# Using PLY For Great Good

Dan Colish

July 13, 2010

# Whirlwind Tour of Grammars

## Definition: Context Free Grammar

- $V$ a finite set of non-terminals or variables
- $\sigma$ a set of terminals
- $R$ the relation of productions
- $S$ the grammars start style

# Whirlwind Tour of Grammars

## Definition: Context Free Grammar

- $V$ a finite set of non-terminals or variables
- $\sigma$ a set of terminals
- $R$ the relation of productions
- $S$ the grammars start style

## Example: $L_{ab} = \{a^n b^n : n \geq 0\}$

- $V = S$
- $\sigma = a, b$
- $R = \{S \rightarrow aSb\}$

# Using grammars in the real world

## A few things they do

- Design & implement languages
- Recognize patterns in input/output
- Construct interactive programs

# Using grammars in the real world

## A few things they do

- Design & implement languages
- Recognize patterns in input/output
- Construct interactive programs

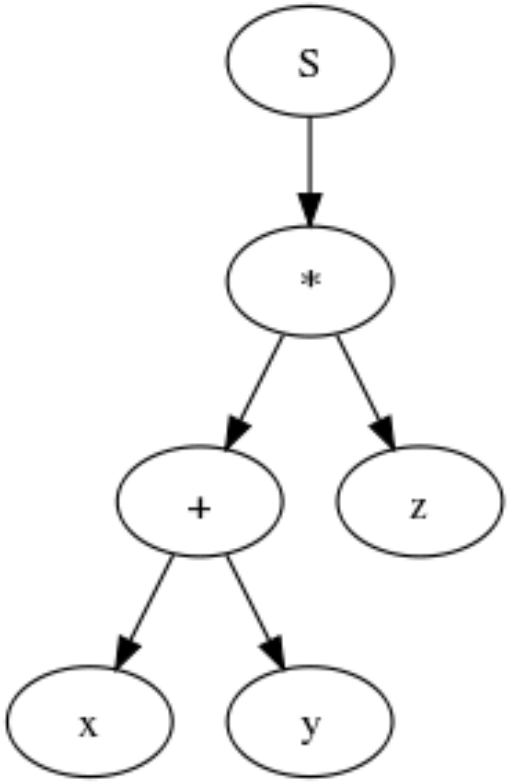
## BNF: the grammar you know

- Developed by John Backus and Peter Naur in the 1960's
- Looks like $<$ *symbol* $>$::= __*expression*__
- Also looks like
  *list_display* ::= "[" [*expression_list*|*list_comprehension*]"]"
  *list_comprehension* ::= *expressionlist_for*

`http://docs.python.org/reference/expressions.html`

# Some more grammar business

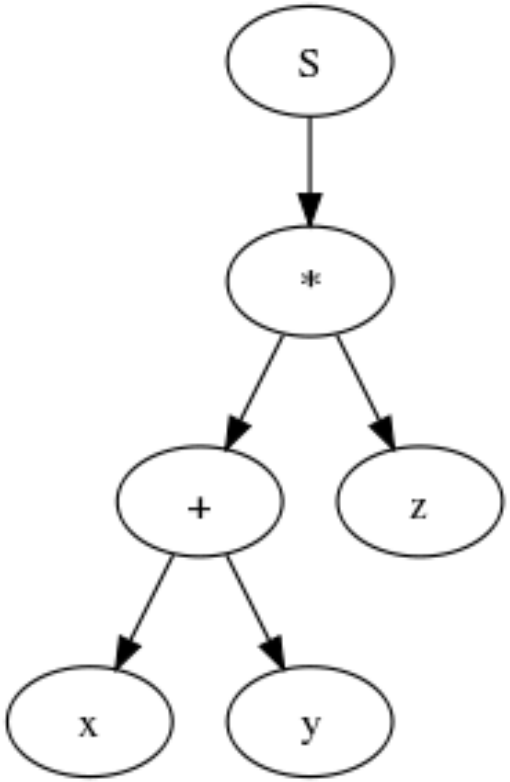
**T**he Parse Tree




`http://en.wikipedia.org/wiki/Context-free_grammar`

# Some more grammar business

**T**he Parse Tree




**T**he Matching Grammar

$$T \to x$$
$$T \to y$$
$$T \to z$$
$$S \to S + T$$
$$S \to S - T$$
$$S \to S * T$$
$$S \to S / T$$
$$T \to (S)$$
$$S \to T$$

`http://en.wikipedia.org/wiki/Context-free_grammar`

# Introducing the tools

## PLY

- For building Yacc Style Grammars
- Defining lexer tokens are the terminal set $\sigma$
- Defining production rules represent $R$
- Production rules are defined in docstrings
- Always resolves Shift/Reduce with Shift

# Introducing the tools

## PLY

- For building Yacc Style Grammars
- Defining lexer tokens are the terminal set $\sigma$
- Defining production rules represent $R$
- Production rules are defined in docstrings
- Always resolves Shift/Reduce with Shift

## Example: PLY Grammar

```
def p_expression_plus(p):
    '''expression : expression PLUS term'''
    p[0] = p[1] + p[3]

def p_expression_term(p):
    '''expression : term'''
    p[0] = p[1]
```

# What to avoid when building grammars

## Don't

- use complex lexer tokens
- write the grammar before the semantics
- work in large increments

# What to avoid when building grammars

## Don't

- use complex lexer tokens
- write the grammar before the semantics
- work in large increments

## Do

- write out the productions for your grammar on paper
- allow the parser to do the syntax not the lexer
- only do one thing in each production
- try optimizing only when your grammar is complete

# What I ended up doing

## The output... grrrr

```
1 subgoal

============================

forall A B : Prop, (A -> (~ ~ A) \/ B)
```

# What I ended up doing

## The output... grrrr

```
1 subgoal

============================

forall A B : Prop, (A -> (~ ~ A) \/ B)
```

## A few problems

- regex won't easily work
- this is still only a simple bit of output

# A bit of grammar the implementation

```
from ply import yacc
from lexer import tokens
__all__ = ['tokens', 'parser', 'precedence']

def p_proofst(p):
    '''proofst : subgoal hyp goal
               | subgoal goal'''
    if len(p) == 4:
        p[0] = dict(subgoal=p[1], hyp=p[2], goal=p[3])
    else:
        p[0] = dict(subgoal=p[1], goal=p[2])

def p_subgoal(p):
    '''subgoal : NUMBER SUBGOAL'''
    p[0] = '_'.join((str(p[1]), str(p[2])))

def p_hyp(p):
    '''hyp : ID COLON PROP hyp
           | ID COLON expr hyp
           | ID COLON PROP
           | ID COLON expr
           '''
    hyp = [dict(name=p[1], type=p[3])]
    if len(p) == 5:
        p[0] = hyp + [p[4]]
    else:
        p[0] = dict(name=p[1], type=p[3])

parser = yacc.yacc(debug=True)
```

# Seventh inning stretch: Demo Time

`http://imgur.com/gallery/f5kzo`

# Additional Resources

- http://en.wikipedia.org/wiki/Context-free_grammar
- http://dinosaur.compilertools.net/
- http://www.dabeaz.com/ply/
- http://www.mostly-decidable.org
- http://github.com/dcolish/Cockerel/tree/master/coqd/parser/