Санкт-Петербургский государственный университет

Кафедра системного программирования

Горохов Артем Владимирович

Поддержка конъюнктивных грамматик в GLL

Курсовая работа

Научный руководитель:
магистр ИТ, ст. преп. Григорьев С. В.

Санкт-Петербург
2016

Оглавление

Аннотация

В работе представлена модернизация реализации алгортма синтаксического анализа регулярных мнрожеств, основанного на алгоритме синтаксического анализа GLL. Модернизация позволяет расширить класс распознаваемых анализатором языков до конъюнктивных, что полезно в некоторых задачах, например синтаксический анализ метагеномных сборок.

Введение

Синтаксический анализ, как правило, используется для построения структурного представления кода с использованием грамматики, описывающей разбираемый язык. Абстрактное синтаксическое дерево, являющееся результатом работы синтаксического анализатора, в дальнейшем используется для проведения статического анализа кода или же в каких-то других целях. Как правило, на вход синтаксическому анализатору подаётся линейная последовательность токенов, представляющая код программы. Однако могут возникать ситуации, когда вход не может быть представлен линейно. Такие ситуации могут возникать, например, при автоматической генерации кода. Генерация может происходить в циклах, с использованием условных операторов или строковых операций. Поэтому для описания генерируемых цепочек можно использовать конечный автомат, порождающий цепочки, который уже не будет являться линейным. Такую задачу будем называть синтаксическим анализом регулярных множеств.

Кроме этого, ещё одной областью, где может быть применим синтаксический анализ регулярных множеств является бионформатика. Одной из часто возникающих задач в биоинформатике является классификация организмов, находящихся в образцах, полученных из окружающей среды [5]. По образцам строится метагеномная сборка, которая содержит в себе смесь из РНК всех содержащихся в сборке организмов. В свою очередь РНК является последовательностью символов в алфавите {A, C, G, T}. РНК организмов, которые относятся к одному и тому же виду, содержат одинаковые подцепочки, которые и необходимо выделить, чтобы классифицировать организм. Как правило, эти подцепочки — это последовательности РНК. РНК может быть описана с помощью грамматики. Метагеномная сборка, в свою очередь, может быть представлена в виде графа с цепочками на рёбрах. Таким образом, в таком графе необходимо найти цепочки, выводимые в грамматике, описывающей РНК.

Грамматики, описывающие структуру РНК, являются неоднознач-

ными. Грамматика называется неоднозначной, если одна и та же цепочка может быть выведена несколькими способами. Такие алгоритмы синтаксического анализа как LR и LL не позволяют обрабатывать неоднозначные грамматики. Для работы с неоднозначными грамматиками существуют алгоритмы обобщённого синтаксического анализа GLR [10], GLL [8]. В рамках проекта YaccConstructor [12, 14] был реализован алгоритм GLL, кроме того, была предложена его модификация для обработки нелинейных входных данных — графов. Реализованная модификация позволяет находить цепочки транспортной РНК(тРНК) в небольших метагеномных сборках, возвращая координаты начала и конца найденной цепочки. Проблема заключается в том, что грамматика для описания тРНК является сильно неоднозначной, что сказывается на производительности и точности полученных результатов. Для повышения точности можно применять конъюнктивные грамматики [4], в которых для описания продукций используется операция конъюнкции. Такие грамматики расширяют класс контекстно-свободных языков и позволяют точнее описать структуру тРНК. Данная работа посвящена описанию модификаций решения на основе алгоритма GLL для работы с конъюнктивными грамматиками.

1. Обзор предметной области

Одной из задач, часто возникающих в биоинформатике, является классификация организмов в образцах, полученных из окружающей среды. Из образцов извлекается смесь РНК всех организмов, которая представляется в виде метагеномнонй сборки. Метагеномная сборка, в свою очередь, может быть представлена в виде конечного автомата, порождающего геномы всех организмов из образца. Для того, чтобы определить, к какому виду относится организм, нужно выделить РНК. Для поиска РНК в метагеномных сборках существуют различные подходы. Некоторые из них используют скрытые модели Маркова [9] для поиска, например, инструмент REAGO [6]. Минусом инструмента является то, что он не работает с метагеномной сборкой, представленной в виде графа, а представление сборки в другом виде требует слишком больших объёмов памяти. Инструмент Xander [11] позволяет работать со сборками, представленными в виде графов, но в основе лежит механизм, обладающий низкой точностью. Синтаксический анализ, также, применяется для анализа метагеномных сборок, например, в инструменте Infernal [3], который не предназначен для работы с графами.

Грамматика, описывающая РНК является сильно неоднозначной и её не всегда можно привести к однозначной форме. Для работы с неоднозначными грамматиками используются алгоритмы обобщённого синтаксического анализа. Принцип работы таких алгоритмов заключается в том, что они просматривают все возможные пути вывода входной цепочки и строят все деревья вывода этой цепочки. Существует инструмент SBP [2] основанный на алгоритме GLR, позволяющий работать с конъюнтивными грамматиками. В данной работе используется алгоритм GLL, так как в среднем он работает быстрее. Алгоритм обобщённого анализа GLL основан на нисходящем анализе и отличается высокой скоростью работы и простотой. В алгоритме для хранения всех деревьев вывода используется структура данных SPPF (Shared Packed Parse Forest) [7]. Эта структура данных позволяет переиспользовать узлы, с одинаковыми поддеревьями под ними. На рисунке 1 показано,

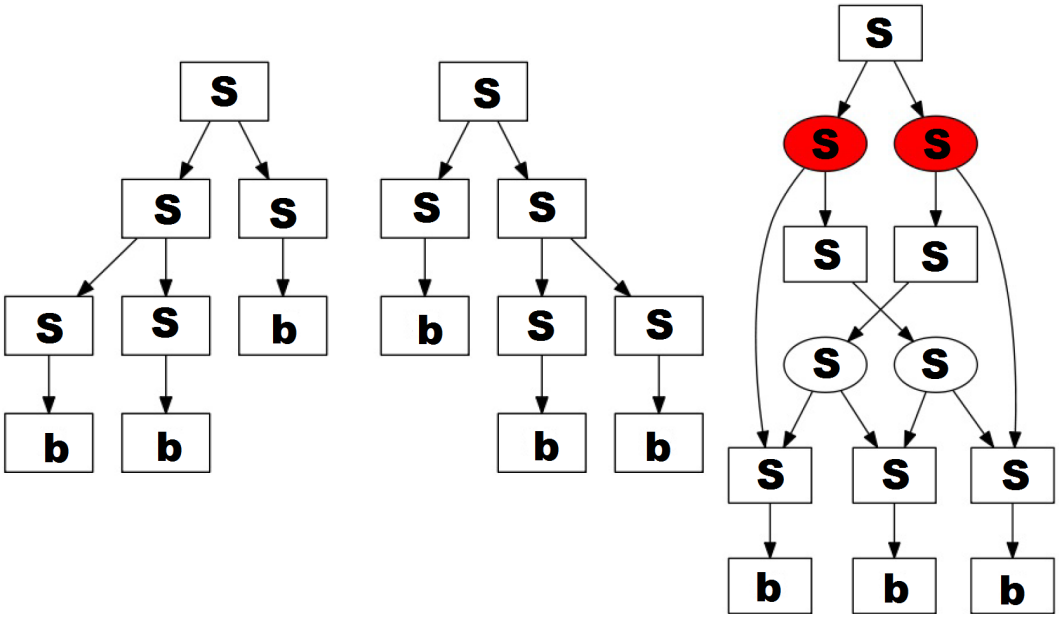

Рис. 1: Преобразование дерева разбора к SPPF

$$
\begin{array}{lcl}
\texttt{S} & ::= & \texttt{A B \& D C} \\
\texttt{A} & ::= & \texttt{a A} \mid \epsilon \\
\texttt{B} & ::= & \texttt{b B c} \mid \epsilon \\
\texttt{C} & ::= & \texttt{c C} \mid \epsilon \\
\texttt{D} & ::= & \texttt{a D b} \mid \epsilon
\end{array}
$$

Рис. 2: Грамматика для контекстно-зависимого языка $\{a^n b^n c^n, n \geq 0\}$

как объединяются разные выводы нетерминала S. Создаются дополнительные узлы, соответствующие каждому из выводов. Выделяются одинаковые поддеревья и остаётся только один экземпляр каждого, на который, в дальнейшем, ссылаются предки.

Алгоритм GLL позволяет работать с любыми КС-грамматиками, в том числе и сильно неоднозначными. Однако наличие сильной неоднозначности в грамматике сказывается на точности получаемых результатов и производительности. Конъюнктивные грамматики при описании правил вывода используют операцию конъюнкции. Цепочка принадлежит языку, задаваемому такой грамматикой, если существует вывод по обоим конъюнктам. На рисунке 2 изображена грамматика для языка не являющегося контекстно-свободным, описанная с помощью операции конъюнкции. Такие грамматики дают возможность точно описать

структуру тРНК, что позволяет снизить количество ошибок при разборе.

2. Постановка задачи

Целью данной работы является добавление поддержки конъюнктивных грамматик в YaccConstructor. Для её достижения были поставлены следующие задачи:

- реализовать поддержку конъюнктивных грамматик в языке спецификации грамматик YARD;
- реализовать поддержку конъюнктивных грамматик в генераторе GLL-анализаторов;
- провести экспериментальные исследования работы алгоритма.

3. Основная часть

В данном разделе рассмотреннно расширение языка YARD и модификация алгоритма синтаксического анализа GLL.

3.1. Архитектура

Ввиду модульной структуры проекта YaccConstructor, задействованную структуру проекта можно разделить на части, показанные на рисунке 3.

В проекте используется язык описания грамматик YARD [13], который поддерживает различные конструкции, упрощающие разработку грамматик: повторения `x*[1..10]`, дизъюнкции `A|B`, условное вхождние и подобные. Перед подачей генератору, дерево разбора грамматики проходит через множество преобразований, в результате которых, грамматика приводится к форме Бэкуса-Наура [1].

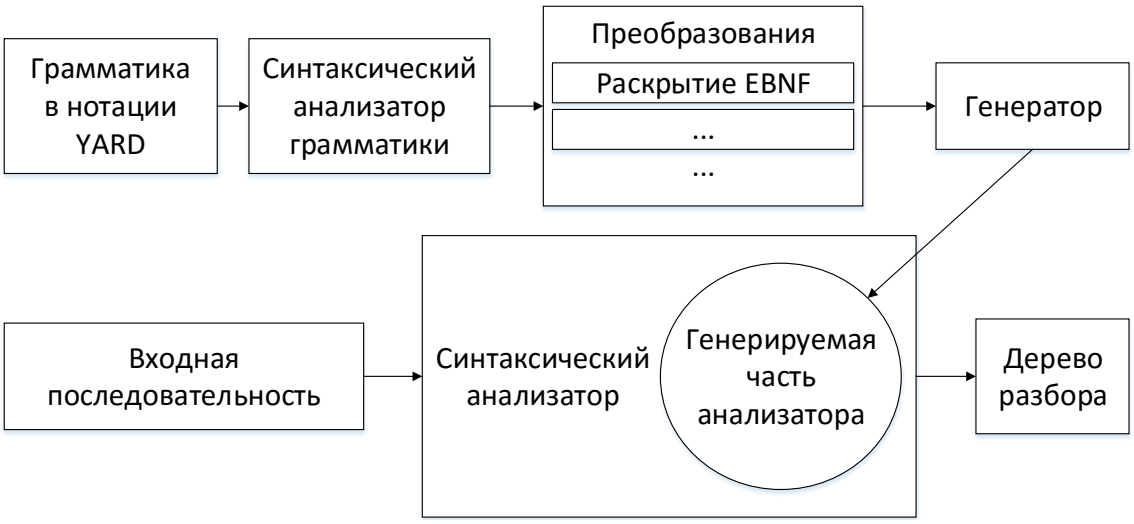

Рис. 3: Структура существующего решения

3.2. Расширение YARD

Для работы с конъюнктивными грамматиками в язык YARD была добавлена новая конструкция: конъюнкция, приоритет которой между

```
S  ::=  A D & B
A  ::=  A a |ϵ
D  ::=  A d |ϵ
B  ::=  A b D |ϵ
```

Рис. 4: Грамматика G_0

```
     S  ::=  Conj0
 Conj0  ::=  Conj0_0 | Conj0_1
Conj0_0 ::=  A D
Conj0_1 ::=  B
     A  ::=  A a |ϵ
     D  ::=  A d |ϵ
     B  ::=  A b D |ϵ
```

Рис. 5: Грамматика G_1

дизъюнкцией и последовательностью. Язык YARD поддерживает различные конструкции вроде повторений (x*[1..10]), которые преобразуются перед подачей генератору. Так как в язык была добавлена новая конструкция, нужно было обеспечить её поддержку во всех существующих преобразованиях.

Кроме того, добавленную конструкцию также необходимо преобразовывать к виду, принимаемому генератором. Предлагается следующее решение: грамматика преобразовывается к контекстно-свободной, с дополнительной информацией о существовании конъюнкции. Для каждой конъюнкции в грамматике создаётся три новых правила: по одному на каждый конъюнкт и одно на дизъюнкцию конъюнктов, в исходном правиле конъюнкция заменяется ссылкой на правило с дизъюнкцией конъюнктов. На рисунках 4 и 5 показан пример начальной грамматики (G_0) и результат её преобразования (G_1). В правиле `S` содержится конъюнкция, которая заменяется на ссылку на сгенерированное правило `Conj0`. Для каждого из конъюнктов также генерируется правило и продукция правила `Conj0` представляет собой выбор между ними.

При этом имена правил генерируются так, что в дальнейшем правила, заменившие конъюнкцию, можно определять по имени.

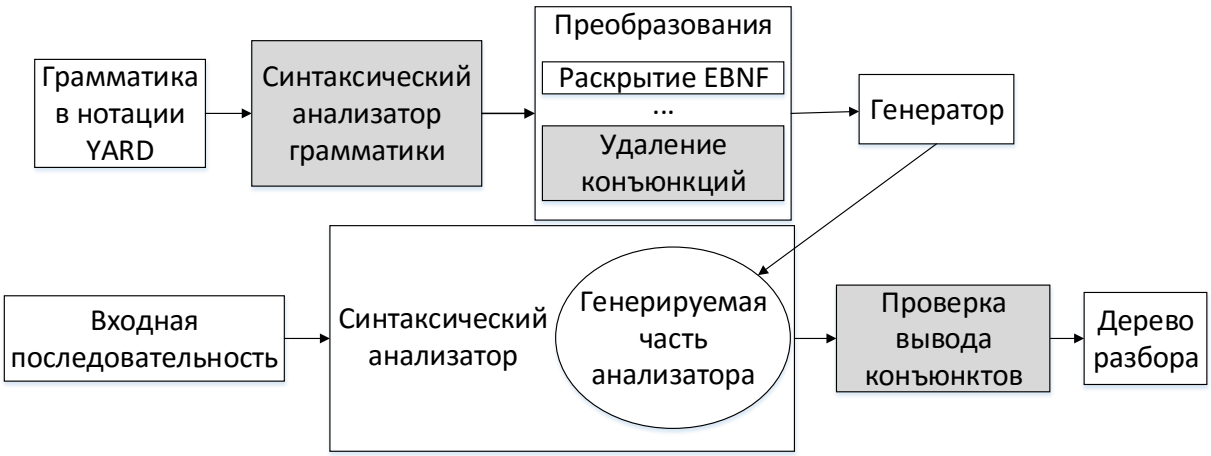

Рис. 6: Структура решения с учётом внесённых изменений

3.3. Модификация алгоритма GLL

После построения SPPF синтаксическим анализатором, нужно убедиться в том, что у правил с именем, соответствующим конъюнкции, есть вывод по обоим продукциям, иначе ветвь вывода нужно исключить из результатов разбора. Заметим, что невозможно проверять это в процессе разбора, т.к. нет возможности отследить, когда построятся все возможные выводы нетерминала.

Данная задача решается с помощью рекурсивного обхода дерева в глубину. Для каждого узла проверяется корректность вывода его потомков. Затем, если узел является нетерминальным, проверяется имя нетерминала, которому он соответствует, если оно является сгенерированным именем правила конъюнкции, то, при наличии 2 выводов нетерминала, поддерево, начиная с текущего нетерминала, считается корректным.

Таким образом время работы алгоритма возрастает на время, необходимое для обхода дерева разбора. Так как размер дерева не превышает кубического от длины входных данных, то сложность алгоритма (в худшем случае) остаётся прежней $O(n^3)$.

4. Эксперименты

Были проведены экспериментальные исследования, целью которых являлась проверка того, что конъюнктивные грамматики позволяют задавать структуру тРНК так, что синтаксический анализатор находит меньше некорректных цепочек.

```
[<Start>]
folded: stem<(any*[1..3]
              stem<any*[7..10]>
              any*[1..3]
              stem<any*[5..8]>
              any*[3..5]
              stem<any*[5..8]>
              )>

stem<s>:
      A stem<s> U
    | U stem<s> A
    | C stem<s> G
    | G stem<s> C
    | G stem<s> U
    | U stem<s> G
    | s

any: A | U | G | C
```

Рис. 7: КС-грамматика вторичной структуры тРНК

	КС-грамматика	Конъюнктивная грамматика
Тест 1. Кол-во ошибок:	15	0
Тест 2. Кол-во ошибок:	5	0
Тест 3. Кол-во ошибок:	11	0

Таблица 1: Количество некорректных цепочек, распознанных синтаксическим анализатором

На рисунках 7 и 8 представлены грамматки, описывающие структу-

```
[<Start>]
folded: stem<subseq> & (any*[7..9] subseq any*[7..9])

subseq: any*[1..3]
        stem<any*[7..10]> & (any*[4..6] any*[7..10] any*[4..6])
        any*[1..3]
        stem<any*[5..8]> & (any*[6] any*[5..8] any*[6])
        any*[3..5]
        stem<any*[5..8]> & (any*[4..5] any*[5..8] any*[4..5])

stem<s>:
      A stem<s> U
    | U stem<s> A
    | C stem<s> G
    | G stem<s> C
    | G stem<s> U
    | U stem<s> G
    | s

any: A | U | G | C
```

Рис. 8: Конъюнктивная грамматика структуры тРНК

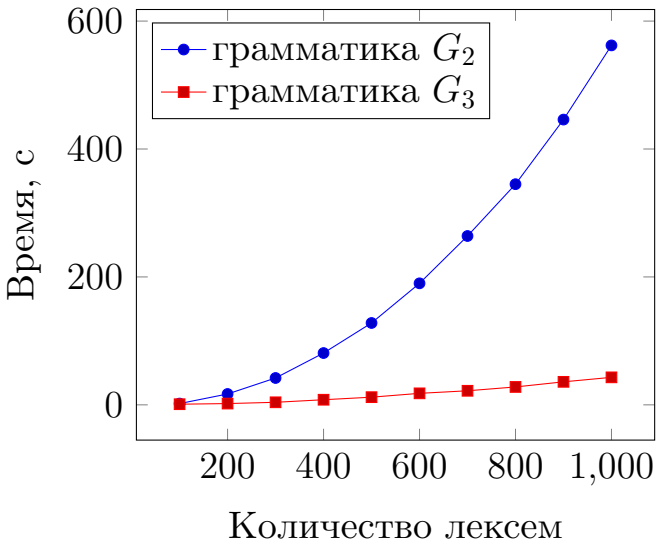

Рис. 9: Среднее время работы алгоритма на конъюнктивной и контекстно-свободной грамматиках тРНК

ACACCCCCCCCUCACCCCCUCCCACCCCCUU

Рис. 10: Пример цепочки нуклеотидов, сгенеированной для экспериментов

ру тРНК. Грамматика G_2 является контекстно-свободной, а грамматика G_3 — конъюнктивной. По данным грамматикам были сгенерированы соответствующие синтаксические анализаторы.

На вход построенным синтаксическим анализаторам подавались сгенерированные цепочки ДНК длиной от 100 до 1000 симовлов. Эти цепочки содержали в себе последовательности тРНК, а также другие последовательности, которые можно ложно признать за тРНК. Напимер, цепочка на рисунке 10, хоть и не является тРНК, распознаётся грамматикой G_2, но не распознаётся грамматикой G_3.

Результаты экспериментов приведены в таблице 1. Из них ясно, что грамматика G_2 не распознаёт ложные цепочки, распознавыемые грамматикой G_3. Время работы синтаксических анализаторов показано на графике, изображённом на рисунке 9. По графику видно, что время работы синтаксического анализатора, построенного по грамматике G_3, значительно превышает время работы другого.

Таким образом, конъюнктивная грамматика позволяет отсеивать цепочки, ложно распознаваемые КС-грамматикой, но за время, значительно большее, чем время работы анализатора по КС-грамматике.

Заключение

В ходе работы получены следующие результаты:

- реализована поддержка конъюнктивных грамматик в языке спецификации грамматик YARD;
- реализована поддержка конъюнктивных грамматик в генераторе GLL-анализаторов;
- результаты экспериментально проверены на небольших метагеномных сборках.

Дальнейшее направление работ

В первую очередь, необходимо снизить время работы алгоритма. Полученная реализация предполагает полный обход дерева разбора, что, безусловно, влияет на производительность алгоритма. Кроме того, можно исследовать возможность расширения класса распознаваемых языков до булевых, на основе полученных результатов.

Список литературы

[1] Crocker David, Overell Paul. Augmented BNF for syntax specifications: ABNF. — 2005.

[2] Megacz Adam. Scannerless boolean parsing // Electronic Notes in Theoretical Computer Science. — 2006. — Vol. 164, no. 2. — P. 97–102.

[3] Nawrocki Eric P, Eddy Sean R. Infernal 1.1: 100-fold faster RNA homology searches // Bioinformatics. — 2013. — Vol. 29, no. 22. — P. 2933–2935.

[4] Okhotin Alexander. Conjunctive grammars // Journal of Automata, Languages and Combinatorics. — 2001. — Vol. 6, no. 4. — P. 519–535.

[5] 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.

[6] Reconstructing 16S rRNA genes in metagenomic data / Cheng Yuan, Jikai Lei, James Cole, Yanni Sun // Bioinformatics. — 2015. — Vol. 31, no. 12. — P. i35–i43.

[7] Rekers Joan Gerard. Parser generation for interactive environments : Ph. D. thesis / Joan Gerard Rekers ; Citeseer. — 1992.

[8] Scott Elizabeth, Johnstone Adrian. GLL parsing // Electronic Notes in Theoretical Computer Science. — 2010. — Vol. 253, no. 7. — P. 177–189.

[9] Stamp Mark. A revealing introduction to hidden Markov models // Department of Computer Science San Jose State University.

[10] Tomita Masaru. Generalized LR parsing. — Springer Science & Business Media, 2012.

[11] Xander: employing a novel method for efficient gene-targeted metagenomic assembly / Qiong Wang, Jordan A Fish, Mariah Gilman et al. // Microbiome. — 2015. — Vol. 3, no. 1. — P. 1.

[12] YaccConstructor. YaccConstructor // YaccConstructor official page. — URL: `http://yaccconstructor.github.io` (online; accessed: 24.05.2016).

[13] YaccConstructor. YARD // YaccConstructor official page. — 2015. — URL: `http://yaccconstructor.github.io/YaccConstructor/yard.html` (online; accessed: 24.05.2016).

[14] Кириленко ЯА, Григорьев СВ, Авдюхин ДА. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем // Научно-технические ведомости СПбГПУ: информатика, телекоммуникации, управление. — 2013. — no. 174. — P. 94–98.