Сравнение возможностей инструментов синтаксического анализа

Леденева Екатерина Юрьевна

3 июня 2021 г.

Генераторы парсеров

- Генератор парсеров — это инструмент, получающий на вход описание грамматики языка и выдающий парсер этого языка
- Примеры — ANTLR, Yacc, Bison, Happy, ...
- Возьмём два генератора парсеров и сравним их возможности
  - ANTLR4 — генератор парсеров семейства LL
  - Bison — генератор парсеров семейства LR

ANTLR4

- ANTLR — ANother Tool for Language Recognition — генератор парсеров, реализующих алгоритм Adaptive LL(*)
  - Adaptive LL(*) использует неограниченный предпросмотр
  - Теоретическая сложность ALL(*) парсера — $\mathcal{O}(n^4)$, однако утверждается, что на практике он обычно работает быстрее, обгоняя GLR
- ANTLR4 принимает все контекстно-свободные грамматики, не содержащие неявную левую рекурсию
- ANTLR4 реализовывает и лексинг, и парсинг
- В грамматике можно использовать регулярные выражения

Bison

- GNU Bison — генератор парсеров, реализующих алгоритм LR или Generalized LR (GLR)
  - GLR, в отличие от LR, обрабатывает все возможные интерпретации входа (в случае неоднозначности)
  - GLR принимает недетерминированные контекстно-свободные грамматики
  - Теоретическая сложность — $\mathcal{O}(n^3)$
- Bison обычно используется вместе с генератором лексеров Flex, так как на вход Bison принимает поток токенов
- В грамматике Bison не поддерживаются регулярные выражения

Левая рекурсия

ANTLR4

- Явная левая рекурсия автоматически преобразовывается в нерекурсивные правила

```
e: e '+' e | e '%' e | ID;
ID: [a-z]+;
```

- Неявная левая рекурсия не поддерживается, при её обнаружении ANTLR выдаёт соответствующую ошибку, указывая правила, участвующие в рекурсии

```
s: u T;
u: s T |;
T: 't';

...
error(119): .../IndirectLeftRecursion.g4::: The following sets of rules are
mutually left-recursive [s, u]
```

Левая рекурсия

Bison

- Bison поддерживает грамматики как с явной, так и неявной левой рекурсией

lex.l

```
...
t    { return T; }
...
```

parse.y

```
...
s: u T;
u: s T |;
```

Неоднозначность грамматик

ANTLR4 & Bison

- ANTLR4 и Bison поддерживают неоднозначные грамматики
- При неоднозначности выбирается первое правило
- Пример неоднозначной грамматики — dangling else

```
stmt:
  expr
| if_stmt;

if_stmt:
  IF expr THEN stmt
| IF expr THEN stmt ELSE stmt;

expr: ID;
```

- В Bison неоднозначность грамматики вызывает reduce-reduce и shift-reduce конфликты, о которых будут выданы предупреждения
  - Опция `-Wcounterexamples` сгенерирует контрпримеры, на которых достигается конфликт

Неоднозначность грамматик

Bison

```
sequence:
  %empty
| sequence words
| sequence redirects
;

words:
  %empty
| words word
;

redirects:
  %empty
| redirects redirect
;

word: WORD;
redirect: REDIRECT;
```

```
parse.y: warning: 3 shift/reduce conflicts [-Wconflicts-sr]
parse.y: warning: 3 reduce/reduce conflicts [-Wconflicts-rr]
parse.y: warning: shift/reduce conflict on token $end [-Wcounterexamples]
  First example: sequence • $end
  Shift derivation
    $accept
    ↳ sequence • $end
  Second example: sequence • $end
  Reduce derivation
    $accept
    ↳ sequence        $end
      ↳ sequence words
                ↳ •
parse.y: warning: reduce/reduce conflict on tokens $end, WORD, REDIRECT [-Wcounterexamples]
  Example: sequence •
  First reduce derivation
    sequence
    ↳ sequence words
              ↳ •
  Example: sequence •
  Second reduce derivation
    sequence
    ↳ sequence redirects
              ↳ •
parse.y: warning: shift/reduce conflict on token $end [-Wcounterexamples]
```

GLR

Bison

- GLR-парсер при конфликтах рассматривает все допустимые интерпретации
- Включается директивой `%glr-parser`
- У неоднозначных правил можно задавать приоритет `%dprec N`
- Можно объединить результат всех рассмотренных веток через `%merge`

GLR

Bison

```
prog:
  %empty
| prog stmt                                { std::cout << std::endl; }
;

stmt:
  expr SEP                                 %dprec 1
| decl                                     %dprec 2
;

expr:
  ID                                       { std::cout << $1 << ' '; }
| TYPENAME LBR expr RBR                    { std::cout << $1 << " <cast> "; }
| expr PLUS expr                           { std::cout << "+ "; }
| expr ASS expr                            { std::cout << "= "; }
;

decl:
  TYPENAME declarator SEP                  { std::cout << $1 << " <declare> "; }
| TYPENAME declarator ASS expr SEP         { std::cout << $1 << " <init-declare> "; }
;

declarator:
  ID                                       { std::cout << "\"" << $1 << "\" "; }
| LBR declarator RBR
;
```

```
T (x) = y+z;
"x" y z + T <init-declare>
```

Недетерминированные контекстно-свободные грамматики

ANTLR4 & Bison

- ANTLR4 и Bison поддерживают недетерминированные контекстно-свободные грамматики — грамматики, распознаваемые недетерминированными магазинными автоматами
- Пример — грамматика палиндромов чётной длины

```
s: A s A | B s B | /* empty */;
A: 'a';
B: 'b';
```

Арифметические выражения

ANTLR4 & Bison

- Можно задавать ассоциативность операторов (токенов)
  - ANTLR4

```
expr:<assoc=right> expr '^' expr
     | expr '*' expr
     | expr '+' expr
     | '(' expr ')'
     | INT
     ;
```

  - Bison

```
%token INT PLUS MULT POW LBR RBR
%left PLUS MULT
%right POW
%%
expr: INT
     | expr POW expr
     | expr MULT expr
     | expr PLUS expr
     | LBR expr RBR
     ;
```

- Приоритеты операторов определяется
  - ANTLR4 — положением правил в грамматике (выше приоритетнее)
  - Bison — положением определений операторов (ниже приоритетнее)

Состояния лексера

ANTLR4 & Flex

- Лексер может иметь состояния
  - Lexer modes в ANTLR4
  - Start conditions в Flex
- Это удобно, например, для парсинга строк многострочных комментариев с контролем правильной вложенности скобок

Состояния лексера

ANTLR4

```
lexer grammar NestedCommentsLexer;

channels { COMMENTS_CHANNEL }

ID: [a-z]+;
SEP: ';';
WS: [ \t\r\n] -> skip;
LBR: '/*' -> pushMode(COMMENT), channel(COMMENTS_CHANNEL);

mode COMMENT;
C_LBR: '/*' -> pushMode(COMMENT), channel(COMMENTS_CHANNEL);
C_RBR: '*/' -> popMode, channel(COMMENTS_CHANNEL);
OTHER: .+? -> channel(COMMENTS_CHANNEL);
```

```
parser grammar NestedCommentsParser;
options { tokenVocab = NestedCommentsLexer; }

parse: (ID SEP)+ EOF;
```

Состояния лексера

Flex & Bison

lex.l

```
%x COMMENT

%%

[a-z]+          return ID;
\;              return SEP;
[:space:]       ;
"/*"            { yy_push_state(COMMENT); }

<COMMENT>{
     "/*"       { yy_push_state(COMMENT); }
     "*/"       { yy_pop_state(); }
     .|\n       ;
}
```

parse.y

```
%token ID SEP

%%

prog: | prog stmt;
stmt: ID SEP    { std::cout << "stmt "; };
```

Семантические предикаты

ANTLR4 & Bison

- Семантические предикаты — это динамические условия, задаваемые пользователем на целевом языке, определяющие возможность применить правило
- ANTLR4
  - Синтаксис — `{ ... }?`
  - Можно задавать предикаты в правилах лексера и парсера
  - В парсере предикат может быть только в начале правила и не может зависеть от токенов, составляющих правило
- Bison
  - Синтаксис — `%?{ ... }`
  - Можно задавать только в парсере
  - Можно располагать предикат в любом месте правила

Семантические предикаты

ANTLR4

- Маленький искусственный пример — числа длины не больше 3, записанные через запятую

```
parse: number (',' number)* EOF;

number
locals [int N = 0]: ({ $N <= 3 }? DIGIT { $N++; } )+;

DIGIT: '0'..'9';
WS: [ \t\r\n] -> skip;
```

- "111,222,333" — принимаемая строка
- "1111111111111111" — не принимается

Семантические предикаты

Bison

- Рассмотрим язык, состоящий из
  - Объявлений типа — `typename <type>`
  - Приведений типа — `(<type>) <id>`
  - Разыменовывания — `*<id>`
  - Умножения и скобок
- Без контекста невозможно определить, что означает выражение (A)*B
  - Умножение — A*B
  - Приведение результата разыменовывания B к типу A
- Если парсер хранит все объявленные типы, он может правильно определить, что такое A

Семантические предикаты

Bison

```
%{
    ...
    std::unordered_set<std::string> types;
%}

...

prog: %empty
    | prog stmt
    ;
stmt: expr SEP
    | decl SEP
    ;
decl: TYPENAME ID          { types.insert($2); };
expr: LBR type RBR expr { std::cout << "typecast" << std::endl; }
    | expr MUL expr      { std::cout << "multiplication" << std::endl; }
    | id
    | dereference
    | LBR expr RBR
    ;
dereference: MUL id;
id: ID                    %?{ !types.contains($1) };
type: ID                  %?{ types.contains($1) };
```

```
(a) *b;                   multiplication
typename a;
(a) *b;                   typecast
(b) *b;                   multiplication
```

Lexer hack

ANTLR4

- Такую же грамматику можно реализовать в ANTLR4 с помощью lexer hack — использования лексером контекста парсера

```
grammar LexerHack;

@header {
    import java.util.*;
}

@parser::members {
    static Set<String> types = new HashSet<String>();
}

parse: (stat ';')+ EOF;
stat: decl | expr;
decl: 'typename' ID          { types.add($ID.text); };
expr: '(' TYPE ')' expr      { System.out.println("typecast"); }
    | expr '*' expr          { System.out.println("multiplication"); }
    | ID
    | dereference
    | '(' expr ')'
    ;
dereference: '*' ID;
TYPE: [a-z]+              { LexerHackParser.types.contains(getText()) }?;
ID: [a-z]+;
WS: [ \t\r\n] -> skip;
```

Сводная таблица возможностей

	ANTLR4	Bison
Левая рекурсия	явная	+
Неоднозначность грамматик	+	+
Генерация контрпримеров	–	+
Недетерминированные CFG	+	+
Приоритеты и ассоциативность	+	+
Состояния лексера	+	+ (flex)
Семантические предикаты	+	+
Регулярные выражения в грамматике	+	–

Грамматики из примеров (с комментариями):
https://github.com/ekiuled/fl-2021-itmo-spr/tree/proj