

二、语法分析

(8. Adaptive $LL(*)$ 语法分析算法)

魏恒峰

hfwei@nju.edu.cn

2023 年 04 月 07 日





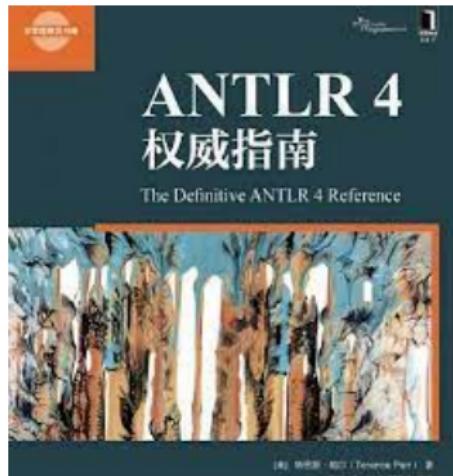
please write a 10-line poem about the Adaptive LL(*), also called ALLStar, parsing algorithm, each line consisting of at least 5 words.

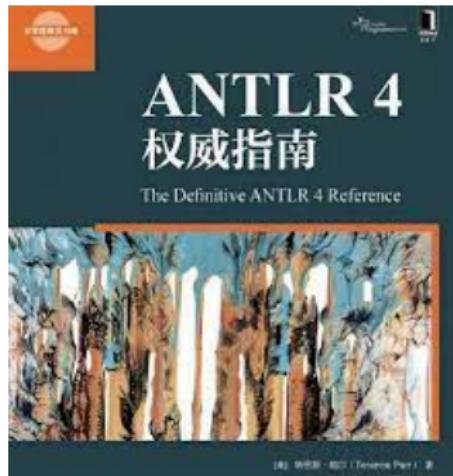


please write a 10-line poem about the Adaptive LL(*), also called ALLStar, parsing algorithm, each line consisting of at least 5 words.

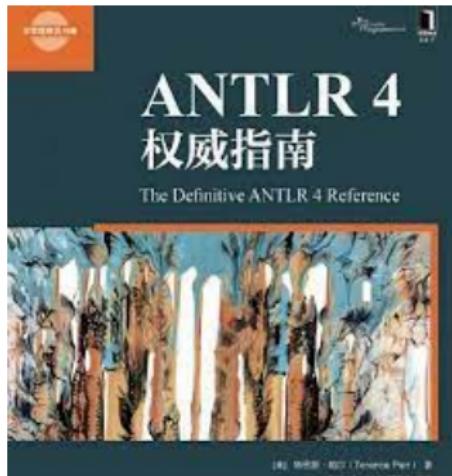


Adaptive LL star, parsing king,
Solves conflicts with precision, no guessing.
A top-down approach, fast and strong,
Ambiguous grammar? It won't be long.
Predictive parsing, a thing of the past,
ALLStar's power will forever last.
Left recursion? It's not a problem,
ALLStar's flexibility will solve them.
Adaptive LL star, a parser supreme,
In parsing theory, it reigns supreme.

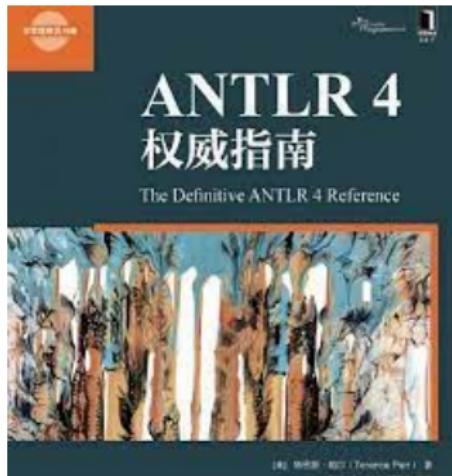




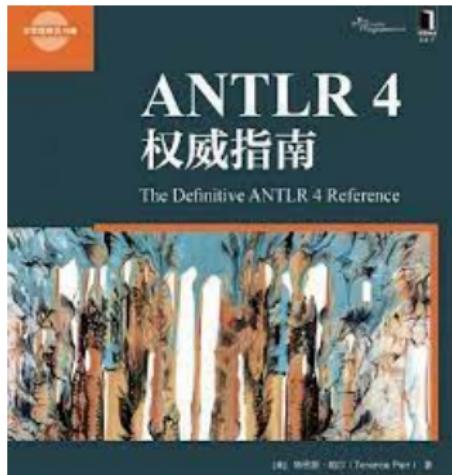
- (1) ANTLR 4 自动将类似 expr 的左递归规则重写成非左递归形式



- (1) ANTLR 4 自动将类似 expr 的**左递归**规则重写成非左递归形式
- (2) ANTLR 4 提供优秀的**错误报告**功能和复杂的**错误恢复**机制



- (1) ANTLR 4 自动将类似 expr 的**左递归**规则重写成非左递归形式
- (2) ANTLR 4 提供优秀的**错误报告**功能和复杂的**错误恢复**机制
- (3) ANTLR 4 使用了一种名为 **Adaptive LL(*)** 的新技术



- (1) ANTLR 4 自动将类似 expr 的**左递归**规则重写成非左递归形式
- (2) ANTLR 4 提供优秀的**错误报告**功能和复杂的**错误恢复**机制
- (3) ANTLR 4 使用了一种名为 **Adaptive LL(*)** 的新技术
- (4) ANTLR 4 几乎能处理**任何文法** (二义性文法✓ 间接左递归✗)

(1995 2011 2014)

ANTLR: A Predicated- $LL(k)$ Parser Generator

T. J. PARR

*University of Minnesota, AHP CRC, 1100 Washington Ave S Ste 101, Minneapolis, MN 55415, U.S.A.
(email: parrt@acm.org)*

AND

R. W. QUONG

*School of Electrical Engineering, Purdue University, W. Lafayette, IN 47907, U.S.A.
(email: quong@ecn.purdue.edu)*

$LL(*)$: The Foundation of the ANTLR Parser Generator

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Kathleen Fisher *

Tufts University
kfisher@eecs.tufts.edu

Adaptive $LL(*)$ Parsing: The Power of Dynamic Analysis

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Sam Harwell

University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher

Tufts University
kfisher@eecs.tufts.edu

courses-at-nju-by-hfwei/compilers-papers-we-love

ANTLR 4 是如何处理**直接左递归与优先级**的?

parser-allstar/LRExpr.g4

```
stat : expr ';' EOF;
```

```
expr : expr '*' expr  
      | expr '+' expr  
      | INT  
      | ID  
      ;
```

parser-allstar/LRExpr.g4

```
stat : expr ';' EOF;
```

```
expr : expr '*' expr  
      | expr '+' expr  
      | INT  
      | ID  
      ;
```

```
antlr4 LRExpr -Xlog (.log)
```

2021-11-25 17:44:23:815 left-recursion LogManager.java:25 expr

```
: ( {} INT<tokenIndex=45>
  | ID<tokenIndex=51>
  )
(
  {precpred(_ctx, 4)}?<p=4> '*'<tokenIndex=27> expr<tokenIndex=29, p=5>
  | {precpred(_ctx, 3)}?<p=3> '+'<tokenIndex=37> expr<tokenIndex=39, p=4>
)*
;
```

```
2021-11-25 17:44:23:815 left-recursion LogManager.java:25 expr
```

```
: ( {} INT<tokenIndex=45>
  | ID<tokenIndex=51>
  )
(
  {precpred(_ctx, 4)}?<p=4> '*'<tokenIndex=27> expr<tokenIndex=29,p=5>
  | {precpred(_ctx, 3)}?<p=3> '+'<tokenIndex=37> expr<tokenIndex=39,p=4>
)*
;
```

```
stat : expr ';' EOF;
```

```
expr : expr '*' expr
      | expr '+' expr
      | INT
      | ID
;
```

```
expr[int _p]
: ( INT
  | ID
)
( {4 >= $_p}? '*' expr[5]
 | {3 >= $_p}? '+' expr[4]
)*
;
```

expr[int _p]

stat : expr ';' EOF;

```
expr : expr '*' expr
      | expr '+' expr
      | INT
      | ID
;
```

对应于一段递归函数 expr(int _p)

```
expr[int _p]
: ( INT
  | ID
  )
( {4 >= $_p}? '*' expr[5]
 | {3 >= $_p}? '+' expr[4]
)*
;
```

1 + 2 + 3

1 + 2 * 3

1 * 2 + 3

根本问题:

究竟是在 `expr` 的**当前调用**中匹配下一个运算符，
还是让 `expr` 的**调用者**匹配下一个运算符。

parser-allstar/LRExprParen.g4

```
stat : expr ';' EOF;  
  
expr : expr '*' expr  
      | expr '+' expr  
      | '(' expr ')' // This rule is highlighted with a red box.  
      | INT  
      | ID  
      ;
```

parser-allstar/LRE ExprParen.g4

```
stat : expr ';' EOF;
expr : expr '*' expr
      | expr '+' expr
      | '(' expr ')'
      | INT
      | ID
      ;
expr[int _p]
: (
  '(' expr[0] ')'
  | INT
  | ID
  )
  ( {5 >= $_p}? '*' expr[6]
  | {4 >= $_p}? '+' expr[5]
  )*
```

parser-allstar/LREExprUS.g4

```
stat : expr ';' EOF;
```

```
expr : '-' expr  
      | expr '!'  
      | expr '+' expr  
      | ID  
      ;
```

```

expr[int _p]
: (
    ID
    | '-' expr[4]
)
( {3 >= $_p}? '!'
| {2 >= $_p}? '+' expr[3]
)*
;

```

$$-a!! \quad -a + b!$$

```
stat : expr ';' EOF ;
expr : <assoc = right> expr '^' expr
      | expr '+' expr
      | INT
      ;
```

```
stat : expr ';' EOF ;
expr : <assoc = right> expr '^' expr
      | expr '+' expr
      | INT
      ;
```

```
expr[int _p]
: ( INT )
( [3] >= $_p)? '^' expr[3]
| [2] >= $_p)? '+' expr[3]
)*
;
```

$$1^2^3 + 4$$

For *left-associative* operators, the right operand gets **one more** precedence level than the operator itself.

Adaptive $LL(*)$ Parsing: The Power of Dynamic Analysis

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Sam Harwell

University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher

Tufts University
kfisher@eecs.tufts.edu

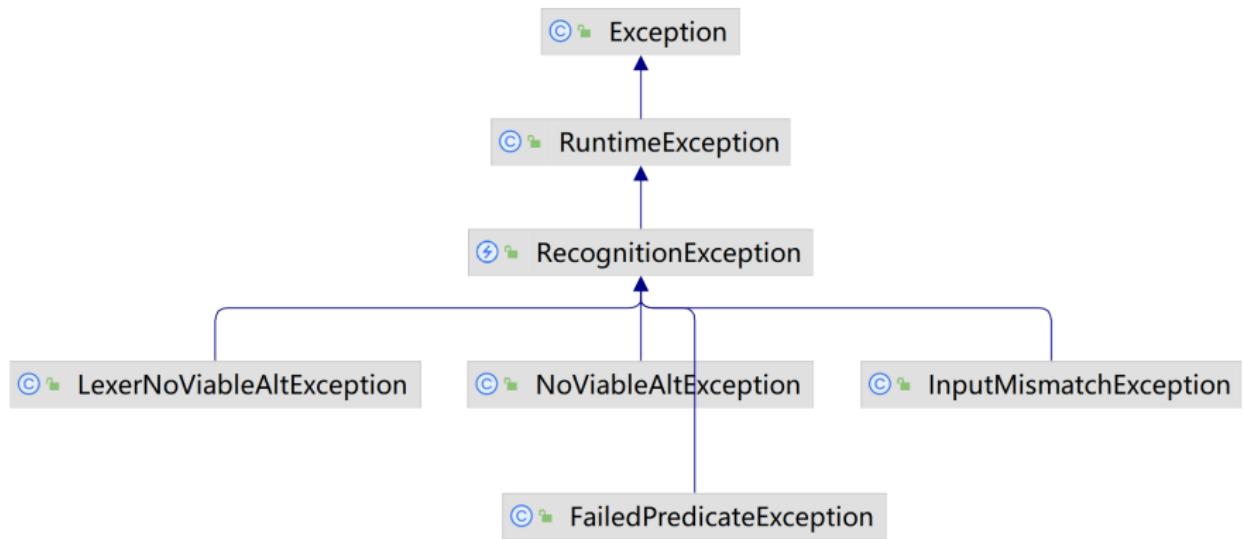
Appendix C: Left-recursion Elimination

For *right-associative* operators, the right operand gets **the same** precedence level as the current operand.

ANTLR 4 是如何进行**错误报告与恢复**的?



共四类词法、语法错误



NoViableAltException

InputMismatchException

Class.g4

LexerNoViableAltException

Class.g4 start rule: prog

Input ror/Class-error-LexerNoViableAltException.txt

```
1 class # { int i; }
```

line 1:6 token recognition error at: '#'
line 1:8 missing ID at '{'

Parse tree Hierarchy Profiler

```
graph TD; prog --> classDef; classDef --> CLASS; classDef --> ID["ID:<missing ID>"]; classDef --> LBRACE["LBRACE:{"]; classDef --> member1["member:1"]; member1 --> INTTYPE["INTTYPE:int"]; member1 --> IDi["ID:i"]; member1 --> SEMI["SEMI;"]
```

遇到未知字符，出现词法错误

InputMismatchException

Class.g4 start rule: prog

Input ar/error/Class-error-InputMismatchException

```
1 class T ; { int i; }
```

line 1:8 extraneous input ';' expecting '{'

Parse tree Hierarchy Profiler

```
graph TD; prog --> classDef; classDef --> CLASS; classDef --> IDT[ID:T]; classDef --> SEMI1[SEMI;]; classDef --> LBRACE[LBRACE:{]; LBRACE --> INTTYPE[int]; LBRACE --> IDI[i]; LBRACE --> SEMI2[SEMI;];
```

输入流中的当前词法单元与当前规则所期望的词法单元不匹配

NoViableAltException

Class.g4 start rule: prog

Input File ar/error/Class-error-NoViableAltException.txt

```
1 class T { int ; }
```

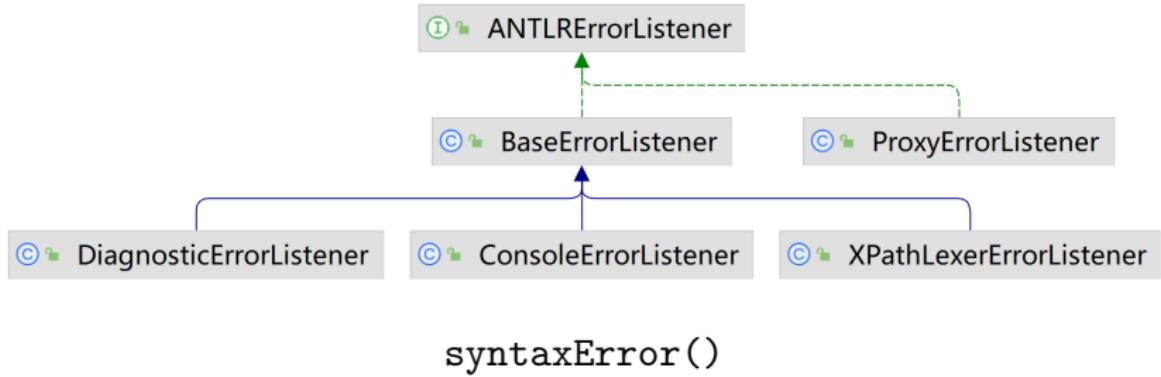
line 1:14 no viable alternative at input 'int';

Parse tree Hierarchy Profiler

```
graph TD; prog --> classDef; classDef --> CLASS; classDef --> IDT["ID:T"]; classDef --> LBRACE["LBRACE:{"]; classDef --> member1["member:1"]; member1 --> INTTYPE["INTTYPE:int"]; member1 --> SEMI["SEMI;"];
```

剩余输入不符合当前规则的任何一个备选分支

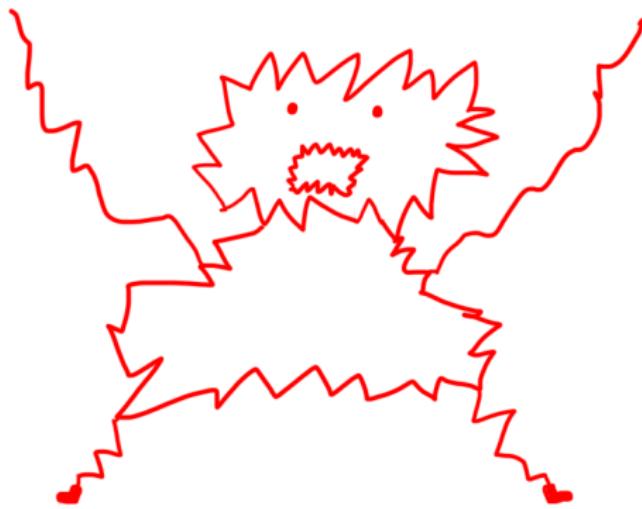
自定义错误报告消息



《ANTLR 4 权威指南》第 9.2 节：修改和转发 ANTLR 的错误消息



报错、**恢复**、继续分析



恐慌/应急 (Panic) 模式：假装成功、调整状态、继续进行

四项基本原则:

- (1) 特殊情况, 特殊处理
- (2) 一般情况, 统一处理
- (3) 统一处理, 精细控制
- (4) 自定义错误处理策略

(1) 特殊情况, 特殊处理

如果下一个词法单元符合预期,
则采用“**单词法符号移除** (single-token deletion)”
或“**单词法符号补全** (single-token insertion)”策略

单词法符号移除

A Class.g4 start rule: prog

Input File antlr\parser\allstar\error\Class-DeleteToken.txt

```
1 class 9 T { int i; }
```

line 1:6 extraneous input '9' expecting ID

Parse tree Hierarchy Profiler

```
graph TD; prog --> classDef; classDef --> CLASS; classDef --> INT9[INT:9]; classDef --> IDT[ID:T]; classDef --> LBRACE; classDef --> member1["member:1"]; member1 --> INTTYPE[int]; member1 --> IDi[i]; member1 --> SEMI[SEMI:;]
```

单词法符号补全

Class.g4 start rule: prog

Input File 1\antlr\parser.allstar\error\Class-AddToken.txt

```
1 class { int i; }
```

line 1:6 missing ID at '{'

Parse tree Hierarchy Profiler

```
graph TD; prog --> classDef; classDef --> CLASS; classDef --> ID["ID:<missing-ID>"]; classDef --> LBRACE["LBRACE:{"]; classDef --> member1["member:1"]; member1 --> INTTYPE["INTTYPE:int"]; member1 --> IDi["ID:i"]; member1 --> SEMI["SEMI:;"]
```

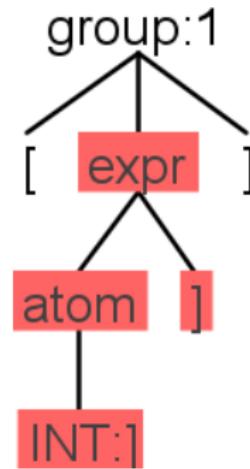
(2) 一般情况, 统一处理

采用“**同步-返回 (sync-and-return)**”策略,
使用“重新同步集合 (resynchronization set)”从当前规则中恢复

$\text{FOLLOWING}(\{\text{expr}, \text{atom}\}) = \{^,]\}$

$\text{FOLLOWING}(\{\text{expr}\}) = []$

```
9 group : '[' expr ']'
10   | '(' expr ')'
11   ;
12
13 expr : atom '^' INT ;
14
15 atom : ID
16   | INT
17   ;
```



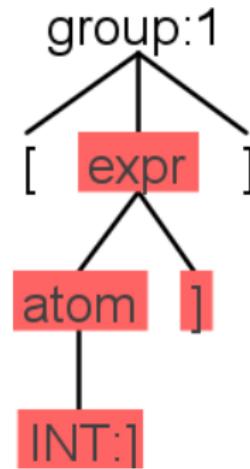
Group.g4

[]

$\text{FOLLOWING}(\{\text{expr}, \text{atom}\}) = \{^,]\}$

$\text{FOLLOWING}(\{\text{expr}\}) = []$

```
9 group : '[' expr ']'  
10   | '(' expr ')' ;  
11 ;  
12  
13 expr : atom '^' INT ;  
14  
15 atom : ID  
16   | INT  
17 ;
```



Group.g4

[]

注意 FOLLOW (静态) 集合与 FOLLOWING (动态) 集合的区别

(3) 统一处理，精细控制

如何从子规则中优雅地恢复出来？

Class.g4 (`member+`)

Class-Subrule-Start.txt (“单词法符号移除”）

Class-Subrule-Loop.txt (“另一次 `member` 迭代”）

Class-Subrule-End.txt (“退出当前 `classDef` 规则”）

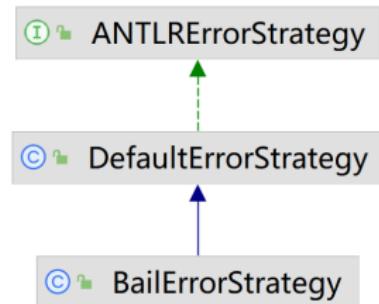
(4) 自定义错误处理策略

比如, (已知语法正确) 关闭默认错误处理功能, 提高运行效率

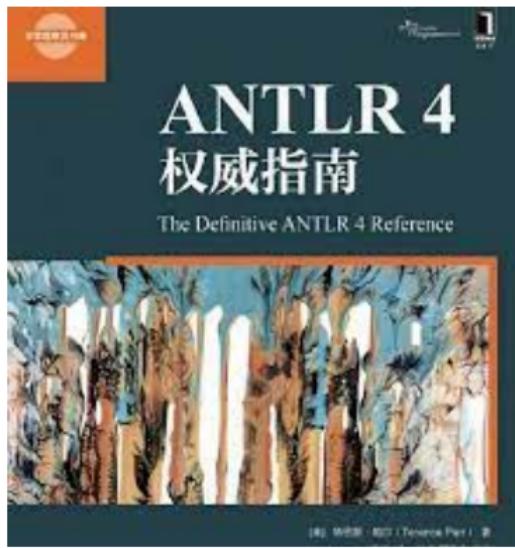
比如, (出错代价太大) 在遇到第一个语法错误时, 就停止分析

(4) 自定义错误处理策略

比如, (已知语法正确) 关闭默认错误处理功能, 提高运行效率



比如, (出错代价太大) 在遇到第一个语法错误时, 就停止分析



第 9 章：错误报告与恢复



Adaptive $LL(*)$ Parsing: The Power of Dynamic Analysis

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Sam Harwell

University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher

Tufts University
kfisher@eecs.tufts.edu

$P = \{S \rightarrow Ac \mid Ad, A \rightarrow aA \mid b\}$

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$

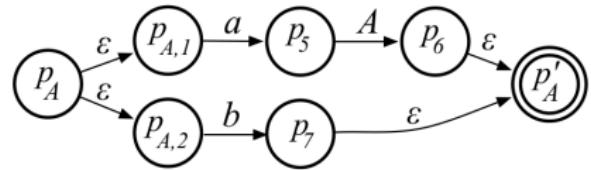
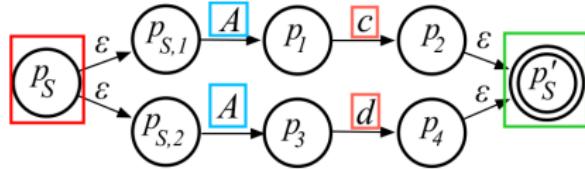
bc vs. bd

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$

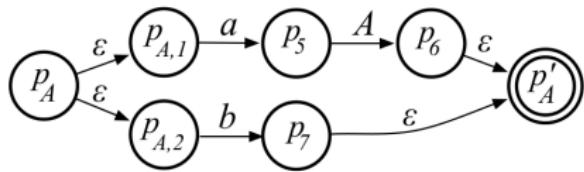
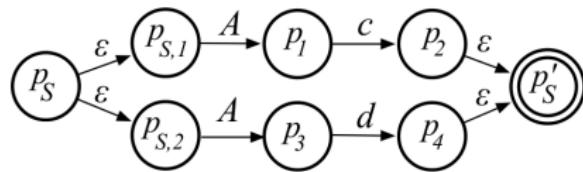
bc vs. bd

不是 $LL(1)$ 文法，也不是 $LL(k)$ 文法 ($\forall k \geq 1$)

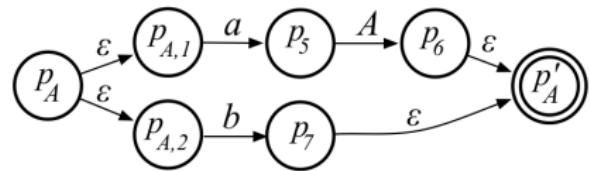
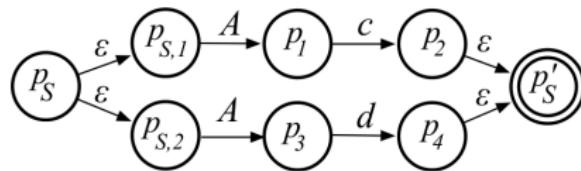
$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



ATN: Augmented Transition Network

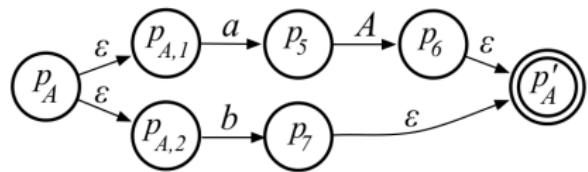
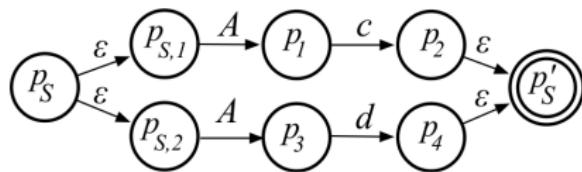


Incrementally and dynamically build up a **lookahead DFA** that **map lookahead phrases to predicated productions.**



Incrementally and dynamically build up a **lookahead DFA** that **map lookahead phrases to predicated productions.**

Upon *bc* and then *bd*



Incrementally and dynamically build up a **lookahead DFA** that **map lookahead phrases to predicated productions**.

Upon *bc* and then *bd*

D_0 (p_{S,1}, 1, []), (p_A, 1, p_1), (p_{A,1}, 1, p_1), (p_{A,2}, 1, p_1)
(p_{S,2}, 2, []), (p_A, 2, p_3), (p_{A,1}, 2, p_3), (p_{A,2}, 2, p_3)

b

D' (p_7, 1, p_1), (p'_A, 1, p_1), (p_1, 1, [])
(p_7, 2, p_3), (p'_A, 2, p_3), (p_3, 2, [])

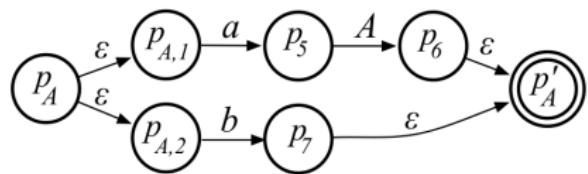
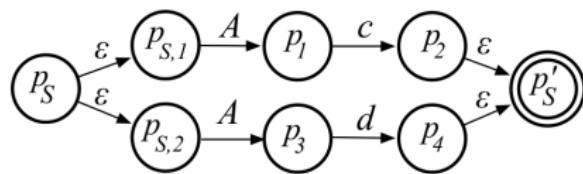
c

f_1 (p_2, 1, []), (p'_S, 1, [])

d

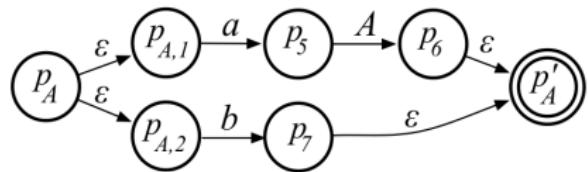
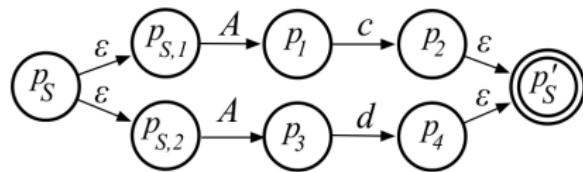
f_2 (p_4, 2, []), (p'_S, 2, [])

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



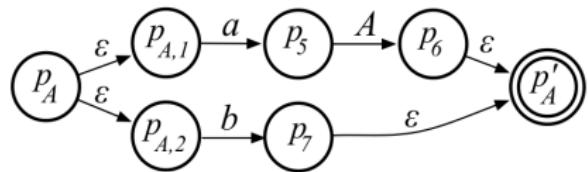
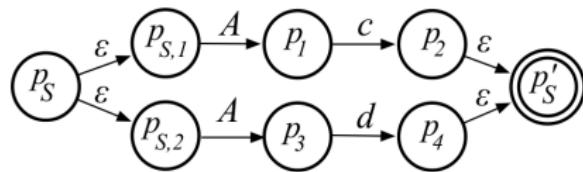
- ▶ Launch subparsers at a decision point, one per alternative productions.

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



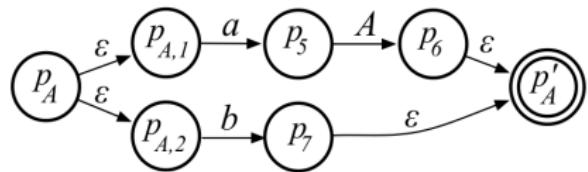
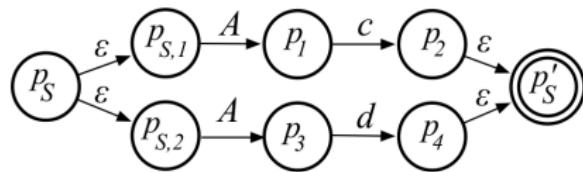
- ▶ Launch subparsers at a decision point, one per alternative productions.
- ▶ These subparsers run in pseudo-parallel to explore all possible paths.

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



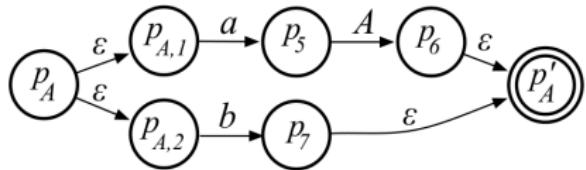
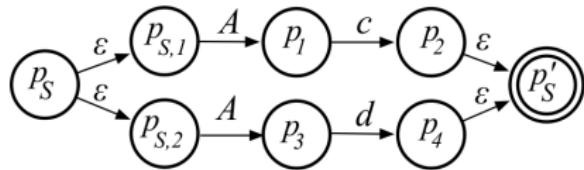
- ▶ Launch subparsers at a decision point, one per alternative productions.
- ▶ These subparsers run in pseudo-parallel to explore all possible paths.
- ▶ Subparsers die off as their paths fail to match the remaining input.

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



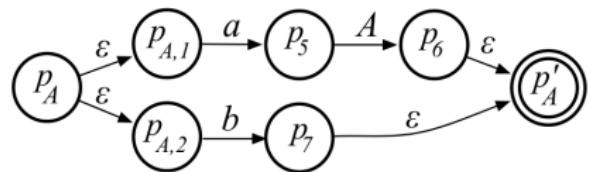
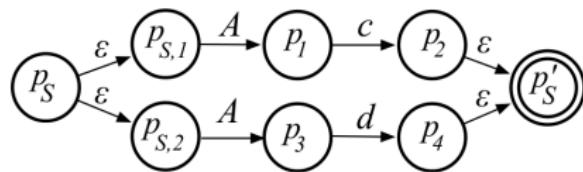
- ▶ Launch subparsers at a decision point, one per alternative productions.
- ▶ These subparsers run in pseudo-parallel to explore all possible paths.
- ▶ Subparsers die off as their paths fail to match the remaining input.
- ▶ Ambiguity: Multiple subparsers coalesce together or reach EOF.

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$

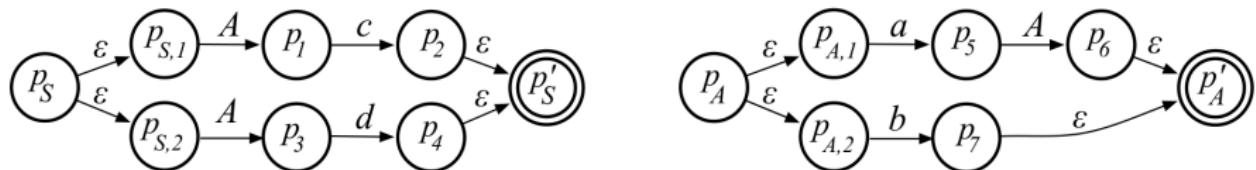


- ▶ Launch subparsers at a decision point, one per alternative productions.
- ▶ These subparsers run in pseudo-parallel to explore all possible paths.
- ▶ Subparsers die off as their paths fail to match the remaining input.
- ▶ Ambiguity: Multiple subparsers coalesce together or reach EOF.
- ▶ Resolution: The first production associated with a surviving subparser.

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



D_0

$(\mathbf{p}_{S,1}, \mathbf{1}, []), (p_A, 1, p_1), (p_{A,1}, 1, p_1), (p_{A,2}, 1, p_1)$
$(\mathbf{p}_{S,2}, \mathbf{2}, []), (p_A, 2, p_3), (p_{A,1}, 2, p_3), (p_{A,2}, 2, p_3)$

b

D'

$(\mathbf{p}_7, \mathbf{1}, \mathbf{p}_1), (p'_A, 1, p_1), (p_1, 1, [])$
$(\mathbf{p}_7, \mathbf{2}, \mathbf{p}_3), (p'_A, 2, p_3), (p_3, 2, [])$

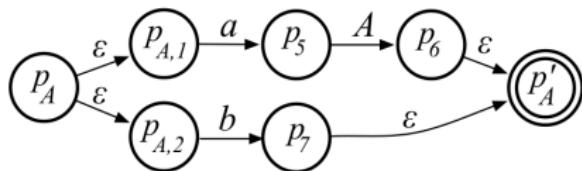
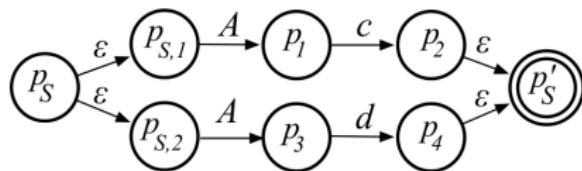
c

d

f_1 $(\mathbf{p}_2, \mathbf{1}, []), (p'_S, 1, [])$ $(\mathbf{p}_4, \mathbf{2}, []), (p'_S, 2, [])$ f_2

Upon bc and then bd

$$P = \{S \rightarrow Ac \mid Ad, \quad A \rightarrow aA \mid b\}$$



D_0

$(\mathbf{p}_{S,1}, \mathbf{1}, []), (p_A, 1, p_1), (p_{A,1}, 1, p_1), (p_{A,2}, 1, p_1)$
$(\mathbf{p}_{S,2}, \mathbf{2}, []), (p_A, 2, p_3), (p_{A,1}, 2, p_3), (p_{A,2}, 2, p_3)$

b

D'

$(\mathbf{p}_7, \mathbf{1}, \mathbf{p}_1), (p'_A, 1, p_1), (p_1, 1, [])$
$(\mathbf{p}_7, \mathbf{2}, \mathbf{p}_3), (p'_A, 2, p_3), (p_3, 2, [])$

c

d

f_1 $(\mathbf{p}_2, \mathbf{1}, []), (p'_S, 1, [])$ $(\mathbf{p}_4, \mathbf{2}, []), (p'_S, 2, [])$ f_2

Upon bc and then bd

Move on terminals and Closure over ϵ and non-terminals

Adaptive $LL(*)$ Parsing: The Power of Dynamic Analysis

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Sam Harwell

University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher

Tufts University
kfisher@eecs.tufts.edu

Adaptive $LL(*)$ Parsing: The Power of Dynamic Analysis

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Sam Harwell

University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher

Tufts University
kfisher@eecs.tufts.edu

paper @ compilers-papers-we-love

Adaptive $LL(*)$ Parsing: The Power of Dynamic Analysis

Terence Parr

University of San Francisco
parrt@cs.usfca.edu

Sam Harwell

University of Texas at Austin
samharwell@utexas.edu

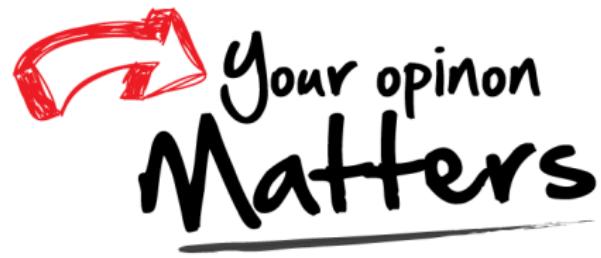
Kathleen Fisher

Tufts University
kfisher@eecs.tufts.edu

paper @ compilers-papers-we-love



Thank You!



Office 926

hfwei@nju.edu.cn