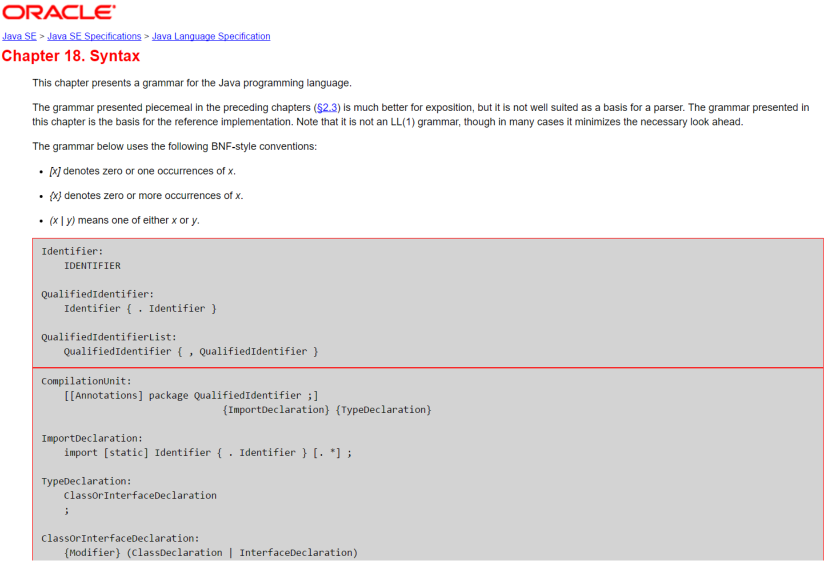
Extended Context-Free Grammars Parsing with Generalized LL

Author: Artem Gorokhov

Saint Petersburg State University
Programming Languages and Tools Lab, JetBrains

4/March/2017

Motivation



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 }
```

```
CompilationUnit:
    [[Annotations] package QualifiedIdentifier ;]
                                {ImportDeclaration} {TypeDeclaration}

ImportDeclaration:
    import [static] Identifier { . Identifier } [. *] ;

TypeDeclaration:
    ClassOrInterfaceDeclaration
    ;

ClassOrInterfaceDeclaration:
    {Modifier} (ClassDeclaration | InterfaceDeclaration)
```

Motivation

```
identifier: IDENTIFIER
qualifiedIdentifier: identifier (. identifier)*
qualifiedIdentifierList: qualifiedIdentifier (, qualifiedIdentifier)*
compilationUnit:
    [[Annotations] Package qualifiedIdentifier ;]
    (importDeclaration)* (typeDeclaration)*
importDeclaration: Import [Static] identifier (. identifier)* [. *] ;
typeDeclaration: classOrInterfaceDeclaration ;
classOrInterfaceDeclaration:
    (Modifier)* (ClassDeclaration | InterfaceDeclaration)
```

⟹

```
identifier: IDENTIFIER
qualifiedIdentifier: identifier many_1
many_1:
    | . identifier many_1
qualifiedIdentifierList: qualifiedIdentifier many_2
many_2:
    | COMMA qualifiedIdentifier many_2
compilationUnit: opt_1 many_3 many_4
opt_2:
    | Annotations
opt_1:
    | opt_2 Package qualifiedIdentifier ;
many_3:
    | importDeclaration many_3
many_4:
    | typeDeclaration many_4
importDeclaration:
    Import opt_3 identifier many_5 opt_4 ;
opt_3:
    | Static
many_5:
    | . identifier many_5
opt_4:
    | . *
typeDeclaration: classOrInterfaceDeclaration ;
classOrInterfaceDeclaration:
    many_6 ( ClassDeclaration | InterfaceDeclaration )
many_6:
    | Modifier many_6
```

Extended Context-Free Grammar

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

Existing solutions

- Mostly LL(k) and LR(k) algorithms
- **No** solution for arbitrary ECFG

Generalized LL

- Based on LL
- Admit arbitrary CFG(including ambiguous)

Generalized LL

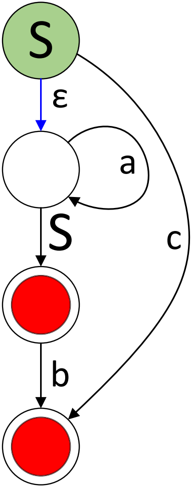
- Based on LL
- Admit arbitrary CFG(including ambiguous)
- **Works only with BNF grammars**

Automata and ECFGs

Grammar G_0

$$S = a^{*}S\ b? \mid c \qquad \Longrightarrow$$

RA for grammar G_0

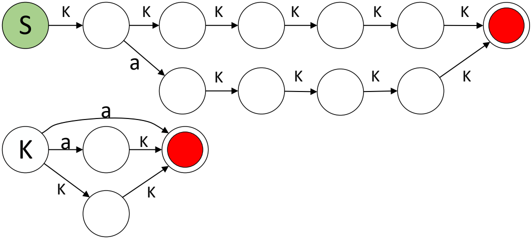


Recursive Automata Minimization

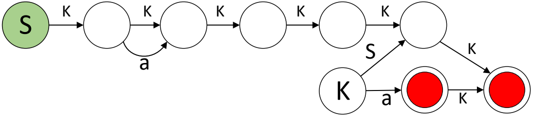
Grammar G_1

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

Automaton for G_1



Minimized automaton for G_1

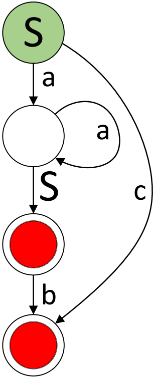


Derivation Trees for Recursive Automata

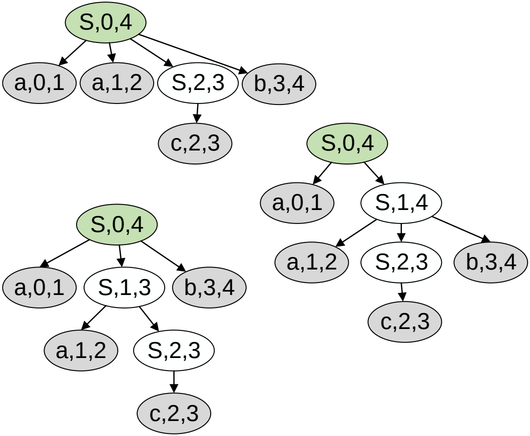
Input:

aacb

Automaton:



Derivation trees:

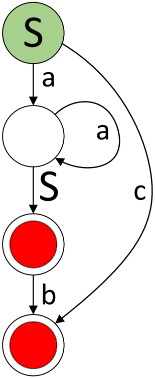


SPPF for Recursive Automata

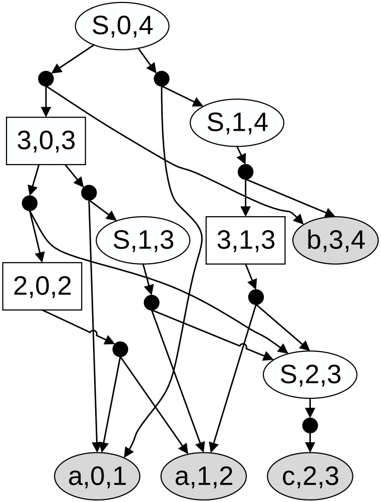
Input:

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**
  - 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 - **state of RA**
  - i - position in input
  - U - stack node
  - T - current parse forest root

Input processing

Input : bc

Grammar:

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

Input processing

Input : $\bullet \; bc$

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:

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

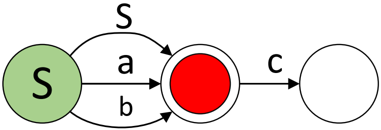
Input processing

Input : $\bullet\ bc$

Grammar:

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

Automaton :

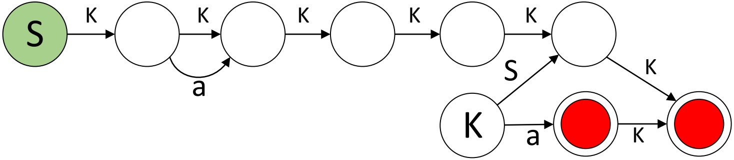


Evaluation

Grammar G_1

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

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 %