16.33 LALR: A parser generator

Author: Arthur Norman

This code serves the same sort of purpose as the well known utilities `yacc` and `bison`, in that it accepts a specification of a contex free grammar and builds a parser for the language that it describes. The techniques used are those described in standard texts (eg Aho, Lam, Sethi and Ullman) as “LALR”.

Grammars are presented to the code here as lists. In a grammar strings will denote terminal symbols and most symbols non-terminals. A small number of special symbols will stand for classes of terminal that are especially useful.

A grammar is given as a list of rules. Each rule has a single non-terminal and then a seqence of productions for it. Each production can optionally be provided with an associated semantic action.

```
GRAMMAR ::= "(" GTAIL
        ;

GTAIL   ::= ")"         % Sequence of rules
        |   RULE GTAIL
        ;

RULE    ::= "(" nonterminal RULETAIL
        ;

RULETAIL::= ")"         % Sequence of productions
        | PRODUCTION RULETAIL
        ;

PRODUCTION ::= "(" "(" PT1 PT2
        ;

PT1     ::= ")"         % Symbols to make one production
        | nonterminal PT1
        | terminal PT1
        ;

PT2     ::= ")"         % Semantic actions as Lisp code
        | lisp-s-expressoin PT2
        ;
```

Before specifying a grammar it is possible to declare the precedence and associativity of some of the terminal symbols it will use. Here is an example:

```
lalr_precedence '("." !:right "^" !:left ("*" "/") ("+" "-"));
```

will arrange that the token "`.`" has highest precedence and that it is left associative. Next comes "^" which is right associative. Both "*" and "/" have the same precedence and both are left associative, and finally "+" and "−" are also equal in precedence but lower than "*".

With those precedences in place one could specify a grammar by

```
lalr_construct_parser '(
   (S   ((!:symbol))
        ((!:number))
        ((S "." S))
        ((S "^" S))
        ((S "*" S))
        ((S "/" S))
        ((S "+" S))
        ((S "-" S))));
```

The special markers `!:symbol` and `!:number` will match any symbols or numbers – for commentary on what count as such see the discussion of the lexter later on. The strings stand for fixed tokens and by virtue of them being used in the grammer the lexer will recognise them specially.