Análisis inter-procedimientos

CI4722 – Lenguajes de Programación III

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Una perspectiva práctica

. . . and now for something completely different

- La discusión sobre análisis intra e inter procedimiento ha sido basada en conjuntos.
  - Conjuntos para la información en cada punto de ejecución.
  - Transferencia basada en transformación de conjuntos.
  - Algoritmo iterativo para encontrar puntos fijos.
- En el fondo, queremos calcular relaciones – lógicas.
  - En lugar de escribir. . .

$$D \in IN[B]$$

  - . . . queremos escribir. . .

```
in(D,B)
```

- Breve, susceptible de ser «inferido» e integrado.

¿Cómo expresar y calcular?

UNIVERSIDAD SIMÓN BOLÍVAR

Datalog

Prolog pour la minorie

- Datalog es una forma simplificada de Prolog.
- **Atomos** – $p(x_1, x_2, \ldots, x_n)$
  - p es un **predicado**
  - x_i son **términos** – variables o constantes.
- Un **átomo fundamental** (*ground atom*) sólo tiene constantes como argumentos – representa un «hecho».
  - Presencia de un «hecho» lo asevera.
  - Ausencia de un «hecho» lo falsifica.
- Un **literal** es un átomo o su negación (`NOT`).
- **Variables** son símbolos que comienzan con mayúscula – el resto (números o símbolos) se toman por su valor presente.

UNIVERSIDAD SIMÓN BOLÍVAR

Datalog

Prolog pour la minorie, part deux

- **Reglas** – $H :- B_1, B_2, \ldots, B_n$
  - H es un **átomo** – «objetivo» o «cabeza» de la regla.
  - Cada B_i es un **literal** – «sub-objetivos».
  - La coma implica conjunción lógica – el libro usa &
- Para *aplicar* una regla:
  - Si existe una sustitución que use átomos fundamentales y haga ciertos todos los sub-objetivos, se dá por cierto el objetivo.
  - Se repite hasta alcanzar un objetivo particular, posiblemente enriqueciendo las relaciones – *memorización* implícita.
- **Programa Datalog** – aplicar reglas sobre átomos fundamentales.
  - El programa...

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- path(X,Z), path(Z,Y).
```

  - ...requiere átomos fundamentales edge para calcular.

UNIVERSIDAD SIMÓN BOLÍVAR

El Origen de los Predicados

De la instancia o de la solución

- **Predicados por Extensión** (*Extensional Database* o EDB) – definidos a priori como hechos conocidos derivados del problema que se pretende resolver.
- **Predicados por Inclusión** (*Intensional Database* o IDB) – definidos por las reglas que buscan la solución al problema.
- Todo predicado debe ser EDB o IDB, pero no ambos – la izquierda de una regla, es necesariamente IDB.
- Usaremos Datalog para resolver problemas de análisis de flujo
  - Predicados EDB – dependen del grafo de flujo.
  - Predicados IDB – reglas del problema de análisis.

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

Construcción del EDB

- Cada instrucción puede *gen*-erar o *kill* definiciones.
- Representar puntos del programa – n-ésima instrucción del bloque B.

Definiciones Disponibles en Datalog

Construcción del EDB

- Cada instrucción puede *gen*-erar o *kill* definiciones.
- Representar puntos del programa – n-ésima instrucción del bloque B.
- El grafo de flujo aporta dos predicados EDB
  - $def(B, N, X)$ – N-ésima instrucción del bloque B *podría* definir X.
  - $succ(B, N, C)$ – bloque C es sucesor del bloque B, que tiene N instrucciones (punto N de B conecta con punto 0 de C).
- Suponiendo el bloque B_1 seguido del bloque B_2

```
b1:
     x   = y + z
     *p  = u
     x   = v
```

  - $def(b1, 1, x)$, $def(b1, 3, x)$ y $def(b1, 2, Y)$ son ciertos
  - $succ(b1, 3, b2)$ es cierto.

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

Construcción del IDB

- Determinar si una definición particular alcanza un punto específico.

Definiciones Disponibles en Datalog

Construcción del IDB

- Determinar si una definición particular alcanza un punto específico.
- Predicado IDB $rd(B, N, C, M, X)$ indica que la definición de X en la M-ésima instrucción del bloque C alcanza el punto N del bloque B.

```
rd(B,N,B,N,X) :- def(B,N,X)
rd(B,N,C,M,X) :- rd(B,N-1,C,M,X),
                 def(B,N,Y),
                 X /= Y
rd(B,0,C,M,X) :- rd(D,N,C,M,X),
                 succ(D,N,B)
```

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

Construcción del IDB

- Determinar si una definición particular alcanza un punto específico.
- Predicado IDB $rd(B, N, C, M, X)$ indica que la definición de X en la M-ésima instrucción del bloque C alcanza el punto N del bloque B.

```
rd(B,N,B,N,X) :- def(B,N,X)
rd(B,N,C,M,X) :- rd(B,N-1,C,M,X),
                 def(B,N,Y),
                 X /= Y
rd(B,0,C,M,X) :- rd(D,N,C,M,X),
                 succ(D,N,B)
```

- Interpretación

Definiciones Disponibles en Datalog

Construcción del IDB

- Determinar si una definición particular alcanza un punto específico.
- Predicado IDB $rd(B, N, C, M, X)$ indica que la definición de X en la M-ésima instrucción del bloque C alcanza el punto N del bloque B.

```
rd(B,N,B,N,X) :- def(B,N,X)
rd(B,N,C,M,X) :- rd(B,N-1,C,M,X),
                 def(B,N,Y),
                 X /= Y
rd(B,0,C,M,X) :- rd(D,N,C,M,X),
                 succ(D,N,B)
```

- Interpretación
  1. Equivalente a *gen*.

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

Construcción del IDB

- Determinar si una definición particular alcanza un punto específico.
- Predicado IDB $rd(B, N, C, M, X)$ indica que la definición de X en la M-ésima instrucción del bloque C alcanza el punto N del bloque B.

```
rd(B,N,B,N,X) :- def(B,N,X)
rd(B,N,C,M,X) :- rd(B,N-1,C,M,X),
                 def(B,N,Y),
                 X /= Y
rd(B,0,C,M,X) :- rd(D,N,C,M,X),
                 succ(D,N,B)
```

- Interpretación
  1. Equivalente a *gen*.
  2. Una definición «pasa de largo» si no la «matan».

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

Construcción del IDB

- Determinar si una definición particular alcanza un punto específico.
- Predicado IDB $rd(B, N, C, M, X)$ indica que la definición de X en la M-ésima instrucción del bloque C alcanza el punto N del bloque B.

```
rd(B,N,B,N,X) :- def(B,N,X)
rd(B,N,C,M,X) :- rd(B,N-1,C,M,X),
                 def(B,N,Y),
                 X /= Y
rd(B,0,C,M,X) :- rd(D,N,C,M,X),
                 succ(D,N,B)
```

- Interpretación
  1. Equivalente a *gen*.
  2. Una definición «pasa de largo» si no la «matan».
  3. Una definición llega a un nuevo bloque, si salió del anterior.

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

¿Podríamos hacerlo mejor?

- *def* es conservadora – apuntadores a «cualquie cosa».

Definiciones Disponibles en Datalog

¿Podríamos hacerlo mejor?

- *def* es conservadora – apuntadores a «cualquie cosa».
- Aprovechamos la información de tipos – (¡hola tabla de símbolos!)
  - $assign(B, N, X)$ indica que la N-ésima instrucción del bloque B tiene el *l-value* X en el lado izquierdo.
  - $type(X, T)$ indica que el *l-value* X tiene tipo T.

UNIVERSIDAD SIMÓN BOLÍVAR

Definiciones Disponibles en Datalog

¿Podríamos hacerlo mejor?

- *def* es conservadora – apuntadores a «cualquie cosa».
- Aprovechamos la información de tipos – (¡hola tabla de símbolos!)
  - $assign(B, N, X)$ indica que la N-ésima instrucción del bloque B tiene el *l-value* X en el lado izquierdo.
  - $type(X, T)$ indica que el *l-value* X tiene tipo T.
- ¡Podemos convertir *def* en EDB!

```
def(B,N,X) :- assign(B,N,X)

def(B,N,X) :- assign(B,N,*P),
              type(X,T),
              type(P,*T)
```

Unificación FTW!

Analizando Apuntadores

Con «Mucho Cuidado»

- Dado un conjunto de apuntadores, ¿son aliases? – si dos apuntadores *podrían* apuntar a un mismo objeto, si.
- Necesariamente inter-procedimiento.
- Problema particularmente difícil en C
  - Porque se puede apuntar a cualquier cosa.
  - Porque hay llamadas indirectas.
- Problema moderadamente difícil en Java (ugh)
  - Los apuntadores («referencias») sólo van al *heap*.
  - Despacho dinámico es un reto.

UNIVERSIDAD SIMÓN BOLÍVAR

Modelo de Apuntadores y Referencias

- Seguiremos un modelo como el de Java:
  - Variables (estáticas o en pila) de tipo «apuntador a T» – referencias.
  - Sólo se apunta a objetos en el *heap*.
  - Un objeto en el *heap* podría tener *campos* cuyo valor es una referencia a otro objeto en el *heap*.

UNIVERSIDAD SIMÓN BOLÍVAR

Modelo de Apuntadores y Referencias

- Seguiremos un modelo como el de Java:
  - Variables (estáticas o en pila) de tipo «apuntador a T» – referencias.
  - Sólo se apunta a objetos en el *heap*.
  - Un objeto en el *heap* podría tener *campos* cuyo valor es una referencia a otro objeto en el *heap*.
- Nuestro análisis inicial será insensible al contexto.
  - Sólo interesa saber que *v puede* apuntar al objeto *h* en el heap – no nos interesa *dónde en el programa*.
  - Podemos diferenciar variables según el espacio de nombres.
  - Denotaremos los objetos en el *heap* según la línea que les dió «origen»

UNIVERSIDAD SIMÓN BOLÍVAR

Modelo de Apuntadores y Referencias

- Seguiremos un modelo como el de Java:
  - Variables (estáticas o en pila) de tipo «apuntador a T» – referencias.
  - Sólo se apunta a objetos en el *heap*.
  - Un objeto en el *heap* podría tener *campos* cuyo valor es una referencia a otro objeto en el *heap*.
- Nuestro análisis inicial será insensible al contexto.
  - Sólo interesa saber que *v puede* apuntar al objeto *h* en el heap – no nos interesa *dónde en el programa*.
  - Podemos diferenciar variables según el espacio de nombres.
  - Denotaremos los objetos en el *heap* según la línea que les dió «origen»
- Terminado el análisis sabremos, para cada variable y campo, a cuáles objetos en el *heap* podrían estar apuntando.

«Points-to Analysis»
aliases si conjuntos tienen intersección no vacía

Ignorando el flujo de control

. . . y sus efectos perniciosos

- Consideremos el siguiente fragmento:

```
h:  a = new Object();
i:  b = new Object();
j:  c = new Object();
    a = b;
    b = c;
    c = a;
```

  - Si seguimos el flujo de control («corremos el programa») – `a` sólo apunta a `i`, `b` sólo apunta a `j`, `c` sólo apunta a `i`,
  - No sólo observamos lo «generado», sino también lo «matado».

UNIVERSIDAD SIMÓN BOLÍVAR

Ignorando el flujo de control

. . . y sus efectos perniciosos

- Consideremos el siguiente fragmento:

```
h:  a = new Object();
i:  b = new Object();
j:  c = new Object();
    a = b;
    b = c;
    c = a;
```

  - Si seguimos el flujo de control («corremos el programa») – `a` sólo apunta a `i`, `b` sólo apunta a `j`, `c` sólo apunta a `i`,
  - No sólo observamos lo «generado», sino también lo «matado».
- Vamos a suponer que las instrucciones se ejecutan en cualquier orden.
  - Mapa global «points to» único – posibles destinos de apuntadores.
  - Instrucciones «generan» pero nunca «matan».
  - Rápido, barato y malo – pero mejor que nada.

UNIVERSIDAD SIMÓN BOLÍVAR

Primer intento, en Datalog

Sólo instrucciones, no importan las llamadas

- `T v = new T()` (**creación de objetos**)
  Crea un objeto en el *heap* y `v` *puede* apuntarle.

Primer intento, en Datalog

Sólo instrucciones, no importan las llamadas

- `T v = new T()` (**creación de objetos)**
  Crea un objeto en el *heap* y `v` *puede* apuntarle.
- `v = w` (**copiar objetos)**
  La variable `v` apunta a lo que apuntaba `w`

UNIVERSIDAD SIMÓN BOLÍVAR

Primer intento, en Datalog

Sólo instrucciones, no importan las llamadas

- `T v = new T()` (**creación de objetos)**
  Crea un objeto en el *heap* y `v` *puede* apuntarle.
- `v = w` (**copiar objetos)**
  La variable `v` apunta a lo que apuntaba `w`
- `v.f = w` (**asignar a campo)**
  El tipo de `v` debe tener un campo `f` referencia. Si `v` apunta al objeto `h` y `w` apunta a `g`, entonces `f` contenido en `h` apunta a `g`.

UNIVERSIDAD SIMÓN BOLÍVAR

Primer intento, en Datalog

Sólo instrucciones, no importan las llamadas

- `T v = new T()` (**creación de objetos)**
  Crea un objeto en el *heap* y `v` *puede* apuntarle.
- `v = w` (**copiar objetos)**
  La variable `v` apunta a lo que apuntaba `w`
- `v.f = w` (**asignar a campo)**
  El tipo de `v` debe tener un campo `f` referencia. Si `v` apunta al objeto `h` y `w` apunta a `g`, entonces `f` contenido en `h` apunta a `g`.
- `v = w.f` (**leer un campo)**
  El tipo de `w` debe tener un campo `f` referencia. Si `w` apunta a un objeto cuyo campo `f` apunta al objeto `h`, entonces `v` apunta a `h`.

UNIVERSIDAD SIMÓN BOLÍVAR

Primer intento, en Datalog

Sólo instrucciones, no importan las llamadas

- `T v = new T()` (**creación de objetos)**
  Crea un objeto en el *heap* y `v` *puede* apuntarle.
- `v = w` (**copiar objetos)**
  La variable `v` apunta a lo que apuntaba `w`
- `v.f = w` (**asignar a campo)**
  El tipo de `v` debe tener un campo `f` referencia. Si `v` apunta al objeto `h` y `w` apunta a `g`, entonces `f` contenido en `h` apunta a `g`.
- `v = w.f` (**leer un campo)**
  El tipo de `w` debe tener un campo `f` referencia. Si `w` apunta a un objeto cuyo campo `f` apunta al objeto `h`, entonces `v` apunta a `h`.
- Acceso compuesto a campos (`v = w.f.g`) se descompone

```
v1 = w.f
v  = v1.g
```

UNIVERSIDAD SIMÓN BOLÍVAR

Primer intento, en Datalog

¿Qué necesitamos calcular?

- $pts(V, H)$ si la variable V podría apuntar a H en *heap*.

```
pts(V,H) :- "H : T V = new T"

pts(V,H) :- "V = W", pts(W,H)

pts(V,H) :- "V = W.F", pts(W,G), htps(G,F,H).
```

- $hpts(H, F, G)$ si el campo F del objeto H podría apuntar al objeto G.

```
hpts(H,F,G) :- "V.F = W",
               pts(W,G),
               pts(V,H)
```

- Los ETB fueron construidos via «pattern matching» – queda como ejercicio escribirlos como predicados formales.
- Noten que *pts* y *hpts* son co-recursivas – la evaluación iterativa de Datalog impide que se «cuelgue».

UNIVERSIDAD SIMÓN BOLÍVAR

«¡Tipo que Java tiene tipos y eso!»

. . . ya vimos ésta película

- Las variables sólo apuntan a tipos compatibles por sub-tipo.
- Podemos agregar tres EDB auxiliares
  - $vType(V, T)$ si la variable V fue *declarada* con tipo T.
  - $hType(H, T)$ si el objeto H fue reservado usando el tipo T. Noten que no se puede saber con precisión si se usan métodos nativos – ¡ser conservadores!
  - $assignable(T, S)$ si un objeto de tipo S puede ser asignado a una variable de tipo T. Esto se deduce de la tabla de símbolos y las librerías provistas por el lenguaje.

Incorporando estos predicados. . .

UNIVERSIDAD SIMÓN BOLÍVAR

Looks legit

```
pts(V,H) :- "H : T V = new T"
pts(V,H) :- "V = W",
            pts(W,H),
            vType(V,H),
            hType(H,S),
            assignable(T,S)
pts(V,H) :- "V = W.F",
            pts(W,G),
            htps(G,F,H).
            vType(V,T),
            hType(H,S),
            assignable(T,S)

hpts(H,F,G) :- "V.F = W",
               pts(W,G),
               pts(V,H)
```

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacerlo inter-procedimiento?

¿Cómo lo hacemos «saltar»?

Ante una llamada a método

```
x = y.n(z1,z2,...,zN)
```

¿Cómo hacerlo inter-procedimiento?

¿Cómo lo hacemos «saltar»?

Ante una llamada a método

```
x = y.n(z1,z2,...,zN)
```

1. Se determina el tipo `t` de `y` (el **invocante**).
   Sea `m` el método con nombre `n` en la superclase más cercana a `t`.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacerlo inter-procedimiento?

¿Cómo lo hacemos «saltar»?

Ante una llamada a método

```
x = y.n(z1,z2,...,zN)
```

1. Se determina el tipo `t` de `y` (el **invocante**).
   Sea `m` el método con nombre `n` en la superclase más cercana a `t`.
2. Los parámetros formales de `m` serán asignados con los objetos apuntados por los parámetros actuales. El parámero formal implícito (`this` en Java) será asignado con el invocante.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacerlo inter-procedimiento?

¿Cómo lo hacemos «saltar»?

Ante una llamada a método

```
x = y.n(z1,z2,...,zN)
```

1. Se determina el tipo `t` de `y` (el **invocante**).
   Sea `m` el método con nombre `n` en la superclase más cercana a `t`.
2. Los parámetros formales de `m` serán asignados con los objetos apuntados por los parámetros actuales. El parámero formal implícito (`this` en Java) será asignado con el invocante.
3. El objeto retornado por `m` se asigna a la variable en el lado izquierdo de la asignación.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacerlo inter-procedimiento?

Por eso, ¿cómo lo hacemos «saltar»?

- Parámetros y resultados serán copias – ya lo hicimos.
- ¿Cómo averigüamos el tipo del invocante?
  - Según la declaración de la variable – demasiado conservador.
  - Tendríamos que saber a qué apunta para saber el destino de la llamada. . .
  - . . . pero sin los destinos no sabemos a qué apuntan las variables.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacerlo inter-procedimiento?

Por eso, ¿cómo lo hacemos «saltar»?

- Parámetros y resultados serán copias – ya lo hicimos.
- ¿Cómo averigüamos el tipo del invocante?
  - Según la declaración de la variable – demasiado conservador.
  - Tendríamos que saber a qué apunta para saber el destino de la llamada...
  - ...pero sin los destinos no sabemos a qué apuntan las variables.

Agregamos destinos mientras calculamos lo apuntado por las llamadas cuyo destino calculamos según lo apuntado por...

UNIVERSIDAD SIMÓN BOLÍVAR

Don't panic!

Comenzamos por unos EDB fáciles de calcular

- $actual(S, I, V)$ indica que V es el I-ésimo parámetro actual pasado en el punto de llamada S.
- $formal(M, I, V)$ indica que V es el I-ésimo parámetro formal declarado en el método M.
- $cha(T, N, M)$ indica que M es el método ejecutado cuando N se invoca sobre un objeto de tipo T. ¡V-Tables!

UNIVERSIDAD SIMÓN BOLÍVAR

Don't panic!

Comenzamos por unos EDB fáciles de calcular

- $actual(S, I, V)$ indica que V es el I-ésimo parámetro actual pasado en el punto de llamada S.
- $formal(M, I, V)$ indica que V es el I-ésimo parámetro formal declarado en el método M.
- $cha(T, N, M)$ indica que M es el método ejecutado cuando N se invoca sobre un objeto de tipo T. ¡V-Tables!

Let's cha-cha...
(en realidad es Class Hierarchy Analysis)

UNIVERSIDAD SIMÓN BOLÍVAR

When you see it...

...you'll recurse bricks!

```
invokes(S,M) :- "S : V.N(...)",
                pts(V,H),
                hType(H,T),
                cha(T,N,M)

pts(V,H) :- invokes(S,M),
            formal(M,I,V),
            actual(S,I,W),
            pts(W,H)
```

UNIVERSIDAD SIMÓN BOLÍVAR

When you see it...

...you'll recurse bricks!

```
invokes(S,M) :- "S : V.N(...)",
                pts(V,H),
                hType(H,T),
                cha(T,N,M)

pts(V,H) :- invokes(S,M),
            formal(M,I,V),
            actual(S,I,W),
            pts(W,H)
```

1. Destino del método – si V apunta a un objeto H de tipo T, y para el tipo T se ejecuta M cuando se invoca N, entonces definitivamente el sitio S podría estar ejecutando a M.

UNIVERSIDAD SIMÓN BOLÍVAR

When you see it...

...you'll recurse bricks!

```
invokes(S,M) :- "S : V.N(...)",
                pts(V,H),
                hType(H,T),
                cha(T,N,M)

pts(V,H) :- invokes(S,M),
            formal(M,I,V),
            actual(S,I,W),
            pts(W,H)
```

1. Destino del método – si V apunta a un objeto H de tipo T, y para el tipo T se ejecuta M cuando se invoca N, entonces definitivamente el sitio S podría estar ejecutando a M.
2. La llamada es «completa» – si en el sitio S se ejecuta M, entonces el I-ésimo actual W se asignará con el I-ésimo formal V, así que V podría apuntar a W.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cuál es el resultado?

Y por favor dime que es suficiente...

- La combinación de reglas co-recursivas permitió calcular de manera insensible al contexto, tanto el análisis «points-to» como deducir el grafo de llamadas.
- El grafo de llamadas se calculó de manera insensible al flujo pero es mucho más preciso que si se hubieran usado sólo las declaraciones y la forma sintáctica.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cuál es el resultado?

Y por favor dime que es suficiente...

- La combinación de reglas co-recursivas permitió calcular de manera insensible al contexto, tanto el análisis «points-to» como deducir el grafo de llamadas.
- El grafo de llamadas se calculó de manera insensible al flujo pero es mucho más preciso que si se hubieran usado sólo las declaraciones y la forma sintáctica.
- Desafortunadamente, si el lenguaje provee «reflexión» o «introspección», no es suficiente:

```
String className = ...;
Class c = Class.forName(className);
Object o = c.newInstance();
T t = (T) o;
```

UNIVERSIDAD SIMÓN BOLÍVAR

Agregando contexto

Lámina corta porque «ya está bueno ya»

- Necesitamos derivar un grafo con clonación –
  cada método M necesitará contexto C en el programa.
- $csInvokes(S, C, M, D)$ si el punto de llamada S en el contexto C invoca al método M en contexto D.

UNIVERSIDAD SIMÓN BOLÍVAR

Agregando contexto

El subtítulo anterior miente

```
pts(V,C,H) :- "H : T V = new T()",
              csInvokes(H,C,_,_)
pts(V,C,H) :- "V = W",
              pts(W,C,H)
pts(V,C,H) :- "V = W.F",
              pts(W,C,G),
              htps(G,F,H)
pts(V,D,H) :- csInvokes(S,C,M,D),
              formal(M,I,V),
              actual(S,I,W),
              pts(W,C,H)
hpts(H,F,G) :- "V.F = W",
               pts(W,C,G),
               pts(V,C,H)
```

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*] (Segunda Edición)
  - Secciones 12.3 a 12.6
  - En la especificación para llamada a métodos, falta el IDB para los efectos del valor de retorno. Escríbala.
  - Ejercicios 12.3.2 a 12.3.7, 12.5.1 y 12.6.1

UNIVERSIDAD SIMÓN BOLÍVAR