Creating a lexical analyser

Marcin Kuta

Theory of Compilation
Laboratory 1

Basic concepts

Basic concepts

Pattern	`[0-9]+`
Token	`INTNUM`
Lexem	`1920`

Example of specification

Example of specification

```
a      { Action1 }
abb    { Action2 }
a*b+   { Action3 }
```

Input: abb

Possible tokenizations

```
a|bb|   { Action1, Action3 }
abb|    { Action2 }
abb|    { Action3 }
```

Rules of lexical specification

Two rules

1. Principle of maximal match
2. Detailed specifications before general specification
    - If an input string matches two patterns, the pattern which appears earlier in the specification list is chosen

Example of specification

Example of specification

```
a       { Action1 }
abb     { Action2 }
a*b+    { Action3 }
```

Input: `abb`

Possible tokenizations

```
a|bb| Action1, Action3
abb| Action2
abb| Action3
```

Example of specification

Example of specification

```
a       { Action1 }
abb     { Action2 }
a*b+    { Action3 }
```

Input: abb

Possible tokenizations

~~a|bb| Action1, Action3~~

abb| Action2

abb| Action3

Example of specification

Example of specification

```
a      { Action1 }
abb    { Action2 }
a*b+   { Action3 }
```

Input: abb

Possible tokenizations

~~a|bb| Action1, Action3~~

abb| Action2

~~abb| Action3~~

Scanner specification

In practice, we distinguish three types of tokens:

- literals
- reserved keywords
- general tokens

Scanner specification in SLY

Literals:

- Their lexems are one-character
- Token can be represented by its one-character lexem

```
literals = { '+', '-', '*', '/' }
literals = "+-*/"
```

Scanner specification in SLY

General tokens

- One token matches many lexems
- Specified with regular expressions

Examples:

NUM - matches many numbers

```
NUM = r"\d+"
```

```
@_(r'\d+')
def NUM(self, t):
    return t
```

Scanner specification in SLY

Reserved keywords:

- One token corresponds to exactly one lexem
- Their lexems are longer than one character
- Reserved keywords match also specification of an identifier, so additional work is needed to distinguish them

```
tokens = { "ID", "EQ", "NEQ", "LE", "GE",
    "BREAK", "CONTINUE", "IF", "ELSE" }

ID = r'[a-zA-Z_][a-zA-Z0-9_]*'

ID['break'] = 'BREAK'
ID['continue'] = 'CONTINUE'
ID['if'] = 'IF'
ID['else'] = 'ELSE'
```

Scanner specification in PLY

Reserved keywords:

```
reserved = {    'break':          'BREAK',
                'continue'     : 'CONTINUE',
                'if'           : 'IF',
                'else'         : 'ELSE',
           }
tokens = [ "ID", "EQ", "NEQ", "LE", "GE" ] + list(
    reserved.values())

def t_ID(t):
    r"[a-zA-Z_]\w*"
    t.type = reserved.get(t.value, 'ID')
    return t
```

Scanner specification

Pattern to be avoided - individual rules for reserved keywords:

```
BREAK = r'break'
CONTINUE = r'continue'
IF = r'if'
ELSE = r'else'
```

References

1. `https://sly.readthedocs.io/en/latest/sly.html`, Sect. Writing a Lexer
2. `https://github.com/dabeaz/sly`
3. http://www.dabeaz.com/ply/ply.html, Sect. 4, Lex