Design and Implementation of Automated Grammar Transformation and Parser Generators
文法自动变换及解析器生成器的设计与实现

Yihan Zhang
张艺瀚

Northeastern University

zhng1573@hotmail.com

2016 年 6 月 2 日

Overview

预备知识与记号

Definition

A context-free grammar(上下文无关文法) G is a quadruple(四元组) $G = (V, T, P, S)$, where V is nonterminal(variable, 非终结符号或变元) set, T is terminal(终结符号) set, P is production(产生式) set with productions shaped like $A \to \alpha, A \in V, \alpha \in (V \cup T)^*$, S is start variable(开始变元).

迭代到不动点

In this frame, we will introduce a technique called "iteration to a fixed point(迭代到不动点)". Take computing the generating symbols(可产生的符号) as an example. A symbol is said to be generating if $A \overset{*}{\Rightarrow} \omega, \omega \in T^*$. In most textbooks, generating symbols are generally computed by the following rules:

- All terminals are generating;
- If $A \to \alpha \in P$, $\alpha = \varepsilon$ or every symbol in α is generating. Then A is generating.

Given any system of set equations, it's easy to get corresponding algorithm using the technique iteration to a fixed point. Algorithm for the last example is:

Algorithm 1 Compute generating variables

Require: $G = (V, T, P, S)$

Ensure: $V_{old} = \emptyset, V_{new} = \{A : A \to \omega \in P, \omega \in T^*\}$

while $V_{old} \neq V_{new}$ **do**

  $V_{old} = V_{new}$

  $V_{new} = V_{old} \cup \{A : A \to \alpha \in P, \alpha \in (T \cup V_{old})^*\}$

end while

$V = V_{new}$

原始 CYK 算法

Algorithm 2 Original CYK

Require: $G = (V, T, P, S)$ in CNF, $a_1 a_2 \cdots a_n \in T^n$

for $i = 0$ **to** $n - 1$ **do**
  $T_{i0} = \{A : A \rightarrow a_i \in P\}$
end for
for $j = 1$ **to** $n - 1$ **do**
  for $i = 0$ **to** $n - j - 1$ **do**
    $T_{ij} = \emptyset$
    for $k = 0$ **to** $j - 1$ **do**
      $T_{ij} = T_{ij} \cup \{A : A \rightarrow BC \in P, B \in T_{ik}, C \in T_{(i+k)(j-k)}\}$
    end for
  end for
end for

Remark

CYK algorithm requires the input grammar to be in CNF(乔姆斯基范式), whose productions are all in the form $A \to BC, B, C \in V$. *Definition of* T_{ij} *in the algorithm is* $T_{ij} = \{A : A \stackrel{*}{\Rightarrow} a_i a_{i+1} \cdots a_{i+j-2} a_{i+j-1}\}$. *After running the algorithm, if* $S \in T_{1n}$, *then* $a_1 a_2 \cdots a_n \in L(G)$.

改进 CYK 算法

Algorithm 3 Improved CYK

Ensure: $M =$ all **false**, $M[0, i, j] =$**true**, $i = 0, 1, \cdots, n, V_j \rightarrow a_i \in P$

for $i = 1$ **to** $n - 1$ **do**
    for $j = 0$ **to** $n - i - 1$ **do**
        for $k = 0$ **to** $i - 1$ **do**
            for all V_A**to**$V_B V_C$ **do**
                if $M[k, j, B], M[i - k - 1, j + k + 1, C]$ **then**
                    $M[i, j, A] =$**true**
                end if
            end for
        end for
    end for
end for

Remark

After running the improved version, if $M[n-1, 0, S]$*, then* $V_S \in T_{1n}$*. In fact,* $V_k \in T[i, j] \iff M[i, j, k]$*. Considering* V *is a finite set, we use a 3-dimensional boolean matrix(三维布尔矩阵)* M *to represent whether a variable is in an entry of the original 2-dimensional variable-set matrix(二维变元集合矩阵)* T*. This improvement avoids union-find operation of sets(集合的并查操作) which may have a high-complexity implementation and makes the algorithm more practical.*

原始 Earley 算法

Given any augmented CFG(**增广文法**,with additional variable S' and production $S' \to S$) G and $a_1 a_2 \cdots a_n \in T^n$, Earley algorithm maintains a list of $n+1$ sets and can be divided into two stages:

1. Initialization(**初始化**): $S_0 = \{(S' \to \cdot S, 0)\}$;
2. Assuming the current position is in the k-th slot of input string, perform the following operation until a fixed point:
   - Predict(**预测**) S_k;
   - Scan(**扫描**) S_k;
   - Complete(**扩充**) S_k.

改进 Earley 算法

Algorithm 4 Earley

for $k = 0$ **to** n **do**
    for all $s \in S_k$ **do**
        if $\cdot$ is at the end **then**
            if nonterminal after $\cdot$ **then**
                Predict(G, k, s)
            else
                Scan$(a_1 a_2 \cdots a_n, k, s)$
            end if
        else
            Complete(k, s)
        end if
    end for

Remark

The improved version modifies three operations to make them act on a single state rather than a state set and puts them into a wider loop, reducing repeated search(减少重复搜索).

系统设计

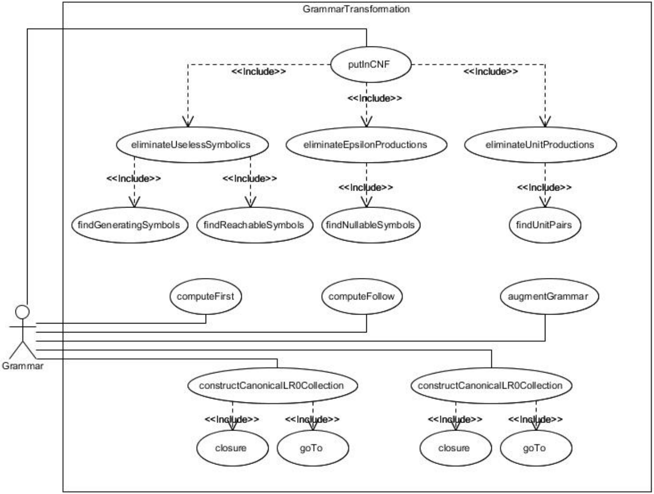


图: 文法自动变换用例图

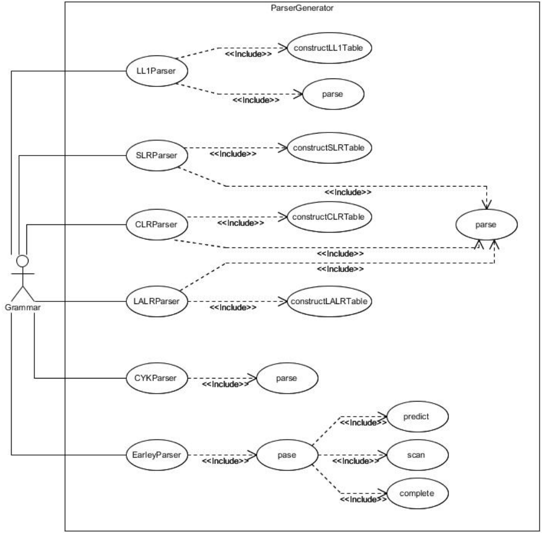

ParserGenerator
Grammar
LL1Parser
constructLL1Table
parse
SLRParser
constructSLRTable
CLRParser
constructCLRTable
LALRParser
constructLALRTable
CYKParser
EarleyParser
pase
predict
scan
complete
<<Include>>

图: 解析器生成器用例图

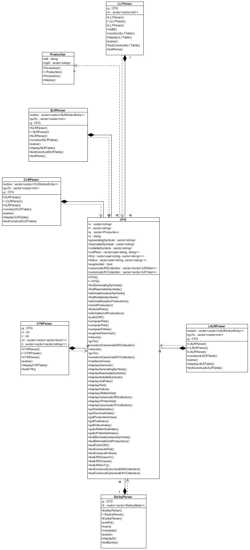

LL1Parser
-g : CFG
-m : vector<vector<int>>
+LL1Parser()
+~LL1Parser()
+LL1Parser()
+setM()
+constructLL1Table()
+displayLL1Table()
+parse()
+testConstructLL1Table()
+testParse()
1
Production
+left : string
+right : vector<string>
+Production()
+~Production()
+Production()
+display()
-p
SLRParser
+action : vector<vector<SLRActionEntry>>
+goTo : vector<vector<int>>
-g : CFG
+SLRParser()
+~SLRParser()
+SLRParser()
+constructSLRTable()
+parse()
+displaySLRTable()
+testConstructSLRTable()
+testParse()
CLRParser
+action : vector<vector<CLRActionEntry>>
+goTo : vector<vector<int>>
-g : CFG
+CLRParser()
+~CLRParser()
+CLRParser()
+constructCLRTable()
+parse()
+displayCLRTable()
+testConstructCLRTable()
-g
-g
-g
CFG
+v : vector<string>
+t : vector<string>
+p : vector<Production>
+s : string
+generatingSymbols : vector<string>
+reachableSymbols : vector<string>
+nullableSymbols : vector<string>
+unitPairs : vector<pair<string, string>>
+first : vector<pair<string, vector<string>>>
+follow : vector<pair<string, vector<string>>>
+augmented : bool
+canonicalLR0Collection : vector<vector<LR0Item>>
+canonicalLR1Collection : vector<vector<LR1Item>>
+CFG()
+~CFG()
+findGeneratingSymbols()
+findReachableSymbols()
+eliminateUselessSymbols()
+findNullableSymbols()
+eliminateEpsilonProductions()
+isUnitProduction()
+findUnitPairs()
+eliminateUnitProductions()
+putInCNF()
+computeFirst()
+computeFirst()
+computeFollow()
+augmentGrammar()
+closure()
+goTo()
+constructCanonicalLR0Collection()
+closure()
+goTo()
+constructCanonicalLR1Collection()
+hasSameCore()
+displayGrammar()
+displayGeneratingSymbols()
+displayReachableSymbols()
+displayNullableSymbols()
+displayUnitPairs()
+displayFirst()
+displayFollow()
+displayLR0ItemSet()
+displayCanonicalLR0Collection()
+displayLR1ItemSet()
+displayCanonicalLR1Collection()
+getVariableIndex()
+getTerminalIndex()
+getProductionIndex()
+getFirstIndex()
+getFollowIndex()
+getLR0ItemSetIndex()
+getLR1ItemSetIndex()
+testEliminateUselessSymbols()
+testEliminateUnitProductions()
+testPutInCNF()
+testComputeFirst()
+testComputeFollow()
+testLR0Closure1()
+testLR0Closure()
+testLR0GoTo()
+testConstructCanonicalLR0Collection()
+testConstructCanonicalLR1Collection()
CYKParser
-g : CFG
-n : int
-r : int
-m : vector<vector<vector<bool>>>
-t : vector<vector<vector<string>>>
+CYKParser()
+~CYKParser()
+CYKParser()
+parse()
+displayCYKTable()
+testCYK()
-g
LALRParser
+action : vector<vector<LALRActionEntry>>
+goTo : vector<vector<int>>
-g : CFG
+LALRParser()
+~LALRParser()
+LALRParser()
+constructLALRTable()
+parse()
+displayLALRTable()
+testConstructLALRTable()
-g
-g
1
EarleyParser
-g : CFG
-S : vector<vector<EarleyState>>
+EarleyParser()
+~EarleyParser()
+EarleyParser()
+predict()
+scan()
+complete()
+parse()
+displayS()
+testEarley()

图: 类图

Automated grammar transformer(simplify any given CFG and compute relevant sets automatically, 文法自动变换: 自动化简任意 CFG 并计算相关集合):

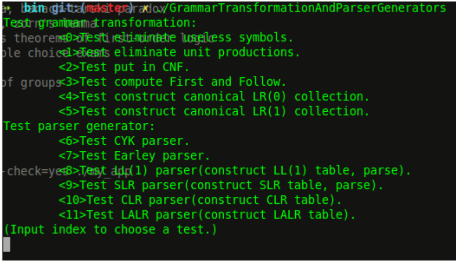
- Eliminate useless symbols(去除无用符号);
- Eliminate ε-productions(去除 ε-产生式);
- Eliminate unit production(去除单一产生式);
- Put in CNF(化为乔姆斯基范式).

Parser generators(generate parse table for any given CFG automatically, 解析器生成器: 自动生成任意 CFG 的语法分析表):

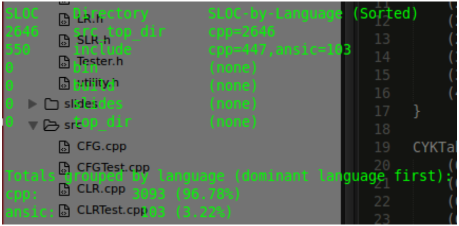
- CYK, Earley
- LL(1)
- SLR, CLR, LALR

系统实现

- Hardware:ThinkPad-Edge-E530, 3.5GiB memory, Intel Core i5-3210M CPU, 2.50GHzx4, Intel Ivybridge Mobile;
- Operating system: Ubuntu 14.04 LTS, 64-bit;
- Tools: C++1y, Sublime Text 3, makefile;
- Environment requirement: g++ (Ubuntu 4.8.4-2ubuntu1 14.04.1), Copyright (C) 2013 Free Software Foundation, Inc.



(a) User interface



(b) 3000+ lines codes

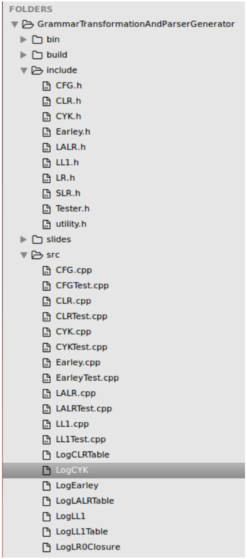


图: Directory structure

系统测试

```
CYKTable = {
        (0, 0) { B, },
        (0, 1) { A, C, },
        (0, 2) { A, C, },
        (0, 3) { B, },
        (0, 4) { A, C, },
        (1, 0) { S, A, },
        (1, 1) { B, },
        (1, 2) { S, C, },
        (1, 3) { S, A, },
        (2, 0) { },
        (2, 1) { B, },
        (2, 2) { B, },
        (3, 0) { },
        (3, 1) { S, A, C, },
        (4, 0) { S, A, C, },
}

CYKTable = {
        (0, 0) { NP, },
        (0, 1) { VP, V, },
        (0, 2) { Det, },
        (0, 3) { N, },
        (0, 4) { P, },
        (0, 5) { Det, },
        (0, 6) { N, },
        (1, 0) { S, },
        (1, 1) { },
        (1, 2) { NP, },
        (1, 3) { },
        (1, 4) { },
        (1, 5) { NP, },
        (2, 0) { },
        (2, 1) { VP, },
        (2, 2) { },
        (2, 3) { },
        (2, 4) { PP, },
        (3, 0) { S, },
        (3, 1) { },
        (3, 2) { },
```

(a) CYK log

```
S = {
        (0, 0) (_S -> . S , 0)
        (0, 1) (S -> . S + M , 0)
        (0, 2) (S -> . M , 0)
        (0, 3) (M -> . M * T , 0)
        (0, 4) (M -> . T , 0)
        (0, 5) (T -> . 1 , 0)
        (0, 6) (T -> . 2 , 0)
        (0, 7) (T -> . 3 , 0)
        (0, 8) (T -> . 4 , 0)
        (1, 0) (T -> 2 . , 0)
        (1, 1) (M -> T . , 0)
        (1, 2) (S -> M . , 0)
        (1, 3) (M -> M . * T , 0)
        (1, 4) (_S -> S . , 0)
        (1, 5) (S -> S . + M , 0)
        (2, 0) (S -> S + . M , 0)
        (2, 1) (M -> . M * T , 2)
        (2, 2) (M -> . T , 2)
        (2, 3) (T -> . 1 , 2)
        (2, 4) (T -> . 2 , 2)
        (2, 5) (T -> . 3 , 2)
        (2, 6) (T -> . 4 , 2)
        (3, 0) (T -> 3 . , 2)
        (3, 1) (M -> T . , 2)
        (3, 2) (S -> S + M . , 0)
        (3, 3) (M -> M . * T , 2)
        (3, 4) (_S -> S . , 0)
        (3, 5) (S -> S . + M , 0)
        (4, 0) (M -> M * . T , 2)
        (4, 1) (T -> . 1 , 4)
        (4, 2) (T -> . 2 , 4)
        (4, 3) (T -> . 3 , 4)
        (4, 4) (T -> . 4 , 4)
        (5, 0) (T -> 4 . , 4)
        (5, 1) (M -> M * T . , 2)
        (5, 2) (S -> S + M . , 0)
        (5, 3) (M -> M . * T , 2)
        (5, 4) (_S -> S . , 0)
        (5, 5) (S -> S . + M , 0)
```

(b) Earley log

```
LL1Table = {
    (E, +) error,
    (E, *) error,
    (E, () E -> T E_ ,
    (E, )) error,
    (E, 0) E -> T E_ ,
    (E, $) error,
    (T, +) error,
    (T, *) error,
    (T, () T -> F T_ ,
    (T, )) error,
    (T, 0) T -> F T_ ,
    (T, $) error,
    (E_, +) E_ -> + T E_ ,
    (E_, *) error,
    (E_, () error,
    (E_, )) E_ ->  ,
    (E_, 0) error,
    (E_, $) E_ ->  ,
    (F, +) error,
    (F, *) error,
    (F, () F -> ( E ) ,
    (F, )) error,
    (F, 0) F -> 0 ,
    (F, $) error,
    (T_, +) T_ ->  ,
    (T_, *) T_ -> * F T_ ,
    (T_, () error,
    (T_, )) T_ ->  ,
    (T_, 0) error,
    (T_, $) T_ ->  ,
}
```

(a) LL(1) table log

```
stk: { E, $, }
x: E
a: 0
E -> T E_

stk: { T, E_, $, }
x: T
a: 0
T -> F T_

stk: { F, T_, E_, $, }
x: F
a: 0
F -> 0

stk: { 0, T_, E_, $, }
x: 0
a: 0
match 0

stk: { T_, E_, $, }
x: T_
a: +
T_ ->

stk: { E_, $, }
x: E_
a: +
E_ -> + T E_

stk: { +, T, E_, $, }
x: +
a: +
match +

stk: { T, E_, $, }
x: T
a: 0
T -> F T_
```

(b) LL(1) parsing log

```
canonicalLR0Collection = {
    I0 = {
        _E -> . E ,
        E -> . E + T ,
        E -> . T ,
        T -> . T * F ,
        T -> . F ,
        F -> . ( E ) ,
        F -> . 0 ,
    }

    I1 = {
        _E -> E . ,
        E -> E . + T ,
    }

    I2 = {
        E -> T . ,
        T -> T . * F ,
    }

    I3 = {
        T -> F . ,
    }

    I4 = {
        F -> ( . E ) ,
        E -> . E + T ,
        E -> . T ,
        T -> . T * F ,
        T -> . F ,
        F -> . ( E ) ,
        F -> . 0 ,
    }

    I5 = {
        F -> 0 . ,
    }

    I6 = {
```

(a) LR(0) collection log

```
action = {
    (0, +) error,
    (0, *) error,
    (0, () shift 4,
    (0, )) error,
    (0, 0) shift 5,
    (0, $) error,
    (1, +) shift 6,
    (1, *) error,
    (1, () error,
    (1, )) error,
    (1, 0) error,
    (1, $) accept,
    (2, +) reduce E -> T ,
    (2, *) shift 7,
    (2, () error,
    (2, )) reduce E -> T ,
    (2, 0) error,
    (2, $) reduce E -> T ,
    (3, +) reduce T -> F ,
    (3, *) reduce T -> F ,
    (3, () error,
    (3, )) reduce T -> F ,
    (3, 0) error,
    (3, $) reduce T -> F ,
    (4, +) error,
    (4, *) error,
    (4, () shift 4,
    (4, )) error,
    (4, 0) shift 5,
    (4, $) error,
    (5, +) reduce F -> 0 ,
    (5, *) reduce F -> 0 ,
    (5, () error,
    (5, )) reduce F -> 0 ,
    (5, 0) error,
    (5, $) reduce F -> 0 ,
    (6, +) error,
    (6, *) error,
    (6, () shift 4,
```

(b) SLR table log

```
canonicalLR1Collection = {
    I0 = {
        _S -> . S , $,
        S -> . C C , $,
        C -> . c C , c,
        C -> . c C , d,
        C -> . d , c,
        C -> . d , d,
    }

    I1 = {
        _S -> S . , $,
    }

    I2 = {
        S -> C . C , $,
        C -> . c C , $,
        C -> . d , $,
    }

    I3 = {
        C -> c . C , c,
        C -> c . C , d,
        C -> . c C , c,
        C -> . d , c,
        C -> . c C , d,
        C -> . d , d,
    }

    I4 = {
        C -> d . , c,
        C -> d . , d,
    }

    I5 = {
        S -> C C . , $,
    }

    I6 = {
        C -> c . C , $,
```

(a) LR(1) collection log

```
action = {
    (0, c) shift 3,
    (0, d) shift 4,
    (0, $) error,
    (1, c) error,
    (1, d) error,
    (1, $) accept,
    (2, c) shift 6,
    (2, d) shift 7,
    (2, $) error,
    (3, c) shift 3,
    (3, d) shift 4,
    (3, $) error,
    (4, c) reduce C -> d ,
    (4, d) reduce C -> d ,
    (4, $) error,
    (5, c) error,
    (5, d) error,
    (5, $) reduce S -> C C ,
    (6, c) shift 6,
    (6, d) shift 7,
    (6, $) error,
    (7, c) error,
    (7, d) error,
    (7, $) reduce C -> d ,
    (8, c) reduce C -> c C ,
    (8, d) reduce C -> c C ,
    (8, $) error,
    (9, c) error,
    (9, d) error,
    (9, $) reduce C -> c C ,
}

goTo = {
    (0, S) 1,
    (0, C) 2,
    (1, S) error,
    (1, C) error,
    (2, S) error,
    (2, C) 5,
```

(b) CLR table log

```
action = {
    (0, c) shift 3,
    (0, d) shift 4,
    (0, $) error,
    (1, c) error,
    (1, d) error,
    (1, $) accept,
    (2, c) shift 3,
    (2, d) shift 4,
    (2, $) error,
    (3, c) shift 3,
    (3, d) shift 4,
    (3, $) error,
    (4, c) reduce C -> d ,
    (4, d) reduce C -> d ,
    (4, $) reduce C -> d ,
    (5, c) error,
    (5, d) error,
    (5, $) reduce S -> C C ,
    (6, c) reduce C -> c C ,
    (6, d) reduce C -> c C ,
    (6, $) reduce C -> c C ,
}

goTo = {
    (0, S) 1,
    (0, C) 2,
    (1, S) error,
    (1, C) error,
    (2, S) error,
    (2, C) 5,
    (3, S) error,
    (3, C) 6,
    (4, S) error,
    (4, C) error,
    (5, S) error,
    (5, C) error,
    (6, S) error,
    (6, C) error,
}
```

(c) LALR table log

结论

We design a C-like grammar(类 C 文法) which supports most basic syntax of C and use (part of) it to test the system. Conclusions are shown below:

- Any grammar which $L(G) \backslash \{\varepsilon\} \neq \emptyset$ can be put in CNF;
- Complexity of CYK and Earley are both $O(n^3)$, while $O(n^2)$ for Earley when handling unambiguous grammars(无二义性文法);
- Construction of parser generators(解析器生成器) is equivalent to construction of parse tables(语法分析表);
- LL(1) parser is equivalent to recursive descent parser without backtrack(无回溯递归下降语法分析器);
- Range of applicable grammars: $\mathrm{SLR} \subset \mathrm{LALR} \subset \mathrm{CLR}$, scale of generated parse tables: $\mathrm{SLR} = \mathrm{LALR} < \mathrm{CLR}$.

Q & A