Invariante

Algoritmos y estructuras de datos

Elías Cerdeira

Fechas importantes

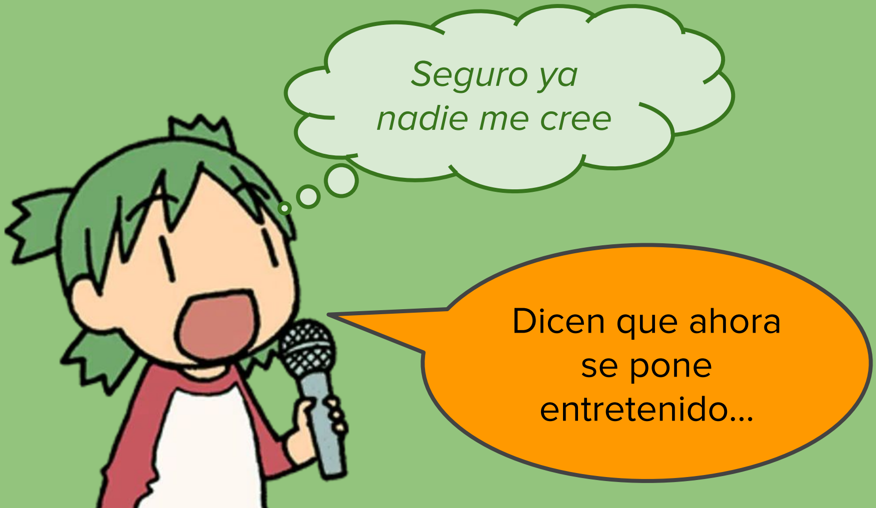
Primer parcial: Viernes 3 de Mayo…

¡Quedan 16 días!

¿Qué vamos a ver hoy?

- Corregir implementaciones de acuerdo a la especificación
- Pensar invariantes y funciones variantes
- Entender cómo encarar una demostración
- **Consultas** (sí, también del TP)

¡A resolver ejercicios!



Corrección de ciclos

Ejercicio 6. Dado el siguiente problema

proc `sumarElementos` (in s: $array < \mathbb{Z} >$) : $\mathbb{Z}$
  requiere $\{|s| \geq 1\}$
  asegura $\{res = \sum_{j=0}^{|s|-1} s[j]\}$

a)
```
res := 0
i := 0
while (i > s.size()) do
        res := res + s[i];
        i := i + 1
endwhile
```

b)
```
res := 0
i := 0
while (i > s.size()) do
        res := res + s[s.size() - i];
        i := i + 1
endwhile
```

i) Corregir las implementaciones

ii) Dar un invariante y función variante para cada implementación

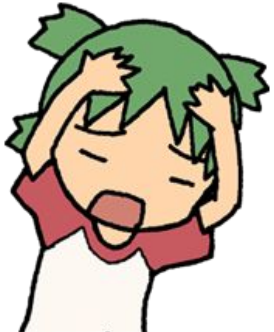
Corrección de ciclos

Ejercicio 6. Dado el siguiente problema

$$\texttt{proc sumarElementos} \ (\text{in s: } array < \mathbb{Z} >) : \mathbb{Z}$$
$$\texttt{requiere} \ \{|s| \geq 1\}$$
$$\texttt{asegura} \ \{res = \sum_{j=0}^{|s|-1} s[j]\}$$

c)
```
res := 0
i := s.size() - 1
while (i > 0) do
        res := res + s[i];
        i := i - 1
endwhile
```

d)
```
res := 0
i := 0
while (i > s.size() / 2) do
        res := res + s[i] + s[s.size() - i];
        i := i + 1
endwhile
```



i) Corregir las implementaciones

ii) Dar un invariante y función variante para cada implementación

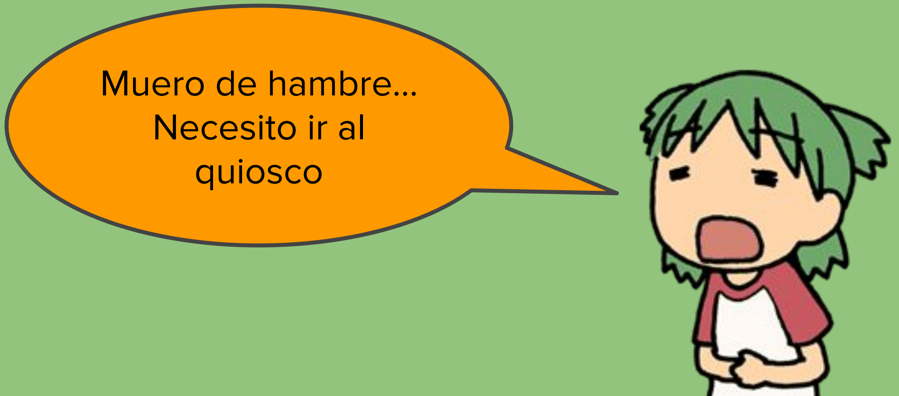
setAt

No podemos usar el ***axioma 1*** para el programa *A[i] := E*. La regla sólo coincide con sentencias del tipo *x := E*, donde *x* es una variable.

- Redefinimos $A[i] := E$ como $A := setAt(A, i, E)$
- $def(setAt(A, i, E)) = (def(A) \land def(i) \land def(E)) \land_L 0 \leq i < |A|$
- Dados $0 \leq i, j < |A|$ tenemos que:

$$\mathbf{setAt(A, i, E)[j]} = \begin{cases} E & \text{si } i = j \\ A[j] & \text{si } i \neq j \end{cases}$$

¡Intervalo!



Corrección de ciclos

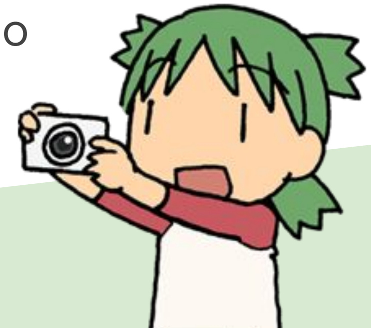
Ejercicio 7. Considerando el siguiente Invariante:

$$I \equiv \{0 \leq i \leq |s| \wedge (\forall j : \mathbb{Z})(0 \leq j < i \rightarrow_L ((j \ mod \ 2 = 0 \wedge s[j] = 2 \times j) \vee (j \ mod \ 2 \neq 0 \wedge s[j] = 2 \times j + 1)))\}$$

- Escribir un programa en SmallLang que se corresponda al invariante dado.
- Defina las P_c, B y Q_c que correspondan a su programa.
- Dar una función variante para que se pueda completar la demostración.

Demostraciones de correctitud

- Para demostrar Ant $\longrightarrow$ Cons podemos usar Ant como hipótesis, es decir, tomar sus afirmaciones como verdaderas
- Para pensar un invariante de ciclo, tengo que observar las cosas que se hacen dentro del ciclo, las pre y poscondiciones. Pensar qué cosas necesito para poder demostrar lo que se requiere en los teoremas
- Para pensar en una función variante observo la guarda y pienso cómo se modifican los valores que intervienen



Para pensar un poco...

Ejercicio 9. Indique si el siguiente enunciado es verdadero o falso; fundamente:

Si dados B y I para un ciclo S existe una función f_v que cumple lo siguiente:

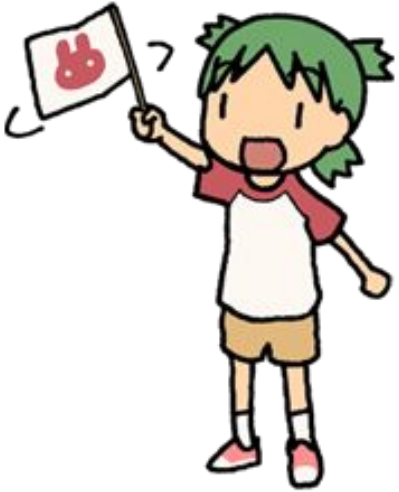
- $\{I \wedge B \wedge f_v = V_0\}S\{f_v > V_0\}$
- $\exists(k : \mathbb{Z})(I \wedge f_v \geq k \rightarrow \neg B)$

entonces el ciclo siempre termina.

¿Qué diferencia tiene con los puntos del teorema de **terminación**?
¿Qué me cambia eso?
¿Me podrían dar un ejemplo?

¿Qué sigue?

- Con la clase de hoy pueden resolver **toda** la **guía 3** (ambas partes)
- La clase que viene vemos **especificación de tipos abstractos de datos (TADs)**
- Luego de esa clase nos queda un **repaso/consulta** y luego el **parcial**



¡Terminamos!

¡Hagan consultas!

Gracias por
acompañarnos