Semantic Web — Spring 2009
Homework #1

Simple XML Parser using Python Lex-Yacc

Behnam Esfahbod
behnam@sharif.edu

April 16, 2009

1 Introduction

This project consists of an XML parser, *xml-ply*, written in Python using *python Lex-Yacc* library, a simple document object model (DOM), and a tree-like view to the document.

Python Lex-Yacc (PLY) [1] is a pure-Python implementation of the popular compiler construction tools *lex* and *yacc*, with the goal to stay fairly faithful to the way in which traditional lex/yacc tools work. PLY consists of two separate modules; `lex.py` and `yacc.py`, both of which are found in a Python package called `ply`.

2 Lexer

The `lex.py` module is used to break input text into a collection of tokens specified by a collection of regular expression rules. Each token is specified by writing a regular expression rule. Each of these rules are are defined by making function declarations with a special prefix `t_` to indicate that it defines a token. For simple tokens, the regular expression can be specified as strings. (Python raw strings are used since they are the most convenient way to write regular expression strings)

The lexer has four *states*. The default state, `INITIAL`, is used for all the text outside the XML tags. The `tag` state is used for the tag opening and closing, the tag name, and the attributes names and assignments. The `attrvalue1` and `attrvalue2` states are used for single-quoted and double-quoted strings of attribute values respectively.

For now, we assume that the XML file doesn't contain XML special tags, DTD tags, nor comments.

3 Parser

The `yacc.py` module is used to recognize language syntax that has been specified in the form of a context free grammar. `yacc.py` uses LR parsing and generates its parsing tables using either the *LALR(1)* or *SLR* table generation algorithms. Here we use SLR table generation algorithm.

Each grammar rule is defined by a Python function where the doc-string to that function contains the appropriate context-free grammar specification. Each function accepts a single argument p that is a sequence containing the values of each grammar symbol in the corresponding rule.

The non-terminal tokens in the grammar are:

root The root element of the XML document

element The document nodes

opentag An opening tag of a node

closetag A closing tag of a node

lonetag A tag which doesn't have any child and ends right in the opening tag

attributes The sequence of attributes of a tag

attribute The pair of attribute name and value

attrvalue Value of an attribute

children The sequence of children of the tag

child A CDATA or Element node

4 Document Object Model

The document object model (DOM) has two models, `Element` and `Cdata`.

The `Element` model represents a node and contains the node name, the dictionary of attribute name-values, and the list of child nodes.

The `Cdata` model represents the CDATA nodes of the document, which only contains the text.

5 Source Code

Listing 1: The parser (parser.py)

```
#!/usr/bin/env python

import sys
from UserString import UserString

from ply import lex, yacc


##################################
# LEXER

class XmlLexer:
    '''The XML lexer'''

    # states:
    #   default:    The default context, non-tag
    #   tag:        The document tag context
    #   attrvalue1: Single-quoted tag attribute value
    #   attrvalue2: Double-quoted tag attribute value

    states = (
        ('tag', 'exclusive'),
        ('attrvalue1', 'exclusive'),
        ('attrvalue2', 'exclusive'),
    )

    tokens = [

        # state: INITIAL
        'CDATA',
        'OPENTAGOPEN',
        'CLOSETAGOPEN',

        # state: tag
        'TAGATTRNAME',
```

```
        'TAGCLOSE',
        'LONETAGCLOSE',
        'ATTRASSIGN',

        # state: attrvalue1
        'ATTRVALUE1OPEN',
        'ATTRVALUE1STRING',
        'ATTRVALUE1CLOSE',

        # state: attrvalue2
        'ATTRVALUE2OPEN',
        'ATTRVALUE2STRING',
        'ATTRVALUE2CLOSE',

    ]


    # Complex patterns
    re_digit       = r'([0-9])'
    re_nondigit    = r'([_A-Za-z])'
    re_identifier  = r'(' + re_nondigit + r'(' + re_digit + r'|' + re_nondigit + r')*)'


    # ANY

    def t_ANY_error(self, t):
        raise SyntaxError("Illegal character '%s'" % t.value[0])
        t.lexer.skip(1)
        pass


    # INITIAL

    t_ignore  = ''

    def t_CLOSETAGOPEN(self, t):
        r'</'
        t.lexer.push_state('tag')
        return t

    def t_OPENTAGOPEN(self, t):
        r'<'
        t.lexer.push_state('tag')
        return t

    def t_CDATA(self, t):
        '[^<]+'
        return t


    # tag: name

    t_tag_ignore  = ' \t'

    def t_tag_TAGATTRNAME(self, t):
        return t
    t_tag_TAGATTRNAME.__doc__ = re_identifier

    def t_tag_TAGCLOSE(self, t):
        r'>'
        t.lexer.pop_state()
        return t

    def t_tag_LONETAGCLOSE(self, t):
        r'/>'
        t.lexer.pop_state()
        return t
```

```
    # tag: attr

    t_tag_ATTRASSIGN     = r'='

    def t_tag_ATTRVALUE1OPEN(self, t):
        r'\''
        t.lexer.push_state('attrvalue1')
        return t

    def t_tag_ATTRVALUE2OPEN(self, t):
        r'"'
        t.lexer.push_state('attrvalue2')
        return t


    # attrvalue1

    def t_attrvalue1_ATTRVALUE1STRING(self, t):
        r'[^\']+'
        t.value = unicode(t.value)
        return t

    def t_attrvalue1_ATTRVALUE1CLOSE(self, t):
        r'\''
        t.lexer.pop_state()
        return t

    t_attrvalue1_ignore  = ''


    # attrvalue2

    def t_attrvalue2_ATTRVALUE2STRING(self, t):
        r'[^"]+'
        t.value = unicode(t.value)
        return t

    def t_attrvalue2_ATTRVALUE2CLOSE(self, t):
        r'"'
        t.lexer.pop_state()
        return t

    t_attrvalue2_ignore  = ''


    # misc

    literals = '$%^'

    def t_ANY_newline(self, t):
        r'\n'
        t.lexer.lineno += len(t.value)


    # Build the lexer
    def build(self, **kwargs):
        self.lexer = lex.lex(object=self, **kwargs)

    # Test it output
    def test(self, data):
        self.lexer.input(data)

        while 1:
            tok = self.lexer.token()
            if not tok: break
            _debug_print_('LEXER', '[%-12s] %s' % (self.lexer.lexstate, tok))
```

```
# Customization
class SyntaxError(Exception):
    pass


################################
# PARSER

tag_stack = []

# Grammer

def p_root_element(p):
    '''root : element
            | element CDATA
    '''
    _parser_trace(p)

    p[0] = p[1]

def p_root_cdata_element(p):
    '''root : CDATA element
            | CDATA element CDATA
    '''
    _parser_trace(p)

    p[0] = p[2]

def p_element(p):
    '''element : opentag children closetag
               | lonetag
    '''
    _parser_trace(p)

    if len(p) == 4:
        p[1].children = p[2]

    p[0] = p[1]

# tag
def p_opentag(p):
    '''opentag : OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE
    '''
    _parser_trace(p)

    tag_stack.append(p[2])
    p[0] = DOM.Element(p[2], p[3])

def p_closetag(p):
    '''closetag : CLOSETAGOPEN TAGATTRNAME TAGCLOSE
    '''
    _parser_trace(p)

    n = tag_stack.pop()
    if p[2] != n:
        raise ParserError('Close tag name ("%s") does not match the corresponding
            open tag ("%s").' % (p[2], n))

def p_lonetag(p):
    '''lonetag : OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE
    '''
    _parser_trace(p)

    p[0] = DOM.Element(p[2], p[3])

# attr
```

```
def p_attributes(p):
    '''attributes : attribute attributes
                  | empty
    '''
    _parser_trace(p)

    if len(p) == 3:
        if p[2]:
            p[1].update(p[2])
            p[0] = p[1]
        else:
            p[0] = p[1]
    else:
        p[0] = {}

def p_attribute(p):
    '''attribute : TAGATTRNAME ATTRASSIGN attrvalue
    '''
    _parser_trace(p)

    p[0] = {p[1]: p[3]}

def p_attrvalue(p):
    '''attrvalue : ATTRVALUE1OPEN ATTRVALUE1STRING ATTRVALUE1CLOSE
                 | ATTRVALUE2OPEN ATTRVALUE2STRING ATTRVALUE2CLOSE
    '''
    _parser_trace(p)

    p[0] = _xml_unescape(p[2])

# child
def p_children(p):
    '''children : child children
                | empty
    '''
    _parser_trace(p)

    if len(p) > 2:
        if p[2]:
            p[0] = [p[1]] + p[2]
        else:
            p[0] = [p[1]]
    else:
        p[0] = []

def p_child_element(p):
    '''child : element'''
    _parser_trace(p)

    p[0] = p[1]

def p_child_cdata(p):
    '''child : CDATA'''
    _parser_trace(p)

    p[0] = DOM.Cdata(p[1])

# empty
def p_empty(p):
    '''empty :'''
    pass

# Error rule for syntax errors
class ParserError(Exception):
    pass

def p_error(p):
    raise ParserError("Parse error: %s" % (p,))
```

```
    pass

# Customization
def _parser_trace(x):
    _debug_print_('PARSER', '[%-16s] %s' % (sys._getframe(1).f_code.co_name, x))

def _yacc_production__str(p):
    #return "YaccProduction(%s, %s)" % (str(p.slice), str(p.stack))
    return "YaccP%s" % (str([i.value for i in p.slice]))
yacc.YaccProduction.__str__ = _yacc_production__str


##################################
# DOM

class DOM:
    class Element:
        # Document object model
        #
        # Parser returns the root element of the XML document

        def __init__(self, name, attributes={}, children=[]):
            self.name = name
            self.attributes = attributes
            self.children = children

        def __str__(self):
            attributes_str = ''
            for attr in self.attributes:
                attributes_str += ' %s="%s"' % (attr, _xml_escape(self.attributes[
                    attr]))

            children_str = ''
            for child in self.children:
                if isinstance(child, self.__class__):
                    children_str += str(child)
                else:
                    children_str += child

            return '<%s%s>%s</%s>'% (self.name, attributes_str, children_str, self.
                name)

        def __repr__(self):
            return str(self)

    class Cdata(UserString):
        pass


##################################
# ESCAPE

_xml_escape_table = {
    "&": "&amp;",
    '"': "&quot;",
    "'": "&apos;",
    ">": "&gt;",
    "<": "&lt;",
    }

def _xml_escape(text):
    L=[]
    for c in text:
        L.append(_xml_escape_table.get(c,c))
    return "".join(L)

def _xml_unescape(s):
    rules = _xml_escape_table.items()
```

```
    rules.reverse()

    for x, y in rules:
        s = s.replace(y, x)

    return s


##################################
# INTERFACE

def xml_parse(data):

    _debug_header('INPUT')
    _debug_print_('INPUT', data)
    _debug_footer('INPUT')

    # Tokenizer
    xml_lexer = XmlLexer()
    xml_lexer.build()

    _debug_header('LEXER')
    xml_lexer.test(data)
    _debug_footer('LEXER')

    # Parser
    global tokens
    tokens = XmlLexer.tokens

    yacc.yacc(method="SLR")

    _debug_header('PARSER')
    root = yacc.parse(data, lexer=xml_lexer.lexer, debug=False)
    _debug_footer('PARSER')

    _debug_header('OUTPUT')
    _debug_print_('OUTPUT', root)
    _debug_footer('OUTPUT')

    return root


def tree(node, level=0, init_prefix=''):
    'Returns a tree view of the XML data'

    prefix = '    '
    attr_prefix = '@'
    tag_postfix = ':\t'
    attr_postfix = ':\t'

    s_node = init_prefix + node.name + tag_postfix
    s_attributes = ''
    s_children = ''

    for attr in node.attributes:
        s_attributes += init_prefix + prefix + attr_prefix + attr + attr_postfix +
            node.attributes[attr] + '\n'

    if len(node.children) == 1 and not isinstance(node.children[0], DOM.Element):
        s_node += node.children[0] + '\n'

    else:
        for child in node.children:
            if isinstance(child, DOM.Element):
                s_children += tree(child, level+1, init_prefix + prefix)

        s_node += '\n'
```

```
    return s_node + s_attributes + s_children

################################
# DEBUG

_DEBUG = {
    'INPUT': False,
    'LEXER': False,
    'PARSER': False,
    'OUTPUT': False,
}

def _debug_header(part):
    if _DEBUG[part]:
        print '--------'
        print '%s:' % part

def _debug_footer(part):
    if _DEBUG[part]:
        pass

def _debug_print_(part, s):
    if _DEBUG[part]:
        print s


################################
# MAIN

def main():
    data = open(sys.argv[1]).read()
    root = xml_parse(data)
    print tree(root)

if __name__ == '__main__':
    main()
```

6 Generated Grammar

Listing 2: The generated grammar (parser.out)

```
Created by PLY version 3.2 (http://www.dabeaz.com/ply)

Grammar

Rule 0     S' -> root
Rule 1     root -> element
Rule 2     root -> element CDATA
Rule 3     root -> CDATA element
Rule 4     root -> CDATA element CDATA
Rule 5     element -> opentag children closetag
Rule 6     element -> lonetag
Rule 7     opentag -> OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE
Rule 8     closetag -> CLOSETAGOPEN TAGATTRNAME TAGCLOSE
Rule 9     lonetag -> OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE
Rule 10    attributes -> attribute attributes
Rule 11    attributes -> empty
Rule 12    attribute -> TAGATTRNAME ATTRASSIGN attrvalue
Rule 13    attrvalue -> ATTRVALUE1OPEN ATTRVALUE1STRING ATTRVALUE1CLOSE
Rule 14    attrvalue -> ATTRVALUE2OPEN ATTRVALUE2STRING ATTRVALUE2CLOSE
Rule 15    children -> child children
Rule 16    children -> empty
Rule 17    child -> element
Rule 18    child -> CDATA
```

```
Rule 19     empty -> <empty>

Terminals, with rules where they appear

ATTRASSIGN           : 12
ATTRVALUE1CLOSE      : 13
ATTRVALUE1OPEN       : 13
ATTRVALUE1STRING     : 13
ATTRVALUE2CLOSE      : 14
ATTRVALUE2OPEN       : 14
ATTRVALUE2STRING     : 14
CDATA                : 2 3 4 4 18
CLOSETAGOPEN         : 8
LONETAGCLOSE         : 9
OPENTAGOPEN          : 7 9
TAGATTRNAME          : 7 8 9 12
TAGCLOSE             : 7 8
error                :

Nonterminals, with rules where they appear

attribute            : 10
attributes           : 7 9 10
attrvalue            : 12
child                : 15
children             : 5 15
closetag             : 5
element              : 1 2 3 4 17
empty                : 11 16
lonetag              : 6
opentag              : 5
root                 : 0

Parsing method: SLR

state 0

    (0) S' -> . root
    (1) root -> . element
    (2) root -> . element CDATA
    (3) root -> . CDATA element
    (4) root -> . CDATA element CDATA
    (5) element -> . opentag children closetag
    (6) element -> . lonetag
    (7) opentag -> . OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE
    (9) lonetag -> . OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE

    CDATA           shift and go to state 3
    OPENTAGOPEN     shift and go to state 1

    lonetag                        shift and go to state 4
    element                        shift and go to state 2
    root                           shift and go to state 5
    opentag                        shift and go to state 6

state 1

    (7) opentag -> OPENTAGOPEN . TAGATTRNAME attributes TAGCLOSE
    (9) lonetag -> OPENTAGOPEN . TAGATTRNAME attributes LONETAGCLOSE

    TAGATTRNAME     shift and go to state 7


state 2

    (1) root -> element .
    (2) root -> element . CDATA
```

```
    $end            reduce using rule 1 (root -> element .)
    CDATA           shift and go to state 8


state 3

    (3) root -> CDATA . element
    (4) root -> CDATA . element CDATA
    (5) element -> . opentag children closetag
    (6) element -> . lonetag
    (7) opentag -> . OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE
    (9) lonetag -> . OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE

    OPENTAGOPEN     shift and go to state 1

    lonetag                         shift and go to state 4
    element                         shift and go to state 9
    opentag                         shift and go to state 6

state 4

    (6) element -> lonetag .

    $end            reduce using rule 6 (element -> lonetag .)
    CDATA           reduce using rule 6 (element -> lonetag .)
    OPENTAGOPEN     reduce using rule 6 (element -> lonetag .)
    CLOSETAGOPEN    reduce using rule 6 (element -> lonetag .)


state 5

    (0) S' -> root .



state 6

    (5) element -> opentag . children closetag
    (15) children -> . child children
    (16) children -> . empty
    (17) child -> . element
    (18) child -> . CDATA
    (19) empty -> .
    (5) element -> . opentag children closetag
    (6) element -> . lonetag
    (7) opentag -> . OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE
    (9) lonetag -> . OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE

    CDATA           shift and go to state 11
    TAGCLOSE        reduce using rule 19 (empty -> .)
    LONETAGCLOSE    reduce using rule 19 (empty -> .)
    CLOSETAGOPEN    reduce using rule 19 (empty -> .)
    OPENTAGOPEN     shift and go to state 1

    element                         shift and go to state 10
    child                           shift and go to state 12
    lonetag                         shift and go to state 4
    children                        shift and go to state 13
    empty                           shift and go to state 14
    opentag                         shift and go to state 6

state 7

    (7) opentag -> OPENTAGOPEN TAGATTRNAME . attributes TAGCLOSE
    (9) lonetag -> OPENTAGOPEN TAGATTRNAME . attributes LONETAGCLOSE
    (10) attributes -> . attribute attributes
    (11) attributes -> . empty
    (12) attribute -> . TAGATTRNAME ATTRASSIGN attrvalue
```

```
    (19) empty -> .

    TAGATTRNAME     shift and go to state 15
    TAGCLOSE        reduce using rule 19 (empty -> .)
    LONETAGCLOSE    reduce using rule 19 (empty -> .)
    CLOSETAGOPEN    reduce using rule 19 (empty -> .)

    attributes                  shift and go to state 17
    empty                       shift and go to state 18
    attribute                   shift and go to state 16

state 8

    (2) root -> element CDATA .

    $end            reduce using rule 2 (root -> element CDATA .)


state 9

    (3) root -> CDATA element .
    (4) root -> CDATA element . CDATA

    $end            reduce using rule 3 (root -> CDATA element .)
    CDATA           shift and go to state 19


state 10

    (17) child -> element .

    CDATA           reduce using rule 17 (child -> element .)
    OPENTAGOPEN     reduce using rule 17 (child -> element .)
    CLOSETAGOPEN    reduce using rule 17 (child -> element .)


state 11

    (18) child -> CDATA .

    CDATA           reduce using rule 18 (child -> CDATA .)
    OPENTAGOPEN     reduce using rule 18 (child -> CDATA .)
    CLOSETAGOPEN    reduce using rule 18 (child -> CDATA .)


state 12

    (15) children -> child . children
    (15) children -> . child children
    (16) children -> . empty
    (17) child -> . element
    (18) child -> . CDATA
    (19) empty -> .
    (5) element -> . opentag children closetag
    (6) element -> . lonetag
    (7) opentag -> . OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE
    (9) lonetag -> . OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE

    CDATA           shift and go to state 11
    TAGCLOSE        reduce using rule 19 (empty -> .)
    LONETAGCLOSE    reduce using rule 19 (empty -> .)
    CLOSETAGOPEN    reduce using rule 19 (empty -> .)
    OPENTAGOPEN     shift and go to state 1

    element                     shift and go to state 10
    child                       shift and go to state 12
    lonetag                     shift and go to state 4
    children                    shift and go to state 20
```

```
    empty                          shift and go to state 14
    opentag                        shift and go to state 6

state 13

    (5) element -> opentag children . closetag
    (8) closetag -> . CLOSETAGOPEN TAGATTRNAME TAGCLOSE

    CLOSETAGOPEN    shift and go to state 21

    closetag                       shift and go to state 22

state 14

    (16) children -> empty .

    CLOSETAGOPEN    reduce using rule 16 (children -> empty .)


state 15

    (12) attribute -> TAGATTRNAME . ATTRASSIGN attrvalue

    ATTRASSIGN      shift and go to state 23


state 16

    (10) attributes -> attribute . attributes
    (10) attributes -> . attribute attributes
    (11) attributes -> . empty
    (12) attribute -> . TAGATTRNAME ATTRASSIGN attrvalue
    (19) empty -> .

    TAGATTRNAME     shift and go to state 15
    TAGCLOSE        reduce using rule 19 (empty -> .)
    LONETAGCLOSE    reduce using rule 19 (empty -> .)
    CLOSETAGOPEN    reduce using rule 19 (empty -> .)

    attribute                      shift and go to state 16
    empty                          shift and go to state 18
    attributes                     shift and go to state 24

state 17

    (7) opentag -> OPENTAGOPEN TAGATTRNAME attributes . TAGCLOSE
    (9) lonetag -> OPENTAGOPEN TAGATTRNAME attributes . LONETAGCLOSE

    TAGCLOSE        shift and go to state 26
    LONETAGCLOSE    shift and go to state 25


state 18

    (11) attributes -> empty .

    TAGCLOSE        reduce using rule 11 (attributes -> empty .)
    LONETAGCLOSE    reduce using rule 11 (attributes -> empty .)


state 19

    (4) root -> CDATA element CDATA .

    $end            reduce using rule 4 (root -> CDATA element CDATA .)


state 20
```

```
    (15) children -> child children .

    CLOSETAGOPEN    reduce using rule 15 (children -> child children .)


state 21

    (8) closetag -> CLOSETAGOPEN . TAGATTRNAME TAGCLOSE

    TAGATTRNAME     shift and go to state 27


state 22

    (5) element -> opentag children closetag .

    $end            reduce using rule 5 (element -> opentag children closetag .)
    CDATA           reduce using rule 5 (element -> opentag children closetag .)
    OPENTAGOPEN     reduce using rule 5 (element -> opentag children closetag .)
    CLOSETAGOPEN    reduce using rule 5 (element -> opentag children closetag .)


state 23

    (12) attribute -> TAGATTRNAME ATTRASSIGN . attrvalue
    (13) attrvalue -> . ATTRVALUE1OPEN ATTRVALUE1STRING ATTRVALUE1CLOSE
    (14) attrvalue -> . ATTRVALUE2OPEN ATTRVALUE2STRING ATTRVALUE2CLOSE

    ATTRVALUE1OPEN  shift and go to state 29
    ATTRVALUE2OPEN  shift and go to state 30

    attrvalue                     shift and go to state 28

state 24

    (10) attributes -> attribute attributes .

    TAGCLOSE        reduce using rule 10 (attributes -> attribute attributes .)
    LONETAGCLOSE    reduce using rule 10 (attributes -> attribute attributes .)


state 25

    (9) lonetag -> OPENTAGOPEN TAGATTRNAME attributes LONETAGCLOSE .

    $end            reduce using rule 9 (lonetag -> OPENTAGOPEN TAGATTRNAME
        attributes LONETAGCLOSE .)
    CDATA           reduce using rule 9 (lonetag -> OPENTAGOPEN TAGATTRNAME
        attributes LONETAGCLOSE .)
    OPENTAGOPEN     reduce using rule 9 (lonetag -> OPENTAGOPEN TAGATTRNAME
        attributes LONETAGCLOSE .)
    CLOSETAGOPEN    reduce using rule 9 (lonetag -> OPENTAGOPEN TAGATTRNAME
        attributes LONETAGCLOSE .)


state 26

    (7) opentag -> OPENTAGOPEN TAGATTRNAME attributes TAGCLOSE .

    CDATA           reduce using rule 7 (opentag -> OPENTAGOPEN TAGATTRNAME
        attributes TAGCLOSE .)
    OPENTAGOPEN     reduce using rule 7 (opentag -> OPENTAGOPEN TAGATTRNAME
        attributes TAGCLOSE .)
    CLOSETAGOPEN    reduce using rule 7 (opentag -> OPENTAGOPEN TAGATTRNAME
        attributes TAGCLOSE .)
```

```
state 27

    (8) closetag -> CLOSETAGOPEN TAGATTRNAME . TAGCLOSE

    TAGCLOSE        shift and go to state 31


state 28

    (12) attribute -> TAGATTRNAME ATTRASSIGN attrvalue .

    TAGATTRNAME     reduce using rule 12 (attribute -> TAGATTRNAME ATTRASSIGN
        attrvalue .)
    TAGCLOSE        reduce using rule 12 (attribute -> TAGATTRNAME ATTRASSIGN
        attrvalue .)
    LONETAGCLOSE    reduce using rule 12 (attribute -> TAGATTRNAME ATTRASSIGN
        attrvalue .)


state 29

    (13) attrvalue -> ATTRVALUE1OPEN . ATTRVALUE1STRING ATTRVALUE1CLOSE

    ATTRVALUE1STRING shift and go to state 32


state 30

    (14) attrvalue -> ATTRVALUE2OPEN . ATTRVALUE2STRING ATTRVALUE2CLOSE

    ATTRVALUE2STRING shift and go to state 33


state 31

    (8) closetag -> CLOSETAGOPEN TAGATTRNAME TAGCLOSE .

    $end            reduce using rule 8 (closetag -> CLOSETAGOPEN TAGATTRNAME
        TAGCLOSE .)
    CDATA           reduce using rule 8 (closetag -> CLOSETAGOPEN TAGATTRNAME
        TAGCLOSE .)
    OPENTAGOPEN     reduce using rule 8 (closetag -> CLOSETAGOPEN TAGATTRNAME
        TAGCLOSE .)
    CLOSETAGOPEN    reduce using rule 8 (closetag -> CLOSETAGOPEN TAGATTRNAME
        TAGCLOSE .)


state 32

    (13) attrvalue -> ATTRVALUE1OPEN ATTRVALUE1STRING . ATTRVALUE1CLOSE

    ATTRVALUE1CLOSE shift and go to state 34


state 33

    (14) attrvalue -> ATTRVALUE2OPEN ATTRVALUE2STRING . ATTRVALUE2CLOSE

    ATTRVALUE2CLOSE shift and go to state 35


state 34

    (13) attrvalue -> ATTRVALUE1OPEN ATTRVALUE1STRING ATTRVALUE1CLOSE .

    TAGATTRNAME     reduce using rule 13 (attrvalue -> ATTRVALUE1OPEN
        ATTRVALUE1STRING ATTRVALUE1CLOSE .)
```

```
    TAGCLOSE        reduce using rule 13 (attrvalue -> ATTRVALUE1OPEN
        ATTRVALUE1STRING ATTRVALUE1CLOSE .)
    LONETAGCLOSE    reduce using rule 13 (attrvalue -> ATTRVALUE1OPEN
        ATTRVALUE1STRING ATTRVALUE1CLOSE .)

state 35

    (14) attrvalue -> ATTRVALUE2OPEN ATTRVALUE2STRING ATTRVALUE2CLOSE .

    TAGATTRNAME     reduce using rule 14 (attrvalue -> ATTRVALUE2OPEN
        ATTRVALUE2STRING ATTRVALUE2CLOSE .)
    TAGCLOSE        reduce using rule 14 (attrvalue -> ATTRVALUE2OPEN
        ATTRVALUE2STRING ATTRVALUE2CLOSE .)
    LONETAGCLOSE    reduce using rule 14 (attrvalue -> ATTRVALUE2OPEN
        ATTRVALUE2STRING ATTRVALUE2CLOSE .)
```

7 Examples

7.1 Example 1

Listing 3: Sample input 1 (sample-1.xml)

```
<Products>
  <Product pid="p123">
    <Name>gizmo</Name>
    <Price>22.99</Price>
    <Description>great</Description>
    <Store>
      <Name>wiz</Name>
      <Phone>555-1234</Phone>
      <Markup>25</Markup>
    </Store>
    <Store>
      <Name>Econo-Wiz</Name>
      <Phone>555-6543</Phone>
      <Markup>15</Markup>
    </Store>
  </Product>
  <Product pid="p231">
    <Name>gizmoPlus</Name>
    <Price>99.99</Price>
    <Description>more features</Description>
    <Store>
      <Name>wiz</Name>
      <Phone>555-1234</Phone>
      <Markup>10</Markup>
    </Store>
  </Product>
  <Product pid="p312">
    <Name>gadget</Name>
    <Price>59.99</Price>
    <Description>good value</Description>
  </Product>
</Products>
```

Listing 4: Output 1 (sample-1.out)

```
Products:
    Product:
        @pid: p123
        Name: gizmo
        Price:  22.99
        Description:  great
```

```
        Store:
            Name: wiz
            Phone:  555-1234
            Markup: 25
        Store:
            Name: Econo-Wiz
            Phone:  555-6543
            Markup: 15
    Product:
        @pid: p231
        Name: gizmoPlus
        Price:  99.99
        Description:  more features
        Store:
            Name: wiz
            Phone:  555-1234
            Markup: 10
    Product:
        @pid: p312
        Name: gadget
        Price:  59.99
        Description:  good value
```

7.2 Example 2

Listing 5: Sample input 2 (sample-2.xml)

```
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="8" unit="dL">Flour</ingredient>
  <ingredient amount="10" unit="grams">Yeast</ingredient>
  <ingredient amount="4" unit="dL" state="warm">Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <instructions>
    <step>Mix all ingredients together.</step>
    <step>Knead thoroughly.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Knead again.</step>
    <step>Place in a bread baking tin.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Bake in the oven at 180(degrees)C for 30 minutes.</step>
  </instructions>
</recipe>
```

Listing 6: Output 2 (sample-2.out)

```
recipe:
    @cook_time: 3 hours
    @name:  bread
    @prep_time: 5 mins
    title:  Basic bread
    ingredient: Flour
        @amount:  8
        @unit:  dL
    ingredient: Yeast
        @amount:  10
        @unit:  grams
    ingredient: Water
        @amount:  4
        @state: warm
        @unit:  dL
    ingredient: Salt
        @amount:  1
        @unit:  teaspoon
```

```
    instructions:
        step: Mix all ingredients together.
        step: Knead thoroughly.
        step: Cover with a cloth, and leave for one hour in warm room.
        step: Knead again.
        step: Place in a bread baking tin.
        step: Cover with a cloth, and leave for one hour in warm room.
        step: Bake in the oven at 180(degrees)C for 30 minutes.
```

References

[1] PLY (Python Lex-Yacc) `http://www.dabeaz.com/ply/`