relabel Grammar

Taken from www.inf.puc-rio.br/~roberto/lpeg/re.html (see Patterns at the bottom) with relabel additions highlighted in green and corrections highlighted in yellow.

```
pattern         <- exp !.
exp             <- S (grammar / alternative)

alternative     <- seq ('/' labels S seq)+
                / seq ('/' S seq)*
seq             <- prefix*
prefix          <- '&' S prefix / '!' S prefix / suffix
suffix          <- primary S (([+*?]
                          / '^' [+-]? num
                          / '->' S (string / num / '{}' / name)
                          / '=>' S name) S)*

primary         <- '(' exp ')' / string / class / defined
                / '%' labels
                / '{:' (name ':')? exp ':}'
                / '=' name
                / '{}'
                / '{~' exp '~}'
                / '{|' exp '|}'
                / '{' exp '}'
                / '.'
                / name !arrow
                / '<' name '>'

grammar         <- definition+
definition      <- name arrow exp

class           <- '[' '^'? item (!']' item)* ']'
item            <- defined / range / .
range           <- . '-' [^]]
labels          <- '{' S label (S ',' S label)* S '}'

S               <- (%s / '--' [^%nl]*)*
name            <- [A-Za-z_][A-Za-z0-9_]*
arrow           <- S '<-'
num             <- [0-9]+
string          <- "'" [^']* "'" / '"' [^"]* '"'
defined         <- '%' name
label           <- num / name
```

Current Status of Error Reporting in relabel

Note: only syntax errors are covered in this report, other errors, such as duplicate rules or unknown predefined patterns, are out of scope

Error Detection

- Very Limited
- Detects errors in sequences (prevents backtracking out of a sequence)

Error Reporting

- Raises an error
- Does not provide line or column number
- Reports the unmatched text (truncates it if greater than 20 characters)

Error Recovery

- None - stops on the first error

Sample

```
-- the closing parenthesis is missing
re.compile [[ "a" / ("b" / "c" ]]
--> pattern error near '("b" / "c" '
```

Annotated relabel Grammar (without Recovery)

Changes are highlighted in red and [pattern]^label means that the label is thrown upon failure of the pattern (not to be confused with character classes, which do not have the label). See the next page for more info.

```
pattern         <- [exp]^NoPatt S [!.]^ExtraChars
exp             <- S (grammar / alternative)

alternative     <- seq (S '/' labels? [S seq]^ExpPatt1)*

seq             <- prefix+
prefix          <- '&' [S prefix]^ExpPatt2 / '!' [S prefix]^ExpPatt3 / suffix
suffix          <- primary (S ([+*?]
                           / '^' [+-]? [num]^ExpNum
                           / '->' [S (string / num / '{}' / name)]^ExpCap
                           / '=>' [S name]^ExpName1))*

primary         <- '(' [exp]^ExpPatt4 [S ')']^MisClose1 / string / class / defined
                / '%' ['{']^ExpNameOrLab [S label]^ExpLab1 [S '}']^MisClose7
                / '{:' (name ':')? [exp]^ExpPatt5 [S ':}']^MisClose2
                / '=' [name]^ExpName2
                / '{}'
                / '{~' [exp]^ExpPatt6 [S '~}']^MisClose3
                / '{|' [exp]^ExpPatt7 [S '|}']^MisClose4
                / '{' [exp]^ExpPattOrClose [S '}']^MisClose5
                / '.'
                / name !arrow
                / '<' [name]^ExpName3 ['>']^MisClose6

grammar         <- definition (S definition)*
definition      <- name arrow [exp]^ExpPatt8

class           <- '[' '^'? [item]^ExpItem (!']' item)* [']']^MisClose8
item            <- defined / range / [^%nl]
range           <- . '-' [^]]
labels          <- '{' [S label]^ExpLab1 (S ',' [S label]^ExpLab2)* [S '}']^MisClose7

S               <- (%s / '--' [^%nl]*)*
name            <- [A-Za-z_][A-Za-z0-9_]*
arrow           <- S '<-'
num             <- [0-9]+
string          <- "'" [^'%nl]* ["'"]^MisTerm1 / '"' [^"%nl]* ['"']^MisTerm2
defined         <- '%' name
label           <- num / name
```

Labels and Error Messages

1. NoPatt - no pattern found
2. ExtraChars - unexpected characters after the pattern

3. ExpPatt1 - expected a pattern after '/' or the label(s)
4. ExpPatt2 - expected a pattern after '&'
5. ExpPatt3 - expected a pattern after '!'
6. ExpPatt4 - expected a pattern after '('
7. ExpPatt5 - expected a pattern after ':'
8. ExpPatt6 - expected a pattern after '{~'
9. ExpPatt7 - expected a pattern after '{|'
10. ExpPatt8 - expected a pattern after '<-'
11. ExpPattOrClose - expected a pattern or closing '}' after '{'

12. ExpNum - expected a number after '^', '+' or '-' (no space)
13. ExpCap - expected a string, number, '{}' or name after '->'
14. ExpName1 - expected the name of a rule after '=>'
15. ExpName2 - expected the name of a rule after '=' (no space)
16. ExpName3 - expected the name of a rule after '<' (no space)
17. ExpLab1 - expected at least one label after '{'
18. ExpLab2 - expected a label after the comma
19. ExpNameOrLab - expected a name or label after '%' (no space)
20. ExpItem - expected at least one item after '[' or '^'

21. MisClose1 - missing closing ')'
22. MisClose2 - missing closing ':}'
23. MisClose3 - missing closing '~}'
24. MisClose4 - missing closing '|}'
25. MisClose5 - missing closing '}'
26. MisClose6 - missing closing '>'
27. MisClose7 - missing closing '}'
28. MisClose8 - missing closing ']'

29. MisTerm1 - missing terminating single quote
30. MisTerm2 - missing terminating double quote

Examples of Each Error

1. **NoPatt** - no pattern found
    - ~
    - ???
2. **ExtraChars** - unexpected characters after the pattern
    - 'p'~
    - 'p'?$?
3. **ExpPatt1** - expected a pattern after '/' or the label(s)
    - 'p' /{1}
    - 'p' /{1} /{2} 'q'
    - 'p' /
    - 'p' / / 'q'
4. **ExpPatt2** - expected a pattern after '&'
    - &
    - & / 'p'
    - 'p' &
    - 'p' / & / 'q'
    - &&
    - !&
5. **ExpPatt3** - expected a pattern after '!'
    - !
    - ! / 'p'
    - 'p' !
    - 'p' / ! / 'q'
    - !!
    - &!
6. **ExpPatt4** - expected a pattern after '('
    - ()
    - ($$$)
7. **ExpPatt5** - expected a pattern after ':'
    - {: *** :}
    - {:group: *** :}
    - x <- {:x:}
8. **ExpPatt6** - expected a pattern after '{~'
    - {~~}
    - { {~ } ~}
    - {~ ^_^ ~}
9. **ExpPatt7** - expected a pattern after '{|'
    - {| |}
    - {|@|}
10. **ExpPatt8** - expected a pattern after '<-'
    - S <-
    - S <- 'p' T <-
11. **ExpPattOrClose** - expected a pattern or closing '}' after '{'
    - {0}

    - { :'p': }
12. **ExpNum** - expected a number after '^', '+' or '-' (no space)
    - ‘p' ^ n
    - 'p'^+(+1)
    - 'p'^-/'q'
13. **ExpCap** - expected a string, number, '{}' or name after '->'
    - 'p' -> {
    - 'p' -> {'q'}
    - 'p' -> / 'q'
    - 'p' -> [0-9]
14. **ExpName1** - expected the name of a rule after '=' (no space)
    - 'p' =>
    - 'p' => 'q'
15. **ExpName2** - expected the name of a rule after '=' (no space)
    - '<' {:tag: [a-z]+ :} '>' '<' = '>'
    - '<' {:tag: [a-z]+ :} '>' '<' = tag '>'
16. **ExpName3** - expected the name of a rule after '<' (no space)
    - <>
    - <123>
    - < hello >
    - <<S>>
17. **ExpLab1** - expected at least one label after '{'
    - 'p' /{} 'q'
    - %{ 'label' }
18. **ExpLab2** - expected a label after the comma
    - 'p' /{1,2,3,} 'q'
    - %{ a,,b,,c }
19. **ExpNameOrLab** - expected a name or label after '%' (no space)
    - % s
    - % {1}
20. **ExpItem** - expected at least one item after '[' or '^'
    - "p" [
      abc
      ] "q"
21. **MisClose1** - missing closing ')'
    - ('p' ('q' / 'r')
22. **MisClose2** - missing closing ':}'
    - {: group: 'p' :}
    - S <- {: 'p' T <- 'q'
23. **MisClose3** - missing closing '~}'
    - 'p' {~ ('q' 'r') / 's'
24. **MisClose4** - missing closing '|}'
    - 'p' {| 'q' / 'r' }
25. **MisClose5** - missing closing '}'
    - { 'p'
26. **MisClose6** - missing closing '>'
    - <patt
    - <insert your name here>

27. **MisClose7** - missing closing '}'
    - '{' %{ a, b '}'
28. **MisClose8** - missing closing ']'
    - [
    - [^
    - []
    - [^]
    - [_-___-_|
29. **MisTerm1** - missing terminating single quote
    - 'That is the question...
30. **MisTerm2** - missing terminating double quote
    - Q <- "To be or not to be...

Changes in the Grammar

1. Unification of labeled and unlabeled choice
    - This fixes a bug of errors being reported twice (once for the labeled path, once for the unlabeled path)
    - This also makes the grammar more flexible as now labeled and unlabeled choices can be mixed within the same rule, which may be convenient
    - This should not break any existing patterns
    - Performance should not change since unlabeled choices are still constructed when there are no labels (instead of labeled choice catching the fail label)

2. Changing `seq` to require at least one prefix (* -> +)
    - There's really no reason to have an empty pattern
    - Empty patterns seem more like an error, and so it shouldn't be allowed ex. ( )
    - If one needs an empty pattern, one can use empty strings instead

3. Extracting trailing whitespace in `suffix`
    - This allows clearer errors to be reported by keeping the error position close to the last accepted token making sure that it is not reported after a comment or on a different line
    - This should not break any existing patterns

4. Only allowing a single label to be thrown
    - This was done to follow the change made in LPegLabel during the rewrite of only one label being allowed to be thrown at a time

5. Limiting strings and character classes to a single line
    - This allows better error messages to be reported by preventing missing terminators from consuming all of the input
    - In practice, strings and character classes stay on one line
    - If newlines are needed, one can use `%nl`, which is arguably more obvious and readable

Error Reporting and Error Recovery

Error Reporting

- Along with the error message, the exact position of the error (in the string) is reported (line and column number)
- The whole line of the error is also reported upon encountering an error
- For clarity, there is a caret (^) pointing to the column of the error in the line
- Whole lines are reported as this is both simpler and easier for users to recognize

Recovery Strategies

1. Skipping to the next safe point
    - This strategy consumes input until the next slash or next rule/definition. This is done as these positions are considered safe points for continuing parsing.
    - Skipping is done intelligently so that slashes and definitions inside a group are skipped (since continuing in that point will cause an error upon encountering the closing bracket).
    - By consuming input, there is a possibility of skipping other errors but this also lowers the risk of generating erroneous error messages.
      Labels under this strategy: ExpPatt1, ExpPatt2, ExpPatt3, ExpPatt4, ExpPatt5, ExpPatt6, ExpPatt7, ExpPatt8, ExpPattOrClose, ExpNum, ExpCap, ExpName1, ExpName2, ExpName3, ExpLab1, ExpLab2, ExpNameOrLab, ExpItem, MisClose6, MisClose7

2. Ignore the error
    - This strategy ignores the error by not consuming anything (aside from what has already been consumed before the error), and just allows the parser to continue. Of course, the error is still recorded for reporting later on.
    - By simply continuing, more errors can be detected and reported.
    - In some scenarios, this may cause errors to cascade, which is why it is only limited to a chosen few labels (the closing bracket delimiters).
      Labels under this strategy: MisClose1, MisClose2, MisClose3, MisClose4, MisClose5

3. Ignore the error and move the position of the last error to point at the start
    - Similar to the previous strategy, the error is ignored and nothing else is consumed.
    - The position of the last error is moved to point at the start for clarity.
    - This strategy is used when failing to terminate strings and character classes because they allow almost all characters to be consumed and so the original reported position may drift away too far from the supposed location of termination.
    - This strategy was inspired by how other compilers (like gcc) deal with unterminated strings
      Labels under this strategy: MisTerm1, MisTerm2, MisClose8

Notes on the Implementation

Recording of Errors

A function named “expect” that takes a pattern and a label is defined to create a pattern that would record the error and throw the label when failing to match the pattern. Recording is done through a match-time capture, which is simply used to call a function that stores the label and error position in the global array of syntax errors. Aside from recording, this function also makes the throwing of a label upon failure clear (compared to using “raw” ordered choice).

Error Recovery

Error recovery is done through the use of labeled choice. The labeled choice takes the possibly failing pattern as its first choice, and the recovery strategy as its second choice. The recovery strategy always matches and returns a dummy pattern for capture so that construction of the pattern may continue without error.

Handling of Non-syntax Errors

Non-syntax errors may occur during the construction of the pattern, which is done while parsing. To keep things simple, non-syntax errors are only reported if there are no syntax errors. Only one non-syntax error can be thrown, which follows the original behaviour of LPeg. It is possible to add recovery for these errors, but it would complicate the implementation and possibly report erroneous errors due to the cascading. Unlike syntax errors, non-syntax errors do not specify the position of the error. In future versions of LPeg(Label), these could be improved. However, at least for now, they are out of the scope of this rewrite, and in practice, it may be overkill as patterns aren’t usually very long or complex (compared to code for example).

Comparison of Error Reporting Before and After

Example Grammar

```
  A <- 'A' /{'lab'} B / !
  B <- %{1, 2 3} 'b' / '6' & / 'B'
  C <- A^B
```

Before

```
lua: pattern error near '!
  B <- %{1, 2 3} 'b...'
```

After

```
lua: syntax error(s) in pattern
L1:C14: expected at least one label after '{'
  A <- 'A' /{'lab'} B / !
             ^
L1:C26: expected a pattern after '!'
  A <- 'A' /{'lab'} B / !
                          ^
L2:C15: missing closing '}'
  B <- %{1, 2 3} 'b' / '6' & / 'B'
              ^
L2:C29: expected a pattern after '&'
  B <- %{1, 2 3} 'b' / '6' & / 'B'
                               ^
L3:C10: expected a number after '^', '+' or '-' (no space)
  C <- A^B
         ^
```

Comments

Before the rewrite, only one error was reported (as there is no error recovery) while the new version successfully reports all errors. The original error reporting did not provide information on the precise position of the error nor did it specify the specific error that encountered. In contrast, the new version properly reports the error position and gives a helpful error message although the 3rd error is only half-right as the real error is a missing comma. A longer list of possible expected tokens may be provided to make the error message more accurate, but this may, ironically, also cause more confusion. It should not be difficult, however, for the user to determine the real error from the information provided (especially since the error position has been shown).

A simple benchmark was also performed using the original LPeg and LPegLabel tests. There was no measurable difference in performance before and after the rewrite.

Limitations and Other Possible Improvements

- Although the rewrite is well-tested, it would be good to test it on more real-world test cases to better understand how the error reporting and error messages can be further improved
- Since re(label) patterns are used as an Embedded Domain Specific Language (EDSL), the line numbers reported could be confusing as they are the line numbers relative to the string and not the file. It may be more convenient for the user if it were relative to the file instead.
- The grammar could be further annotated to include even more labels to give more specific error messages. For example, the ExpNum label could be split into 3 different labels to handle the different cases specifically. Doing so will clutter the grammar though, so it's better to leave it as is as long as the error message is clear enough.
- Many labels are quite similar to each other and could be grouped together to conserve labels. However, doing so would result to less specific errors. To solve this problem, a more complex error throwing mechanism could be created to attach additional information to the error thrown. For example, we could merge the different missing closing bracket labels into one and allow the expect function to have carry an additional attachment, which would be the closing bracket missing in this example. It is not yet clear if the trade-offs are worth it for this.
- Often compilers limit the number of errors reported so that the programmer will not be overwhelmed by them and to prevent further cascading of errors. Such limits could be implemented for relabel as well but shouldn't be needed.
- In a few cases, the parser will erroneously report errors after recovery. It would be good to study these cases to see if there is a way to prevent them from happening while still reporting all errors accurately. Some cases may be difficult or even impossible to handle though; for example, multiple strings in the same line where one of them is not terminated -- it'd be impossible to know for sure where it should've terminated. Perhaps in the far far future, A.I. will be able to solve this.
- The procedure for getting the lines in the file for reporting is very simple but also not very efficient (it splits all lines in the file and keeps them all in memory). Optimizations can be done to improve this so that less memory will be consumed. These optimizations are probably unnecessary for the scale of grammars and files LPeg usually handles though.
- The parser could look at the next few tokens found in the position of the error to provide a better guess and more specific error message on what error the user did. This is extremely hard to get right though, and could even lead to more confusing errors (when it makes a wrong guess). It's probably not worth pursuing this and best to keep the error messages conservative.