciones con Cómputos Invariantes.
2. Para cada instrucción `x := y + z` encontrada, verificar que:
   - Está en un bloque que domina a **todas** las salidas de L,
   - **Ningún** otro bloque de L define a `x`, y
   - Sólo **esa** definición de `x` alcanza a **todos** los usos de `x` en L.

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Movimiento de Código

Sea L el conjunto de bloques de un Ciclo Natural.
Cada bloque dispone de información de Dominadores.
Cada instrucción dispone de información de Definiciones Activas.

1. Encontrar las instrucciones con Cómputos Invariantes.
2. Para cada instrucción `x := y + z` encontrada, verificar que:
   - Está en un bloque que domina a **todas** las salidas de L,
   - **Ningún** otro bloque de L define a `x`, y
   - Sólo **esa** definición de `x` alcanza a **todos** los usos de `x` en L.
3. Si cumple las tres condiciones, moverla al *pre-header* preservando el orden en que fue encontrada durante el paso (1).

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones impiden introducir defectos

- Condiciones de Dominación y Asignación Única.
  - Garantizan que el valor de `x` calculado por la instrucción a mover es *efectivamente* el valor al salir de L.
  - Mover la instrucción al *pre-header* mantiene esa definición como la única para todo L y por ende será el valor al salir.

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones impiden introducir defectos

- Condiciones de Dominación y Asignación Única.
  - Garantizan que el valor de `x` calculado por la instrucción a mover es *efectivamente* el valor al salir de L.
  - Mover la instrucción al *pre-header* mantiene esa definición como la única para todo L y por ende será el valor al salir.
- Condición de Uso Propio
  - Garantiza que cualquier uso de `x` en L usaba y continuará usando la definición hecha por la instrucción.
  - Mover la instrucción al *pre-header* mantiene esa definición como la única para todo L y por ende será común a los usos en L.

UNIVERSIDAD SIMÓN BOLÍVAR

Condiciones impiden introducir defectos

- Condiciones de Dominación y Asignación Única.
  - Garantizan que el valor de `x` calculado por la instrucción a mover es *efectivamente* el valor al salir de L.
  - Mover la instrucción al *pre-header* mantiene esa definición como la única para todo L y por ende será el valor al salir.
- Condición de Uso Propio
  - Garantiza que cualquier uso de `x` en L usaba y continuará usando la definición hecha por la instrucción.
  - Mover la instrucción al *pre-header* mantiene esa definición como la única para todo L y por ende será común a los usos en L.
- La Condición de Dominación garantiza que la instrucción se ejecutaba al menos *una* vez en el ciclo – después de moverla al *pre-header* se ejecutará *exactamente* una vez.

Mismo cómputo, pero mejorado.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Y la información de Flujo de Datos?

Hay que mantenerla vigente para continuar aplicando mejoras

- Usos de la variable x definida por la instrucción que movimos siguen siendo alcanzados pero ahora desde el *pre-header* – sólo hay que ajustar la referencia al *pre-header*.
- Definiciones de operandos usadas por la instrucción que movimos:
  - Estaban fuera de L – siguen alcanzando al *pre-header*.
  - Estaban dentro de L – también fueron movidos al *pre-header* y están antes de la instrucción.
- El *pre-header* es nuevo dominador inmediato del *header*

El mantenimiento de estructuras es económico.

UNIVERSIDAD SIMÓN BOLÍVAR

Eliminación de Variables de Inducción

- Una **Variable de Inducción** de un ciclo L es aquella que cada vez que cambia (incremento o decremento) lo hace en una cantidad *constante*.
- Método general para eliminar **Variables de Inducción Básicas**
  - Básicas – sólo son asignadas de la forma

    ```
    i := i + c
    ```

  - Derivadas – variables definidas una sola vez dentro de L, y como función lineal de otra variable de inducción.

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

Convenciones

- Sea L el conjunto de bloques de un Ciclo Natural.
- Se dispone de la información de Cómputos Invariantes previamente detectados que ya han sido movidos.
- Cada instrucción dispone de información de Definiciones Activas.
- El Algoritmo detectará las variables de inducción `j`, cada una de las cuales tiene asociada una tripleta de la forma

  `j : (i,c,d)`

  tales que
  - `i` es la variable de inducción básica.
  - `c` y `d` son constantes tales que cuando `j` está definida, su valor es $c \times i + d$.

Diremos que `j` pertenece a la *familia* de `i`

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

1. Encontrar definiciones de variables básicas – resultarán tuplas

```
j : (j,1,0)
```

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

1. Encontrar definiciones de variables básicas – resultarán tuplas

```
j : (j,1,0)
```

2. Encontrar variables `k` con *exactamente una* asignación de la forma

```
k := j * b      k := b * j
k := j + b      k := b + j
k := j - b      k := b - j
k := j / b
```

y calcular su tupla.

. . . con mucho cuidado

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

Cálculo de las tuplas

Encontramos una variable `k` que depende de `j`

Algoritmo de Detección de Variables de Inducción

Cálculo de las tuplas

Encontramos una variable `k` que depende de `j`

- Si `j` es básica, `k` está en su familia – tupla depende de la operación

```
k := j * b      k : (j,b,0)
k := j + b      k : (j,1,b)
```

Algoritmo de Detección de Variables de Inducción

Cálculo de las tuplas

Encontramos una variable `k` que depende de `j`

- Si `j` es básica, `k` está en su familia – tupla depende de la operación

```
k := j * b      k : (j,b,0)
k := j + b      k : (j,1,b)
```

- Si `j` no es básica, entonces está en la familia de alguna `i` – es necesario que se cumplan dos condiciones:
  - No puede haber asignaciones a `i` entre las asignaciones de `j` y de `k`.
  - Ninguna definición de `j` *fuera* de L alcanza a `k`.

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

Cálculo de las tuplas

Encontramos una variable `k` que depende de `j`

- Si `j` es básica, `k` está en su familia – tupla depende de la operación

```
k := j * b      k : (j,b,0)
k := j + b      k : (j,1,b)
```

- Si `j` no es básica, entonces está en la familia de alguna `i` – es necesario que se cumplan dos condiciones:
  - No puede haber asignaciones a `i` entre las asignaciones de `j` y de `k`.
  - Ninguna definición de `j` *fuera* de L alcanza a `k`.
- Generamos la tupla a partir de `j : (i,c,d)` según la operación

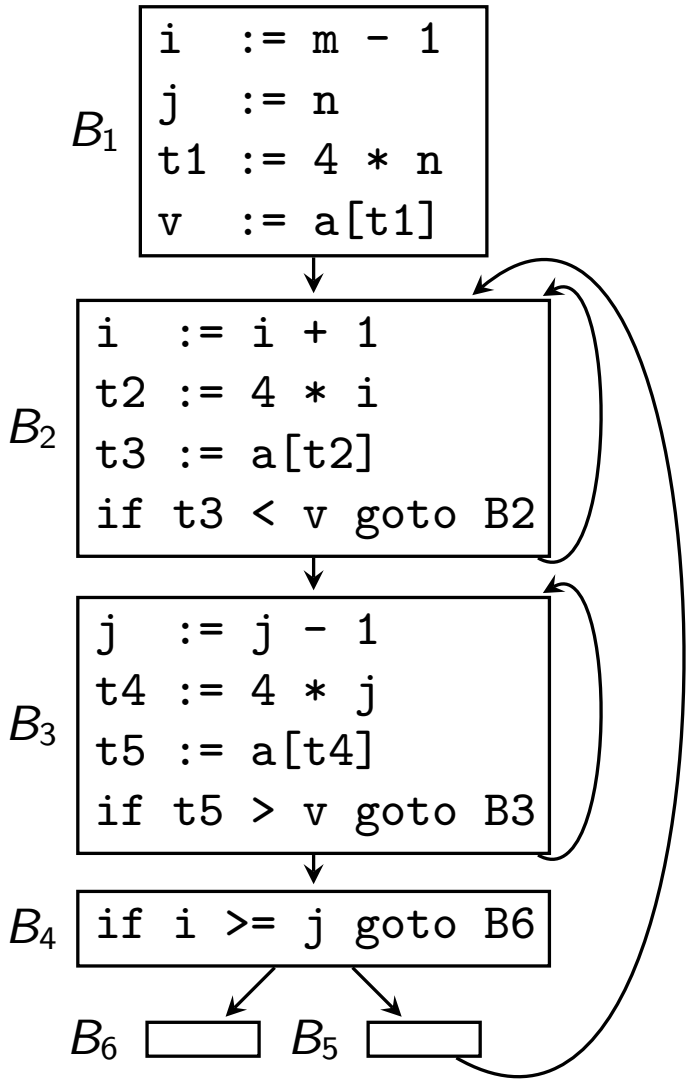
```
k := b * j      k : (i,b*c,b*d)
k := j + b      k : (i,c,b+d)
```

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

Ciclo B_2

- Variable Básica `i`
  - `i := i + 1` – única asignación.
  - Tupla `i : (i,1,0)`
- Variable Indirecta `t2`
  - `t2 := 4 * i` – única asignación.
  - En la familia de `i`
  - Tupla `t2 : (i,4,0)`

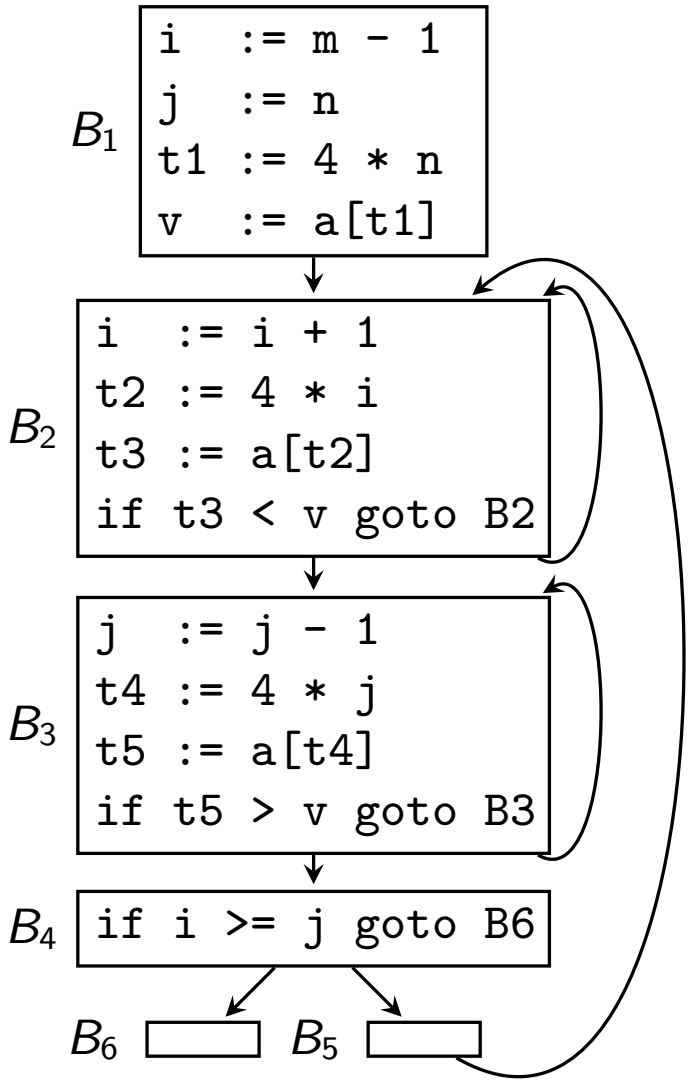


UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Detección de Variables de Inducción

Ciclo B_3

- Variable Básica `j`
  - `j := j - 1` – única asignación.
  - Tupla `j : (j,1,0)`
- Variable Indirecta `t4`
  - `t4 := 4 * j` – única asignación.
  - En la familia de `j`
  - Tupla `t4 : (j,4,0)`



UNIVERSIDAD SIMÓN BOLÍVAR

Reducción de Fuerza

- Identificadas las *familias* de variables de inducción, se modifican las instrucciones para usar sumas o restas en lugar de multiplicaciones.
- Para cada variable de inducción básica `i`,
  repetir para cada `j : (i,c,d)` en la familia

Reducción de Fuerza

- Identificadas las *familias* de variables de inducción, se modifican las instrucciones para usar sumas o restas en lugar de multiplicaciones.
- Para cada variable de inducción básica `i`,
  repetir para cada `j : (i,c,d)` en la familia
  1. Crear una nueva variable temporal `s` –
     ¡si dos o más de la familia tienen la misma tripleta, compartirán `s`!

UNIVERSIDAD SIMÓN BOLÍVAR

Reducción de Fuerza

- Identificadas las *familias* de variables de inducción, se modifican las instrucciones para usar sumas o restas en lugar de multiplicaciones.
- Para cada variable de inducción básica `i`,
  repetir para cada `j : (i,c,d)` en la familia
  1. Crear una nueva variable temporal `s` –
     ¡si dos o más de la familia tienen la misma tripleta, compartirán `s`!
  2. Reemplazar la asignación a `j` por `j := s`.

UNIVERSIDAD SIMÓN BOLÍVAR

Reducción de Fuerza

- Identificadas las *familias* de variables de inducción, se modifican las instrucciones para usar sumas o restas en lugar de multiplicaciones.
- Para cada variable de inducción básica `i`,
  repetir para cada `j : (i,c,d)` en la familia
  1. Crear una nueva variable temporal `s` –
     ¡si dos o más de la familia tienen la misma tripleta, compartirán `s`!
  2. Reemplazar la asignación a `j` por `j := s`.
  3. Después de cada asignación `i := i + n` en L, incorporar

```
s := s + c * n
```

UNIVERSIDAD SIMÓN BOLÍVAR

Reducción de Fuerza

- Identificadas las *familias* de variables de inducción, se modifican las instrucciones para usar sumas o restas en lugar de multiplicaciones.
- Para cada variable de inducción básica `i`,
  repetir para cada `j : (i,c,d)` en la familia
  1. Crear una nueva variable temporal `s` –
     ¡si dos o más de la familia tienen la misma tripleta, compartirán `s`!
  2. Reemplazar la asignación a `j` por `j := s`.
  3. Después de cada asignación `i := i + n` en L, incorporar

     ```
     s := s + c * n
     ```

  4. Agregar `s : (i,c,d)` a la familia de `i`.

UNIVERSIDAD SIMÓN BOLÍVAR

Reducción de Fuerza

- Identificadas las *familias* de variables de inducción, se modifican las instrucciones para usar sumas o restas en lugar de multiplicaciones.
- Para cada variable de inducción básica `i`,
  repetir para cada `j : (i,c,d)` en la familia
  1. Crear una nueva variable temporal `s` –
     ¡si dos o más de la familia tienen la misma tripleta, compartirán `s`!
  2. Reemplazar la asignación a `j` por `j := s`.
  3. Después de cada asignación `i := i + n` en L, incorporar

     ```
     s := s + c * n
     ```

  4. Agregar `s : (i,c,d)` a la familia de `i`.
  5. Agregar la inicialización al **final** del *pre-header*

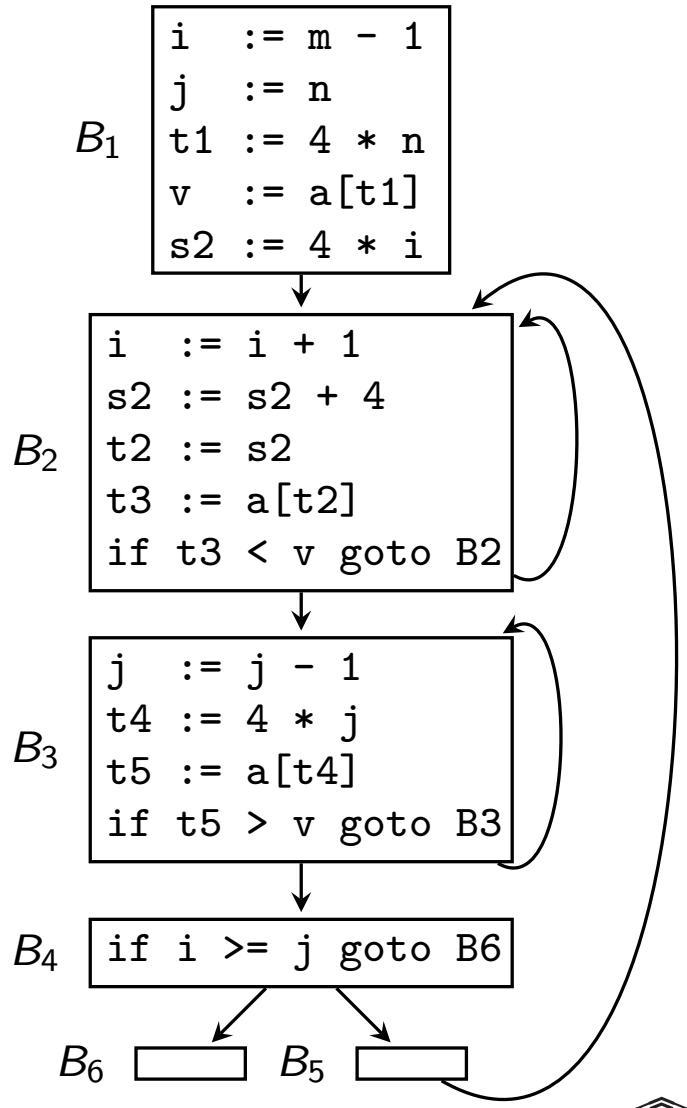
     ```
     s := c * i
     s := s + d
     ```

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Reducción de Fuerza

Sobre el Ciclo B_2

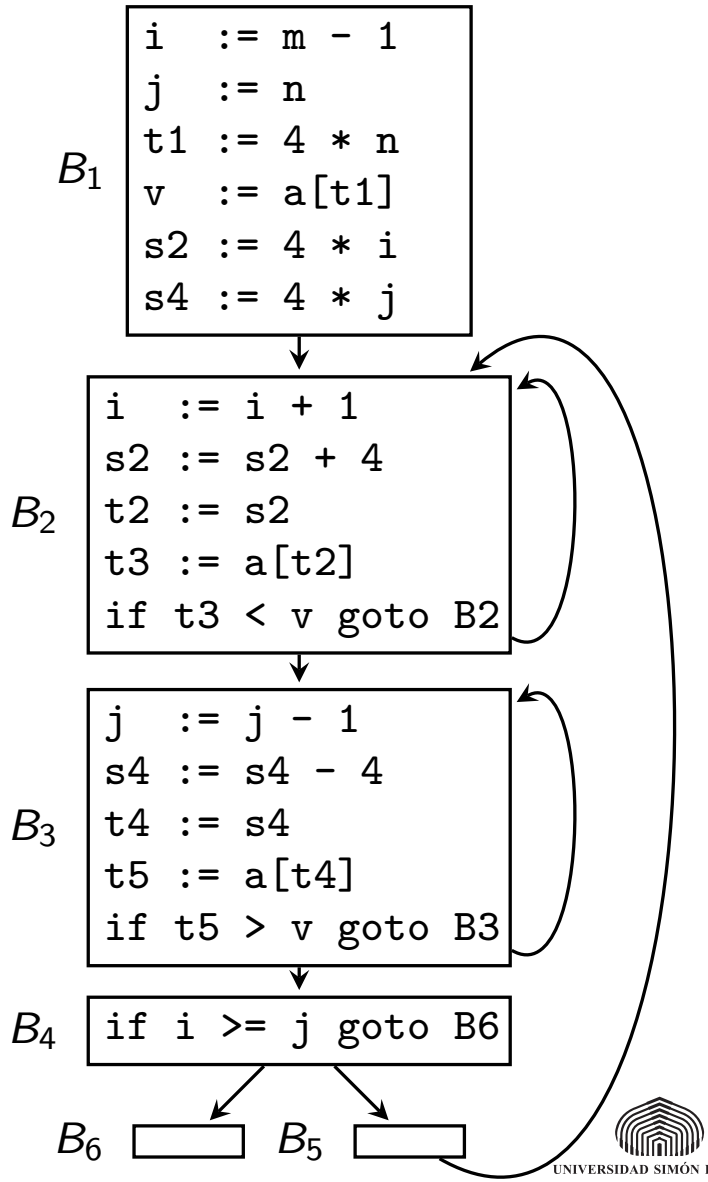
- Variable Básica `i`
  - Nueva variable `s2`
  - Reemplazar `t2 := 4 * i` por `t2 := s2`
  - Agregar `s2 := s2 + 4` después de la asignación de `i`.
  - Incorporar la inicialización al *pre-header* – usamos B_1 directamente.
  - Incorporar `s2 : (i,4,0)` a la familia de `i`.



Algoritmo de Reducción de Fuerza

Sobre el Ciclo B_3

- Variable Básica `j`
  - Nueva variable `s4`
  - Reemplazar `t4 := 4 * j` por `t4 := s4`
  - Agregar `s4 := s4 - 4` después de la asignación de `j`.
  - Incorporar la inicialización al *pre-header* – usamos B_1 directamente.
  - Incorporar `s4 : (j,4,0)` a la familia de `j`.



Eliminación de Variables de Inducción

Hit the road, Jack!

- Sea la variable de inducción básica `i`, tal que sólo se usa para:
  - Calcular otras variables de inducción en su familia.
  - Evaluar saltos condicionales.
- Podemos reemplazarla por alguna de sus derivadas.
  - Seleccionamos la `j : (i,c,d)` con los valores más simples de `c` y `d` – preferiremos `d` cero o uno, y `c` potencia de dos.
  - Modificamos los condicionales donde aparezca `i` para usar `j`.

Hemos de considerar dos casos.

UNIVERSIDAD SIMÓN BOLÍVAR

Eliminación de Variables de Inducción

Caso 1 – Una de las variables no es de inducción

```
if i op x goto B
```

- `x` **no** es variable de inducción
- Existe `j : (i,c,d)` con los `c` y `d` más simples.
- Cambiamos el condicional por

```
r := c * x
r := r + d
if j op r goto B
```

  - `r` es una nueva variable temporal.
  - La primera instrucción es `r := x` si `c = 1`.
  - Puede omitirse la segunda instrucción si `d = 0`.

UNIVERSIDAD SIMÓN BOLÍVAR

Eliminación de Variables de Inducción

Ambas variables son de inducción

`if i1 op i2 goto B`

- Ambas variables de inducción.
- Caso Fácil
  - `j1 : (i1,c,d)` y `j2 : (i2,c,d)`
  - Basta usar `if j1 op j2 goto B`.
- Caso Complejo
  - `j1 : (i1,c1,d1)` y `j2 : (i2,c2,d2)`
  - Hay que deducir la correlación matemática –
    no vale la pena hacer el cambio si requiere multiplicaciones.

UNIVERSIDAD SIMÓN BOLÍVAR

Eliminación de Variables de Inducción

- Después de sustituir las variables en los condicionales, las asignaciones a las variables sustituidas no son necesarias.
- Durante la reducción de fuerza se agregaron instrucciones `j := s`
  - Si no hay asignaciones a `s` entre la copia y el uso de `j`, reemplazar todos los usos de `j` por `s` y eliminar la copia.
  - Fácil de hacer en un mismo bloque – ¡es el caso frecuente!
  - Para hacerlo entre bloques diferentes es necesario calcular la información de uso de expresiones.

UNIVERSIDAD SIMÓN BOLÍVAR

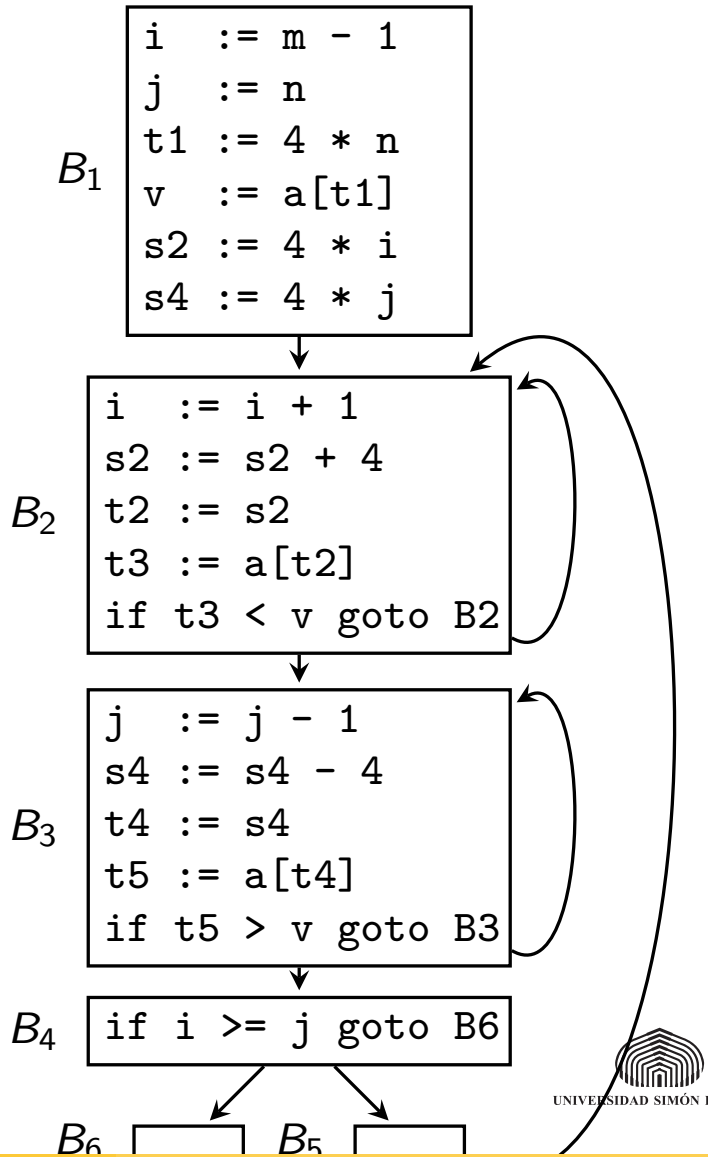
Algoritmo de Eliminación de Variables de Inducción

Sobre el Ciclo B_2

- `i` no puede eliminarse –
  usada en condicional *fuera* de B_2.
- `s2` no puede eliminarse –
  se usa para acceder al arreglo.

Sobre el Ciclo B_3

- `j` no puede eliminarse –
  usada en condicional *fuera* de B_3.
- `s4` no puede eliminarse –
  se usa para acceder al arreglo.



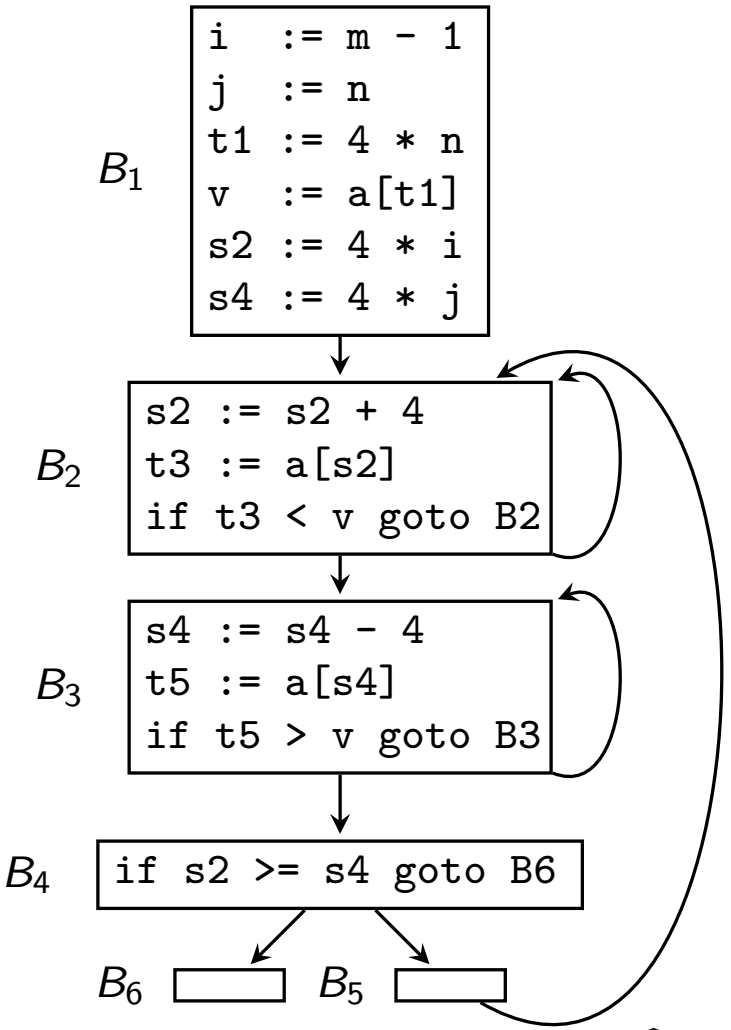
Algoritmo de Eliminación de Variables de Inducción

Sobre el Ciclo B_2, B_3, B_4, B_5

- `s2` familia de `i` – `s4` familia de `j`
- `i` y `j` sólo se usan en el condicional B_4.
- Encontramos el "caso fácil"

```
s2 : (i,4,0)
s4 : (j,4,0)
```

- Susituimos `i` por `s2` y `j` por `s4` en B_4.
- Eliminamos las asignaciones a `i` y `j`.
- Sustituimos `t2` por `s2`, y `t4` por `s4`.



UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*] (Primera Edición)
  - Sección 10.7
  - Ejercicios 10.1 a 10.3

UNIVERSIDAD SIMÓN BOLÍVAR