Corso di Programmazione

I Accertamento del 30 Novmbre 2004 / A

cognome e nome

Risolvi i seguenti esercizi, riporta le soluzioni in modo chiaro negli appositi spazi e giustifica sinteticamente le risposte. Dovrai poi consegnare queste schede con le soluzioni, avendo cura di scrivere il tuo nome nell'intestazione e su ciascun eventuale foglio aggiuntivo che si renda necessario.

1. Procedure in Scheme

Con riferimento alla procedura *h* così definita:

```
(define h
  (lambda (x y)  ; x, y >= 0 interi
    (if (or (< x 2) (< y 2))
        1
        (+ (h (- x 2) y) (h x (- y 2)))
    ) ))
```

calcola i risultati della valutazione di ciascuna delle seguenti espressioni Scheme:

(h 1 3) ⟶ *1* *(h 5 3)* ⟶ *3*

(h 3 3) ⟶ *2* *(h 7 3)* ⟶ *4*

Generalizza inoltre il risultato della valutazione dell'espressione *(h 2m+1 3)* per $m \geq 0$:

(h 2m+1 3) ⟶ *m + 1*

2. Procedure in Scheme

Completa la procedura *merge-strings* che, date due stringhe *u* e *v*, le fonde in un'unica stringa in modo tale che risultino sovrapposti un suffisso di *u* e un prefisso di *v* della massima lunghezza possibile. Per esempio, valutando l'espressione Scheme *(merge-strings "arti**cola**" "**cola**zione")* si vuole come risultato la stringa *"arti**cola**zione"*.

```
(define merge-strings

  (lambda (u v)

    (let ((m (string-length u))  (n (string-length v)) )

      (cond ((> n m)

             (string-append (merge-strings u (substring v 0  (- n 1) ))

                            (substring v (- n 1) n)))

            ((string=? v  (substring u (- m n) m) )

             u)

            (else

              (string-append (merge-strings u (substring v 0 (- n 1)))
              (substring v (- n 1) n))                                    )

      )) ))
```

3. Definizione di procedure in Scheme

Derfinisci formalmente un programma in Scheme per risolvere il seguente problema: date due stringhe *u*, *v*, si vuole conoscere la posizione della prima occorrenza di *u* come sottostringa di *v*, dove le posizioni sono numerate a partire da *1* (corrispondente al caso in cui *u* è un prefisso di *v*). Affinché *u* sia una sottostringa di *v*, i caratteri di *u* devono comparire in *v* nello stesso ordine e senza che vi siano altri caratteri interposti. Se *u* non è una sottostringa di *v*, allora il programma deve restituire il valore convenzionale *0*. Per esempio, se *u* e *v* sono le stringhe *"arte"* e *"p**arte**cipazione"*, rispettivamente, allora la posizione richiesta è *2*.

```
(define position
  (lambda (u v)
    (let ((m (string-length u)) (n (string-length v)))
      (cond ((> m n) 0)
            ((string=? u (substring v 0 m)) 1)
            (else
             (let ((p (position u (substring v 1 n))))
               (if (= p 0) 0 (+ p 1)) ))
            ))
    ))
```

4. Definizione di procedure in Scheme

Formalizza una procedura in Scheme che, data una funzione *f* definita nell'intervallo di numeri naturali *[0, p−1]* e a valori naturali e data l'ampiezza *p* dell'intervallo, assuma come valore la corrispondente estensione periodica di periodo *p*, cioè la funzione *g* definita su tutto il dominio dei naturali e con le seguenti proprietà: *g* coincide con *f* nell'intervallo *[0, p−1]* e *g(x+kp) = g(x)* per ogni *x, k* naturali.

```
(define periodic-ext
  (lambda (f p) ; p > 0 Nat, f: [0,p-1] -> Nat
    (lambda (n)
      (f (remainder n p))
      )
    ))
```

5. Dimostrazioni per induzione

Considera la procedura *h* dell'esercizio 1 e dimostra per induzione che, per qualunque numero naturale $m \geq 0$, il risultato della valutazione dell'espressione Scheme *(h m 4)* è:

$$\frac{(\lfloor \frac{m}{2} \rfloor + 1) \cdot (\lfloor \frac{m}{2} \rfloor + 2)}{2}$$

In particolare:

- Scrivi formalmente la proprietà che intendi dimostrare per induzione:

$$\forall\, m \in \mathbb{N}\,. \qquad (h\ m\ 4) \quad \longrightarrow \quad \frac{(\lfloor \frac{m}{2} \rfloor + 1) \cdot (\lfloor \frac{m}{2} \rfloor + 2)}{2}$$

- Scrivi formalmente la proprietà che esprime il caso base:

(h 0 4) $\longrightarrow$ 1
(h 1 4) $\longrightarrow$ 1

- Scrivi formalmente l'ipotesi induttiva: $preso\ m > 1\ in\ \mathbb{N}$

$$\forall\, x < m\,. \qquad (h\ x\ 4) \quad \longrightarrow \quad \frac{(\lfloor \frac{x}{2} \rfloor + 1) \cdot (\lfloor \frac{x}{2} \rfloor + 2)}{2}$$

- Scrivi formalmente la proprietà che si deve dimostrare come passo induttivo: $per\ m\ fissato\ sopra$

$$(h\ m\ 4) \quad \longrightarrow \quad \frac{(\lfloor \frac{m}{2} \rfloor + 1) \cdot (\lfloor \frac{m}{2} \rfloor + 2)}{2}$$

- Dimostra il caso base:

immediato

- Dimostra il passo induttivo: $occorre\ dimostrare\ preliminarmente\ che$

$$\forall\, m \in \mathbb{N}\,. \qquad (h\ m\ 2) \quad \longrightarrow \quad \lfloor \frac{m}{2} \rfloor + 1$$

6. Ricorsione di coda

Risolvi il problema posto dall'esercizio 2, utilizzando la *ricorsione di coda* al posto della ricorsione generale.

```
(define merge-strings-iter
  (lambda (u v) ; u, v stringhe
    (merge-str u v (string-length v))
    ))

(define merge-str
  (lambda (u v k)
    (let ((m (string-length u)) (n (string-length v)))
      (cond ((> k m)
             (merge-str u v (- k 1)))
            ((string=? (substring u (- m k) m) (substring v 0 k))
             (string-append u (substring v k n)))
            (else
             (merge-str u v (- k 1)))
    )) ))
```

Corso di Programmazione
I Accertamento del 30 Novmbre 2004 / B

cognome e nome

Risolvi i seguenti esercizi, riporta le soluzioni in modo chiaro negli appositi spazi e giustifica sinteticamente le risposte. Dovrai poi consegnare queste schede con le soluzioni, avendo cura di scrivere il tuo nome nell'intestazione e su ciascun eventuale foglio aggiuntivo che si renda necessario.

1. Procedure in Scheme

Con riferimento alla procedura *h* così definita:

```
(define h
  (lambda (x y)  ; x, y >= 0 interi
    (if (or (< x 2) (< y 2))
        1
        (+ (h (- x 2) y) (h x (- y 2)))
    ) ))
```

calcola i risultati della valutazione di ciascuna delle seguenti espressioni Scheme:

(h 2 1) ⟶ *1*   *(h 2 5)* ⟶ *3*

(h 2 3) ⟶ *2*   *(h 2 7)* ⟶ *4*

Generalizza inoltre il risultato della valutazione dell'espressione *(h 2 2n−1)* per $n > 0$:

(h 2 2n−1) ⟶ *n*

2. Procedure in Scheme

Completa la procedura *merge-strings* che, date due stringhe *u* e *v*, le fonde in un'unica stringa in modo tale che risultino sovrapposti un suffisso di *u* e un prefisso di *v* della massima lunghezza possibile. Per esempio, valutando l'espressione Scheme *(merge-strings "arti**cola**" "**cola**zione")* si vuole ottenere come risultato la stringa *"arti**cola**zione"*.

```
(define merge-strings

  (lambda (u v)

    (let ((m (string-length u))  (n (string-length v))  )

      (cond ((> m n)

             (string-append (substring u 0 1)

                            (merge-strings (substring u  1  m) v)))

            ((string=? u  (substring v 0 m)  )

             v)

            (else

              (string-append (substring u 0 1)
              (merge-strings (substring u 1 m) v))  )

      )) ))
```

3. Definizione di procedure in Scheme

Derfinisci formalmente un programma in Scheme per risolvere il seguente problema: date due stringhe *u*, *v*, si vuole conoscere quante volte *u* occorre come sottostringa di *v*. Affinché *u* sia una sottostringa di *v*, i caratteri di *u* devono comparire in *v* nello stesso ordine e senza che vi siano altri caratteri interposti; se *u* non è una sottostringa di *v*, il numero di occorrenze è *0*. Per esempio, se *u* e *v* sono le stringhe *"smo"* e *"co**smo**politi**smo**"*, rispettivamente, allora il numero di occorrenze è *2*.

```
(define occurrences
  (lambda (u v)
    (let ((m (string-length u)) (n (string-length v)))
      (cond ((> m n) 0)
            ((string=? u (substring v 0 m))
             (+ (occurrences u (substring v 1 n)) 1))
            (else
             (occurrences u (substring v 1 n)))
            ))
    ))
```

4. Definizione di procedure in Scheme

Formalizza una procedura in Scheme che, data una funzione *g* definita nell'intervallo di numeri naturali *[0, n]* e a valori naturali e dato un numero naturale positivo $q \leq n+1$, assuma come valore la funzione periodica *f* di periodo *q* con le seguenti proprietà: *f* è definita su tutto il dominio dei naturali, coincide con *g* nell'intervallo $[0, q-1]$ e $f(x+kq) = f(x)$ per ogni *x, k* naturali.

```
(define periodic-ext
  (lambda (g q) ; 0 < q <= n+1 Nat, g: [0,n] -> Nat
    (lambda (n)
      (g (remainder n q))
      )
    ))
```

5. Dimostrazioni per induzione

Considera la procedura *h* dell'esercizio 1 e dimostra per induzione che, per qualunque numero naturale $n \geq 0$, il risultato della valutazione dell'espressione Scheme *(h 5 n)* è:

$$\frac{(\lfloor \frac{n}{2} \rfloor + 1) \cdot (\lfloor \frac{n}{2} \rfloor + 2)}{2}$$

In particolare:

- Scrivi formalmente la proprietà che intendi dimostrare per induzione:

$$\forall\, n \in \mathbb{N}\,. \qquad \textit{(h 5 n)} \qquad \longrightarrow \qquad \frac{(\lfloor \frac{n}{2} \rfloor + 1) \cdot (\lfloor \frac{n}{2} \rfloor + 2)}{2}$$

- Scrivi formalmente la proprietà che esprime il caso base:

$$\textit{(h 5 0)} \qquad \longrightarrow \qquad 1$$
$$\textit{(h 5 1)} \qquad \longrightarrow \qquad 1$$

- Scrivi formalmente l'ipotesi induttiva: $preso\ n > 1\ in\ \mathbb{N}$

$$\forall\, x < n\,. \qquad \textit{(h 5 x)} \qquad \longrightarrow \qquad \frac{(\lfloor \frac{x}{2} \rfloor + 1) \cdot (\lfloor \frac{x}{2} \rfloor + 2)}{2}$$

- Scrivi formalmente la proprietà che si deve dimostrare come passo induttivo: $per\ n\ fissato\ sopra$

$$\textit{(h 5 n)} \qquad \longrightarrow \qquad \frac{(\lfloor \frac{n}{2} \rfloor + 1) \cdot (\lfloor \frac{n}{2} \rfloor + 2)}{2}$$

- Dimostra il caso base:

immediato

- Dimostra il passo induttivo: $occorre\ dimostrare\ preliminarmente\ che$

$$\forall\, n \in \mathbb{N}\,. \qquad \textit{(h 5 n)} \qquad \longrightarrow \qquad \lfloor \tfrac{n}{2} \rfloor + 1$$

6. Ricorsione di coda

Risolvi il problema posto dall'esercizio 2, utilizzando la *ricorsione di coda* al posto della ricorsione generale.

```
(define merge-strings-iter
  (lambda (u v) ; u, v stringhe
    (merge-str 0 u v)
    ))

(define merge-str
  (lambda (k u v)
    (let ((m (string-length u)) (n (string-length v)))
      (cond ((> (- m k) n)
             (merge-str (+ k 1) u v))
            ((string=? (substring u k m) (substring v 0 (- m k)))
             (string-append (substring u 0 k) v))
            (else
             (merge-str (+ k 1) u v))
    )) ))
```