Métodos para la construcción de software fiable: Interpretación Abstracta

María del Mar Gallardo Melgarejo

Pedro Merino Gómez

Dpto. de Lenguajes y Ciencias de la Computación

Universidad de Málaga

(gallardo,pedro)@lcc.uma.es

Interpretación Abstracta: Objetivos

- Es una teoría general *basada en semánticas* para definir análisis estáticos de programas que, por construcción, son correctos.
- Objetivo: extraer información segura/correcta sobre el *comportamiento dinámico* de los programas sin necesidad de ejecutarlos para todas sus entradas; y hacer esto *automáticamente*
- El término de *interpretación abstracta* significa que al programa se le da una semántica distinta de la estándar.

Interpretación Abstracta: Idea Intuitiva

- Dado un programa P escrito en un lenguaje de programación L, una interpretación abstracta de P se obtiene realizando una ejecución abstracta.
- Una ejecución abstracta/no estándar consiste en: ejecutar instrucciones (o evaluar expresiones, satisfacer objetivos, etc) usando datos que pertenecen a dominios abstractos en lugar de los datos usados en las computaciones habituales
- El objetivo es: Deducir información sobre lo que ocurre en cualquier ejecución estándar a partir de lo obtenido en la ejecución abstracta.

IA: Terminación y aproximación

Ventajas de usar valores abstractos

- Computabilidad: es posible asegurar que los resultados del análisis se obtienen en tiempo finito
- Generalización: Los resultados obtenidos describen el el comportamiento del programa en algunas ejecuciones.

Desventajas

- La información obtenida es siempre una aproximación de la que realmente se produce en una ejecución estándar.

IA: Ejemplo: ejecución estándar

x,y,z:integer;
{y=2,z=-3}
x := y * z
{x=-6}

x,y,z:integer;
{y=-4,z=-3}
x := y * z
{x= 12}

x,y,z:integer;
{y=2,z=-3}
x := y + z
{x=-1}

x,y,z:integer;
{y=2,z=3}
x := y + z
{x=5}

IA: Ejemplo Regla de los signos

–Abstracción de datos:

AbsInteger = {0,+,-,?}

α:Integer $\longrightarrow$ AbsInteger

$\alpha(n) = + \text{ sii } n > 0$

$\alpha(n) = 0 \text{ sii } n = 0$

$\alpha(n) = - \text{ sii } n < 0$

IA: Ejemplo Regla de los signos

–Abstracción de operaciones:

$\alpha(*) = \otimes$

$\otimes$	**-**	**0**	**+**	**?**
-	+	0	-	?
0	0	0	0	0
+	-	0	+	?
?	?	0	?	?

$\alpha(+) = \oplus$

$\oplus$	**-**	**0**	**+**	**?**
-	-	-	?	?
0	-	0	+	?
+	?	+	+	?
?	?	?	?	?

IA: Ejemplo: Ejecución Abstracta

x,y,z:Absinteger;
{y= +, z= - }
x := y ⊗ z
{x = -}

x,y,z: Absinteger;
{y= -, z= - }
x := y ⊗ z
{x = -}

x,y,z: Absinteger;
{y = +, z = -}
x := y ⊕ z
{x = ?}

x,y,z: Absinteger;
{y = +,z = +}
x := y ⊕ z
{x = +}

IA: Ejemplo #2: Prueba del nueve

Este método usa los valores abstractos 0, 1, …, 8 para detectar errores en cálculos mentales.

A op B = C ??? (A,B,C números grandes)

Estrategia

A1:=A;B1:=B;C1:=C;
Mientras A1 ≥ 9 hacer
    A1 := sumadigitos(A1)
Mientras B1 ≥ 9 hacer
    B1 := sumadigitos(B1)
Aux := A1 op B1
Mientras Aux ≥ 9 hacer
    Aux := sumadigitos(Aux)

Mientras C1 ≥ 9 hacer
    C1 := sumadigitos(C1)

Aux ≠ C1 ⇒ A op B ≠ C

Aux = C1 ⇏ A op B = C

IA: Ejemplo #3

- Por ejemplo, considera el cálculo siguiente

  $123 * 457 + 76543 = ? = 132654$

  - $123 \rightarrow 6,$
  - $457 \rightarrow 16 \rightarrow 7$
  - $76543 \rightarrow 25 \rightarrow 7$
  - $6 * 7 = 42 \rightarrow 6,$
  - $6 + 7 \rightarrow 13 \rightarrow 4$
  - $132654 \rightarrow 21 \rightarrow 3$

  Como $3 \neq 4$ deducimos que el cálculo es incorrecto

IA: Ejemplo #4

– Corrección del método

$[a]_9 = a \bmod 9$ si $a = 9*a_c + a_r \Rightarrow [a]_9 = a_r$

$a* b = c \Rightarrow [a * b]_9 = [c]_9$ o de forma equivalente

$[a * b]_9 \neq [c]_9 \Rightarrow a* b \neq c$

$a + b = c \Rightarrow [a + b]_9 = [c]_9$ o de forma equivalente

$[a + b]_9 \neq [c]_9 \Rightarrow a + b \neq c$

IA: Ejemplo #4

– Corrección del método $[a * b]_9 = [[a]_9 * [b]_9]_9$ (prod)

$a = 9*a_c + a_r \qquad b = 9*b_c + b_r$

$a*b = 81*a_c*b_c + 9*a_c*b_r + 9*b_c*a_r + a_r*b_r$

$[a * b]_9 = [a_r * b_r]_9 = [[a]_9 * [b]_9]_9$

$[a + b]_9 = [[a]_9 + [b]_9]_9$ (sum)

$a = 9*a_c + a_r \qquad b = 9*b_c + b_r$

$a + b = 9*a_c + 9*b_c + a_r + b_r$

$[a + b]_9 = [a_r + b_r]_9 = [[a]_9 + [b]_9]_9$

IA: Ejemplo #4

– Corrección del método

$$[a]_9 = [\text{sumadigitos}(a)]_9$$

Por inducción sobre el número de dígitos de a.

1.- Si a es un dígito entonces se cumple trivialmente trivialmente.

IA: Ejemplo #4

– Corrección del método

$$[a]_9 = [\text{sumadigitos}(a)]_9$$

2.- Supongamos que se satisface para todos los números con menos de n dígitos, y veámoslo para un número a con n>1 dígitos

$a = 10^* a_1 + a_2$ donde a_1 tiene n-1 dígitos y a_1 tiene 1 dígito.

Usando (prod) y (sum) tenemos:

$[a]_9 = [10^* a_1 + a_2]_9 = [[10^* a_1]_9 + [a_2]_9]_9 =$

$= [[[10]_9 * [a_1]_9]_9 + [a_2]_9]_9 = [[a_1]_9 + [a_2]_9]_9 =$

Usando la hipótesis de inducción tenemos

$= [[\text{sumadigitos}(a_1)]_9 + [a_2]_9]_9 =$

Finalmente por (sum)

$= [\text{sumadigitos}(a_1) + a_2]_9 = [\text{sumadigitos}(a)]_9$

IA: Necesidad de aproximación

- Debido a la insolubilidad del problema de la parada (y de casi cualquier otra cuestión concerniente al comportamiento de los programas) ningún análisis que termina siempre puede ser, en general, exacto. Por lo tanto, tenemos tres alternativas:

IA: Necesidad de aproximación #2

- Considerar sistemas con un número finito de comportamientos finitos (por ejemplo, programas sin bucles) o propiedades decidibles (por ejemplo el chequeo de tipos en Pascal).
  - Desafortunadamente, muchos problemas interesantes no pueden expresarse de este modo.
- Pedir interactivamente ayuda en caso de duda.
  - la experiencia ha demostrado que los usuarios normalmente no puede deducir conclusiones útiles a partir de hechos producidos por una máquina. Por esto los sistemas de demostración interactiva son menos útiles de los que se esperaba.
- Aceptar información correcta pero aproximada

IA: Necesidad de aproximación #3

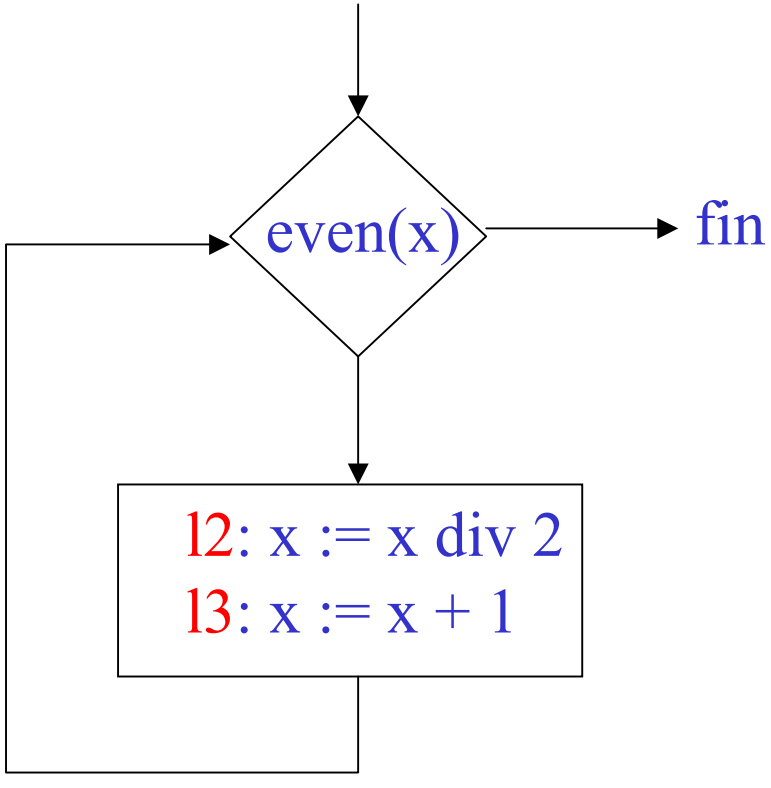
- Objetivo
  - encontrar descripciones seguras del comportamiento de un programa, produciendo respuestas que, aunque a veces son demasiado conservativas en relación con el comportamiento real del programa, nunca dan información incorrecta.
  - En un sentido formal, buscamos una relación $\sqsubseteq$ en lugar de la igualdad. El efecto es que para obtener computabilidad hace falta perder precisión.

IA: Corrección

- Un análisis aproximado es seguro si los resultados que da nunca son falsos. Se permite que sean imprecisos pero no incorrectos.
  - Por ejemplo, si una variable booleana J toma alguna vez el valor true, podemos decir que no sabemos qué valor toma pero no que es igual a false.
- Para definir seguridad es esencial comprender de forma precisa cómo se interpretan los valores abstractos en relación con las computaciones reales.

IA: Corrección / Imprecisión

```
l1:while even(x) do
      l2: x := x div 2
      l3: x := x + 1
   end;
l4:
```



IA: Corrección / Imprecisión

Semántica operacional concreta
(State , $\rightarrow$)

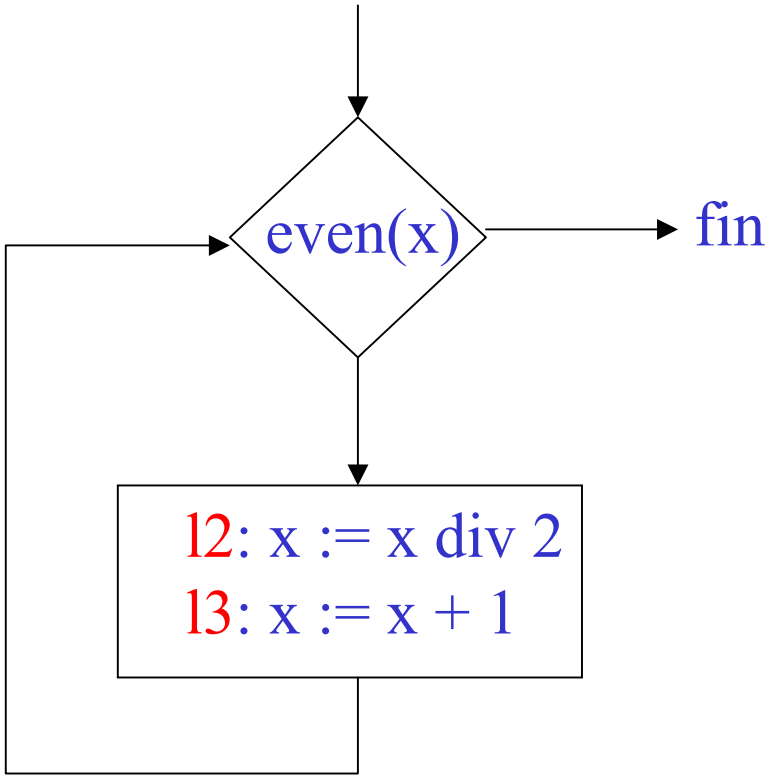
State = Label $\times$ Store
Store = Var $\rightarrow$ Value
$\rightarrow \subseteq$ State $\times$ State

```
l1:while even(x) do
      l2: x := x div 2
      l3: x := x + 1
   end;
l4:
```

Reglas de transición concretas

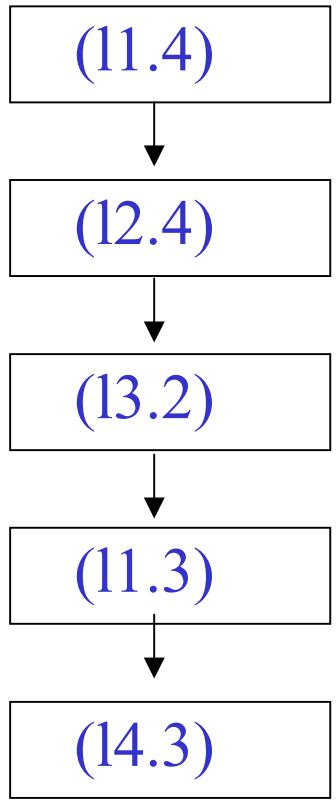
$(l1,2n) \rightarrow (l2,2n)$
$(l2,2n) \rightarrow (l3,n)$
$(l3,n) \rightarrow (l1,n + 1)$
$(l1,2n+1) \rightarrow (l4,2n + 1)$

IA: Corrección / Imprecisión



$(l1,2n) \rightarrow (l2,2n)$
$(l2,2n) \rightarrow (l3,n)$
$(l3,n) \rightarrow (l1,n + 1)$
$(l1,2n+1) \rightarrow (l4,2n + 1)$

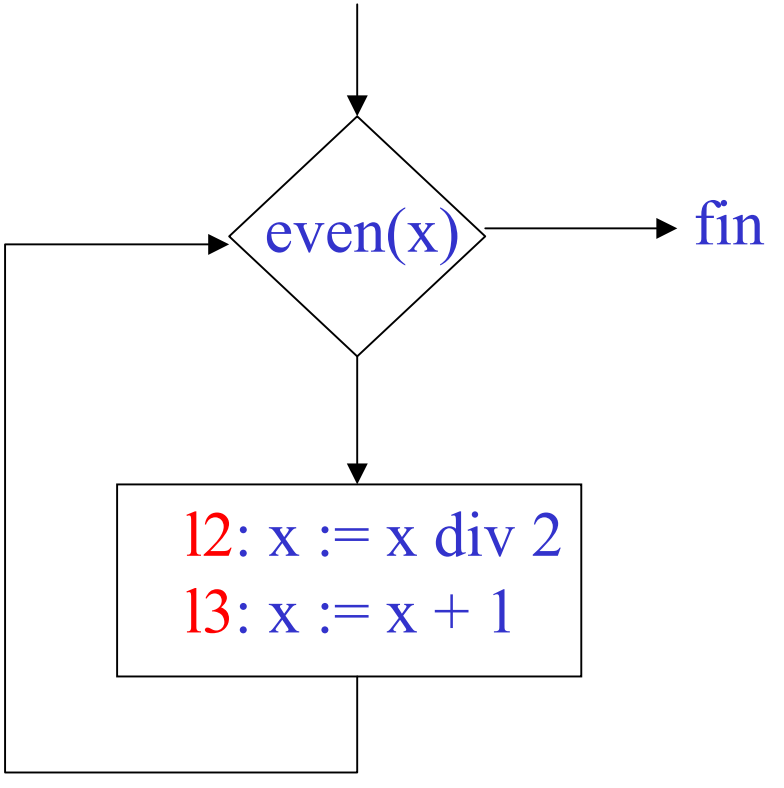
Una ejecución concreta



IA: Corrección / Imprecisión

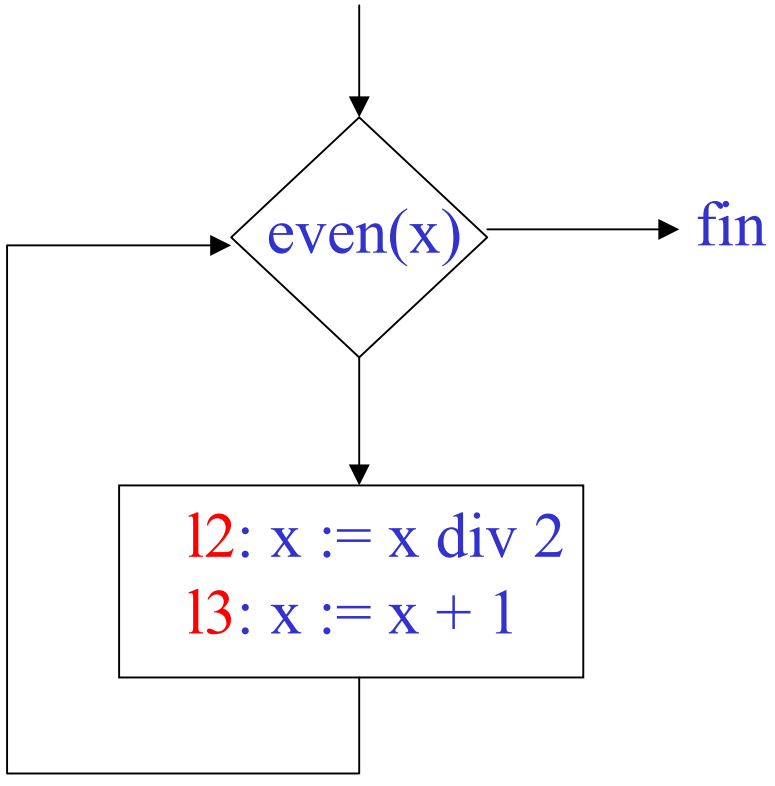
Semántica operacional abstracta
(AbState , $\rightarrow$)

AbState = Label $\times$ AbsStore
Store = Var $\rightarrow$ AbsValue
$\rightarrow \subseteq$ AbsState $\times$ AbsState



AbsValue = {e,o,?}
α: Value $\rightarrow$ AbsValue
$\alpha(2n) = e$
$\alpha(2n+1) = o$

IA: Corrección / Imprecisión



Reglas de transición abstractas

(l1,e) → (l2,e)
(l2,e) → (l3,e)
(l2,e) → (l3,o)
(l3,e) → (l1,o)
(l3,o) → (l1,e)
(l1,o) → (l4,o)

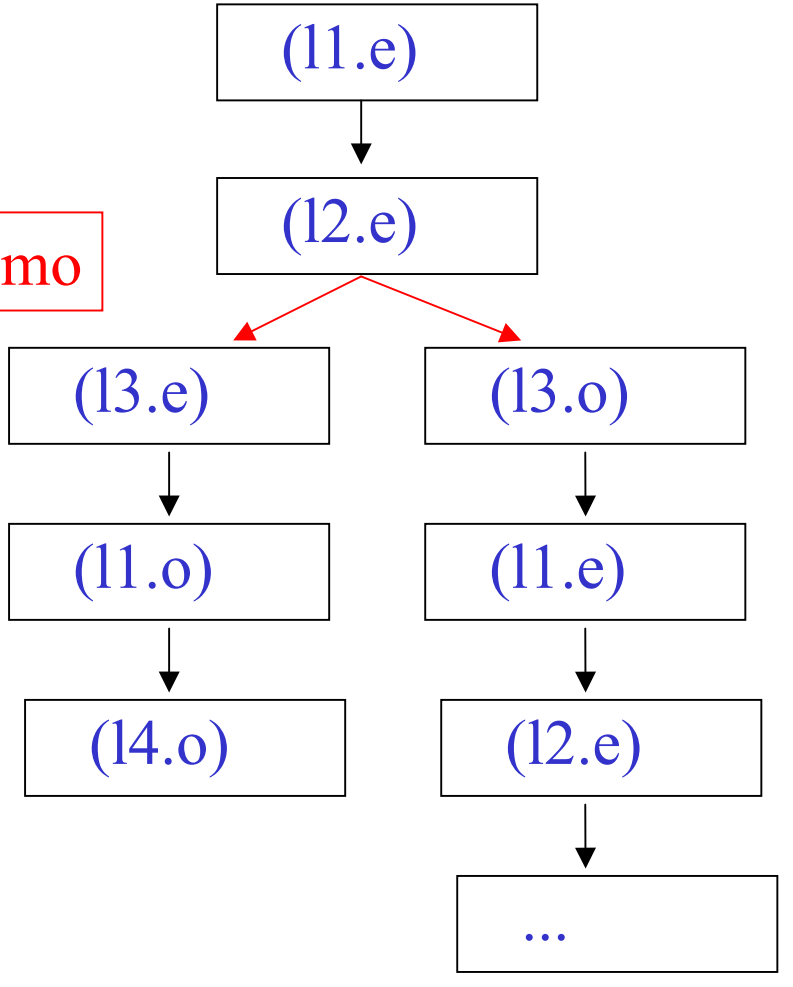
IA: Corrección / Imprecisión

Reglas de transición abstractas

$(l1,e) \rightarrow (l2,e)$
$(l2,e) \rightarrow (l3,e)$
$(l2,e) \rightarrow (l3,o)$
$(l3,e) \rightarrow (l1,o)$
$(l3,o) \rightarrow (l1,e)$
$(l1,o) \rightarrow (l4,o)$

Indeterminismo

Una ejecución abstracta



IA: Simulación

Noción de abstracción

$a \in$ AbsInteger es una abstracción de
$n \in$ Integer sii $\alpha(n) = a$

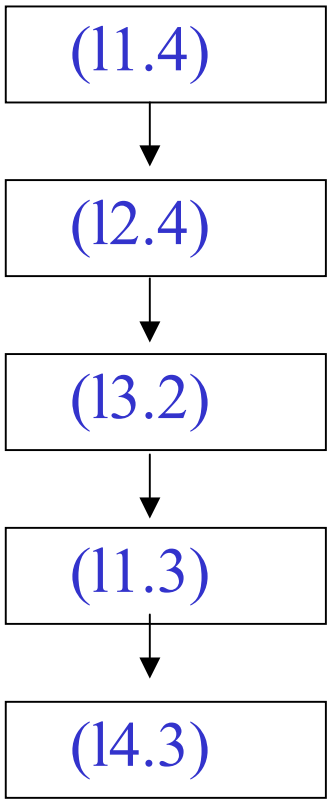
Simulación

(AbState , $\rightarrow$) es una simulación de (State , $\rightarrow$) sii

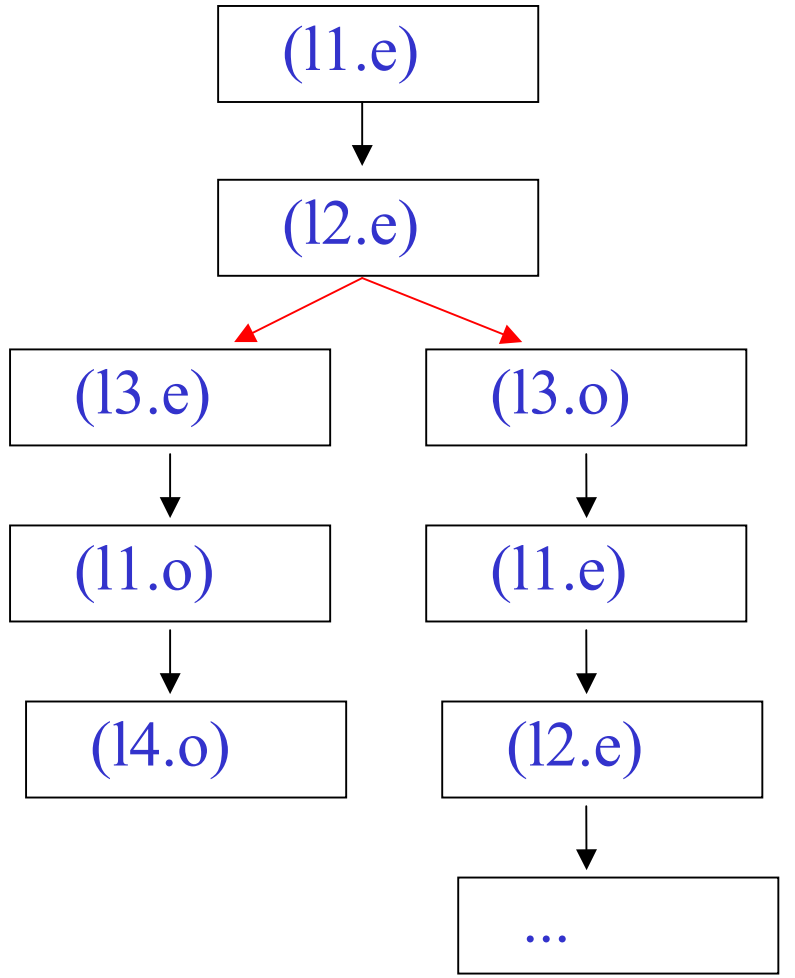
si $(l,n) \rightarrow (l',n')$ y $\alpha(n) = a$ entonces existe $a' \in$ AbsInteger tal que $(l,n) \rightarrow (l',n')$ y $\alpha(n') = a'$

IA: Simulación

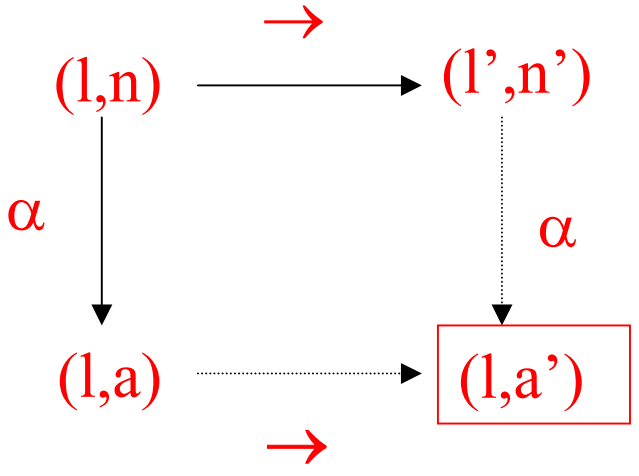
Una ejecución concreta

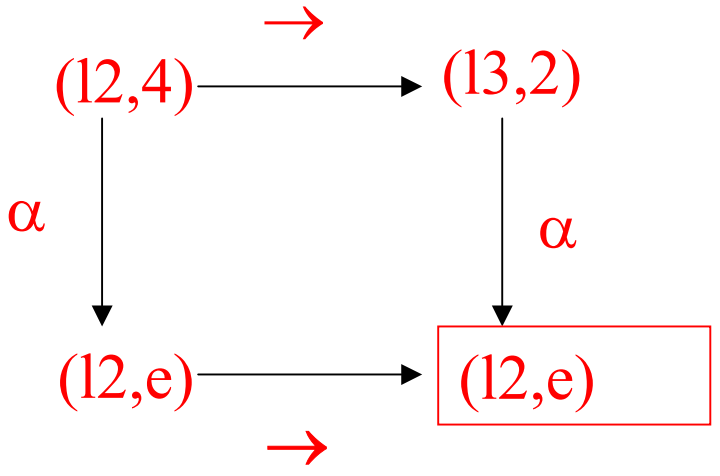


Una ejecución abstracta



IA: Simulación





IA: Bisimulación??

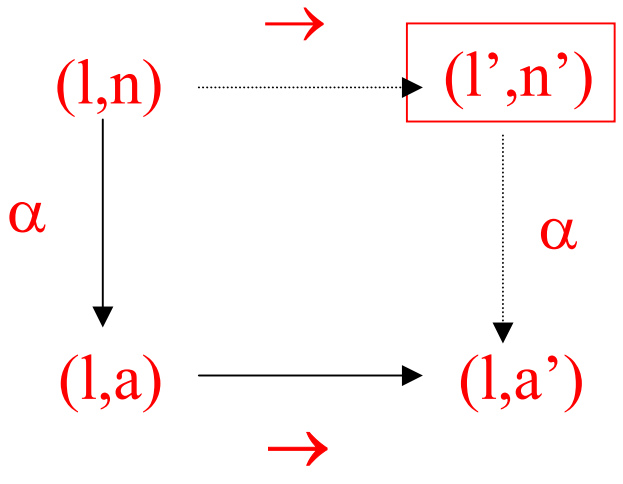
Bisimulación

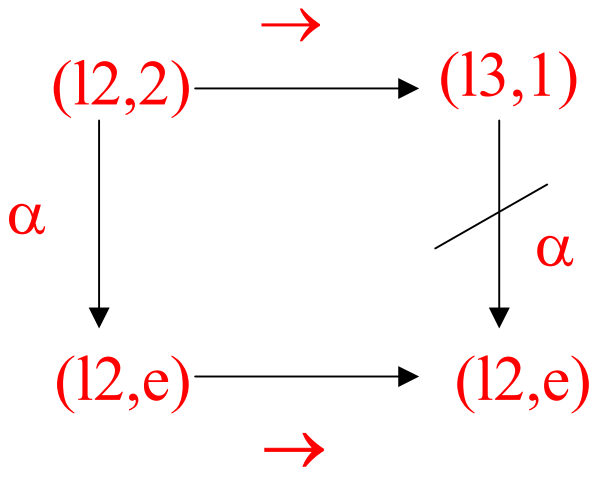
(AbState , $\rightarrow$) es bisimilar
a (State , $\rightarrow$) sii es una simulación y

si (l,a) $\rightarrow$ (l',a') $\alpha(n) = a$ entonces
existe n' $\in$ Integer tal que
(l,n) $\rightarrow$ (l',n') y $\alpha(n') = a'$

IA: Bisimulación??

No es bisimulación





No es bisimulación y por lo tanto hay pérdida de información

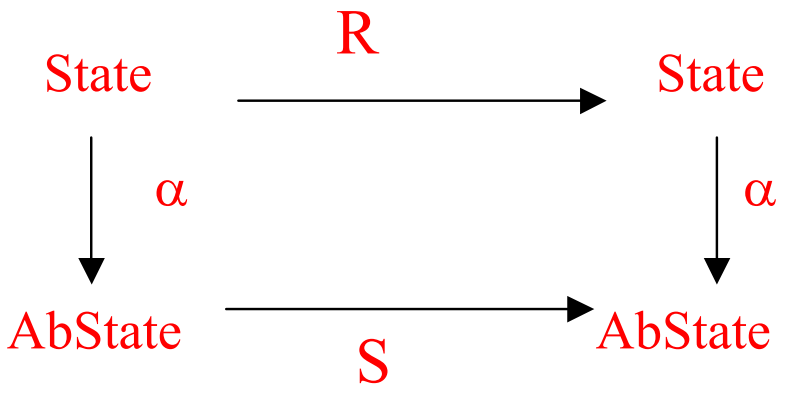
IA: Imprecisión

Noción de abstracción extendida

$\alpha : State \rightarrow AbState$

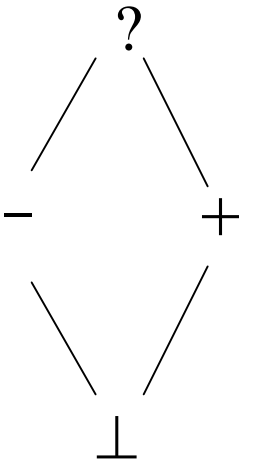
$\sigma \in AbState$ es una abstracción de $s \in State$ sii $\alpha(s) \sqsubseteq \sigma$

Además si $\rightarrow$ **y** $\rightarrow$ son funciones (R y S) entonces la máxima precisión equivale a la conmutatividad del diagrama



$$\alpha(R(l,n)) = S(\alpha((l,n)))$$

IA: No es siempre homomórfica



AbState = Label × AbState
AbState = Variable → {+, -, ?}
next : State → State **next** : AbState → AbState

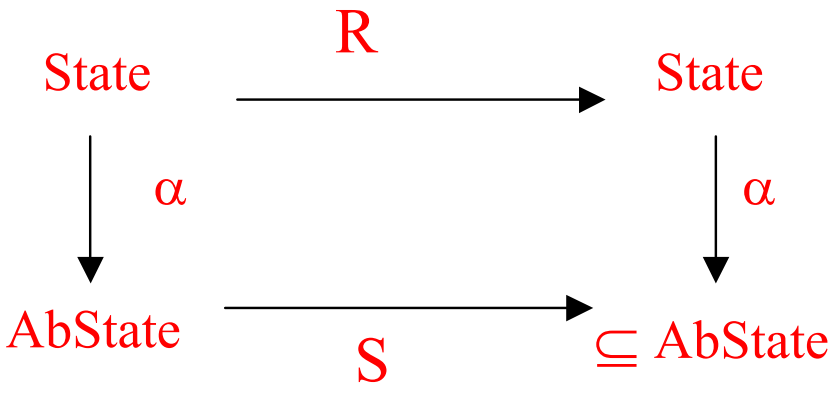
p : y := x + y; goto q

α(next((p, [x→1,y→-2]))) = α((q, [x→1,y→-1])) =
(q,[x→+,y→-])

next(α((p, [x→1,y→-2]))) = **next**((p, [x→+,y→-])) =
(q,[x→+,y→?])

Así la conmutatividad deseada falla.

IA: No es siempre homomórfica



$$\alpha(R(l,n)) \sqsubseteq S(\alpha((l,n)))$$

- En general, lo mejor que podemos esperar es una simulación semihomomórfica.

IA: No es siempre homomórfica

•Una forma simple de hacerlo es que el dominio abstracto tenga un orden parcial $\sqsubseteq$ asociado, donde $x \sqsubseteq y$ significa que x es una descripción más precisa que y, por ejemplo,

$$+ \sqsubseteq ?$$

si x es una descripción/abstracción segura de un valor preciso v ($\alpha(v) \sqsubseteq x$) y $x \sqsubseteq y$, entonces esperamos que y sea también una descripción segura de v, es decir, $\alpha(v) \sqsubseteq y$

IA: No es siempre homomórfica

- Las computaciones que utilizan valores abstractos no pueden ser más precisas que las que utilizan los valores reales.
- requerimos que para cada i y $x_1,\ldots, x_{ki} \in$ State

$$\alpha(a_i(x_1, \ldots, x_{ki})) \sqsubseteq b_i(\alpha x_1, \ldots, \alpha x_{ki})$$

donde a_i son operaciones funcionales sobre del programa abstracto

b_i son las correspondientes sobre el abstracto y además son monótonas:

$x \sqsubseteq y$ entonces $b_i(x) \sqsubseteq b_i(y)$

IA: No es siempre homomórfica

– La monotonía implica que

$$\alpha(\text{next}^n(s)) \sqsubseteq \textbf{next}^n(\alpha(s))$$

para todos los estados s y $n \geq 0$

Por inducción sobre n.

n = 1: $\beta(\text{next}(s)) \sqsubseteq \textbf{next}(\beta(s))$ es la hipótesis

n > 1: Supongamos que $\beta(\text{next}^n(s)) \sqsubseteq \textbf{next}^n(\beta(s))$ para todo s.

$\beta(\text{next}^{n+1}(s)) = \beta(\text{next}(\text{next}^n(s))) \sqsubseteq \textbf{next}(\beta(\text{next}^n(s))) \sqsubseteq$

$\sqsubseteq \textbf{next}^{n+1}(\beta(s))$ (β es monótona)

– Las computaciones de secuencias de transiciones de estados se aproximan de forma segura.

¿Cómo integramos los dos modelos?

– La ejecución real no puede simularse de una forma biunívoca de forma abstracta:

p: if x > y then goto p else goto r

$$\textbf{next}((p,[x\rightarrow+,y\rightarrow+])) = \begin{cases}(q,[x\rightarrow+,y\rightarrow+])\\(r,[x\rightarrow+,y\rightarrow+])\end{cases}$$

ya que las descripciones aproximadas contienen demasiada poca información para determinar la salida del test.

Operacionalmente esto lleva al nodeterminismo: el argumento de **next** no determina unívocamente el resultado.

IA: Indeterminismo

- Para tratar el indeterminismo elevamos

$$\text{next} : \text{State} \rightarrow \text{State}$$

para que funciones sobre conjunto de estados, es decir

$$\wp\,\text{next} : \wp(\text{State}) \rightarrow \wp(\text{State})$$

definida como

$$\wp\,\text{next}(\text{state-set}) = \{\text{next}(s) \mid s \in \text{State-set}\}$$

junto con una función abstracción

$$\alpha : \wp(\text{State}) \rightarrow \text{AbState}.$$

Este método desarrollado por Cousot and Cousot permite que **next** siga siendo una función.

IA: Conceptos Básicos

A: **While** n ≠ 1 **do**

B: **if** even(n)

then (C: n := n div 2; D:)

else (E: n := 3 * n + 1 F:)

G:

- El problema de Collatz (aún sin resolver) en la teoría de números consiste en determinar si este programa termina para todos los posibles valores iniciales *n*.

IA: Conceptos Básicos

Abstracción de una ejecución (n = 5).

A: **While** n ≠ 1 **do**

B: **if** even(n)

then (C: n := n div 2; D:)

else (E: n := 3 * n + 1 F:)

G:

n en A	n en B	n en C	n en D	n en E	n en F	n en G
5,16,8,4,2,1	5,16,8,4,2	16,8,4,2	8,4,2,1	5	16	1
⊤	⊤	par	⊤	impar	par	impar

IA: Conceptos Básicos #3

- **Extensión a todas las posibles ejecuciones**.
  - Este resultado se obtuvo realizando sólo una ejecución, y luego abstrayendo su salida.
  - Este análisis puede no terminar, y además no describe todas las ejecuciones.
  - La cuestión es: cómo obtener información sobre la paridad de n válida para todas las posibles ejecuciones.
  - Una forma natural es simular la computación, pero hacerla usando los valores abstractos

    Abs = {⊤, par, impar, ⊥}

IA: Conceptos Básicos #4

A: **while** n ≠ 1 **do**
B: **if** even(n)
then (C: n := n div 2; D:)
else (E: n := 3 * n + 1 F:)
G:

- Informalmente,
  - será siempre par en los puntos C y F,
  - será siempre impar en E,
  - a veces será par y a veces impar en los puntos B y D
  - será impar en G, siempre que el flujo de control alcance G.
  - Las operaciones individuales puede simularse mediante propiedades conocidas de los números, por ejemplo, 3n + 1 es par si n es impar y viceversa, mientras que n div 2 puede ser par o impar.

IA: Conceptos Básicos #5

- Simular el programa completo no es tan directo como simular una ejecución simple debido a que la ejecución sobre valores abstractos puede no ser en general determinista: debe tener en cuenta todas las posibles secuencias de ejecución sobre los datos reales que satisfagan la descripción de datos abstracta.

IA: Conceptos Básicos #6

- Metodología general
  - definir un orden parcial sobre el conjunto de datos abstractos, de manera que siempre cambien en la misma dirección durante la interpretación abstracta, reduciendo así los problemas de terminación.
  - Almacenar la información en una estructura separada, usualmente asociada a los puntos del programa (tales como los puntos de entrada a los bloques básicos)
  - Construir a partir del programa un sistema de ecuaciones de flujo de datos, una para cada punto del programa
  - Resolver las ecuaciones de flujo de datos (usualmente calculando el mayor o menor punto fijo).

IA: Conceptos Básicos #7

- Ninguno de los métodos clásicos de análisis de programa, relacionaba formalmente la semántica del lenguaje cuyos programa estaban siendo analizados.
- En su lugar formalizaban los métodos particulares que utilizaban. En particular ninguno podía incluir la ejecución real como un caso especial de la interpretación abstracta (aunque fuera no computable).
- Esta fue la aportación de Cousot en 1977.

IA: Semántica recolectora o acumuladora

- Asocia a cada punto del programa el conjunto de todos los valores de las variables que pueden ocurrir alguna vez cuando el control del programa alcanza dicho punto, cuando el programa se ejecuta con unos datos iniciales.
- Una gran variedad de análisis de flujo de datos (aunque no todos) pueden realizarse encontrando una aproximación finitamente computable de la semántica recolectora.

IA: Semántica recolectora #2

- La semántica recolectora aplica puntos del programa con conjuntos de estados $\wp(\text{State})$
  - $(\wp(\text{State}), \subseteq)$ es un retículo.
  - el supremo $A \cup B$ y el ínfimo $A \cap B$.
  - El retículo $\wp(\text{State})$ es completo, lo que significa que cualquier colección de conjuntos de estados tienen su supremo e ínfimo en el conjunto $\wp(\text{State})$.

IA: Semántica recolectora #3

- Una interpretación abstracta se obtiene definiendo la función abstracción $\alpha: \wp(\text{Store}) \rightarrow \text{Abs}$, donde Abs en un retículo de las descripciones de los conjuntos de estados.
  - Los símbolos $\vee$ y $\wedge$ suelen usarse para expresar el supremo y el ínfimo en Abs.
- La función abstracción suele utilizarse con su dual llamada función concretización $\gamma : \text{Abs} \rightarrow \wp(\text{Store})$.

- (α, γ) suelen formar una conexión de Galois

IA: Semántica recolectora #4

- Para un programa con una variable, podríamos usar el retículo

$$\text{Abs} = \{\bot, \top, \text{par}, \text{impar}\}$$

$$\alpha: \wp(\text{State}) \rightarrow \text{Abs}$$

- $\alpha(\emptyset) = \bot$
- $\alpha(c) = \text{par}$, si $c \subseteq \{2n: n \in N\}$
- $\alpha(c) = \text{impar}$, si $c \subseteq \{2n + 1: n \in N\}$
- $\alpha(c) = \top$, en otro caso

$$\gamma : \text{Abs} \rightarrow \wp(\text{Store}).$$

- $\gamma(\bot) = \emptyset$
- $\gamma(\text{par}) = \{2n: n \in N\}$
- $\gamma(\text{impar}) = \{2n + 1: n \in N\}$
- $\gamma(\top) = N$

IA: Semántica recolectora #5

- La ia puede considerarse como ejecutar el programa sobre un retículo de descripciones abstractas de los estados imprecisas pero computables, en lugar de usar el retículo preciso pero no computable de la semántica recolectora
- La computabilidad se alcanza utilizando un retículo noetheriano, es decir, sin cadenas ascendentes infinitas. También pueden utilizarse retículos más generales y aplicar técnicas de widening para obtener la computabilidad.

IA: Semántica recolectora #6

– Sea p_0 el punto inicial de un programa y sea p otro punto del programa. El conjunto de las configuraciones que llegan al punto p, empezando desde un conjunto S_0 de posibles estados iniciales se define como:

$$acc_p = \{s \mid (p,s) = next^n((p_0,s_0)) \text{ para algún } s_0 \in S_0 , n \geq 0\}$$

– La semántica recolectora asocia a cada punto del programa el conjunto acc_p

IA: Semántica recolectora #7

– Para el programa del ejemplo sólo hay una variable así que el conjunto de estados tiene la forma:

$$\{[n \rightarrow a_1], [n \rightarrow a_2], [n \rightarrow a_3], \dots\}$$

que denotamos con

$$\{a_1, a_2, a_3, \dots\}$$

- $S_0 = \{5\}$

acc_A	acc_B	acc_C	acc_D	acc_E	acc_F	acc_G
{5}	{5,16,8,4,2}	{16,8,4,2}	{8,4,2,1}	{5}	{16}	{1}

IA: Semántica recolectora #8

A: **while** n ≠ 1 **do**

B: **if** even(n)

then (C: n := n div 2; D:)

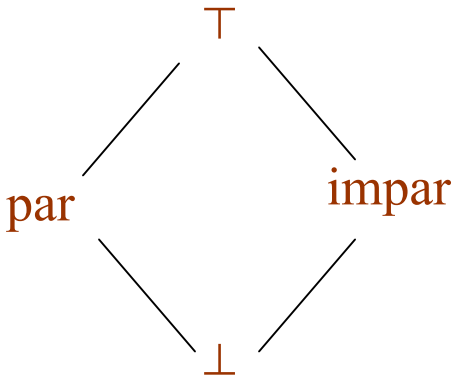
else (E: n := 3 * n + 1 F:)

G:

- Ecuaciones de flujo de datos
  - $acc_A = S_0$
  - $acc_B = (acc_A \cup acc_D \cup acc_F) \cap \{n \mid n \in \{0,1,2,\dots\} \setminus \{1\}\}$
  - $acc_C = acc_B \cap \{0, 2, 4, \dots\}$
  - $acc_D = \{n \text{ div } 2 \mid n \in acc_C\}$
  - $acc_E = acc_B \cap \{1,3,5,\dots\}$
  - $acc_F = \{3n + 1, \mid n \in acc_E\}$
  - $acc_G = (acc_A \cup acc_D \cup acc_F) \cap \{1\}$
  - tienen un único punto fijo por la completitud de $\wp$(State) y es fácil ver que es justo la tupla de valores anterior.
  - El conjunto de ecuaciones puede obtenerse directamente a partir del programa.

IA: Semántica Recolectora #9

- Interpretación abstracta del ejemplo
- La relación de orden sobre Abs viene dada por el siguiente diagrama:



abs_A	abs_B	abs_C	abs_D	abs_E	abs_F	abs_G
impar	⊤	par	⊤	impar	par	impar

IA: Ejemplo

- **Abstracción del conjunto de todas las ejecuciones**.
  - Modelar las ecuaciones anteriores aplicandole α a los conjuntos obtenidos.
  - α definida es claramente monótona
  - Modelamos la inclusión de conjuntos $\subseteq$, mediante $\sqsubseteq$ que se aplica sobre las ejecuciones abstractas
  - Modelamos $\cup$ mediante $\vee$ y $\cap$ mediante $\wedge$.

IA: Ejemplo #2

- Abstracción de las operaciones

$$f_{n \text{ div } 2}\,(abs) = \begin{cases} \bot, \text{ si } abs = \bot \\ \top, \text{ en otro caso} \end{cases}$$

$$f_{3n+1}\,(abs) = \begin{cases} \bot, & \text{si } abs = \bot \\ par, & \text{si } abs = par \\ impar, & \text{si } abs = impar \\ \top, & \text{si } abs = \top \end{cases}$$

- $f_{n \text{ div } 2}$ y f_{3n+1} están definidas a mano, pero puede hacerse de forma sistemática.

IA: Ejemplo #3

– Ecuaciones de flujo de datos abstractas

A: **while** $n \neq 1$ **do**
  B: **if** even(n)
    then (C: n := n div 2; D:)
    else (E: n := 3 * n + 1 F:)
G:

$abs_A = \alpha(S_0)$
$abs_B = (abs_A \vee abs_D \vee abs_F) \wedge \top$
($\wedge \top$ puede eliminarse)
$abs_C = abs_B \wedge$ par
$abs_D = f_{n \text{ div } 2}(abs_C)$
$abs_E = abs_B \wedge$ impar
$abs_F = f_{3n+1}(abs_E)$
$abs_G = (abs_A \vee abs_D \vee abs_F) \wedge$ impar

IA: Ejemplo #4

- Abs es completo
- Todos los operadores y funciones que aparecen son monótonos
- La ecuación tiene un único punto fijo.
- La función α es también monótona y si también fuera u homomorfismo con respecto a $\cup$, $\vee$, y $\cap$, $\wedge$, la menor solución a la ecuaciones de flujo aproximadas sería exactamente:

$$abs_A = \alpha(acc_A), \ldots, abs_G = \alpha(acc_G)$$

- Sin embargo, no es un homomorfismo ya que por ejemplo,

$$\alpha(\{2\}) \wedge \alpha(\{4\}) = par \neq \alpha(\{2\} \cap \{4\})$$

IA: Ejemplo #5

- Por otro lado, sí es cierto que:
  - $\alpha(A \cup B) \sqsubseteq \alpha(A) \vee \alpha(B)$
  - $\alpha(\{n \text{ div } 2 \mid n \in A\}) \sqsubseteq f_{n \text{ div } 2}(\alpha(A))$
  - $\alpha(A \cap B) \sqsubseteq \alpha(A) \wedge \alpha(B)$
  - $\alpha(\{2n + 1 \mid n \in A\}) \sqsubseteq f_{3n+1}(\alpha(A))$
- Por lo que se tiene que

$$\alpha(acc_A) \sqsubseteq abs_A$$
$$\alpha(acc_B) \sqsubseteq abs_B$$
$$\ldots$$
$$\alpha(acc_G) \sqsubseteq abs_G$$

IA: Ejemplo #5

$$abs_A = \alpha(S_0)$$

$$abs_B = (abs_A \vee abs_D \vee abs_F) \wedge \top$$

($\wedge \top$ puede eliminarse)

$$abs_C = abs_B \wedge par$$

$$abs_D = f_{n \text{ div } 2}(abs_C)$$

$$abs_E = abs_B \wedge impar$$

$$abs_F = f_{3n+1}(abs_E)$$

$$abs_G = (abs_A \cup abs_D \cup abs_F) \wedge impar$$

IA: Ejemplo #6

– Siguiendo la computación iterativa del menor punto fijo, suponiendo $S_0 = \{5\}$

abs_A	abs_B	abs_C	abs_D	abs_E	abs_F	abs_G	Iteración
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	0
impar	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	1
impar	impar	$\bot$	$\bot$	$\bot$	$\bot$	impar	2
impar	impar	$\bot$	$\bot$	impar	$\bot$	impar	3
impar	impar	$\bot$	$\bot$	impar	par	impar	4
impar	$\top$	$\bot$	$\bot$	impar	par	impar	5
impar	$\top$	par	$\bot$	impar	par	impar	6
impar	$\top$	par	$\top$	impar	par	impar	7,8,…

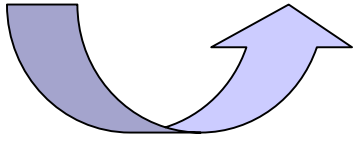
- n es siempre par en los puntos C y F, y siempre es impar en los puntos E y G.

Transformación basada en la IA

– El análisis de flujo revela que el programa podría ser un poco más eficiente si desplegamos el bucle en el punto F. La razón es que los tests n ≠ 1 y "par n" deben ser ambos true en la iteración después de F, así que no hay que comprobarlos.

A: **While** n ≠ 1 **do**
B: **if** even(n)
then (C: n := n div 2; D:)
else (E: n := 3 * n + 1 F:)
G:

While n ≠ 1 **do**
if even(n)
then (n := n div 2;)
else (else n := (3 * n + 1) div 2

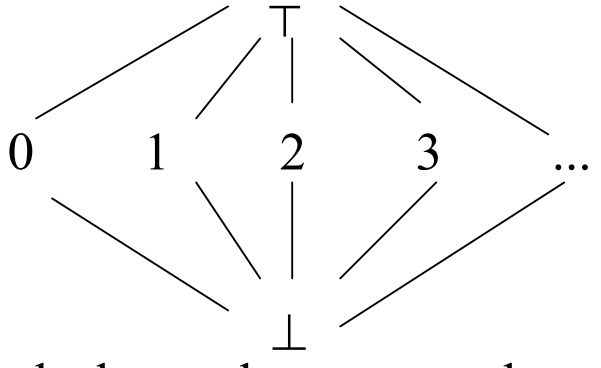


IA: Terminación

- El menor punto fijo puede encontrarse (como es usual) iterando a partir de $[pp_1 \rightarrow \bot, ..., pp_m \rightarrow \bot]$
- Por la monotonía de $\wedge$, f_{ndiv2}, etc estos valores sólo pueden crecer o permanecer estables, así que las iteraciones siempre terminan cuando el retículo de datos aproximados no tiene cadenas ascendentes infinitas (o cuando es finito).

IA: Terminación

- **Propagación de Constantes**. Este es un ejemplo de un retículo infinito pero con altura finita.
- Se utiliza para detectar que las variables no varían y tiene Abs = {⊥, ⊤, 0,1,… }, con la relación de orden que muestra el dibujo:



- $\alpha(V) = \bot$, si $V = \varnothing$
- $\alpha(V) = n$, si $V = \{n\}$
- $\alpha(V) = \top$, en otro caso

- Hay también retículos en los que toda las cadenas ascendentes tienen altura finita aunque el retículo en sí es no acotado, como por ejemplo Abs = (N , ≥)

IA: Corrección

- El análisis del programa de Collatz es claramente correcto en el siguiente sentido:
- si el control llega a C entonces el valor de n será par y lo mismo para el resto de los puntos del programa y datos abstractos.
- La corrección (o validez) del análisis par/impar para todos los programas posibles puede verificarse fácilmente, dada la estrecha relación entre las ecuaciones de flujo y las definidas por la semántica recolectora.

IA: Corrección #2

- Un análisis similar, pero más sencillo, es el análisis de la alcanzabilidad (para eliminar código inalcanzable) sirve para ilustrar la corrección.
- Este análisis usa Abs ={⊥, ⊤} con la relación $\bot \sqsubseteq \top$ y la función abstracción definida como sigue (donde $a \in$ Abs y $S \subseteq$ State):

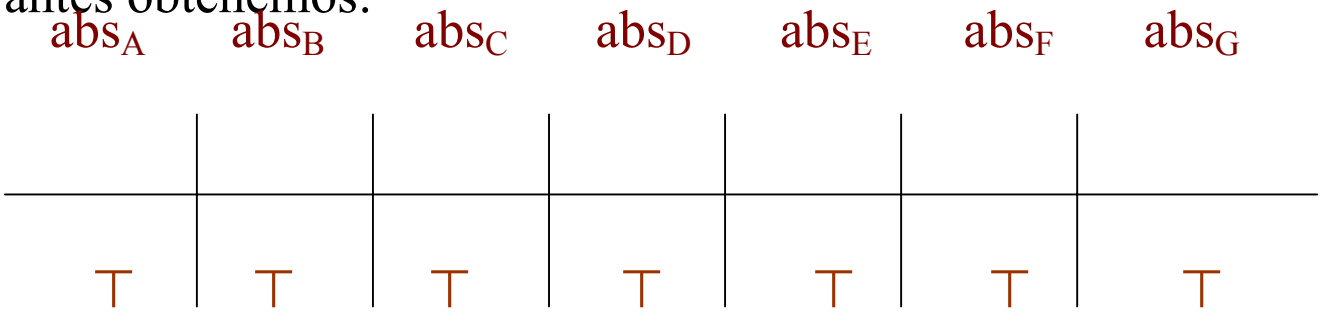
$$\alpha(S) = \begin{cases} \bot & \text{si } S = \varnothing \\ \top & \text{en otro caso} \end{cases}$$

$$f_{ndiv2}(a) = \begin{cases} \bot & \text{si } a = \bot \\ \top & \text{en otro caso} \end{cases}$$

$$f_{3n+1}(a) = \begin{cases} \bot & \text{si } a = \bot \\ \top & \text{en otro caso} \end{cases}$$

IA: Corrección #3

- Intuitivamente, $\bot$ abstrae sólo al conjunto vacío de estados, con lo que describe los puntos del programa no alcanzables, mientras que $\top$ describe los puntos alcanzables. Calculando el punto fijo como antes obtenemos:



- Todos los puntos del programa, incluyendo G, son alcanzables, independientemente del valor inicial n.
- La alcanzabilidad de G para la entrada n implica la terminación, y es bien conocido que esta es una cuestión abierta.

IA: Corrección #4

- Un análisis más cuidadoso revela que $\bot$ en un punto del programa representa que p no puede ser alcanzado, mientras que $\top$ representa que p *podría* ser alcanzado. Esto no implica la terminación del programa
- El ejemplo muestra que debemos examinar la corrección con más cuidado.

IA: Corrección #5

- Propiedades deseables del conjunto de valores abstractos Abs
  - (Abs, $\sqsubseteq$, $\vee$, $\wedge$, $\bot$, $\top$) debe ser un retículo completo
  - Si Abs es finito entonces es completo.
- Propiedades deseables de la función abstracción y concretización

  α : Conc → Abs

  γ : Abs → Conc

IA: Corrección #6

- Propiedades deseables de la funciones abstracción y concretización

α : Conc → Abs

γ : Abs → Conc

1. α y γ son monótonas
2. $\forall a \in Abs, a = \alpha(\gamma(a))$
3. $\forall c \in Conc, c \sqsubseteq_{conc} \gamma(\alpha(c))$

– La condición 1 expresa que los valores abstractos mayores representan conjuntos de estados mayores.

– La condición 2 es natural

– La condición 3 nos dice que concretizar un conjunto previamente abstraído nos hace perder información.

IA: Corrección #7

- Estas tres condiciones implican que Conc y Abs forman una inserción de Galois.

- Estas condiciones equivalen a decir que:

  * α es continua
  * $\forall c \in Conc, a \in Abs, \alpha(c) \sqsubseteq_{Abs} a$ sii $c \sqsubseteq_{conc} \gamma(a)$

Bibliografía

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