Un iteratore per le s-espressioni

Nuova specifica di `Sexpr`

```
public class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 public Sexpr ()
 public Sexpr (String s)
 public Sexpr cons (Sexpr s) throws NullPointerException
 public void rplaca (Sexpr s) throws NotANodeException
 public void rplacd (Sexpr s) throws NotANodeException
 public Sexpr car () throws NotANodeException
 public Sexpr cdr () throws NotANodeException
 public Iterator leaves ()
  // EFFECTS: ritorna un generatore che produrrà le foglie
  // nella frontiera di this (come Strings), da sinistra a
  // destra
  // REQUIRES: this non deve essere modificato finché
  // il generatore è in uso
  }
```

Implementazione del generatore 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;
 public Iterator leaves ()
  // REQUIRES: this non deve essere ciclico
  // EFFECTS: ritorna un generatore che produrrà le foglie
  // nella frontiera di this (come Strings), da sinistra a
  // destra
  // REQUIRES: this non deve essere modificato finché
  // il generatore è in uso
  {return new LeavesGen(this,numerofoglie());}
 private int numerofoglie () {
   if (foglia) {if (stringa == "") {return 0; }
     else {return 1; } }
   try {return (car().numerofoglie() +
     cdr().numerofoglie()); }
   catch (NotANodeException e) {return 0; } }
```

✓ il try & catch solo per evitare di dichiarare l'eccezione!

Implementazione del generatore 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;
 private static class LeavesGen  implements Iterator {
   private LeavesGen figlio; // sottogeneratore corrente
   private Sexpr io; // il mio albero
   private int quanti; // numero di elementi ancora da generare
   // la funzione di astrazione (ricorsiva!)
   // α(c) = se c.quanti = 0 allora [],
   // se c.quanti = 1 allora [c.io.stringa] ,
   // altrimenti α(c.figlio) + α(LeavesGen(c.destra))
```

Implementazione del generatore 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;
 private static class LeavesGen  implements Iterator {
   private LeavesGen figlio; // sottogeneratore corrente
   private Sexpr io; // il mio albero
   private int quanti; // numero di elementi ancora da generare
   // l'invariante di rappresentazione
   // I(c) = (c.quanti = 0 e c.io.foglia e
   // c.io.stringa = "") oppure
   // (c.quanti = 1 e c.io.foglia e c.io.stringa != "")
   // oppure
   // (c.quanti > 0 e c.io != null e c.figlio != null e
   // c.quanti = c.figlio.quanti +
   //   c.io.destra.numerofoglie())
```

Esercizio per casa

- implementare `repOK` e `toString` per il generatore e provarli su qualche esempio per vedere se sono giusti

Implementazione del generatore 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;
 private static class LeavesGen  implements Iterator {
   private LeavesGen figlio; // sottogeneratore corrente
   private Sexpr io; // il mio albero
   private int quanti; // numero di elementi ancora da generare
   LeavesGen (Sexpr s,int n)  {
    //REQUIRES: s != null
    quanti = n;
    if (quanti > 0)
     {io = s; if (io.foglia) return;
      try {figlio = new LeavesGen(io.car(),
        io.car().numerofoglie()); }
      catch (NotANodeException e) {}
      return; }
    return;}
```

- ✓ il try & catch solo per evitare di dichiarare l'eccezione!

Implementazione del generatore 4.1

```
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;
 private static class LeavesGen  implements Iterator {
    private LeavesGen figlio; // sottogeneratore corrente
    private Sexpr io; // il mio albero
    private int quanti; // numero di elementi ancora da generare
    // I(c) = (c.quanti = 0 e c.io.foglia e c.io.stringa = "") oppure
    // (c.quanti = 1 e c.io.foglia e c.io.stringa != "") oppure
    // (c.quanti > 0 e c.io != null e c.figlio != null e
    // c.quanti = c.figlio.quanti + c.io.destra.numerofoglie())

 LeavesGen (Sexpr s,int n)  {
     //REQUIRES: s != null
     quanti = n;
     if (quanti > 0)
     {io = s; if (io.foglia) return;
       try {figlio = new LeavesGen(io.car(),
        io.car().numerofoglie()); }
       catch (NotANodeException e) {}
       return; }
     return;}
```

✓ dimostrare che il costruttore soddisfa l'invariante

Implementazione del generatore 5

```
public class Sexpr {
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 private static class LeavesGen  implements Iterator {
   private LeavesGen figlio; // sottogeneratore corrente
   private Sexpr io; // il mio albero
   private int quanti; // numero di elementi ancora da generare
   public boolean hasNext() { return quanti > 0;}
   public Object next () throws NoSuchElementException {
    if (quanti == 0) throw new
       NoSuchElementException("Sexpr.leaves");
    quanti--; if (io.foglia) {return io.stringa;}
    try {return figlio.next(); }
    catch (NoSuchElementException e) {}
    try {figlio = new LeavesGen(io.cdr(),
      io.cdr().numerofoglie()); return figlio.next(); }
    catch (NotANodeException e) {
      throw new NoSuchElementException("Sexpr.leaves");}
```