Análisis inter-procedimientos

CI4722 – Lenguajes de Programación III

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis entre procedimientos

- Análisis intra-procedimientos es simple e impreciso en muchos casos.
  - Sólo se contempla un procedimiento a la vez.
  - Ignora los efectos entre procedimientos.
  - Conservadores – presumen que se altera «cualquier cosa».

Análisis entre procedimientos

- Análisis intra-procedimientos es simple e impreciso en muchos casos.
  - Sólo se contempla un procedimiento a la vez.
  - Ignora los efectos entre procedimientos.
  - Conservadores – presumen que se altera «cualquier cosa».
- Analizar el flujo de control entre procedimientos.
  - Información fluye entre llamador y llamado.
  - Análisis de aliases, en particular apuntadores a subrutinas, permiten mejorar el despacho indirecto.

Análisis Global – Influencia Local.

UNIVERSIDAD SIMÓN BOLÍVAR

Grafo de Llamadas

¿Cuál procedimiento llama a cuál(es) otro(s)?

- Conjunto de nodos y aristas tales que:
  - Un nodo por cada procedimiento del programa.
  - Un nodo por cada **punto de llamada** (*call site*) – puntos en los cuales hay llamadas a procedimientos.
  - Si el punto de llamada c *podría* llamar al procedimiento p, entonces hay una arista $c \rightarrow p$.

UNIVERSIDAD SIMÓN BOLÍVAR

Grafo de Llamadas

¿Cuál procedimiento llama a cuál(es) otro(s)?

- Conjunto de nodos y aristas tales que:
  - Un nodo por cada procedimiento del programa.
  - Un nodo por cada **punto de llamada** (*call site*) – puntos en los cuales hay llamadas a procedimientos.
  - Si el punto de llamada c *podría* llamar al procedimiento p, entonces hay una arista $c \rightarrow p$.
- Si el lenguaje sólo permite llamadas directas:
  - Destino de cada llamada calculado estáticamente.
  - Grafo de llamadas trivial de construir.

UNIVERSIDAD SIMÓN BOLÍVAR

Grafo de Llamadas

¿Cuál procedimiento llama a cuál(es) otro(s)?

- Conjunto de nodos y aristas tales que:
  - Un nodo por cada procedimiento del programa.
  - Un nodo por cada **punto de llamada** (*call site*) – puntos en los cuales hay llamadas a procedimientos.
  - Si el punto de llamada c *podría* llamar al procedimiento p, entonces hay una arista $c \rightarrow p$.
- Si el lenguaje sólo permite llamadas directas:
  - Destino de cada llamada calculado estáticamente.
  - Grafo de llamadas trivial de construir.
- Si el lenguaje permite llamadas indirectas:
  - Destino de cada llamada calculado dinámicamente.
  - Grafo incluye varias aristas desde el mismo punto de llamada.

UNIVERSIDAD SIMÓN BOLÍVAR

Grafo de llamadas indirectas

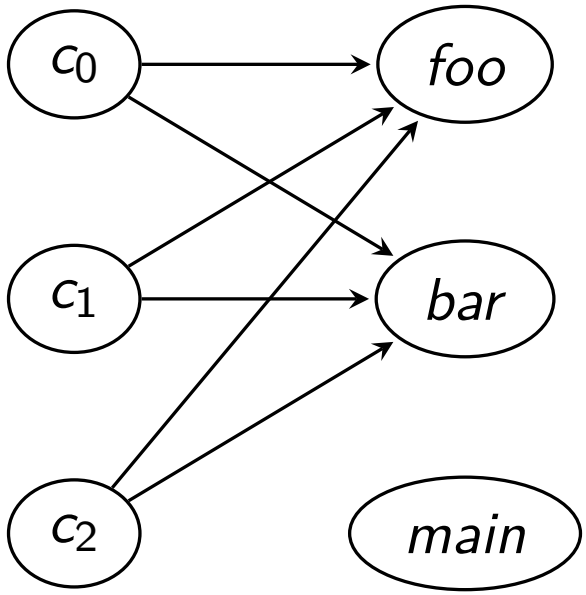
Aplica en métodos virtuales, clausuras y apuntadores.

```
   int (*pf)(int);          /* Apuntador a función int -> i

   int foo(int x) {
     if (x < 10)
c0:    return (*pf)(x+1);
     else
       return x;
   }
   int bar(int y) {
     pf = &foo;             /* pf apunta a ‘foo‘ */
c1:  return (*pf)(y);
   }
   void main() {
     pf = &bar;             /* pf apunta a ‘bar‘ */
c2:  (*pf)(42);
   }
```

¿A qué *puede* estar apuntando `pf`?

Análisis conservador

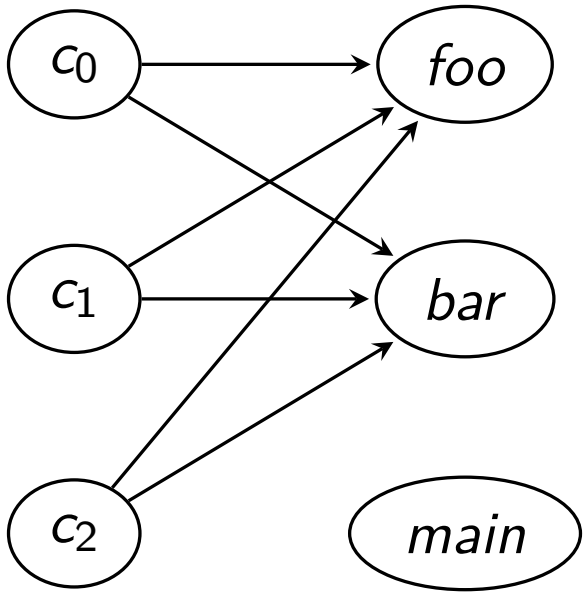


- Sólo observando los tipos de las funciones:
  - `pf` apunta a función `int -> int`.
  - `foo` y `bar` son funciones `int -> int`.
  - `main` no es función `int -> int`.
- Cada llamada a través de `*pf` *podría* invocar `foo` o `bar`

UNIVERSIDAD SIMÓN BOLÍVAR

¿A qué *puede* estar apuntando `pf`?

Análisis más cuidadoso

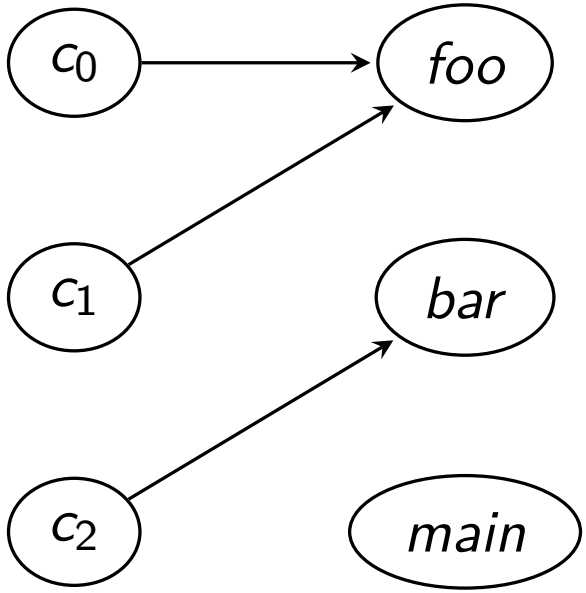


- Observando lo que se asigna al apuntador – sin importar cuándo:
  - `main` pone `pf` a apuntar a `bar`.
  - `bar` pone `pf` a apuntar a `foo`.
  - `pf` nunca es puesto a apuntar a `main`.
- Cada llamada a través de `*pf` *podría* invocarse `foo` o `bar`
- Es el mismo resultado – pues no nos importa *cuándo* se asigna.

UNIVERSIDAD SIMÓN BOLÍVAR

¿A qué *puede* estar apuntando `pf`?

Análisis «Con Mucho Cuidado»



- Observando lo que se asigna al apuntador y cuándo:
  - En `c2`, `pf` sólo puede apuntar a `bar`.
  - En `c1`, `pf` sólo puede apuntar a `foo`.
  - Como `foo` *nunca* cambia a `pf`, en `c0` `pf` sólo puede apuntar a `foo`.
- Obtenemos un grafo de llamadas mucho más preciso.

UNIVERSIDAD SIMÓN BOLÍVAR

Referencias o apuntadores a funciones

- Necesaria una *aproximación estática*:
  - Procedimientos *potencialmente* invocados.
  - Tipos de los parámetros empleados.
  - Tipo del objeto invocante – en lenguajes orientados a objetos.
- Análisis iterativo
  - Base – destinos observables estáticamente.
  - Inducción – agregar destinos luego de análisis de tipos y contexto
  - Final – convergencia al no poder agregar más destinos.

Análisis Inter-Procedimientos
permite encontrar una solución aproximada.

UNIVERSIDAD SIMÓN BOLÍVAR

Sensibilidad al contexto

- Cada llamada es diferente según el contexto –
  ¡por eso es que hacemos procedimientos!
- Quisiéramos identificar llamadas «iguales» o «diferentes», para conocer su influencia en el contexto.

UNIVERSIDAD SIMÓN BOLÍVAR

Sensibilidad al contexto

```
    for (i = 0; i < n; i++) {
c1:     t1    = f(0);
c2:     t2    = f(243);
c3:     t3    = f(243);
        X[i] = t1 + t2 + t3
    }

    int f(int v) {
        return (v+1);
    }
```

- Quisiéramos que el análisis:
  1. Notara que `f` siempre se invoca con constantes.
  2. Por tanto, `f` siempre retornará una constante.
  3. Es más, `X[i]` recibe una constante – ¡qué siempre es la misma!
- El análisis debe «reconocer» el contexto de cada llamada.

UNIVERSIDAD SIMÓN BOLÍVAR

Ser insensibles a la sensibilidad

Wait, what?

- Considerar cada llamada como un `goto`.
- El grafo de llamadas tiene aristas adicionales:
  - Entre el punto de llamadas y el inicio del procedimiento.
  - Entre cada `return` y la posición de retorno.
- Agregamos asignaciones explícitas:
  - De cada parámetro actual a cada parámetro formal.
  - Del valor de retorno a la variable que lo recibe.
- Las llamadas se volvieron «un procedimiento» –
  se analiza con técnicas intra-procedimiento.

UNIVERSIDAD SIMÓN BOLÍVAR

Ser insensibles a la sensibilidad

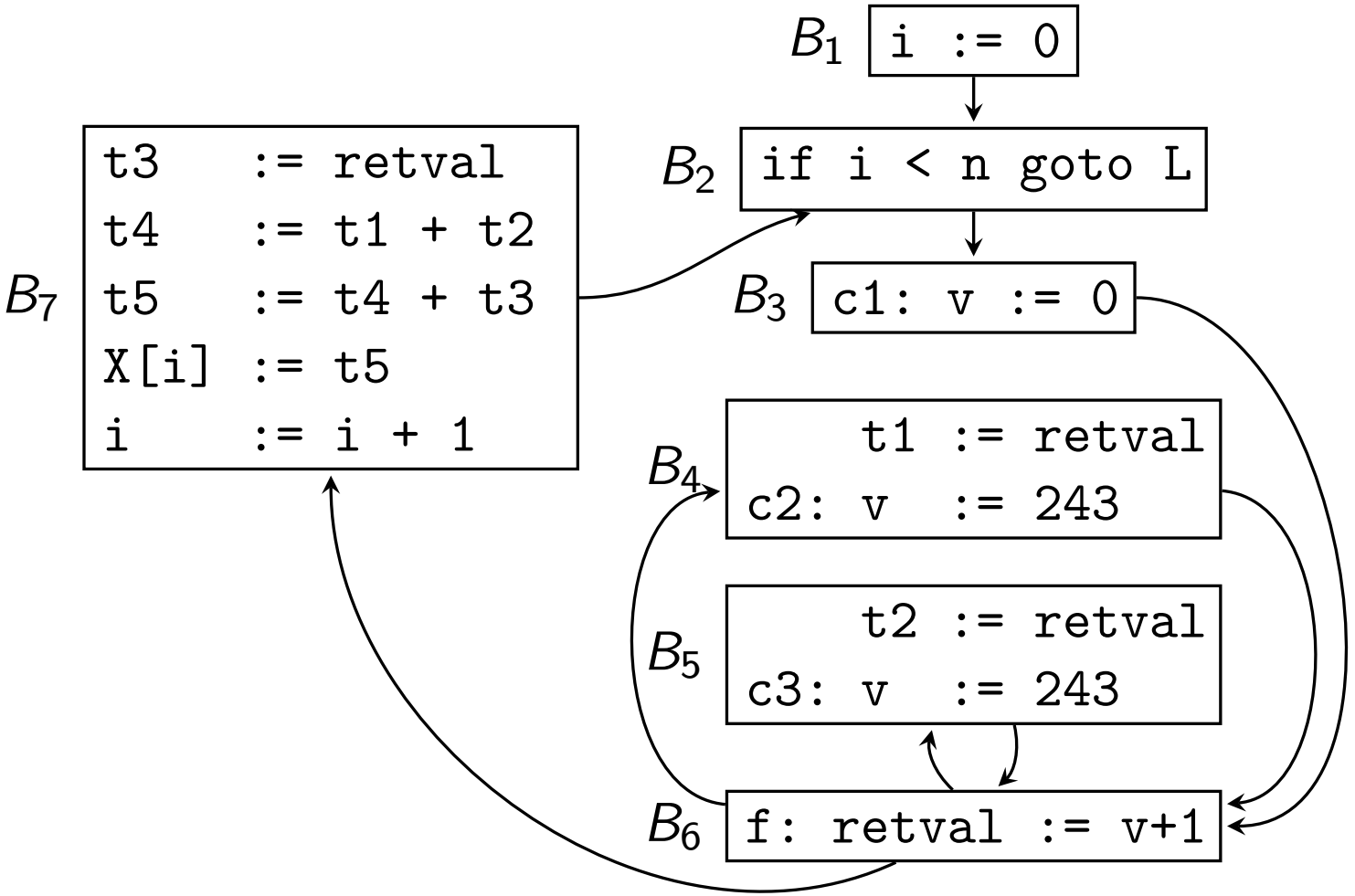
Wait, what?

- Considerar cada llamada como un `goto`.
- El grafo de llamadas tiene aristas adicionales:
  - Entre el punto de llamadas y el inicio del procedimiento.
  - Entre cada `return` y la posición de retorno.
- Agregamos asignaciones explícitas:
  - De cada parámetro actual a cada parámetro formal.
  - Del valor de retorno a la variable que lo recibe.
- Las llamadas se volvieron «un procedimiento» –
  se analiza con técnicas intra-procedimiento.

Análisis Insensible al Contexto
simple pero muy impreciso.

UNIVERSIDAD SIMÓN BOLÍVAR

De llamadas a procedimiento



UNIVERSIDAD SIMÓN BOLÍVAR

Cadena de Llamadas

```
    for (i = 0; i < n; i++) {
c1:     t1    = g(0);
c2:     t2    = g(243);
c3:     t3    = g(243);
        X[i] = t1 + t2 + t3
    }
    int g(int v) {
c4:     return f(v)
    }
    int f(int v) {
        return (v+1);
    }
```

- Contexto definido por pila de ejecución – la **cadena de llamadas**.
- Tres cadenas de llamada para `f` – $(c_1 c_4)$, $(c_2 c_4)$, $(c_3 c_4)$

UNIVERSIDAD SIMÓN BOLÍVAR

¿Y si hay recursión?

Cadenas «infinitas»

```
    for (i = 0; i < n; i++) {
c1:     t1    = g(0);
c2:     t2    = g(243);
c3:     t3    = g(243);
        X[i] = t1 + t2 + t3
    }
    int g(int v) {
        if (v > 1)
c4:       return g(v-1)
        else
c5:       return f(v)
    }
    int f(int v) {
        return (v+1);
    }
```

UNIVERSIDAD SIMÓN BOLÍVAR

¿Y si hay recursión?

```
    int g(int v) {
        if (v > 1)
c4:       return g(v-1)
        else
c5:       return f(v)
    }
```

- Si g es llamado con un c positivo, habrá c llamadas recursivas a g.
- En cada llamada, su argumento decrementa en uno.
- Si la cadena de llamada es $(c_2(c_4^n))$, entonces v es $243 - n$
- Cadenas de llamadas posibles para `f`:
  - $(c_1 c_5)$ – g invoca `f` inmediatamente, con 0.
  - $(c_2 c_4 \ldots c_4 c_5)$ – g invoca `f` después de 242 recursiones, con 1.
  - $(c_3 c_4 \ldots c_4 c_5)$ – g invoca `f` después de 242 recursiones, con 1.

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis de contexto $k-$limitado

- Usar los k puntos de llamada más recientes en lugar de la cadena completa de llamadas.
- Insensibilidad al contexto cuando k es cero.
- Puede simplificar algunos casos de análisis, pero hacerlo imposible en otros.
- Alternativa mixta:
  - Sensibilidad al contexto en cadenas de llamada acíclicas.
  - Colapsar las llamadas recursivas en una sola – (c_2, c_4^*, c_5).

Número de contextos puede ser exponencial en la cantidad de procedimientos.

UNIVERSIDAD SIMÓN BOLÍVAR

Sensibilidad por Clonación

Hello Dolly!

- Cada punto de interés genera una copia idéntica del invocado.
- Aplicar análisis insensible al contexto sobre el grafo con copias.
- No se copia el código en la práctica – sólo nombres diferentes.

UNIVERSIDAD SIMÓN BOLÍVAR

Call Cloning

```
    for (i = 0; i < n; i++) {
c1:     t1    = g1(0);
c2:     t2    = g2(243);
c3:     t3    = g3(243);
        X[i] = t1 + t2 + t3
    }
    int g1(int v) {
c41:    return f1(v)
    }
    int g2(int v) {
c42:    return f2(v)
    }
    int g3(int v) {
c43:    return f3(v)
    }
    ...
```

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis Resumido

Funciones de transferencia por procedimiento

- «Resumir» cada procedimiento encapsulando su comportamiento – qué se «observa» al invocarlo, obviando los detalles internos.
- Cada procedimiento se modela como una región:
  - Con un sólo punto de entrada.
  - Cada pareja llamador-llamado se estudia como exterior-interior.
- El análisis se completa en dos fases:
  1. Desde los procedimientos «hoja» hacia arriba – calcular una función de transferencia para cada procedimiento.
  2. Desde el programa principal hacia abajo – propagando la información de cada llamada hacia el llamado.

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis Resumido

Regulando la precisión del resultado

- Análisis completo sensible al contexto requiere propagar contexto diferentes preservando resultados.
  - Requiere algún tipo de «clonación».
  - Costoso en espacio y tiempo.
- Análisis parcial sensible al contexto requiere combinar contextos diferentes en uno sólo.
  - Establecer un operador de combinación razonable y conservador.
  - Muy eficiente, menos preciso.
- En presencia de recursión
  - Calcular componentes fuertemente conexas del grafo.
  - En la primera fase, no visitarlas hasta que *todos* sus sucesores hayan sido visitados.
  - Al visitar una componente fuertemente conexa, usar el método iterativo para calcular las funciones de transferencia.

UNIVERSIDAD SIMÓN BOLÍVAR

Casos de uso

¿Cuándo vale la pena esta complejidad?

- Invocación de métodos virtuales – `x.m()`
  - Determinar el `m` específico – *inlining*.
  - Requiere acceso a todo el código fuente.

Casos de uso

¿Cuándo vale la pena esta complejidad?

- Invocación de métodos virtuales – `x.m()`
  - Determinar el `m` específico – *inlining*.
  - Requiere acceso a todo el código fuente.
- Análisis de aliasing y apuntadores.

UNIVERSIDAD SIMÓN BOLÍVAR

Casos de uso

¿Cuándo vale la pena esta complejidad?

- Invocación de métodos virtuales – `x.m()`
  - Determinar el `m` específico – *inlining*.
  - Requiere acceso a todo el código fuente.
- Análisis de aliasing y apuntadores.
- Paralelismo
  - Identificar dependencias entre datos para simplificar ciclos.
  - Análisis intra-procedural encuentra la *mayoría* de las dependencias, pero nunca las encuentra todas.
  - Análisis inter-procedural puede encontrar todas las interesantes – aquellas relacionadas con ciclos paralelizables.

UNIVERSIDAD SIMÓN BOLÍVAR

Casos de uso

¿Cuándo vale la pena esta complejidad?

- Invocación de métodos virtuales – `x.m()`
  - Determinar el `m` específico – *inlining*.
  - Requiere acceso a todo el código fuente.
- Análisis de aliasing y apuntadores.
- Paralelismo
  - Identificar dependencias entre datos para simplificar ciclos.
  - Análisis intra-procedural encuentra la *mayoría* de las dependencias, pero nunca las encuentra todas.
  - Análisis inter-procedural puede encontrar todas las interesantes – aquellas relacionadas con ciclos paralelizables.
- Detección parcial de errores y vulnerabilidades.
  - Inyección de SQL.
  - Buffer overflows.
  - Incosistencia de usos – *locks*, *masks*

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*] (Segunda Edición)
  - Secciones 12.1 y 12.2
  - Ejercicios 12.1.1 y 12.1.2

UNIVERSIDAD SIMÓN BOLÍVAR