

Санкт-Петербургский государственный университет

Кафедра системного программирования

Ковалев Дмитрий Александрович

Синтаксический анализ данных,  
представленных в виде  
контекстно-свободной грамматики

Выпускная квалификационная работа

Научный руководитель:  
к. ф.-м. н., доц. Григорьев С. В.

Рецензент:  
программист НИУ ИТМО Авдюхин Д. А.

Санкт-Петербург  
2017

SAINT PETERSBURG STATE UNIVERSITY

Chair of Software Engineering

Dmitry Kovalev

# Parsing data represented as context-free grammar

Graduation Project

Scientific supervisor:  
associate professor Semyon Grigorev

Reviewer:  
Programmer at ITMO University Avdiukhin Dmitrii

Saint-Petersburg  
2017

# Оглавление

<b>Введение</b>	<b>4</b>
<b>1. Постановка задачи</b>	<b>6</b>
<b>2. Обзор</b>	<b>7</b>
2.1. Рекурсивные автоматы и КС-грамматики . . . . .	7
2.2. Разрешимость задачи синтаксического анализа контекстно- свободного представления . . . . .	8
2.3. GLL-алгоритм и его модификации . . . . .	10
2.3.1. Оригинальный GLL-алгоритм . . . . .	10
2.3.2. Поддержка грамматик в EBNF . . . . .	11
2.3.3. Синтаксический анализ графов . . . . .	12
2.4. Проект YaccConstructor . . . . .	13
<b>3. Алгоритм синтаксического анализа     контекстно-свободного представления</b>	<b>14</b>
3.1. Описание алгоритма . . . . .	14
3.2. Доказательство корректности . . . . .	14
<b>4. Реализация и тестирование</b>	<b>15</b>
4.1. Архитектура предложенного решения . . . . .	15
4.2. Тестирование производительности . . . . .	15
<b>Заключение</b>	<b>16</b>
<b>Список литературы</b>	<b>17</b>
<b>Приложение</b>	<b>19</b>

# Введение

Контекстно-свободные грамматики, наряду с регулярными выражениями, активно используются для решения задач, связанных с разработкой формальных языков и синтаксических анализаторов. Одним из основных достоинств контекстно-свободных грамматик является возможность задания широкого класса языков при сохранении относительной компактности представления. Благодаря данному свойству, грамматики также представляют интерес в такой области информатики, как кодирование и сжатие данных. В частности, существует ряд алгоритмов, позволяющих производить сжатие текстовой информации, используя в качестве конечного [5] или промежуточного [2] представления контекстно-свободную грамматику (grammar-based compression).

Стандартной процедурой при работе с текстовыми данными является поиск в них определенных шаблонов, которые могут быть заданы строкой или регулярным выражением. В настоящее время большие объемы информации, как правило, хранятся и передаются по сети в сжатом виде, поэтому актуальной задачей становится поиск шаблонов непосредственно в компактном контекстно-свободном представлении текста. Такой подход позволяет избежать дополнительных затрат памяти на восстановление исходной формы данных и, в некоторых случаях, увеличивает скорость выполнения запроса. Шаблон здесь может быть, как и при поиске в обычном тексте, строкой (compressed pattern matching), сжатой строкой (fully compressed pattern matching) или регулярным выражением.

Известны ситуации, в которых для задания шаблона необходимо использовать более выразительные средства. Примером может служить одна из задач биоинформатики — поиск определенных подпоследовательностей в геноме организма. Так, для классификации и исследования образцов, полученных в результате секвенирования, в них могут искать гены, описывающие специфические рРНК. Структура таких генов, как правило, задается при помощи контекстно-свободной грамматики [6]. Для уменьшения объемов памяти, необходимых для хранения боль-

шого количества геномов, используются различные алгоритмы сжатия, в том числе основанные на получении контекстно-свободной структуры исходных последовательностей.

Задача поиска КС-шаблонов при использовании КС-представления данных формулируется следующим образом: необходимо найти все строки, принадлежащие пересечению двух языков, один из которых задается грамматикой шаблона, а второй представляет собой язык всех подстрок исходного множества строк, описываемого грамматикой, полученной в результате сжатия данных. Назовем такой поиск *синтаксическим анализом данных, представленных в виде КС-грамматики*. В общем случае задача неразрешима, так как сводится к задаче о проверке пересечения двух языков, порождаемых произвольными КС-грамматиками, на пустоту. Для постановки экспериментов в области биоинформатики необходимо точнее исследовать возможность проведения синтаксического анализа КС-представления и разработать прототип алгоритма, позволяющего решить данную задачу.

# 1. Постановка задачи

Целью данной работы является разработка алгоритма синтаксического анализа данных, представленных в виде контекстно-свободной грамматики. Для ее достижения были поставлены следующие задачи.

- Определить ограничения, при которых синтаксический анализ контекстно-свободного представления является разрешимой задачей.
- Разработать алгоритм синтаксического анализа КС-представления данных с учетом поставленных ограничений и доказать его корректность.
- Реализовать предложенный алгоритм.
- Провести тестирование и апробацию.

## 2. Обзор

Для того чтобы упростить ход рассуждений, касающихся контекстно-свободных грамматик, в данной работе используется понятие *рекурсивного автомата* — абстракции, позволяющей задавать произвольную КС-грамматику. Ее описание приводится в первом параграфе обзора. Второй параграф посвящен вопросу о разрешимости задачи синтаксического анализа контекстно-свободного представления данных и его связи с фундаментальными проблемами теории формальных языков.

Предлагаемый в данной работе алгоритм основан на алгоритме синтаксического анализа регулярных множеств, который, в свою очередь, является модификацией алгоритма обобщенного синтаксического анализа Generalized LL. Об этих алгоритмах, а также о проекте, в рамках которого проведена разработка предложенного решения, также будет рассказано в обзоре.

### 2.1. Рекурсивные автоматы и КС-грамматики

Введем понятие рекурсивного автомата, которое потребуется для дальнейшего изложения

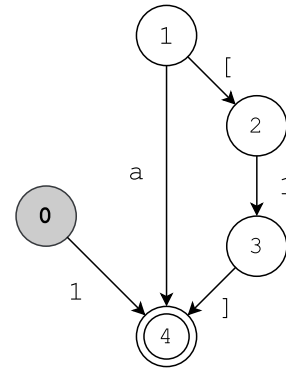
**Определение 1.** Рекурсивный автомат  $R$  — это пятерка  $(\Sigma, Q, \delta, q_0, q_f)$ , где  $\Sigma$  — конечное множество терминальных символов,  $Q$  — конечное множество состояний автомата,  $\delta : Q \times (\Sigma \cup Q) \rightarrow 2^Q$  — функция переходов,  $q_0 \in Q$  — начальное состояние,  $q_f$  — конечное состояние.

Можно заметить, что данное определение практически идентично определению стандартного конечного автомата. Единственное отличие состоит в том, что метками на ребрах рекурсивного автомата могут как терминальные символы (терминальные переходы), так и состояния (нетерминальные переходы). Класс рекурсивных автоматов обладает такой же выразительностью, как и контекстно-свободные грамматики, т.е. позволяет описать любой контекстно-свободный язык. Более того, грамматика тривиальным образом может быть преобразована в рекур-

сивный автомат (обратное тоже верно) [9]. Пример рекурсивного автомата, построенного по грамматике, можно увидеть на рисунке 1.

$$\begin{aligned} S' &::= S \\ S &::= [ S ] \\ S &::= a \end{aligned}$$

(a) Грамматика  $G_1$



(b) Рекурсивный автомат для  $G_1$

Рис. 1: Преобразование между грамматикой и рекурсивным автоматом

## 2.2. Разрешимость задачи синтаксического анализа контекстно-свободного представления

Как было сказано ранее, задачу поиска шаблона, при условии, что и шаблон, и данные, в которых осуществляется поиск, представлены контекстно-свободными грамматиками, мы назовем синтаксическим анализом контекстно-свободного представления.

Для доказательства предложений, сформулированных далее, будет использоваться следующая теорема [4].

**Теорема 1** (Nederhof, Satta). Пусть  $G_1$  — произвольная контекстно-свободная грамматика,  $G_2$  — грамматика, которая не содержит непосредственной или скрытой рекурсий. Тогда проблема проверки пустоты пересечения языков, порождаемых данными грамматиками, относится к классу *PSPACE-complete*.

Рассмотрим случай, когда грамматика данных задает ровно одну строку. Пусть  $G_t$  — произвольная КС-грамматика, задающая шаблоны для поиска, а  $G_d$  — КС-грамматика, которая не содержит непосредственной или скрытой рекурсий.  $L(G_t)$  и  $L(G_d)$  — языки, порождаемые грамматиками, при этом  $L(G_d) = \{\omega\}$ , где  $\omega$  — исходные данные, к



которым был применен алгоритм сжатия. Необходимо определить, существуют ли такие строки  $\omega'$ , что  $\omega' \in L(G_t)$  и  $\omega'$  — подстрока  $\omega$ .

**Предложение 1.** *При выполнении описанных условий задача синтаксического анализа КС-представления разрешима.*

*Доказательство.* Пользуясь эквивалентностью представлений, можно записать грамматику  $G_d$  в виде рекурсивного автомата  $R_d$ . Рассмотрим рекурсивный автомат  $R_{i,j}$ , полученный из  $R_d$  путем замены стартового состояния на  $i \in Q(R_d)$  и назначения терминирующего (финального, из которого не может быть совершено переходов) состояния  $j \in Q(R_d)$ . Такой автомат описывает грамматику, которая является представлением некоторой подстроки  $\omega$ . Рассмотрев все возможные пары  $i$  и  $j$ , получаем конечное множество грамматик, для каждой из которых необходимо проверить, содержится ли строка, порождаемая ей, в языке  $L(G_t)$ . Согласно теореме 1, такая проверка является разрешимой задачей и принадлежит к классу PSPACE-complete.  $\square$

Отдельно отметим, что для описанных процедур используется лишь исходный автомат, эквивалентный грамматике  $G_d$ . Условия задачи поиска шаблонов непосредственно в контекстно-свободном представлении, таким образом, выполняются. Верна также разрешимость более общей задачи.

**Предложение 2.** *Пусть грамматика  $G_d$  задает конечное множество строк  $L(G_d) = \{\omega_1, \dots, \omega_n\}$ . Необходимо определить, существуют ли строки  $\omega'$ , для которых верно:  $\omega' \in L(G_t)$  и  $\omega'$  — подстрока одной из строк  $\omega_i \in L(G_d)$ . Данная задача разрешима и принадлежит классу PSPACE-complete.*

*Доказательство.* Как и в предыдущем доказательстве, используем запись грамматики в виде рекурсивного автомата  $R_d$  и рассмотрим автоматы  $R_{i,j}$ . В данном случае каждый из этих автоматов представляет собой грамматику, которая порождает некоторое конечное множество подстрок исходных строк из  $L(G_d)$ . Проверка пустоты пересечения такой грамматики с  $G_t$  также соответствует условиям теоремы 1.  $\square$

В случае, когда грамматика  $G_d$  представляет собой бесконечный регулярный язык (т.е. содержит левую и/или правую рекурсию), разрешимость задачи поиска шаблонов установить не удастся. Подход, использованный ранее в доказательстве предложений, не может быть применен, так как части рекурсивного автомата, представляющего грамматику  $G_d$ , также могут содержать рекурсивные переходы, что выходит за рамки условия теоремы 1. Проверка разрешимости и определение класса сложности задачи проверки пустоты пересечения произвольной и регулярной КС-грамматик в настоящее время остаются открытыми проблемами [4].

## 2.3. GLL-алгоритм и его модификации

Классические алгоритмы нисходящего и восходящего синтаксического анализа предполагают использование грамматики, которая является в достаточной мере однозначной. В противном случае, управляющие таблицы анализаторов содержат конфликты, из-за чего нельзя гарантировать корректное поведение на любых входных данных. Для работы с сильно неоднозначными грамматиками используются алгоритмы *обобщенного синтаксического анализа*, которые позволяют рассмотреть все возможные пути разбора строки и построить соответствующие деревья вывода. Поиск шаблонов не требует наличия деревьев вывода, поэтому в дальнейшем алгоритмы синтаксического анализа рассматриваются только как механизм, позволяющий определить принадлежность строки языку.

### 2.3.1. Оригинальный GLL-алгоритм

Generalized LL (GLL) [8] — алгоритм, обобщающий идеи нисходящего синтаксического анализа. GLL, в отличие от стандартных LL-алгоритмов, позволяет использовать для анализа произвольную контекстно-свободную грамматику, в том числе содержащую леворекурсивные правила. Вместе с тем, GLL наследует такие полезные свойства алгоритмов нисходящего анализа, как непосредственная связь с

грамматикой и простота отладки и диагностики ошибок.

Для обработки неоднозначностей GLL разделяет стек анализатора на несколько ветвей, каждая из которых соответствует возможному пути разбора. При таком подходе необходимо компактное представление множества стеков, в качестве которого выступает Graph Structured Stack (GSS). В работе [1] была представлена модификация GSS, которая позволяет увеличить эффективность GLL-анализа. Вершины такого представления хранят в себе номер нетерминала и позицию в строке, с которой начался разбор подстроки, соответствующей ему. На ребрах хранятся позиции в грамматике (вида  $X \rightarrow \alpha A \cdot \beta$ ), на которые необходимо вернуться после завершения разбора нетерминала.

Основной идеей GLL является использование дескрипторов, позволяющих полностью описывать состояние анализатора в текущий момент времени.

**Определение 2.** Дескриптор — это тройка  $(L, u, i)$ , где

- $L$  — текущая позиция в грамматике вида  $A \rightarrow \alpha \cdot \beta$
- $u$  — текущая вершина GSS
- $i$  — позиция во входном потоке

В процессе работы поддерживается глобальная очередь дескрипторов. В начале каждого шага исполнения алгоритм берет следующий в очереди дескриптор и производит действия в зависимости от позиции в грамматике и текущего входного символа, передвигая соответствующие указатели. При наличии конфликтов в грамматике алгоритм добавляет дескрипторы для каждого возможного пути анализа в конец очереди.

### 2.3.2. Поддержка грамматик в EBNF

В работе Артема Горохова [3] была описана модификация GLL, которая позволяет использовать грамматики, записанные в расширенной форме Бэкуса-Наура (EBNF). Грамматика такого вида трансформируется в соответствующий рекурсивный автомат, в котором затем минимизируется количество состояний. Синтаксический анализ произво-

дится без построения управляющих таблиц: алгоритм обходит рекурсивный автомат в соответствии со входным потоком символов. При обработке текущего дескриптора  $(C_S, C_U, i)$ , где  $C_S$  — вершина автомата (эквивалент позиции в грамматике),  $C_U$  — вершина GSS,  $i$  — позиция в строке, могут возникать следующие ситуации.

- $C_S$  — финальное состояние. Показывает, что разбор текущего нетерминала был завершен. Необходимо осуществить возврат из  $C_U$  по меткам на исходящих из нее ребрах.
- Присутствует нетерминальный переход из  $C_S$ . В данном случае необходимо начать разбор указанного нетерминала  $X$ . Для этого в GSS должна быть создана новая вершина  $(X, i)$ , если она не создавалась ранее, а текущая вершина автомата изменена на стартовую для  $X$ .
- Присутствует терминальный переход из  $C_S$ . Необходимо сравнить терминал на ребре автомата с текущим входным символом. Если они совпадают, то осуществить переход в вершину автомата, на которую указывает ребро, и передвинуть указатель в строке.

За счет уменьшения количества состояния в автомате удастся достичь прироста в производительности по сравнению со стандартным GLL-алгоритмом.

### 2.3.3. Синтаксический анализ графов

Стандартными входными данными для алгоритмов синтаксического анализа являются линейные последовательности токенов. На основе GLL был разработан алгоритм, который позволяет производить синтаксический анализ регулярных множеств строк, представленных в виде конечного автомата (который, в свою очередь, является ориентированным графом с токенами на ребрах).

Поддержка нелинейного входа не потребовала существенных изменений в оригинальном алгоритме. Дескрипторы модифицированного

алгоритма хранят номер вершины входного графа вместо позиции в строке. Также, на шаге исполнения просматривается не единственный текущий символ, а множество символов на ребрах, исходящих из текущей вершины.

Производительность данного алгоритма, как и обычного GLL, может быть увеличена при помощи представления входной грамматики в виде рекурсивного автомата. В таком случае, алгоритм будет производить обход двух автоматов — рекурсивного и конечного. Ситуации, возникающие при обработке дескрипторов, не отличаются от описанных ранее ситуаций для линейного входа. Псевдокод данной модификации приведен в приложении.

## 2.4. Проект YaccConstructor

YaccConstructor — исследовательский проект лаборатории языковых инструментов JetBrains на математико-механическом факультете СПбГУ, направленный на исследования в области лексического и синтаксического анализа. Проект включает в себя одноименную модульную платформу для разработки лексических и синтаксических анализаторов, содержащую большое количество компонент: язык описания грамматик YARD, преобразования над грамматиками и др. Основным языком разработки является F#.

Ранее в рамках YaccConstructor были реализованы генераторы GLL-анализаторов, описание которых было приведено в данном обзоре.

## 3. Алгоритм синтаксического анализа контекстно-свободного представления

### 3.1. Описание алгоритма

// ЧЕРНОВИК

Мы пытаемся решать следующую задачу: пусть  $G_1 = (N, T, S, P)$  — произвольная КС-грамматика,  $G_2$  — NSE КС-грамматика. Алгоритм принимает на вход два рекурсивных автомата,  $M_1$  и  $M_2$ , построенных по грамматикам  $G_1$  и  $G_2$  соответственно, при этом в автомате  $M_2$  левые/правые рекурсивные вызовы заменены на циклы, как в обычном конечном автомате.

Результатом работы алгоритма являются тройки вида  $(X, n_1, n_2)$ , где  $X \in N$ ,  $n_1, n_2$  — номера состояний автомата  $M_2$ . Для каждой из таких троек выполняется следующее утверждение:  $\exists \omega \in T^*$  такая, что  $X \rightarrow^* \omega$  в  $G_1$  и  $\omega \in L(M')$ , где  $M'$  — рекурсивный автомат, полученный из  $M_2$  путем замены начального и конечного состояния на  $n_1$  и  $n_2$  соответственно.

Получая такие результаты, мы, по сути, отвечаем на вопрос о проверке пустоты пересечения КС-языка и регулярного, представленных в необычных абстракциях. Для КС-языка мы используем рекурсивный автомат, а для регулярного — нечто среднее между конечным автоматом и NSE грамматикой (это нечто все еще использует стек, но только для обработки нерекурсивных вызовов). Такое представление эквивалентно по выразительности NSE и, следовательно, FA (см. рис. ??).

### 3.2. Доказательство корректности

*Теорема 2. Завершаемость. Алгоритм завершает работу за конечное число шагов для произвольных входных данных*

*Теорема 3. Корректность. ???*

## **4. Реализация и тестирование**

### **4.1. Архитектура предложенного решения**

todo

### **4.2. Тестирование производительности**

todo

# Заключение

В ходе данной работы получены следующие результаты.

- Определены ограничения, при которых синтаксический анализ контекстно-свободного представления является разрешимой задачей.
- Разработан алгоритм синтаксического анализа КС-представления, учитывающий поставленные ограничения, и доказана его корректность. Полученный алгоритм является модификацией алгоритма синтаксического анализа графов на основе GLL.
- Предложенный алгоритм реализован на языке программирования F# в рамках исследовательского проекта YaccConstructor.
- Проведена апробация: тестирование производительности и тестирование на синтетических данных.

В дальнейшем планируется провести апробацию на реальных данных (метагеномных сборках) и доказать теоретическую оценку сложности алгоритма.

Исходный код проекта YaccConstructor может быть найден на сайте <https://github.com/YaccConstructor/YaccConstructor>.



## Список литературы

- [1] Afroozeh Ali, Izmaylova Anastasia. Faster, Practical GLL Parsing // Compiler Construction: 24th International Conference, CC 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings / Ed. by Björn Franke. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2015. — P. 89–108. — ISBN: 978-3-662-46663-6. — URL: [http://dx.doi.org/10.1007/978-3-662-46663-6\\_5](http://dx.doi.org/10.1007/978-3-662-46663-6_5).
- [2] Arimura Mitsuharu. A grammar-based compression using a variation of Chomsky normal form of context free grammar // 2016 International Symposium on Information Theory and Its Applications (ISITA). — 2016.
- [3] Gorokhov Artem, Grigorev Semyon. Extended Context-Free Grammars Parsing with Generalized LL. — 2017.
- [4] Nederhof Mark-Jan, Satta Giorgio. The Language Intersection Problem for Non-recursive Context-free Grammars // Inf. Comput. — 2004. — Aug. — Vol. 192, no. 2. — P. 172–184. — URL: <http://dx.doi.org/10.1016/j.ic.2004.03.004>.
- [5] Nevill-Manning Craig G., Witten Ian H. Identifying Hierarchical Structure in Sequences: A Linear-time Algorithm // J. Artif. Int. Res. — 1997. — Sep. — Vol. 7, no. 1. — P. 67–82. — URL: <http://dl.acm.org/citation.cfm?id=1622776.1622780>.
- [6] Quantifying variances in comparative RNA secondary structure prediction / James WJ Anderson, Ádám Novák, Zsuzsanna Sükösd et al. // BMC Bioinformatics. — 2013. — Vol. 14, no. 1. — P. 149. — URL: <http://dx.doi.org/10.1186/1471-2105-14-149>.
- [7] Ragozina Anastasiya. GLL-based relaxed parsing of dynamically generated code. — 2016.

- [8] Scott Elizabeth, Johnstone Adrian. GLL parsing // Electronic Notes in Theoretical Computer Science. — 2009. — Vol. 253, no. 7.
- [9] Tellier Isabelle. Learning recursive automata from positive examples // Revue des Sciences et Technologies de l'Information-Série RIA: Revue d'Intelligence Artificielle. — 2006. — Vol. 20, no. 6. — P. 775–804.

# Приложение

## Псевдокод алгоритма синтаксического анализа графов

```
function CREATE( $S_{call}, S_{next}, u, i, w$ )  
   $A \leftarrow \Delta(S_{call})$   
  if ( $\exists$  GSS node labeled  $(A, i)$ ) then  
     $v \leftarrow$  GSS node labeled  $(A, i)$   
    if (there is no GSS edge from  $v$  to  $u$  labeled  $(S_{next}, w)$ ) then  
      add GSS edge from  $v$  to  $u$  labeled  $(S_{next}, w)$   
      for  $((v, z) \in \mathcal{P})$  do  
         $(y, N) \leftarrow \text{getNodes}(S_{next}, u.nonterm, w, z)$   
         $(\_, \_, h) \leftarrow y$   
        add( $S_{next}, u, h, y$ )  
        if  $N \neq \$$  then  
           $(\_, \_, h) \leftarrow N$ ; pop( $u, h, N$ )  
  else  
     $v \leftarrow$  new GSS node labeled  $(A, i)$   
    create GSS edge from  $v$  to  $u$  labeled  $(S_{next}, w)$   
    add( $S_{call}, v, i, \$$ )  
  return  $v$   
  
function POP( $u, i, z$ )  
  if  $((u, z) \notin \mathcal{P})$  then  
     $\mathcal{P}.add(u, z)$   
    for all GSS edges  $(u, S, w, v)$  do  
       $(y, N) \leftarrow \text{getNodes}(S, v.nonterm, w, z)$   
      add( $S, v, i, y$ )  
      if  $N \neq \$$  then pop( $v, i, N$ )  
  
function PARSE  
   $R.enqueue(StartState, newGSSnode(StartNonterminal, 0), 0, \$)$   
  while  $R \neq \emptyset$  do  
     $(C_S, C_U, i, C_N) \leftarrow R.dequeue()$ 
```

```

if ( $C_N = \$$ ) and ( $C_S$  is final state) then
   $eps \leftarrow \text{getNodeT}(\varepsilon, i)$ 
   $(\_, N) \leftarrow \text{getNodes}(C_S, C_U.\text{nonterm}, \$, eps)$ 
  pop( $C_U, i, N$ )
for each  $transition(C_S, label, S_{next})$  do
  switch  $label$  do
    case  $Terminal(x)$  where ( $x = input[i]$ )
       $T \leftarrow \text{getNodeT}(x, i)$ 
       $(y, N) \leftarrow \text{getNodes}(S_{next}, C_U.\text{nonterm}, C_N, T)$ 
      if  $N \neq \$$  then pop( $C_U, i + 1, N$ )
      if  $S_{next}$  have multiple ingoing transitions then
        add( $S_{next}, C_U, i + 1, y$ )
      else
         $R.\text{enqueue}(S_{next}, C_U, i + 1, y)$ 
    case  $Nonterminal(S_{call})$ 
      create( $S_{call}, S_{next}, C_U, i, C_N$ )
if SPPF node ( $StartNonterminal, 0, input.length$ ) exists then
  return this node
else report failure

```