AST generator specification

Evgenii Balai

September 19, 2014

Contents

1 Vocabulary

- *identifier* - a sequence of alphanumeric or underscore characters starting with a letter.
- *numeral* - a sequence of characters used to represent a number. Numerals can be in one of the following forms:
  1. $d_1 \ldots d_k$
  2. $d_1 \ldots d_n.d_1 \ldots d_m$

  possible preceded by a minus sign, where each d_i is a digit, $k > 0, n \geq 0$ and $m > 0$.

2 Input Specification

The input consists of the following:

1. lexicon file (defined in section 2.1)
2. grammar file (defined in section 2.2)
3. source file (defined in section 2.3)

2.1 Lexicon file

A lexicon file is an ASCII file that contains a specification of types of terminal symbols (*lexems*) that may occur in a program file. Each lexem type is declared as

$$lexem\_type\_name = regex \tag{1}$$

where *lexem_type_name* is an identifier and *regex* is a regular expression following python syntax[1] whose special characters can be ., $*$, $+$, ?, $\{m\}$, $\{m, n\}$, [],\ or |.

We will say that statement (1) *defines* a lexem type named *lexem_type_name*.

Lexicon file may contain multiple declarations of the form (1), one per line, where no lexem type name occurs in the left hand side of a statement more than once, and no lexem type name is a member of the set $\{id, num\}$.

2.2 Grammar File

A grammar file is an ASCII file that contains a specification of non-terminals occurring in the produced abstract syntax tree as well as the desired structure of the tree. Every grammar file must be associated with a lexicon file. We will refer to the set of lexem type names defined in the file as S_L.

Non-terminals are specified by statements of the form

$$nt[\mathscr{L}] = \alpha_1\ \alpha_2\ \ldots\ \alpha_n \tag{2}$$

where

1. nt is an identifier which is not a member of the set $\{id, num\} \cup S_L$;
2. each α_i is an identifier;
3. $\mathscr{L}$ is a space-separated sequence of the form $id\ \alpha_{k_1}\ \ldots\ \alpha_{k_m}$ where id is an identifier and each α_{k_i} is an element of the sequence $\alpha_1 \ldots \alpha_n$ on the right hand side of (2).

A grammar file may contain a sequence of statements of the form (2) such that

1. each α_i is in one of the forms b or b_r where
   - b is a name of a non-terminal whose name appears in the left hand side of another statement of the form (2), or is an element of the set of identifiers $\{id, num\} \cup S_L$
   - r is a natural number in the range $1..n$
2. if α_i and α_j are two different elements of the sequence on the right hand side of (2), then
   - at least one of them is of the form b_r, where r is a natural number in the range $1..n$
   - if α_i is of the form b_r1 and α_j is of the form b_r2, where $r1$ and $r2$ are two natural numbers in the range $1..n$, then $r1 \neq r2$.

3. if there are two statements

$$nt^1[\mathscr{L}] = \alpha_1^1\ \alpha_2^1\ \ldots\ \alpha_{n_1}^1$$

and

$$nt^2[\mathscr{L}] = \alpha_1^2\ \alpha_2^2\ \ldots\ \alpha_{n_2}^2$$

in the grammar file, then nt^1 is not of the form nt^2_r, where r is a natural number.

2.3 Source File

A source file is a file containing arbitrary collection of ASCII characters.

2.4 Input Example

In this example we will define Algebraic Chess Notation [2] by means of lexicon and grammar defined in sections 2.1 and 2.2 and give an example of a game described in this notation.

2.4.1 Lexicon file

```
figure = K|Q|R|B|N
file = [a-h]
rank = [1-8]
cell = [a-h][1-8]
capture_char = x
space = \s
spaces = \s+
dot = \.
short_castling =
en_passant = e\.p\.
natural_number = [1-9][0-9]+
long_castling = 0-0-0
short_castling = 0-0
plus = \+
pound_sign = #
game_over = 1-0|1/2-1/2|0-1
```

2.4.2 Grammar file

```
game[game move_d] = move_d
game[game move_d game] = move_d  spaces game
move_d[move number move] = natural_number dot space move_1 move_2
move_d[move natural_number end_of_game] = natural_number dot space move_1 game_over
move[pawn_move cell] = cell
move[move figure cell] = figure_spec cell
move[capture figure_1 cell] = figure_spec capture_char cell
move[pawn_capture file cell] = file capture_char cell
move[pawn_special_capture] = file capture_char cell en_passant
move[promotion cell figure] = cell figure
move[castling long_castling] = long_castling
```

```
move[castling short_castling] = short_castling
check[check move] = move plus
checkmate[checkmate move] = move pound_sign
figure_spec[fig figure] = figure
figure_spec[fig figure file] = figure file
figure_spec[fig figure file] = figure rank
figure_spec[fig figure cell] = figure cell
```

2.4.3 Source File

```
1. e4 c5 2. Nf3 d6 3. Bb5+ Bd7 4. Bxd7+ Qxd7 5. c4 Nc6 6. Nc3 Nf6
7. 0-0 g6 8. d4 cxd4 9. Nxd4 Bg7 10. Nde2 Qe6  11. Nd5 Qxe4
12. Nc7+ Kd7 13. Nxa8 Qxc4 14. Nb6+ axb6 15. Nc3 Ra8 16. a4 Ne4
17. Nxe4 Qxe4 18. Qb3 f5 19. Bg5 Qb4 20. Qf7 Be5 21. h3 Rxa4
22. Rxa4 Qxa4 23. Qxh7 Bxb2 24. Qxg6 Qe4 25. Qf7 Bd4 26. Qb3 f4
27. Qf7 Be5 28. h4 b5 29. h5 Qc4 30. Qf5+ Qe6 31. Qxe6+ Kxe6
32. g3 fxg3 33. fxg3 b4 (the World Team did not trust 33...Bxg3
34.h6 Be5 35.h7 Bg7 36.Rf8 b4 37.h8=Q Bxh8 38.Rxh8) 34. Bf4 Bd4+
35. Kh1! b3 36. g4 Kd5 37. g5 e6 38. h6 Ne7 39. Rd1 e5 40. Be3 Kc4
41. Bxd4 exd4 42. Kg2 b2 43. Kf3 Kc3 44. h7 Ng6 45. Ke4 Kc2
46. Rh1 d3 47. Kf5 b1=Q 48. Rxb1 Kxb1 49. Kxg6 d2 50. h8=Q d1=Q
51. Qh7 b5 52. Kf6+ Kb2 53. Qh2+ Ka1 54. Qf4 b4 55. Qxb4 Qf3+
56. Kg7 d5 57. Qd4+ Kb1 58. g6 Qe4 59. Qg1+ Kb2 60. Qf2+ Kc1
61. Kf6 d4 62. g7 1{0
```

3 Output specification

The output is obtained in two steps

1. *Lexing*. *Lexer Module* takes a source file and lexicon file and outputs a sequence of annotated lexems as specified in section 3.1.
2. *Parsing*. *Parser Module* takes an output of the lexer module and grammar file as an input and returns an abstract tree as specified in section 3.2.

3.1 Lexing

Given a lexicon file L, by S_L we denote the set of all names of lexem types defined in L. The *lexicon dictionary* is a dictionary that maps each member of the $S_L \cup \{id, num\}$ into a regular expression as follows:

1. if $l \in S_L$, the regular expression is on the right hand side of the statement

   $$l = expr$$

   appearing in L;
2. if l is id, the regular expression is

   $$[a-z][a-z\_]+$$

3. if l is num, the regular expression is

$$-?[1-9][0-9]+|-?0\backslash.[0-9]+|-?[1-9][0-9]+\backslash.[0-9]+$$

For each member l of the set $S_L \cup \{id, num\}$ by $\mathscr{E}[l]$ we denote the regular expression that corresponds to l in the lexicon dictionary.

We will say that a string S *matches* a regular expression E if ???

3.2 Parsing

References

[1] Python Software Foundation. Regular expression operations – python 3.4.1 documentation. https://docs.python.org/3.4/library/re.html.

[2] Wikipedia. Algebraic notation (chess). http://en.wikipedia.org/wiki/Algebraic_notation_%28chess%29.