La forma SSA en el testing de programas con arreglos

Luis M. Peñaranda

27 de febrero de 2006

Testing

- funcional
  - white-box
  - código, flujo de control criterios
- estructural
  - black-box
  - especificación formal

Trabajos anteriores

- Rapps & Weyuker
  Testing basado en flujo de datos
- Dacharry
  R&W + SSA

Presente trabajo

- R&W
- SSA
- Arreglos
- Implementación

Lenguaje usado

Variables

Arreglos

Sentencia de entrada: `ACCEPT x1, .., xn.`

Sentencia de salida: `DISPLAY x1, .., xn.`

Sentencia de asignación: `COMPUTE y = f(x1, .., xn).`

Sentencia de transferencia: `MOVE x TO y.`

Etiqueta: `etiq.`

Sentencia de salto: `GO TO etiq.`

Sentencia condicional:

```
IF p(x1, .., xn) sent1 ELSE sent2.
```

Sentencia de fin: `STOP RUN.`

Flujo de control

El grafo de flujo de control (GFC) se define como

$$G = (V, E).$$

$v_1 \rightarrow v_2 \in G \iff$

- v_1 sentencia de salto, v_2 destino
- v_1 condicional, v_2 destino posible
- v_1 destino en condicional, v_2 siguiente al condicional

Programa de ejemplo

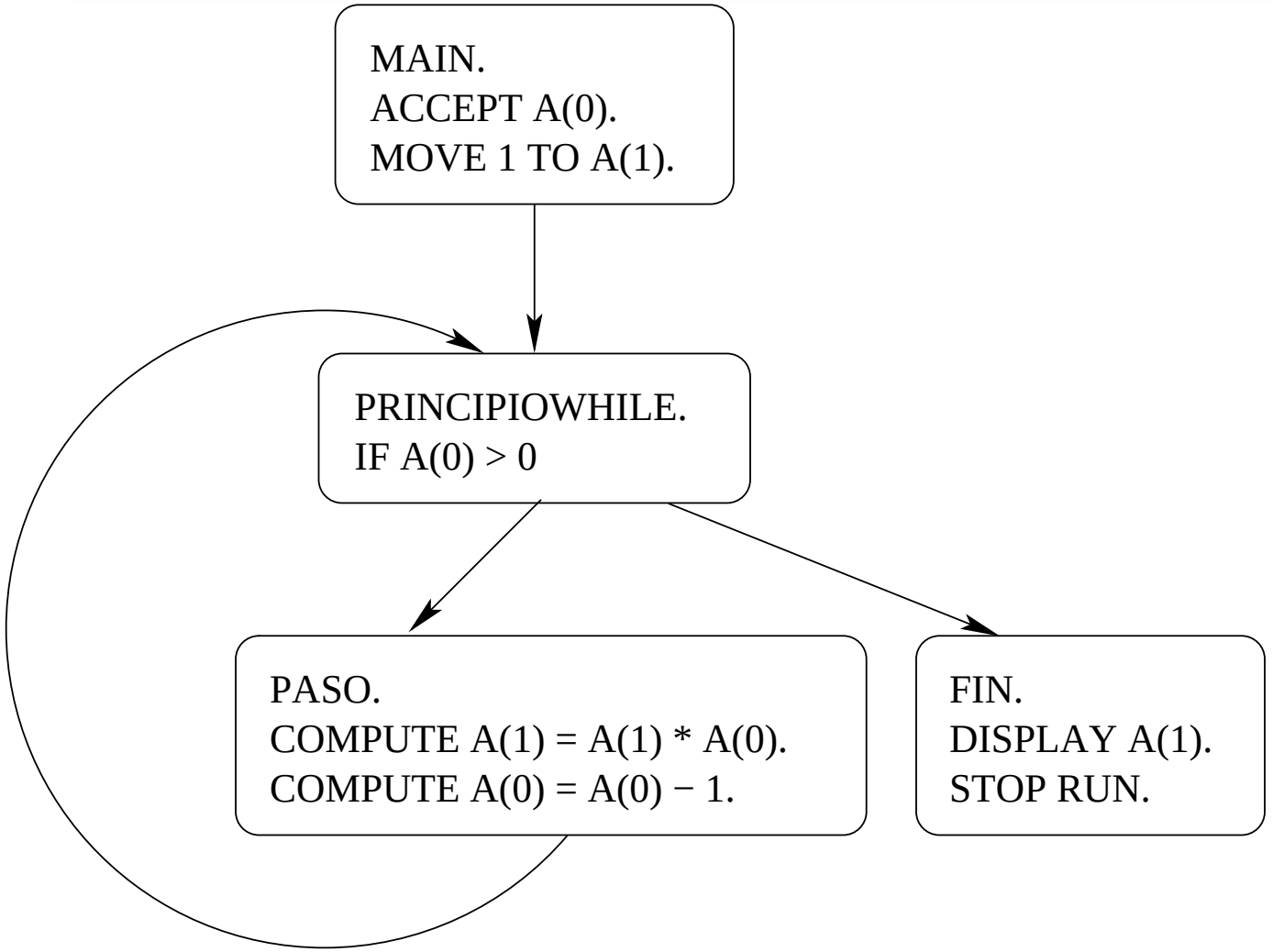
```
MAIN.
ACCEPT A(0).
MOVE 1 TO A(1).

PRINCIPIOWHILE.
IF A(0)>0
        GO TO PASO.
ELSE
        GO TO FIN.

PASO.
COMPUTE A(1) = A(1) * A(0).
COMPUTE A(0) = A(0) - 1.
GO TO PRINCIPIOWHILE.

FIN.
DISPLAY A(1).
STOP RUN.
```

GFC del ejemplo



Flujo de datos

Recorrer el GFC y obtener información sobre las relaciones entre variables.

R&W definen:

- def
- p-uso
- c-uso

defs y usos

Sentencia	def	c-uso	p-uso
`COMPUTE y = f(x1, .., xn).`	`y`	`x1` a `xn`	
`MOVE x TO y.` (`x` es una variable)	`y`	`x`	
`MOVE` *valor* `TO y.`	`y`		
`ACCEPT x1, .., xn.`	`x1` a `xn`		
`DISPLAY x1, .., xn.`		`x1` a `xn`	
`IF p(x1, .., xn) sent1 ELSE sent2.`			`x1` a `xn`

Definición-uso

Conceptos previos:

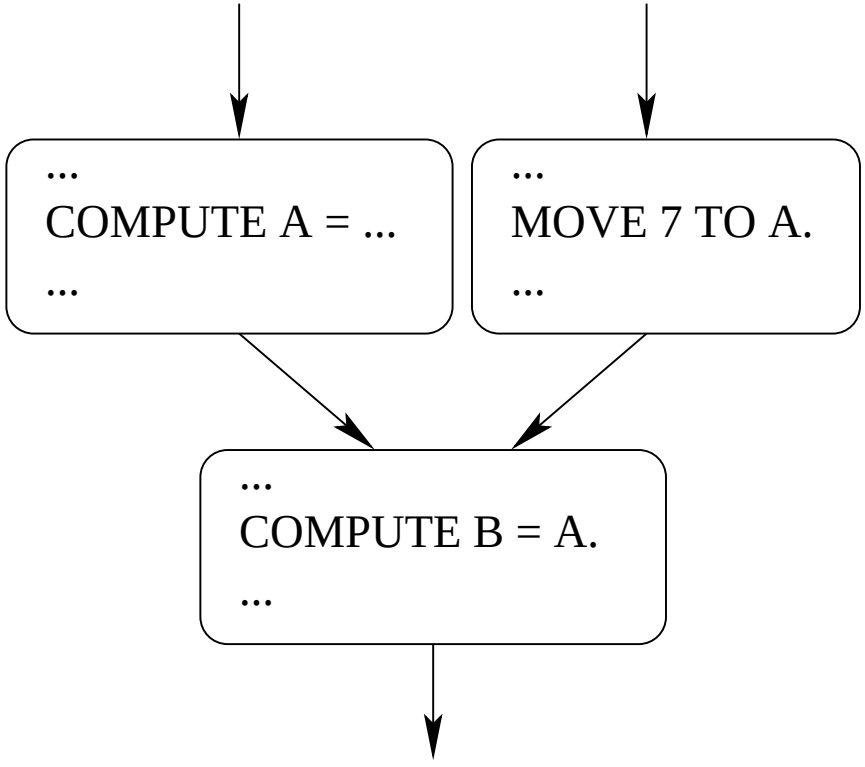
- caminos libres de definición
- defs locales y globales
- grafo def-uso (GFC+información)
- dcu, dpu

La forma SSA

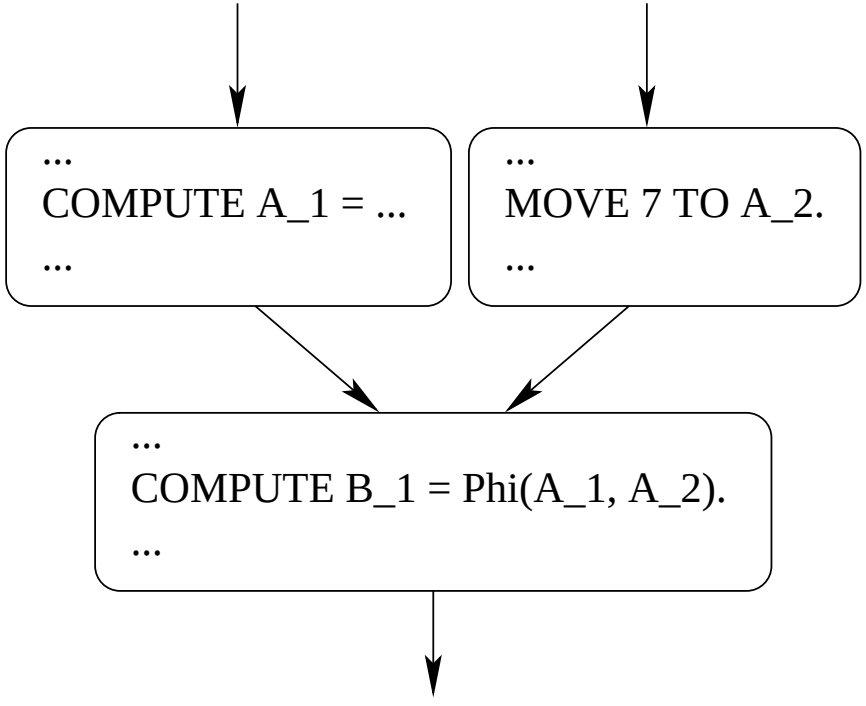
Ventajas:

- análisis del flujo de datos más simples
- tamaño de la representación de cadenas definición-uso
- tiempo
- distintos usos de una variable se transforman en distintas variables (eliminación de dependencias innecesarias)

Problema



ϕ-funciones



Dominancia

- x *domina* a y si cada camino de lados dirigidos desde la raíz a y debe pasar por x
- en SSA, las definiciones dominan a los usos
- x domina estrictamente a y si x domina a y y $x \neq y$
- la frontera de dominancia de x es el conjnto de todos los nodos w tales que x domina a un predecesor de w pero no domina estrictamente a w

Cómputo de la frontera de dominancia

$calcularDF[n] =$

$S \leftarrow \{\}$

para todo nodo y en $sucesores[n]$

si $idom[y] \neq n$

$S \leftarrow S \cup \{y\}$

para todo hijo c de n en el árbol de dominadores

$calcularDF[c]$

para todo elemento w de $DF[c]$

si n no domina a w

$S \leftarrow S \cup \{w\}$

$DF[n] \leftarrow S$

Eliminación de código muerto

$W \leftarrow$ una lista de todas las variables del programa SSA
mientras W no es vacía
    quitar alguna variable v de W
    si la lista de usos de v es vacía
        sea S la sentencia de definición
        si el único efecto lateral de S es la asignación a v
            quitar S del programa
            por cada variable x_i usada por S
                quitar S de la lista de usos de x_i
                $W \leftarrow W \cup \{x_i\}$

Propagación simple de constantes

$W \leftarrow$ una lista de todas las sentencias del programa SSA
mientras W no es vacía
    quitar alguna sentencia S de W
    si S es $v \leftarrow \phi(c, c, \ldots, c)$ para alguna constante c
        reemplazar S por $v \leftarrow c$
    si S es $v \leftarrow c$ para alguna constante c
        quitar S del programa
        por cada sentencia T que usa v
            reemplazar c por v en T
            $W \leftarrow W \cup \{T\}$

Gratis con la propagación simple

Pueden incorporarse fácilmente:

- propagación de copia
- plegado de constantes
- condiciones constantes
- código inalcanzable

Propagación condicional de constantes

- necesita que el grafo tenga la propiedad de *único sucesor o predecesor*
- no asume que un bloque básico puede ser ejecutado hasta que tiene evidencia de que puede serlo
- no asume que una variable no es constante hasta que tiene evidencia de ello

Criterios R&W

- todos-los-nodos
- todos-los-lados
- todas-las-definiciones
- todos-los-p-usos
- todos-loc-c-usos/algunos-p-usos
- todos-los-p-usos/algunos-c-usos
- todos-los-usos
- todas-las-definiciones-usos
- todos-los-caminos

Criterios Dacharry

Definiciones previas:

- ϕ-vars(x_i)
- c-uso-ssa
- p-uso-ssa

Criterios:

- análogos a los de R&W; se agrega “(ssa)” después del nombre, para diferenciarlos.

Ejemplo de criterio SSA

Sea S un grafo SSA de un programa y P un conjunto de caminos completos de S. P satisface el criterio todos-los-p-usos/algunos-c-usos(ssa) si por cada nodo n de S y por cada variable x_i definida en n (que no corresponda a la definición de una ϕ-función) existen en P caminos desde n hasta todos los sucesores de todos los nodos que contengan un p-uso-ssa de x_i. En caso de no haber ningún ningún nodo que contenga un p-uso-ssa de x_i en S, P debe contener un camino desde n hasta un nodo que contenga un c-uso-ssa de x_i.

Criterios para arreglos

- ¿Por qué se necesita testear arreglos?
- Definición de i-uso
- ¿Qué se está testeando con los i-usos?
- Criterios: análogos a los de R&W y Dacharry; se agrega “(arr)” después del nombre, para diferenciarlos.

Ejemplo de criterio para arreglos

Sea S un grafo SSA de un programa y P un conjunto de caminos completos de S. P satisface el criterio *todos-los-p-usos/algunos-c-usos(arr)* si por cada nodo n de G y cada variable x definida en n, P contiene un camino libre de definición con respecto a x desde n hasta:

1. todos los lados de $dpu(x, n)$ y, si es vacío, hasta algún nodo de $dcu(x, n)$, y
2. todos los sucesores de todos los nodos de $diu(x, n)$ donde x esté usado subindizando un arreglo usado en un predicado y, si no hay en $diu(x, n)$ ningún nodo que cumpla esta condición, hasta algún nodo de $diu(x, n)$ donde x esté usado como subíndice de un arreglo usado para cómputo.

Criterios para arreglos con SSA I

- Definición de i-uso-ssa
- Criterios: análogos a todos los anteriores; se agrega "(arr-ssa)" después del nombre, para diferenciarlos.

Ejemplo de criterio:

Sea S un grafo SSA de un programa y P un conjunto de caminos completos de S. P satisface el criterio *todos-los-p-usos/algunos-c-usos(arr-ssa)* si por cada nodo n de S y cada variable x_i definida en n (que no corresponda a la definición de una ϕ-función) existen en P caminos desde n hasta:

Criterios para arreglos con SSA II

1. todos los sucesores de todos los nodos que contengan un p-uso-ssa de x_i y, si ninguno cumple esta condición, hasta algún nodo que contenga un c-uso-ssa de x_i, y
2. todos los sucesores de todos los nodos que contengan un i-uso-ssa de x_i donde x_i esté usado subindizando un arreglo usado en un predicado y, si ninguno cumple esta condición, hasta un sucesor de algún nodo que contenga un i-uso-ssa de x_i donde x_i esté usado como subíndice de un arreglo usado para cómputo.

Cadenas DUA

- ¿Qué testean los criterios anteriores?
- ¿Qué necesitamos para terminar de testear?
- Cadenas definición-uso de arreglos (DUA):
  - definidas
  - indefinidas
  - aparentes

Verificación en dos fases

Proceso de selección de casos de prueba:

- elegir un criterio “(arr)”
- verificar variables escalares con el criterio elegido $\rightarrow C_1$
- verificar cadenas DUA con el criterio análogo al elegido para DUA $\rightarrow C_2$
- el conjunto de caminos a ejercitar será $C = C_1 \cup C_2$

Implementación de una herramienta

Qué hace y qué no hace:

- pasa a SSA
- analiza las cadenas definición-uso
- recibe un conjunto de caminos y analiza si cumple criterios
- no genera casos de prueba
- no testea

Lenguaje elegido

Objective Caml

- capacidad para manejar estructuras de datos complejas y de gran tamaño
- buena performance del código compilado
- disponibilidad de herramientas para el desarrollo de grandes piezas de software
- librerías apropiadas para manejar todo tipo de estructuras de datos
- estructura modular

Conclusiones

- se amplió una técnica de testing para que capture la esencia de los datos estructurados
- se extendió la técnica mencionada para que haga uso de la forma SSA en los análisis
- se implementó una herramienta haciendo uso de los resultados del presente trabajo

Trabajos futuros

- trabajos teóricos: características más generales del lenguaje
- implementación: completar la herramienta, aplicando mejoras futuras de la teoría o haciéndola usable

¿Preguntas?

Y, tal vez, respuestas.

Gracias por su atención

luis@fceia.unr.edu.ar

www.fceia.unr.edu.ar/~luis