Universidad Nacional Autónoma de México
Semestre 2025-1
Práctica: Bloques Básicos,
Gráfica de control de flujo y Análisis de variables vivas
Profesora: Ariel Adara Mercado Martínez

Objetivo:

Implementar en Java la división del código intermedio en bloques básicos para generar la gráfica de control de flujo y realizar el análisis de variables vivas de cada bloque.

Actividades

1. Programar la clase que represente las cuádruplas.
2. Programar una clase que represente un bloque básico.
3. Programar una clase que represente la gráfica de control de flujo
4. Programar la clase que genera la gráfica de control de flujo, al mismo tiempo divide el código intermedio en bloques básicos.
5. Programar la clase que realiza el análisis de variables vivas

Entrada: Lista de instrucciones en código de tres direcciones instrucciones
Salida: Gráfica de Control de Flujo (cfg)

```
bloquesBasicos ← ∅;
bloqueActual ← crearNuevoBloque();
cfg ← nueva ControlFlowGraph();
para cada instrucción inst ∈ instrucciones hacer
    si inst es una etiqueta o un salto (incondicional o condicional) entonces
        si bloqueActual no está vacío entonces
            agregarBloque(bloquesBasicos, bloqueActual);
            cfg.addBlock(bloqueActual);
            bloqueActual ← crearNuevoBloque();
        fin
        si inst es una etiqueta entonces
            bloqueActual.agregarInstruccion(inst);
        fin
        si inst es un salto entonces
            bloqueActual.agregarInstruccion(inst);
            agregarBloque(bloquesBasicos, bloqueActual);
            cfg.addBlock(bloqueActual);
            bloqueActual ← crearNuevoBloque();
        fin
    fin
    en otro caso
        bloqueActual.agregarInstruccion(inst);
    fin
fin
si bloqueActual no está vacío entonces
    agregarBloque(bloquesBasicos, bloqueActual);
    cfg.addBlock(bloqueActual);
fin
para cada bloque b ∈ cfg.getBlocks() hacer
    para cada instrucción inst ∈ b.getInstructions() hacer
        setSuccessor(b, inst);
    fin
fin
devolver cfg
```

Algoritmo 1: Generador de Bloques Básicos y CFG

Entrada: Bloque básico bloqueActual, Instrucción inst
Salida: Sucesores actualizados en la gráfica de control de flujo cfg

```
si inst es un salto incondicional entonces
    destino ← cfg.getBlockByLabel(inst.destino) ;
    bloqueActual.addSuccessor(destino);
fin
si no, si inst es un salto condicional entonces
    destino ← cfg.getBlockByLabel(inst.destino) ;
    bloqueActual.addSuccessor(destino);
    sucesorSiguiente ← cfg.getNextBlock(bloqueActual) ;
    si sucesorSiguiente ≠ ∅ entonces
        bloqueActual.addSuccessor(sucesorSiguiente);
    fin
fin
en otro caso
    sucesorSiguiente ← cfg.getNextBlock(bloqueActual);
    si sucesorSiguiente ≠ ∅ entonces
        bloqueActual.addSuccessor(sucesorSiguiente);
    fin
fin
```

Algoritmo 2: Establecer Sucesores de un Bloque Básico

Entrada: Gráfica de Control de Flujo (cfg)
Salida: Conjuntos de variables vivas in y out para cada bloque básico en cfg
para cada *bloque b* $\in$ *cfg.getBasicBlocks()* **hacer**
  Inicializar b.liveVariableSets.in y b.liveVariableSets.out como conjuntos vacíos;
fin
repetir
  para cada *bloque b* $\in$ *cfg.getBasicBlocks()* **hacer**
    $out_{nuevo} \leftarrow$ calcularOut(b, cfg);
    $in_{nuevo} \leftarrow$ calcularIn(b, out_{nuevo});
    si $out_{nuevo} \neq$ *b.liveVariableSets.out* ***or*** $in_{nuevo} \neq$ *b.liveVariableSets.in* **entonces**
      b.liveVariableSets.out $\leftarrow$ out_{nuevo};
      b.liveVariableSets.in $\leftarrow$ in_{nuevo};
      Cambiar el indicador de cambio a verdadero;
    fin
  fin
hasta que *no haya cambios en ningún conjunto*;

Algoritmo 3: Análisis de Variables Vivas en la CFG

Entrada: Bloque básico b, Gráfica de Control de Flujo cfg
Salida: Conjunto out para el bloque b
outSet $\leftarrow \emptyset$;
para cada *bloque sucesor s* $\in$ *cfg.getSuccessors(b)* **hacer**
  outSet $\leftarrow$ outSet $\cup$ s. liveVariableSets.in;
fin
devolver $outSet$

Algoritmo 4: Calcular Conjunto out de un Bloque

Entrada: Bloque básico b, Conjunto outSet del bloque b
Salida: Conjunto in para el bloque b
inSet $\leftarrow$ outSet;
para *instrucción quad en b.getInstructions() en orden inverso* **hacer**
  si *quad.result no es nulo* **entonces**
    inSet $\leftarrow$ inSet $\setminus$ {quad. result};
  fin
  si *quad.arg1 no es nulo* **entonces**
    inSet $\leftarrow$ inSet $\cup$ {quad. arg1};
  fin
  si *quad.arg2 no es nulo* **entonces**
    inSet $\leftarrow$ inSet $\cup$ {quad. arg2};
  fin
fin
devolver $inSet$

Algoritmo 5: Calcular Conjunto in de un Bloque