Razonamientos de corrección sobre ciclos

Algoritmos y Estructuras de Datos I

Ciclos (repaso)

- Recordemos la sintaxis de un ciclo:

```
while (B) {
     cuerpo del ciclo
}
```

Ciclos (repaso)

- Recordemos la sintaxis de un ciclo:

  ```
  while (B) {
      cuerpo del ciclo
  }
  ```

- Se repite el cuerpo del ciclo mientras la guarda `B` se cumpla, cero o más veces. Cada repetición se llama una iteración.

Ciclos (repaso)

- Recordemos la sintaxis de un ciclo:

  ```
  while (B) {
        cuerpo del ciclo
  }
  ```

- Se repite el cuerpo del ciclo mientras la guarda `B` se cumpla, cero o más veces. Cada repetición se llama una iteración.
- La ejecución del ciclo termina si no se cumple la guarda al comienzo de su ejecución o bien luego de ejecutar una iteración.

Ciclos (repaso)

- Recordemos la sintaxis de un ciclo:

  ```
  while (B) {
        cuerpo del ciclo
  }
  ```

- Se repite el cuerpo del ciclo mientras la guarda `B` se cumpla, cero o más veces. Cada repetición se llama una iteración.
- La ejecución del ciclo termina si no se cumple la guarda al comienzo de su ejecución o bien luego de ejecutar una iteración.
- Si/cuando el ciclo termina, el estado resultante es el estado posterior a la última instrucción del cuerpo del ciclo.

Ejemplo

- proc $sumar(\text{in } n : \mathbb{Z}, \text{out } result : \mathbb{Z})\{$
  Pre $\{n \geq 0\}$
  Post $\{result = \sum_{i=1}^{n} i\}$
  $\}$

-
```
int sumar(int n) {
  int i = 1;
  int suma = 0;

  while( i <= n ) {
    suma = suma + i;
    i = i + 1;
  }
  return suma;
}
```

Ejemplo

- Estados al finalizar cada iteración del ciclo, para $n = 6$:

Iteración	i	suma
0	1	0
1	2	1
2	3	3
3	4	6
4	5	10
5	6	15

Ejemplo

- Estados al finalizar cada iteración del ciclo, para $n = 6$:

Iteración	i	suma
0	1	0
1	2	1
2	3	3
3	4	6
4	5	10
5	6	15

- **Observación:** Luego de cada iteración, suma $= \sum_{k=1}^{i-1} k$.

- Esta afirmación se denomina un invariante del ciclo.

Invariante de un ciclo

- **Definición.** Un predicado I es un invariante de un ciclo si:
  1. I vale antes de comenzar el ciclo, y
  2. si $I \wedge B$ al comenzar una iteración arbitraria, entonces vale I al finalizar la ejecución del cuerpo del ciclo.
- Un invariante es una propiedad que el ciclo mantiene constante a lo largo de las iteraciones.
- Por ejemplo, otro invariante de este ciclo es $I' \equiv i \geq 0$.

Ejemplo

-
```
int i = 1;
int suma = 0;

while( i <= n ) {
  suma = suma + i;
  i = i + 1;
}
```

- Es fácil ver que $I \equiv \text{suma} = \sum_{k=1}^{i-1} k$ vale al principio del ciclo, cuando $i = 1$ y $suma = 0$.
- Formalmente, si $P_C \equiv i = 1 \wedge \text{suma} = 0$ es el estado previo al ciclo, tenemos que $P_C \Rightarrow I$.

Ejemplo

- También podemos ver que la propiedad suma $= \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.

```
suma = suma + i;



i = i + 1;
```

Ejemplo

- También podemos ver que la propiedad suma $= \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.
- $\{\text{suma} = \sum_{k=1}^{i-1} k \wedge i = i_0 \wedge \text{suma} = \text{suma}_0\}$

```
suma = suma + i;




i = i + 1;
```

Ejemplo

- También podemos ver que la propiedad $\mathrm{suma} = \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.
- $\{\mathrm{suma} = \sum_{k=1}^{i-1} k \wedge i = i_0 \wedge \mathrm{suma} = \mathrm{suma}_0\}$

  ```
  suma = suma + i;
  ```

  $\{\mathrm{suma} = \mathrm{suma}_0 + i \wedge i = i_0\}$

  ```
  i = i + 1;
  ```

Ejemplo

- También podemos ver que la propiedad $\text{suma} = \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.
- $\{\text{suma} = \sum_{k=1}^{i-1} k \land i = i_0 \land \text{suma} = \text{suma}_0\}$

```
suma = suma + i;
```

$\{\text{suma} = \text{suma}_0 + i \land i = i_0\}$
$\Rightarrow \quad \{\text{suma} = \sum_{k=1}^{i-1} k + i \land i = i_0\}$

```
i = i + 1;
```

Ejemplo

- También podemos ver que la propiedad $\text{suma} = \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.
- $\{\text{suma} = \sum_{k=1}^{i-1} k \land i = i_0 \land \text{suma} = \text{suma}_0\}$

```
suma = suma + i;
```

$$
\begin{array}{ll}
\{\text{suma} = \text{suma}_0 + i \land i = i_0\} & \\
\quad \Rightarrow & \{\text{suma} = \sum_{k=1}^{i-1} k + i \land i = i_0\} \\
\text{''} \Rightarrow \text{''} & \{\text{suma} = \sum_{k=1}^{i} k \land i = i_0\}
\end{array}
$$

```
i = i + 1;
```

Ejemplo

- También podemos ver que la propiedad suma $= \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.
- $\{\mathsf{suma} = \sum_{k=1}^{i-1} k \wedge i = i_0 \wedge \mathsf{suma} = \mathsf{suma}_0\}$

```
suma = suma + i;
```

$\{\mathsf{suma} = \mathsf{suma}_0 + i \wedge i = i_0\}$

$\Rightarrow \quad \{\mathsf{suma} = \sum_{k=1}^{i-1} k + i \wedge i = i_0\}$

$" \Rightarrow " \quad \{\mathsf{suma} = \sum_{k=1}^{i} k \wedge i = i_0\}$

```
i = i + 1;
```

$\{\mathsf{suma} = \sum_{k=1}^{i_0} k \wedge i = i_0 + 1\}$

Ejemplo

- También podemos ver que la propiedad suma $= \sum_{k=1}^{i-1} k$ se mantiene en cada ejecución del cuerpo del ciclo.
- Agregamos metavariables para las variables que cambian.
- $\{\mathsf{suma} = \sum_{k=1}^{i-1} k \wedge i = i_0 \wedge \mathsf{suma} = \mathsf{suma}_0\}$

```
suma = suma + i;
```

$\{\mathsf{suma} = \mathsf{suma}_0 + i \wedge i = i_0\}$
$\Rightarrow \quad \{\mathsf{suma} = \sum_{k=1}^{i-1} k + i \wedge i = i_0\}$
$\text{"} \Rightarrow \text{"} \quad \{\mathsf{suma} = \sum_{k=1}^{i} k \wedge i = i_0\}$

```
i = i + 1;
```

$\{\mathsf{suma} = \sum_{k=1}^{i_0} k \wedge i = i_0 + 1\}$
$\Rightarrow \{\mathsf{suma} = \sum_{k=1}^{i-1} k\} \equiv \{I\}$

Ejemplo

- Tenemos entonces:
  1. I vale justo antes de comenzar el ciclo.
  2. I vale luego de cada iteración.

Ejemplo

- Tenemos entonces:

    1. I vale justo antes de comenzar el ciclo.
    2. I vale luego de cada iteración.

- Esto implica que I también vale cuando el ciclo termina.

Ejemplo

- Tenemos entonces:
  1. I vale justo antes de comenzar el ciclo.
  2. I vale luego de cada iteración.
- Esto implica que I también vale cuando el ciclo termina.
- Al terminar el ciclo tenemos $i = n + 1$, y entonces estamos en el siguiente estado:

$$\begin{aligned}
I \ \wedge \ i = n+1 \quad & \equiv \quad suma = \sum_{k=1}^{i-1} k \ \wedge \ i = n+1 \\
& \Rightarrow \quad suma = \sum_{k=1}^{n} k \\
& \equiv \quad Q
\end{aligned}$$

Ejemplo

- **Atención!** El argumento anterior tiene problemas formales (los pueden detectar?), y por lo tanto no es correcto.
- En las próximas transparencias formalizamos la noción del invariante de un ciclo, y arreglamos estos problemas reforzando el invariante.

Invariante de un ciclo

- Los invariantes de un ciclo permiten razonar sobre su corrección. Llamamos ...
  1. P_C: Precondición del ciclo,
  2. Q_C: Postcondición del ciclo,
  3. B: Guarda del ciclo,
  4. I: Un invariante del ciclo.
- Si se cumplen estas condiciones ...
  1. $P_C \Rightarrow$ I,
  2. $\{I \wedge B\}$ `cuerpo del ciclo` $\{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
- ... entonces el ciclo es parcialmente correcto respecto de su especificación (si termina, termina en Q_C).

Teorema del invariante

- **Teorema del invariante**. Si existe un predicado I tal que ...

  1. $P_C \Rightarrow \mathrm{I}$,
  2. $\{I \land B\}$ S $\{I\}$,
  3. $I \land \neg B \Rightarrow Q_C$,

  entonces el ciclo **while(B) S** es parcialmente correcto respecto de la especificación (P_C, Q_C).

- Este teorema es la herramienta principal para argumentar la corrección de ciclos.

- El teorema del invariante se puede demostrar formalmente (más detalle en las próximas teóricas!).

Ejemplo

- Verifiquemos estas tres condiciones con el ejemplo anterior, y con ...

  1. $I \equiv \text{suma} = \sum_{k=1}^{i-1} k$,
  2. $P_C \equiv i = 1 \wedge \text{suma} = 0$,
  3. $Q_C \equiv \text{suma} = \sum_{k=1}^{n} k$:

- En primer lugar, debemos verificar que $P_C \Rightarrow I$:

$$i = 1 \wedge \text{suma} = 0 \;\Rightarrow\; \text{suma} = \sum_{k=1}^{i-1} k.$$

Ejemplo

- ¿Es cierto que $\{I \wedge B\} S \{I\}$?

$$I \wedge B : \{i \leq n \wedge \text{suma} = \sum_{k=1}^{i-1} k\}$$

```
suma = suma + i;
i = i + 1;
```

$$I : \{\text{suma} = \sum_{k=1}^{i-1} k\}$$

- No se cumple si $i < 0$! Sin embargo, sabemos que $i \geq 1$ a lo largo del ciclo.

$\Rightarrow$ Lo agregamos al invariante!
$I \equiv i \geq 1 \wedge \text{suma} = \sum_{k=1}^{i} k$.

- Ahora sí es cierto que el invariante se preserva a lo largo de las iteraciones.

Ejemplo

- Finalmente, ¿es cierto que $I \wedge \neg B \Rightarrow Q_C$?

$$i \geq 1 \wedge \text{suma} = \sum_{k=1}^{i-1} k \wedge i > n \;\Rightarrow\; \text{suma} = \sum_{k=1}^{n} k \text{ ?}$$

- **No!** Si $i = n + 2$, entonces la implicación no vale!
- Sin embargo, sabemos que esto no puede pasar, puesto que $i \leq n + 1$ a lo largo del ciclo.

$\Rightarrow$ Reforzamos el invariante!

Ejemplo

- $I \equiv 1 \leq i \leq n+1 \wedge \text{suma} = \sum_{k=1}^{i-1} k$.
- Ahora sí tenemos que $I \wedge \neg B \Rightarrow Q_C$:

$$
\begin{aligned}
& 1 \leq i \leq n+1 \wedge \text{suma} = \sum_{k=1}^{i-1} k \wedge i > n \\
\Rightarrow \quad & i = n+1 \wedge \text{suma} = \sum_{k=1}^{i-1} k \\
\Rightarrow \quad & \text{suma} = \sum_{k=1}^{n} k \equiv Q_C
\end{aligned}
$$

- Los dos primeros puntos se siguen verificando con el nuevo invariante. Luego, el ciclo es parcialmente correcto respecto de su especificación.

Algunas observaciones

- $I \equiv 1 \leq i \leq n+1 \wedge \mathsf{suma} = \sum_{k=1}^{i-1} k$.
  1. En general, un buen invariante debe incluir el rango de la(s) variable(s) de control del ciclo.
  2. Además, debe incluir alguna afirmación sobre el acumulador del ciclo.
- Cuando tenemos un invariante I que permite demostrar la corrección parcial del ciclo, nos referimos a I como el invariante del ciclo.
  1. El invariante de un ciclo caracteriza las acciones del ciclo, y representa al algoritmo que el ciclo implementa.
- En general, es sencillo argumentar informalmente la terminación del ciclo (más detalles en las próximas teóricas).

Otro ejemplo

- Recordemos la especificación para determinar si un entero es primo.
- proc $primo(\text{in } n : \mathbb{Z}, \text{out } result : \text{Bool })\{$
  Pre $\{n \geq 2\}$
  Post $\{result = true \leftrightarrow esPrimo(n)\}$
  $\}$

Otro ejemplo

- Recordemos la especificación para determinar si un entero es primo.
- proc $primo(\text{in } n : \mathbb{Z}, \text{out } result : \text{Bool})\{$
  Pre $\{n \geq 2\}$
  Post $\{result = true \leftrightarrow esPrimo(n)\}$
  $\}$
- En este caso, recurrimos al predicado auxiliar esPrimo, que nos da la pauta de lo que sucede en el ciclo:
- pred esPrimo$(n : \mathbb{Z})$ $\{$
  $n > 1 \ \land \ (\forall n' : \mathbb{Z})(1 < n' < n \rightarrow_L n \bmod n' \neq 0)$
  $\}$

Otro ejemplo

- ```
  boolean primo(int n) {
    int i = 2;
    int divisores = 0;

    while( i < n ) {
      if( n % i == 0 ) {
        divisores += 1;
      } else {
        // no hacer nada
      }
      i = i + 1;
    }
    return divisores == 0;
  }
  ```
- Cuál es el invariante de este ciclo?
  ```

Otro ejemplo

- En cada iteración, la variable **divisores** cuenta cuántos divisores tiene **n** entre 0 e $i - 1$.

Otro ejemplo

- En cada iteración, la variable **divisores** cuenta cuántos divisores tiene **n** entre 0 e $i-1$.
- $I \equiv 2 \leq i \leq n \wedge$
  $\text{divisores} = \sum_{j=2}^{i-1}(\text{if } n \bmod j = 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$.

Otro ejemplo

- En cada iteración, la variable **divisores** cuenta cuántos divisores tiene **n** entre 0 e $i-1$.
- $I \equiv 2 \leq i \leq n \wedge$
  $\text{divisores} = \sum_{j=2}^{i-1} (\text{if } n \bmod j = 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$.
- Con esta definición, es fácil ver que se cumplen las tres condiciones (por qué?):
  1. $P_C \Rightarrow$ I,
  2. $\{I \wedge B\}$ `cuerpo del ciclo` $\{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,

Otro ejemplo

- En cada iteración, la variable **divisores** cuenta cuántos divisores tiene **n** entre 0 e $i - 1$.
- $I \equiv 2 \leq i \leq n \wedge$
  $\text{divisores} = \sum_{j=2}^{i-1}(\text{if } n \bmod j = 0 \text{ then } 1 \text{ else } 0 \text{ fi})\}$.
- Con esta definición, es fácil ver que se cumplen las tres condiciones (por qué?):
  1. $P_C \Rightarrow$ I,
  2. $\{I \wedge B\}$ `cuerpo del ciclo` $\{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
- Esto nos permite argumentar que el ciclo es correcto respecto de su especificación.

¿Qué pasa si el programa no termina?

- proc *sumar*(in $n : \mathbb{Z}$, out *result* : $\mathbb{Z}$){
  Pre $\{n \geq 0\}$
  Post $\{result = \sum_{i=1}^{n} i\}$
  }

- ```
  int sumar(int n) {
    int i = 1;
    int suma = 0;
    while( i <= n ) {
      // no hacer nada
    }
    return suma;
  }
  ```

- ¿Es este programa correcto con respecto a esta especificación?
  ```

¿Qué pasa si el programa no termina?

```
int sumar(int n) {
  int i = 1;
  int suma = 0;
  while( i <= n ) {
    // no hacer nada
  }
  return suma;
}
```

- Si $n < 0$, no se cumple la precondición
- Si $n == 0$, el programa retorna 0 (resultado correcto)
- Si $n > 0$, el programa no termina (y entonces no se proporciona ningún resultado!)

¿Qué pasa si el programa no termina?

- proc *sumar*(in $n : \mathbb{Z}$, out *result* $: \mathbb{Z}$){
  Pre $\{n \geq 0\}$
  Post $\{result = \sum_{i=1}^{n} i\}$
  }

- Esto quiere decir: "Si el programa `sumar` se comienza a ejecutar en un estado tal que $n \geq 0$, entonces:
  1. el programa termina
  2. el estado al finalizar la ejecución cumple que $result = \sum_{i=1}^{n} i$"

¿Qué pasa si el programa no termina?

- proc $sumar(\text{in } n : \mathbb{Z}, \text{out } result : \mathbb{Z})\{$
  Pre $\{n \geq 0\}$
  Post $\{result = \sum_{i=1}^{n} i\}$
  $\}$

- Esto quiere decir: "Si el programa `sumar` se comienza a ejecutar en un estado tal que $n \geq 0$, entonces:
  1. el programa termina
  2. el estado al finalizar la ejecución cumple que $result = \sum_{i=1}^{n} i$"

- Por lo tanto, la implementación propuesta `sumar` no cumple la especificación `sumar`!

¿Qué pasa si el programa no termina?

- proc $sumar$(in $n : \mathbb{Z}$, out $result : \mathbb{Z}$){
  Pre $\{n \geq 0\}$
  Post $\{result = \sum_{i=1}^{n} i\}$
  }

- Esto quiere decir: "Si el programa `sumar` se comienza a ejecutar en un estado tal que $n \geq 0$, entonces:
  1. el programa termina
  2. el estado al finalizar la ejecución cumple que $result = \sum_{i=1}^{n} i$"

- Por lo tanto, la implementación propuesta `sumar` no cumple la especificación `sumar`!

- Sin embargo, notar que esta implementación es parcialmente correcta (por qué?).