Earley Parsing and Examples

Sometimes the second one is even better!

Outline

- Earley's Algorithm
  - Chart States
  - Operations
  - Example
- MyEarley.py
- PA3.jison
- Grammar “Conflicts”
  - Shift/Reduce

We've got an educational film for readers aged nine to fifteen, who hang out on message boards and rate games that haven't come out yet. The rest of you are excused.
"YOUR FRIEND AT THE END"
The PerioD
HEY KIDS!
NOW, I KNOW YOU LIKE MY FRIENDS EXCLAMATION POINT AND CAPITAL LETTER. WE CAN HAVE FUN, TOO!
zeldA!!! more like celda
I GO AT THE END OF YOUR SENTENCES - LIKE THIS! I LET PEOPLE KNOW WHEN IT'S OKAY TO BREATHE!
©2002 MIKE KRAHULIK AND JERRY HOLKINS
WWW.PENNY-ARCADE.COM

Administrivia

- Midterm 1 will take place Thursday in class
  - Everything including today is fair game.
- Class vote, pick one option:
  - You may bring one page of printed or hand-written notes, front-and-back (= 2 sides).
  - The test is open book: you may use any printed materials including your printed notes and/or the textbook and/or other printed readings.

- Think.

In One Slide

- Earley parsers are top-down and use dynamic programming. An Earley state records incremental information: when we started, what has been seen so far, and what we expect to see. The Earley chart holds a set of states for each input position. Shift, reduce and closure operations fill in the chart.
- You enjoy parsing. Parsing is easy and fun.

Review: Earley States

- Let *X* be a non-terminal
- Let *a* and *b* be (possibly-empty) sequences of terminals and non-terminals
- Let *X* → *ab* be a production in your grammar
- Let *j* be a position in the input
- Each **Earley State** is a tuple < *X* → *a* • *b* , *j* >
  - We are currently parsing an *X*
  - We have seen *a*, we expect to see *b*
  - We started parsing this *X* after seeing the first *j* tokens from the input.

Review: Earley Parse Table

- An **Earley parsing table** (or **chart**) is a one-dimensional array. Each array element is a set of Earley states.
  - chart[i] holds the set of valid parsing states we could be in after seeing the first *i* input tokens
- Then the string $tok_1 \ldots tok_n$ is in the language of a grammar with start symbol S *iff*
  - chart[n] contains < S → ab• , 0 > for some production rule S → ab in the grammar.
  - We then say the parser **accepts** the string.

Review: Filling In The Chart

- Three operations build up chart[n]
- The first is called **shift** or **scan**.
  - It corresponds to “seeing the next expected token” or “helping to confirm the current hypothesis” or “we're winning”.
- Example:
  - chart[1] contains $< E \rightarrow E \bullet + E , 0 >$
  - 2nd token is “+”
  - Then put $< E \rightarrow E + \bullet E , 0 >$ in chart[2]

Review: Filling In The Chart (2)

- The second operation is the **closure** or **predictor**.
  - It corresponds to "expanding rewrite rules" or "substituting in the definitions of non-terminals"
- Suppose the grammar is:

  S → E   E → E + E | E – E | int

- If chart[0] has < S → • E , 0 > then add

  < E → • E + E , 0 >

  < E → • E – E , 0 >

  < E → • int, 0 >

Review: Filling In The Chart (3)

- The third operation is **reduction** or **completion**.
  - It corresponds to “finishing a grammar rewrite rule” or “being done parsing a non-terminal” or “doing a rewrite rule in reverse and then shifting over the non-terminal”.
- Suppose:
  - E → int | E + E | E – E | (E) , input is “(int”
  - chart[2] contains < E → int • , 1 >
  - chart[1] contains < E → (• E) , 0 >
  - Then chart[2] += < E → (E •) , 0 >

Shift Practice

- chart[3] contains

  < S → E • , 0 >

  < E → E • + E , 0>

  < E → E • - E , 2>

  < E → int • , 2 >

  < E → E • - E , 0 >

  < E → E - E • , 0 >

  < E → E • + E , 2 >

- The 4th token is "+". What does shift bring in?

Shift Practice

- chart[3] contains

  < S → E • , 0 >

  < E → E • + E , 0>

  < E → E • - E , 2>

  < E → int • , 2 >

  < E → E • - E , 0 >

  < E → E - E • , 0 >

  < E → E • + E , 2 >

- The 4th token is “+”. What does **shift** bring in?

  < E → E + • E , 0>

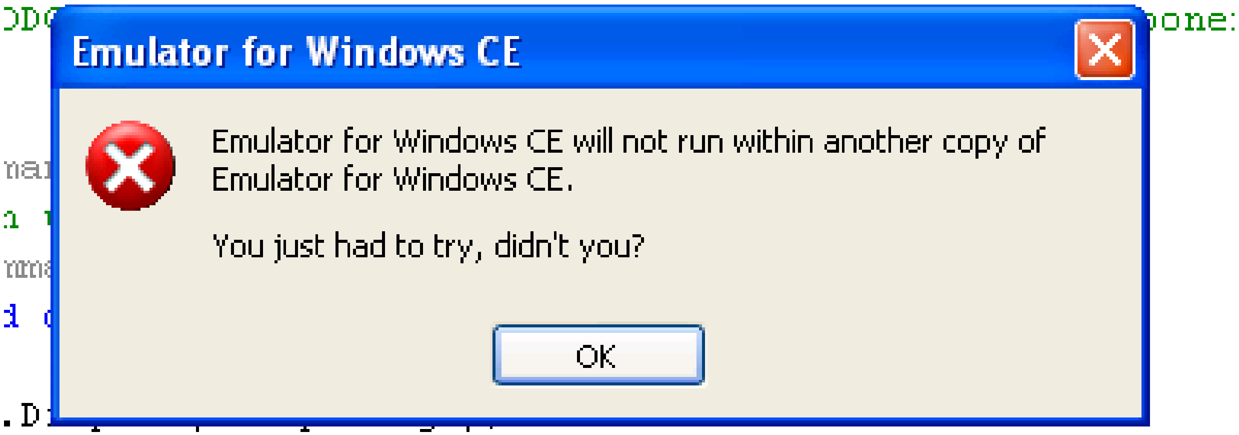
  < E → E + • E , 2 >

  … are both added to chart[4].

Closure Practice

- Grammar is
  - S → E   E → E + E | E - E | (E) | int
- chart[4] contains:

  < E → E + • E , 0 >   < E → E + • E , 2 >
- What does the closure operation bring in?



Emulator for Windows CE will not run within another copy of Emulator for Windows CE.

You just had to try, didn't you?

Closure Practice

- Grammar is
  - S → E   E → E + E | E – E | (E) | int
- chart[4] contains:

  < E → E + • E , 0 >   < E → E + • E , 2 >

- What does the closure operation bring in?

  < E → • E + E , 4 >   < E → • E – E , 4 >

  < E → • (E) , 4 >   < E → • int , 4 >

  … are all added to chart[4].

Reduction Practice

- chart[4] contains:
  - < E → E + • E , 0 >
  - < E → • E + E , 4 >
  - < E → • (E) , 4 >
  - < E → E + • E , 2 >
  - < E → • E - E , 4 >
  - < E → • int , 4 >
- chart[5] contains:
  - < E → int • , 4 >
- What does the reduce operator bring in?

Reduction Practice

- chart[4] contains:

  $< E \rightarrow E + \bullet\ E , 0 >$

  $< E \rightarrow \bullet\ E + E , 4 >$

  $< E \rightarrow \bullet\ (E) , 4 >$

  $< E \rightarrow E + \bullet\ E , 2 >$

  $< E \rightarrow \bullet\ E - E , 4 >$

  $< E \rightarrow \bullet\ \text{int} , 4 >$

- chart[5] contains:
  - $< E \rightarrow \text{int}\ \bullet , 4 >$
- What does the reduce operator bring in?

  $< E \rightarrow E + E\ \bullet , 0 >$

  $< E \rightarrow E\ \bullet + E , 4 >$

  $< E \rightarrow E + E\ \bullet , 2 >$

  $< E \rightarrow E\ \bullet - E , 4 >$

  - … are all added to chart[5]. (Plus more in a bit!)

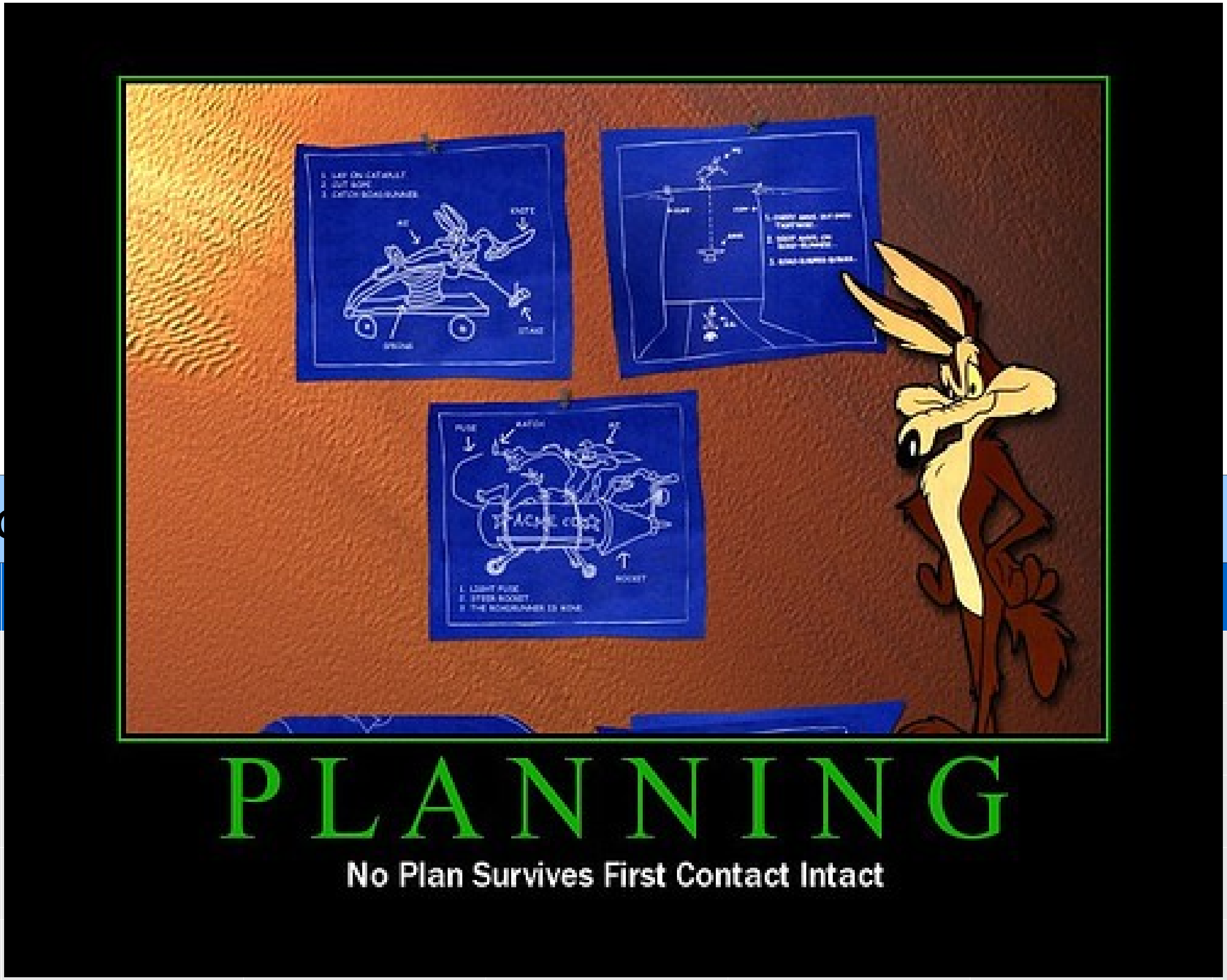
Earley Parsing Algorithm

- Input: CFG G, Tokens $tok_1 \ldots tok_n$
- Work:

```
chart[0] = { < S → •ab , 0 > }
for i = 0 to n
  repeat
    use shift, reduce and closure on chart[i]
  until no new states are added
```

- Output:
  - true iff < S → ab• , 0 > in chart[n]

Massive Earley Example



ic

chart[0]						chart[6]
S→•F ,0						

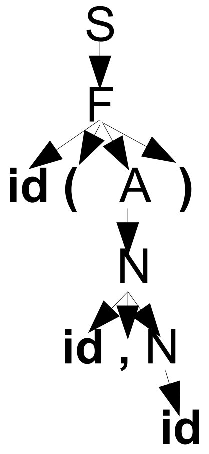
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0						

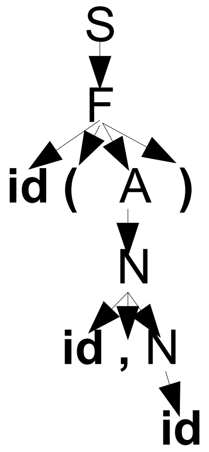
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0						
F→•id(A) ,0						

Closure on F

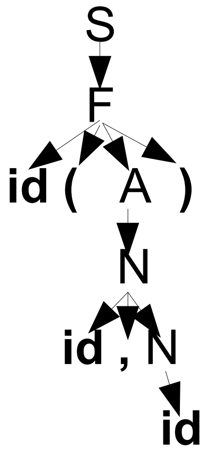
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0					
F→•id(A) ,0						

Shift on “id”

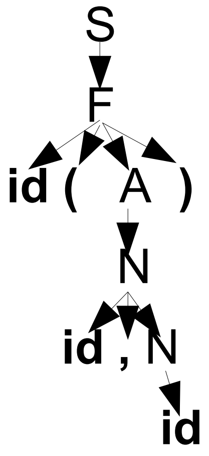
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0				
F→•id(A) ,0						

Shift on “(”

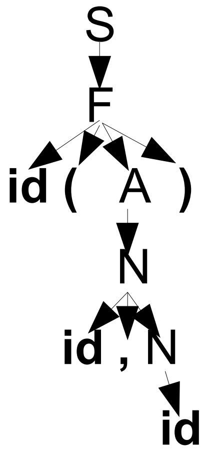
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0				
F→•id(A) ,0		A→•N ,2				
		A→• ,2				

Closure on A

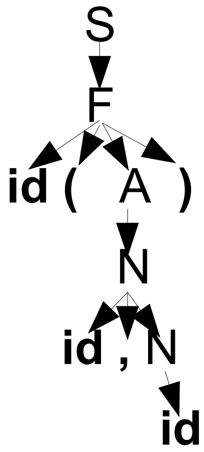
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0				
F→•id(A) ,0		A→•N ,2				
		A→• ,2				
		N→•id ,2				
		N→•id,N ,2				

Closure on N

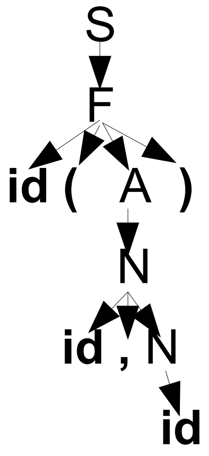
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0				
F→•id(A) ,0		A→•N ,2				
		A→• ,2				
		N→•id ,2				
		N→•id,N ,2				
		F→id(A•) ,0				

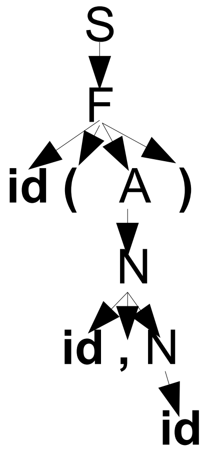
Reduce on A

Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input
id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2			
F→•id(A) ,0		A→•N ,2	N→id•,N ,2			
		A→• ,2				
		N→•id ,2				
		N→•id,N ,2				
		F→id(A•) ,0				

Shift on “id”

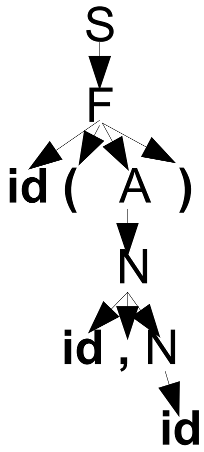
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2			
F→•id(A) ,0		A→•N ,2	N→id•,N ,2			
		A→• ,2	A→N• ,2			
		N→•id ,2				
		N→•id,N ,2				
		F→id(A•) ,0				

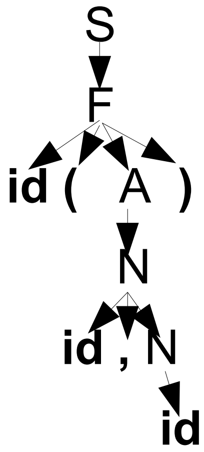
Reduce on N

Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input
id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2			
F→•id(A) ,0		A→•N ,2	N→id•,N ,2			
		A→• ,2	A→N• ,2			
		N→•id ,2	F→id(A•) ,0			
		N→•id,N ,2				
		F→id(A•) ,0				

Reduce on A

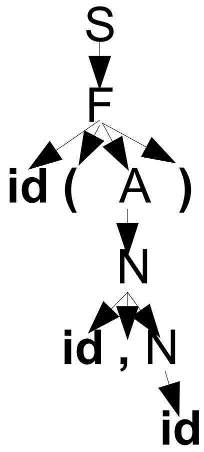
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2		
F→•id(A) ,0		A→•N ,2	N→id•,N ,2			
		A→• ,2	A→N• ,2			
		N→•id ,2	F→id(A•) ,0			
		N→•id,N ,2				
		F→id(A•) ,0				

Shift on “,”

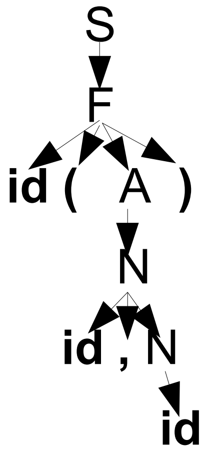
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2		
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4		
		A→• ,2	A→N• ,2	N→•id,N ,4		
		N→•id ,2	F→id(A•) ,0			
		N→•id,N ,2				
		F→id(A•) ,0				

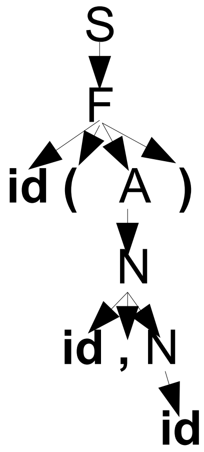
Closure on N

Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input
id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	
		A→• ,2	A→N• ,2	N→•id,N ,4		
		N→•id ,2	F→id(A•) ,0			
		N→•id,N ,2				
		F→id(A•) ,0				

Shift on “id”

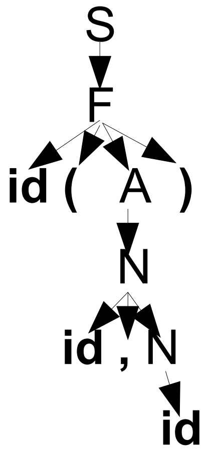
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	
		A→• ,2	A→N• ,2	N→•id,N ,4	N→id,N• ,2	
		N→•id ,2	F→id(A•) ,0			
		N→•id,N ,2				
		F→id(A•) ,0				

Reduce on N

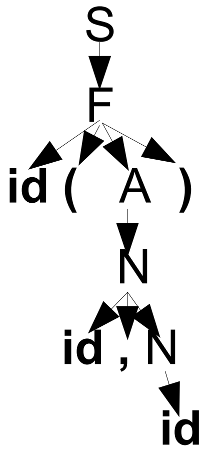
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	
		A→• ,2	A→N• ,2	N→•id,N ,4	N→id,N• ,2	
		N→•id ,2	F→id(A•) ,0		A→N• ,2	
		N→•id,N ,2				
		F→id(A•) ,0				

Reduce on N

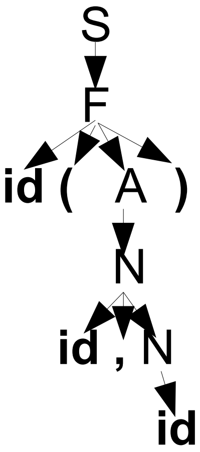
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	
		A→• ,2	A→N• ,2	N→•id,N ,4	N→id,N• ,2	
		N→•id ,2	F→id(A•) ,0		A→N• ,2	
		N→•id,N ,2			F→id(A•) ,0	
		F→id(A•) ,0				

Reduce on A

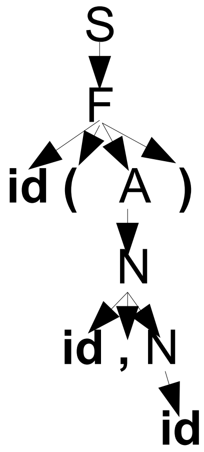
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	F→id(A)• ,0
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	
		A→• ,2	A→N• ,2	N→•id,N ,4	N→id,N• ,2	
		N→•id ,2	F→id(A•) ,0		A→N• ,2	
		N→•id,N ,2			F→id(A•) ,0	
		F→id(A•) ,0				

Shift on ")"

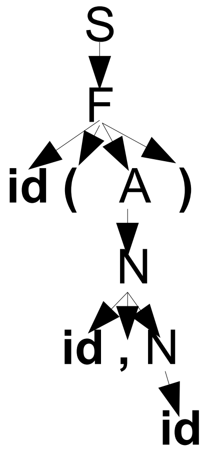
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	F→id(A)• ,0
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	S→F• ,0
		A→• ,2	A→N• ,2	N→•id,N ,4	N→id,N• ,2	
		N→•id ,2	F→id(A•) ,0		A→N• ,2	
		N→•id,N ,2			F→id(A•) ,0	
		F→id(A•) ,0				

Reduce on F

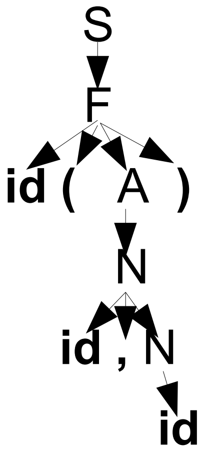
Massive Earley Example

Grammar

S → F
F → id (A)
A → N
A → ε
N → id
N → id , N

Input

id (id , id)



id (id , id)

chart[0]	chart[1]	chart[2]	chart[3]	chart[4]	chart[5]	chart[6]
S→•F ,0	F→id•(A) ,0	F→id(•A) ,0	N→id• ,2	N→id,•N ,2	N→id• ,4	F→id(A)• ,0
F→•id(A) ,0		A→•N ,2	N→id•,N ,2	N→•id ,4	N→id•,N ,4	S→F• ,0
		A→• ,2	A→N• ,2	N→•id,N ,4	N→id,N• ,2	
		N→•id ,2	F→id(A•) ,0		A→N• ,2	
		N→•id,N ,2			F→id(A•) ,0	
		F→id(A•) ,0				

Accept!

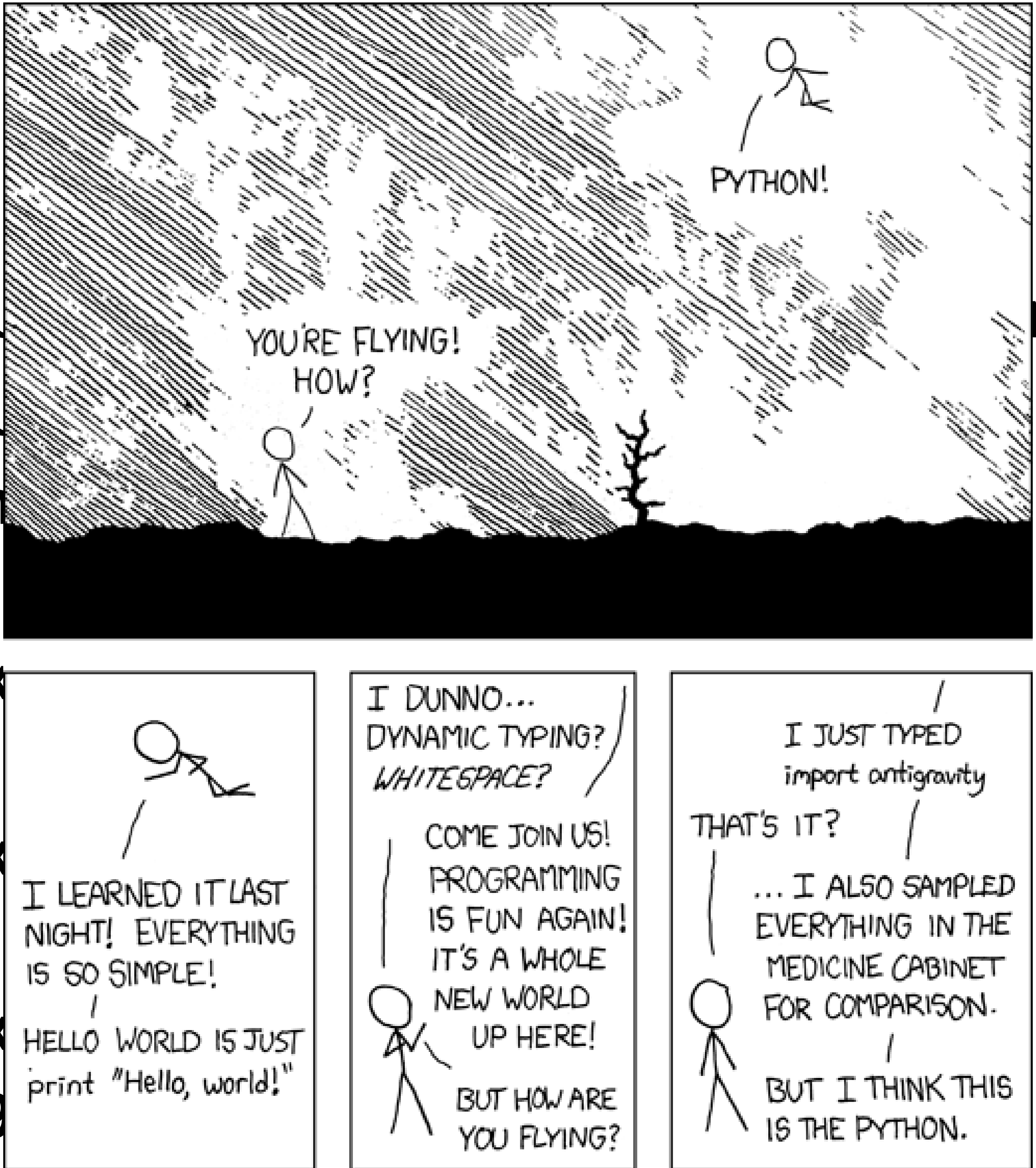
Let's Implement It

- We'll use Python and Functional Programming
- Recall: List Comprehensions

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [ x*x for x in range(10) ]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> [ x for x in range(10) if x > 5 ]
[6, 7, 8, 9]
>>> [ x*x for x in range(10) if x > 5 ]
[36, 49, 64, 81]
```

- We'll u mming
- Recall

```
>>> ran
[0, 1,
>>> [ x
[0, 1,                                              ]
>>> [ x                                              ]
[6, 7,
>>> [ x                                          5 ]
[36, 49
```


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Data Structure Decisions

- For brevity, we'll use Lists and Tuples.
    - Named Tuples in Python 3, Classes, etc

```
grammar = [
  ("S", ["F"]),
  ("F", ["id", "(", "A", ")"]),
  ("A", [ ] ),
  ("A", ["N"] ),
  ("N", ["id", ]),
  ("N", ["id", ",", "N" ]),
]
tokens = [ "id" , "(" , "id", "," , "id", ")" ]
# X→ab.cd, i == (“X”,[“a”,”b”],[“c”,”d”],i)
```

Initialization

```
# By convention, the starting rule is
# the first rule in the grammar.
start_rule = grammar[0]

# The starting parse state is "S -> . abcd , from 0"
start_state = (start_rule[0], [], start_rule[1], 0)

# The parsing chart is a one-dimensional array,
# initially empty.
chart = {}
for i in range(len(tokens)+1):  chart[i] = [ ]

# Start by placing the starting state in chart[0].
chart[0] = [ start_state ]
```

Shift

```
# If chart[i] contains "X -> ab.cd , from j"
# and c is token[i] then add:
# "X -> abc.d , from j" to chart[i+1]
def shift(tokens, i, x, ab, cd, j):
  if cd <> [] and tokens[i] == cd[0]:
    c = cd[0]
    d = cd[1:]
    abc = ab + [c]
    new_chart_state = (x, abc, d, j)
    new_chart_index = i + 1
    return [(new_chart_index, new_chart_state)]
  else:
    return []
```

Closure

```
# If chart[i] contains "X -> ab.cd , from j":
#       and cd is not empty
#       and c is a non-terminal
#       and there is a grammar rule "c -> pqr"
# Then add:
#       "c -> . pqr , from i"
#       to chart[i]
def closure(grammar,i,x,ab,cd,j):
  return [ (i , (rule[0],[],rule[1],i)) \
        for rule in grammar \
        if cd <> [] and cd[0] == rule[0] ]
```

Reduction

```
# If chart[i] contains "X -> ab. , from j"
#  (that is: cd is empty)
#  and chart[j] contains "Y -> pq.Xr , from k"
# Then add
#  "Y -> pqX.r , from k" to chart[i]
def reduction(chart,i,x,ab,cd,j):
  return [ (i, (jstate[0], jstate[1] + [x],
                (jstate[2])[1:], jstate[3] ))
    for jstate in chart[j]
    if cd == [] and jstate[2] <> []
                and (jstate[2])[0] == x ]
```

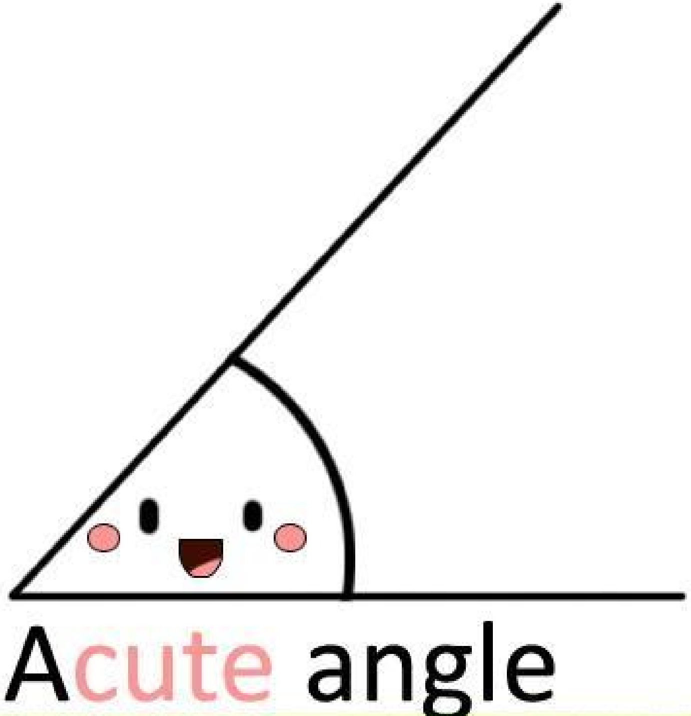
Main Loop

```
# Step 2: Dynamic Programming
for i in range(len(tokens)):
  # Apply shift, closure and reduction until
  # no new parsing states are added to the chart.
  def apply_shift_closure_reduction():
    if any([add_to_chart(chart,
             shift(tokens,i,x,ab,cd,j) +
             closure(grammar,i,x,ab,cd,j) +
             reduction(chart,i,x,ab,cd,j))
            for x, ab, cd, j in chart[i] ]):
      apply_shift_closure_reduction()
      # do it again if any changes

  apply_shift_closure_reduction()
```

Example

```
grammar3 = [
  ("S", ["E"]),
  ("E", ["E", "-", "E" ]),
  ("E", ["E", "+", "E" ]),
  ("E", ["(", "E", ")" ]),
  ("E", ["int"]),
]
tokens3 = [ "int", "-", "int" ]
chart[0]
  S ->  . E             , from 0
  E ->  . E - E         , from 0
  E ->  . E + E         , from 0
  E ->  . ( E )         , from 0
  E ->  . int           , from 0
chart[1]
  E -> int .            , from 0
  S -> E .              , from 0
...
String Accepted: True
```



PA3 in JavaScript: parser.jison

```
%token PLUS MINUS INT
%left PLUS MINUS
%start program

%%
program: exp EOF { return $1; }
       ;

exp: exp PLUS exp  { $$ = ["plus_node", $1, $3]; }
   | exp MINUS exp { $$ = ["minus_node", $1,$3]; }
   | INT           { $$ = ["int_node",
                           Number(yytext) ]; }
   ;
```

PA3 in JavaScript: main.js

```
var cl_lex = [
  ['INT', "11"] ,
  ['PLUS' ] ,
  ['INT', "22"] ,
  ['MINUS' ] ,
  ['INT', "33"] ,
  ['EOF' ] ,
]
var token_count = 0

var parser =
    require("./parser").parser;
```

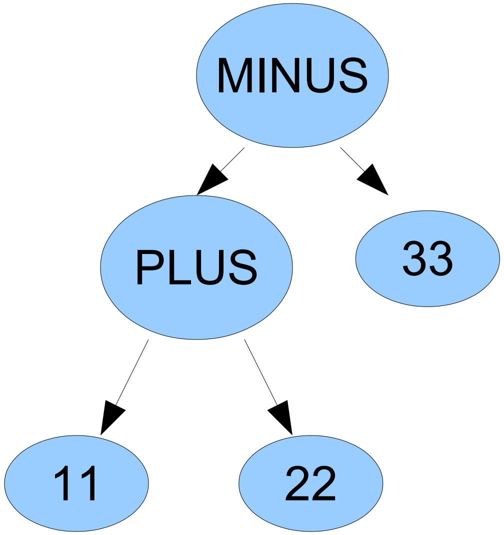
```
parser.lexer = {
  lex : function() {
    var cl_lex_entry =
        cl_lex[token_count++] ;
    var token = cl_lex_entry[0] ;
    var lexeme = cl_lex_entry[1] ;
    parser.lexer.yytext = lexeme ;
    return token;
  },
  setInput : function(str) { }
}

var final_ast = parser.parse("");

console.log(final_ast);
```

PA3 in JavaScript Output:

```
$ node main.js
[ 'minus_node',
    [ 'plus_node',
        [ 'int_node', 11 ],
        [ 'int_node', 22 ]
    ],
    [ 'int_node', 33 ]
]
```



PA3 Not Shown Here

- Reading in the .cl-lex file
- Handling line number information
- Printing out the AST in the desired format
- Adding parsing rules for whole classes and not just simple expressions
- Massive testing effort
  - diff vs. “cool --parse” requires “almost done”
- Dealing with ambiguity (“conflicts”)
  - Let's do this one now.

Conflicts

- Add “%token NEG” and “exp: NEG exp”.
- Oh noes:

```
Conflict in grammar: multiple actions possible when lookahead token is
PLUS in state 8
- reduce by rule: exp -> NEG exp
- shift token (then go to state 6)
Conflict in grammar: multiple actions possible when lookahead token is
MINUS in state 8
- reduce by rule: exp -> NEG exp
- shift token (then go to state 7)

States with conflicts:
State 8
  exp -> NEG exp .          #lookaheads= EOF PLUS MINUS
  exp -> exp . PLUS exp
  exp -> exp . MINUS exp
```

Con

- Add "%token NEG" a
- Oh noes:

```
Conflict in grammar: multiple acti
PLUS in state 8
- reduce by rule: exp -> NEG exp
- shift token (then go to state 6)
Conflict in grammar: multiple acti
MINUS in state 8
- reduce by rule: exp -> NEG exp
- shift token (then go to state 7)

States with conflicts:
State 8
  exp -> NEG exp .          #lookah
  exp -> exp . PLUS exp
  exp -> exp . MINUS exp
```

Conflict Interpretation

- So some table entry has all three:
  - exp → NEG exp .
  - exp → exp . PLUS exp
  - exp → exp . MINUS exp
- What would the input have to look like to get to that table entry?

Internet Explorer
Question of the day: Which technological invention do you think has impacted our lives more - the telephone or the internet?
about a minute ago · Like · Comment

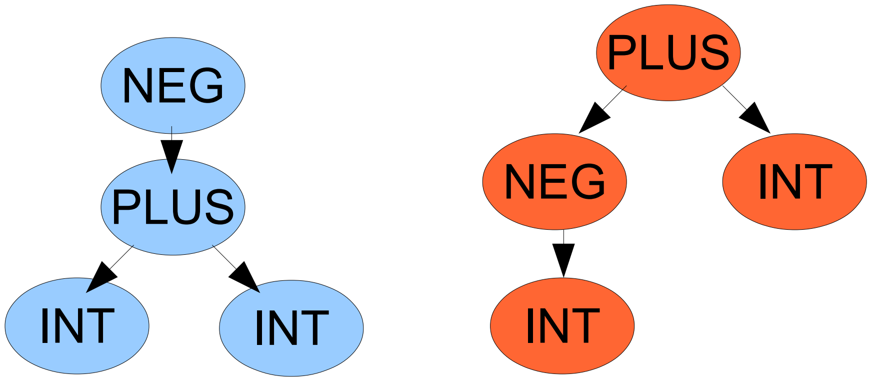
Billy [redacted] You know you can post Polls on facebook now, right? IE, Always a little behind the times.
2 seconds ago · Like

Conflict Interpretation

- So some table entry has all three:
  - exp → NEG exp .
  - exp → exp . PLUS exp
  - exp → exp . MINUS exp
- What would the input have to look like to get to that table entry?
  - NEG INT . PLUS INT

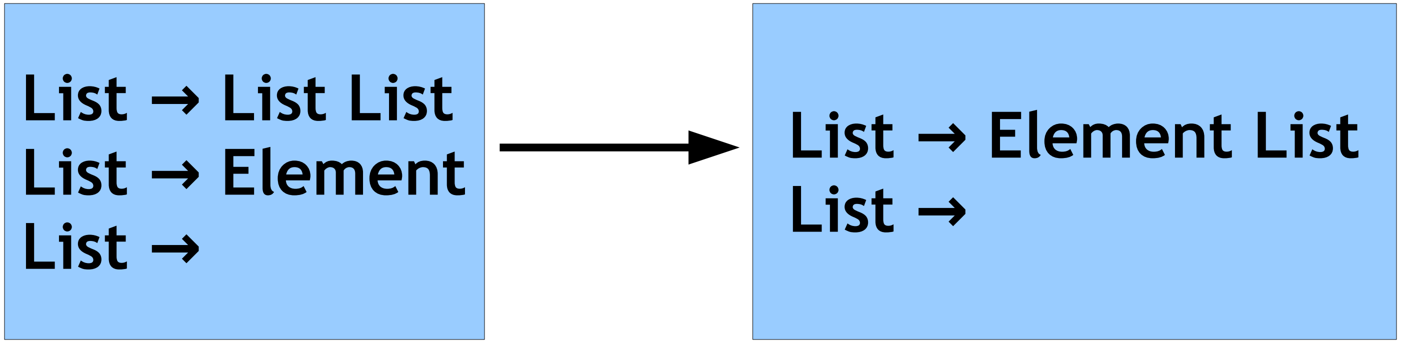
Conflict Interpretation

- So some table entry has all three:
  - exp → NEG exp .
  - exp → exp . PLUS exp
  - exp → exp . MINUS exp
- What would the input have to look like to get to that table entry?
  - NEG INT . PLUS INT



Conflict Solution

- Shift/Reduce
  - Carefully specify precedence and associativity of operators (and sometimes of random tokens).
    - In last example, NEG has higher precedence than PLUS or MINUS.
- Reduce/Reduce
  - Rewrite grammar to avoid gross ambiguity:



List → List List
List → Element
List →

List → Element List
List →

Homework

- Midterm 1 Next Class

- PS3 recommended for next Tuesday
- PA3 due next Tuesday
- CA2 due next Tuesday