Métodos para la construcción de software fiable: Análisis estático

María del Mar Gallardo Melgarejo

Pedro Merino Gómez

Dpto. de Lenguajes y Ciencias de la Computación

Universidad de Málaga

(gallardo,pedro)@lcc.uma.es

Tres técnicas

- Análisis del flujo de datos
- Interpretación Abstracta

} *Análisis estático*

- Model checking ← *Análisis dinámico*

- Abstract Model checking

Análisis estático

- Conjunto de técnicas para predecir en tiempo de compilación *aproximaciones correctas* y *computables* del conjunto de valores o comportamientos que pueden ocurrir dinámicamente cuando un programa se ejecuta en un computador.
- Objetivos
  - Optimización del código generado por el compilador
  - Validación del software

Ejemplo:
Terminación y aproximación

```
read(x);
if x>0 then
  y:=1
else
  y:=2;S;(S no contiene asignaciones a y)
{¿qué valores toma y en este punto cualquiera
  que sea la ejecución de este programa?}
z := y
```

Ejemplo:
Terminación y aproximación

```
read(x);
if x>0 then
  y:=1
else
  y:=2;S;(S no contiene asignaciones a y)
```

{{y = 1}, {y = 2}} ← **Respuesta correcta**

```
z := y
```

Ejemplo:
Terminación y aproximación

```
read(x);
if x>0 then
  y:=1
else
  y:=2;S;(S no contiene asignaciones a y)
```

{{y = 1}} ← **Respuesta correcta?**

si S no termina es correcta y
más **precisa** que **{{y = 1}, {y = 2}}**

```
z := y
```

Ejemplo:
Terminación y aproximación

```
read(x);
if x>0 then
  y:=1
else
  y:=2;S;(S no contiene asignaciones a y)
```

{{y = 1}, {y = 2}, {y = 27}} ← **Respuesta correcta?**

```
z := y
```

- Es correcta pero **menos precisa** que **{{y = 1}, {y = 2}}**
- El reto está en encontrar respuestas *útiles* y correctas.
- Esta respuesta puede ser útil. Sabemos que y puede almacenarse en un byte de memoria.

Ejemplo:
Terminación y aproximación

```
read(x);
if x>0 then
  y:=1
else
  y:=2;S;(S no contiene asignaciones a y)
{x es entero}
z := y
```

- Esta respuesta es *correcta* pero es **poco útil**

Lenguaje WHILE

```
[y := x]¹ ; [z := 1]² ;
while [y > 1]³ do
    ([z := z * y]⁴ ; [y := y - 1]⁵);
[y := 0]⁶
```

- Tests
- Asignaciones
- Selecciones
- Bucles
- Cada *bloque elemental* tiene asociada una etiqueta que representa un punto del programa donde se obtiene información

Ejemplo: Análisis del ámbito de las definiciones

- Definición = Asignación
- La **definición** $[x := a]^l$ alcanza el punto **p** del programa (típicamente la entrada o salida de un bloque elemental) sii
  - hay una ejecución del programa en la que se pasa por el punto **p** y
  - la *última vez* que se le asignó un valor a *x* fue en *l*.
- En el ejemplo, $[y:=x]^1$ alcanza $[z := 1]^2$ o, de forma simplificada, el punto 2.
- El análisis viene dado por $RD = (RD_{entry}, RD_{exit})$

Ejemplo: Análisis del ámbito de las definiciones (AAD)

```
[y:=x]1 ; [z := 1]2 ;
while [y > 1]3 do
    ([z := z * y]4 ; [y := y - 1]5);
[y := 0]6
```

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	(x,?), (y,?), (z,?)	(x,?), (y,1), (z,?)
2	(x,?), (y,1), (z,?)	(x,?), (y,1), (z,2)
3	(x,?), (y,1), (y,5), (z,2), (z,4)	(x,?), (y,1), (y,5), (z,2), (z,4)
4	(x,?), (y,1), (y,5), (z,2), (z,4)	(x,?), (y,1), (y,5), (z,4)
5	(x,?), (y,1), (y,5), (z,4)	(x,?), (y,5), (z,4)
6	(x,?), (y,1), (y,5), (z,2), (z,4)	(x,?), (y,6), (z,2), (z,4)

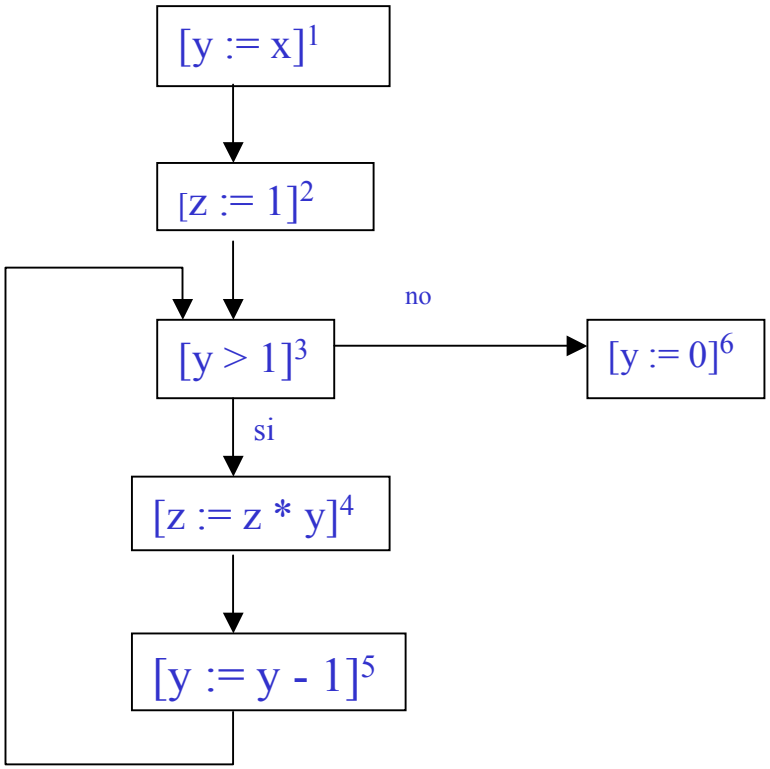
Ejemplo: Análisis del ámbito de las definiciones (AAD)

```
[y:=x]1 ; [z := 1]2 ;
while [y > 1]3 do
    ([z := z * y]4 ; [y := y - 1]5);
[y := 0]6
```

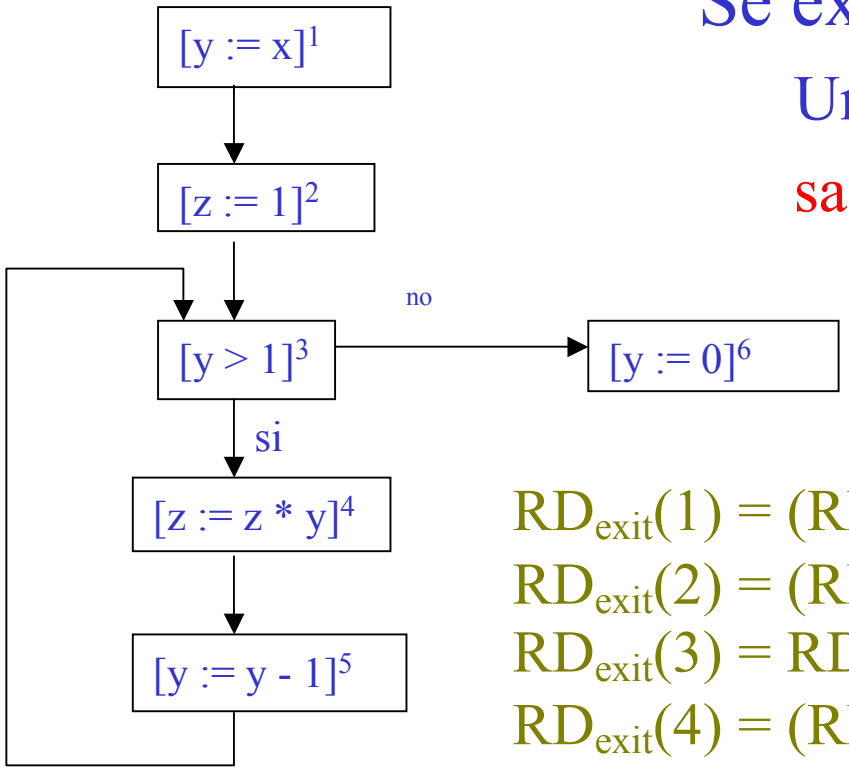
l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	(x,?), (y,?), (z,?)	(x,?), (y,1), (z,?)
2	(x,?), (y,1), (z,?)	(x,?), (y,1), (z,2)
3	(x,?), (y,1), (y,5), (z,2), (z,4)	(x,?), (y,1), (y,5), (z,2), (z,4)
4	(x,?), (y,1), (y,5), (z,2), (z,4)	(x,?), (y,1), (y,5), (z,4)
5	(x,?), (y,1), (y,5), (z,4), (z,2)	(x,?), (y,5), (z,4)
6	(x,?), (y,1), (y,5), ~~(z,2)~~, (z,4)	(x,?), (y,6), (z,2), (z,4)

Análisis de flujo de datos

```
[y:=x]1 ; [z := 1]2 ;
while [y > 1]3 do ([z := z * y]4 ; [y := y - 1]5);
[y := 0]6
```



Análisis de flujo de datos
Método ecuacional



Se extraen dos tipos de ecuaciones

Una que relaciona la

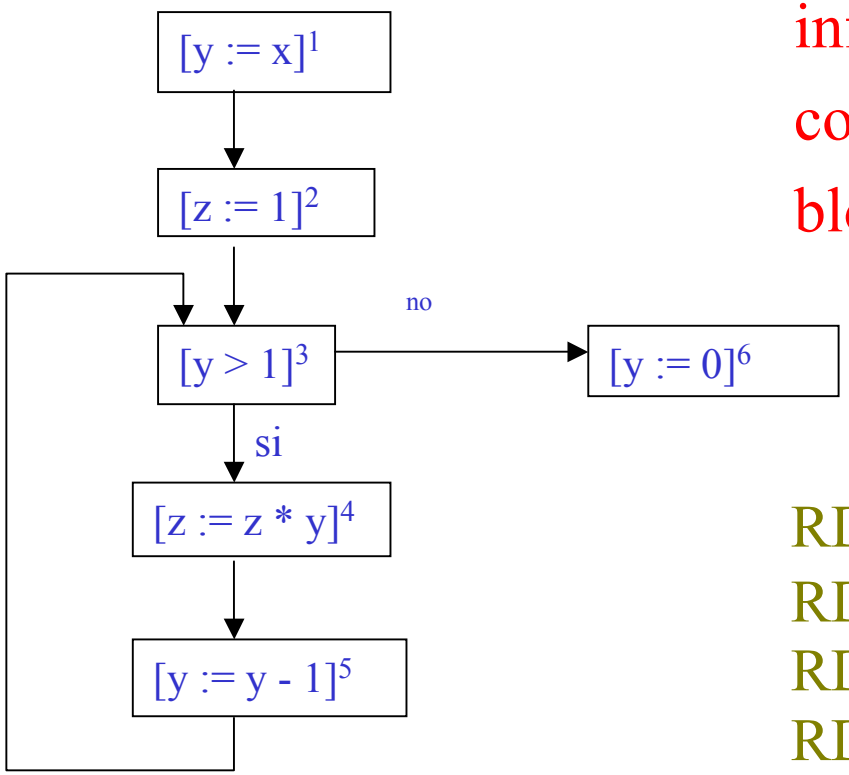
salida/entrada en un mismo bloque

$$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(y,l) | \, l \in Lab\}) \cup \{(y,1)\}$$
$$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(z,l) | \, l \in Lab\}) \cup \{(z,2)\}$$
$$RD_{exit}(3) = RD_{entry}(3)$$
$$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(z,l) | \, l \in Lab\}) \cup \{(z,4)\}$$
$$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(y,l) | \, l \in Lab\}) \cup \{(y,5)\}$$
$$RD_{exit}(6) = (RD_{entry}(6) \setminus \{(y,l) | \, l \in Lab\}) \cup \{(y,6)\}$$

Análisis de flujo de datos

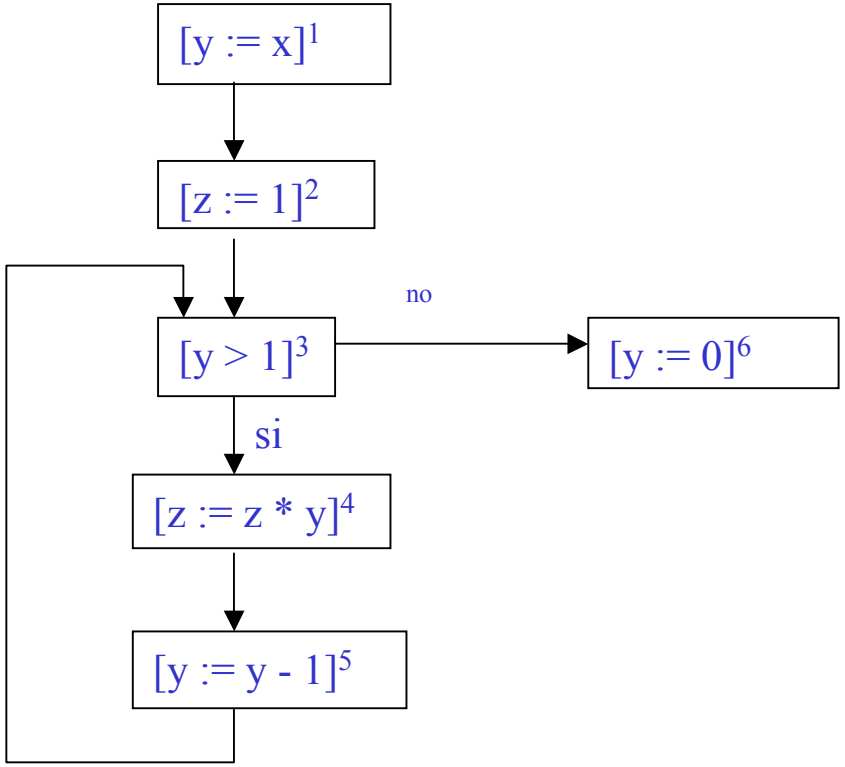
Método ecuacional

Otra que relaciona la información de entrada de un bloque con la de salida de todos los bloques que se conectan con él



$$RD_{entry}(2) = RD_{exit}(1)$$
$$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$$
$$RD_{entry}(4) = RD_{exit}(3)$$
$$RD_{entry}(5) = RD_{exit}(4)$$
$$RD_{entry}(6) = RD_{exit}(3)$$

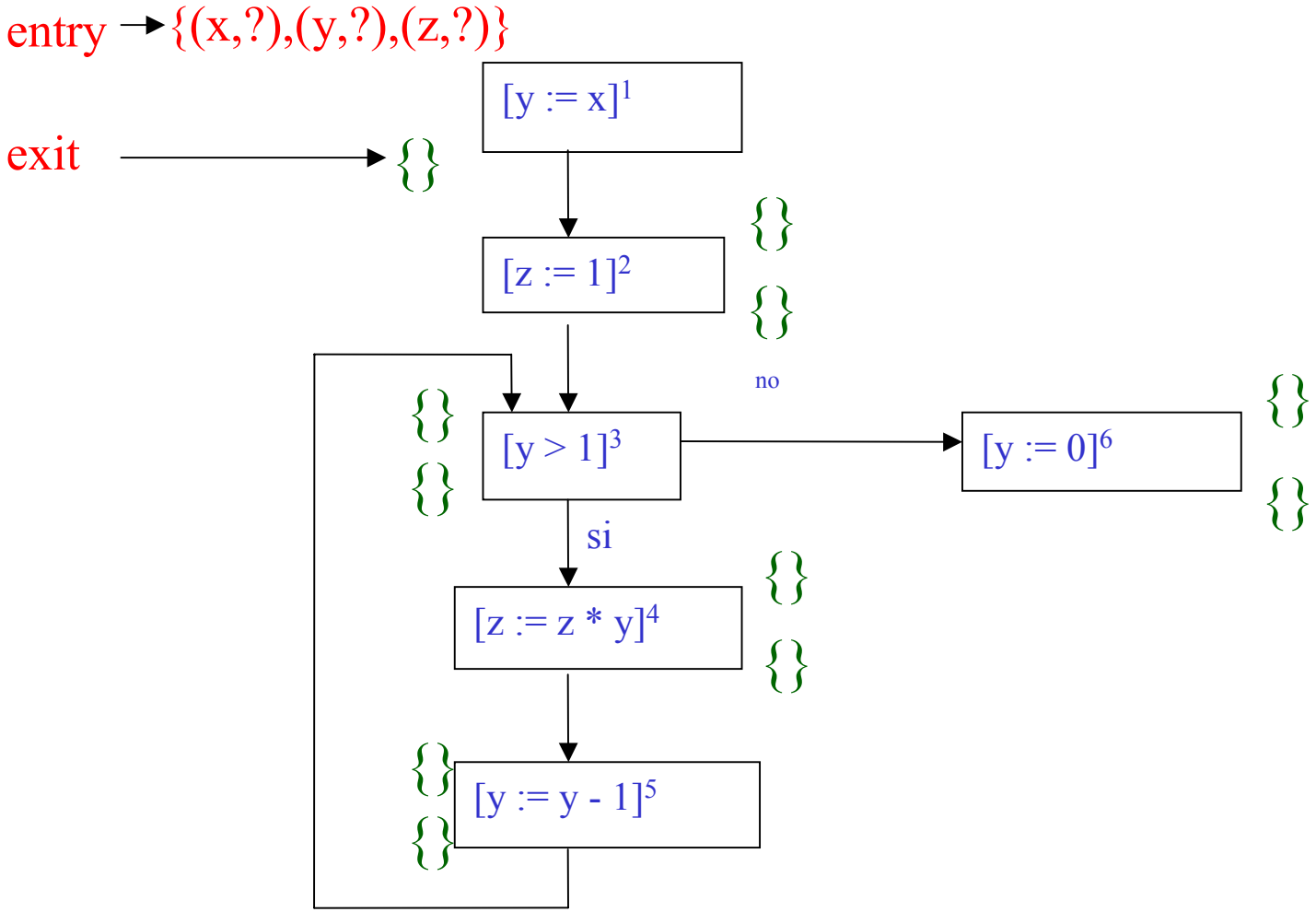
Análisis de flujo de datos
Método ecuacional



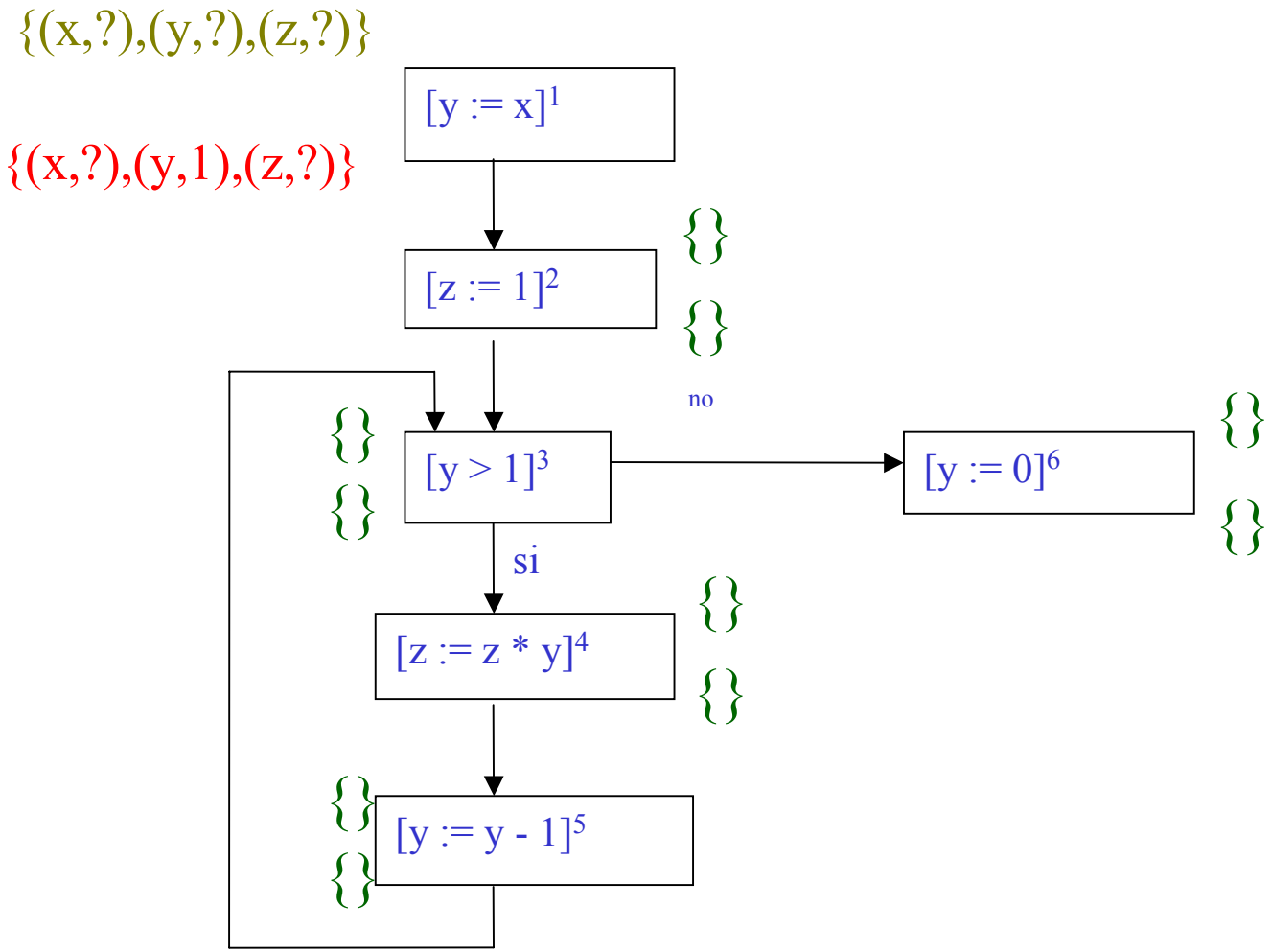
Inicialmente

$RD_{entry}(1) = \{(x,?),(y,?),(z,?)\}$

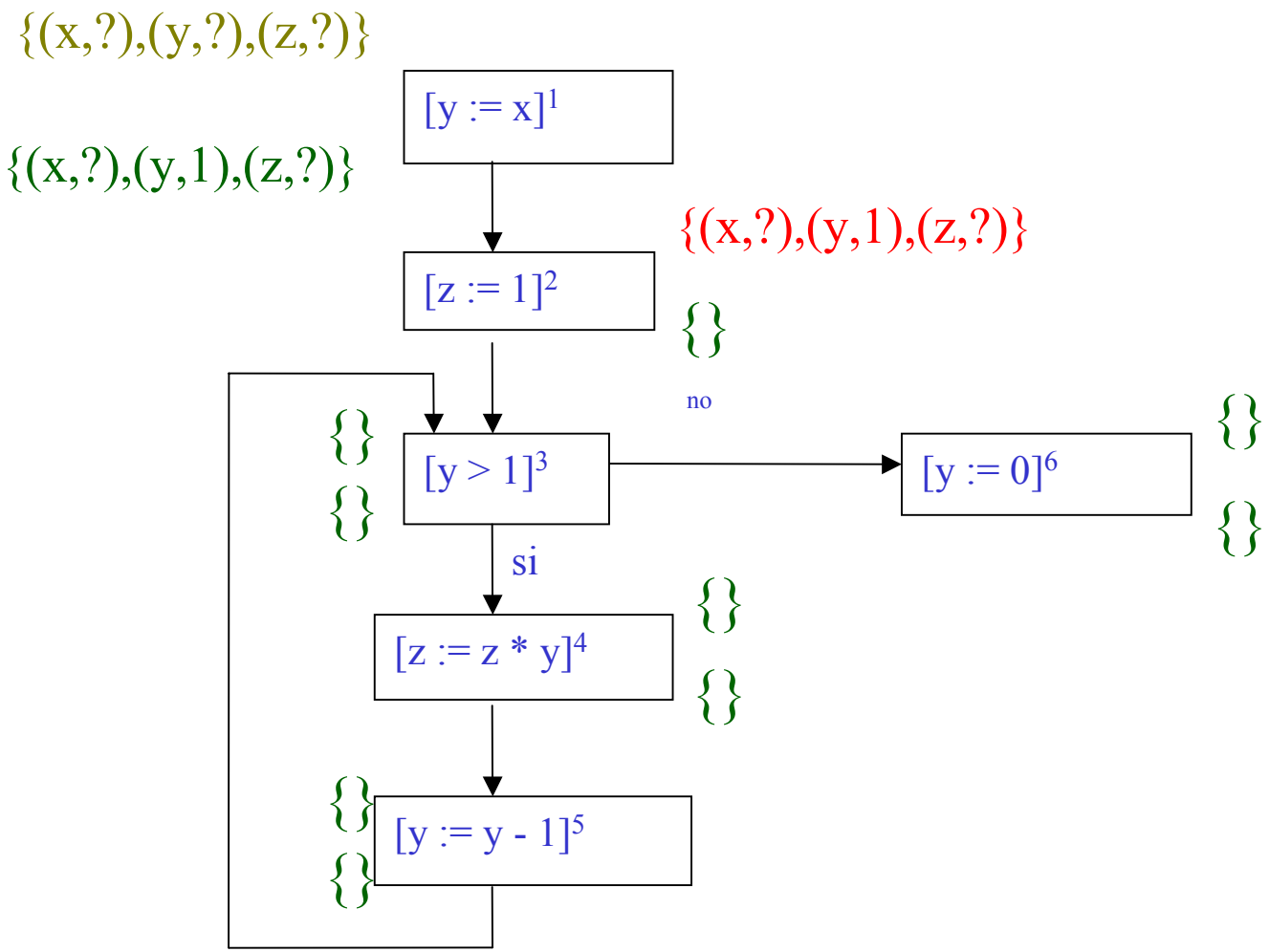
Análisis de flujo de datos
Método ecuacional



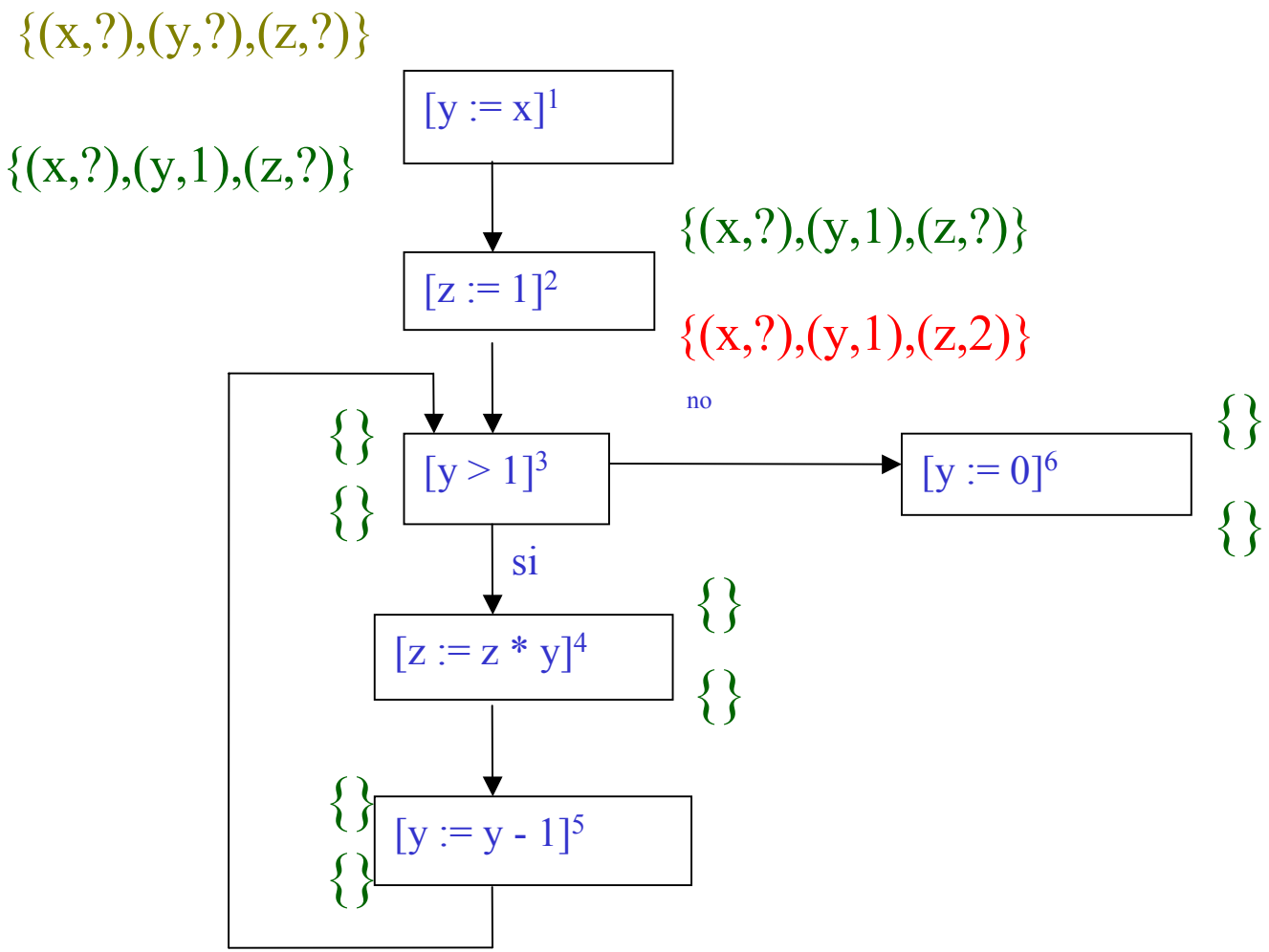
Análisis de flujo de datos
Método ecuacional



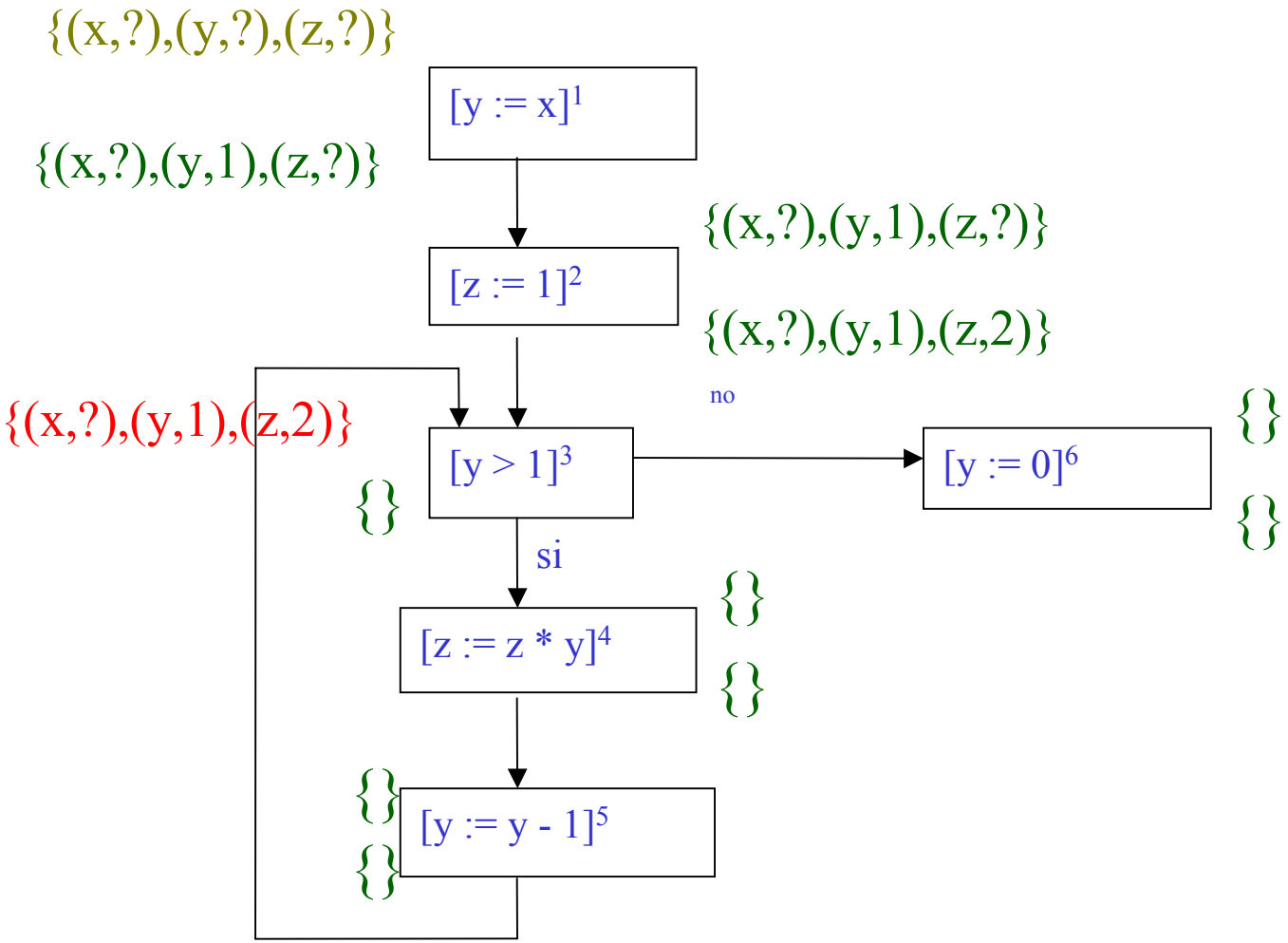
Análisis de flujo de datos
Método ecuacional



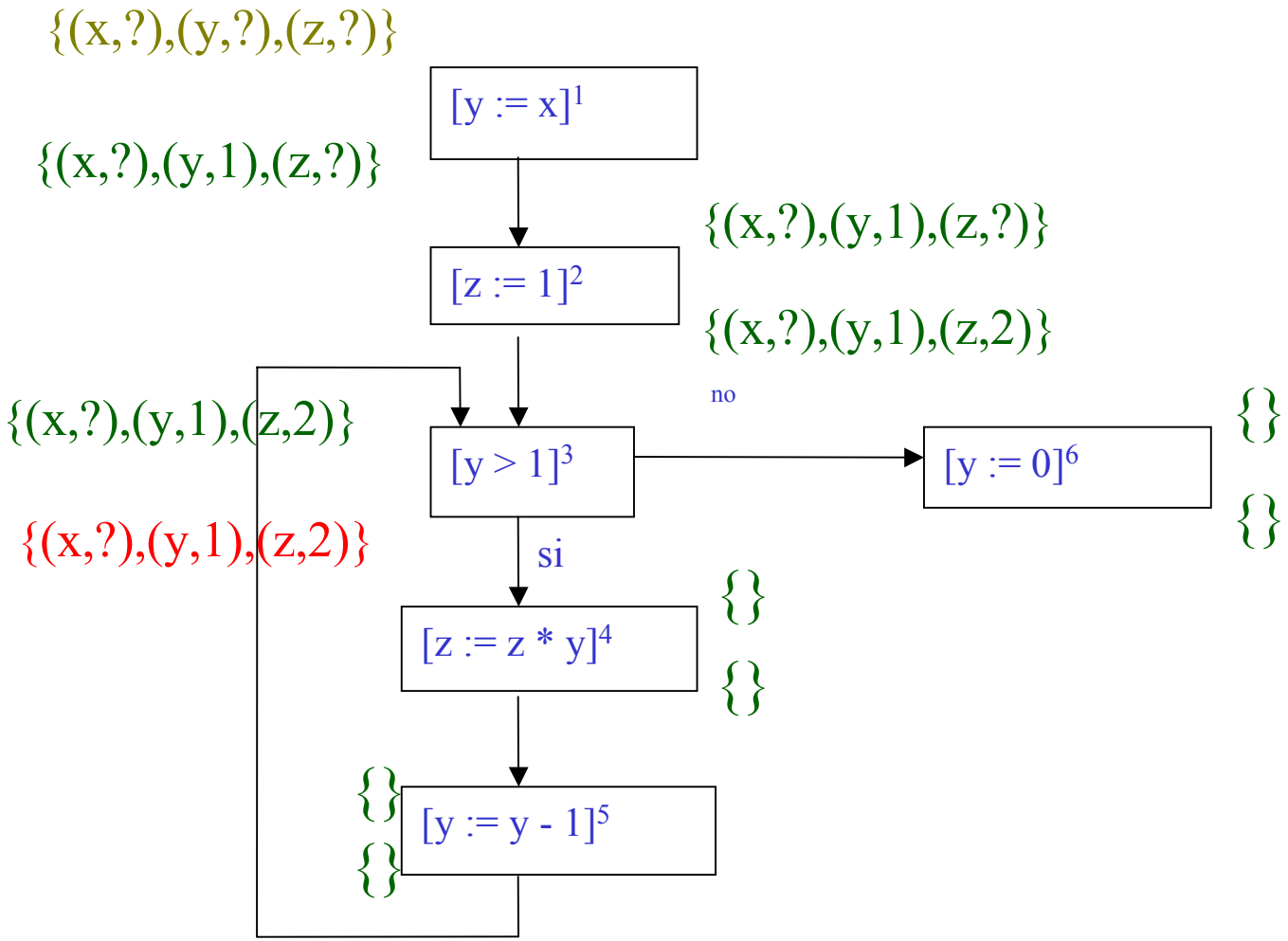
Análisis de flujo de datos
Método ecuacional



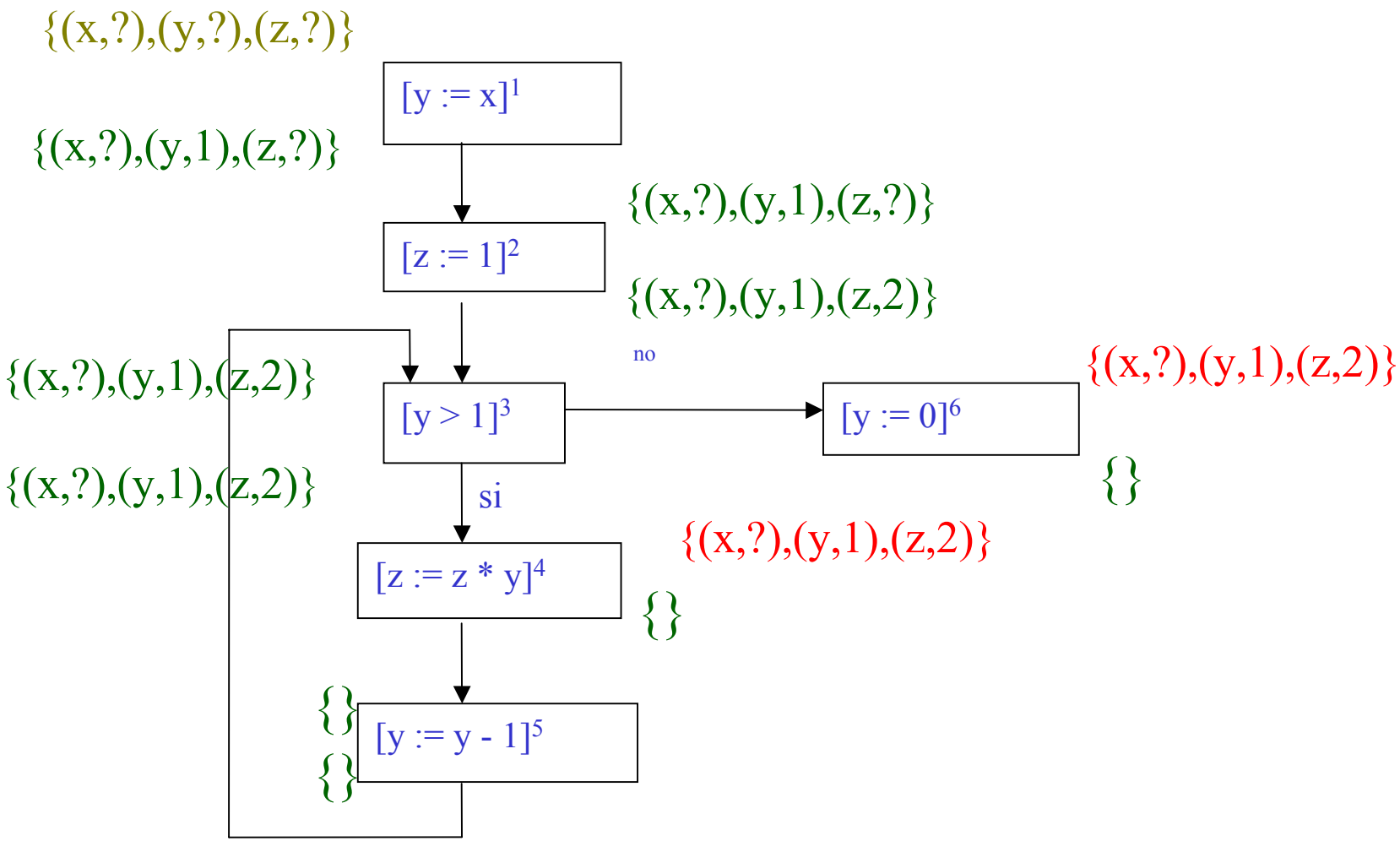
Análisis de flujo de datos
Método ecuacional



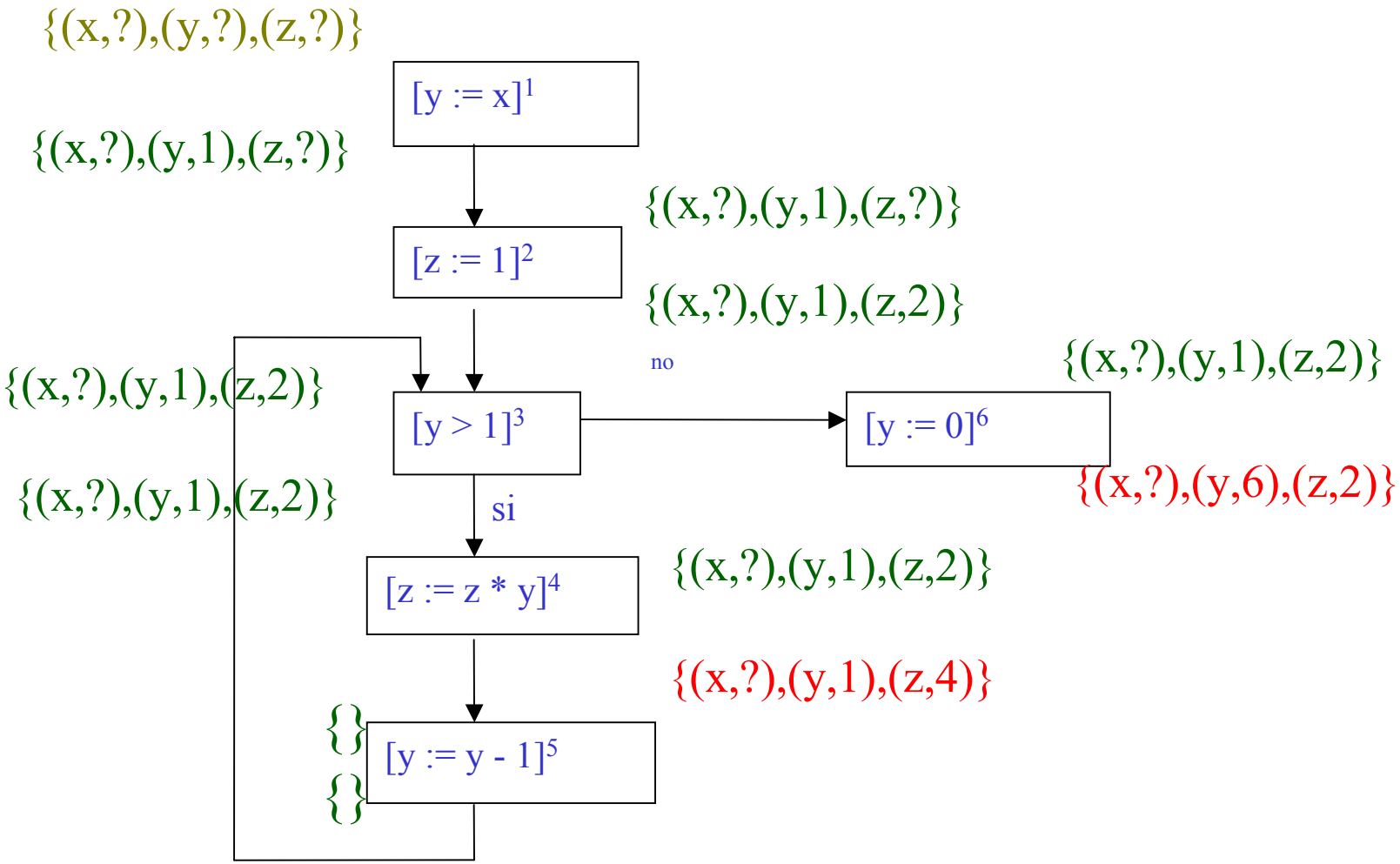
Análisis de flujo de datos
Método ecuacional



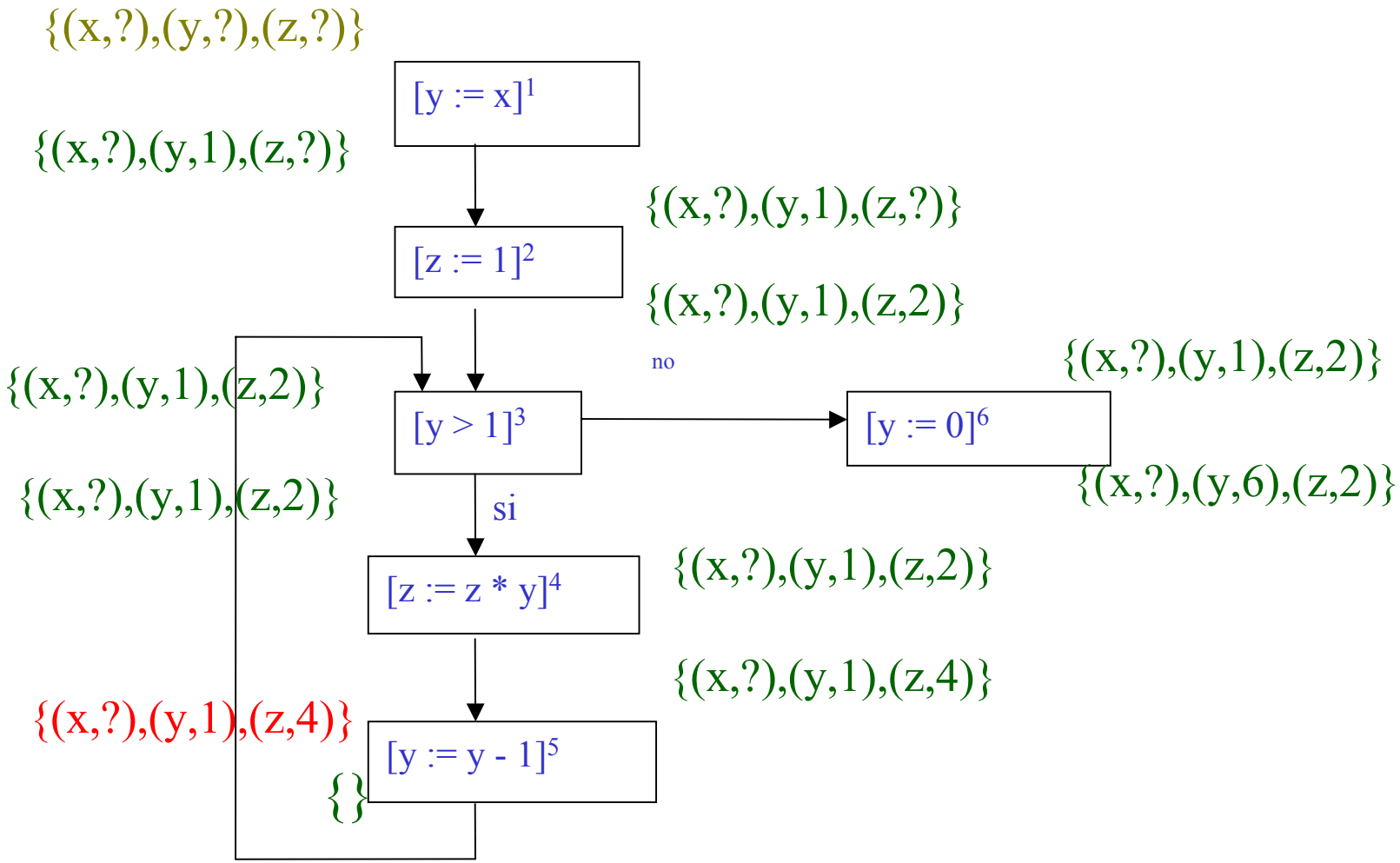
Análisis de flujo de datos
Método ecuacional



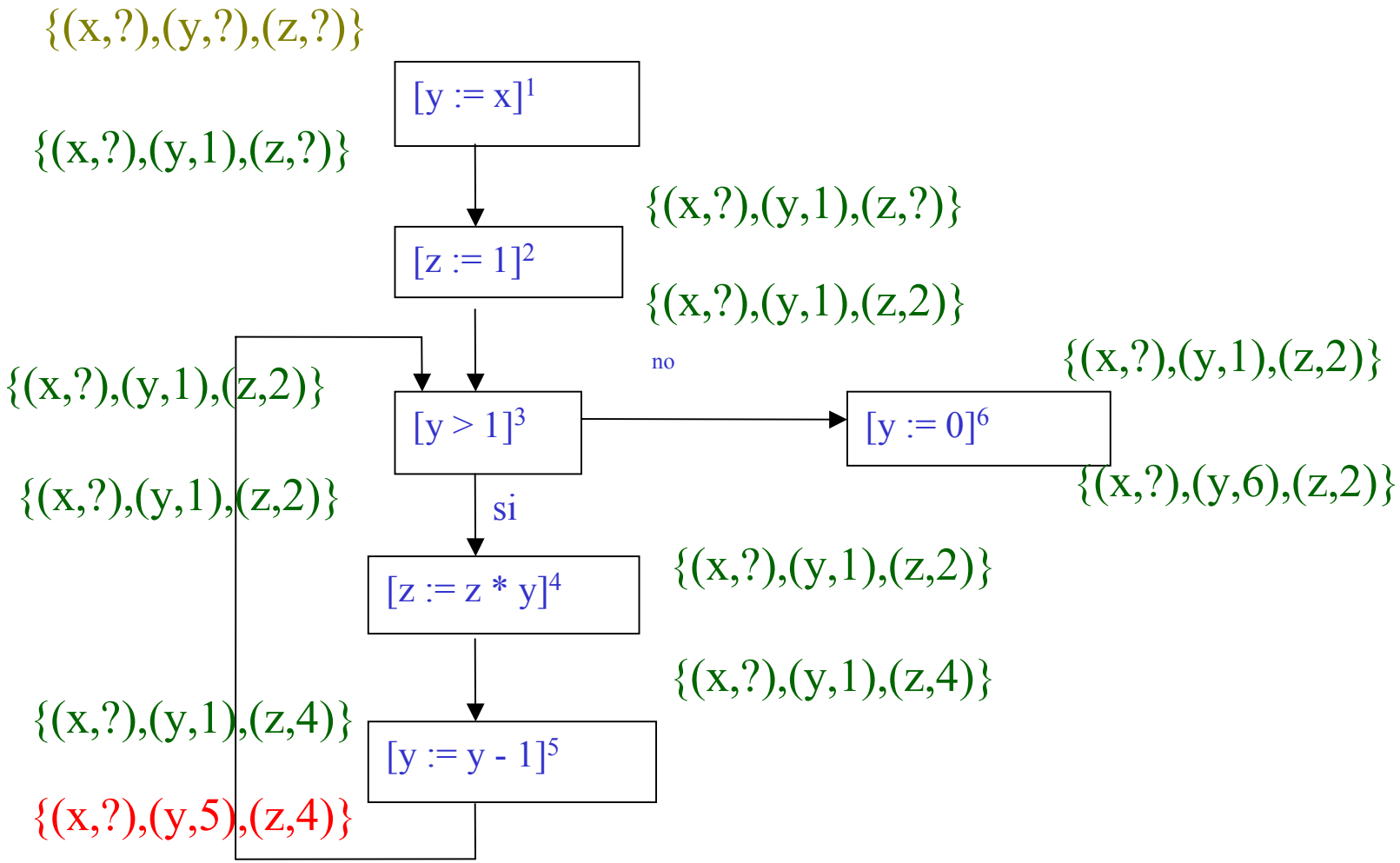
Análisis de flujo de datos
Método ecuacional



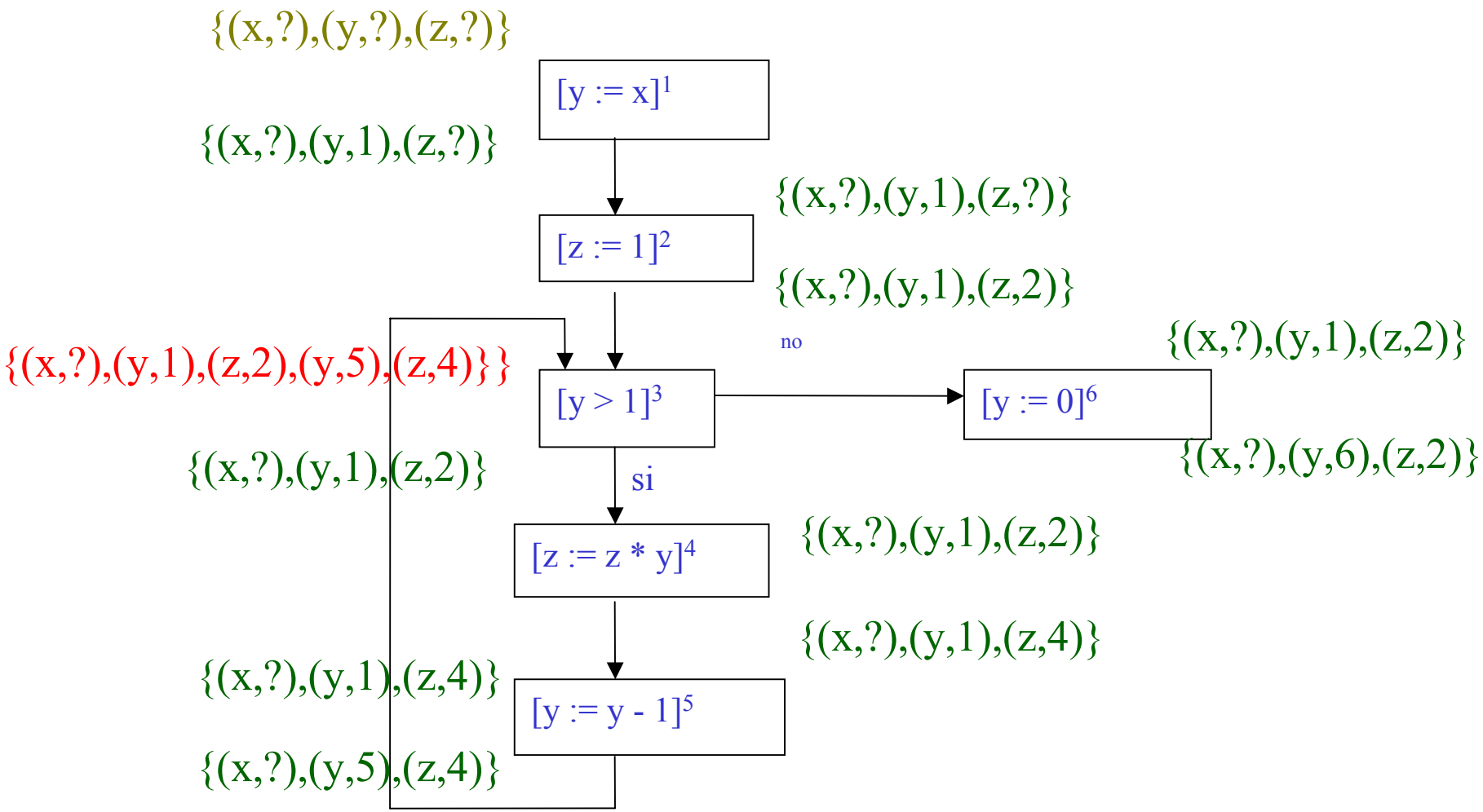
Análisis de flujo de datos
Método ecuacional



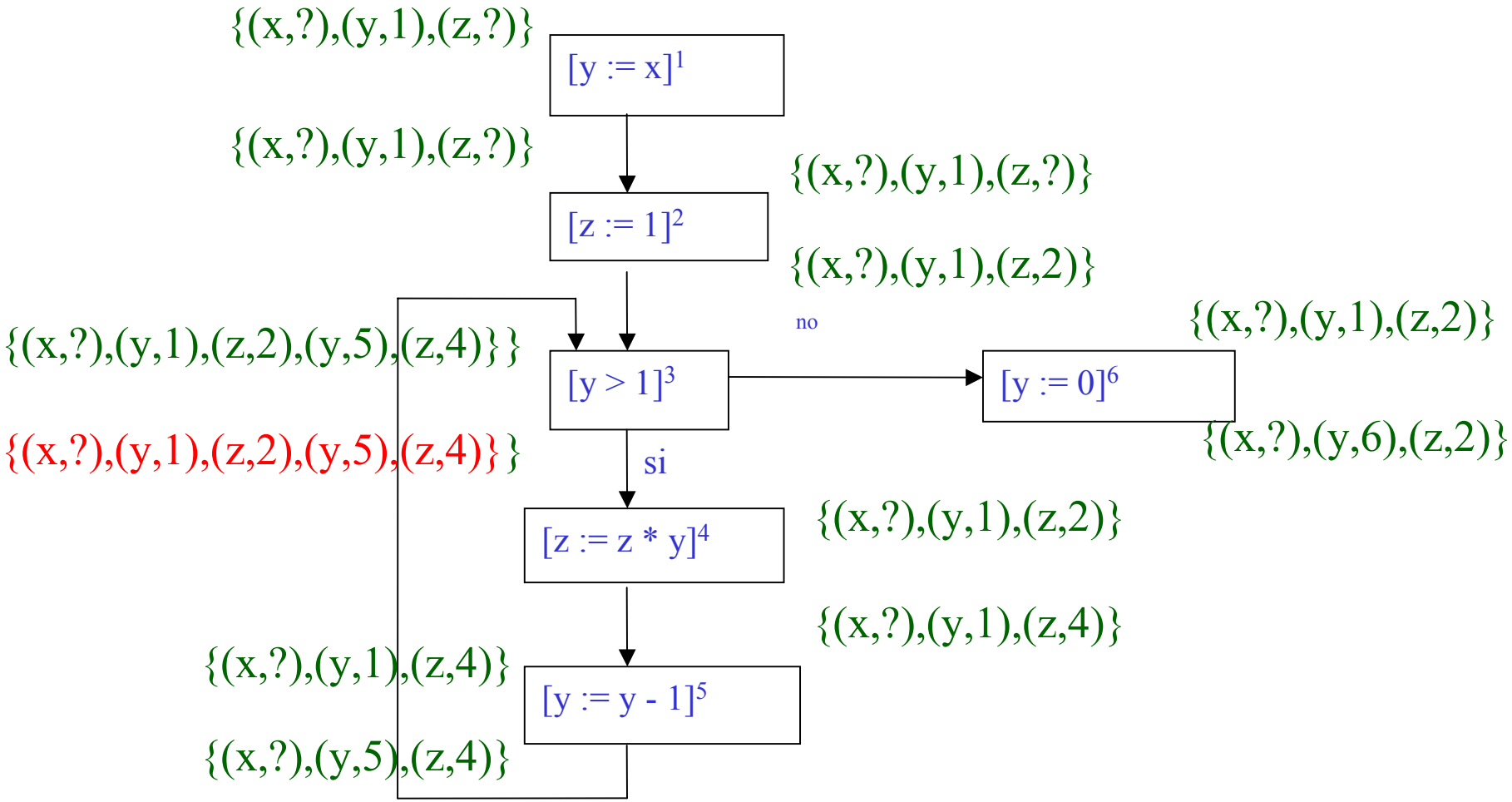
Análisis de flujo de datos
Método ecuacional



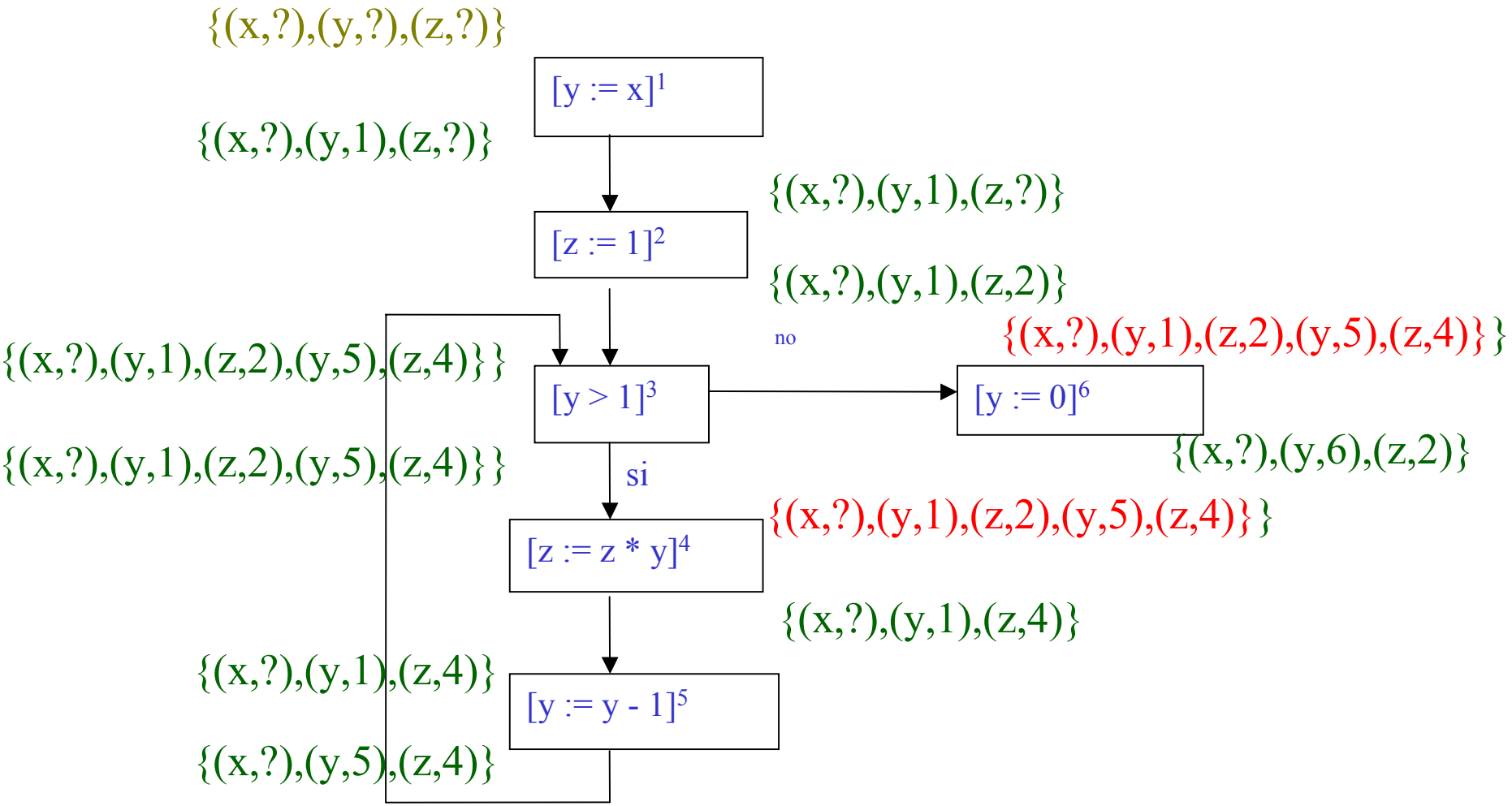
Análisis de flujo de datos
Método ecuacional



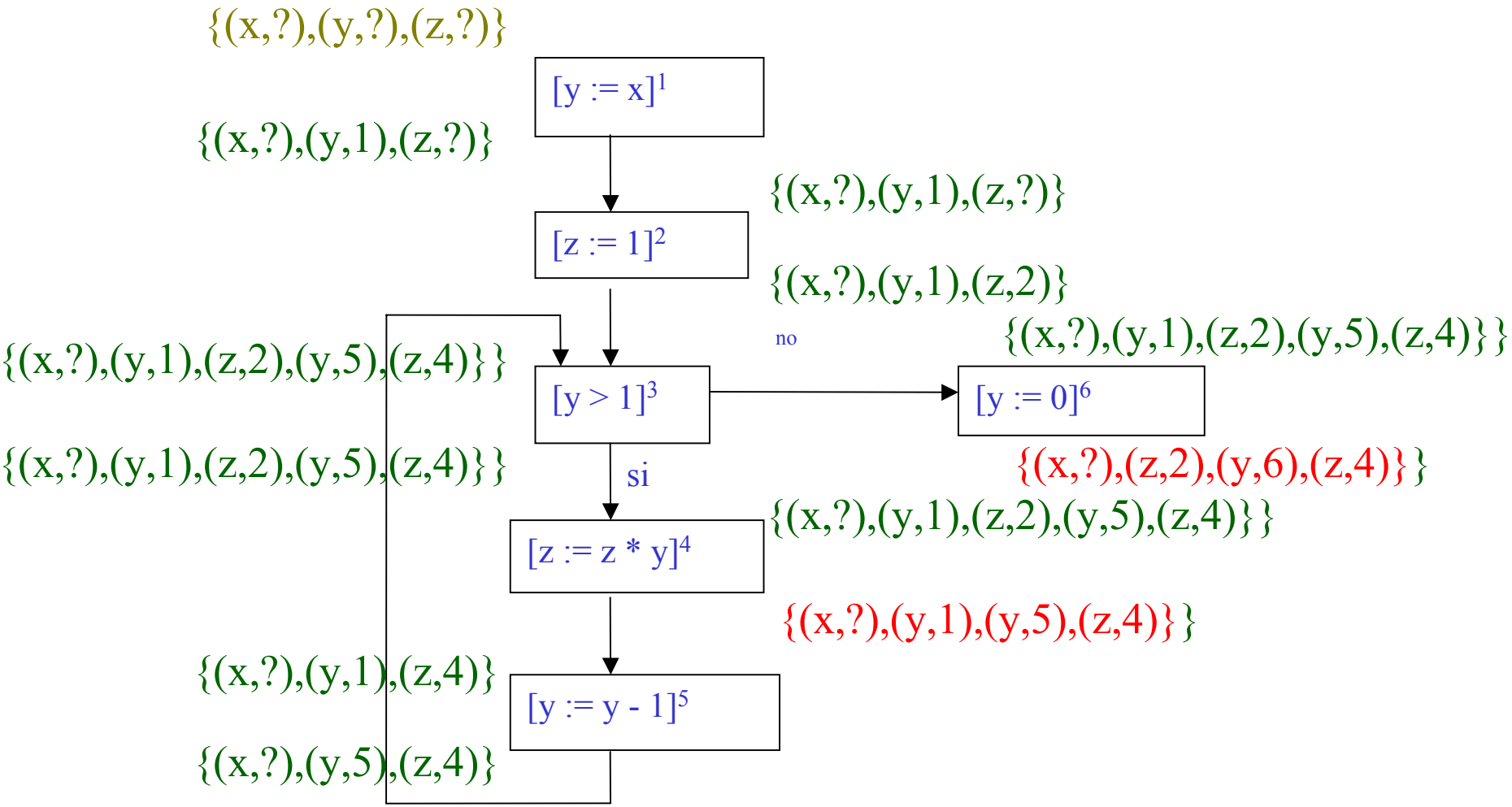
Análisis de flujo de datos
Método ecuacional



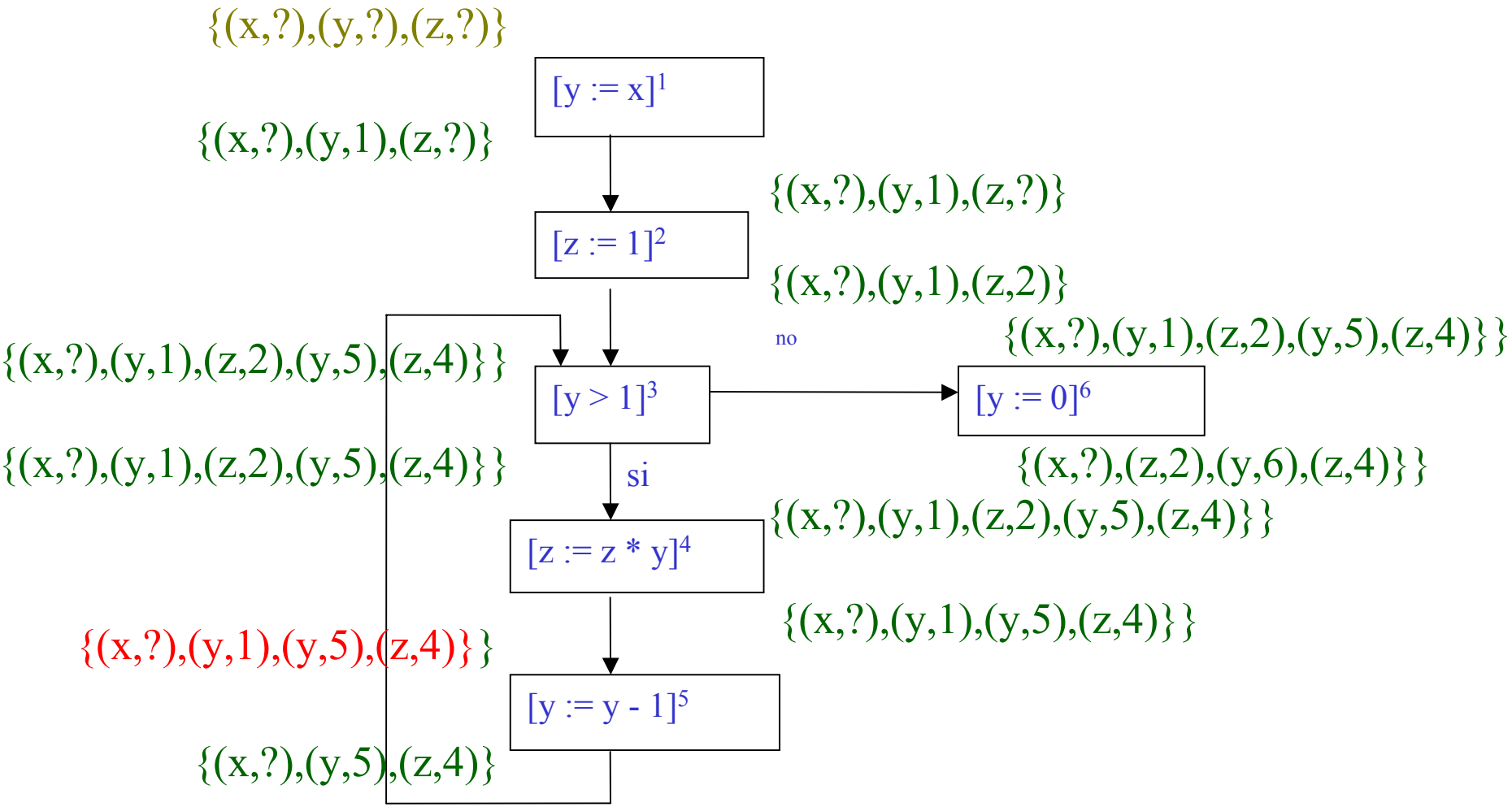
Análisis de flujo de datos
Método ecuacional



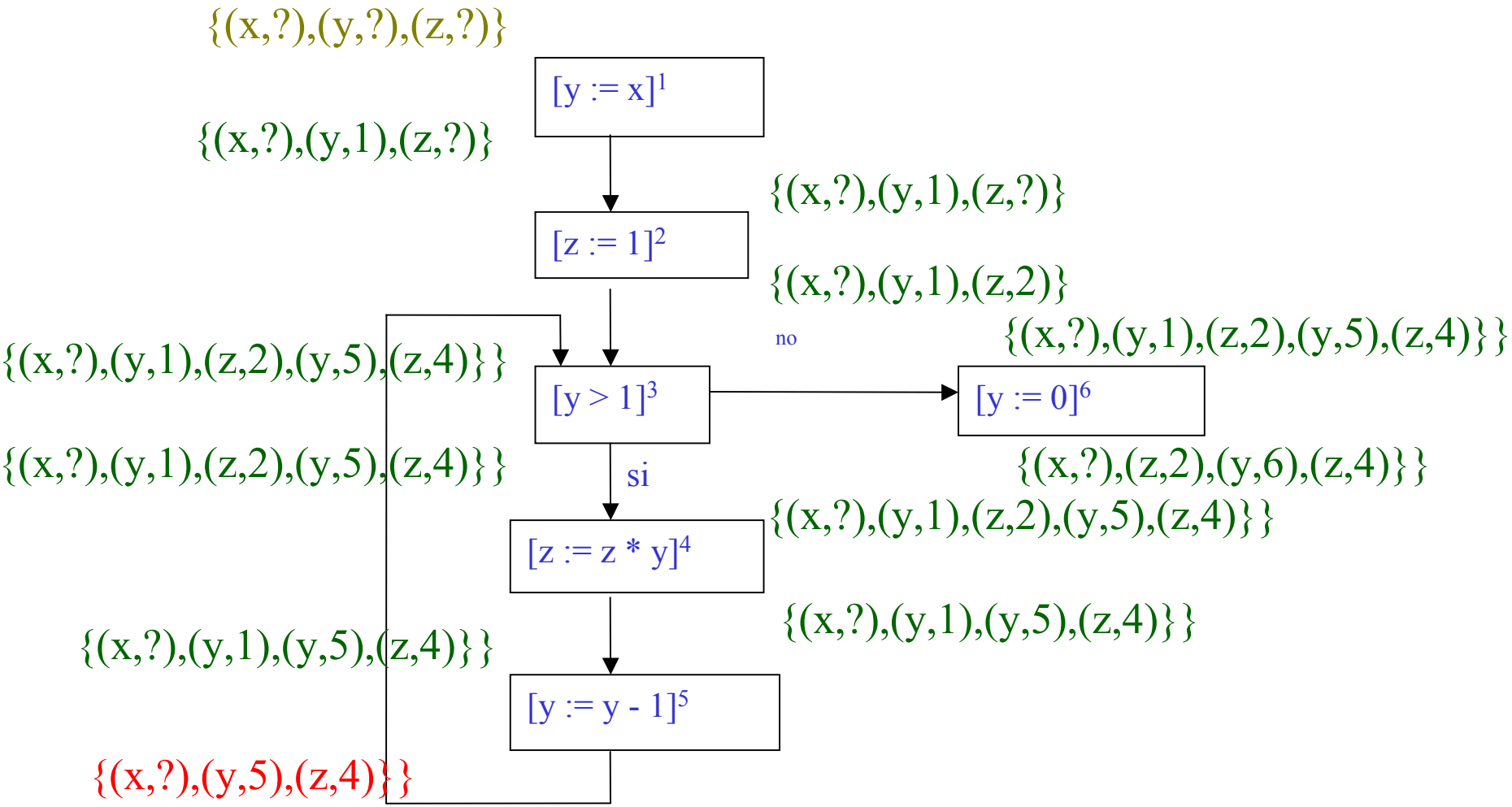
Análisis de flujo de datos
Método ecuacional



Análisis de flujo de datos
Método ecuacional

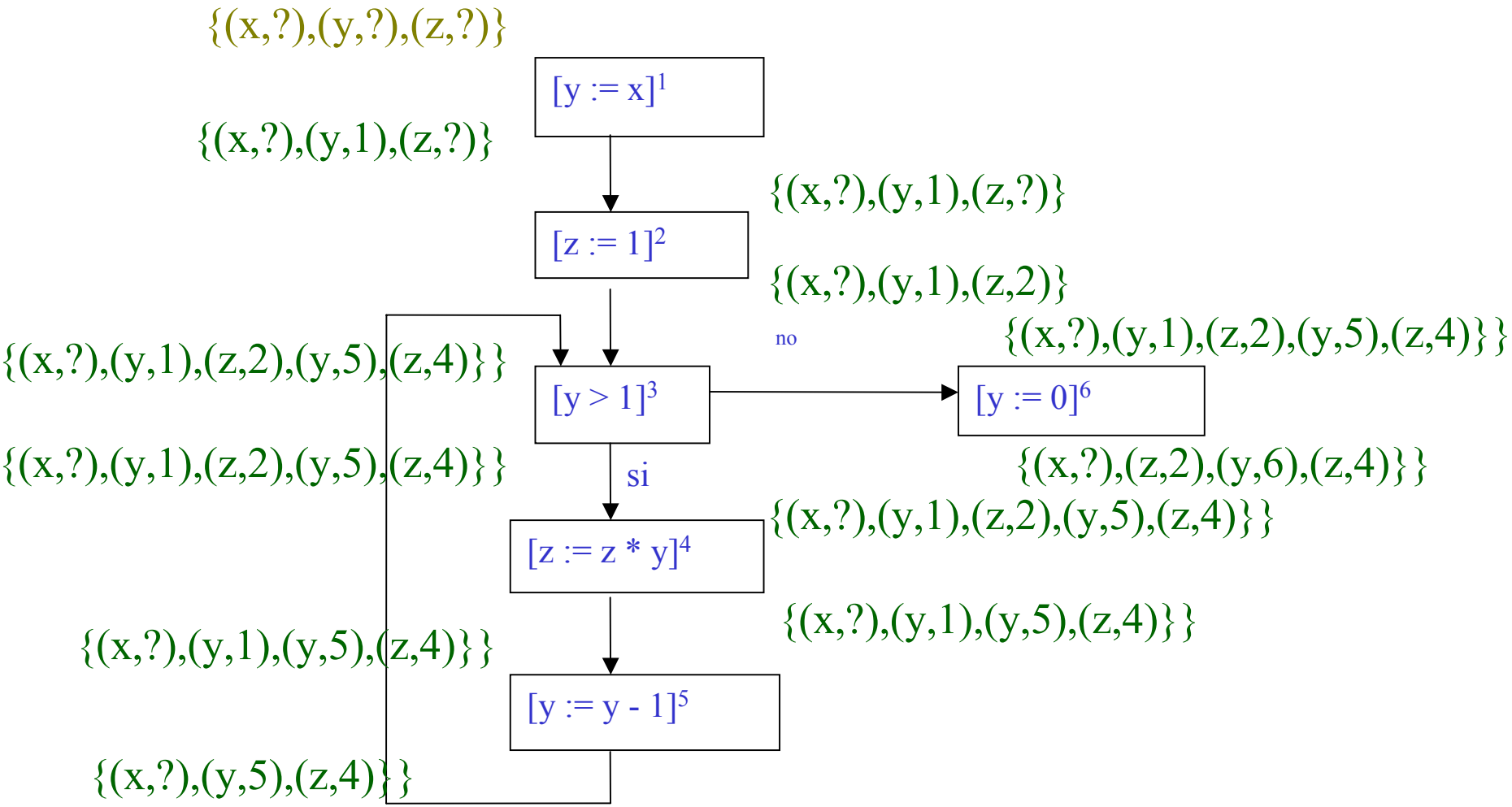


Análisis de flujo de datos
Método ecuacional



Análisis de flujo de datos
Método ecuacional

Punto fijo



AFD. Método Ecuacional

La menor solución

- El sistema de ecuaciones anterior puede escribirse como una función que transforma conjuntos de definiciones - pares del tipo (var,lab) - en conjuntos de definiciones

$$F : (\wp(Var_* \times Lab_*))^{12} \rightarrow (\wp(Var_* \times Lab_*))^{12}$$

$$Var^*=\{x,y,z\} \text{ y } Lab =\{1,\ldots,6,?\}$$

$$F(RD) = (F_{entry}(1)(RD), F_{exit}(1)(RD), \ldots , F_{entry}(12)(RD), F_{exit}(12)(RD))$$

P. e. $F_{entry}(3)(\ldots, RD_{exit}(2), \ldots ,RD_{exit}(5), \ldots) = RD_{exit}(2) \cup RD_{exit}(5)$

– Buscamos RD tal que F(RD) = RD

AFD. Método Ecuacional. La menor solución #2

- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ está parcialmente ordenado

  $RD \subseteq RD'$ sii $\forall\, i : RD_i \subseteq RD'_i$,

  siendo $RD = (RD_1, \ldots, RD_{12})$ y $RD' = (RD'_1, \ldots, RD'_{12})$

AFD. Método Ecuacional. La menor solución #2

- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ está parcialmente ordenado

  $RD \subseteq RD'$ sii $\forall\, i : RD_i \subseteq RD'_i$,

  siendo $RD = (RD_1, \ldots, RD_{12})$ y $RD' = (RD'_1, \ldots, RD'_{12})$

- $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$ es un retículo completo
  - $\emptyset = (\emptyset, \ldots, \emptyset)$ es el ínfimo
  - $RD \cup RD' = (RD_1 \cup RD'_1, \ldots, RD_{12} \cup RD'_{12})$

AFD. Método Ecuacional. La menor solución #2

- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ está parcialmente ordenado

  $RD \subseteq RD'$ sii $\forall\, i : RD_i \subseteq RD'_i$,

  siendo $RD = (RD_1, \ldots, RD_{12})$ y $RD' = (RD'_1, \ldots, RD'_{12})$

- $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$ es un retículo completo
  - $\emptyset = (\emptyset, \ldots, \emptyset)$ es el ínfimo
  - $RD \cup RD' = (RD_1 \cup RD'_1, \ldots, RD_{12} \cup RD'_{12})$
- F es una función monótona sobre $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$

AFD. Método Ecuacional. La menor solución #2

- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ está parcialmente ordenado

  $RD \subseteq RD'$ sii $\forall\, i : RD_i \subseteq RD'_i$,

  siendo $RD = (RD_1, \ldots, RD_{12})$ y $RD' = (RD'_1, \ldots, RD'_{12})$

- $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$ es un retículo completo
  - $\emptyset = (\emptyset, \ldots, \emptyset)$ es el ínfimo
  - $RD \cup RD' = (RD_1 \cup RD'_1, \ldots, RD_{12} \cup RD'_{12})$
- F es una función monótona sobre

  $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$
- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ es finito

AFD. Método Ecuacional. La menor solución #2

- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ está parcialmente ordenado

  $RD \subseteq RD'$ sii $\forall\, i : RD_i \subseteq RD'_i$,

  siendo $RD = (RD_1, \ldots, RD_{12})$ y $RD' = (RD'_1, \ldots, RD'_{12})$
- $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$ es un retículo completo
  - $\emptyset = (\emptyset, \ldots, \emptyset)$ es el ínfimo
  - $RD \cup RD' = (RD_1 \cup RD'_1, \ldots, RD_{12} \cup RD'_{12})$
- F es una función monótona sobre $((\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}, \subseteq)$
- $(\wp(\mathrm{Var}_* \times \mathrm{Lab}_*))^{12}$ es finito
- $\emptyset \subseteq F(\emptyset) \subseteq F^2(\emptyset) \subseteq \ldots \subseteq F^m(\emptyset) \subseteq$ se estabiliza en el menor punto fijo

Algoritmo para calcular el menor punto fijo

- Buscamos RD tal que $RD = F(RD)$
- donde $F: (\wp(Var_* \times Lab_*))^{12} \rightarrow (\wp(Var_* \times Lab_*))^{12}$
  - es una función monótona
  - $(\wp(Var_* \times Lab_*))^{12}$ es finito
- la solución deseada se puede obtener como $F^n(\varnothing)$ para cualquier n tal que $F^n(\varnothing) = F^{n+1}(\varnothing)$
- sabemos que dicho n además existe.

Algoritmos. Iteración Caótica

$RD = (RD_1, \ldots, RD_{12})$

$F(RD) = (F(RD_1), \ldots, F(RD_{12}))$

INPUT: Ecuaciones del ejemplo para el análisis del ámbito de definiciones.

OUTPUT: La menor solución $RD = (RD_1, \ldots, RD_{12})$

MÉTODO: **Paso 1**: Inicialización

$RD_1 := \varnothing; \ldots; RD_{12} := \varnothing$

Paso 2: Iteración

while $RD_j \neq F_j(RD_1, \ldots, RD_{12})$ para algún j do

$RD_j = F_j (RD_1, \ldots, RD_{12})$

Otros análisis de flujo de datos

- Análisis de las expresiones disponibles

  Available expressions -- AE
- Análisis del ámbito de las definiciones (2)

  Reaching definitions -- RD
- Análisis de las expresiones muy ocupadas

  Very busy expressions -- VBE
- Análisis de las variables libres

  Live variables -- LV

Terminología: Etiquetas iniciales

- Stmt ::= $[x:=a]^l$ | $[skip]^l$ | Stmt ; Stmt |
  if $[b]^l$ then Stmt else Stmt | while $[b]^l$ do Stmt

- init : Stmt → Lab
  devuelve la etiqueta inicial de una instrucción

  $init([x:=a]^l) = l$
  $init([skip]^l) = l$
  $init(S_1 ; S_2) = init(S_1)$
  $init(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2) = l$
  $init(\text{while } [b]^l \text{ do } S) = l$

Terminología: Etiquetas finales

– final:Stmt → $\wp$(Lab)

devuelve el conjunto de etiquetas finales de una instrucción

final($[x:=a]^l$) = l

final($[skip]^l$) = l

final(S_1 ; S_2) = final(S_2)

final(if $[b]^l$ then S_1 else S_2) = final(S_1) $\cup$ final(S_2)

final(while $[b]^l$ do S) = l

Terminología: Bloques

- Blocks ::= $[x:=a]^l \mid [skip]^l \mid [b]^l$
- blocks : Stmt → $\wp$(Blocks)

  Para acceder a las instrucciones simples o tests asociados a una instrucción usamos la función blocks

$blocks([x:=a]^l) = \{[x.=a]^l\}$

$blocks([skip]^l) = \{[skip]^l\}$

$blocks(S_1 ; S_2) = blocks(S_1) \cup blocks(S_2)$

$blocks(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2) = \{[b]^l\} \cup blocks(S_1) \cup blocks(S_2)$

$blocks(\text{while } [b]^l \text{ do } S) = \{[b]^l\} \cup blocks(S)$

Terminología: Etiquetas

- labels: Stmt → $\wp$(Lab)

  etiquetas de una instrucción

  - labels(S) = $\{l \mid [B]^l \in \text{blocks}(S)\}$

- Claramente
  - init(S) $\in$ labels(S)
  - final(S) $\subseteq$ labels(S)

Terminología: Flujo de datos

- flow : Stmt → $\wp(\text{Lab} \times \text{Lab})$
  - Asocia instrucciones con conjuntos de flechas

$$\text{flow}([x.=a]^l) = \varnothing$$
$$\text{flow}([\text{skip}]^l) = \varnothing$$
$$\text{flow}(S_1 ; S_2) = \text{flow}(S_1) \cup \text{flow}(S_2) \cup \{(l,\text{init}(S_2)) \mid l \in \text{final}(S_1)\}$$
$$\text{flow}(\text{if } [b]^l \text{ then } S_1 \text{ else } S_2) = \text{flow}(S_1) \cup \text{flow}(S_2) \cup \{(l,\text{init}(S_1)), (l,\text{init}(S_2))\}$$
$$\text{flow}(\text{while } [b]^l \text{ do } S) = \text{flow}(S) \cup \{(l,\text{init}(S))\} \cup \{(l',l) \mid l' \in \text{final}(S)\}$$

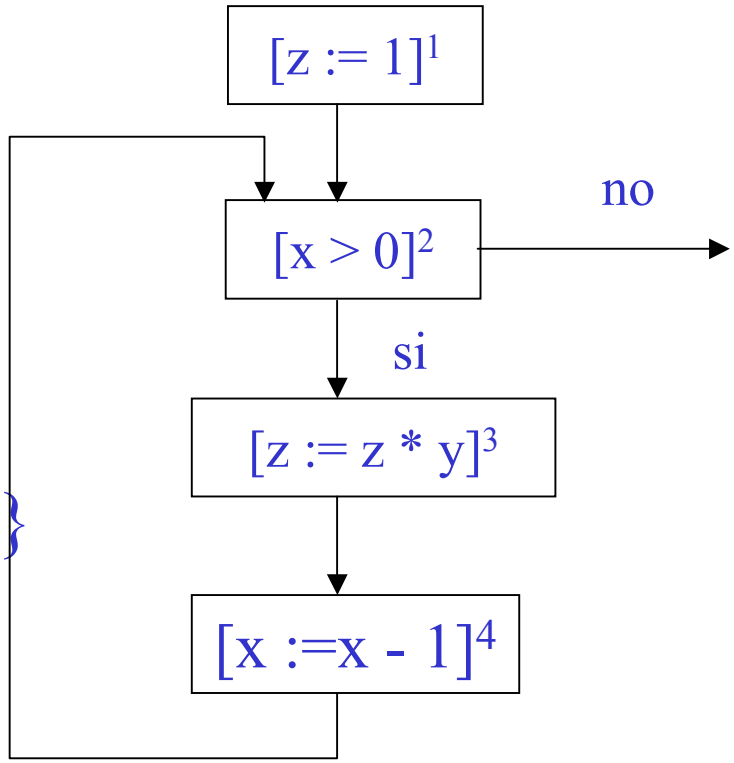
Terminología: Flujo de datos #2

labels(S) y flow(S) representan el grafo S

Power = $[z:=1]^1$; while $[x > 0]^2$ do ($[z := z * y]^4$; $[x := x - 1]^5$)

- init(power) = 1
- final(power) = {2}
- labels(power) = {1,2,3,4}
- flow(power) =

  {(1,2), (2,3), (3,4), (4,2)}

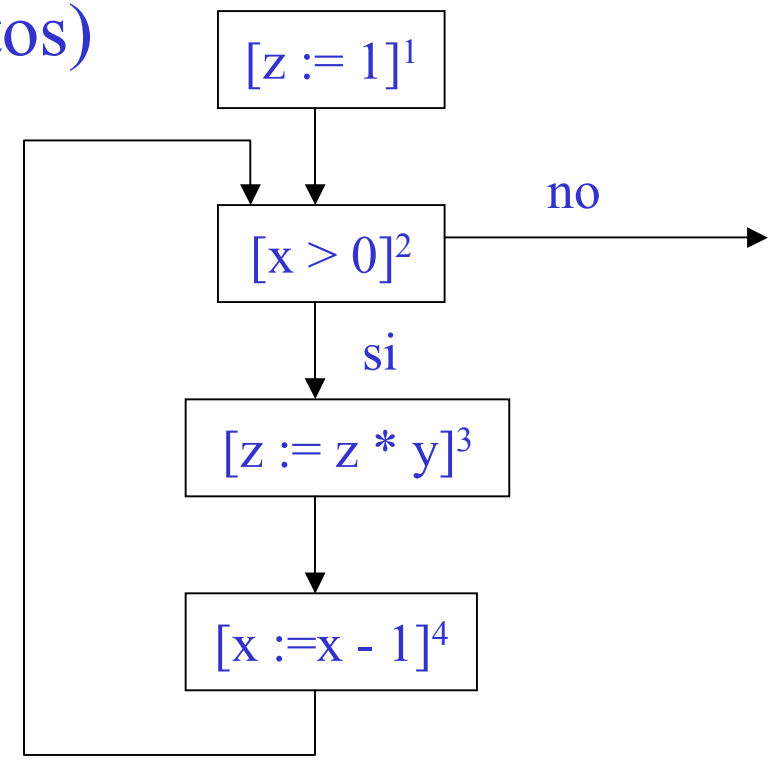


Terminología: Flujo de datos #3

flow se utiliza en la formalización de los análisis hacia delante (que siguen la misma dirección que el flujo de datos)

- labels(S) = {init(S)} ∪
  {l | (l,l') ∈ flow(S)} ∪
  {l' | (l,l') ∈ flow(S)}

- si S ≠ $[B]^l$
  labels(S) =
  {l | (l,l') ∈ flow(S)} ∪
  {l' | (l,l') ∈ flow(S)}

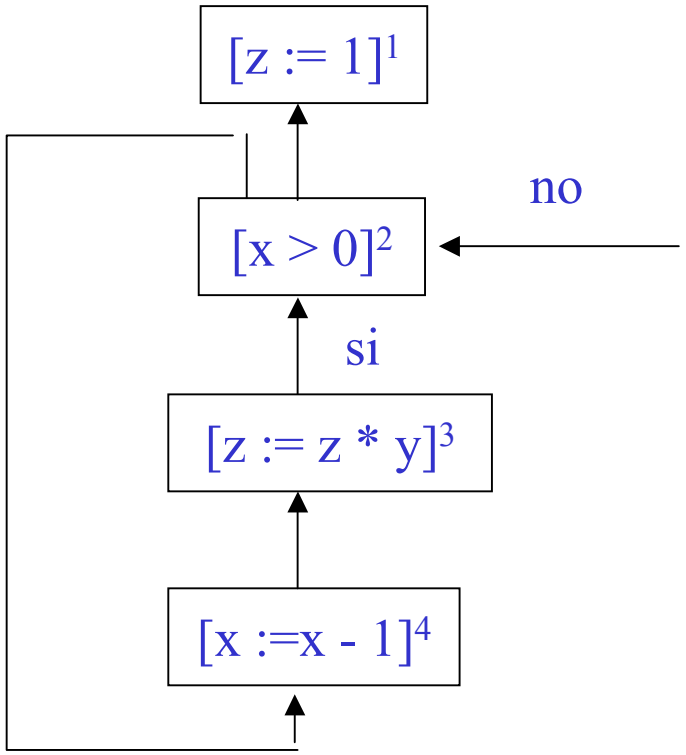


Terminología: Flujo de datos #4

- flow^R : Stmt → $\wp(\text{Lab} \times \text{Lab})$
  - $\text{flow}^R(S) = \{(l,l') \mid (l',l) \in \text{flow}(S)\}$

$\text{flow}^R(\text{power}) =$

$\{(2,1), (2,4), (3,2), (4,3)\}$



Análisis del Flujo de datos
Preliminares

- **S**$_*$ es el programa que estamos analizando (las instrucciones de alto nivel)
- **Lab**$_*$ representa las etiquetas (labels(S$_*$)) que aparecen en S$_*$,
- **Var**$_*$ representa las variables de S$_*$
- **Blocks**$_*$ representa los bloques elementales (blocks(S$_*$))
- **AExp**$_*$ representa el conjunto de subexpresiones aritméticas no triviales de S$_*$;
  - una expresión es trivial si es una variable o una constante.
- **AExp(a)** y **AExp(b)** representa el conjunto de subexpresiones aritméticas (booleanas) no triviales de una expresión dada.

Análisis del Flujo de datos
Preliminares #2

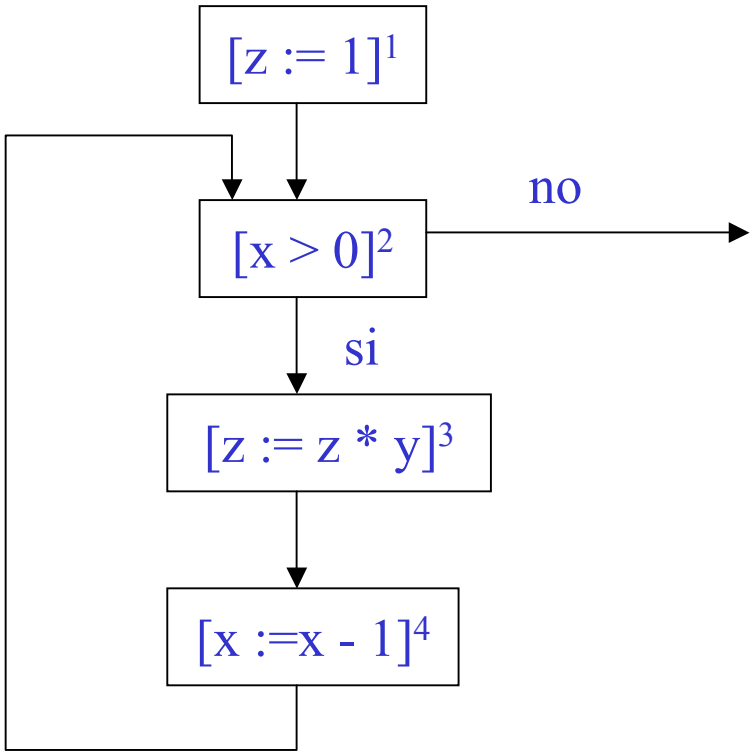
- suponemos que el programa S_* tiene **entradas aisladas**, es decir,
  - $\forall\, l \in \text{Lab} : (l, \text{init}(S_*)) \notin \text{flow}(S_*)$

  Esto ocurre siempre que S_* no comience en un bucle while.
- A veces supondremos que el programa también tiene **salidas aisladas**, es decir,
  - $\forall\, l_1 \in \text{final}(S_*)\ \forall\, l_2 \in \text{Lab} : (l_1, l_2) \notin \text{flow}(S_*)$

- Una instrucción, S, es **etiqueta-consistente** sii:
  - $[B_1]^l, [B_2]^l \in \text{blocks}(S)$ implica que $B_1 = B_2$

Análisis del Flujo de datos Preliminares #3

- power tiene entradas aisladas,
- Es claramente etiqueta-consistente
- No tiene salidas aisladas



Análisis de las Expresiones Disponibles

- Determina:
  - Para cada punto del programa, las expresiones que han sido ya calculadas, y que ya no se modifican posteriormente, en todos los caminos que llegan a dicho punto.
  - Esta información es útil para evitar la re-computación de una expresión. (Nos centramos en expresiones aritméticas)

Análisis de las Expresiones Disponibles #2

$[x := a+b]^1$; $[y := a * b]^2$;
while $[y > a+b]^3$ do ($[a := a + 1]^4$; $[x := a+b]^5$)

la expresión **a + b** está disponible cada vez que la ejecución alcanza el test (etiqueta 3) en el bucle; como consecuencia, la expresión no tiene que ser recomputada.

Análisis de las Expresiones Disponibles #3

Función $kill_{AE}$

$$kill_{AE} : Blocks_* \rightarrow \wp(AExp_*)$$

- $kill_{AE}([x := a]^l) = \{a' \in AExp_* \mid x \in FV(a')\}$
- $kill_{AE}([skip]^l) = \emptyset$
- $kill_{AE}([b]^l) = \emptyset$

– Un bloque *mata* (kill) una expresión si cualquiera de las variables usadas en la expresión son modificadas en el bloque

– Los bloques de **test** y **skip** no matan ninguna expresión y las **asignaciones** matan cualquier expresión que use la variable que aparece en la parte izquierda de la asignación.

Análisis de las Expresiones Disponibles #4

Función gen_{AE}

$$gen_{AE} : Blocks_* \rightarrow \wp(AExp_*)$$

- $gen_{AE}([x := a]^l) = \{a' \in AExp(a) | \ x \notin FV(a')\}$
- $gen_{AE}([skip]^l) = \emptyset$
- $gen_{AE}([b]^l) = AExp(b)$

- Una expresión generada (gen) es una expresión que es evaluada en el bloque y que no utiliza ninguna de las variables modificadas en el bloque.

Análisis de las Expresiones Disponibles #5

Ecuaciones de flujo de datos

- El análisis viene definido por las funciones AE_{entry} y AE_{exit} que aplican etiquetas a conjuntos de expresiones:

$$AE_{entry}, AE_{exit} : Lab_* \rightarrow \wp(AExp_*)$$

- $$AE_{entry}(l) = \begin{cases} \emptyset, & \text{si } l = init(S_*) \\ \cap\{AE_{exit}(l') \mid (l',l) \in flow(S_*)\}, & \text{en otro caso} \end{cases}$$

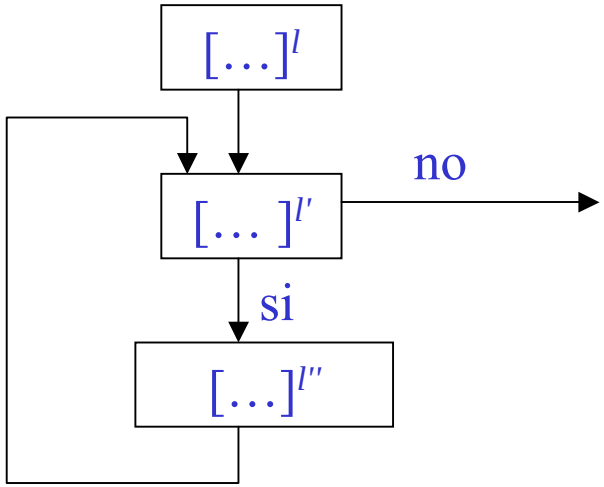
- $$AE_{exit}(l) = (AE_{entry}(l) \setminus kill_{AE}(B^l)) \cup gen_{AE}(B^l)$$
  $$\text{donde } B^l \in blocks(S_*)$$

Análisis de las Expresiones Disponibles #6

$[z := x + y]^{l}$; while $[true]^{l'}$ do $[skip]^{l''}$

- $AE_{entry}(l) = \varnothing$
- $AE_{entry}(l') = AE_{exit}(l) \cap AE_{exit}(l'')$
- $AE_{entry}(l'') = AE_{exit}(l')$

- $AE_{exit}(l) = AE_{entry}(l) \cup \{x + y\}$
- $AE_{exit}(l') = AE_{entry}(l')$
- $AE_{exit}(l'') = AE_{entry}(l'')$



Análisis de las Expresiones Disponibles #7

- Simplificando
  - $AE_{entry}(l) = \varnothing$
  - $AE_{entry}(l') = \{x + y\} \cap AE_{entry}(l')$
  - $AE_{entry}(l'') = AE_{entry}(l')$
- Estas ecuaciones tienen dos soluciones
  - $(\varnothing, \{x + y\}, \{x + y\})$
  - $(\varnothing, \varnothing, \varnothing)$
- La primera solución da más información: la expresión

  x + y está disponible cada vez que entramos en *l'*. Así que pedimos la **mayor solución** del conjunto de ecuaciones

Análisis de las Expresiones Disponibles #8

$[x := a + b]^1$; $[y := a * b]^2$;
while $[y > a + b]^3$ do ($[a := a + 1]^4$; $[x := a + b]^5$)

$AE_{entry}(1) = \varnothing$
$AE_{entry}(2) = AE_{exit}(1)$
$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$
$AE_{entry}(4) = AE_{exit}(3)$
$AE_{entry}(5) = AE_{exit}(4)$

l	$kill_{AE}(l)$	$gen_{AE}(l)$
1	$\varnothing$	{a + b}
2	$\varnothing$	{a * b}
3	$\varnothing$	{a + b}
4	{a + b, a*b, a + 1}	$\varnothing$
5	$\varnothing$	{a + b}

$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$
$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$
$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$
$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a*b, a + 1\}$
$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$

Análisis de las Expresiones Disponibles #8

$AE_{entry}(1) = \varnothing$
$AE_{entry}(2) = AE_{exit}(1)$
$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$
$AE_{entry}(4) = AE_{exit}(3)$
$AE_{entry}(5) = AE_{exit}(4)$

$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$
$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$
$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$
$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a*b, a + 1\}$
$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	$\varnothing$	$\varnothing$
2	$\varnothing$	$\varnothing$
3	$\varnothing$	$\varnothing$
4	$\varnothing$	$\varnothing$
5	$\varnothing$	$\varnothing$

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	$\varnothing$	{a+b}
2	$\varnothing$	{a*b}
3	$\varnothing$	{a+b}
4	$\varnothing$	$\varnothing$
5	$\varnothing$	{a+b}

Análisis de las Expresiones Disponibles #8

$AE_{entry}(1) = \varnothing$
$AE_{entry}(2) = AE_{exit}(1)$
$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$
$AE_{entry}(4) = AE_{exit}(3)$
$AE_{entry}(5) = AE_{exit}(4)$

$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$
$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$
$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$
$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a*b, a + 1\}$
$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	$\varnothing$	{a+b}
2	$\varnothing$	{a*b}
3	$\varnothing$	{a+b}
4	$\varnothing$	$\varnothing$
5	$\varnothing$	{a+b}

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	$\varnothing$	{a+b}
2	{a+b}	{a*b}
3	{a+b}	{a+b}
4	{a+b}	$\varnothing$
5	$\varnothing$	{a+b}

Análisis de las Expresiones Disponibles #8

$AE_{entry}(1) = \varnothing$
$AE_{entry}(2) = AE_{exit}(1)$
$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$
$AE_{entry}(4) = AE_{exit}(3)$
$AE_{entry}(5) = AE_{exit}(4)$

$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$
$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$
$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$
$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a*b, a + 1\}$
$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	∅	{a+b}
2	{a+b}	{a*b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	∅	{a+b}
2	{a+b}	{a*b,a+b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

Análisis de las Expresiones Disponibles #8

$[x := a + b]^1; [y := a * b]^2 ;$
$\text{while } [y > a + b]^3 \text{ do } ([a := a + 1]^4 ; [x := a + b]^5)$

$AE_{entry}(1) = \varnothing$
$AE_{entry}(2) = AE_{exit}(1)$
$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$
$AE_{entry}(4) = AE_{exit}(3)$
$AE_{entry}(5) = AE_{exit}(4)$

$AE_{exit}(1) = AE_{entry}(1) \cup \{a + b\}$
$AE_{exit}(2) = AE_{entry}(2) \cup \{a * b\}$
$AE_{exit}(3) = AE_{entry}(3) \cup \{a + b\}$
$AE_{exit}(4) = AE_{entry}(4) \setminus \{a + b, a*b, a + 1\}$
$AE_{exit}(5) = AE_{entry}(5) \cup \{a + b\}$

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	∅	{a+b}
2	{a+b}	{a*b,a+b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

l	$AE_{entry}(l)$	$AE_{entry}(l)$
1	∅	{a+b}
2	{a+b}	{a*b,a+b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

Análisis del ámbito de las definiciones (RD)

- **Determina**:
  - Para cada punto del programa, qué asignaciones pueden haber sido hechas y no sobreescritas, cuando la ejecución del programa llega a ese punto a través de algún camino

  - La aplicación principal de RD es la construcción de enlaces directos entre los bloques que producen valores y bloques que los usan.

RD #2

Función $kill_{RD}$

$$kill_{RD} : Blocks_* \rightarrow \wp(Var_* \times Lab_*^?)$$

- $kill_{RD}([x := a]^l) = \{(x,?)\} \cup$
  $\{(x,l') | B^{l'}$ es una asignación a x en $S_*\}$
- $kill_{RD}([skip]^l) = \varnothing$
- $kill_{RD}([b]^l) = \varnothing$

– $kill_{RD}$ produce el conjunto de pares (var ,label) destruidas por el bloque.

– Una asignación [v:=a] es destruida si el bloque asigna un nuevo valor a la variable.

RD #3

Función gen_{RD}

$$gen_{RD} : Blocks_* \rightarrow \wp(Var_* \times Lab_*^?)$$

- $gen_{RD}([x := a]^l) = \{(x,l)\}$
- $gen_{RD}([skip]^l) = \emptyset$
- $gen_{RD}([b]^l) = \emptyset$

– gen_{RD} produce el conjunto de pares (var ,label) generadas por el bloque.
– Sólo las asignaciones generan definiciones.

RD #5

Ecuaciones de flujo de datos

- El análisis viene definido por las funciones RD_{entry} y RD_{exit} que aplican etiquetas a conjuntos de pares (var,label):

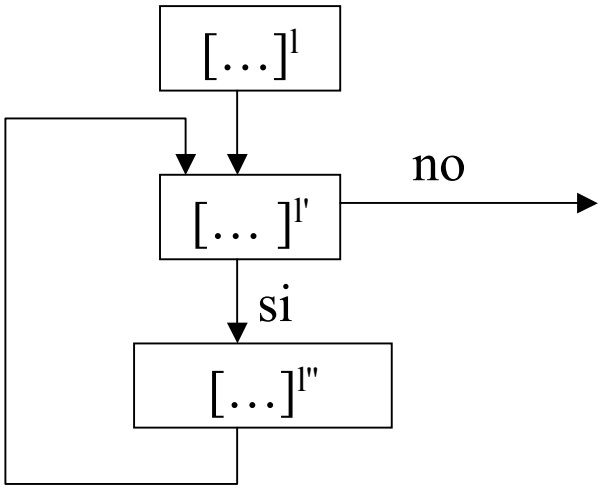
$$RD_{entry}, RD_{exit} : Lab_* \rightarrow \wp(Var_* \times Lab_*^?)$$

- $RD_{entry}(l) = \begin{cases} \{(x,?) \mid x \in FV(S_*)\}\ ,\ \text{si } l = init(S_*) \\ \bigcup \{RD_{exit}(l') \mid (l',l) \in flow(S_*)\}, \text{ en otro caso} \end{cases}$
- $RD_{exit}(l) = (RD_{entry}(l) \setminus kill_{RD}(B^l)) \cup gen_{RD}(B^l)$

donde $B^l \in blocks(S_*)$

RD #5

$[z := x + y]^{l}$; while $[true]^{l'}$ do $[skip]^{l''}$



- $RD_{entry}(l) = \{(x,?), (y,?), (z,?)\}$
- $RD_{entry}(l') = RD_{exit}(l) \cup RD_{exit}(l'')$
- $RD_{entry}(l'') = RD_{exit}(l')$

- $RD_{exit}(l) = (RD_{entry}(l) \setminus \{(z,?)\}) \cup \{(z,l)\}$
- $RD_{exit}(l') = RD_{entry}(l')$
- $RD_{exit}(l'') = RD_{entry}(l'')$

RD #6

- Simplificando

  $RD_{entry}(l') = \{(x,?), (y,?), (z,l)\} \cup RD_{entry}(l')$

- Estas ecuaciones tienen muchas soluciones

  S es solución sii $\{(x,?), (y,?), (z,l)\} \subseteq S$

- como l' no genera nuevas soluciones, la solución más precisa es $\{(x,?), (y,?), (z,l)\}$ – nos interesa la menor solución de la ecuación

RD #7

$[x := 5]^1; [y := 1]^{2;}$
$\text{while } [x > 1]^3 \text{ do } ([y := x * y]^4 ; [x := x - 1]^5)$

$RD_{entry}(1) = \{(x,?), (y,?)\}$
$RD_{entry}(2) = RD_{exit}(1)$
$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$
$RD_{entry}(4) = RD_{exit}(3)$
$RD_{entry}(5) = RD_{exit}(4)$

l	$kill_{AE}(l)$	$gen_{AE}(l)$
1	{(x,?), (x,1), (x,5)}	{(x,1)}
2	{(y,?), (y,2), (y,4)}	{(y,2)}
3	∅	∅
4	{(y,?), (y,2), (y,4)}	{(y,4)}
5	{(x,?), (x,1), (x,5)}	{(x,5)}

$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x,?), (x,1),(x,5)\}) \cup \{(x,l)\}$
$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y,?), (y,2),(y,4)\}) \cup \{(y,2)\}$
$RD_{exit}(3) = RD_{entry}(3)$
$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y,?), (y,2),(y,4)\}) \cup \{(y,4)\}$
$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x,?), (x,1),(x,5)\}) \cup \{(x,5)\}$

RD #8

$[x := 5]^1; [y := 1]^2;$
$\text{while } [x > 1]^3 \text{ do } ([y := x * y]^4 ; [x := x - 1]^5)$

$RD_{entry}(1) = \{(x,?), (y,?)\}$
$RD_{entry}(2) = RD_{exit}(1)$
$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$
$RD_{entry}(4) = RD_{exit}(3)$
$RD_{entry}(5) = RD_{exit}(4)$

l	$RD_{entry}(l)$	$RD_{exit}(l)$
1	{(x,?), (y,?)}	{(x,1), (y,?)}
2	{(x,1), (y,?)}	{(x,1), (y,2)}
3	{(x,1), (y,2), (y,4), (x,5)}	{(x,1), (y,2), (y,4), (x,5)}
4	{(x,1), (y,2), (y,4), (x,5)}	{(x,1), (y,4), (x,5)}
5	{(x,1), (y,4), (x,5)}	{(y,4), (x,5)}

$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x,?), (x,1),(x,5)\}) \cup \{(x,l)\}$
$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y,?), (y,2),(y,4)\}) \cup \{(y,2)\}$
$RD_{exit}(3) = RD_{entry}(3)$
$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y,?), (y,2),(y,4)\}) \cup \{(y,4)\}$
$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x,?), (x,1),(x,5)\}) \cup \{(x,5)\}$

Análisis de las Expresiones muy Ocupadas (VBE)

- **Determina**:
  - Una expresión **exp** está muy ocupada a la salida de una etiqueta ***l*** si, cualquier camino que parte de ***l*** utiliza **exp** antes de que cualquiera de las variables que contiene **exp** sean redefinidas.
  - El objetivo del Análisis de las Expresiones muy Ocupadas es determinar para cada punto del programa, qué expresiones están muy ocupadas.
  - Un posible uso es evaluar la expresión en el bloque y guardar su valor para un uso posterior.

VBE #2

if $[a > b]^1$
then $([x := b - a]^2; [y := a - b]^3)$
else $([y := b - a]^4; [x := a - b]^5)$

- Las expresiones a – b y b – a están ambas muy ocupadas al comienzo de la selección [1]; pueden guardarse en ese punto produciendo un ahorro de espacio en el tamaño del código generado por este programa.

VBE #3

Función $kill_{VB}$

$kill_{VB} : Blocks_* \rightarrow \wp(AExp_*)$

- $kill_{VB}([x := a]^l) = \{a' \in AExp_* \mid x \in FV(a')\}$
- $kill_{VB}([skip]^l) = \emptyset$
- $kill_{VB}([b]^l) = \emptyset$

– Se define igual que $Kill_{AE}$ para el análisis de las expresiones disponibles.

– Un bloque mata (kill) una expresión si cualquiera de las variables usadas en la expresión es modificada por el bloque.

VBE #4

Función gen_{VB}

$gen_{VB} : Blocks_* \rightarrow \wp(AExp_*)$

- $gen_{VB}([x := a]^l) = AExp(a)$
- $gen_{VB}([skip]^l) = \varnothing$
- $gen_{VB}([b]^l) = AExp(b)$

– Todas las expresiones que aparecen en un bloque están muy ocupadas a la entrada del bloque (al contrario de lo que ocurría para el Análisis de las Expresiones disponibles).

VBE #5

Ecuaciones de flujo de datos

- El análisis viene definido por las funciones VB_{entry} y VB_{exit} que aplican etiquetas a conjuntos de expresiones

$$VB_{entry}, VB_{exit}: Lab_* \rightarrow \wp(AExp_*)$$

- $$VB_{exit}(l) = \begin{cases} \emptyset, & \text{si } l = final(S_*) \\ \bigcap \{VB_{entry}(l') \mid (l',l) \in flow^R(S_*)\}, & \text{en otro caso} \end{cases}$$

- $$VB_{entry}(l) = (VB_{exit}(l) \setminus kill_{VB}(B^l)) \cup gen_{VB}(B^l)$$
  donde $B^l \in blocks(S_*)$

VBE #6

– $VB_{exit}(l) = \begin{cases} \varnothing, & \text{si } l = final(S_*) \\ \cap\{VB_{entry}(l') \mid (l',l) \in flow^R(S_*)\}, & \text{en otro caso} \end{cases}$

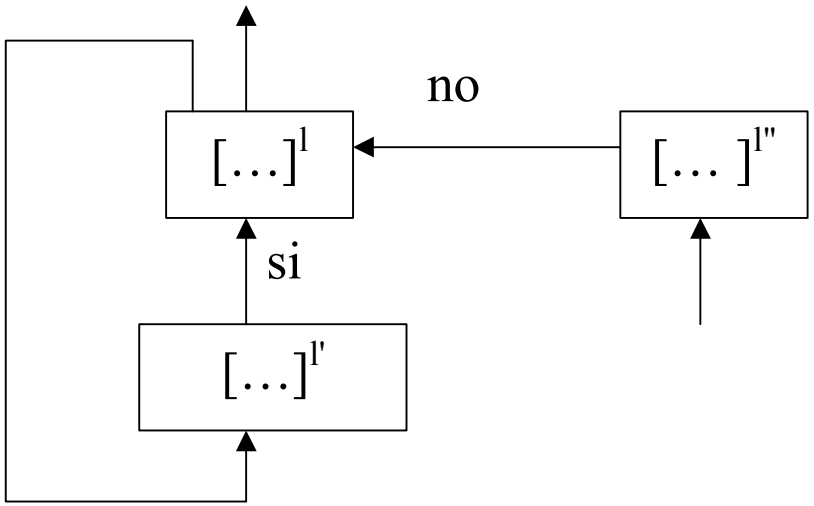
– El análisis VBE es un análisis hacia atrás (inverso)

– Las funciones propagan información contra el flujo del programa:

- una expresión está muy ocupada a la salida de un bloque si está muy ocupada a la entrada de cada bloque que lo sigue.
- ninguna expresión está muy ocupada a la salida del bloque final.

VBE #7

(while $[x > 1]^{l}$ do $[skip]^{l'}$); $[x := x + 1]^{l''}$



- $VB_{entry}(l) = VB_{exit}(l)$
- $VB_{entry}(l') = VB_{exit}(l')$
- $VB_{entry}(l'') = \{x + 1\}$

- $VB_{exit}(l) = VB_{entry}(l') \cap VB_{entry}(l'')$
- $VB_{exit}(l') = VB_{entry}(l)$
- $VB_{exit}(l'') = \emptyset$

VBE #8

- Simplificando
  - $VB_{exit}(l) = VB_{exit}(l) \cap \{x + 1\}$

- Esta ecuación tiene dos soluciones
  - $\varnothing$
  - $\{x + 1\}$

- La solución que da más información es $\{x + 1\}$ – nos interesa la mayor solución de la ecuación

VBE #9

if $[a > b]^1$
then $([x := b - a]^2; [y := a - b]^3)$
else $([y := b - a]^4; [x := a - b]^5)$

l	$kill_{VB}(l)$	$gen_{VB}(l)$
1	$\varnothing$	$\varnothing$
2	$\varnothing$	$\{b - a\}$
3	$\varnothing$	$\{a - b\}$
4	$\varnothing$	$\{b - a\}$
5	$\varnothing$	$\{a - b\}$

$VB_{entry}(1) = VB_{exit}(1)$
$VB_{entry}(2) = VB_{exit}(2) \cup \{b - a\}$
$VB_{entry}(3) = \{a - b\}$
$VB_{entry}(4) = VB_{exit}(4) \cup \{b - a\}$
$VB_{entry}(5) = \{a - b\}$

$VB_{exit}(1) = VB_{entry}(2) \cap VB_{entry}(4)$
$VB_{exit}(2) = VB_{entry}(3)$
$VB_{exit}(3) = \varnothing$
$VB_{exit}(4) = VB_{entry}(5)$
$VB_{entry}(5) = \varnothing$

VBE #10

if $[a > b]^1$
then $([x := b - a]^2; [y := a - b]^3)$
else $([y := b - a]^4; [x := a - b]^5)$

l	$VB_{entry}(l)$	$VB_{exit}(l)$
1	{a – b, b – a}	{a – b, b – a}
2	{a – b, b – a}	{a – b}
3	{a – b}	∅
4	{a – b, b – a}	{a – b}
5	{a – b}	∅

$VB_{entry}(1) = VB_{exit}(1)$
$VB_{entry}(2) = VB_{exit}(2) \cup \{b - a\}$
$VB_{entry}(3) = \{a - b\}$
$VB_{entry}(4) = VB_{exit}(4) \cup \{b - a\}$
$VB_{entry}(5) = \{a - b\}$

$VB_{exit}(1) = VB_{entry}(2) \cap VB_{entry}(4)$
$VB_{exit}(2) = VB_{entry}(3)$
$VB_{exit}(3) = \emptyset$
$VB_{exit}(4) = VB_{entry}(5)$
$VB_{entry}(5) = \emptyset$

Análisis de las Variables Vivas (LV)

- Una variable está viva a la salida de una etiqueta si hay un camino que parte de la etiqueta en la que la variable es usada y no redefinida.
- El análisis de las variables vivas determina para cada punto del programa, qué variables pueden estar vivas a la salida de dicho punto.
- Este análisis podría usarse como base para la Eliminación del Código Muerto (Dead Code Elimination). Si una variable no está viva a la salida de una etiqueta entonces, si el bloque elemental es una asignación a la variable, el bloque elemental puede eliminarse.

LV #2

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$
$(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y*y]^6)\ [x := z]^7$

- La variable x no está viva a la salida de la etiqueta 1; la primera asignación del programa es redundante.
- Tanto x como y están vivas a la salida de la etiqueta 3.

LV #3

Función $kill_{LV}$

$$kill_{LV} : Blocks_* \rightarrow \wp(Var_*)$$

- $kill_{LV}([x := a]^l) = \{x\}$
- $kill_{LV}([skip]^l) = \emptyset$
- $kill_{LV}([b]^l) = \emptyset$

– La variable que aparece a la izquierda de una asignación es destruida por la asignación;

– Los tests y skip no destruyen variables.

LV #4

Función gen_{LV}

$$gen_{LV} : Blocks_* \rightarrow \wp(Var_*)$$

- $gen_{LV}([x := a]^l) = FV(a)$
- $gen_{LV}([skip]^l) = \varnothing$
- $gen_{LV}([b]^l) = FV(b)$

– gen_{LV} produce el conjunto de variables que aparecen en el bloque

LV #5

Ecuaciones de flujo de datos

$$LV_{entry}, LV_{exit}: Lab_* \rightarrow \wp(Var_*)$$

– El análisis viene dado por las funciones LV_{entry} y LV_{exit} que asocian etiquetas y conjuntos de variables:

$$LV_{exit}(l) = \begin{cases} \emptyset, & \text{si } l = final(S_*) \\ \cup\{LV_{entry}(l') \mid (l',l) \in flow^R(S_*)\}, & \text{en otro caso} \end{cases}$$

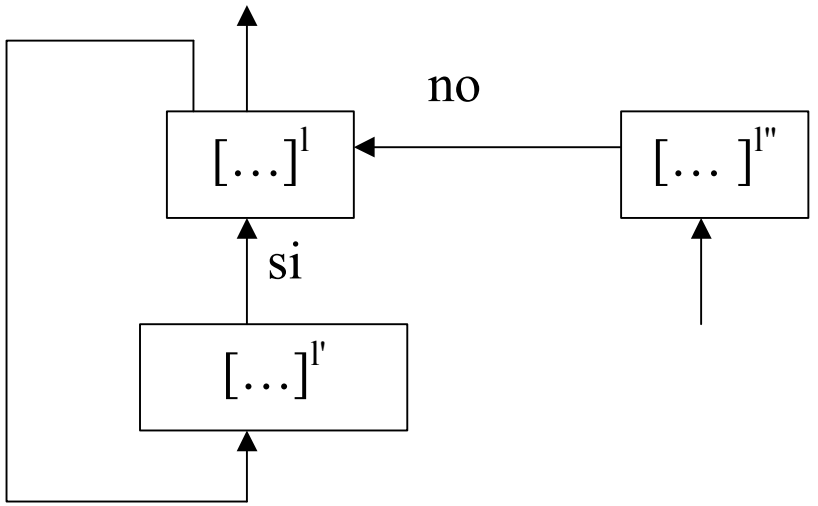
$$LV_{entry}(l) = (LV_{exit}(l) \setminus kill_{LV}(B^l)) \cup gen_{LV}(B^l)$$
$$\text{donde } B^l \in blocks(S_*)$$

LV #6

- $LV_{exit}(l) = \begin{cases} \varnothing, & \text{si } l = final(S_*) \\ \cup\{LV_{entry}(l') \mid (l',l) \in flow^R(S_*)\}, & \text{en otro caso} \end{cases}$
- Suponemos que S_* es etiqueta-consistente y que tiene salidas aisladas.
- La ecuación para $LV_{exit}(l)$ incluye una variable en el conjunto de las variables vivas (a la salida de l) si está viva a la entrada de cualquiera de los bloques que siguen a l; si no ocurre esto es $\varnothing$.
- El análisis es un análisis hacia atrás (inverso).

LV #7

(while [x > 1]^{l} do [skip]$^{l'}$); [x := x + 1]$^{l''}$



- $LV_{entry}(l) = LV_{exit}(l) \cup \{x\}$
- $LV_{entry}(l') = LV_{exit}(l')$
- $LV_{entry}(l'') = \{x\}$

- $LV_{exit}(l) = LV_{entry}(l') \cup LV_{entry}(l'')$
- $LV_{exit}(l') = LV_{entry}(l)$
- $LV_{exit}(l'') = \varnothing$

LV #8

(while [x > 1]^{l} do [skip]$^{l'}$); [x := x + 1]$^{l''}$

- Simplificando

$$LV_{exit}(l) = LV_{exit}(l) \cup \{x\}$$

  - Cualquier superconjunto de {x} es una solución.
  - Las optimizaciones basadas en este análisis se basan en las variables muertas (dead) – cuanto menor es el conjunto de las variables vivas, es posible hacer más optimizaciones.
  - Por lo tanto estamos interesados en la menor solución {x} de la ecuación.

LV #9

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$
$(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y*y]^6)\ [x := z]^7$

- $LV_{entry}(1) = LV_{exit}(1) \setminus \{x\}$
- $LV_{entry}(2) = LV_{exit}(2) \setminus \{y\}$
- $LV_{entry}(3) = LV_{exit}(3) \setminus \{x\}$
- $LV_{entry}(4) = LV_{exit}(4) \setminus \{x, y\}$
- $LV_{entry}(5) = (LV_{exit}(5) \setminus \{z\}) \cup \{y\}$
- $LV_{entry}(6) = (LV_{exit}(6) \setminus \{z\}) \cup \{y\}$
- $LV_{entry}(7) = \{z\}$

l	$kill_{LV}(l)$	$gen_{LV}(l)$
1	{x}	∅
2	{y}	∅
3	{x}	∅
4	∅	{x, y}
5	{z}	{y}
6	{z}	{y}
7	{x}	{z}

LV #10

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$
$(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y*y]^6)\ [x := z]^7$

- $LV_{exit}(1) = LV_{entry}(2)$
- $LV_{exit}(2) = LV_{entry}(3)$
- $LV_{exit}(3) = LV_{entry}(4)$
- $LV_{exit}(4) = LV_{entry}(5) \cup LV_{entry}(6)$
- $LV_{exit}(5) = LV_{exit}(7)$
- $LV_{exit}(6) = LV_{exit}(7)$
- $LV_{exit}(7) = \varnothing$

l	$kill_{LV}(l)$	$gen_{LV}(l)$
1	{x}	∅
2	{y}	∅
3	{x}	∅
4	∅	{x, y}
5	{z}	{y}
6	{z}	{y}
7	{x}	{z}

LV #11

$[x := 2]^1; [y := 4]^2; [x := 1]^3;$

$(\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y*y]^6)\ [x := z]^7$

– Usando la Iteración Caótica para calcular la solución obtenemos:

l	$LV_{entry}(l)$	$LV_{exit}(l)$
1	$\varnothing$	$\varnothing$
2	$\varnothing$	{y}
3	{y}	{x, y}
4	{x, y}	{y}
5	{y}	{z}
6	{y}	{z}
7	{z}	$\varnothing$

Información derivada del AFD

- Es conveniente enlazar directamente las etiquetas de las instrucciones que producen valores con las etiquetas de las instrucciones que las usan.

$[x := 0]^1; [x :=3]^2;$
$(\text{if } [z = x]^3 \text{ then } [z := 0]^4 \text{ else } [z := x]^5);$
$[y:=x]^6; [x := y + z]^7$

- Los enlaces que, para cada uso de una variable, asocian todas las asignaciones que alcanzan ese uso se llaman Cadenas de uso-definición o ud-cadenas.
- 3 --> 2

Información derivada del AFD #2

$[x := 0]^1; [x :=3]^2;$
$(\text{if } [z = x]^3 \text{ then } [z := 0]^4 \text{ else } [z := x]^5);$
$[y:=x]^6; [x := y + z]^7$

- Los enlaces que cada asignación de una variable la asocian con todos los puntos en los que se usa dicha variable se llaman cadenas de definición-uso o du-cadenas.
- 2-->{3,5,6}

Información derivada del AFD #3

$$\text{clear}(x,l,l') = \begin{cases} \exists l_1, \ldots, l_n : (l_1 = l) \wedge (l_n = l') \wedge (n > 0) \wedge \\ (\forall i \in \{1,\ldots,n-1\} : (l_i, l_{i+1}) \in \text{flow}(S_*)) \wedge \\ (\forall i \in \{1,\ldots,n-1\} : \neg\text{def}(x,l_i)) \wedge \text{use}(x,l_n) \end{cases}$$

- $l_1, \ldots, l_n$ es una cadena limpia para x si
  - ninguno de los bloques etiquetados con $l_1, \ldots, l_n$ asigna a x un valor y
  - l_n la usa x.

Información derivada del AFD #4

$$use(x,l) = (\exists B : [B]^l \in blocks(S_*) \wedge x \in gen_{LV}([B]^l))$$

$$def(x,l) = (\exists B : [B]^l \in blocks(S_*) \wedge x \in kill_{LV}([B]^l))$$

Información derivada del AFD #5

ud: $\mathbf{Var_* \times Lab_* \rightarrow \wp(Lab_*)}$

$$ud(x,l') = \begin{cases} \{l \mid def(x,l) \wedge \exists\, l'' : (l,l'') \in flow(S_*) \wedge clear(x,l'',l')\} \\ \cup\ \{? \mid clear(x,init(S_*),l')\} \end{cases}$$

- ud(x,l') devuelve las etiquetas l donde la variable x, usada en l, podría haber obtenido su valor;
  - puede ser en una etiqueta l o
  - puede no estar inicializada como indica la ocurrencia de ?.

Información derivada del AFD #6

$$\mathbf{du: Var_* \times Lab_* \rightarrow \wp(Lab_*)}$$

$$du(x,l) = \begin{cases} \{l' \mid def(x,l) \wedge \exists\, l'' : (l,l'') \in flow(S_*) \wedge clear(x,l'',l')\} & \text{si } l \neq ? \\ \{l' \mid clear(x,init(S_*),l')\} & \text{si } l = ? \end{cases}$$

- du(x,l) devuelve las etiquetas en las que el valor asignado a x en l podría haber sido usado; distinguimos entre el caso en que x obtiene su valor en el programa y el caso en que no está inicializada.
- Además se tiene que:

$$du(x,l) = \{l' \mid l \in ud(x,l')\}$$

Información derivada del AFD #7

$[x := 0]^1$; $[x := 3]^2$;
(if $[z = x]^3$ then $[z := 0]^4$ else $[z := x]^5$);
$[y := x]^6$; $[x := y + z]^7$

ud(x,l)	x	y	z
1	∅	∅	∅
2	∅	∅	∅
3	{2}	∅	{?}
4	∅	∅	∅
5	{2}	∅	∅
6	{2}	∅	∅
7	∅	{6}	{4, 5}

Información derivada del AFD #8

$[x := 0]^1; [x :=3]^2;$
$(\text{if } [z = x]^3 \text{ then } [z := 0]^4 \text{ else } [z := x]^5);$
$[y:=x]^6; [x := y + z]^7$

du (x,l)	x	y	z
1	∅	∅	∅
2	{3, 5, 6}	∅	∅
3	∅	∅	∅
4	∅	∅	{7}
5	∅	∅	{7}
6	∅	{7}	∅
7	∅	∅	∅
?	∅	∅	{3}

Información derivada del AFD #9

- Aplicaciones:
  - Eliminación del Código Muerto (Dead Code Elimination);
    - para el programa del ejemplo anterior podemos eliminar el bloque etiquetado con 1 ya que no habrá ningún uso del valor asignado a x antes de que sea reasignado en el siguiente bloque.
  - Movimiento del Código (Code Motion);
    - en el programa del ejemplo, el bloque 6 puede moverse justo antes del condicional ya que sólo usa variables asignadas en bloques anteriores y el condicional no usa la variable asignada en el bloque 6.

Información derivada del AFD #10

- Las definiciones de las ud- y du-cadenas no son constructivas.
- Para definir de forma constructiva las ud-cadenas podemos usar RD_{entry}

$$UD : Var_* \times Lab_* \rightarrow \wp(Lab_*)$$

$$UD(x,l) = \begin{cases} \{l' \mid (x,l') \in RD_{entry}(l)\}, \text{ si } x \in gen_{LV}(B^l) \\ \emptyset, \text{ en otro caso} \end{cases}$$

- Similarmente, podemos definir la función $DU : Var_* \times Lab_* \rightarrow \wp(Lab_*)$ para las du-cadenas basándonos en funciones dadas anteriormente.

Semántica Operacional Estructural (SOS)

- Estados
  - $\sigma \in$ State = Var $\rightarrow$ Z
- Configuración
  - un par consistente en una instrucción y un estado o es un estado <S,σ>;
  - o simplemente un estado σ.
- Transición
  - <S,σ> $\rightarrow$ σ'
    - expresa cómo cambia la configuración con un paso de la computación (semántica de paso pequeño)

Semántica Operacional Estructural (SOS) #2

- En configuración <S,σ> podrían ocurrir una de las dos siguientes opciones:
  - La ejecución termina después de un paso y sabemos que da el resultado σ', o
  - La ejecución no termina después de un paso y lo representamos mediante una nueva configuración <S',σ'> donde S' es el resto del programa y σ' el estado actualizado.

Semántica Operacional Estructural (SOS) #3

- Expresiones aritméticas y booleanas
  - A : AExp → (State → Z)
  - B : BExp → (State → Z)

$$\frac{A : AExp \rightarrow (State \rightarrow Z)}{\begin{array}{c} A[[x]]\sigma = \sigma(x) \\ A[[n]]\ \sigma = N[[x]] \\ A[[a_1\ op_a\ a_2]]\ \sigma = A[[a_1]]\sigma\ op_a\ A[[a_2]]\sigma \end{array}}$$

$$\frac{B : AExp \rightarrow (State \rightarrow Z)}{\begin{array}{c} B[[not\ b]]\sigma = \neg\ B[[b]]\sigma \\ B[[a_1\ op_b\ a_2]]\ \sigma = B[[a_1]]\sigma\ op_b\ B[[a_2]]\sigma \\ B[[a_1\ op_r\ a_2]]\ \sigma = A[[a_1]]\sigma\ op_r\ A[[a_2]]\sigma \end{array}}$$

Semántica de las expresiones en WHILE

Semántica Operacional Estructural (SOS) #4

[ass] $\langle [x := a]^1, \sigma \rangle \rightarrow \sigma[x \rightarrow A[[a]]\sigma$

[skip] $\langle [skip]^1, \sigma \rangle \rightarrow \sigma$

- La cláusula [ass] especifica que la asignación $x := a$ se ejecuta en un paso;
  - $\sigma[x \rightarrow A[[a]]\sigma]$ es el estado que es como σ salvo que x es aplicado a $A[[a]]\sigma$, es decir, el valor al que a se evaluará en el estado σ. Formalmente,

$$(\sigma[x \rightarrow A[[a]]\sigma])(y) = \begin{cases} A[[a]]\sigma, & \text{si } x = y \\ \sigma(y), & \text{en otro caso} \end{cases}$$

Semántica Operacional Estructural (SOS) #5

$$[seq_1] \quad \frac{< S_1, \sigma > \longrightarrow < S'_1, \sigma' >}{< S_1; S_2, \sigma > \longrightarrow < S'_1; S_2, \sigma' >}$$

$$[seq_2] \quad \frac{< S_1, \sigma > \longrightarrow \sigma'}{< S_1; S_2, \sigma > \longrightarrow < S_2, \sigma' >}$$

- El primer paso al ejecutar S_1; S_2 es el primer paso de ejecutar S_1.
  - $[seq_2]$ Sólo se necesita un paso para terminar S_1. Pasamos a $<S_2,\sigma'>$ y estamos listos para ejecutar S_2.
  - $[seq_1]$ S_1 no termina en un paso, pasamos a otra configuración $<S'_1,\sigma'>$; queda por ejecutar el resto de S_1 y todo S_2. La nueva configuración es $<S'_1;S_2,\sigma'>$.

Semántica Operacional Estructural (SOS) #6

$[if_1]$ $\quad \langle \text{if } [b]^l \text{ then } S_1 \text{ else } S_2 , \sigma \rangle \rightarrow \langle S_1, \sigma \rangle,$
if $B[[b]]\sigma = true$

$[if_2]$ $\quad \langle \text{if } [b]^l \text{ then } S_1 \text{ else } S_2 , \sigma \rangle \rightarrow \langle S_2, \sigma \rangle,$
if $B[[b]]\sigma = false$

- el primer paso de la computación seleccionará la rama adecuada basándose en el valor actual de las expresiones booleanas.

Semántica Operacional Estructural (SOS) #7

$[wh_1]$ $\langle \text{while } [b^l] \text{ do } S , \sigma \rangle \rightarrow \langle S ; \text{while } [b^l] \text{ do } S , \sigma \rangle$
if $B[[b]]\sigma = true$

$[wh_2]$ $\langle \text{while } [b^l] \text{ do } S , \sigma \rangle \rightarrow \sigma$
if $B[[b]]\sigma = false$

- $[wh_1]$ si las expresiones booleanas se evalúan a true entonces el primer paso es desplegar el bucle.
- $[wh_2]$ la ejecución termina si la expresión booleana se evalúa a false.

Semántica Operacional Estructural (SOS) #8

- **Secuencias de Derivación**
  - Una secuencia de derivación para una instrucción S_1 y un estado σ_1 puede tener dos formas

    - (secuencia finita) corresponde a una computación que termina: $\langle S_1,\sigma_1\rangle, \ldots,\langle S_n,\sigma_n\rangle$, donde

      $\langle S_i,\sigma_i\rangle \rightarrow \langle S_{i+1},\sigma_{i+1}\rangle$ para $i = 1, \ldots, n-1$

      $\langle S_n,\sigma_n\rangle \rightarrow \sigma_{n+1}$;
    - (secuencia infinita) corresponde a una computación infinita: $\langle S_1,\sigma_1\rangle, \ldots,\langle S_i,\sigma_i\rangle, \ldots$, donde

      $\langle S_i,\sigma_i\rangle \rightarrow \langle S_{i+1},\sigma_{i+1}\rangle$ para todo $i \geq 1$; lo que.

Semántica Operacional Estructural (SOS) #9

- Ejemplo (factorial)

  suponemos que σ_{nms} aplica x con n, y con m y z con s.

$<[y:=x]^1 ; [z := 1]^2 ; \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{300}> \rightarrow$

$< [z := 1]^2 ; \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{330}> \rightarrow$

$< \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{331}> \rightarrow$

$< [z := z * y]^4 ; [y := y - 1]^5; \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6,$

$\sigma_{331}> \rightarrow$

$< [y := y - 1]^5; \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{333}> \rightarrow$

Semántica Operacional Estructural (SOS) #10

$< \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{323}> \rightarrow$

$< [z := z * y]^4 ; [y := y - 1]^5; \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{323}> \rightarrow$

$< [y := y - 1]^5; \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{326}> \rightarrow$

$< \text{while } [y > 1]^3 \text{ do } ([z := z * y]^4 ; [y := y - 1]^5); [y := 0]^6, \sigma_{316}> \rightarrow$

$< [y := 0]^6, \sigma_{316}> \rightarrow \sigma_{306}$

- las etiquetas no tienen efecto sobre la semántica. Nunca son inspeccionadas.

Bibliografía

F. Nielson, H. R. Nielson, C. Hankin,
Principles of Program Analysis, 1998, Springer