```
──────────────── MODULE RemoveRedundantParens ────────────────
EXTENDS Integers, Sequences

CONSTANT TokId

Token ≜ [type : {"left", "right", "other"}, id : TokId]

RECURSIVE ParenDepth(_, _)
```

This is a comment in which $t.type > t.id$ so it looks nice.

```
ParenDepth(seq, i) ≜
  IF i = 0
    THEN 0
    ELSE CASE seq[i].type = "left"    → ParenDepth(seq, i − 1) + 1
             □seq[i].type   = "right"  → ParenDepth(seq, i − 1) − 1
             □seq[i].type   = "other"  → ParenDepth(seq, i − 1)

IsWellFormed(seq) ≜  ∧ ∀ i ∈ 1 . . Len(seq) : ParenDepth(seq, i) ≥ 0
                     ∧ ParenDepth(seq, Len(seq)) = 0
ExprOfMaxLen(n) ≜
  UNION {{s ∈ [1 . . i → Token] : IsWellFormed(s)} : i ∈ 0 . . n}
──────────────────────────────────────────────────────────────
```

The basic idea of the following algorithm is that it walks along the expression keeping *unmatchedLeft* equal to the sequence

$$\langle\langle i\_1, i\_1+1, \ldots, i\_1+j\_1\rangle, \langle i\_2, i\_2+2, \ldots, i\_2+j\_2\rangle, \ldots\rangle$$

where the element $\langle i\_k, \ldots, i\_k+j\_k\rangle$ means that there is a sequence of consecutive left parens at position $i\_k, \ldots, i\_k+j\_k$ for which the corresponding right parens have not been encountered. Left parens and "other" tokens are put into *out* as they are found. Left parens are removed from *out* when their matching right parens are found and the pair are found to be redundant. A right parens is also put into *out* immediately and removed when it is determined to be redundant, which will be on the next iteration. Note that left parens are removed from out from right to left, so the index of the left *paren* that is to be removed has not been changed because of the previous removal of a left *paren*.

```
–algorithm Remove {
 variables in ∈ ExprOfMaxLen(5),
           out = ⟨⟩,
           unmatchedLeft = ⟨⟩,
           i = 1,
           justFoundLeft = FALSE,
             \ * true means that the token at i − 1 is a left paren
           justFoundRight = FALSE;
             \ * true means that the token at i − 1 is a right paren
{while(i ≤ Len(in)){
   if(in[i].type = "left"){
     if(justFoundLeft){
       unmatchedLeft[Len(unmatchedLeft)] :=
         Append(unmatchedLeft[Len(unmatchedLeft)], i);
       out := Append(out, in[i])
```

```
        }
        else{
        }
      }
      else if(in[i].type = "right"){
      }
      else{
      };
      i := i + 1;
    }
  }
}
```

* Modification History
* Last modified *Mon Dec* 19 18:17:21 *PST* 2011 by *lamport*
* Created *Mon Dec* 19 17:20:10 *PST* 2011 by *lamport*