

语法分析

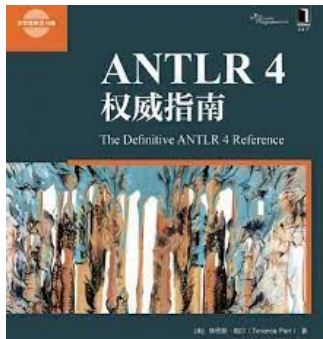
(3. Adaptive $LL(*)$ 语法分析算法)

魏恒峰

hfwei@nju.edu.cn

2022 年 11 月 30 日





- (1) ANTLR 4 自动将类似 `expr` 的左递归规则重写为非左递归形式
- (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, AHPCRC, 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/papers-we-love](https://courses-at-nju-by-hfwei.com/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
    ;

```

根本原因:

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。

```
antlr4 LRExpr -Xlog
```

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
;

```



```

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

```

1 + 2 + 3 1 + 2 * 3 1 * 2 + 3

parser-allstar/LRExprParen.g4

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

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

parser-allstar/LRExprUS.g4

stat : expr ';' EOF;

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

```

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

```

$-a!!$ $-a + b!$

```

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

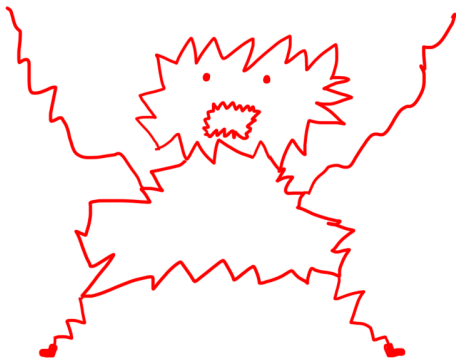
Appendix C: Left-recursion Elimination

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?



报错、恢复、继续分析

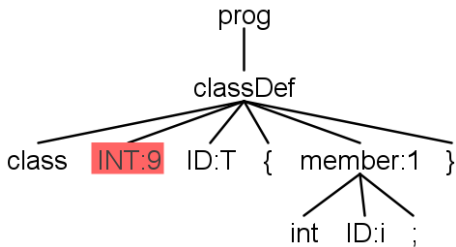


恐慌/应急 (Panic) 模式: 假装成功、调整状态、继续进行

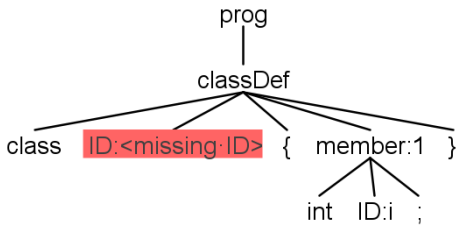
如果**下一个词法单元**符合预期,
则采用“**单词法符号移除** (single-token deletion)”
或“**单词法符号补全** (single-token insertion)” 策略

Class.g4

Class-DeleteToken.txt



Class-AddToken.txt

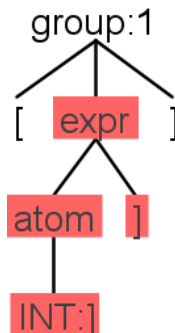
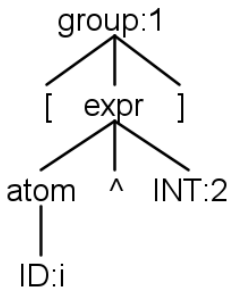


采用“**同步-返回 (sync-and-return)**”策略,
使用“**重新同步集合 (resynchronization set)**”从**当前规则**中恢复

Group.g4

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

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



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

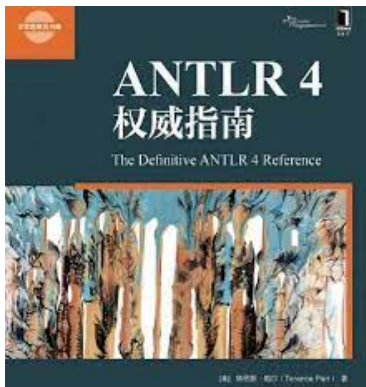
如何从子规则中优雅地恢复出来？

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”)



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

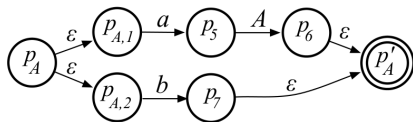
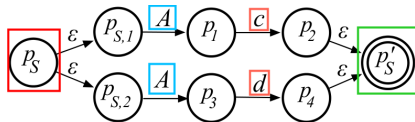
Sam Harwell
University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher
Tufts University
kfisher@eecs.tufts.edu

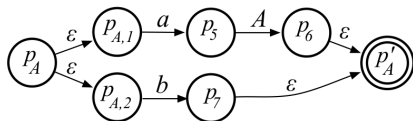
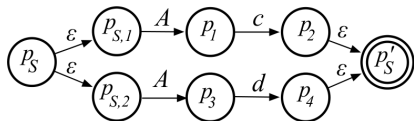
$$P = \{S \rightarrow Ac \mid Ad, A \rightarrow aA \mid b\}$$

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

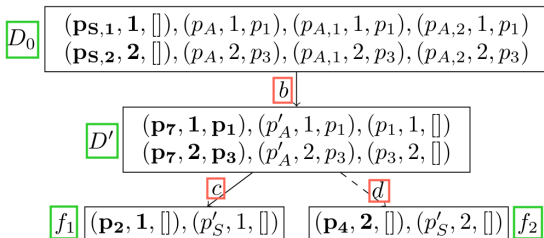
$$P = \{ S \rightarrow Ac \mid Ad, A \rightarrow aA \mid b \}$$

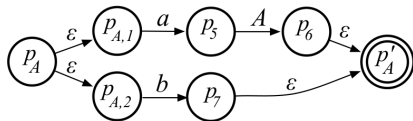
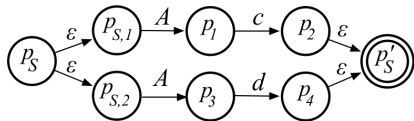


ATN: Augmented Transition Network

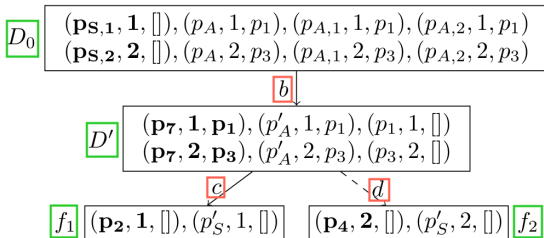
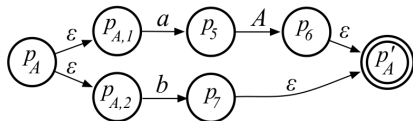
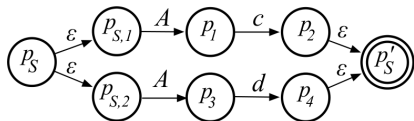


Incrementally and dynamically build up a *lookahead DFA* that map lookahead phrases to predicated productions.





- ▶ 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.



Upon bc and then bd

move-closure!!!

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

Terence Parr
University of San Francisco
parrt@cs.usfca.edu

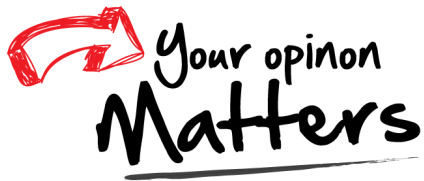
Sam Harwell
University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher
Tufts University
kfisher@eecs.tufts.edu

附加作业: paper @ compilers-papers-we-love



Thank
You!



Office 926

hfwei@nju.edu.cn