Поддержка расширенных контекстно-свободных грамматик в алгоритме синтаксического анализа Generalized LL

Автор: Горохов Артем

Санкт-Петербургский Государственный Университет

19 апреля 2017

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 }
```

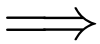
Расширенные контекстно-свободные грамматики

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

Результат преобразования в BNF

7 нетерминалов

```
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)
```



18 нетерминалов

```
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 }
```

Существующие решения

- ANTLR, Yacc, Bison

Существующие решения

- ANTLR, Yacc, Bison
  - Не могут использовать ECFG без преобразования
  - Допускают только подклассы контекстно-свободных языков (LL(k), LR(k))

Существующие решения

- ANTLR, Yacc, Bison
  - Не могут использовать ECFG без преобразования
  - Допускают только подклассы контекстно-свободных языков (LL(k), LR(k))
- Работы о синтаксическом анализе ECFG

Существующие решения

- ANTLR, Yacc, Bison
  - Не могут использовать ECFG без преобразования
  - Допускают только подклассы контекстно-свободных языков (LL(k), LR(k))
- Работы о синтаксическом анализе ECFG
  - Нет инструментов
  - LL(k), LR(k)

Существующие решения

- ANTLR, Yacc, Bison
  - Не могут использовать ECFG без преобразования
  - Допускают только подклассы контекстно-свободных языков (LL(k), LR(k))
- Работы о синтаксическом анализе ECFG
  - Нет инструментов
  - LL(k), LR(k)
- Generalized LL

Существующие решения

- ANTLR, Yacc, Bison
  - Не могут использовать ECFG без преобразования
  - Допускают только подклассы контекстно-свободных языков (LL(k), LR(k))
- Работы о синтаксическом анализе ECFG
  - Нет инструментов
  - LL(k), LR(k)
- Generalized LL
  - Допускают произвольные CFG (включая неоднозначные)
  - Не могут использовать ECFG без преобразований

Существующие решения

- ANTLR, Yacc, Bison
  - Не могут использовать ECFG без преобразования
  - Допускают только подклассы контекстно-свободных языков (LL(k), LR(k))
- Работы о синтаксическом анализе ECFG
  - Нет инструментов
  - LL(k), LR(k)
- **Generalized LL**
  - Допускают произвольные CFG (включая неоднозначные)
  - Не могут использовать ECFG без преобразований

Биоинформатика

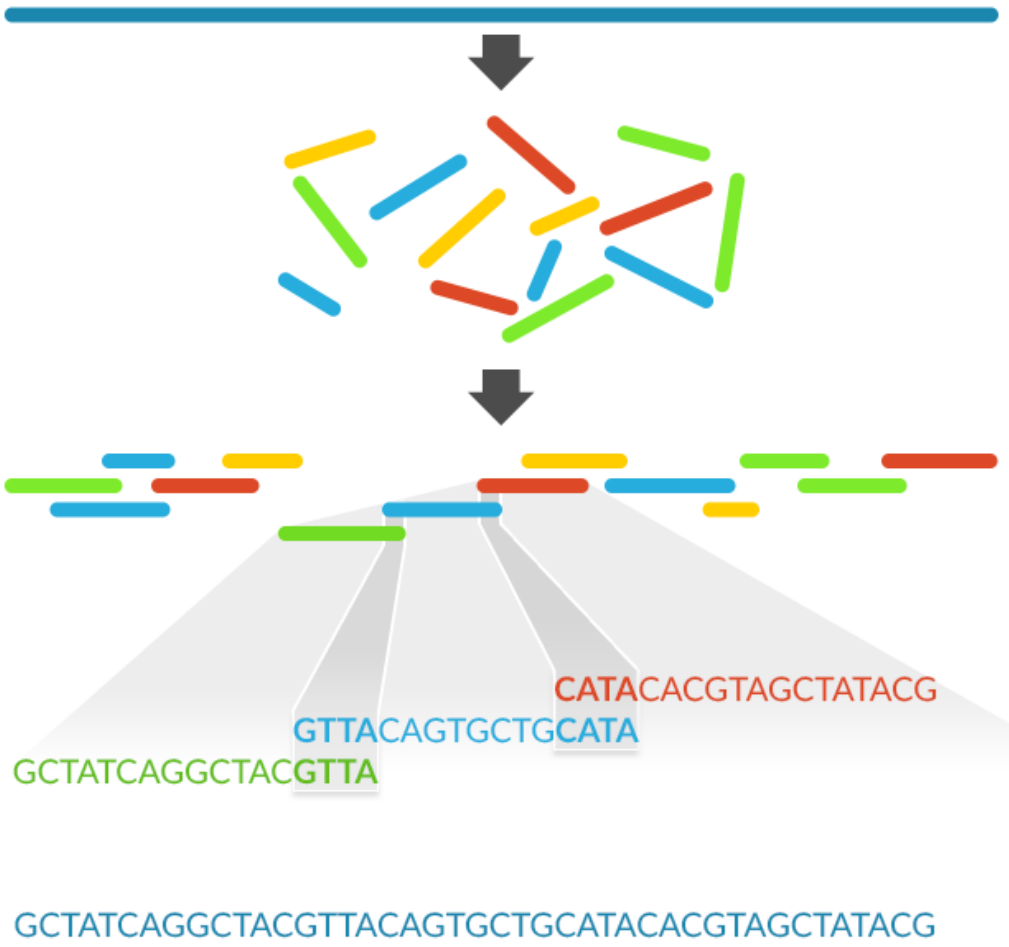
- Множество задач, связанных с обработкой и пониманием биологических данных
- Одна из задач — поиск организмов в метагеномных сборках

Геном

- Геном — длинная последовательность нуклеотидов
- На деле строка над алфавитом {A, C, G, U}

Получение данных

- Из биологического материала читаются короткие строчки
- Эти кусочки склеиваются в более длинные строки
- Множество строчек — сборка
- Данных очень много, поэтому строится граф, пути в котором содержат полученные строки



Метагеномная сборка

- Изучаем набор генов всех микроорганизмов в образце
- Нужно уметь определять содержащиеся в сборке огранизмы

Как ищем

- Такие последовательности как тРНК, рРНК и др. позволяют провести классификацию организма
- У этих последовательностей есть вторичная структура, которая может быть описана КС-грамматикой

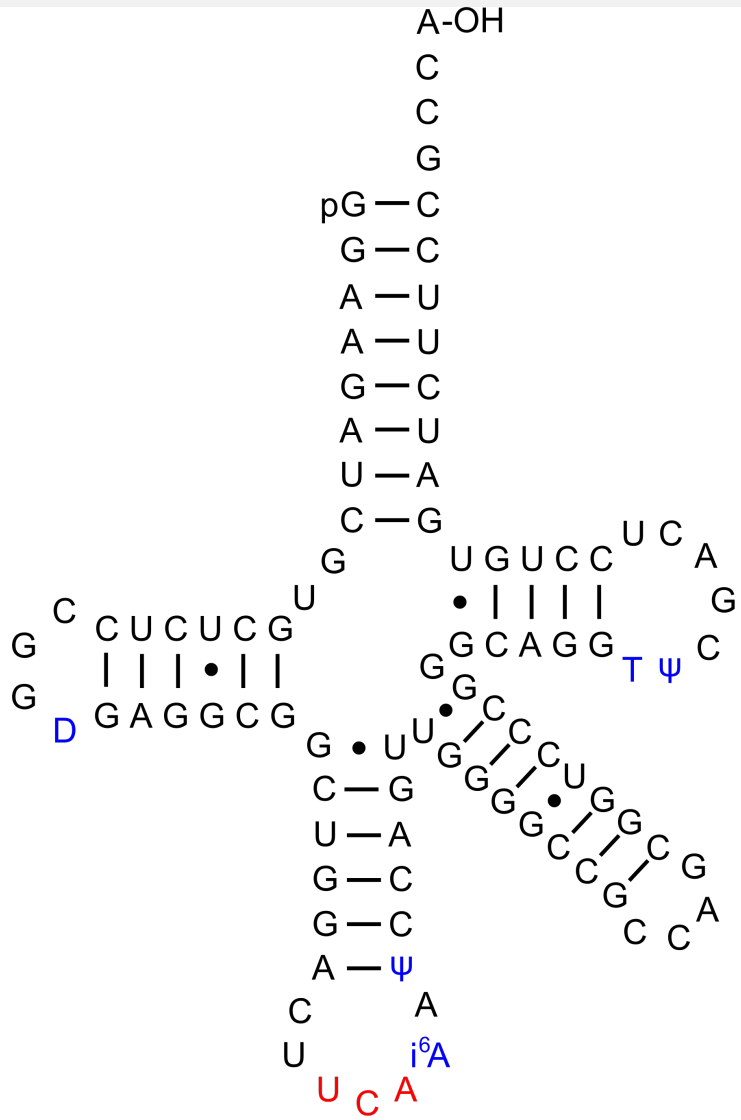


Рис.: Структура тРНК

YaccConstructor

- В рамках проекта реализован алгоритм, основанный на алгоритме GLL
- Умеет решать задачу поиска линейных цепочек в графе, удовлетворяющих KC-грамматике

Цель и задачи

Цель работы: разработать и реализовать модификацию алгоритма GLL, работающую с расширенными контекстно-свободными грамматиками, и проверить, как полученный алгоритм повлияет на производительность поиска структур, заданных с помощью контекстно-свободной грамматики, в метагеномных сборках. Для её достижения были поставлены следующие задачи:

- Выбрать или разработать подходящее представление ECFG
- Спроектировать структуру данных для представления леса разбора по ECFG
- Разработать алгоритм на основе Generalized LL, строящий лес разбора по ECFG
- Реализовать алгоритм в рамках проекта YaccConstructor
- Провести эксперименты и сравнение

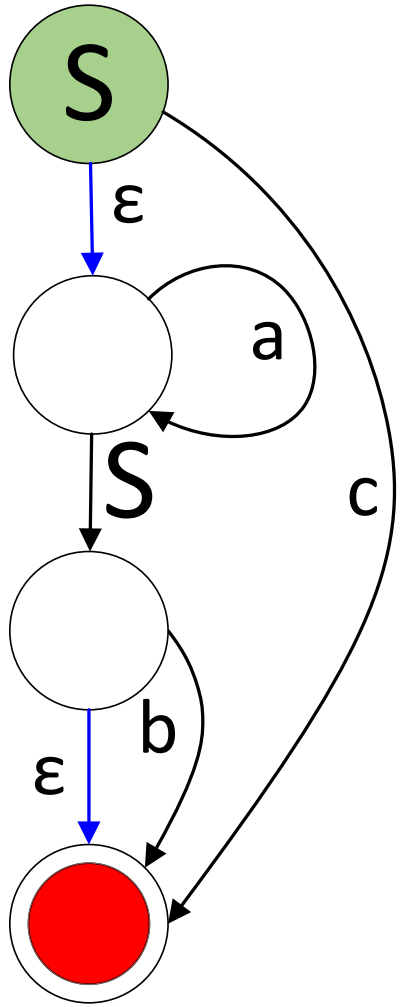
Автоматы и ECFG

Грамматика G_0

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

$\Longrightarrow$

RA для грамматики G_0

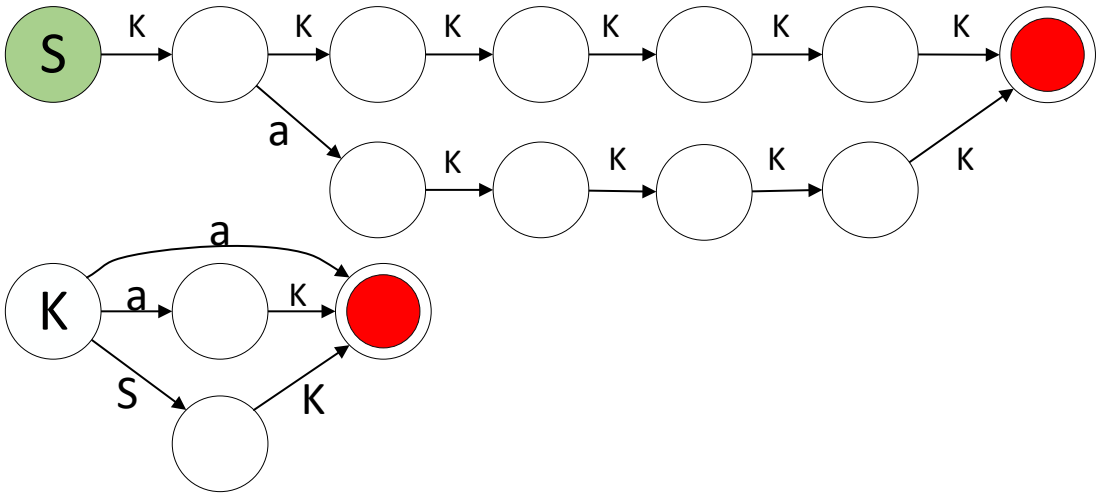


Минимизация рекурсивных автоматов

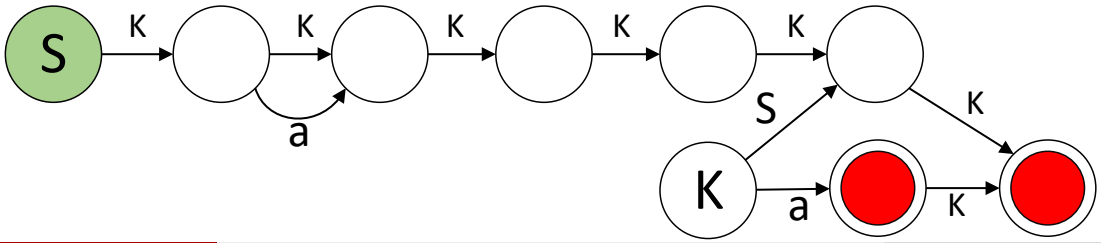
Грамматика G_1

$$S = K\ K\ K\ K\ K\ K \mid K\ a\ K\ K\ K\ K$$
$$K = S\ K \mid a\ K \mid a$$

Автомат для G_1



Минимизированный автомат для G_1

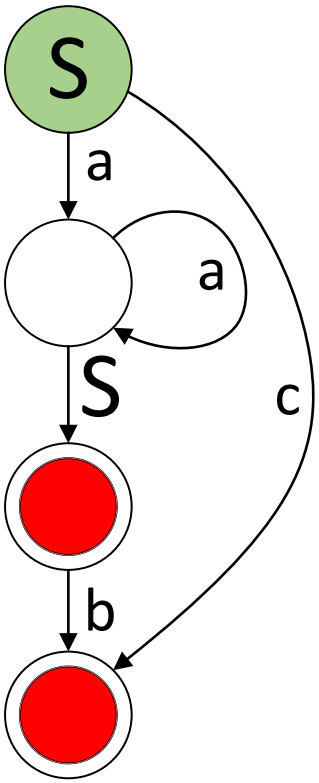


Деревья вывода для рекурсивных автоматов

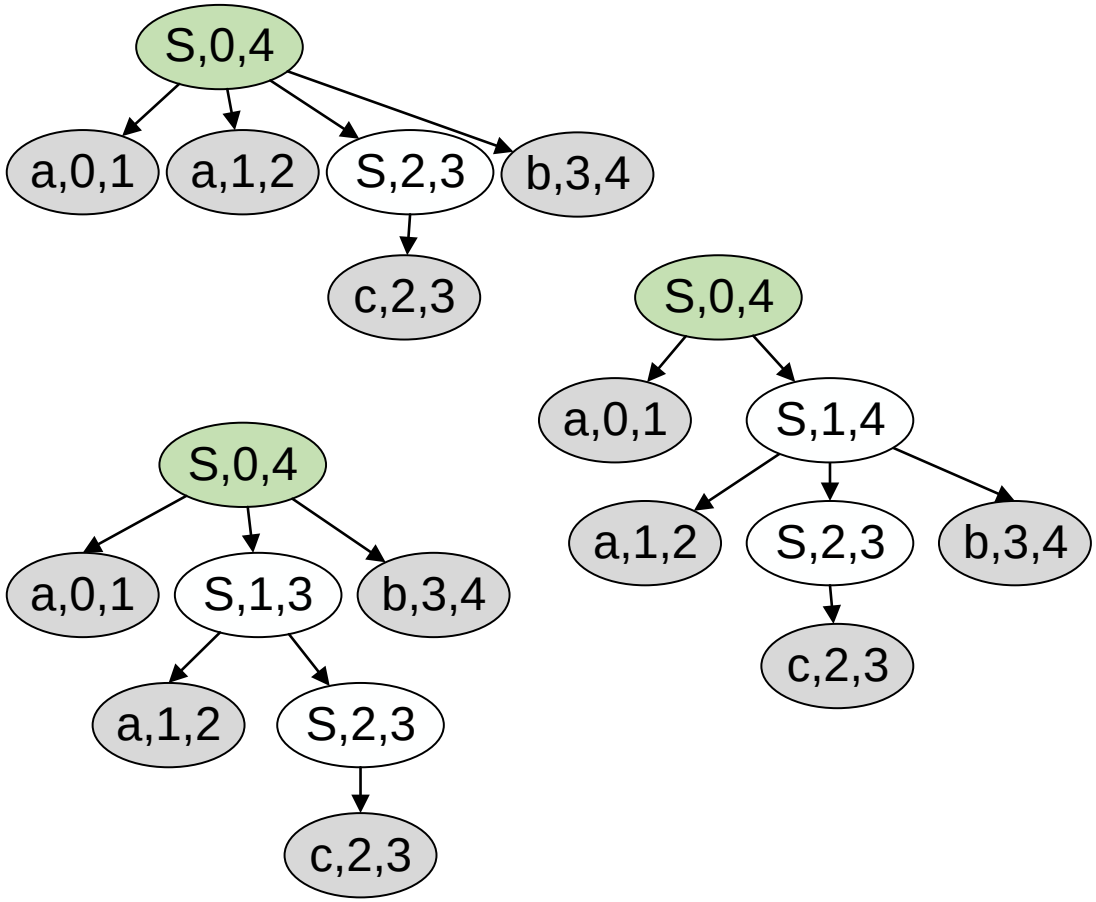
Вход:

aacb

Автомат:



Деревья вывода:

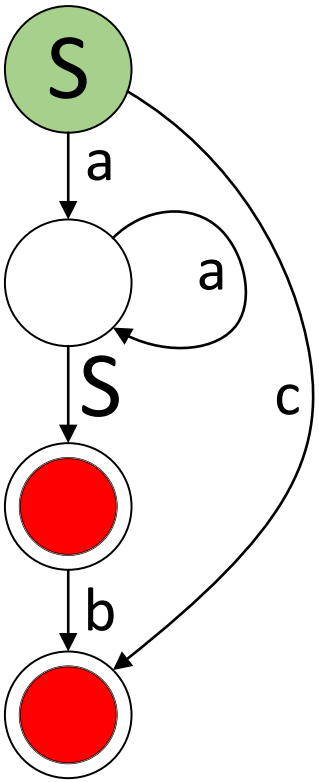


SPPF для рекурсивных автоматов

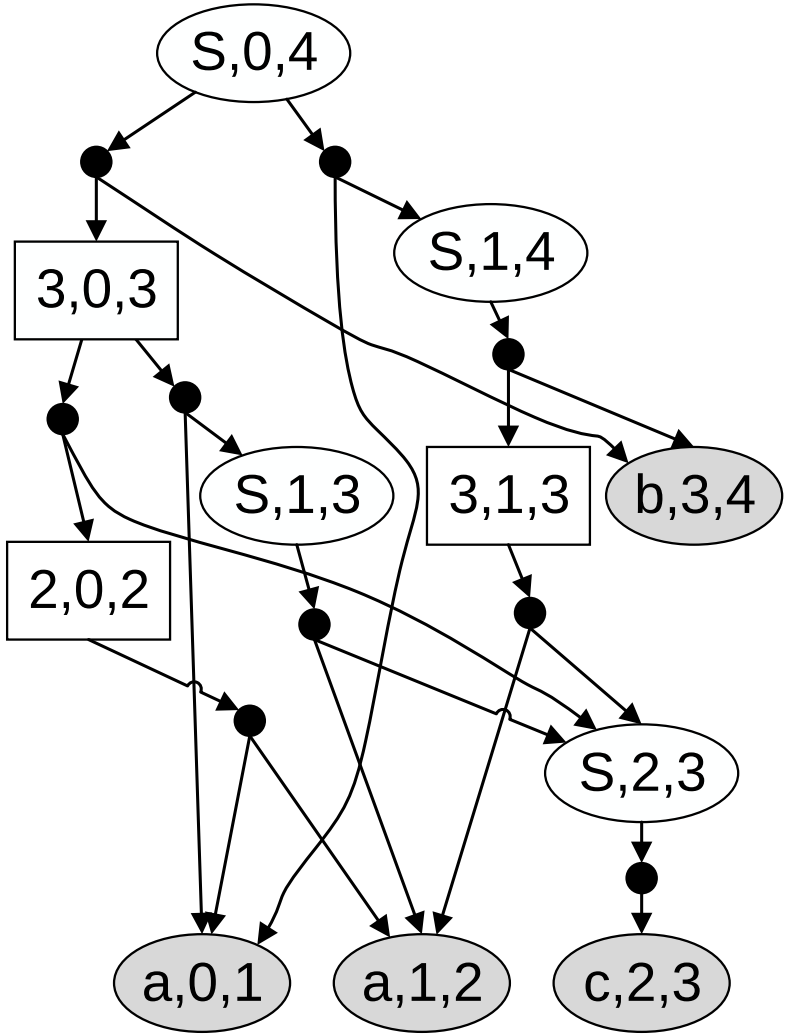
Вход:

aacb

Автомат:



Shared Packed Parse Forest:

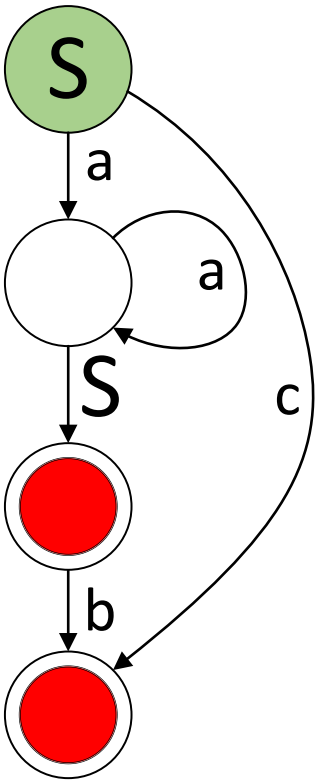


SPPF для рекурсивных автоматов

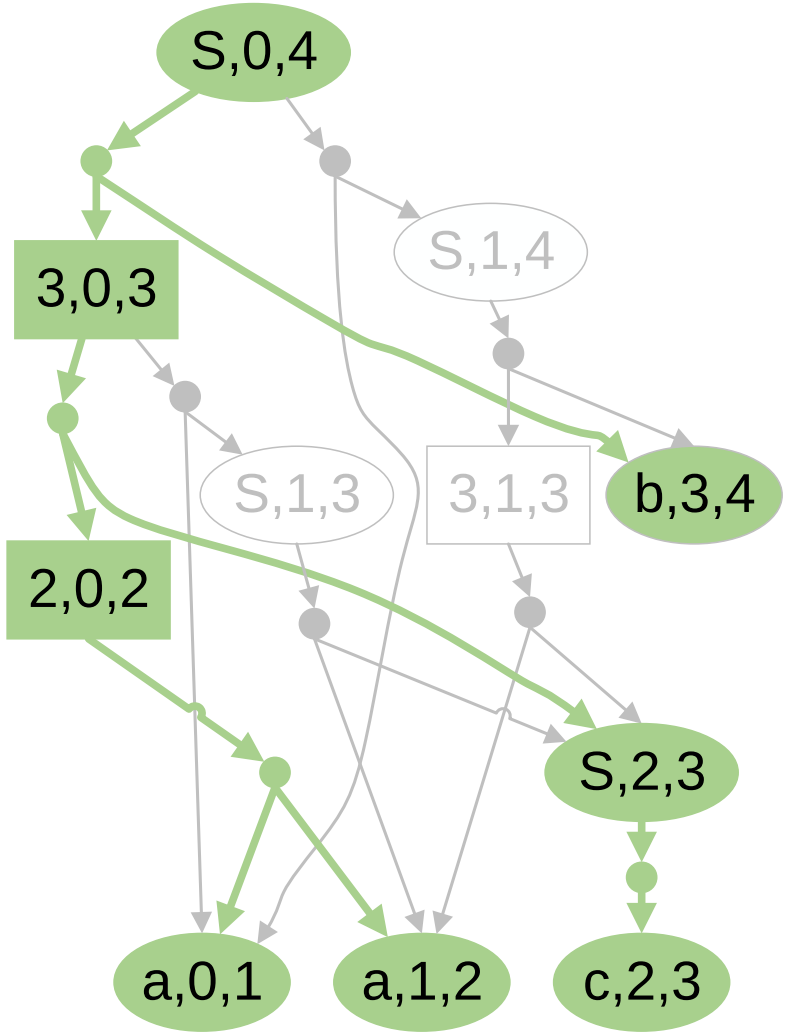
Вход:

aacb

Автомат:



Shared Packed Parse Forest:

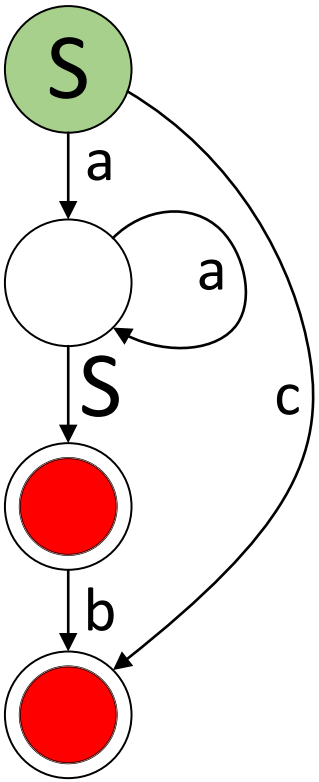


SPPF для рекурсивных автоматов

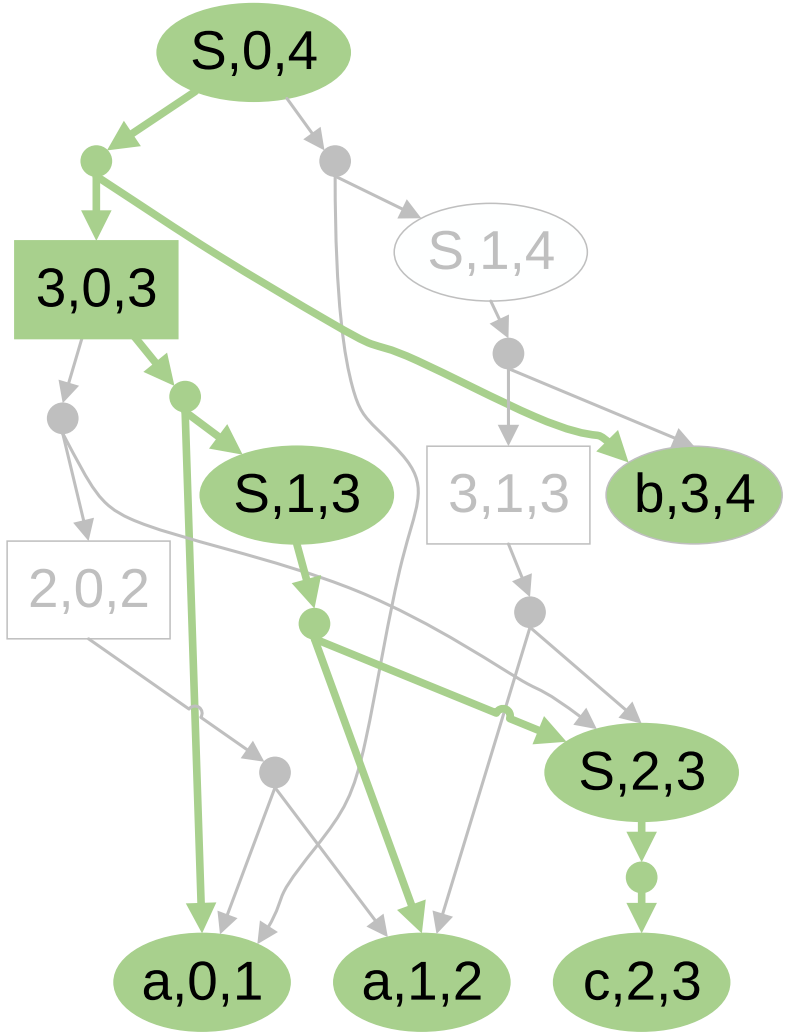
Вход:

aacb

Автомат:



Shared Packed Parse Forest:

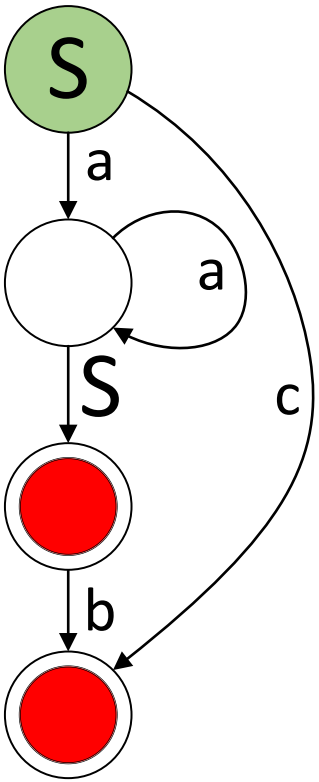


SPPF для рекурсивных автоматов

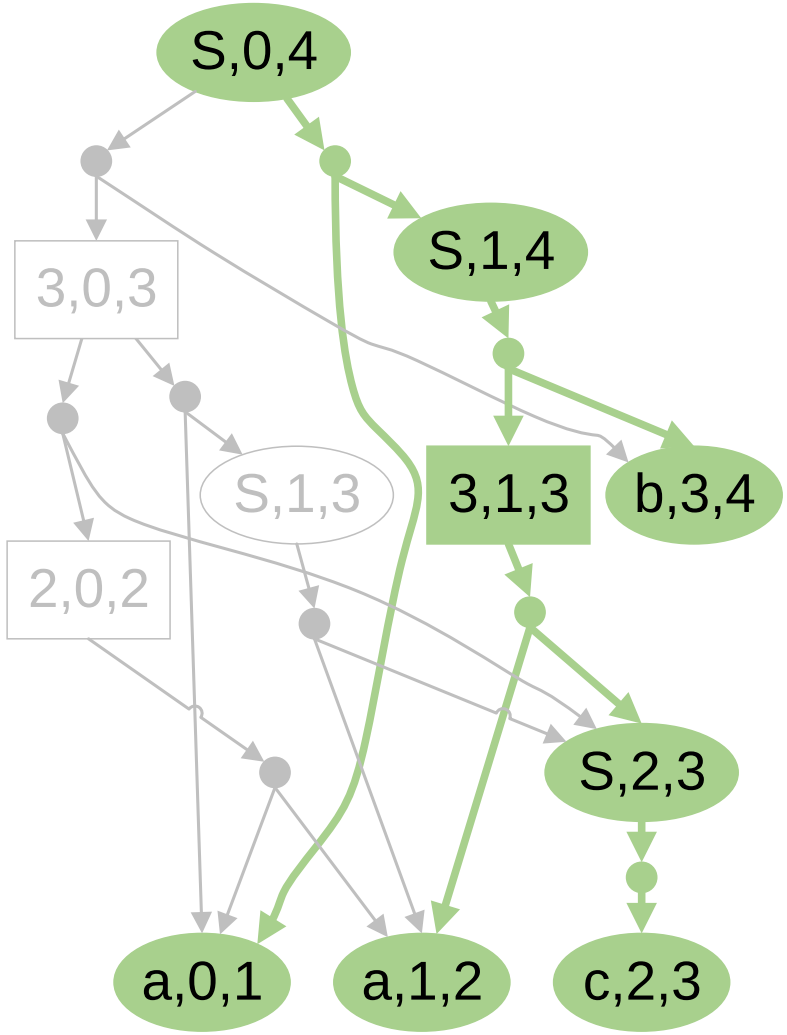
Вход:

aacb

Автомат:



Shared Packed Parse Forest:



Постоение леса разбора в оригинальном алгоритме

- Очередь дескрипторов
- Дескриптор (G, i, U, T) однозначно определяет состояние процесса разбора
  - G - позиция в грамматике
  - i - позиция во входе
  - U - узел стека разбора
  - T - корень построенного леса разбора

Пример постоения леса разбора в оригинальном алгоритме

Вход : bc

Грамматика:

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

Пример постоения леса разбора в оригинальном алгоритме

Вход : bc

Грамматика:

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

Пример постоения леса разбора в оригинальном алгоритме

Вход : $\bullet\ bc$

Грамматика:

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

Очередь дескрипторов

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

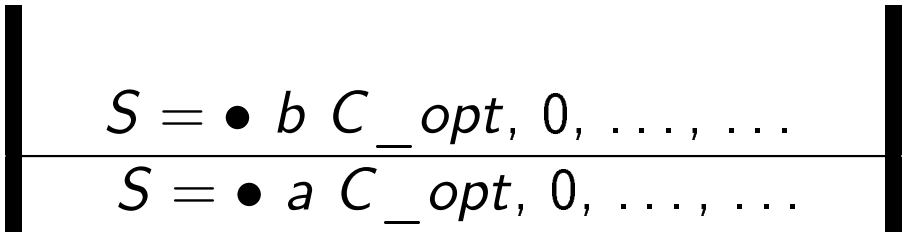
Пример постоения леса разбора в оригинальном алгоритме

Вход : $\bullet\ bc$

Грамматика:

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

Очередь дескрипторов



Пример постоения леса разбора в оригинальном алгоритме

Вход : $\bullet\ bc$

Грамматика:

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

Очередь дескрипторов

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

Пример постоения леса разбора в оригинальном алгоритме

Вход : $\bullet\ bc$

Грамматика:

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

Очередь дескрипторов

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

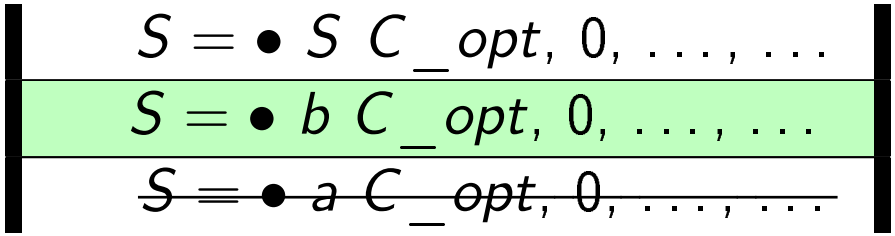
Пример постоения леса разбора в оригинальном алгоритме

Вход : $\bullet\ bc$

Грамматика:

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

Очередь дескрипторов



Пример постоения леса разбора в оригинальном алгоритме

Вход : $b \bullet c$

Грамматика:

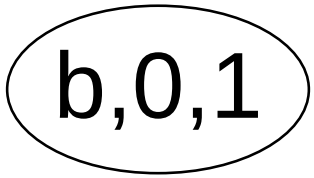
$$\begin{aligned} S = &\quad a\ C\_opt \\ \mid &\quad b \bullet\ C\_opt \\ \mid &\quad S\ C\_opt \\ C\_opt = &\quad \varepsilon \mid c \end{aligned}$$

Очередь дескрипторов

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

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

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



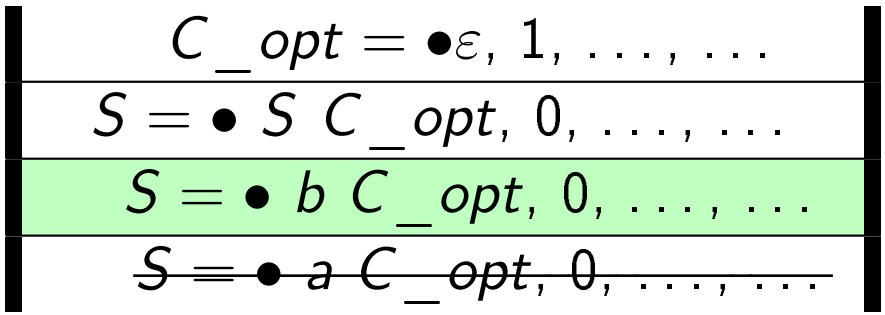
Пример постоения леса разбора в оригинальном алгоритме

Вход : $b \bullet c$

Грамматика:

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

Очередь дескрипторов



Пример постоения леса разбора в оригинальном алгоритме

Вход : $b \bullet c$

Грамматика:

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

Очередь дескрипторов

$C\_opt = \bullet c,\ 1, \ldots, \ldots$
$C\_opt = \bullet \varepsilon,\ 1, \ldots, \ldots$
$S = \bullet\ S\ C\_opt,\ 0, \ldots, \ldots$
$S = \bullet\ b\ C\_opt,\ 0, \ldots, \ldots$
~~$S = \bullet\ a\ C\_opt,\ 0, \ldots, \ldots$~~

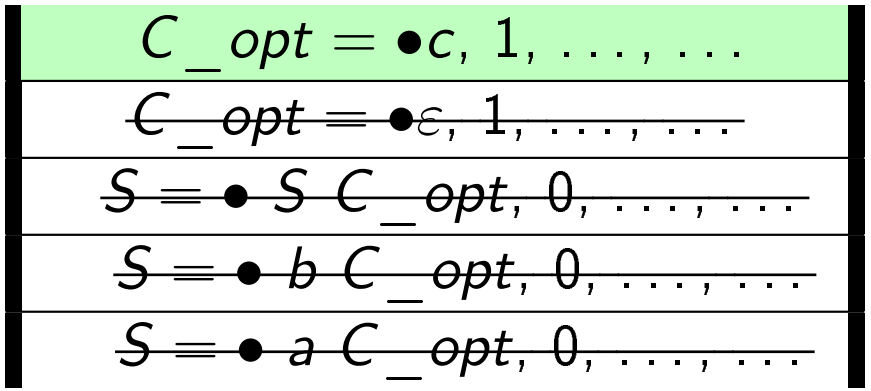
Пример постоения леса разбора в оригинальном алгоритме

Вход : $b \bullet c$

Грамматика:

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

Очередь дескрипторов



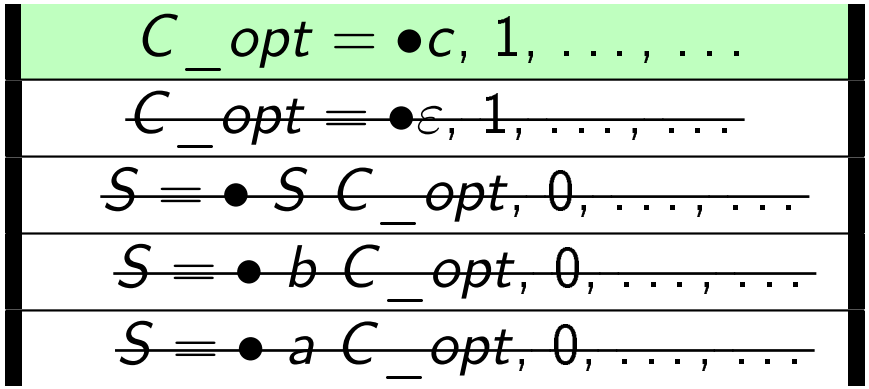
Пример постоения леса разбора в оригинальном алгоритме

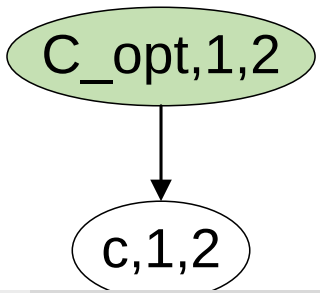
Вход : $bc\bullet$

Грамматика:

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

Очередь дескрипторов





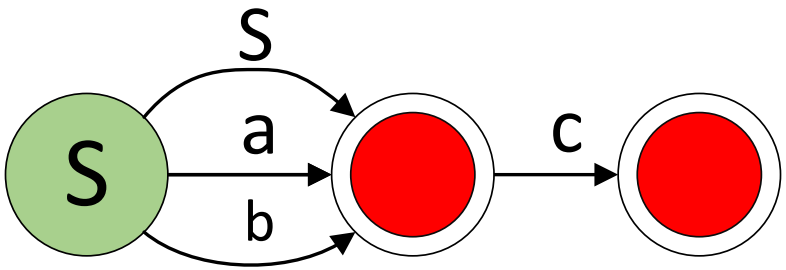
Постоение леса разбора по автомату

- Очередь дескрипторов
- Дескриптор (G, i, U, T) однозначно определяет состояние процесса разбора
  - G - ~~позиция в грамматике~~ состояние автомата
  - i - позиция во входе
  - U - узел стека разбора
  - T - корень построенного леса разбора

Пример постоения леса разбора по автомату

Вход : bc

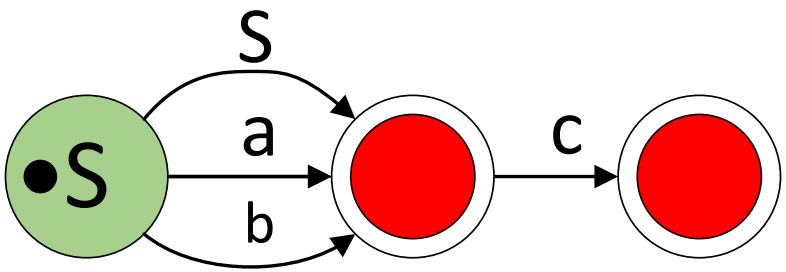
Автомат :



Пример постоения леса разбора по автомату

Вход : • bc

Автомат :



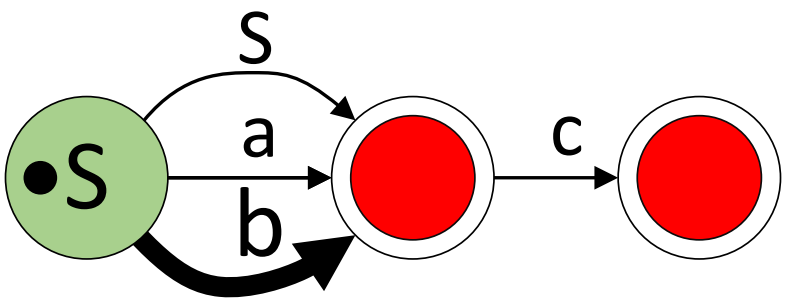
Очередь дескрипторов

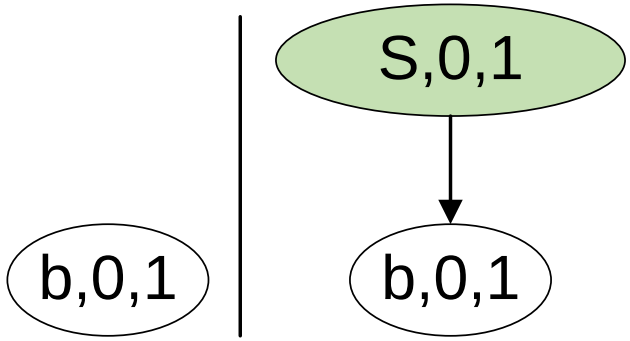
$S, 0, \ldots, \ldots$

Пример постоения леса разбора по автомату

Вход : • bc

Автомат :





Реализация

Алгоритм реализован в рамках проекта YaccConstructor

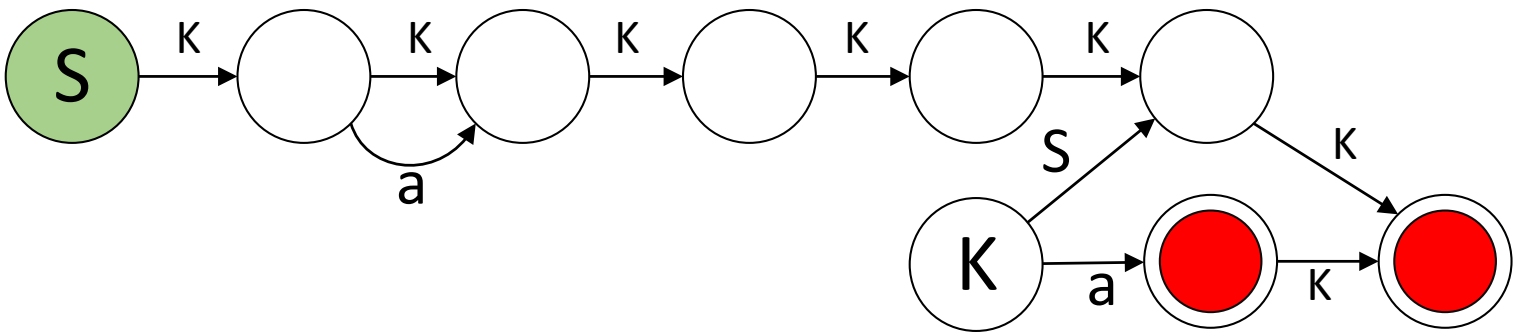
- Архитектура проекта модульная, поэтому понадобилось лишь встроить непосредственно генератор парсеров
- .net, *F#*

Эксперименты

Грамматика G_1

$$S = K\ K\ K\ K\ K\ K \mid K\ a\ K\ K\ K\ K$$
$$K = S\ K \mid a\ K \mid a$$

Рекурсивный автомат для грамматики G_1



Результаты экспериментов для входа a^{40}

	Использование памяти			Время, с
	Дескрипторы	Рёбра стека	узлы SPPF	
Грамматика	7,940	6,974	111,127,244	81
RA	5,830	4,234	74,292,078	54
Ratio	27%	39%	33 %	35 %

Поиск в метагеномных сборках

	Использование памяти			Время, мин
	Дескрипторы	Рёбра стека	Узлы стека	
Грамматика	21,134,080	7,482,789	2,731,529	02.26
RA	9,153,352	2,792,330	839,148	01.25
Ratio	57%	63%	69 %	45 %

Результаты

В рамках данной работы разработана и реализована модификация алгоритма GLL, работающая с расширенными контекстно-свободными грамматиками и показано, что полученный алгоритм повышает производительность поиска структур заданных с помощью контекстно-свободной грамматики в метагеномных сборках:

- В качестве подходящего представления ECFG предложены рекурсивные автоматы
- Спроектирована структура данных для представления леса разбора по ECFG на основе SPPF
- Разработан алгоритм на основе Generalized LL, строящий лес разбора по ECFG
- Алгоритм реализован в рамках проекта YaccConstructor
- Эксперименты показали двухкратный прирост производительности по сравнению с существуюшим решением
- Выступление на конференции "Инструменты и методы анализа программ"