Le s-espressioni

Un nuovo esempio completo: le s-espressioni

- `Sexpr`
- alberi binari (possibilmente “vuoti”) che hanno sulle foglie atomi (stringhe)
- sono la struttura dati base del linguaggio LISP
  - modificabile

Specifica di `Sexpr` 1

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
// costruttori
 public Sexpr ()
   // EFFECTS: inizializza this alla Sexpr vuota
 public Sexpr (String s)
   // EFFECTS: inizializza this alla foglia contenente s
// metodi
 public Sexpr cons (Sexpr s) throws
    NullPointerException
  // EFFECTS: costruisce un nuovo albero binario che ha
  // this come sottoalbero sinistro ed s come
  // sottoalbero destro. Se s è indefinita,
  // solleva NullPointerException
 public void rplaca (Sexpr s) throws
    NotANodeException
  // EFFECTS: rimpiazza in this il sottoalbero sinistro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked)
```

Specifica di Sexpr 2

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 public void rplacd (Sexpr s) throws
    NotANodeException
  // EFFECTS: rimpiazza in this il sottoalbero destro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked)
 public Sexpr car () throws NotANodeException
  // EFFECTS: ritorna il sottoalbero sinistro di this.
  // Se this non è un nodo binario solleva
  // NotANodeException (checked)
 public Sexpr cdr () throws NotANodeException
  // EFFECTS: ritorna il sottoalbero destro di this.
  // Se this non è un nodo binario solleva
  // NotANodeException (checked)
```

Specifica di `Sexpr` 3

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 public boolean nullS ()
  // EFFECTS: ritorna true se this è l'albero vuoto,
  // altrimenti ritorna false
 public boolean atom ()
  // EFFECTS: ritorna false se this è un albero binario,
  // altrimenti ritorna true.
 }
```

Implementazione di `Sexpr` 1

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
```

- ✓ la rep contiene
  - una variabile `boolean` che ci dice se l'albero è vuoto oppure consiste di una sola foglia
  - la variabile stringa che contiene l'eventuale stringa associata alla foglia
  - due (puntatori a) `Sexpr` che contengono i sottoalberi sinistro e destro, rispettivamente
- ✓ implementazione *ricorsiva*

Implementazione di `Sexpr` 2

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // la funzione di astrazione (ricorsiva!)
 // α(c) =
 //   se c.foglia && c.stringa = “” , Sexpr vuota
 //   se c.foglia && c.stringa = s, foglia s
 //   altrimenti è l’albero che ha come sottoalberi
 //   sinistro e destro α(c.sinistro) e α(c.destro)
 // l’invariante di rappresentazione (ricorsivo!)
 // I(c) = c.foglia oppure
 // (!c.foglia e c.sinistro != null e c.destro != null
 // e I(c.sinistro) e I(c.destro))
```

Implementazione di Sexpr 3

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
// costruttori
 public Sexpr ()
   // EFFECTS: inizializza this alla Sexpr vuota
  {foglia = true; stringa = ""; }
 public Sexpr (String s)
   // EFFECTS: inizializza this alla foglia contenente s
  {foglia = true; stringa = s; }
```

Implementazione di `Sexpr` 3.1

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // α(c) =
 //   se c.foglia && c.stringa = “”, Sexpr vuota
 //   se c.foglia && c.stringa = s, foglia s
 //   altrimenti è l’albero che ha come sottoalberi
 //   sinistro e destro α(c.sinistro) e α(c.destro)
 // I(c) = c.foglia oppure
 // (!c.foglia e c.sinistro != null e c.destro != null
 // e I(c.sinistro) e I(c.destro))
 public Sexpr ()
   // EFFECTS: inizializza this alla Sexpr vuota
  {foglia = true; stringa = ""; }
```

- ✓ l’invariante è soddisfatto (`foglia` è `true`)
- ✓ la specifica è soddisfatta

α(c) = Sexpr vuota, perché foglia && stringa = “”

Implementazione di `Sexpr` 3.2

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // α(c) =
 //   se c.foglia && c.stringa = “” , Sexpr vuota
 //   se c.foglia && c.stringa = s, foglia s
 //   altrimenti è l’albero che ha come sottoalberi
 //   sinistro e destro α(c.sinistro) e α(c.destro)
 // I(c) = c.foglia oppure
 // (!c.foglia e c.sinistro != null e c.destro != null
 // e I(c.sinistro) e I(c.destro))
public Sexpr (String s)
   // EFFECTS: inizializza this alla foglia contenente s
  {foglia = true; stringa = s; }
```

- ✓ l’invariante è soddisfatto (`foglia` è `true`)
- ✓ la specifica è soddisfatta

α(c) = foglia s, perché foglia && stringa = s

Implementazione di `Sexpr` 4

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 public Sexpr cons (Sexpr s) throws
    NullPointerException
  // EFFECTS: costruisce un nuovo albero binario che ha
  // this come sottoalbero sinistro ed s come
  // sottoalbero destro. Se s è indefinito,
  // solleva NullPointerException
 {if (s == null) throw new
      NullPointerException ("Sexpr.cons");
  Sexpr nuovo = new Sexpr();
  nuovo.sinistro = this; nuovo.destro = s;
  nuovo.foglia = false; return nuovo; }
```

Implementazione di `Sexpr` 4.1

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // I(c) = c.foglia oppure
 // (!c.foglia e c.sinistro != null e c.destro != null
 // e I(c.sinistro) e I(c.destro))
 public Sexpr cons (Sexpr s) throws
    NullPointerException
  {if (s == null) throw new
      NullPointerException ("Sexpr.cons");
   Sexpr nuovo = new Sexpr();
   nuovo.sinistro = this; nuovo.destro = s;
   nuovo.foglia = false; return nuovo; }
```

- ✓ *!nuovo.foglia*
- ✓ *nuovo.sinistro != null e nuovo.destro != null*
- ✓ *I(nuovo.sinistro) e I(nuovo.destro), perché per induzione I(this) e I(s)*

Implementazione di `Sexpr` 4.2

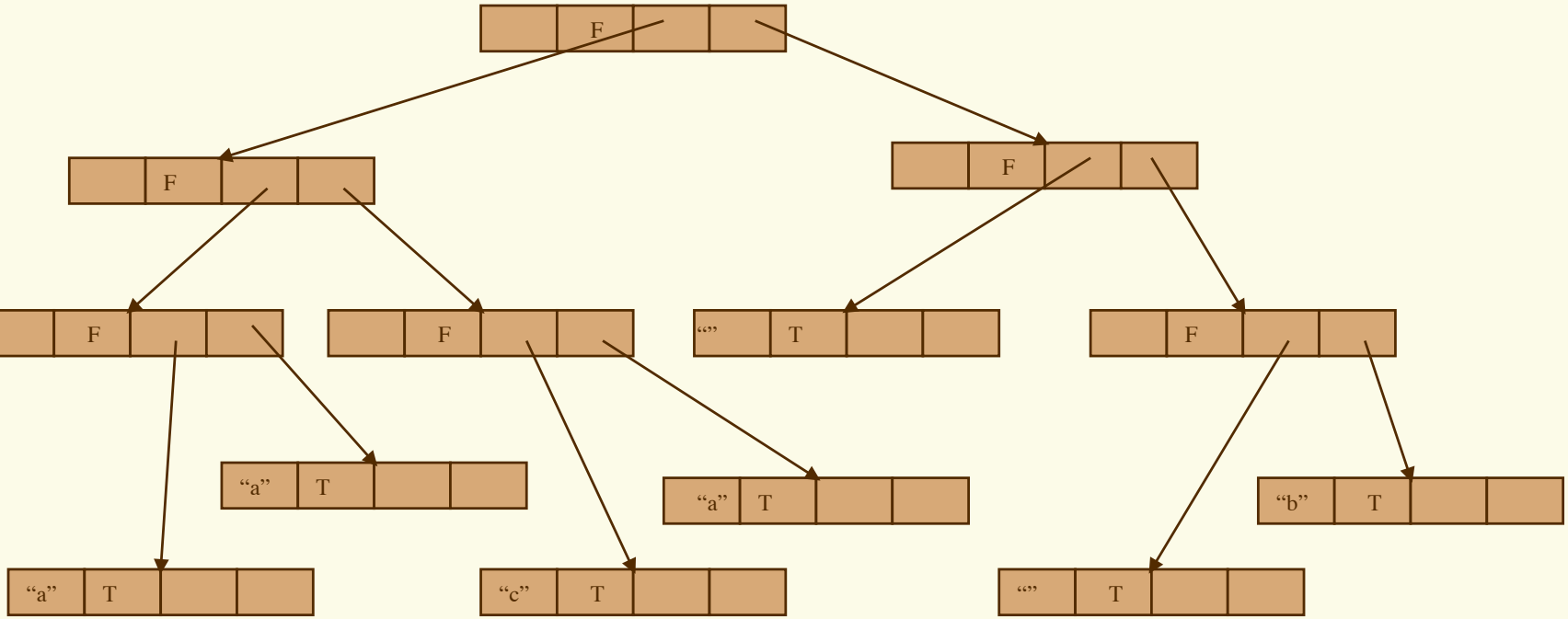
```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // α(c) =
 //   se c.foglia && c.stringa = "" , Sexpr vuota
 //   se c.foglia && c.stringa = s, foglia s
 //   altrimenti è l'albero che ha come sottoalberi
 //   sinistro e destro α(c.sinistro) e α(c.destro)
 public Sexpr cons (Sexpr s) throws
    NullPointerException
  // EFFECTS: costruisce un nuovo albero binario che ha
  // this come sottoalbero sinistro ed s come
  // sottoalbero destro. Se s è indefinito,
  // solleva NullPointerException
 {if (s == null) throw new
      NullPointerException ("Sexpr.cons");
  Sexpr nuovo = new Sexpr();
  nuovo.sinistro = this; nuovo.destro = s;
  nuovo.foglia = false; return nuovo; }
```

- ✓ correttezza ovvia

Come è fatta una `Sexpr`

- vediamo l'albero prodotto dalla espressione

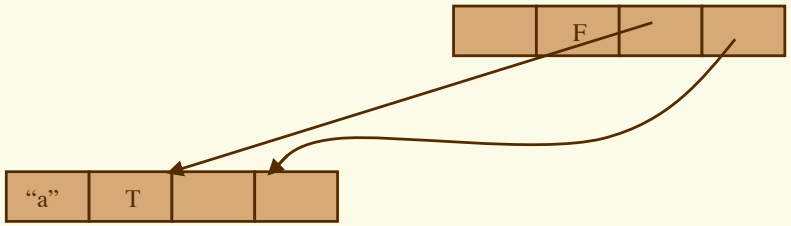
```
((((new Sexpr("a")).cons(new Sexpr("a"))).
  cons((new Sexpr("c")).cons(new Sexpr("a")))).
   cons((new Sexpr()).cons((new Sexpr()).cons(new Sexpr
   ("b"))))))
```



Come è fatta una `Sexpr` 2

- sottoalberi possono essere condivisi

```
Sexpr s1 = new Sexpr("a");
Sexpr s2 = s1.cons(s1);
```



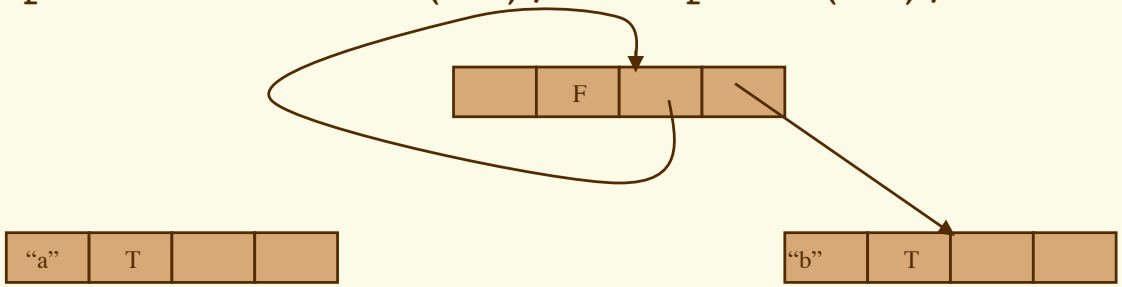
Implementazione di `Sexpr` 5

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 public void rplaca (Sexpr s) throws
    NotANodeException
  // EFFECTS: rimpiazza in this il sottoalbero sinistro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked)
  {if (foglia) throw new
            NotANodeException("Sexpr.rplaca");
    sinistro = s; return; }
```

Come è fatta una `Sexpr` 3

- usando le operazioni che modificano (`rplaca` e `rplacd`) si possono costruire strutture cicliche

```
Sexpr s1 = new Sexpr("a"); Sexpr s2 = new Sexpr(”b");
Sexpr s3 = s1.cons(s2); s3.rplaca(s3);
```



Implementazione di `Sexpr` 5.1

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // I(c) = c.foglia oppure
 // (!c.foglia e c.sinistro != null e c.destro != null
 // e I(c.sinistro) e I(c.destro))
 public void rplaca (Sexpr s) throws
    NotANodeException
    {if (foglia) throw new
           NotANodeException("Sexpr.rplaca");
    sinistro = s; return; }
```

- ✓ l'invariante non è soddisfatto!
  - `foglia` è `false`
  - `destro` non è modificato
  - `sinistro` soddisfa l'invariante per induzione ma nessuno ci garantisce che valga `sinistro != null`
  - dobbiamo modificare specifica e implementazione per garantire che valga `s != null`

Implementazione di `Sexpr` 5.1.1

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // I(c) = c.foglia oppure
 // (!c.foglia e c.sinistro != null e c.destro != null
 // e I(c.sinistro) e I(c.destro))
 public void rplaca (Sexpr s) throws
    NotANodeException, NullPointerException
    {if (foglia) throw new
           NotANodeException("Sexpr.rplaca");
     if (s == null) throw new
           NullPointerException("Sexpr.rplaca");
    sinistro = s; return; }
```

- ora l'invariante è soddisfatto
- le dimostrazioni servono anche a trovare gli errori!

Implementazione di `Sexpr` 5.2

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 // α(c) =
 //   se c.foglia && c.stringa = “” , Sexpr vuota
 //   se c.foglia && c.stringa = s, foglia s
 //   altrimenti è l’albero che ha come sottoalberi
 //   sinistro e destro α(c.sinistro) e α(c.destro)
 public void rplaca (Sexpr s) throws
    NotANodeException
  // EFFECTS: rimpiazza in this il sottoalbero sinistro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked). Se s è indefinito solleva
  // NullPointerException
  {if (foglia) throw new
           NotANodeException("Sexpr.rplaca");
   if (s == null) throw new
           NullPointerException("Sexpr.rplaca");
   sinistro = s; return; }
```

✓ soddisfa chiaramente la specifica

Implementazione di `Sexpr` 6

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 public void rplacd (Sexpr s) throws
    NotANodeException
  // EFFECTS: rimpiazza in this il sottoalbero destro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked). Se s è indefinito solleva
  // NullPointerException

  {if (foglia) throw new
           NotANodeException("Sexpr.rplacd");
   if (s == null) throw new
           NullPointerException("Sexpr.rplacd");
    destro = s; return; }
```

- ✓ cambiata come `rplaca`

Implementazione di `Sexpr` 7

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 public Sexpr car () throws NotANodeException
  // EFFECTS: ritorna il sottoalbero sinistro di this.
  // Se this non è un nodo binario solleva
  // NotANodeException (checked)
  {if (foglia) throw new
            NotANodeException("Sexpr.car");
   return sinistro; }
 public Sexpr cdr () throws NotANodeException
  // EFFECTS: ritorna il sottoalbero destro di this.
  // Se this non è un nodo binario solleva
  // NotANodeException (checked)
  {if (foglia) throw new
            NotANodeException("Sexpr.cdr");
   return destro; }
```

Implementazione di `Sexpr` 8

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 public boolean nullS ()
  // EFFECTS: ritorna true se this è l'albero vuoto,
  // altrimenti ritorna false.
  {if (! foglia) return false;
   if (stringa == "") return true; return false; }
 public boolean atom ()
  // EFFECTS: ritorna false se this è un albero binario,
  // altrimenti ritorna true.
  {return foglia;}
```