Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
CI–4722 – Lenguajes de Programación III
Abril – Julio 2025

Tarea II

(15 puntos)

A continuación encontrará 4 preguntas, cada una compuesta de diferentes sub–preguntas. El valor de cada pregunta (y sub–pregunta) estará expresado entre paréntesis al inicio de las mismas.

En aquellas preguntas donde se le pida ejecutar un algoritmo o procesar una entrada, incluya los pasos relevantes de la ejecución del mismo con los cuales usted pudo alcanzar su conclusión. Sea lo más detallado y preciso posible en sus razonamientos y procedimientos.

La entrega se realizará por correo electrónico a rmonascal@gmail.com hasta las 11:59pm. VET del Miércoles, 25 de Junio de 2025.

1. (5 puntos) Considere un TAC cuyas expresiones son todas números naturales (esto es, no permiten números negativos) y todas las variables son inicializadas en un número natural arbitrario.

   La sintaxis y operaciones disponibles para este TAC se entienden con la siguiente gramática:

$$
\begin{array}{rcl}
TAC & \rightarrow & TAC \quad \backslash n \quad LINE \\
 & | & LINE \\
LINE & \rightarrow & \textbf{id} \quad : \quad INSTR \\
 & | & INSTR \\
INSTR & \rightarrow & \textbf{id} \quad := \quad ARIT \\
 & | & advance \quad \textbf{id} \\
 & | & goto \quad \textbf{id} \\
 & | & goif \quad CMP \quad \textbf{id} \\
ARIT & \rightarrow & \textbf{num} \quad + \quad \textbf{num} \\
 & | & \textbf{num} \quad - \quad \textbf{num} \\
 & | & \textbf{num} \quad * \quad \textbf{num} \\
 & | & \textbf{num} \quad / \quad \textbf{num} \\
CMP & \rightarrow & \textbf{num} \quad < \quad \textbf{num} \\
 & | & \textbf{num} \quad > \quad \textbf{num} \\
 & | & \textbf{num} \quad == \quad \textbf{num} \\
 & | & \textbf{num} \quad != \quad \textbf{num}
\end{array}
$$

   La mayoría de las operaciones aritméticas y relacionales tienen la semántica convencional. Además, existirá una instrucción especial `advance`, de forma tal que `advance x` es equivalente a `x = x + 1`. Las operaciones de resta y división, sin embargo, tienen semánticas especiales en este contexto:

   - La resta es siempre mayor o igual a cero: $a -_{\mathbb{N}} b = max(a -_{\mathbb{Z}} b, 0)$
   - La división es truncada al natural inmediatamente inferior: $a /_{\mathbb{N}} b = \lfloor a /_{\mathbb{R}} b \rfloor$

Considerando este TAC que trabaja sobre números naturales:

a) (3 puntos) Queremos saber si una variable en nuestro TAC puede llegar a ser igual a cero (por algún camino posible de ejecución).

Como ejemplo, consideremos el siguiente programa:

```
      advance b
      goif b < c then
      a := b - c
      goto end
then: a := b + c
end:  c := b / c
```

Notemos que dependiendo de los valores iniciales (que son desconocidos) el flujo puede seguir uno de dos caminos hasta llegar a la última instrucción.

Primeramente se avanza `b`, haciendo que sea imposible que `b` sea igual a cero. Luego, si `b < c` se hace la asignación `a := b + c`. Después de esta instrucción es imposible que `a` sea cero, ya que `b` no puede ser cero y sumarle un número natural cualquiera no puede producir cero. Si, por el contrario, `b >= c` entonces se ejecuta la asignación `a := b - c`. Independientemente del valor de `b`, una resta siempre podrá resultar en cero. Finalmente, la última instrucción es una asignación `c := b / c`, por lo que es posible que `c` siga valiendo cero (también es posible que esté indefinida, pero aún no nos preocuparemos por eso).

En conclusión, justo después de la última instrucción, es posible que `a` y `c` sean cero, pero imposible que `b` lo sea.

Plantee el problema de encontrar las variables que son potencialmente cero para cada instrucción como un problema de flujo de datos, proponiendo la construcción de IN y OUT (incluyendo el estado inicial) y la función de transferencia asociada. Diga también si dicho razonamiento será *hacia adelante* o *hacia atrás*.

b) (2 puntos) Usando la información de qué variables son potencialmente cero, establezca cuáles expresiones de división tienen riesgo de estar indefinidas (por tener denominador igual a cero).

Considerando el mismo ejemplo de la parte anterior, la asignación `c := b / c` es de riesgo, pues es posible que `c` haya sido igual a cero antes de evaluar `b / c`.

2. (5 puntos) Considere el siguiente fragmento de código en pseudo–código que realiza la operación *merge*:

```
i := 0;
j := 0;
k := 0;
while (i < n && j < m) {
    if (a[i] < b[j]) {
        c[k] := a[i];
        i := i + 1;
    } else {
        c[k] := b[j];
        j := j + 1;
    }
    k := k + 1;
}
while (i < n) {
    c[k] := a[i];
    i := i + 1;
    k := k + 1;
}
while (j < m) {
    c[k] := b[j];
    j := j + 1;
    k := k + 1;
}
```

Puede suponer que n y m son enteros positivos. También puede suponer que a es un arreglo de enteros de tamaño n, b es un arreglo de enteros de tamaño m y c es un arreglo de enteros de tamaño $n + m$.

a) (1 punto) Construya el TAC que se genera a partir del fragmento de código anterior.

b) (1 punto) Del TAC obtenido, construya el grafo de flujo asociado.

c) (3 puntos) Del TAC obtenido, construya el árbol de dominadores, mostrando los diferentes tipos de aristas resultantes (avance, retroceso, retorno y cruce). Con esta información identifique los ciclos naturales y diga si el grafo es reducible o no.

3. (5 puntos) Considere el siguiente fragmento de código en TAC, que calcula en la variable s el valor de $\sum_{i=0}^{n} 3 \times i + a[i]$, dado n que es un entero positivo y a un arreglo de enteros de tamaño n.

```
L1: i := 0
    s := 0
L2: t1 := 3 * i
    t2 := a[i]
    t3 := t1 + t2
    s := s + t3
    i := i + 1
    if i < n goto L2
    print(s)
```

Aplique el algoritmo de detección de variables de inducción, conduciendo a la reducción de fuerza y posible eliminación de las mismas. Indique claramente las familias de variables identificadas con las tripletas correspondientes, y el código mejorado definitivo.