Kay Schlühr (ˈkaɪ ʃlyɐ)

kay@fiber-space.de

EasyExtend

R.I.P

Langscape

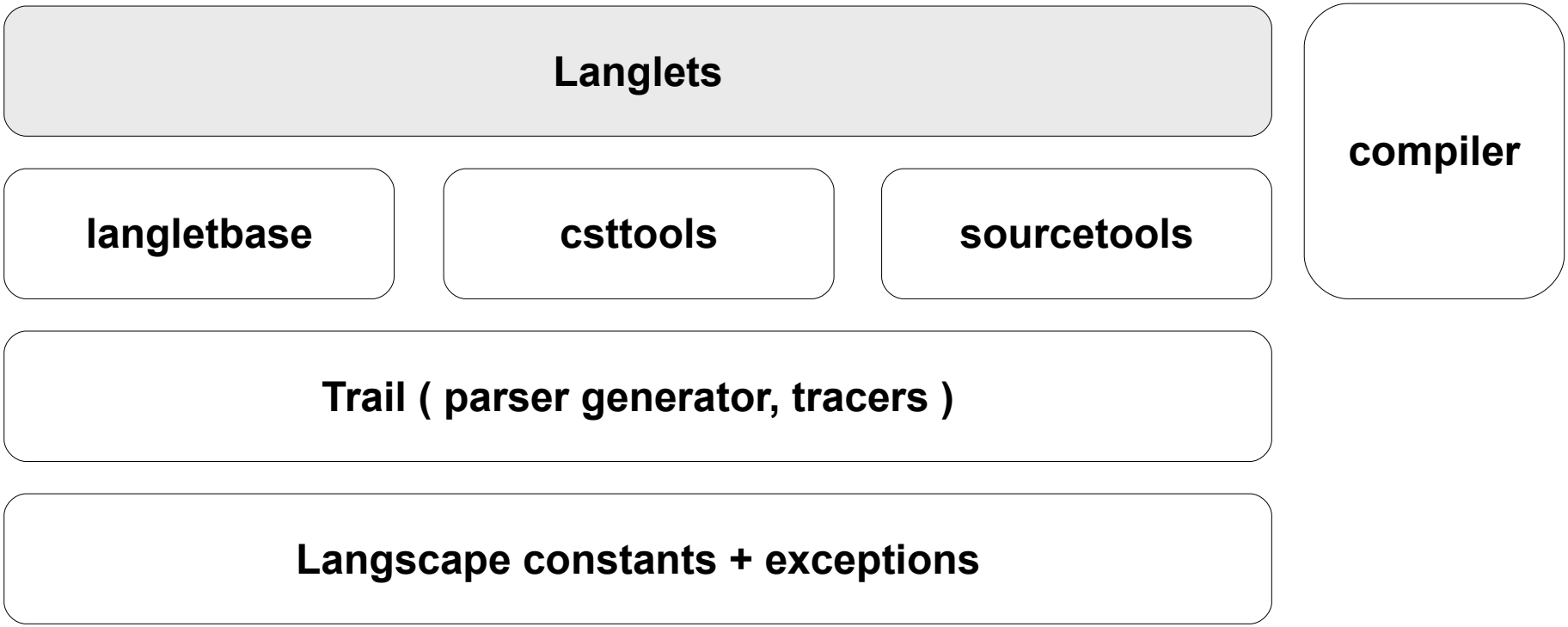
Langscape

Syntax oriented tools

- Syntax definition and parsing (analysis)
- Expression generation (synthesis)
- General language transformations $\mathcal{L}1$ -> $\mathcal{L}2$
- Syntax guided search and replace

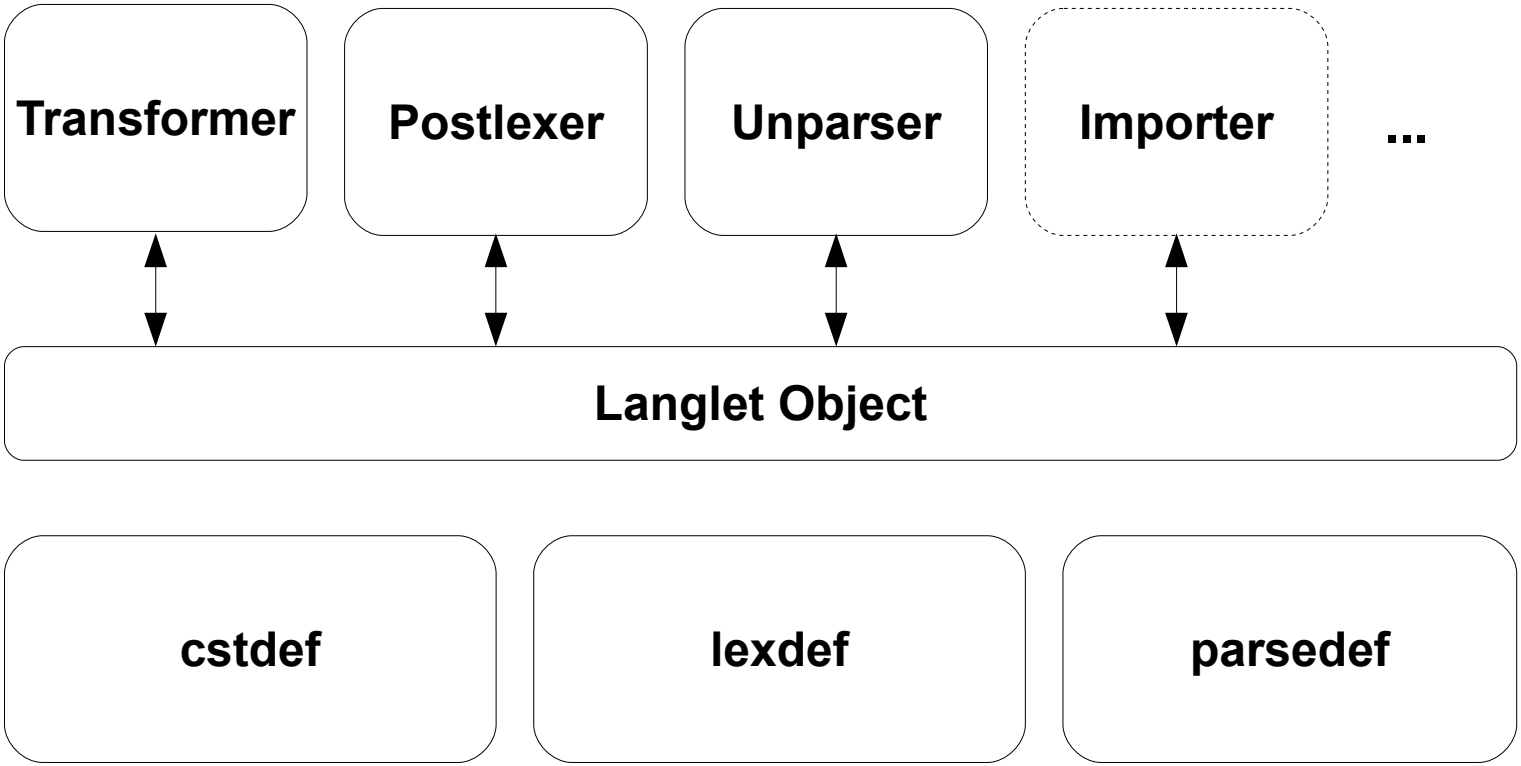
Langscape Architecture

A rough architectural sketch...



Langlet Architecture

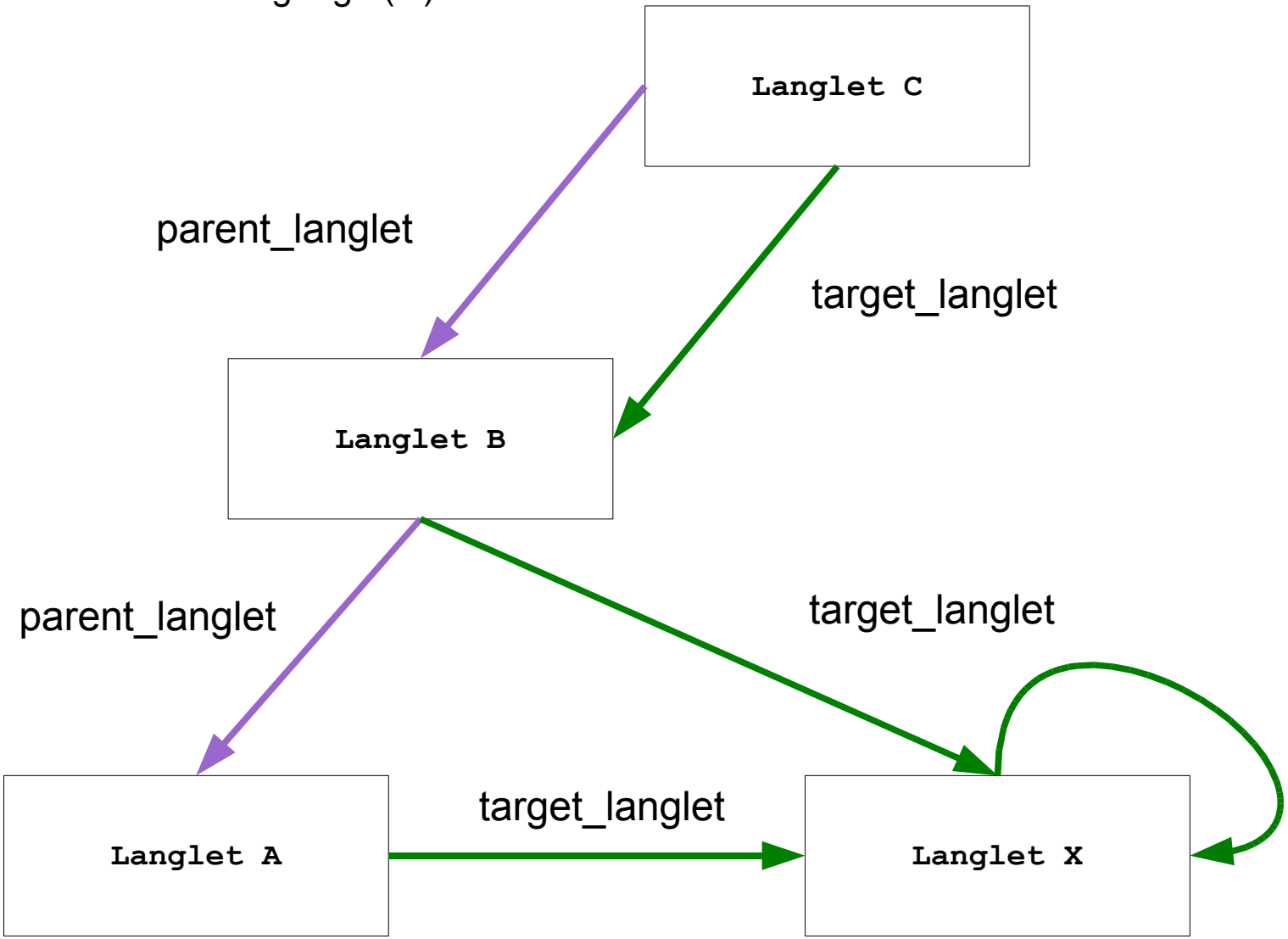
A rough architectural sketch...



Langlet Inheritance

target_langlet = transformation target (M)

parent_langlet = extended language (O)



Langlet methods

tokenization

```
>>> python = langscape.load_langlet(“python”)
>>> python.tokenize(“print 'a',2”)
<TokenStream: [[520, 'print', 1, (0, 5)], [3, "'a'", 1, (6, 9)],
               [12, ',', 1, (9, 10)], [2, '2', 1, (10, 11)],
               [0, '', 2, (0, 0)]] >
```

parsing

```
>>> cst = python.parse(“print 'a',2”)
>>> cst
[1001, [1010, [1011, [1012, [1015, [520, 'print', 1, (0, 5)], ...
[2, '2', 1, (10, 11)]]]]]]]]]]]]]]]], [4, '\n', 1, (11, 12)]]],
[0, '', 2, (0, 0)]]
```

unparsing

```
>>> python.unparse(cst)
"print 'a', 2\n"
```

More langlet methods

symbols & token

```
>>> python.symbol.test
1047
>>> python.token.COMMA
12
>>> python.parse_nfa.keywords[“print”]
520
```

node names

```
>>> python.get_node_name(520)
'print'
>>> python.get_node_name(12)
'COMMA'
```

pprint

```
>>> python.pprint(python.parse(“print 7”))

file_input  -- NT`0.1001
  stmt  -- NT`0.1010
    simple_stmt  -- NT`0.1011
      small_stmt  -- NT`0.1012
        print_stmt  -- NT`0.1015
            ...
```

Searching nodes

find_node, find_all, find_one_of, find_all_of, find_node_chain, find_all_chains, find_token_chain, find_chain_gen,...

find_node

```
>>> from langscape.csttools.cstsearch find_node, find_all
>>> cst = python.parse("print (str(a+b))")
>>> expr = find_node(cst, python.symbol.expr)
>>> python.unparse(expr)
'(str(a+b))'
```

find_all

```
>>> arith_expr = find_all(expr, python.symbol.arith_expr)
>>> for nd in arith_expr:
...     python.unparse(nd)
...
'(str(a+b))'
'str(a+b)'
'a+b'
```

Node construction

Construct `return_stmt: 'return' [testlist]`

```
>>> cst = python.fn.return_stmt("a")
>>> cst
[1021, [507, 'return'], [1070, [1047, [1048, [1049, [1050, [1051, [1053,
[1054, [1055, [1056, [1057, [1058, [1059, [1060, [1061,
[1, 'a', 1, (0, 1)]]]
```

Node construction

Construct `return_stmt: 'return' [testlist]`

```
>>> cst = python.fn.return_stmt(“a”)
>>> cst
[1021, [507, 'return'], [1070, [1047, [1048, [1049, [1050, [1051, [1053,
[1054, [1055, [1056, [1057, [1058, [1059, [1060, [1061,
[1, 'a', 1, (0, 1)]]]
```

CST interpolation

Building CST segments

```
>>> from langscape.csttools.cstsegments import SegmentTree
>>> sgtree = SegmentTree(python)
>>> sgtree.create()
>>> sgtree[python.symbol.test:python.symbol.atom]
[1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061]
```

or generalized CST segments

```
>>> sgtree[python.symbol.stmt:python.symbol.test]
[1010, 1011, [1012, 4], 1013, 1070, 1047]
>>> python.get_node_name(1012)
'small_stmt'
```

Node builders

Building `return_stmt: 'return' [testlist]`

```
python.fn.return_stmt("a")
```

```
symbol.return_stmt
```

Start-Symbol

```
[507, 'return']
```

Autoinsertion: required '**`return`**' keyword.

```
[symbol.testlist,[
 symbol.test,[
 ...
 symbol.atom,[
[token.NAME, 'a']...]
```

Token creation: `'a' -> [token.NAME, 'a']`

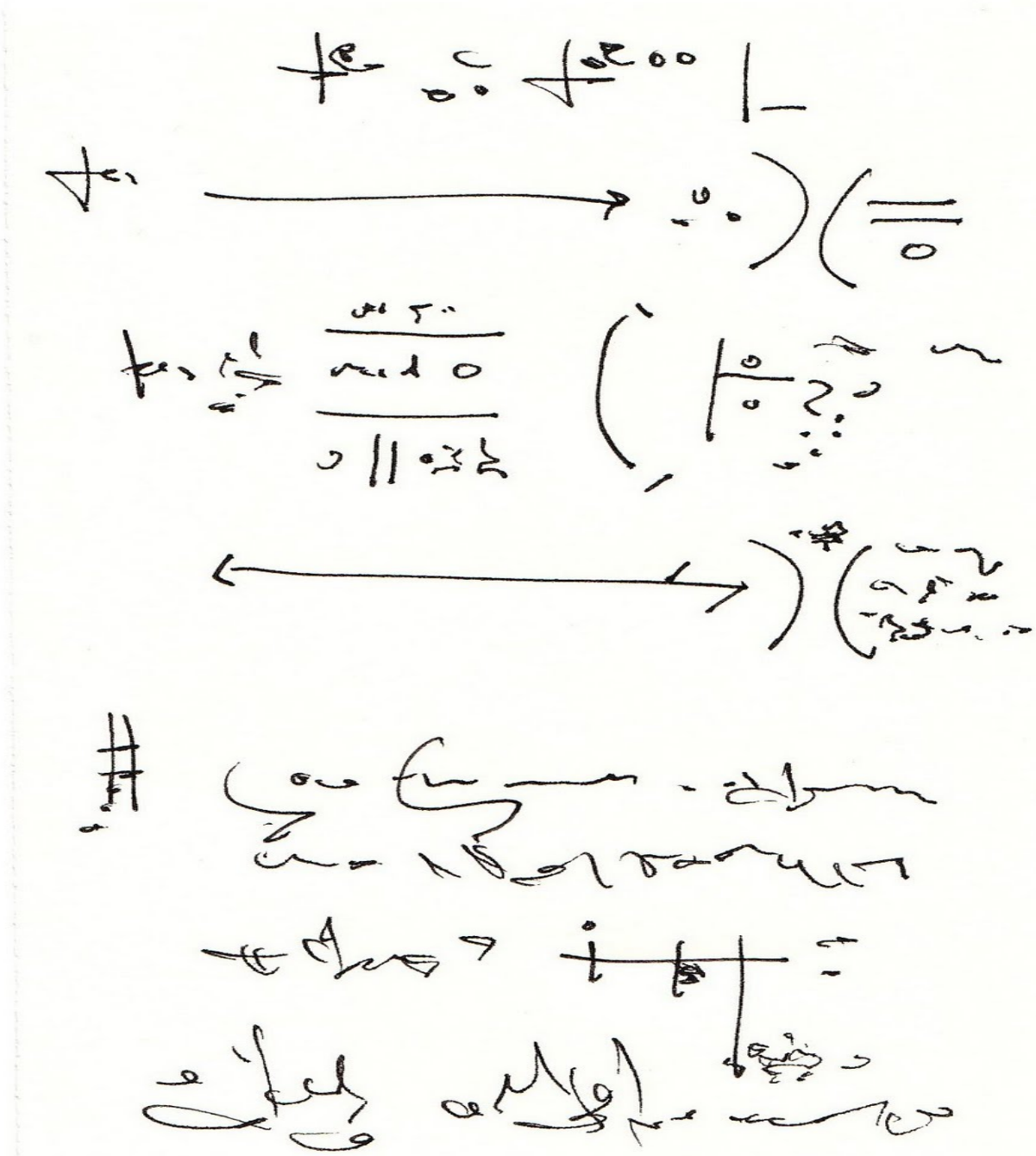
Segment building:

```
sgtree[symbol.testlist:token.NAME] ->
[symbol.testlist, ..., token.NAME]
```

Insert token and nest:

```
[symbol.testlist,[.., [token.NAME, 'a'] ..]
```

On Trail



Grammars Grammar

```
file_input: ( rule | NEWLINE )* ENDMARKER
rule: NAME ':' rhs NEWLINE
rhs: alt ( '|' alt )*
alt: item+
item: '[' rhs ']' | atom [ '*' | '+' ]
atom: '(' rhs ')' | NAME | STRING
```

Grammars Grammar - Terminals

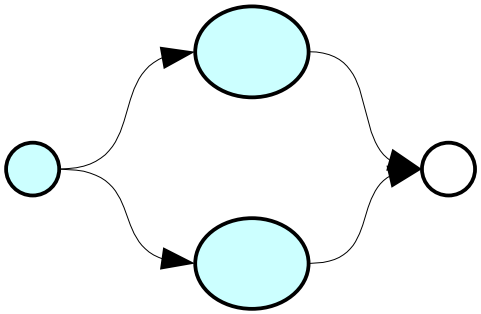
```
file_input: ( rule | NEWLINE )* ENDMARKER
rule: NAME ':' rhs NEWLINE
rhs: alt ( '|' alt )*
alt: item+
item: '[' rhs ']' | atom [ '*' | '+' ]
atom: '(' rhs ')' | NAME | STRING
```

Terminals

Grammars Grammar - Alternative

```
file_input: ( rule | NEWLINE )* ENDMARKER
rule: NAME ':' rhs NEWLINE
rhs: alt ( '|' alt )*
alt: item+
item: '[' rhs ']' | atom [ '*' | '+' ]
atom: '(' rhs ')' | NAME | STRING
```

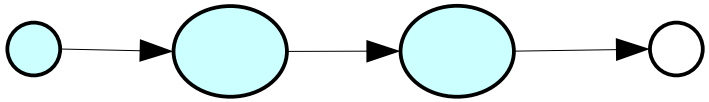
Alternative



Grammars Grammar - Cat

```
file_input: ( rule | NEWLINE )* ENDMARKER
rule: NAME ':' rhs NEWLINE
rhs: alt ( '|' alt )*
alt: item+
item: '[' rhs ']' | atom [ '*' | '+' ]
atom: '(' rhs ')' | NAME | STRING
```

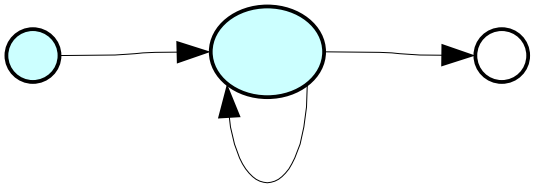
Concatenation



Grammars Grammar - *

```
file_input: ( rule | NEWLINE )* ENDMARKER
rule: NAME ':' rhs NEWLINE
rhs: alt ( '|' alt )*
alt: item+
item: '[' rhs ']' | atom [ '*' | '+' ]
atom: '(' rhs ')' | NAME | STRING
```

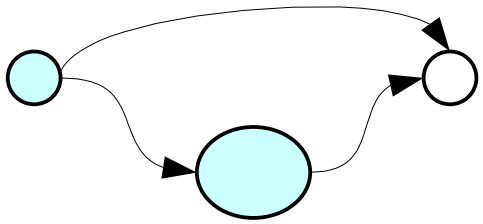
Kleene Star



Grammars Grammar - Maybe?

```
file_input: ( rule | NEWLINE )* ENDMARKER
rule: NAME ':' rhs NEWLINE
rhs: alt ( '|' alt )*
alt: item+
item: '[' rhs ']' | atom [ '*' | '+' ]
atom: '(' rhs ')' | NAME | STRING
```

Maybe



The parser sandwich

```
Lexer    : string       -> token-stream
Postlexer: token-stream -> token-stream
Parser   : token-stream -> parse-tree
```

The parser sandwich

```
Lexer    : string       -> token-stream
Postlexer: token-stream -> token-stream
Parser   : token-stream -> parse-tree
```

Design Principles

- Keep Lexer and Parser simple, clean, generic and grammar oriented
- Handle tangled, context sensitive mess in the Postlexer, a scripting layer, which outputs a nice and eventually corrected token-stream

The parser sandwich – Layer 1

```
Lexer    : string       -> token-stream
Postlexer: token-stream -> token-stream
Parser   : token-stream -> parse-tree
```

- Declarative
- CFG (EBNF)
- Lexertoken = charsets
- Functional token like ANY, STOP, ...
- Support for NFA expansions
- Debugging support (--dbg-lexer)

The parser sandwich – Layer 2

```
Lexer    : string       -> token-stream
Postlexer: token-stream -> token-stream
Parser   : token-stream -> parse-tree
```

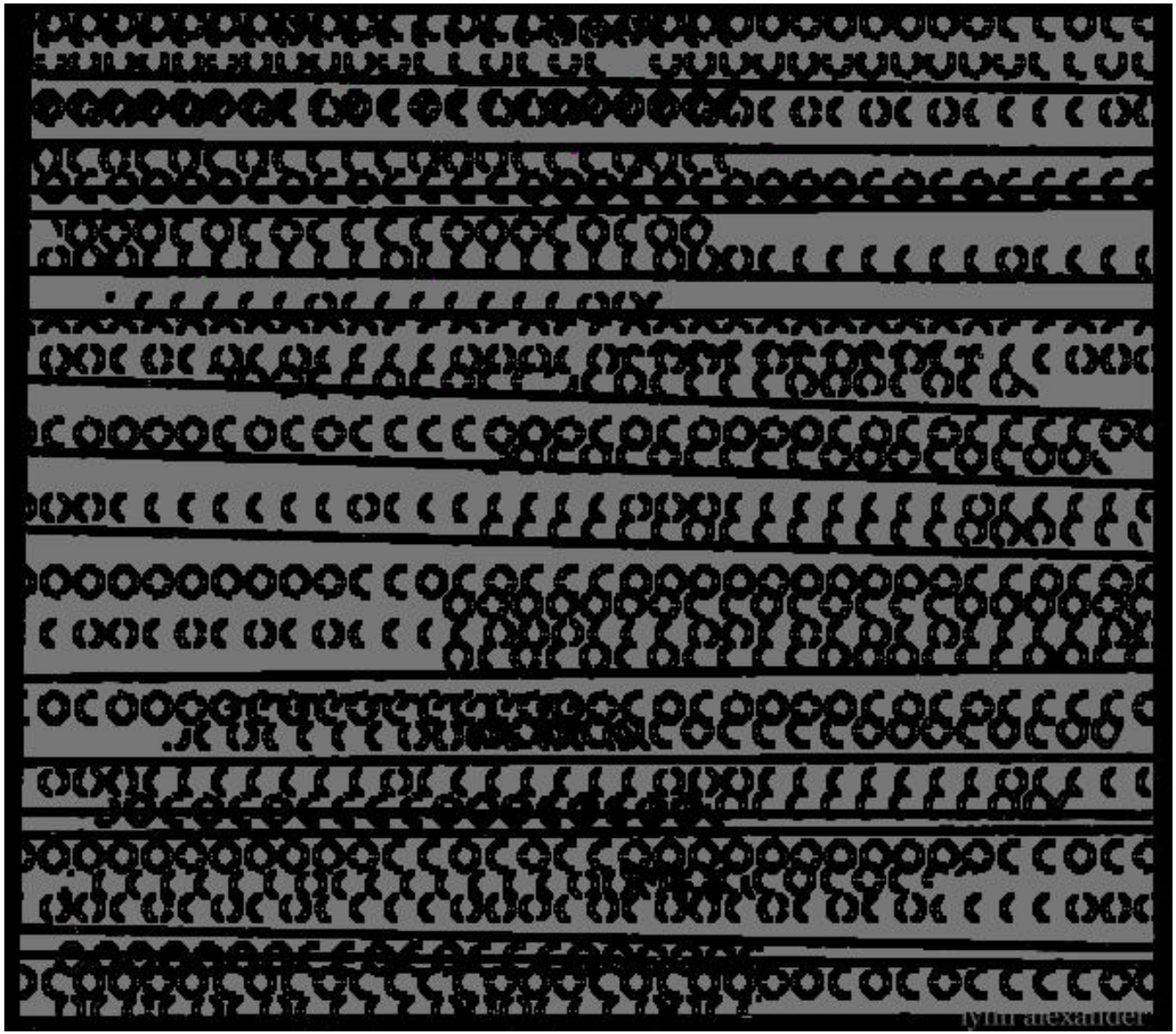
- Imperative/ OO / FP ... GP programming paradigm de jour
- Context sensitive analysis (symbol-tables in C)
- Inserting/Removing auxiliary token (INTRON -> NEWLINE, INDENT, DEDENT in Python)
- Strong tool support e.g. syntactical searchwithout parsing

The parser sandwich – Layer 3

```
Lexer    : string       -> token-stream
Postlexer: token-stream -> token-stream
Parser   : token-stream -> parse-tree
```

- Declarative
- CFG (EBNF)
- Support for NFA expansion + backtracking & memoization
- Debugging support (--dbg-parser)

Groundwork



From grammar rule to NFA

```
listmaker: (test ( list_for | (',' test)* [','] ))
```

Indexing

```
(listmaker,0): ((test,1) ((list_for,2) | ((',',3)(test,4))*
                                                  [(',',5)])
```


Table generation (follow sets)

```
{(listmaker,0):  [(test,1)],
 (test, 1):      [(list_for,2),(',', 3),(',',5),(None,'-')],
 (list_for,2):   [(None,'-')],
 (',', 3):       [(test, 4)],
 (test,4):       [(',', 3),(',',5),(None,'-')],
 (',',5):        [(None,'-')]
}
```

From grammar rule to NFA

```
listmaker: (test ( list_for | (',' test)* [',']))
```

Indexing

```
(listmaker,0): ((test,1) ((list_for,2) | ((',',3)(test,4))*
                                                  [(',',5)])
```

Table generation (follow sets)

```
{(listmaker,0):  [(test,1)],
 (test, 1):      [(list_for,2),(',', 3),(',',5),(None,'-')],
 (list_for,2):   [(None,'-')],
 (',', 3):       [(test, 4)],
 (test,4):       [(',', 3),(',',5),(None,'-')],
 (',',5):        [(None,'-')]
}
```

NFA Transition Table

listmaker: test (list_for | (',' test)* [','])

state/follow	comma	test	list_for	○
(listmaker,0)		(test,1)		
(test,1)	(comma,3) (comma,5)		(list_for, 2)	
(comma,3)		(test,4)		
(comma,5)				○
(test,4)	(comma,3) (comma,5)			○
(list_for,2)				○

NFA Tracing – Step 0

→ **listmaker**

state/follow	comma	test	list_for	○
(listmaker,0)		(test,1)		
(test,1)	(comma,3) (comma,5)		(list_for, 2)	○
(comma,3)		(test,4)		
(comma,5)				○
(test,4)	(comma,3) (comma,5)			○
(list_for,2)				○

NFA Tracing – Step 1

→ **listmaker** → **test**

state/follow	comma	test	list_for	○
(listmaker,0)		(test,1)		
(test,1)	(comma,3) (comma,5)		(list_for, 2)	○
(comma,3)		(test,4)		
(comma,5)				○
(test,4)	(comma,3) (comma,5)			○
(list_for,2)				○

NFA Tracing – Step 2

→ **listmaker** → **test** → **comma**

state/follow	comma	test	list_for	○
(listmaker,0)		(test,1)		
(test,1)	(comma,3) (comma,5)		(list_for, 2)	○
(comma,3)		(test,4)		
(comma,5)				○
(test,4)	(comma,3) (comma,5)			○
(list_for,2)				○

NFA Tracing – Step 3

→ listmaker → test → comma → test

state/follow	comma	test	list_for	○
(listmaker,0)		(test,1)		
(test,1)	(comma,3) (comma,5)		(list_for, 2)	○
(comma,3)		(test,4)		
(comma,5)				○
(test,4)	(comma,3) (comma,5)			○
(list_for,2)				○

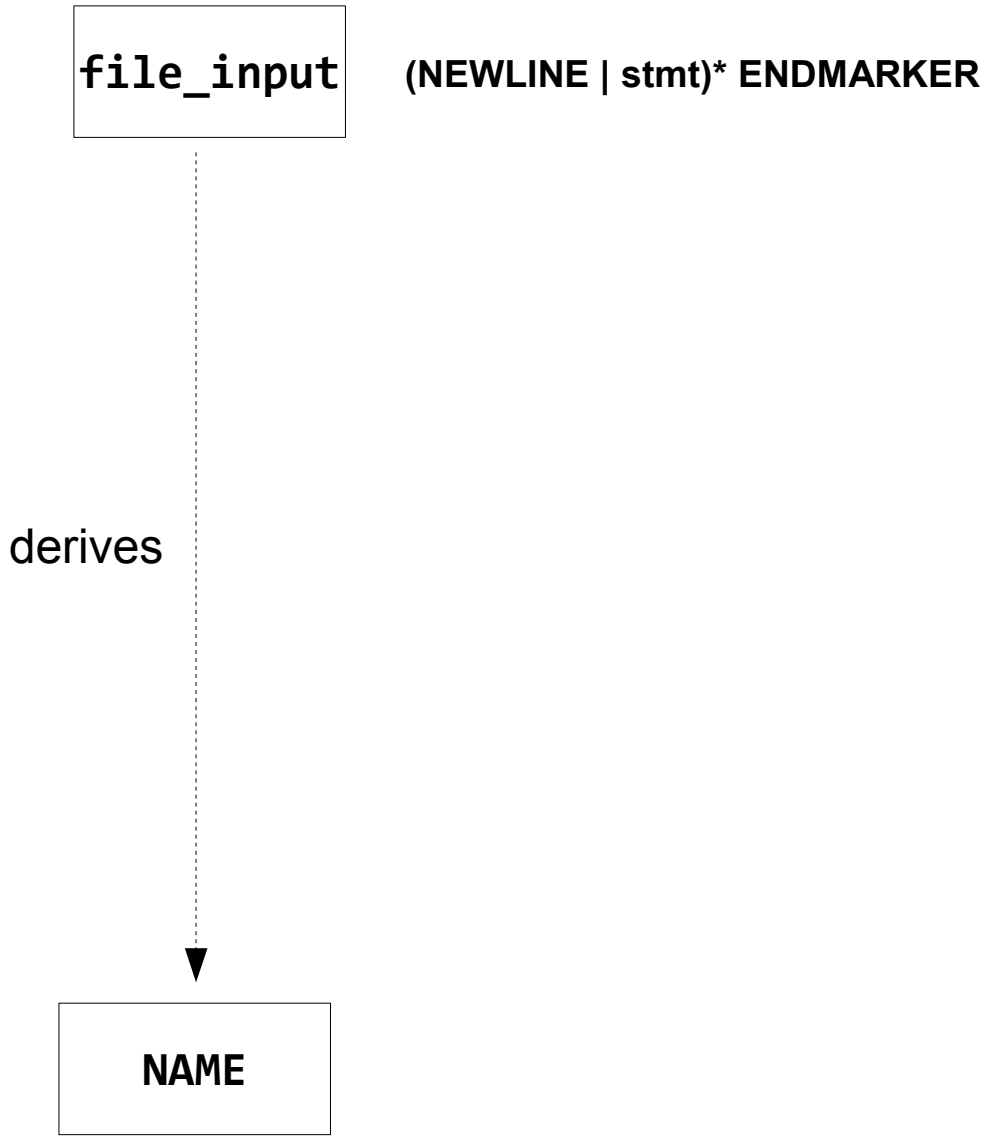
NFA Tracing – Step 4

→ listmaker → test → comma → test → None

state/follow	comma	test	list_for	○
(listmaker,0)		(test,1)		
(test,1)	(comma,3) (comma,5)		(list_for, 2)	○
(comma,3)		(test,4)		
(comma,5)				○
(test,4)	(comma,3) (comma,5)			○
(list_for,2)				○

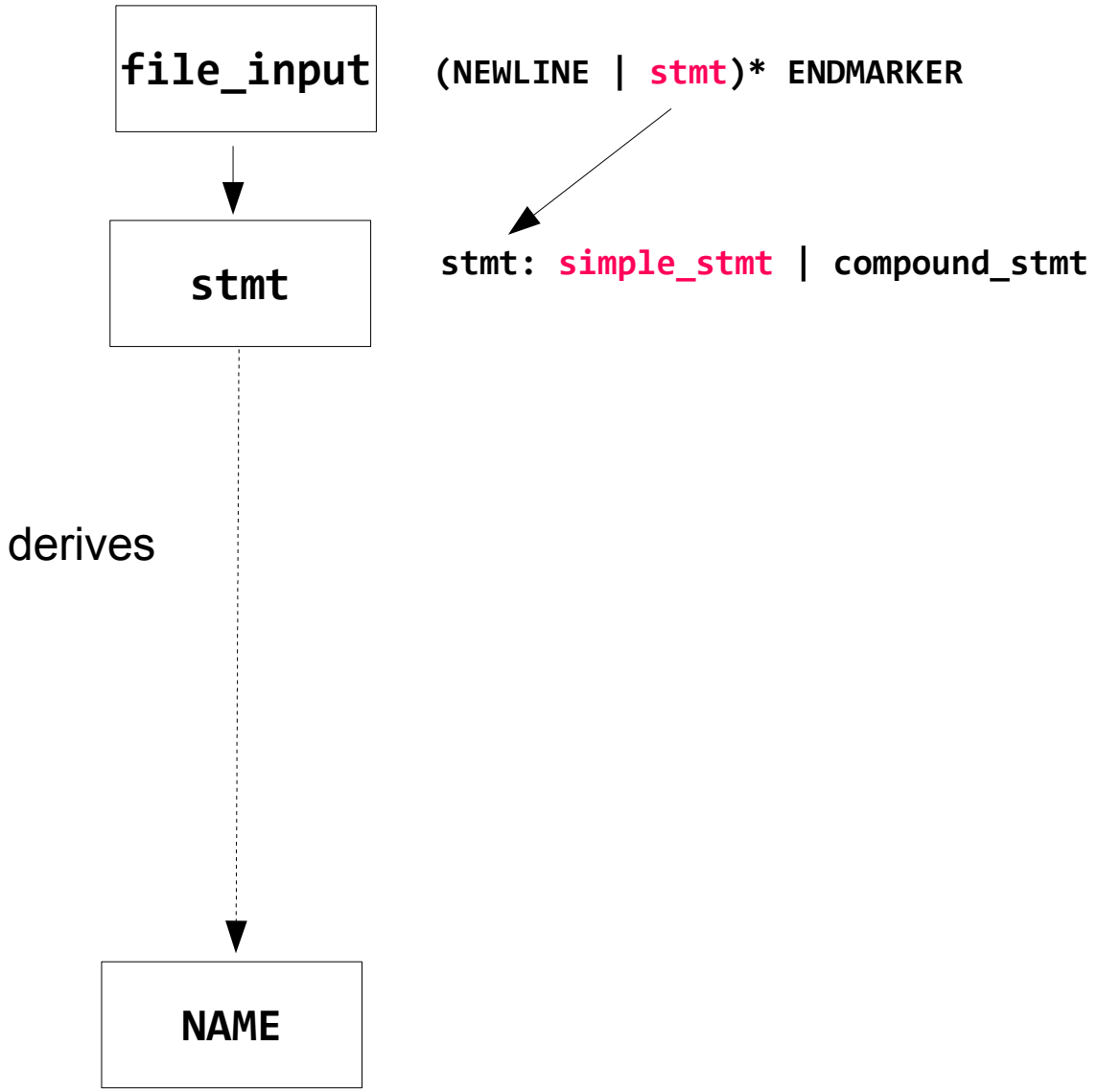
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



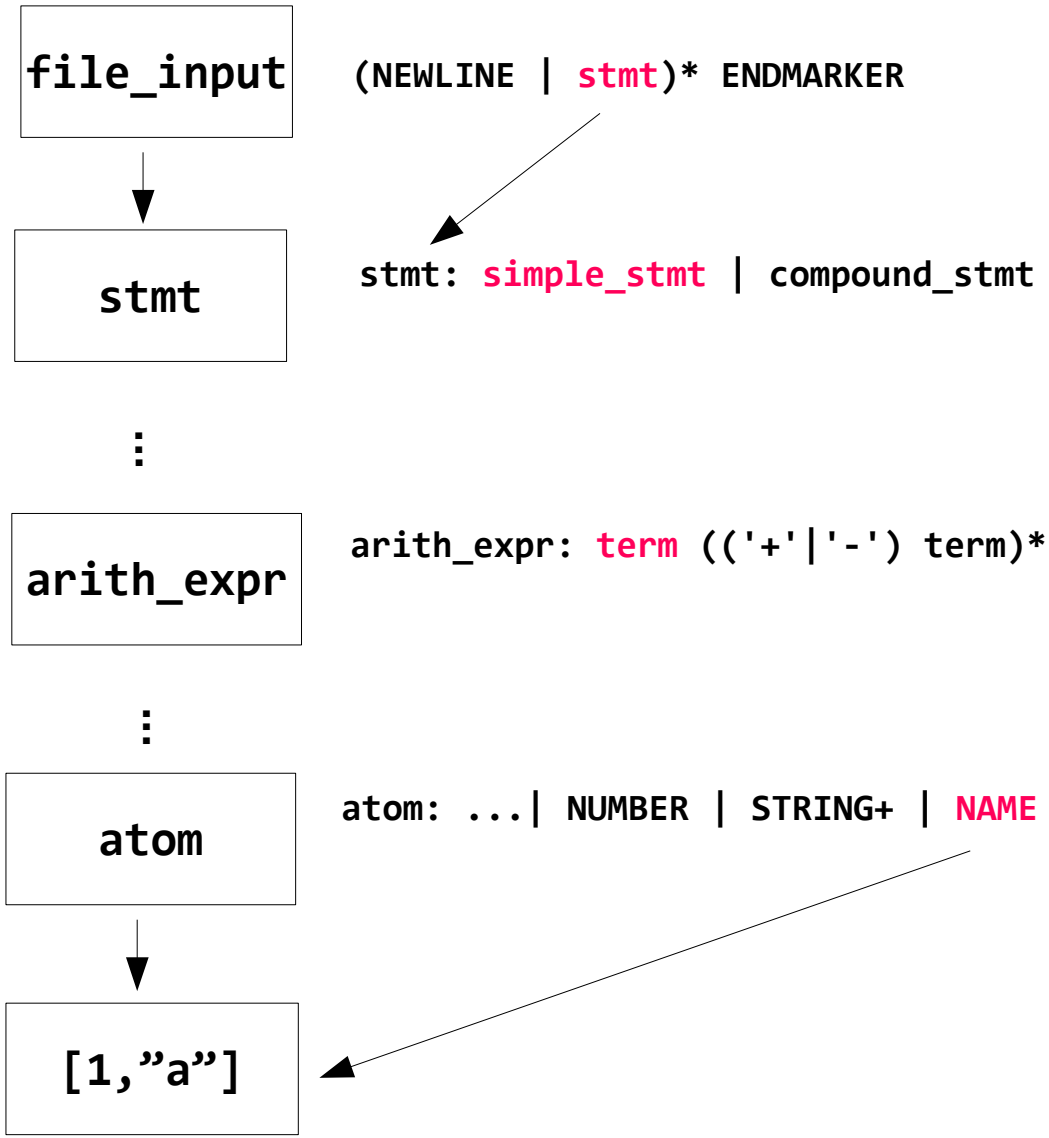
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



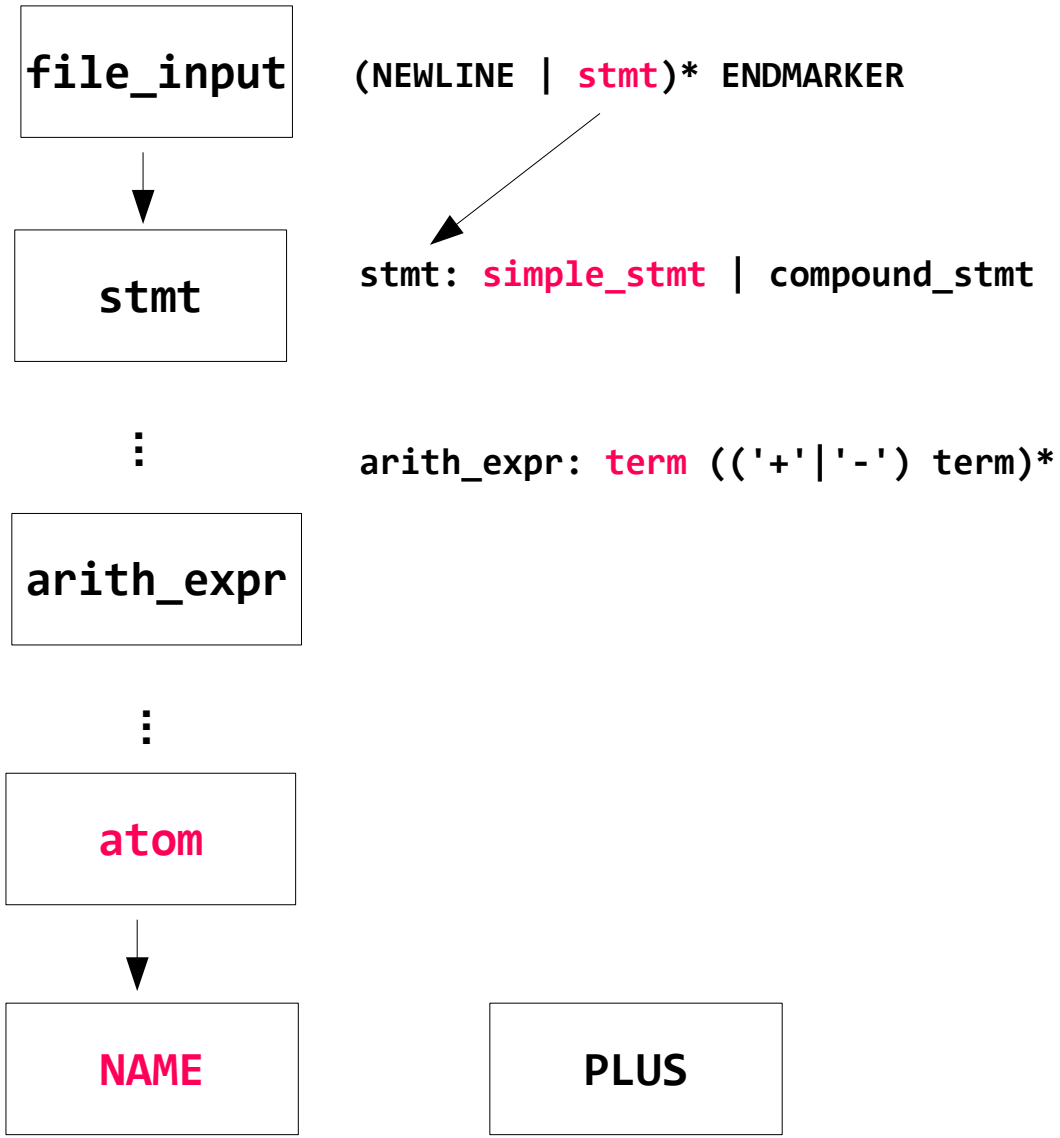
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



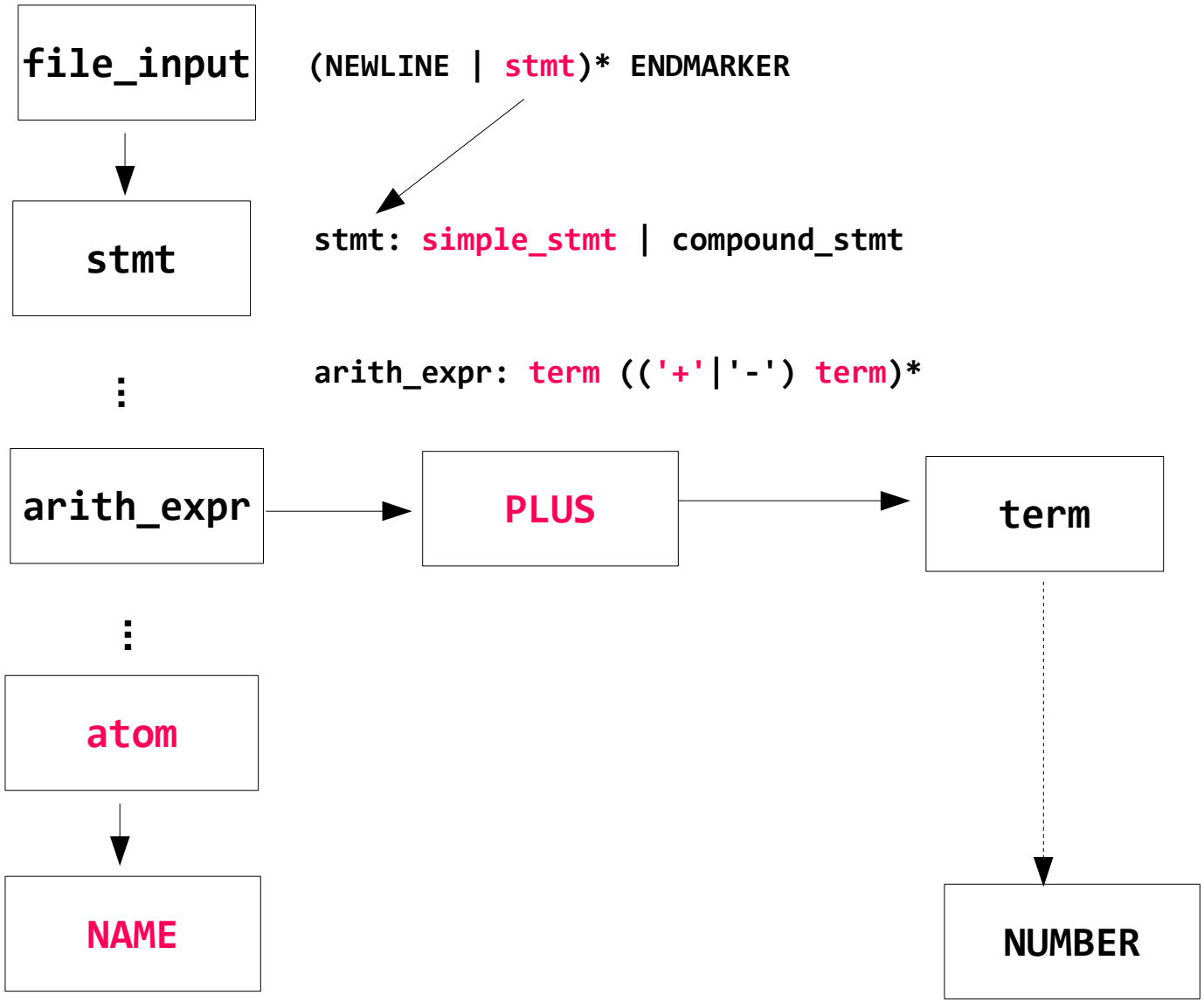
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



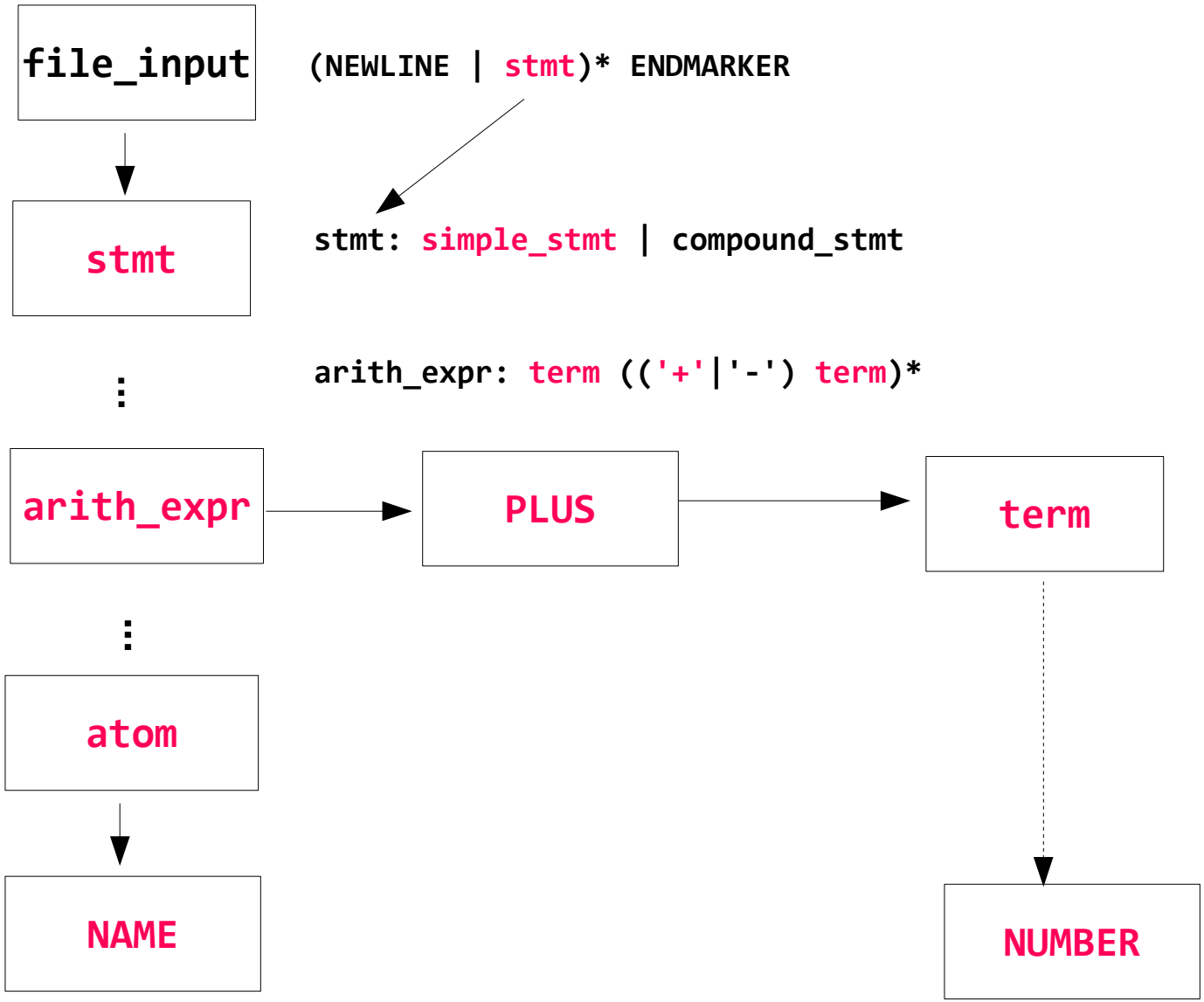
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



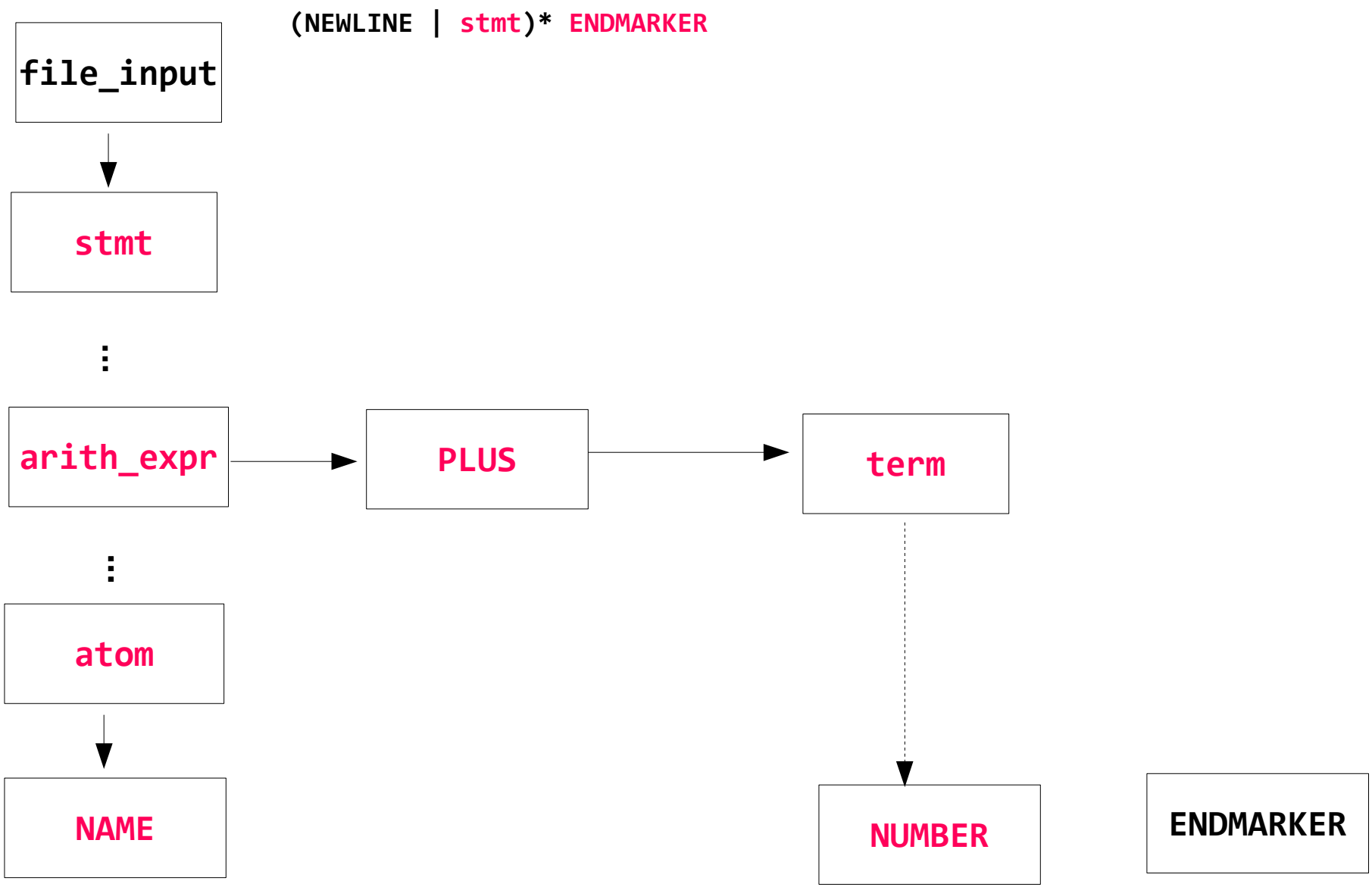
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



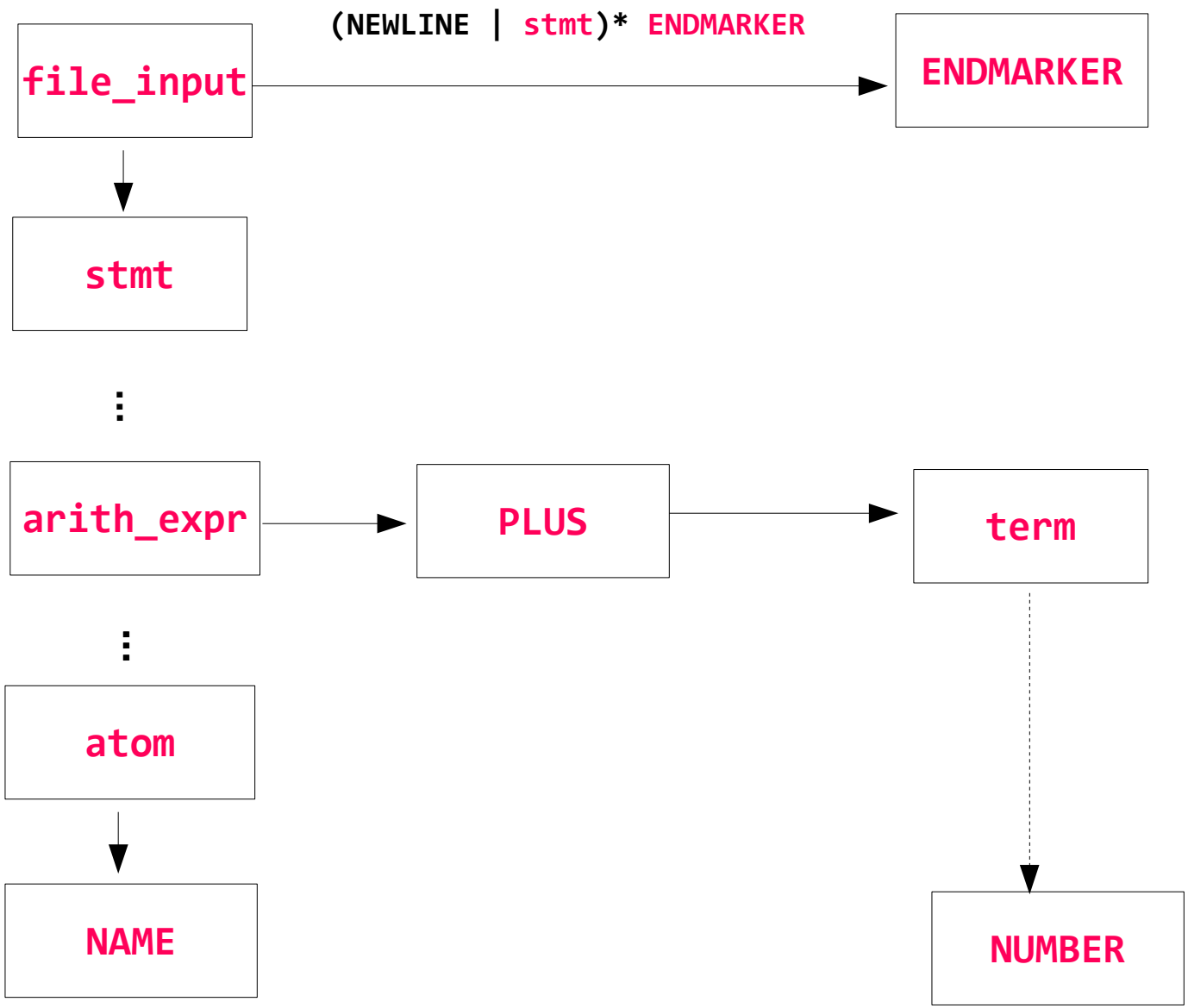
Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



Top down parsing

Build parse tree from “a+1” -> NAME,PLUS,NUMBER,ENDMARKER



Embedding

Grammar with First/First conflict - grammar not in LL(1)

```
R: A | B
A: 'x' 'y'
B: 'x' 'z'
```

Possible resolution strategy: careful embedding of A, B in R

```
R: ('x' 'y' <A> | 'x' 'z' <B> )
A: 'x' 'y'
B: 'x' 'z'
```

Embedding

Grammar with First/First conflict - grammar not in LL(1)

```
R: A | B
A: 'x' 'y'
B: 'x' 'z'
```

Possible resolution strategy: careful embedding of A, B in R

```
R: ('x' 'y' <A> | 'x' 'z' <B> )
A: 'x' 'y'
B: 'x' 'z'
```

Advancing states

Preparing states for embedding

```
(R,0): (A,1), (B,2)
(A,1): (None,'-')
(B,2): (None,'-')
```

→

```
(R,0,0,R): (A,1,0,R),(B,2,0,R)
(A,1,0,R): (None,'-',0,R)
(B,2,0,R): (None,'-',0,R)
```

Advancing states

Preparing states for embedding

```
(R,0): (A,1), (B,2)
(A,1): (None,'-')
(B,2): (None,'-')
```

→

```
(R,0,0,R): (A,1,0,R),(B,2,0,R)
(A,1,0,R): (None,'-',0,R)
(B,2,0,R): (None,'-',0,R)
```

```
(A,0)  : ('x',1)
('x',1): ('y',2)
('y',2): (None,'-')
```

→

```
(A,0,0,A)  : ('x',1,0,A)
('x',1,0,A): ('y',2,0,A)
('y',2,0,A): (None,'-',0,A)
```

```
(B,0)  : ('x',1)
('x',1): ('z',2)
('z',2): (None,'-')
```

→

```
(B,0,0,B)  : ('x',1,0,B)
('x',1,0,B): ('z',2,0,B)
('y',2,0,B): (None,'-',0,B)
```

NFA Expansion

```
(R,0,0,R)   : ('x',1,0,A),('x',4,0,B)
('x',1,0,A): ('y',2,0,A)
('y',2,0,A): (A,3,1,R)
(A,3,1,R)   : (None,'-',0,R)
('x',4,0,B): ('z',5,0,B)
('z',5,0,B): (B,6,1,R)
(B,6,1,R)   : (None,'-',0,R)
```

Parsetree from Trace

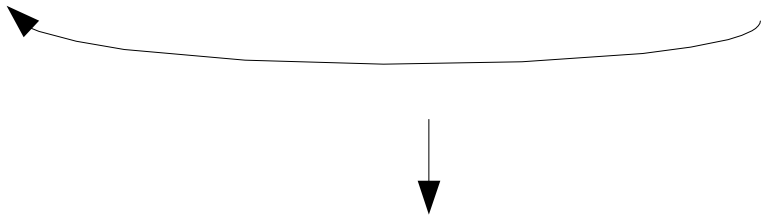
```
(R,0,0,R)   : ('x',1,0,A),('x',4,0,B)
('x',1,0,A): ('y',2,0,A)
('y',2,0,A): (A,3,1,R)
(A,3,1,R)   : (None,'-',0,R)
('x',4,0,B): ('z',5,0,B)
('z',5,0,B): (B,6,1,R)
(B,6,1,R)   : (None,'-',0,R)
```

```
(R,0,0,R) ('x',1,0,A) ('y',1,0,A) (A,3,1,R) (None,'-',0,R)
```

Parsetree from Trace

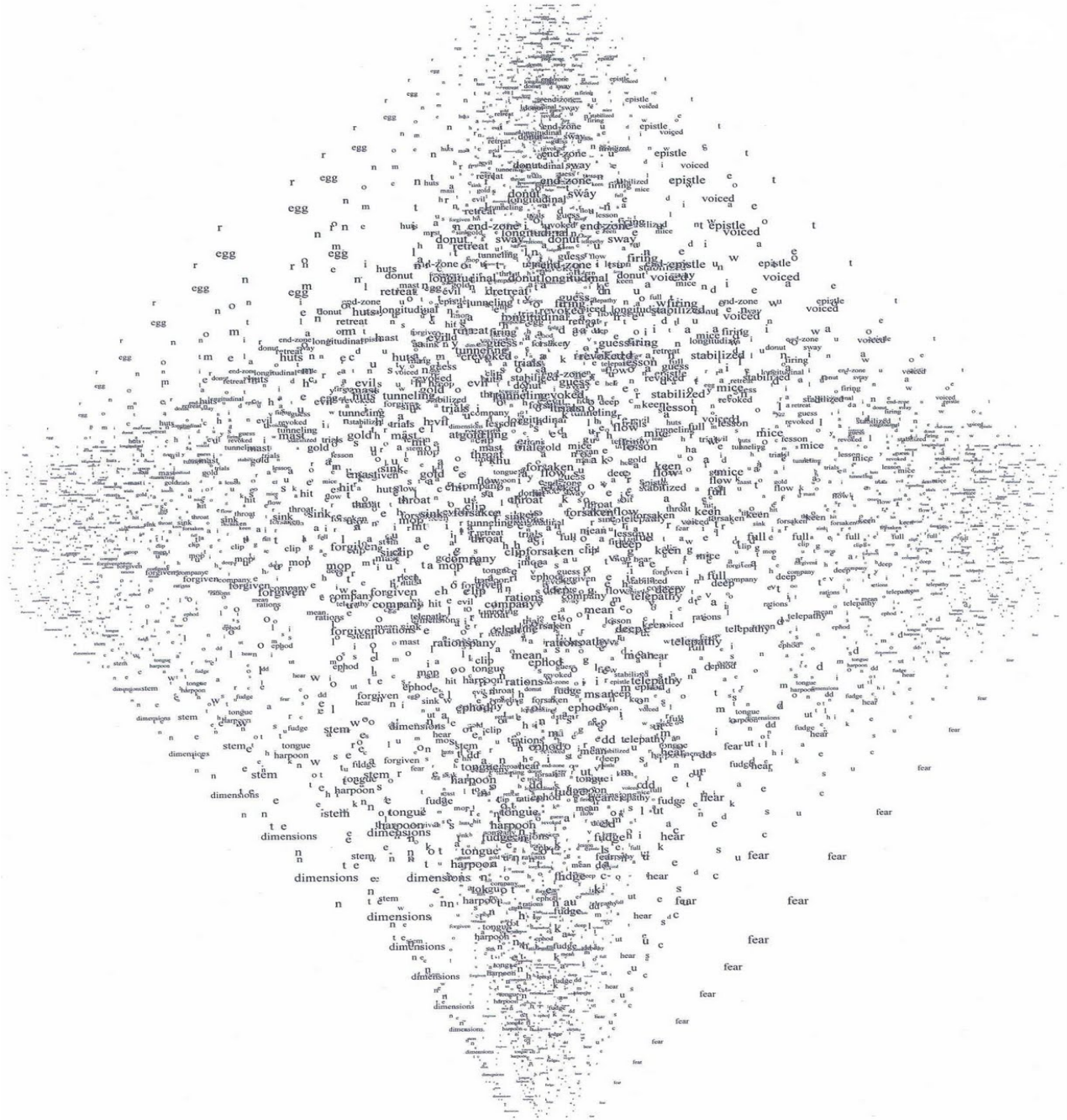
```
(R,0,0,R)   : ('x',1,0,A),('x',4,0,B)
('x',1,0,A): ('y',2,0,A)
('y',2,0,A): (A,3,1,R)
(A,3,1,R)   : (None,'-',0,R)
('x',4,0,B): ('z',5,0,B)
('z',5,0,B): (B,6,1,R)
(B,6,1,R)   : (None,'-',0,R)
```

```
(R,0,0,R) ('x',1,0,A) ('y',1,0,A) (A,3,1,R) (None,'-',0,R)
```



```
[R,[A,['x'],['y']]]
```

Langlets in the langscape



Gallery

thunk

```
x = property:
    def fget(self):
        return self._x

    def fset(self, value):
        self._x = value*2
```

switch

```
switch ip:
    case 127.0.0.1:
        print “localhost”
    case 192.168.1.33:
        print “wireless”
else:
    pass
```

repeat

```
repeat:
    self._x+=1
    f(self._x)
until:
    g(self.x) == True
```

if .. as

```
if re.compile(s).match as m:
    if m.span(0)[0] as k:
        self._x = k
```

Thunk Transformation

```
x = property:
    def fget(self):
        return self._x

    def fset(self, value):
        self._x = value*2
```

```
def thunk():
    def fget(self):
        return self._x

    def fset(self, value):
        self._x = value

    return locals()

x = property(**thunk())

del thunk
```

From Gallery to Python – part II

```
switch ip:
    case 127.0.0.1:
         print “localhost”
     case  192.168.1.33:
         print “wireless”
else:
     pass
```

Switch Transformation

```
if isChainlet(ip):
    SELECT_35412 = (ip).select(
       (ip.IPv4((127, 0, 0, 1,))),
       (ip.IPv4((192, 168, 1, 33,))))
else:
    SELECT_35412 = ip
if (ip.IPv4((127, 0, 0, 1,))) == SELECT_35412:
    print "localhost"
elif (ip.IPv4((192, 168, 1, 33,))) == SELECT_35412:
    print "wireless"
else:
    pass
del SELECT_35412
```

Wanted!

Embedded XML literals in Python and access syntax as in

http://learn.adobe.com/wiki/display/Flex**/E4X**

but ...

Wanted!

Embedded XML literals in Python and access syntax as in

http://learn.adobe.com/wiki/display/Flex**/E4X**

but ...

XML is **ugly!**

E4X -> _4_ -> P4D

P4D literals

```
elm a:
    b(x = 0):
        “text”
    c:
        d(y=0 z=1)
        d(y=1 z=0)
```

P4D

eval

```
p4d> L = [1, 2, 3]
p4d> elm X:
....     item: &L
p4d> print X.p4dstr()
X:
    item: "1"
    item: "2"
    item: "3"
```

access

```
p4d> elm y:
....     x(a=0)
p4d> y.x.@a
'0'
```

names

```
p4d> p = a:b:
....     x-y(s = 0)
p4d> p
P4D<a:b>
p4d> p.child("x-y").@s
'0'
```

special

```
p4d> b = bl:x:
....     tag: "00"
....     len: &LEN
....     val: "00 07 AF"
....
p4d> b.hex()
"00 03 00 07 AF"
```

Teuton

Eine deutsche Programmiersprache

```
# Auf der Mauer auf der Lauer

Wanze = "Wanze "
tanzen = "tanzen "
k = 0
solange Wanze:
    Wanze  = Wanze[:-1]
    tanzen = tanzen[:-1]
    drucke "Auf der Mauer, auf der Lauer,"
    drucke "sitzt ne kleine %s."%Wanze
    drucke "Auf der Mauer, auf der Lauer,"
    drucke "sitzt ne kleine %s."%Wanze
    drucke "Seht euch mal die %s an,"%Wanze
    drucke "wie die %s %s kann,"%(Wanze,tanzen)
    drucke "Auf der Mauer, auf der Lauer,"
    drucke "sitzt ne kleine %s!"%Wanze
    drucke
```

Mehr Teuton

```
Klasse Mücke(Objekt):

  def steche(selbst, Opfer, n=1):
    wenn Opfer ist nicht Nichts:
      für i im Intervall(0, n):
          Opfer.bilde_quaddel(i)
```

```
teuton_kw = {
    "tu":"exec",
    "entf":"del",
    "drucke":"print",
    "weiter":"continue",
    "zurueck":"return",
    "zurück":"return",
    "ergibt":"yield",
    "behandle":"raise",
    "von":"from",
    "behaupte":"assert",
    "wenn":"if",
    "wennsonst":"elif",
    "versuche":"try",
    "ausser":"except",
    "schliesslich":"finally",
    "sonst":"else",
    "solange":"while",
    "fuer":"for",
    "für":"for",
    "ausser":"except",
    "Klasse":"class",
    "und":"and",
    "oder":"or",
    "nicht":"not",
    "im":"in"}
```

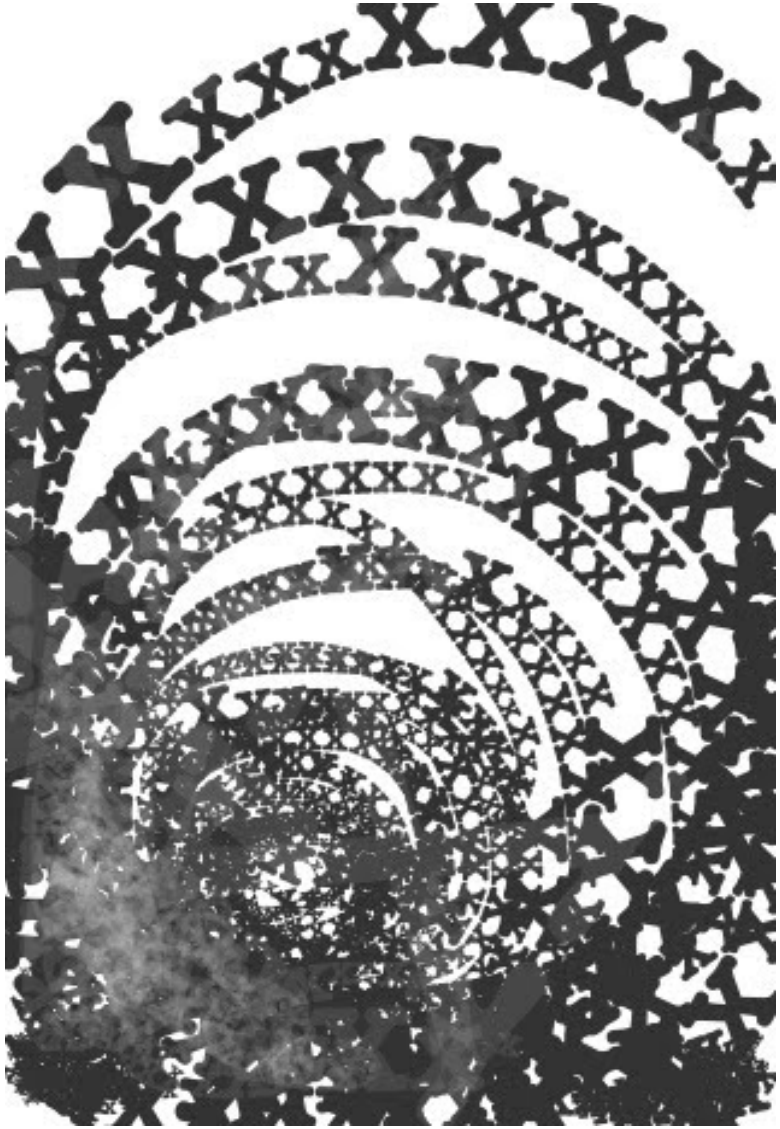
```
class booleana(int):
    def __init__(self,i):
        int.__init__(self,i)

    def __repr__(self):
        return self.name

Wahr = booleana(True)
Wahr.name = "Wahr"

Falsch = booleana(False)
Falsch.name = "Falsch"
```

Sourcetools



- Syntax directed search and replace
- Code Templates
- Systematic expression generation
- Genetic expression generation

Breeding Grammars

Genetic Programming for Grammar generation

- Expression Type: Grammar
- Population size: 40
- Generations: 400
- Goal: Find smallest grammar with an NFA expanding to 1000 states
- Found:
  - Grammar size measured in number of token: 48
  - Biggest rule in number of states: 999

```
S: d b 'u'+ a a* a | b d
a: d d d
b: [a a] a
c: 'x' 'u' 'w' e c 'v'
d: ['u' [c 'y' 'w'] e e] c
e: c
```

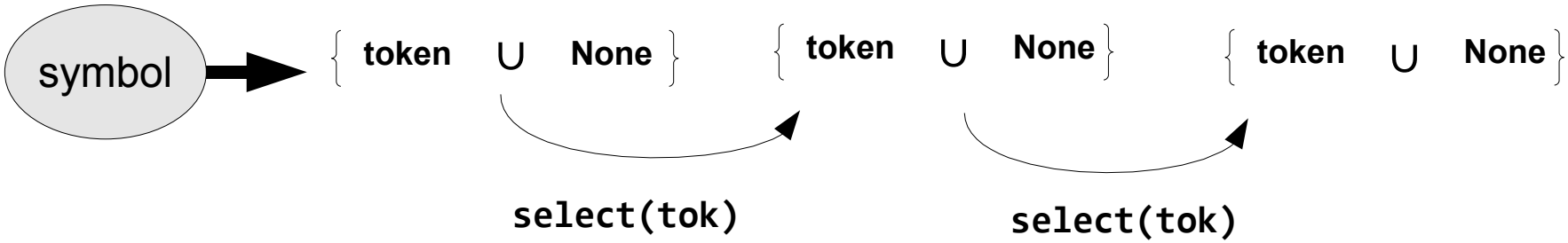
Breeding Bugs (tanzende Wanzen ...)

Testdata generation for bug detection

- Detected: NFA expansion failure using hundreds of grammars
- Tiny grammar for debugging

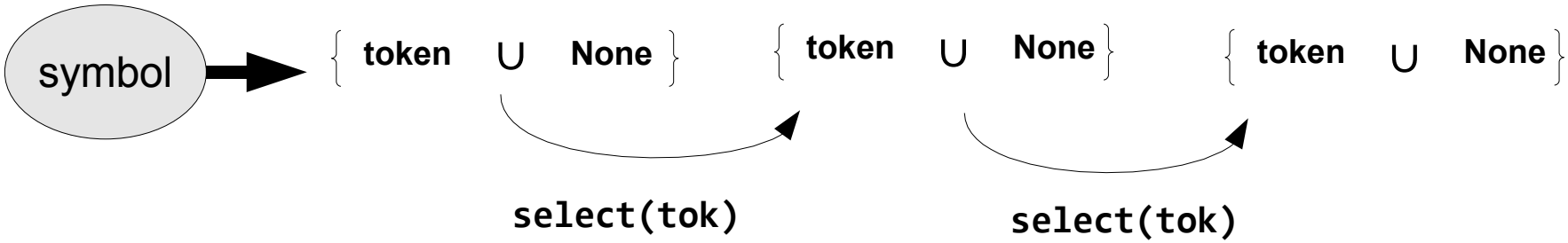
```
S: a+ 'x'
a: (b b)*
b: 'x'
```

Trails Tracers: more than NFA tracing




```
>>> from langscape.trail.nfatracer import TokenTracer
>>> tracer = TokenTracer(python_langlet, python_langlet.symbol.test)
>>> tracer.selectables()
['TILDE', 'NAME', 'NUMBER', 'STRING', 'LPAR', 'LSQB', 'lambda', ...]
```

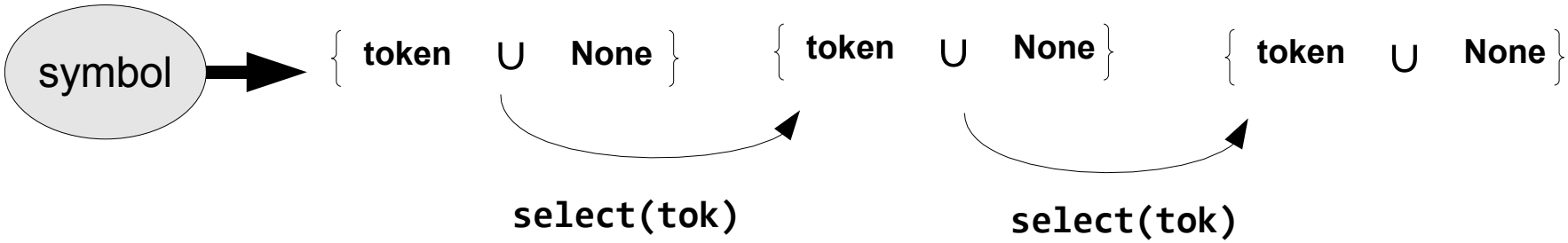
Trails Tracers: more than NFA tracing




```
>>> from langscape.trail.nfatracer import TokenTracer
>>> tracer = TokenTracer(python_langlet, python_langlet.symbol.test)
>>> tracer.selectables()
['TILDE', 'NAME', 'NUMBER', 'STRING', 'LPAR', 'LSQB', 'lambda', ...]
>>> tracer.select(python_langlet.token.LPAR)
['TILDE', 'NAME', 'NUMBER', 'STRING', 'yield', 'RPAR', 'LSQB', ...]
```

Token Tracers

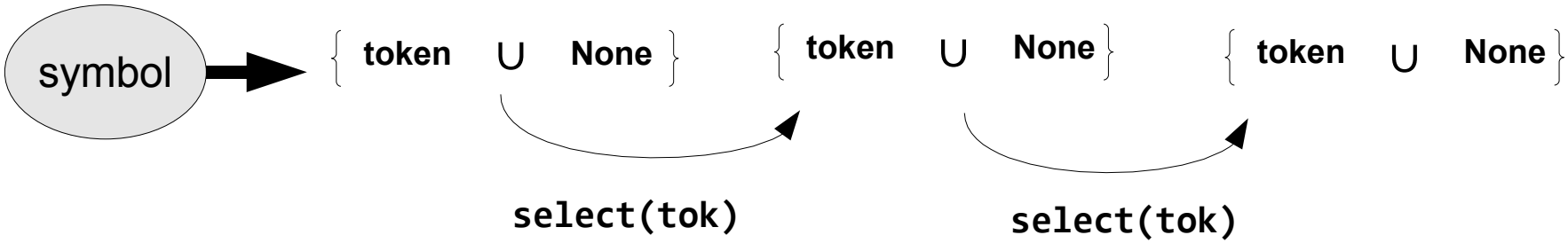
Trails Tracers: more than NFA tracing




```
>>> from langscape.trail.nfatracer import TokenTracer
>>> tracer = TokenTracer(python_langlet, python_langlet.symbol.test)
>>> tracer.selectables()
['TILDE', 'NAME', 'NUMBER', 'STRING', 'LPAR', 'LSQB', 'lambda', ...]
>>> tracer.select(python_langlet.token.LPAR)
['TILDE', 'NAME', 'NUMBER', 'STRING', 'yield', 'RPAR', 'LSQB', ...]
>>> tracer.select(python_langlet.token.NAME)
['is', '<>', 'in', 'if', 'LPAR', 'RPAR', 'for', 'COMMA', 'PLUS',...]
```

Token Tracers

Trails Tracers: more than NFA tracing




```
>>> from langscape.trail.nfatracer import TokenTracer
>>> tracer = TokenTracer(python_langlet, python_langlet.symbol.test)
>>> tracer.selectables()
['TILDE', 'NAME', 'NUMBER', 'STRING', 'LPAR', 'LSQB', 'lambda', ...]
>>> tracer.select(python_langlet.token.LPAR)
['TILDE', 'NAME', 'NUMBER', 'STRING', 'yield', 'RPAR', 'LSQB', ...]
>>> tracer.select(python_langlet.token.NAME)
['is', '<>', 'in', 'if', 'LPAR', 'RPAR', 'for', 'COMMA', 'PLUS',...]
>>> tracer.select(python_langlet.token.RPAR)
[None, 'is', '<>', 'in', 'if', 'LPAR', 'LSQB', ...]
```

Why TokenTracers?

Benefits

- Simple and efficient implementation (no backtracking)
- Derivable from grammars, just like CFG parsers

Usage

- Error detection in token streams
- Auto-insertion of missing token
- Support for syntactical search
- Expression generation

TokenTracer perspectives

Pure Tracer Challenge

- Getting rid of non-terminals in the implementation
- Understanding periods in traces (exhaustion)

Application

Two grammars G1, G2 which produce the same tracers identify the same language

Safe grammar refactoring G1 -> G2 possible

Future of Langscape

Universal Mind Decoder by Brian Lucas

Future of Langscape

Need for Speed

Give alternative implementations (C++, Scala) for Trail and some other parts of the framework.

Expansion vs Backtracking. The numbers

I still don't know if expansion is really worth it.

Measure, measure, measure!

If so, study cycles and possible extensions of trace based parsing on NFA level.

C

A C langlet, maybe even a preprocessor, written in Langscape will be the next practical think I need.

Future of Langscape

Need for Speed

Give alternative implementations (C++, Scala) for Trail and some other parts of the framework.

Expansion vs Backtracking. The numbers

I still don't know if expansion is really worth it.

Measure, measure, measure!

If so, study cycles and possible extensions of trace based parsing on NFA level.

C

A C langlet, maybe even a preprocessor, written in Langscape will be the next practical think I need.

Future of Langscape

Need for Speed

Give alternative implementations (C++, Scala) for Trail and
some other parts of the framework.

Expansion vs Backtracking. The numbers

I still don't know if expansion is really worth it.

Measure, measure, measure!

If so, study cycles and possible extensions of trace based parsing on NFA level.

C

A C langlet, maybe even a preprocessor, written in Langscape will be
the next practical think I need.

Thanks for coming!

www.fiber-space.de/wordpress

code.google.com/p/langscape/

Credits

The *asemic writing* artwork is taken from

http://thenewpostliterate.blogspot.com