Análisis de Flujo

CI4722 – Lenguajes de Programación III

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

¿Qué hacer con aliases?

- Dos o más expresiones denotan la misma *dirección* de memoria se denominan **aliases**.
  - Los apuntadores introducen aliases explícitos.
  - Los procedimientos con pasaje de parámetro por referencia o copia-resultado introducen aliases implícitos.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Qué hacer con aliases?

- Dos o más expresiones denotan la misma *dirección* de memoria se denominan **aliases**.
  - Los apuntadores introducen aliases explícitos.
  - Los procedimientos con pasaje de parámetro por referencia o copia-resultado introducen aliases implícitos.
- El análisis de flujo de datos se complica por la necesidad de tomar decisiones conservadores en cuanto al código – esto es, a falta de información *precisa* y *completa*:
  - Asignaciones a través de apuntadores cambian cualquier variable.
  - Acceso a través de apuntadores consultan cualquier variable.

Muchas más variables vivas y definiciones activas de lo que hay en realidad.

UNIVERSIDAD SIMÓN BOLÍVAR

Un lenguaje con apuntadores

Muy simple, pero ilustra los problemas y soluciones

- Consideremos un lenguaje tal que
  - Dispone de enteros y reales de cuatro bytes cada uno.
  - Arreglos de estos tipos primitivos.
  - Apuntadores a los tipos básicos y a los arreglos – no hay apuntadores hacia apuntadores.
- Nos bastará saber que un apuntador p apunta hacia *algún* elemento de un arreglo a sin importar cuál elemento en particular.
  - Típicamente los apuntadores son "cursores" sobre arreglos.
  - El análisis conservador siempre terminará diciendo que apunta sobre *algún* elemento del arreglo – ¿para qué complicarnos?

UNIVERSIDAD SIMÓN BOLÍVAR

Aritmética sobre nuestros apuntadores

- Si p apunta a un tipo básico
  - Sumarle o restarle un entero *puede* producir un *entero*.
  - *Nunca* será considerado un apuntador.

Aritmética sobre nuestros apuntadores

- Si `p` apunta a un tipo básico
  - Sumarle o restarle un entero *puede* producir un *entero*.
  - *Nunca* será considerado un apuntador.
- Si `p` apunta a un arreglo
  - Sumarle o restarle un entero produce un apuntador que posiblemente apunte a otra posición dentro del mismo arreglo.
  - Cualquier otra operación *nunca* produce un apuntador.

UNIVERSIDAD SIMÓN BOLÍVAR

Aritmética sobre nuestros apuntadores

- Si p apunta a un tipo básico
  - Sumarle o restarle un entero *puede* producir un *entero*.
  - *Nunca* será considerado un apuntador.
- Si p apunta a un arreglo
  - Sumarle o restarle un entero produce un apuntador que posiblemente apunte a otra posición dentro del mismo arreglo.
  - Cualquier otra operación *nunca* produce un apuntador.
- Mover un apuntador de un arreglo a otro mediante aritmética, aunque posible, es demencial y no lo contemplaremos.
  - Depende de una implantación que asegure contigüidad.
  - Un optimizador de código sólo debe basarse en las expresiones de alto nivel para decidir cómo mejorarlas.

UNIVERSIDAD SIMÓN BOLÍVAR

Aritmética sobre nuestros apuntadores

- Si p apunta a un tipo básico
  - Sumarle o restarle un entero *puede* producir un *entero*.
  - *Nunca* será considerado un apuntador.
- Si p apunta a un arreglo
  - Sumarle o restarle un entero produce un apuntador que posiblemente apunte a otra posición dentro del mismo arreglo.
  - Cualquier otra operación *nunca* produce un apuntador.
- Mover un apuntador de un arreglo a otro mediante aritmética, aunque posible, es demencial y no lo contemplaremos.
  - Depende de una implantación que asegure contigüidad.
  - Un optimizador de código sólo debe basarse en las expresiones de alto nivel para decidir cómo mejorarlas.
- Nuestras suposiciones simplifican la identificación de apuntadores:
  - Los símbolos declarados como tales en el programa.
  - Temporales asignadas con un apuntador, más una constante.

UNIVERSIDAD SIMÓN BOLÍVAR

Efectos del uso de apuntadores

¿A qué puede apuntar `p`?

Sean `p` y `q` apuntadores. Sea `c` una constante entera.

Efectos del uso de apuntadores

¿A qué puede apuntar `p`?

Sean `p` y `q` apuntadores. Sea `c` una constante entera.

1. Después de `p := &x`
   - `p` apunta *solamente* a `x`.
   - Si `x` es un arreglo, entonces `p` *podría* continuar apuntando a `x` después de una instrucción `p := &x + c`

UNIVERSIDAD SIMÓN BOLÍVAR

Efectos del uso de apuntadores

¿A qué puede apuntar `p`?

Sean `p` y `q` apuntadores. Sea `c` una constante entera.

1. Después de `p := &x`
   - `p` apunta *solamente* a `x`.
   - Si `x` es un arreglo, entonces `p` *podría* continuar apuntando a `x` después de una instrucción `p := &x + c`
2. Después de `p := q + c`, si `q` *podía* apuntar a un arreglo, `p` también *podría* estar apuntándole.

UNIVERSIDAD SIMÓN BOLÍVAR

Efectos del uso de apuntadores

¿A qué puede apuntar `p`?

Sean `p` y `q` apuntadores. Sea `c` una constante entera.

1. Después de `p := &x`
   - `p` apunta *solamente* a `x`.
   - Si `x` es un arreglo, entonces `p` *podría* continuar apuntando a `x` después de una instrucción `p := &x + c`
2. Después de `p := q + c`, si `q` *podía* apuntar a un arreglo, `p` también *podría* estar apuntándole.
3. Después de `p := q`, `p` *puede* apuntar a lo mismo que apuntaba `q`.

UNIVERSIDAD SIMÓN BOLÍVAR

Efectos del uso de apuntadores

¿A qué puede apuntar `p`?

Sean `p` y `q` apuntadores. Sea `c` una constante entera.

1. Después de `p := &x`
   - `p` apunta *solamente* a `x`.
   - Si `x` es un arreglo, entonces `p` *podría* continuar apuntando a `x` después de una instrucción `p := &x + c`
2. Después de `p := q + c`, si `q` *podía* apuntar a un arreglo, `p` también *podría* estar apuntándole.
3. Después de `p := q`, `p` *puede* apuntar a lo mismo que apuntaba `q`.
4. Asignar a `p` un valor que no sea apuntador, hace que apunte a nada.

UNIVERSIDAD SIMÓN BOLÍVAR

Efectos del uso de apuntadores

¿A qué puede apuntar `p`?

Sean `p` y `q` apuntadores. Sea `c` una constante entera.

1. Después de `p := &x`
   - `p` apunta *solamente* a `x`.
   - Si `x` es un arreglo, entonces `p` *podría* continuar apuntando a `x` después de una instrucción `p := &x + c`
2. Después de `p := q + c`, si `q` *podía* apuntar a un arreglo, `p` también *podría* estar apuntándole.
3. Después de `p := q`, `p` *puede* apuntar a lo mismo que apuntaba `q`.
4. Asignar a `p` un valor que no sea apuntador, hace que apunte a nada.
5. Asignaciones a otras variables no alteran el conjunto de objetos a los cuales apunta `p` – supone que no hay apuntadores hacia apuntadores.

UNIVERSIDAD SIMÓN BOLÍVAR

Hacia un problema de Análisis de Flujo

- Para cada bloque B calcularemos un conjunto de pares (p, x)
  - p es un apuntador, x es una variable
  - p *podría* apuntar a x.

Hacia un problema de Análisis de Flujo

- Para cada bloque B calcularemos un conjunto de pares (p, x)
  - p es un apuntador, x es una variable
  - p *podría* apuntar a x.
- ¿Cuál es el efecto de un bloque B sobre el conjunto?
  - Las reglas establecen el cambio causado por las instrucciones.
  - Los cambios ocurren a medida que se avanza en el programa.

Construyamos la función de transferencia.

UNIVERSIDAD SIMÓN BOLÍVAR

La función de transferencia

La más complicada hasta ahora

1. Asignación a un apuntador usando un arreglo `x` –
   - Apuntar a la base de un arreglo

     `p := &x`
   - Apuntar a un desplazamiento de la base

     `p := &x + c`

La función de transferencia

La más complicada hasta ahora

1. Asignación a un apuntador usando un arreglo `x` –
   - Apuntar a la base de un arreglo

     `p := &x`
   - Apuntar a un desplazamiento de la base

     `p := &x + c`
   - Invalidar cualquier referencia previa posible via `p`.
   - Incorporar la posible referencia de `p` a `x`.

$$f_s(x) = (x - \{(p, b) | \forall b \text{ variable}\}) \cup \{(p, x)\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

La función de transferencia

La más complicada hasta ahora

2. Copia de un apuntador con desplazamiento

   `p := q + c`

La función de transferencia

La más complicada hasta ahora

3. Copia de un apuntador con desplazamiento

```
p := q + c
```

   - Invalidar cualquier referencia previa posible via `p`.
   - Incorporar la posible referencia de `p` a cualquier referencia via `q`, siempre que sean referencias hacia un arreglo.

$$f_s(x) = (x - \{(p, b) | \forall b \text{ variable}\}) \cup \{(p, b) | (q, b) \in x \wedge b \text{ es arreglo}\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

La función de transferencia

La más complicada hasta ahora

4. Copia simple de apuntadores

   `p := q`

La función de transferencia

La más complicada hasta ahora

5. Copia simple de apuntadores

   `p := q`

   - Invalidar cualquier referencia previa posible via `p`.
   - Incorporar la posible referencia de `p` a cualquier referencia via `q`.

$$f_s(x) = (x - \{(p, b) | \forall b \text{ variable}\}) \cup \{(p, b) | (q, b) \in x\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

La función de transferencia

La más complicada hasta ahora

6. Asignar cualquier cosa a un apuntador `p`

La función de transferencia

La más complicada hasta ahora

6. Asignar cualquier cosa a un apuntador `p`
   - Invalidar cualquier referencia previa posible via `p`.

$$f_s(x) = (x - \{(p, b) | \forall b \text{ variable}\})$$

La función de transferencia

La más complicada hasta ahora

6. Asignar cualquier cosa a un apuntador `p`
   - Invalidar cualquier referencia previa posible via `p`.

$$f_s(x) = (x - \{(p, b) | \forall b \text{ variable}\})$$

7. Cualquier instrucción que no sea asignación a un apuntador

$$f_s(x) = x$$

$f_B(x)$ es la composición de ls $f_s(x)$ para cada instrucción

UNIVERSIDAD SIMÓN BOLÍVAR

Es un problema de Análisis de Flujo

- Dominio – tuplas (p, x) de apuntadores y variables.
- Dirección – *forward flow*.
- Base – $OUT[ENTRY] = \emptyset$
- Función de Transferencia $f_B(x)$ definida previamente.
- Inicialización – $OUT[B] = \emptyset$
- Ecuaciones

$$OUT[B] = f_B(IN[B])$$

$$IN[B] = \bigcup_{P \text{ predecesor de } B} OUT[P]$$

UNIVERSIDAD SIMÓN BOLÍVAR

Aprovechando la información calculada

Refinar otros problemas de análisis

- Dado un apuntador `p` sabemos a cuales variables *podría* estar apuntando – ¿cómo aprovechar ese dato?

UNIVERSIDAD SIMÓN BOLÍVAR

Aprovechando la información calculada

Refinar otros problemas de análisis

- Dado un apuntador `p` sabemos a cuales variables *podría* estar apuntando – ¿cómo aprovechar ese dato?
- Definiciones Activas – ¿cuál es el efecto de `*p := a`?
  - Genera una definición para todo lo que *podría* estar apuntado por `p`.
  - Mata a `b` sólo si `b` **no** es arreglo y es lo **único** a lo que apunta `p`.

UNIVERSIDAD SIMÓN BOLÍVAR

Aprovechando la información calculada

Refinar otros problemas de análisis

- Dado un apuntador `p` sabemos a cuales variables *podría* estar apuntando – ¿cómo aprovechar ese dato?
- Definiciones Activas – ¿cuál es el efecto de `*p := a`?
  - Genera una definición para todo lo que *podría* estar apuntado por `p`.
  - Mata a `b` sólo si `b` **no** es arreglo y es lo **único** a lo que apunta `p`.
- Variables Vivas – ¿cuál es el efecto de `*p := a`?
  - Usa ambos valores.
  - Define a `b` sólo si `b` es lo **único** a lo que apunta `p`.

UNIVERSIDAD SIMÓN BOLÍVAR

Aprovechando la información calculada

Refinar otros problemas de análisis

- Dado un apuntador `p` sabemos a cuales variables *podría* estar apuntando – ¿cómo aprovechar ese dato?
- Definiciones Activas – ¿cuál es el efecto de `*p := a`?
  - Genera una definición para todo lo que *podría* estar apuntado por `p`.
  - Mata a `b` sólo si `b` **no** es arreglo y es lo **único** a lo que apunta `p`.
- Variables Vivas – ¿cuál es el efecto de `*p := a`?
  - Usa ambos valores.
  - Define a `b` sólo si `b` es lo **único** a lo que apunta `p`.
- Variables Vivas – ¿cuál es el efecto de `a := *p`?
  - Usa ambos valores, además de cualquier `b` al cual `p` podría apuntar.
  - Es una definición de `a`.

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis entre procedimientos

- Los programas son una colección de varios diagramas de flujo, relacionados entre sí según las llamadas entre procedimientos.
- El análisis de flujo entre procedimientos debe contemplar la forma en que las definiciones y usos de variables se ven afectados por cada procedimiento.
- Parámetros de procedimientos introducen alias

UNIVERSIDAD SIMÓN BOLÍVAR

Modelo de análisis con procedimientos

- Supongamos un lenguaje que permite procedimientos
  - Recursivos.
  - Alcance global y estático.
  - Sin alcance en bloques anidados.
  - Pasaje de parámetros por *referencia*.
- Supondremos que los diagramas de flujo para cada procedimiento
  - Tienen un único punto de entrada y un único punto de salida.
  - Cualquier nodo está en al menos un camino entre el nodo de entrada y el nodo de salida.

UNIVERSIDAD SIMÓN BOLÍVAR

`p` llama a `q`

- Durante el análisis de `p` es posible encotrar una llamada `q(u,v)`.
- Para Definiciones Activas o Expresiones Disponibles necesitamos saber cuáles variables *pueden* ser cambiadas por `q`.

UNIVERSIDAD SIMÓN BOLÍVAR

`p` llama a `q`

- Durante el análisis de `p` es posible encotrar una llamada `q(u,v)`.
- Para Definiciones Activas o Expresiones Disponibles necesitamos saber cuáles variables *pueden* ser cambiadas por `q`.
  - `q` puede definir sus locales – esto no influye en `p` incluso si `q = p`.

UNIVERSIDAD SIMÓN BOLÍVAR

`p` llama a `q`

- Durante el análisis de `p` es posible encotrar una llamada `q(u,v)`.
- Para Definiciones Activas o Expresiones Disponibles necesitamos saber cuáles variables *pueden* ser cambiadas por `q`.
  - `q` puede definir sus locales – esto no influye en `p` incluso si `q = p`.
  - `q` puede definir alguna global – fácil determinarlo mirando el cuerpo de `q` o llamadas hechas por `q`.

UNIVERSIDAD SIMÓN BOLÍVAR

`p` llama a `q`

- Durante el análisis de `p` es posible encotrar una llamada `q(u,v)`.
- Para Definiciones Activas o Expresiones Disponibles necesitamos saber cuáles variables *pueden* ser cambiadas por `q`.
  - `q` puede definir sus locales – esto no influye en `p` incluso si `q = p`.
  - `q` puede definir alguna global – fácil determinarlo mirando el cuerpo de `q` o llamadas hechas por `q`.
  - `q` puede definir `u` o `v` directamente – fácil determinarlo mirando el cuerpo de `q`.

UNIVERSIDAD SIMÓN BOLÍVAR

p llama a q

- Durante el análisis de p es posible encotrar una llamada q(u,v).
- Para Definiciones Activas o Expresiones Disponibles necesitamos saber cuáles variables *pueden* ser cambiadas por q.
  - q puede definir sus locales – esto no influye en p incluso si q = p.
  - q puede definir alguna global – fácil determinarlo mirando el cuerpo de q o llamadas hechas por q.
  - q puede definir u o v directamente – fácil determinarlo mirando el cuerpo de q.
  - q puede definir u o v indirectamente – fácil determinarlo mirando las llamadas hechas por q.

UNIVERSIDAD SIMÓN BOLÍVAR

p llama a q

- Durante el análisis de p es posible encotrar una llamada q(u,v).
- Para Definiciones Activas o Expresiones Disponibles necesitamos saber cuáles variables *pueden* ser cambiadas por q.
  - q puede definir sus locales – esto no influye en p incluso si q = p.
  - q puede definir alguna global – fácil determinarlo mirando el cuerpo de q o llamadas hechas por q.
  - q puede definir u o v directamente – fácil determinarlo mirando el cuerpo de q.
  - q puede definir u o v indirectamente – fácil determinarlo mirando las llamadas hechas por q.

¡Puede haber aliases que no hemos tomado en cuenta!

UNIVERSIDAD SIMÓN BOLÍVAR

Un método para encontrar aliases

Calcular una relación de equivalencia

- Relación de equivalencia "*puede* ser alias de"
- No distinguimos entre ocurrencias de una variable en llamadas diferentes a un mismo procedimiento – distinguimos variables locales con mismo nombre en diferentes procedimientos.
- Si determinamos que dos variables pueden ser aliases en algún fragmento del programa, supondremos que lo son en *todo* el programa.

UNIVERSIDAD SIMÓN BOLÍVAR

Estimación de aliases

Un algoritmo simple

1. Renombrar variables según sea necesario hasta que
   - Parámetros de procedimientos tengan nombres diferentes entre si.
   - Locales de procedimientos tengan nombres diferentes entre si.
   - Parámetros y locales tengan nombres diferentes entre si.

UNIVERSIDAD SIMÓN BOLÍVAR

Estimación de aliases

Un algoritmo simple

1. Renombrar variables según sea necesario hasta que
   - Parámetros de procedimientos tengan nombres diferentes entre si.
   - Locales de procedimientos tengan nombres diferentes entre si.
   - Parámetros y locales tengan nombres diferentes entre si.
2. Si existe el procedimiento $p(x_1, x_2, \ldots, x_n)$ y tenemos la llamada $p(y_1, y_2, \ldots, y_n)$, entonces $\forall i, x_i \equiv y_i$

UNIVERSIDAD SIMÓN BOLÍVAR

Estimación de aliases

Un algoritmo simple

1. Renombrar variables según sea necesario hasta que
   - Parámetros de procedimientos tengan nombres diferentes entre si.
   - Locales de procedimientos tengan nombres diferentes entre si.
   - Parámetros y locales tengan nombres diferentes entre si.
2. Si existe el procedimiento $p(x_1, x_2, \ldots, x_n)$ y tenemos la llamada $p(y_1, y_2, \ldots, y_n)$, entonces $\forall i, x_i \equiv y_i$
3. Calcular la clausura reflexiva y transitiva de la correspondencia
   - Si $x \equiv y$, agregar $y \equiv x$.
   - Si $x \equiv y \wedge y \equiv z$, agregar $x \equiv z$.

UNIVERSIDAD SIMÓN BOLÍVAR

Estimación de aliases

Un algoritmo simple

1. Renombrar variables según sea necesario hasta que
   - Parámetros de procedimientos tengan nombres diferentes entre si.
   - Locales de procedimientos tengan nombres diferentes entre si.
   - Parámetros y locales tengan nombres diferentes entre si.
2. Si existe el procedimiento $p(x_1, x_2, \ldots, x_n)$ y tenemos la llamada $p(y_1, y_2, \ldots, y_n)$, entonces $\forall i, x_i \equiv y_i$
3. Calcular la clausura reflexiva y transitiva de la correspondencia
   - Si $x \equiv y$, agregar $y \equiv x$.
   - Si $x \equiv y \wedge y \equiv z$, agregar $x \equiv z$.

Si el sistema de tipos es bueno, los conjuntos son pequeños

UNIVERSIDAD SIMÓN BOLÍVAR

Ejemplo de Estimación de Aliases

```
global g, h
proc main()
  local i
  g := ...
  foo(h,i)
end
proc foo(w,x)
  x := ...
  bar(w,w)
  bar(g,x)
end
proc bar(y,z)
  local k
  h := ...
  foo(k,y)
end
```

UNIVERSIDAD SIMÓN BOLÍVAR

Ejemplo de Estimación de Aliases

- `main` llama a `foo` – $\texttt{h} \equiv \texttt{w}$, $\texttt{i} \equiv \texttt{x}$

```
global g, h
proc main()
  local i
  g := ...
  foo(h,i)
end
proc foo(w,x)
  x := ...
  bar(w,w)
  bar(g,x)
end
proc bar(y,z)
  local k
  h := ...
  foo(k,y)
end
```

UNIVERSIDAD SIMÓN BOLÍVAR

Ejemplo de Estimación de Aliases

- `main` llama a `foo` –
  `h` $\equiv$ `w`, `i` $\equiv$ `x`
- Primera llamada de `foo` a `bar` –
  `w` $\equiv$ `y`, `w` $\equiv$ `z`

```
global g, h
proc main()
  local i
  g := ...
  foo(h,i)
end
proc foo(w,x)
  x := ...
  bar(w,w)
  bar(g,x)
end
proc bar(y,z)
  local k
  h := ...
  foo(k,y)
end
```

UNIVERSIDAD SIMÓN BOLÍVAR

Ejemplo de Estimación de Aliases

- `main` llama a `foo` – $\texttt{h} \equiv \texttt{w}$, $\texttt{i} \equiv \texttt{x}$
- Primera llamada de `foo` a `bar` – $\texttt{w} \equiv \texttt{y}$, $\texttt{w} \equiv \texttt{z}$
- Segunda llamada de `foo` a `bar` – $\texttt{g} \equiv \texttt{y}$, $\texttt{x} \equiv \texttt{z}$

```
global g, h
proc main()
  local i
  g := ...
  foo(h,i)
end
proc foo(w,x)
  x := ...
  bar(w,w)
  bar(g,x)
end
proc bar(y,z)
  local k
  h := ...
  foo(k,y)
end
```

UNIVERSIDAD SIMÓN BOLÍVAR

Ejemplo de Estimación de Aliases

- `main` llama a `foo` –
  `h` $\equiv$ `w`, `i` $\equiv$ `x`
- Primera llamada de `foo` a `bar` –
  `w` $\equiv$ `y`, `w` $\equiv$ `z`
- Segunda llamada de `foo` a `bar` –
  `g` $\equiv$ `y`, `x` $\equiv$ `z`
- `bar` llama a `foo` –
  `k` $\equiv$ `w`, `y` $\equiv$ `x`

```
global g, h
proc main()
  local i
  g := ...
  foo(h,i)
end
proc foo(w,x)
  x := ...
  bar(w,w)
  bar(g,x)
end
proc bar(y,z)
  local k
  h := ...
  foo(k,y)
end
```

UNIVERSIDAD SIMÓN BOLÍVAR

Ejemplo de Estimación de Aliases

- `main` llama a `foo` –
  $\texttt{h} \equiv \texttt{w}$, $\texttt{i} \equiv \texttt{x}$
- Primera llamada de `foo` a `bar` –
  $\texttt{w} \equiv \texttt{y}$, $\texttt{w} \equiv \texttt{z}$
- Segunda llamada de `foo` a `bar` –
  $\texttt{g} \equiv \texttt{y}$, $\texttt{x} \equiv \texttt{z}$
- `bar` llama a `foo` –
  $\texttt{k} \equiv \texttt{w}$, $\texttt{y} \equiv \texttt{x}$

Todos son aliases

```
global g, h
proc main()
  local i
  g := ...
  foo(h,i)
end
proc foo(w,x)
  x := ...
  bar(w,w)
  bar(g,x)
end
proc bar(y,z)
  local k
  h := ...
  foo(k,y)
end
```

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis de Flujo de Datos y Procedimientos

Idea general

- *change*[*p*] – globales o parámetros que pueden ser cambiados en una llamada a p *sin* tomar en cuenta los alias.
- *def*[*p*] – globales o parámetros que tienen definiciones explícitas en p.

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis de Flujo de Datos y Procedimientos

Idea general

- *change*[*p*] – globales o parámetros que pueden ser cambiados en una llamada a p *sin* tomar en cuenta los alias.
- *def* [*p*] – globales o parámetros que tienen definiciones explícitas en p.
- Los cambios efectuados por un procedimiento se calculan como

$$change[p] = def[p] \cup A \cup G$$

donde

- A es el conjunto de globales o parámetros de p, tales que son usadas como parámetros para q, llamada desde p, y pertenecen a *change*[*q*].
- G es el conjunto de globales en *change*[*q*], para todo q llamado por p.

UNIVERSIDAD SIMÓN BOLÍVAR

Análisis de Flujo de Datos y Procedimientos

Idea general

- *change*[*p*] – globales o parámetros que pueden ser cambiados en una llamada a p *sin* tomar en cuenta los alias.
- *def*[*p*] – globales o parámetros que tienen definiciones explícitas en p.
- Los cambios efectuados por un procedimiento se calculan como

$$change[p] = def[p] \cup A \cup G$$

  donde
  - A es el conjunto de globales o parámetros de p, tales que son usadas como parámetros para q, llamada desde p, y pertenecen a $change[q]$.
  - G es el conjunto de globales en $change[q]$, para todo q llamado por p.
- Algoritmo de punto fijo – converge rápido al inicializar con

$$change[p] = def[p]$$

UNIVERSIDAD SIMÓN BOLÍVAR

Eficiencia del algoritmo

Orden de visita a los procedimientos

- Visitar primero aquellos procedimientos que no llaman a otros.
  - Tiene que haber al menos uno así.
  - Para todos ellos $change[p] = def[p]$
- Visitar luego aquellos procedimientos que llaman a procedimientos que no llaman a otros.
  - Se puede calcular $change[p]$ directamente, pues ya disponemos de todos los $change[q]$ calculados en el paso anterior.

UNIVERSIDAD SIMÓN BOLÍVAR

Eficiencia del algoritmo

Orden de visita a los procedimientos

- Visitar primero aquellos procedimientos que no llaman a otros.
  - Tiene que haber al menos uno así.
  - Para todos ellos $change[p] = def[p]$
- Visitar luego aquellos procedimientos que llaman a procedimientos que no llaman a otros.
  - Se puede calcular $change[p]$ directamente, pues ya disponemos de todos los $change[q]$ calculados en el paso anterior.
- Generalizar construyendo un *Grafo de Llamadas*
  - Arista $p \rightarrow q$ si p llama a q.
  - Si no hay recursión mutua, se obtiene un DAG.
  - Si hay recursión mutua, procesar primero el DAG – luego converger por punto fijo usando los ciclos.

UNIVERSIDAD SIMÓN BOLÍVAR

Variables cambiadas por procedimientos

Algoritmo general – Inicialización

input: colección de procedimientos $p_1, \ldots, p_n$

- Si el Grafo de Llamadas es acíclico, supondremos que p_i llama a p_j cuando $j < i$.
- Si el Grafo de Llamadas tiene ciclos, no suponemos nada.

ouput: $change[p_i]$ para $1 \leq i \leq n$

for all p **do**
  calcular $def[p]$ por inspección
end for
for all p **do**
  $change[p] \leftarrow def[p]$
end for

UNIVERSIDAD SIMÓN BOLÍVAR

Variables cambiadas por procedimientos

Algoritmo general – Cálculo

while haya cambios en cualquier $change[p]$ **do**
  for $i \leftarrow 1$ **to** n **do**
    for all q llamado por p_i **do**
      $change[p_i] \leftarrow change[p_i] \cup$ globales en $change[q]$
      for all parámetro formal x de q **do**
        if $x \in change[q]$ **then**
          for all llamada a q en p_i **do**
            if a (actual que corresponde a x) es global **or** es formal de p_i **then**
              $change[p_i] \leftarrow change[p_i] \cup a$
            end if
          end for
        end if
      end for
    end for
  end for
end while

Aprovechando la información de cambio

Refinación de otros análisis de flujo de datos

- Expresiones Disponibles – ¿cuándo matar expresiones?
  - Una definición para `a` mata expresiones que involucren `a` o sus aliases.
  - Una llamada a `q` no redefine `a` a menos que $\texttt{a} \in \mathit{change}[q]$.
- Expresiones Disponibles – ¿cuándo disponer de expresiones?
  - Una llamada a `q` genera `a+b` si todos los caminos de ejecución pasan por ella sin redefiniciones para `a` ni `b` – usar $\mathit{change}[q]$ dentro de `q`.
  - Una ocurrencia de `x+y` corresponde con `a+b` sólo si para toda llamda de `q`, `a` es alias de `x` y `b` es alias de `y`.

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*] (Primera Edición)
  - Sección 10.8
  - Ejercicios 10.1 a 10.3

UNIVERSIDAD SIMÓN BOLÍVAR