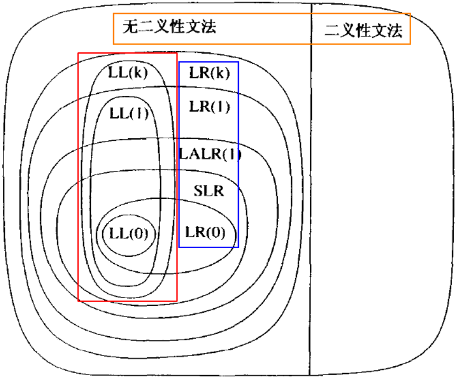
二、语法分析
(7. Adaptive $LL(*)$ 语法分析算法)

魏恒峰

hfwei@nju.edu.cn

2024 年 04 月 07 日

$LL(1)$ 语法分析算法的处理能力有限 (左递归文法, 带左公因子的文法)



ANTLR 4 采用的 Adaptive $LL(*)$ 语法分析算法功能强大

(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 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/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
     ;
```

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>
        )*
    ;
```

```
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]
      )*
    ;
```

$$1 + 2 + 3 \qquad 1 + 2 * 3 \qquad 1 * 2 + 3$$

Algorithm 1 将左递归文法改写为等价的迭代版本

```
procedure EXP(p)
    MATCH(ID | INT)
    while !EOF() do
        if 4 ≥ p then
            MATCH(∗)    EXP(5)
            continue
        if 3 ≥ p then
            MATCH(+)    EXP(4)
```

$$1 + 2 + 3 \qquad 1 + 2 * 3 \qquad 1 * 2 + 3$$

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 ')'
     | INT
     | ID
     ;
```

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

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]
      )*
    ;
```

$$-a!! \qquad \text{-}a + b!$$

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

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

$$1 \,\hat{}\, 2 \,\hat{}\, 3 + 4$$

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.

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

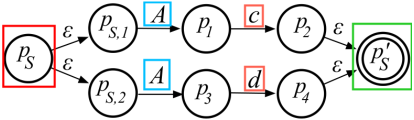
$$P=\{S \to Ac \mid Ad,\ \ A \to aA \mid b\}$$

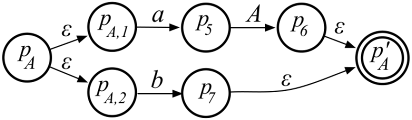
$$bc \quad vs. \quad bd$$

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

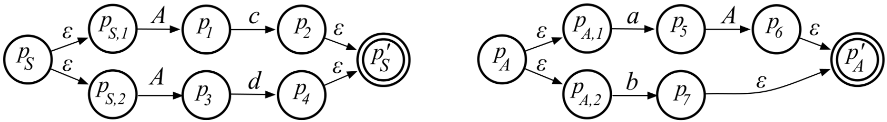
$\boxed{\text{动态分析, 而非静态分析: } \textbf{Adaptive } LL(*)}$

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



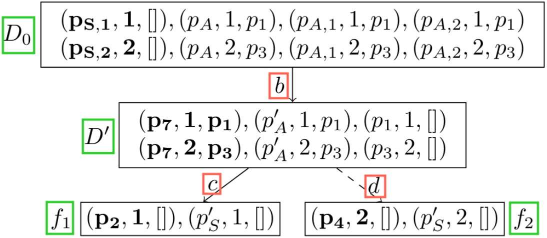


ATN: Augmented Transition Network

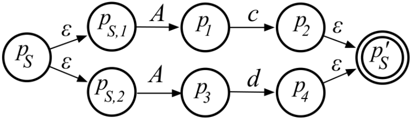


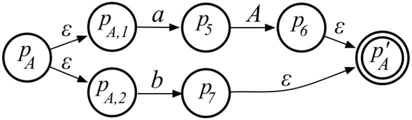
Incrementally and dynamically build up a **lookahead DFA** that **map lookahead phrases to predicated productions**.

Upon bc and then bd



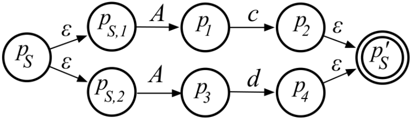
$$P = \{S \to Ac \mid Ad, \;\; A \to aA \mid b\}$$

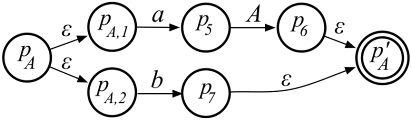




- Launch subparsers at a decision point, one per alternative productions.
- These subparsers run in pseudo-parallel to explore all possible paths.
- Subparsers die off as their paths fail to match the remaining input.
- Ambiguity: Multiple subparsers coalesce together or reach EOF.
- Resolution: The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

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





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

$\downarrow b$

D': $(\mathbf{p_7}, \mathbf{1}, \mathbf{p_1}), (p'_A, 1, p_1), (p_1, 1, [])$
$(\mathbf{p_7}, \mathbf{2}, \mathbf{p_3}), (p'_A, 2, p_3), (p_3, 2, [])$

$\swarrow c$ $\searrow d$

f_1: $(\mathbf{p_2}, \mathbf{1}, []), (p'_S, 1, [])$ f_2: $(\mathbf{p_4}, \mathbf{2}, []), (p'_S, 2, [])$

Upon bc and then bd

Move on terminals and **Closure** over ϵ and non-terminals

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

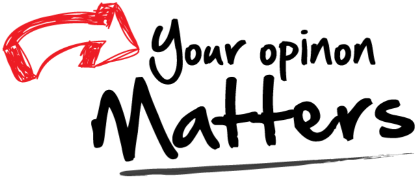
Terence Parr
University of San Francisco
parrt@cs.usfca.edu

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

Kathleen Fisher
Tufts University
kfisher@eecs.tufts.edu

paper @ compilers-papers-we-love


Thank You!



Office 926

hfwei@nju.edu.cn