~~(± 3 5)~~

$(\underbrace{(\text{if } \#t \quad 3 \quad 5)})$

error

(if

let

quote

---

error

(+ 3 5) ←

Another type of parsing

Backup

The parsing that you did for Scheme

- stack push pop thing
- an example of an LR parser
  (bottom-up parser)

One downside of that algorithm that we used is that it is totally Scheme-specific

There are other approaches, both LL and LR, that work "automatically" for lots of grammars.

We're going to look at one of those:

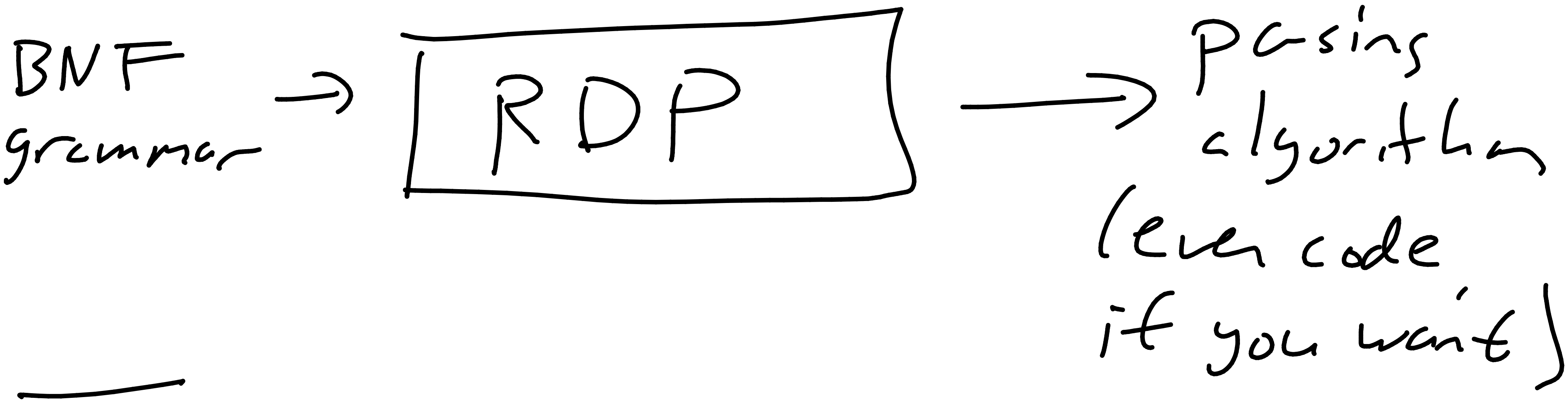
Recursive Descent Parsing

LL approach (top-down)

In CS 251, we are discussing 2 different parsers

1. Scheme-specific approach you used on part 4

2. RDP, which is all in-class

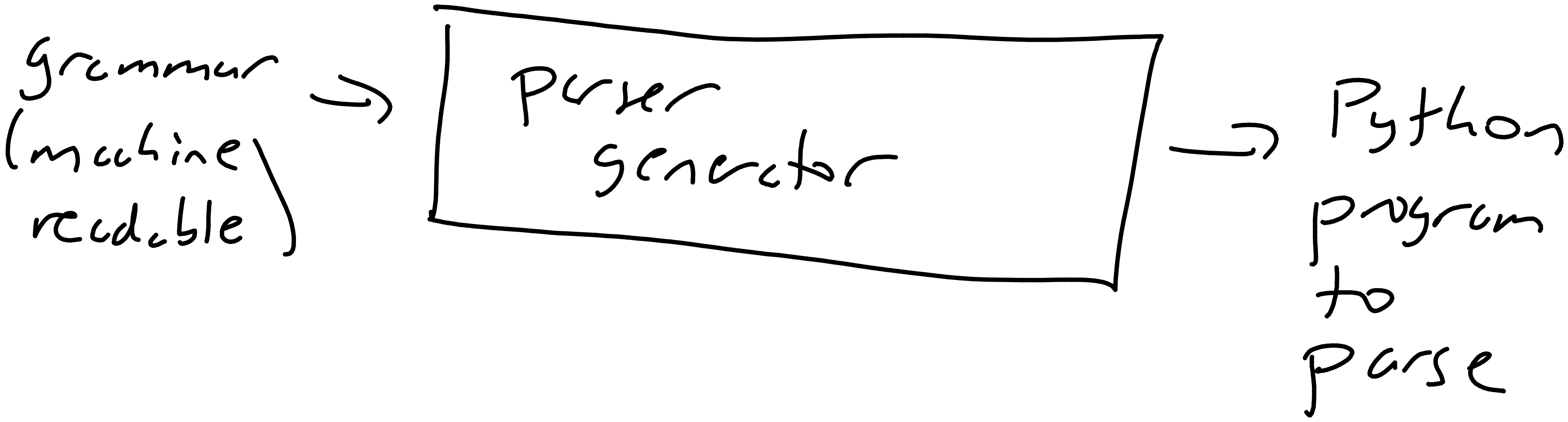


BNF grammar → RDP → parsing algorithm (even code if you want)

Example of another Scheme parser that I wrote in Python

$$
\begin{aligned}
*\ & \text{<P>} && \rightarrow \text{<E>} \\
*\ & \text{<E>} && \rightarrow \text{atom} \ \mid\ \text{'}\ \text{<E>} \ \mid\ (\ \text{<E>}\ \text{<Es>}\ ) \\
*\ & \text{<Es>} && \rightarrow \text{<E>}\ \text{<Es>} \ \mid\ \varepsilon
\end{aligned}
$$

The parsing program I wrote _is_ the grammar



```
def P():
    if token in ['atom', "'", '(']:
        E()
    else:
        raise Exception('Parse error')

def E():
    if token == 'atom':
        match('atom')
    elif token == "'":
        match("'")
        E()
    elif token == "(":
        match("(")
        E()
        Es()
        match(")")
    else:
        raise Exception('Parse error')

def Es():
    if token in ['atom', "'", '(']:
        E()
        Es()
    elif token == ")":
        pass
```

the only brainy part

```
def P():
    if token in ['atom' , "'" , '(']:
        E()
    else:
        raise Exception('Parse error')

def E():
    if token == 'atom':
        match('atom')
    elif token == "'":
        match("'")
        E()
    elif token == "(":
        match("(")
        E()
        Es()
        match(")")
    else:
        raise Exception('Parse error')

def Es():
    if token in ['atom' , "'" , '(']:
        E()
        Es()
    elif token == ")":
        pass

if __name__=='__main__':
    """Open up the file, grab the program,
    and parse it."""
```

<E> → atom

<E> → ' <E>

<E> → (~ ~)

<Es> →
<E><Es>

<Es> → ε