Extended Context-Free Grammars Parsing with Generalized LL

Author: Artem Gorokhov

Saint Petersburg State University
Programming Languages and Tools Lab, JetBrains

March 4 2017

Java SE > Java SE Specifications > Java Language Specification

Chapter 18. Syntax

This chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *(x | y)* means one of either *x* or *y*.

```
Identifier:
    IDENTIFIER
QualifiedIdentifier:
    Identifier { . Identifier }
QualifiedIdentifierList:
    QualifiedIdentifier { , QualifiedIdentifier }
```

Extended Context-Free Grammar

$$
\begin{aligned}
S &= a\ M^* \\
M &= a?\ (B\ K)^+ \\
&\mid u\ B \\
B &= c \mid \varepsilon
\end{aligned}
$$

```
ident: IDENTIFIER
qualiId: ident {DOT ident}
qualifiedIdList: qualiId {COMMA qualiId}
compilationUnit:
    [[Annotations] Package qualiId SEMI]
    {importDecl} {typeDecl}
importDecl: Import [Static] ident
    {DOT ident} [DOT STAR] SEMI
typeDecl: classOrInterfaceDecl SEMI
classOrInterfaceDecl:
    {Modifier} (ClassDecl | InterfaceDecl)
```

$\Longrightarrow$

```
ident: IDENTIFIER
qualiId: ident many_1
many_1:
    | ident many_1
qualifiedIdList: qualiId many_2
many_2:
    | COMMA qualiId many_2
compilationUnit: opt_1 many_3 many_4
opt_2:
    | Annotations
opt_1:
    | opt_2 Package qualiId SEMI
many_3:
    | importDecl many_3
many_4:
    | typeDecl many_4
importDecl:
    Import opt_3 ident many_5 opt_4 SEMI
opt_3:
    | Static
many_5:
    | DOT ident many_5
opt_4:
    | DOT STAR
typeDecl: classOrInterfaceDecl SEMI
alt_1: ClassDecl | InterfaceDecl
classOrInterfaceDecl:
    many_6 alt_1
many_6:
    | Modifier many_6
```

ORACLE

Java SE > Java SE Specifications > Java Language Specification

Chapter 18. Syntax

This chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *(x | y)* means one of either *x* or *y*.

```
Identifier:
    IDENTIFIER
QualifiedIdentifier:
    Identifier { . Identifier }
QualifiedIdentifierList:
    QualifiedIdentifier { , QualifiedIdentifier }
```

ORACLE

Java SE > Java SE Specifications > Java Language Specification

Chapter 18. Syntax

it is not an LL(1) grammar

This chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *(x | y)* means one of either *x* or *y*.

```
Identifier:
    IDENTIFIER
QualifiedIdentifier:
    Identifier { . Identifier }
QualifiedIdentifierList:
    QualifiedIdentifier { , QualifiedIdentifier }
```

Existing solutions

- ANTLR, Yacc, Bison

Existing solutions

- ANTLR, Yacc, Bison
  - Can't use ECFG without transformation
  - Admit only subclass of ECFG (LL(k), LR(k))

Existing solutions

- ANTLR, Yacc, Bison
  - Can't use ECFG without transformation
  - Admit only subclass of ECFG (LL(k), LR(k))
- Some research on ECFG parsing

Existing solutions

- ANTLR, Yacc, Bison
  - Can't use ECFG without transformation
  - Admit only subclass of ECFG (LL(k), LR(k))
- Some research on ECFG parsing
  - No tools
  - LL(k), LR(k)

Existing solutions

- ANTLR, Yacc, Bison
  - Can't use ECFG without transformation
  - Admit only subclass of ECFG (LL(k), LR(k))
- Some research on ECFG parsing
  - No tools
  - LL(k), LR(k)
- Generalized LL

Existing solutions

- ANTLR, Yacc, Bison
  - Can't use ECFG without transformation
  - Admit only subclass of ECFG (LL(k), LR(k))
- Some research on ECFG parsing
  - No tools
  - LL(k), LR(k)
- Generalized LL
  - Admit arbitrary CFG(including ambiguous)
  - Can't use ECFG without transformation

Existing solutions

- ANTLR, Yacc, Bison
  - Can't use ECFG without transformation
  - Admit only subclass of ECFG (LL(k), LR(k))
- Some research on ECFG parsing
  - No tools
  - LL(k), LR(k)
- **Generalized LL**
  - Admit arbitrary CFG(including ambiguous)
  - Can't use ECFG without transformation

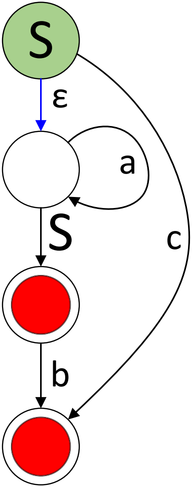
Automata and ECFGs

Grammar G_0

$$S = a^* S\ b? \mid c$$

$\Longrightarrow$

RA for grammar G_0

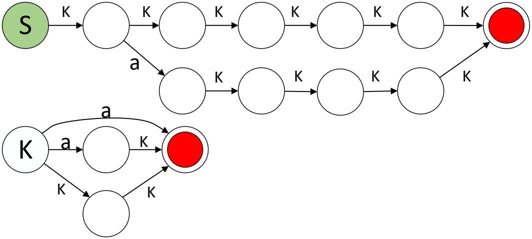


Recursive Automata Minimization

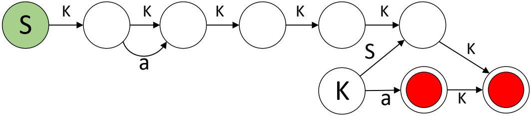
Grammar G_1

$$
\begin{aligned}
S = &\quad K\ K\ K\ K\ K\ K \mid K\ a\ K\ K\ K\ K \\
K = &\quad S\ K \mid a\ K \mid a
\end{aligned}
$$

Automaton for G_1



Minimized automaton for G_1

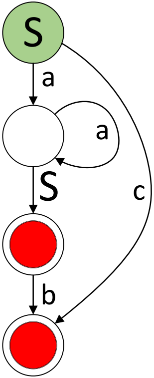


Derivation Trees for Recursive Automata

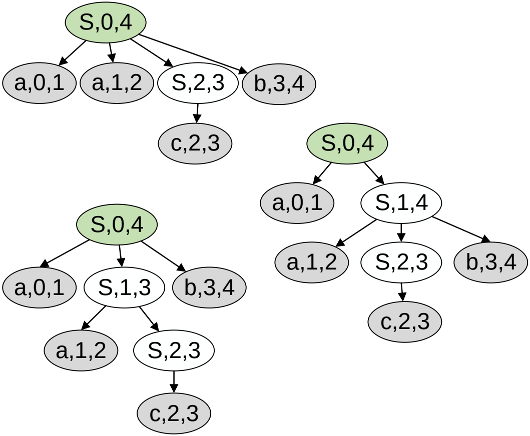
Input:

aacb

Automaton:



Derivation trees:

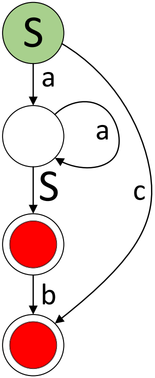


SPPF for Recursive Automata

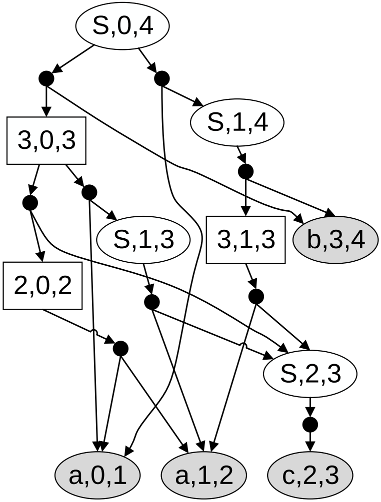
Input:

aacb

Automaton:



Shared Packed Parse Forest:



Mb highlight each tree?

Input processing

- Descriptors queue
- Descriptor (G, i, U, T) uniquely defines parsing process state
  - G - position in grammar
  - i - position in input
  - U - stack node
  - T - current parse forest root

Input processing

- Descriptors queue
- Descriptor (G, i, U, T) uniquely defines parsing process state
  - G - ~~position in grammar~~ state of RA
  - i - position in input
  - U - stack node
  - T - current parse forest root

Input processing

Input : bc

Grammar:

$$S = (a \mid b \mid S)\ c?$$

Input processing

Input : bc

Grammar:

$$
\begin{aligned}
S = &\quad a\ C\_opt \\
\mid &\quad b\ C\_opt \\
\mid &\quad S\ C\_opt \\
C\_opt = &\quad \varepsilon \mid c
\end{aligned}
$$

Input processing

Input : $\bullet\ bc$

Grammar:

$$\begin{aligned} S = &\quad \bullet\ a\ C\_opt \\ | &\quad b\ C\_opt \\ | &\quad S\ C\_opt \\ C\_opt = &\quad \varepsilon \mid c \end{aligned}$$

$$\left\| \begin{array}{l} \\ S = \bullet\ a\ C\_opt,\ 0,\ \ldots,\ \ldots \end{array} \right\|$$

Input processing

Input : $\bullet\ bc$

Grammar:

$$
\begin{aligned}
S = &\quad a\ C\_opt \\
| &\quad \bullet\ b\ C\_opt \\
| &\quad S\ C\_opt \\
C\_opt = &\quad \varepsilon \mid c
\end{aligned}
$$

$$
\left|\begin{array}{c}
S = \bullet\ b\ C\_opt,\ 0,\ \ldots,\ \ldots \\
\hline
S = \bullet\ a\ C\_opt,\ 0,\ \ldots,\ \ldots
\end{array}\right|
$$

Input processing

Input : $\bullet$ bc

Grammar:

$$
\begin{aligned}
S = \quad & a\ C\_opt \\
| \quad & b\ C\_opt \\
| \quad & \bullet\ S\ C\_opt \\
C\_opt = \quad & \varepsilon \mid c
\end{aligned}
$$

$S = \bullet\ S\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ b\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ a\ C\_opt,\ 0,\ \ldots,\ \ldots$

Input processing

Input : $\bullet \ bc$

Grammar:

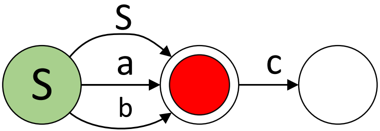
$$\begin{aligned} S = &\quad a\ C\_opt \\ | &\quad b\ C\_opt \\ | &\quad S\ C\_opt \\ C\_opt = &\quad \varepsilon \mid c \end{aligned}$$

$S = \bullet\ S\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ b\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ a\ C\_opt,\ 0,\ \ldots,\ \ldots$

Input processing

Input : bc

Automaton :

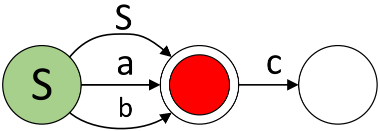


$S = \bullet\ S\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ b\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ a\ C\_opt,\ 0,\ \ldots,\ \ldots$

Input processing

Input : $\bullet$ *bc*

Automaton :

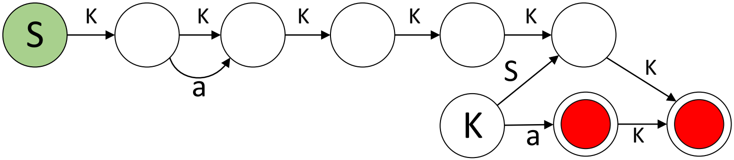


$S = \bullet\ S\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ b\ C\_opt,\ 0,\ \ldots,\ \ldots$
$S = \bullet\ a\ C\_opt,\ 0,\ \ldots,\ \ldots$

Grammar G_1

$$S = \quad K\ K\ K\ K\ K\ K\ | K\ a\ K\ K\ K\ K$$
$$K = \quad S\ K \mid a\ K \mid a$$

RA for grammar G_1



Experiment results for input a^{40}

	Descriptors	Stack Edges	Stack Nodes	SPPF Nodes
BNF Grammar	7,940	6,974	80	111,127,244
Minimized RA	5,830	4,234	80	74,292,078
Difference	27%	39%	0 %	33 %

Why did we make it?

Graph parsing results

	Descriptors	Stack Edges	Stack Nodes	Time, min
BNF Grammar	21,134,080	7,482,789	2,731,529	02.26
Minimized RA	9,153,352	2,792,330	839,148	01.25
Difference	57%	63%	69 %	45 %