

Теория формальных языков. Лекции и практики. Заметки.

Семён Григорьев

7 сентября 2021 г.

Содержание

1	Правила работы на курсе	4
1.1	Оценка за курс	4
1.2	Домашние задачи	5
1.3	Летучки	6
2	План занятий	7
3	Лекция 1: Введение	10
4	Практика 1	12
4.1	Григорьев С.В.	12
5	Лекция 2: Регулярные языки	13
6	Практика 2	14
6.1	Григорьев С.В.	14
7	Лекция 3. Контекстно-свободные грамматики	14
8	Практика 3	14
8.1	Григорьев С.В.	14
9	Лекция 4	15
10	Практика 4	15
10.1	Григорьев С.В.	15
11	Лекция 5	15
11.1	Нормальная форма Хомского (НФХ)	15
11.2	Лемма о накачке для КС языков	16
12	Практика 5	17
12.1	Григорьев С.В.	17
13	Лекция 6	18
13.1	Свойства замкнутости КС языков	18
13.2	Алгоритм СУК	18
14	Практика 6	18
14.1	Григорьев С.В.	18

15 Лекция 7	20
15.1 Алгоритм на основе матричного произведения	20
15.2 Алгоритм на основе тензорного произведения	20
16 Практика 7	20
16.1 Григорьев С.В.	20
17 Лекция 8	21
18 Практика 8	21
18.1 Григорьев С.В.	21
18.1.1 Лексический анализ	21
18.1.2 Абстрактный и конкретный синтаксис	22
19 Лекция 9	24
20 Практика 9	24
20.1 Григорьев С.В.	24
21 Лекция 10	29
22 Практика 10	29
22.1 Григорьев С.В.	29
23 Лекция 11	29
24 Практика 11	30
24.1 Григорьев С.В.	30
25 Лекция 12	30
26 Практика 12	30
26.1 Григорьев С.В.	30
27 Лекция 13	30
28 Практика 11	31
28.1 Григорьев С.В.	31

1 Правила работы на курсе

Коротко зафиксируем основные правила работы на курсе: из чего формируется оценка, как сдавать домашние работы и т.д.

1.1 Оценка за курс

Оценка за курс складывается из баллов, полученных за работу в семестре. Баллы начисляются за следующее.

- За домашние работы (балл за каждую задачу указывается отдельно). При этом у каждой работы есть жёсткий дедлайн и после него балл уменьшается вдвое.
- За летучки (короткие, 5-10 минут, контрольные работы). Летучка оценивается от 1 до 0 баллов с шагом 0.25. Сами по себе баллы за летучки не суммируются, но служат для корректировки баллов за домашние работы.

Итоговая оценка за курс — это взвешенная сумма баллов за задачи, где вес — баллы за ближайшую справа летучку. Пусть, например, в курсе было 4 задачи и 2 летучки, упорядоченные хронологически как это показано в таблице 1. В этой же таблице можно увидеть, какие баллы получил Эталон Примерович за задачи и летучки, а также его итоговый балл, который вычисляется следующим образом:

$$\underbrace{(4 + 5)}_{\text{Первые две домашки}} * \underbrace{0.25}_{\text{Летучка 1}} + \underbrace{(2.5 + 3)}_{\text{Вторые две домашки}} * \underbrace{0.75}_{\text{Летучка 2}} = 6.375.$$

Таблица 1: Пример оценок за курс

ФИО	ДЗ 1	ДЗ 2	Летучка 1	ДЗ 3	ДЗ 4	Летучка 2	Итог
Эталон Примерович	4	5	0.25	2.5	3	0.75	6.375

Для трансляции баллов в оценку используется таблица 2.

Таблица 2: Конвертация баллов в оценки

Балл	ECTS	Классика
91–100	A	5
81–90	B	4
71–80	C	4
61–70	D	3
51–60	E	3
0–50	F	2

Решение задач можно продолжать до тех пор, пока не будет набрано достаточно баллов для получения неотрицательной оценки (с учётом понижения баллов после дедлайнов и с учётом летучек¹). Если даже при всех решённых задачах баллов не достаточно для получения

¹Поэтому летучки имеет смысл писать всегда. Они могут влияют на задачи, даже если те сданы после написания летучки.

неотрицательной оценки, то выдаются дополнительные задачи². Баллы за дополнительные задачи таковы, чтобы обеспечить разве что получение минимальной положительной оценки.

Все текущие результаты оформляются в виде таблицы на Google Drive. Таблица доступна всем студентам на чтение.

1.2 Домашние задачи

Домашние задачи представляют из себя практические задачи на программирование или постановку экспериментов (сравнение и анализ производительности различных алгоритмов, решений). Они анонсируются по ходу семестра вместе с баллами и дедлайнами. У задачи есть два дедлайна: **мягкий** и **жёсткий**. **Мягкий** наступает через 4 дня после выдачи задачи (если не оговорено иное) и нужен для того, чтобы гарантировать своевременную проверку задачи. Задача, ревью на которую запрошено до мягкого дедлайна гарантированно будет проверена с возможностью исправить замечания до наступления жёсткого дедлайна. Если задача сдана после мягкого дедлайна, то своевременность проверки уже не гарантируется. То есть возможны следующие ситуации/

- Задача сразу решена правильно и тогда не зависимо от времени проверки за неё даётся полный балл
- Задача решена не правильно но проверена и исправлена до жёсткого дедлайна. Тогда получается полный балл. Но, как было сказано выше, нет гарантии, что хватит времени на исправление.
- Задача решена не правильно и это выяснилось после жёсткого дедлайна: тогда за неё уже нельзя получить полный балл.

Жёсткий дедлайн не может наступить раньше, чем через 6 дней с момента анонса задачи, как правило равен неделе, но может быть и больше. После жёсткого дедлайна баллы разово снижаются в два раза³. Любая задача может быть либо зачтена полностью, либо не зачтена вообще⁴. Полный балл за все задачи — 100. Каждый раз задаётся (и, соответственно, сдаётся) одна задача⁵.

Язык программирования — Python. Набор инструментов и структура репозитория для домашних задач фиксированы и представлены здесь: <https://github.com/JetBrains-Research/formal-lang-course>. Набор библиотек следующий.

- `cfrq_data` для работы со входными данными (графами).
- `sciPy` для базовых операций с матрицами.
- `pygraphblas` для оптимизированных операций с матрицами на CPU.
- `pyspbla` (`pusubool`?) для булевых матриц на GPU.
- `pyformlang` для работы с языками (регулярные выражения, конечные автоматы, КС грамматики).

²Дополнительные задачи выдаются только после решения основных.

³То есть задача, сданная в любой момент после жёсткого дедлайна стоит в два раза меньше, чем сданная вовремя. Судя по всему, Эталон Примерович из таблицы 1 сдал задачу 3 после жёсткого дедлайна.

⁴То есть частично решённых задач быть не может. А значит, и неполных баллов.

⁵Таким образом, идеальный расклад — одна задача в неделю. На некоторые задачи, правда, может выделяться больше времени.

- ANTLR для синтаксического анализа.
- pyTest для организации тестов.
- pyDOT для визуализации графов.

Так как часть задач будет состоять в постановке экспериментов, то для них требуется оформление отчёта по поставленным экспериментам. Отчёт оформляется как Python notebook в Google Colab, который содержит как код для постановки экспериментов (чтобы было видно, что и как измеряли), так и весь сопроводительный текст, графики и т.д.

Качество оформления отчёта влияет на то, будет ли зачтена задача. То есть, даже если всё замерено грамотно, но отчёт оформлен небрежно, то задача может быть не зачтена.

Задачи сдаются через GitHub. Процедура выглядит следующим образом.

1. Студент создаёт репозиторий, в который приглашает ассистентов, помогающих с проверкой домашних заданий (анонсируются на первой паре). Запись об этом репозитории добавляется в таблицу с результатами.
2. Студенты распределяются преподавателем среди ассистентов.
3. Репозиторий снабжается readme, системой автоматической сборки, системой автоматического тестирования, проверкой качества кода. Для упрощения этого шага необходимо пользоваться предоставленным шаблоном: достаточно сделать его fork.
4. Новая задача решается в отдельной ветке. Если это задача на постановку экспериментов, то ссылка на notebook добавляется в readme репозитория. При решении задачи необходимо следовать структуре, заданной в репозитории-шаблоне. Задача снабжается тестами. Качество тестов — обязанность студента. Задача может быть не принята из-за плохого тестового покрытия. После того, как все автоматические проверки пройдены, открывается реквест в основную ветку. Если и реквест прошёл все автоматические проверки, то запрашивается ревью у соответствующего ассистента⁶.
5. Ассистент выполняет проверку, оставляет комментарии, выносит вердикт. Если задача зачтена, то реквест мёржится и закрывается студентом, а балл за задачу заносится в таблицу ассистентом. Иначе вносятся исправления, добавляются к этому же реквесту⁷, заново запрашивается ревью⁸. и так до тех пор, пока задача не будет зачтена⁹.

Необходимо иметь ввиду, что многие задачи связаны между собой (следующая использует результаты предыдущих), поэтому не всегда возможно безболезненно не делать какую-то одну задачу из середины списка.

1.3 Летучки

Летучка — небольшая, на 5–10 минут, проверочная работа, проводимая в начале пары. Летучка проверяет базовые знания: определения, теоремы, алгоритмы, свойства алгоритмов.

⁶Запрос ревью — есть попытка сдачи, которая регламентируется дедлайнами. Запрашивать ревью можно только на полностью решённую задачу. Нельзя запрашивать ревью на частичные решения.

⁷Реквест не закрывать. Это позволяет отслеживать историю замечаний.

⁸Запрашивать ревью обязательно, так как каждый коммит проверять никто не будет.

⁹Ну, или до тех пор, пока не отпадёт необходимость её решать.

Она содержит одно задание, которое может быть как теоретическим (написать определение чего либо), так и практическим (показать шаги какого-то алгоритма для заданного входа).

Летучка оценивается от 0 до 1 балла с шагом в 0.25 и используется как вес для домашних работ. Оценки летучек не обсуждаются и не корректируются. Летучки не переписываются.

Точная дата летучки заранее не анонсируется, однако их примерное местоположение может быть известно заранее, так как обычно они привязываются к тем или иным блокам материала.

Точное количество летучек также заранее не известно, но обычно это 4–5. Так как могут быть уважительные причины пропустить летучку, то в конце семестра за две случайно выбранные летучки с нулём баллов выставляется 0.75 баллов¹⁰.

При занятиях в аудитории летучка пишется на листке бумаги, на котором, кроме решения задач, указывается ФИО студента и вариант (который назначается преподавателем). Листочки сдаются строго по истечению отведённого времени.

При занятии в удалённом формате всё происходит точно также, только сдаётся фотография (скан) листочка. Способ сдачи указывается преподавателем перед началом летучки. Время необходимое на отправку фото входит во время всей летучки¹¹.

Примеры вопросов на летучку.

1. Какова сложность алгоритма Хеллингса относительно размера графа?
2. В чём отличие НФХ от ОНФХ?
3. Дайте определение рекурсивного конечного автомата.
4. Нарисуйте начальное состояние таблицы СУК для данного входа и грамматики.
5. Принимается ли данная цепочка данным автоматом? Если да, то предъявите соответствующую последовательность переходов.
6. Какова сложность тензорного произведения относительно размера исходных матриц?

2 План занятий

1. Введение. О чём курс: общая структура, что будет и чего не будет. Правила получения оценки за курс. Базовые определения.
2. Иерархия Хомского. Основные классы языков. За пределами иерархии Хомского. Нестроковые языки.
3. Задача поиска пути с ограничениями в терминах формальных языков. Варианты постановки, прикладное значение, теоретические вопросы.
4. Регулярные языки, конечные автоматы (детерминированные, недетерминированные), регулярные выражения. Операции над ними. Операции над автоматами как операции над их матрицами смежности. Поиск путей с регулярными ограничениями.

¹⁰Просто чтобы написать самому было лучше, чем прогулять и понадеяться на случайность.

¹¹Так что будьте осторожны. Если на летучку дали 10 минут, то через 10 минут фото листочка должно быть у преподавателя. А не через 10 минут срочно начнётся поиск внезапно пропавшего телефона и т.д.

5. Контекстно-свободные языки. Нормальная и ослабленная нормальная формы Хомского. Поиск путей с КС ограничениями. CYK и Hellings.
6. Матричный алгоритм КС запросов.
7. Тензорный алгоритм КС запросов.
8. Дерево разбора и поиск путей. SPPF.
9. Синтаксический анализ языков программирования. Лексика и синтаксис. Тонкости, проблемы, инструменты.
10. ANTLR, LL, ещё раз про неоднозначности.
11. Семантика языков программирования. Интерпретаторы. Что делать с деревом разбора.
12. Атрибутные грамматики.
13. Немного о том, что за КС тоже есть жизнь.

Общая цель курса — посмотреть на формальные языки с прикладной точки зрения. При этом предлагается попробовать применить их сразу в двух областях: классический синтаксический анализ языков программирования и анализ графов.

В ходе курса будет предложено разработать небольшой инструментарий для выполнения запросов к графам. Окажется, что алгоритмы для некоторых задач анализа графов непосредственно основаны на алгоритмах из теории формальных языков и синтаксического анализа. Далее, будет предложено разработать язык запросов, позволяющий использовать разработанные алгоритмы. Необходимо будет разработать сам язык, лексический и синтаксический анализаторы для него, интерпретатор. Интерпретатор будет использовать разработанные алгоритмы выполнения запросов к графам.

Таким образом, в результате работы над задачами у каждого должно получиться приложение, предоставляющее REPL для работы с графами, поддерживающее следующие возможности.

1. Проверить, какие графы доступны.
2. Узнать базовую информацию о выбранных графах (количество вершин, рёбер, метки на рёбрах и т.д.).
3. Сгенерировать синтетический граф.
4. Описать запрос.
5. Задать запрос к некоторым графам.

В качестве примера можно ориентироваться на SQL-консоль.

Гипотетический пример работы:

```
GQL> list graphs
geospecies
fof
drivers
GQL> list labels 'fof'
```



```

11
12
GQL> let S = 11 S 12 | eps
S: constraint
GQL> select v1,v2 form fof where v1 -S-> v2
0, 10
2, 9
....

```

Примерные темы задач с баллами.

- 5 Развернуть репозиторий, снабдить его всем необходимым (сборка, тесты). Научиться подгружать графы из набора данных, запрашивать у них вершины и рёбра. На основе этой функциональности реализовать первые тесты. Реализовать консольный интерфейс, позволяющий получить количество вершин и рёбер для указанного графа. Далее этот интерфейс будет расширяться и остальные задачи должны уметь работать с консолью.
- 2 Реализовать преобразование регулярного выражения в ДКА.
- 2 Реализовать преобразование графа в НКА.
- 5 Реализовать алгоритм выполнения регулярных запросов через тензорное произведение.
- 11 Сравнение производительности пересечения автоматов через тензорное произведение и стандартное из Pyformlang.
- 2 Реализовать преобразование контекстно-свободной грамматики в НФХ и ОНФХ.
- 5 Реализовать построение рекурсивного конечного автомата и его минимизацию.
- 5 Реализовать алгоритм синтаксического анализа СΥΚ.
- 5 Реализовать алгоритм Хеллингса для задачи достижимости с контекстно-свободными ограничениями.
- 5 Реализовать матричный алгоритм (для булевых матриц) для задачи достижимости с контекстно-свободными ограничениями.
- 5 Реализовать тензорный алгоритм (для булевых матриц) для задачи достижимости с контекстно-свободными ограничениями.
- 15 Сравнение производительности алгоритмов для задачи достижимости с контекстно-свободными ограничениями: Хеллингс, матричный, тензорный.
- 5 Разработать конкретный синтаксис языка запросов к графам. Описать его в виде документации в репозитории. Снабдить примерами
- 5 Реализовать его парсер языка запросов к графам, согласно разработанному синтаксису.
- 3 Реализовать печать дерева разбора в DOT на основе стандартных возможностей ANTLR.
- 20 Реализовать интерпретатор языка запросов к графам.

3 Лекция 1: Введение

Алфавит, язык. Операции над строками. Операции над языками.

Какие вопросы можно задавать о языках: о пустоте, универсальности, о построении пересечения, о пустоте пересечения, о вложенности, об эквивалентности.

Базовые способы задания: перечисление, генератор, распознаватель.

Взаимосвязь теории формальных языков с другими областями, области её применения.

- Синтаксический анализ языков программирования: в компиляторах, интерпретаторах, средах разработки, других инструментах.
- Анализ естественных языков. Активность в этой области несколько спала, так как на передний план сейчас вышли различные методы машинного обучения. Однако и в этой области ведутся работы. Примеры конференций:
 - International Conference on Parsing Technologies (IWPT-2020)
 - FG: Formal Grammar (FG-2020)
- Статический анализ кода.
 - Различные задачи межпроцедурного анализа. Основной подход — language reachability. Основоположник — Томас Репс. Примеры работ.
 - * Thomas Reps. 1997. Program analysis via graph reachability. In Proceedings of the 1997 international symposium on Logic programming (ILPS '97). MIT Press, Cambridge, MA, USA, 5–19.
 - * Qirun Zhang and Zhendong Su. 2017. Context-sensitive data-dependence analysis via linear conjunctive language reachability. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017). Association for Computing Machinery, New York, NY, USA, 344–358. DOI:<https://doi.org/10.1145/3037697.3037744>
 - * Kai Wang, Aftab Hussain, Zhiqiang Zuo, Guoqing Xu, and Ardalan Amiri Sani. 2017. Graspan: A Single-machine Disk-based Graph System for Interprocedural Static Analyses of Large-scale Systems Code. In Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17). Association for Computing Machinery, New York, NY, USA, 389–404. DOI:<https://doi.org/10.1145/3037697.3037744>
 - * Lu Y., Shang L., Xie X., Xue J. (2013) An Incremental Points-to Analysis with CFL-Reachability. In: Jhala R., De Bosschere K. (eds) Compiler Construction. CC 2013. Lecture Notes in Computer Science, vol 7791. Springer, Berlin, Heidelberg
 - Интерливинг (или шафл) языков для верификации многопоточных программ.
 - * Approximating the Shuffle of Context-free Languages to Find Bugs in Concurrent Recursive Programs
 - * Flick N.E. (2015) Quotients of Unbounded Parallelism. In: Leucker M., Rueda C., Valencia F. (eds) Theoretical Aspects of Computing - ICTAC 2015. ICTAC 2015. Lecture Notes in Computer Science, vol 9399. Springer, Cham
 - Система типов Java: Radu Grigore, Java Generics are Turing Complete.
- Графовые базы данных. Поиск путей с ограничениями.

- Maurizio Nolé and Carlo Sartiani. 2016. Regular Path Queries on Massive Graphs. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management (SSDBM '16). Association for Computing Machinery, New York, NY, USA, Article 13, 1–12. DOI:<https://doi.org/10.1145/2949689.2949711>
- Jochem Kuijpers, George Fletcher, Nikolay Yakovets, and Tobias Lindaaker. 2019. An Experimental Study of Context-Free Path Query Evaluation Methods. In Proceedings of the 31st International Conference on Scientific and Statistical Database Management (SSDBM '19). Association for Computing Machinery, New York, NY, USA, 121–132. DOI:<https://doi.org/10.1145/3335783.3335791>
- Jelle Hellings. Querying for Paths in Graphs using Context-Free Path Queries.
- Биоинформатика. В основном это анализ геномных и белковых последовательностей.
 - Witold Dyrka, Mateusz Pyzik, Francois Coste, and Hugo Talibert. Estimating probabilistic context-free grammars for proteins using contact map constraints.
 - James WJ Anderson, Paula Tataru, Joe Staines, Jotun Hein, and Rune Lyngso. Evolving stochastic context-free grammars for RNA secondary structure prediction.
 - Ryan Zier-Vogel. Predicting RNA secondary structure using a stochastic conjunctive grammar.
- Машинное обучение.
 - Matt J. Kusner, Brooks Paige, José Miguel Hernández-Lobato. Grammar Variational Autoencoder. Опубликовано в 2017 году и уже больше 200 цитирований.
 - TAG Parsing with Neural Networks and Vector Representations of Supertags. К разговору об обработке естественных языков.
 - Jungo Kasai, Robert Frank, Pauli Xu, William Merrill, Owen Rambow. End-to-end Graph-based TAG Parsing with Neural Networks.
- Языки — это не только про строки.
 - Языки деревьев: Tree Automata Techniques and Applications.
 - Языки графов:
 - * Graph Grammars
 - * HYPEREDGE REPLACEMENT GRAPH GRAMMARS
 - * (Re)introducing Regular Graph Languages
 - * Hyperedge Replacement: Grammars and Languages
 - ...
- Теория групп. Как правило, это проблема слов группы или дополнение к ней.
 - Anisimov, A.V. Group languages. Cybern Syst Anal (1971) 7: 594.
 - David E. Muller, Paul E. Schupp, Groups, the Theory of ends, and context-free languages, Journal of Computer and System Sciences, Volume 26, Issue 3, 1983, Pages 295-310, ISSN 0022-0000
 - HOLT, D., REES, S., ROVER, C., & THOMAS, R. (2005). GROUPS WITH CONTEXT-FREE CO-WORD PROBLEM. Journal of the London Mathematical Society, 71(3), 643-657. doi:10.1112/S002461070500654X

- Groups with Context-Free Co-Word Problem and Embeddings into Thompson’s Group V
- Kropholler, R. & Spriano, D. (2019). Closure properties in the class of multiple context-free groups. Groups Complexity Cryptology, 11(1), pp. 1-15. Retrieved 13 Feb. 2020, from doi:10.1515/gcc-2019-2004
- Word problems of groups, formal languages and decidability
- Прочая забавная математика.
 - Немного топологии в теории формальных языков: Salvati S. On is an n-MCFL. – 2018.
 - Salvati S. MIX is a 2-MCFL and the word problem in Z2 is captured by the IO and the OI hierarchies //Journal of Computer and System Sciences. – 2015. – Т. 81. – №. 7. – С. 1252-1277.
 - О том, как задачи из теории графов связаны с теорией формальных языков: Abboud, Amir & Backurs, Arturs & Williams, Virginia. (2015). If the Current Clique Algorithms are Optimal, So is Valiant’s Parser. 98-117. 10.1109/FOCS.2015.16.
 - A context-free grammar for the Ramanujan-Shor polynomials

4 Практика 1

Детали о том, как будет проходить практика.

4.1 Григорьев С.В.

Немного про описания языков. Пописать языковые уравнения, грамматики. Посмотреть на операции над языками.

Постановка задачи на весь семестр.

Запросы к графовым базам данных. Контекст задачи, примеры графовых БД (RedisGraph, Neo4j, ...), задача о путях в принципе.

Ссылка на второй конспект.

Задача: реализовать свою "графовую миниБД".

Реализация: оформление, инструменты, языки.

- Ограничений на язык реализации нет.
- Ограничений на использование библиотек нет. Главное — не нарушать лицензии и чтобы можно было вносить изменения в библиотеку (при необходимости).
- Каждый создаёт под решение репозиторий на GitHub и снабжает его всем необходимым: readme, лицензия, CI-сборка с тестированием, инструкции по локальному развёртыванию.
- Разработка ведётся в отдельной ветке и когда очередная часть задачи готова к сдаче — делаем pull request в master и добавляем меня (gsvglit) в ревьюеры.

Задачи на дом.

1. Выбрать язык программирования, на котором будет вестись разработка.
2. Создать репозиторий на GitHub.
3. Настроить CI-сборку и тестирование.
4. Реализовать подгрузку графов из RDF используя готовые библиотеки.

5 Лекция 2: Регулярные языки

Иерархия Хомского. Проблемы с ней. Классы языков.

Грамматики. Системы переписывания.

Регулярные множества. Регулярные языки. Регулярные выражения.

$$V^* = \bigcup_{i=0}^{\infty} V^i$$

Конечные автоматы. Система переходов.

Язык, задаваемый автоматом.

Понятие выводимости (\vdash^*).

Конфигурация: $\langle \text{Состояние}, \text{Остаток} \rangle$.

Полный автомат и вершина-сток.

Детерминизация, алгоритм Томпсона.

НКА: $\langle \Sigma, Q, s \in Q, T \in Q, \delta : Q \times \Sigma \rightarrow 2^Q \rangle$

ДКА: $\langle \Sigma, Q_d, s_d \in Q_d, T_d \in Q_d, \delta_d : Q_d \times \Sigma \rightarrow Q_d \rangle$, где:

- $Q_d = \{q_d \mid q_d \in 2^Q\}$,
- $s_d = \{s\}$,
- $T_d = \{q \in Q_d \mid \exists p \in T : p \in q\}$,
- $\delta_d(q, c) = \{\delta(a, c) \mid a \in q\}$.

ε -замыкание.

1. Транзитивное замыкание отношения ε -перехода.
2. Обработка финальных состояний
3. Добавление переходов: если $\delta(v_0, \varepsilon) = v_1, \delta(v_1, c) = v_2$, то добавим $\delta(v_0, c) = v_2$.
4. Удалим ε -переходы.

Эквивалентность автоматов. Эквивалентность состояний: состояния эквивалентны если нет различающей строки.

Минимизация.

Теорема Клини об эквивалентности автоматов и регулярных языков.

Построение автомата по регулярному выражению.

Построение регулярного выражения по автомату: устранение вершин.

6 Практика 2

6.1 Григорьев С.В.

Построение минимального ДКА по регулярному выражению.

Домашнее задание.

1. Реализовать функцию (можно с применением библиотек), которая принимает на вход регулярное выражение в виде строки и строит по нему минимальный ДКА.
2. Реализовать необходимые тесты на построение ДКА по регулярному выражению.
3. Реализовать (можно с применением библиотек) пересечение минимального ДКА и НКА без ϵ -переходов.
4. Реализовать необходимые тесты на пересечение ДКА и НКА.

7 Лекция3. Контекстно-свободные грамматики

Левосторонние и правосторонние грамматики и регулярные языки. Неразрешимость задачи проверки того, что грамматика задаёт регулярный язык. Статья на эту тему: Self-embedded context-free grammars with regular counterparts. Грамматика \rightarrow регулярка и регулярка \rightarrow грамматика.

Вывод цепочки в грамматике, левосторонний, правосторонний вывод, неоднозначные и однозначные грамматики. Примеры. Существенно неоднозначные языки.

Дерево вывода. Соотношение между деревьями и выводами. Примеры.

Расширенные контекстно-свободные грамматики.

8 Практика 3

8.1 Григорьев С.В.

Пересечение автоматов — это тензорное произведение матриц смежности. Пример.

Про коммутативность пересечения и некоммутативность тензорного произведения.

Домашнее задание.

1. Реализовать консольный клиент, позволяющий
 - (a) загрузить RDF-файл
 - (b) вывести список меток рёбер
 - (c) задать к загруженному графу регулярный запрос с возможностью указать представление результата: пустота ответа, автомат сдампит в файл в формате DOT (<https://www.graphviz.org/doc/info/lang.html>), пара (кол-во рёбер, кол-во вершин) в результирующем автомате
 - (d) выйти из клиента.
2. Подгрузку RDF и выполнение запросов реализовать на основе уже существующей функциональности.

3. Провести замеры производительности на графах из репозитория https://github.com/JetBrains-Research/CFPQ_Data. Графы брать из подпапки `data/graphs/RDF`. Так как в графах присутствуют одинаковые отношения, то можно один и тот же запрос выполнять на всех графах. Отчёт оформить в виде раздела в README репозитория в виде таблицы.

Эти эксперименты проводятся локально! Не надо таскать репозиторий с графами за собой. Для тестов клиента использовать маленькие синтетические RDF.

4. Реализовать необходимые тесты на работоспособность клиента.

9 Лекция 4

Рекурсивные автоматы. Построение, интерпретация.

10 Практика 4

10.1 Григорьев С.В.

Больше подробностей про рекурсивные автоматы: тотальная минимизация. Как их применять для КС запросов. Тензоры + транзитивное замыкание.

Домашнее задание.

1. Реализовать выполнение регулярных запросов через тензорное произведение. Для тензорного произведения использовать существующие библиотеки линейной алгебры. Обратите внимание на то, что матрицы должны быть разреженными. Скорее всего, удобно будет использовать представление в виде набора булевых матриц.
2. Интегрировать новую реализацию в клиент наравне со старой.
3. Провести замеры производительности на графах из репозитория https://github.com/JetBrains-Research/CFPQ_Data. Графы брать из подпапки `data/graphs/RDF`. Так как в графах присутствуют одинаковые отношения, то можно один и тот же запрос выполнять на всех графах. Отчёт оформить в виде раздела в README репозитория в виде таблицы. Сравнить с результатами предыдущей задачи.

Эти эксперименты проводятся локально! Не надо таскать репозиторий с графами за собой. Для тестов клиента использовать маленькие синтетические графы и запросы.

4. Реализовать необходимые тесты на работоспособность алгоритма через тензорное произведение.

11 Лекция 5

11.1 Нормальная форма Хомского (НФХ)

Определение 11.1. КС грамматика находится в нормальной форме Хомского если любое правило имеет один из трёх видов:

1. $S \rightarrow \varepsilon$
2. $N_i \rightarrow t_j$
3. $N_i \rightarrow N_j N_k, N_j \neq S, N_k \neq S$

Важно: стартовый нетерминал не встречается в правых частях правил, ε -продукция только для стартового нетерминала.

Note. Любую КС грамматику можно преобразовать к нормальной форме Хомского.

Преобразование в НФХ. Шаги.

1. Устранение длинных правил.
2. Устранение ε -правил.
3. Устранение цепных правил.
4. Устранение бесполезных нетерминалов
 - (a) Удаление непорождающих нетерминалов
 - (b) Удаление недостижимых нетерминалов
5. Устранение продукций с правой частью длины 2, содержащей терминалы.

Надо не забыть добавить новый стартовый нетерминал, если нужно: чтобы вывести из него ε и чтобы не встречался в правых частях правил.

Важно. Порядок применения шагов преобразования важен.

1. Второй шаг можно поднять наверх, но это приведёт к более существенному разрастанию результирующей грамматики.
2. Подшаги шага 4 нельзя менять местами. Попробуйте поприменять их к грамматике:

$$\begin{aligned} S &\rightarrow AB \mid a \\ A &\rightarrow b \end{aligned}$$

Материалы по преобразованию в НФХ.

11.2 Лемма о накачке для КС языков

Теорема 11.1. Пусть L — контекстно-свободный язык над алфавитом Σ , тогда существует такое n , что для любого слова $\omega \in L$, $|\omega| \geq n$ найдутся слова $u, v, x, y, z \in \Sigma^*$, для которых верно: $uvxyz = \omega$, $vy \neq \varepsilon$, $|vxy| \leq n$ и для любого $k \geq 0$ $uv^kxy^kz \in L$.

Идея доказательства леммы о накачке.

1. Для любого КС языка можно найти грамматику в нормальной форме Хомского.

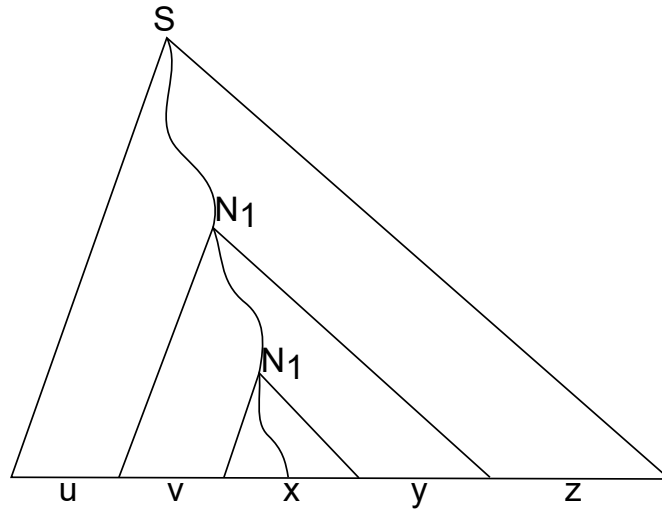


Рис. 1: Разбиение цепочки для леммы о накачке

2. Очевидно, что если брать достаточно длинные цепочки, то в дереве вывода этих цепочек, на пути от корня к какому-то листу обязательно будет нетерминал, встречающийся минимум два раза. Если m — количество нетерминалов в НФХ, то длины 2^{m+1} должно хватить. Это и будет n из леммы.
3. Возьмём путь, на котором есть хотя бы дважды повторяется некоторый нетерминал. Скажем, это нетерминал N_1 . Пойдём от листа по этому пути. Найдём первое появление N_1 . Цепочка, задаваемая поддеревом для этого узла — это x из леммы.
4. Пойдём дальше и найдём второе появление N_1 . Цепочка, задаваемая поддеревом для этого узла — это uxy из леммы.
5. Теперь мы можем копировать кусок дерева между этими повторениями N_1 и таким образом накачивать исходную цепочку.

Надо только проверить выполнение ограничений на длины.

Материалы по лемме о накачке для КС языков.

Проверить неконтекстно-свободность языка $L = \{a^n b^n c^n \mid n > 0\}$.

12 Практика 5

12.1 Григорьев С.В.

Преобразование в нормальную форму Хомского.

Формат входа:

1. Одна продукция на строку.
2. Продукция — это список терминалов и нетерминалов через пробел, начинающийся с нетерминала (левая часть продукции).
3. Нетерминалы — заглавные буквы с опциональным числовым суффиксом.
4. Терминалы — строчные буквы с опциональным числовым суффиксом.

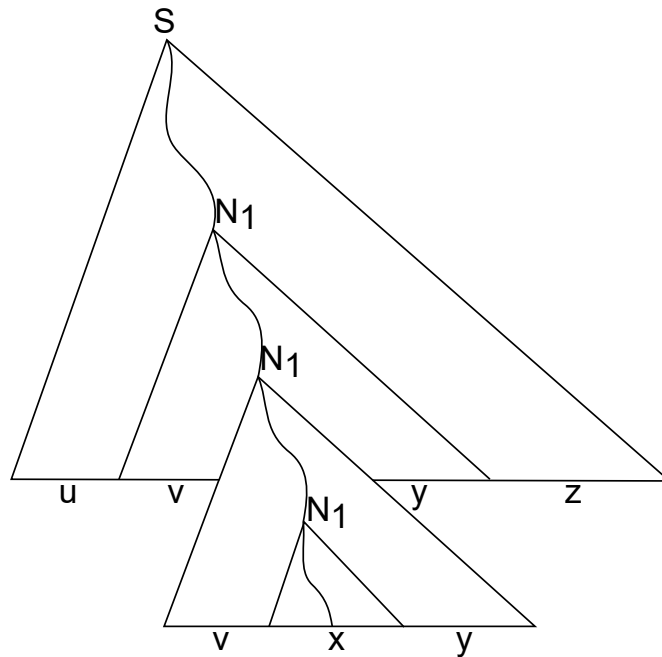


Рис. 2: Пример накачки цепочки с рисунка 1

5. Специальный символ `eps` для обозначения ε .

Пример входа, описывающего грамматику $S \rightarrow aSbS \mid \varepsilon$:

```
S a S b S
S eps
```

Домашнее задание.

1. Реализовать преобразование в нормальную форму Хомского. На входе файл с грамматикой, на выходе — файл с грамматикой в НФХ в том же формате, что и вход.

13 Лекция 6

13.1 Свойства замкнутости КС языков

Глава 2.6 конспекта.

13.2 Алгоритм СҮК

Глава 4.1 “Алгоритм СҮК” конспекта.

14 Практика 6

14.1 Григорьев С.В.

Алгоритм СҮК и алгоритм Хеллингса.

Формат входа для СҮК:

1. Грамматика: смотри предыдущее ДЗ.
2. Входная строка: терминалы разделены пробелами.

Пример входа, описывающего грамматику $S \rightarrow aSbS \mid \varepsilon$:

```
S a S b S
S eps
```

Пример входной строки:

```
a a b a a b b b
```

Формат входа алгоритма Хеллингса:

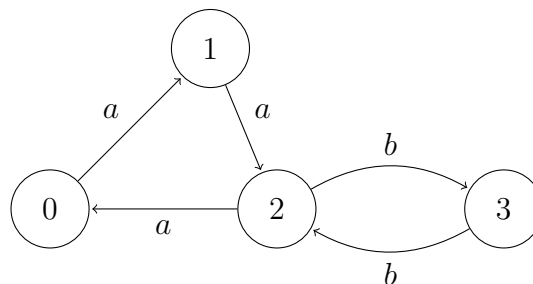
1. Грамматика: тот же формат, что и для СУК. НО! ИСпользуем преобразование а ослабленную НФХ.
2. Входной граф: файл в котором на каждой строке записано ребро в виде тройки

$\langle \text{вершина} \text{ метка_ребра } \text{вершина} \rangle$.

Элементы тройки разделены пробелами.

3. Можно считать, что все вершины графа — числа от нуля, идущие подряд.

Пример входного графа:



Пример описания входного графа:

```
0 a 1
1 a 2
2 a 0
2 b 3
3 b 2
```

Домашнее задание.

1. Реализовать алгоритм СУК для линейного входа. На вход принимаются два файла: с грамматикой и входной строкой. Результат (выводится ли входная цепочка в грамматике) печатается в консоль.
2. Реализовать алгоритм Хеллингса. На вход принимается файл с грамматикой и файл с графом. В результирующий файл печатается грамматика в ослабленной НФХ (с которой непосредственно работал алгоритм) и множество пар достижимых вершин для стартового нетерминала (одна пара на строку, две вершины через пробел)

15 Лекция 7

Алгоритмы решения задачи контекстно-свободной достижимости, основанные на операциях линейной алгебры.

15.1 Алгоритм на основе матричного произведения

Глава 5.1 конспекта.

15.2 Алгоритм на основе тензорного произведения

Глава 6 конспекта.

16 Практика 7

16.1 Григорьев С.В.

Алгоритмы на основе линейной алгебры.

Для реализации предлагается использовать следующие библиотеки. Так как с булевыми не везде хорошо, то будем использовать те типы, которые поддерживаются: `Int`, `float` и т.д.

- Для языка Python — разреженные матрицы в `scipy` и соответствующие операции работы с ними: `scipy.sparse.kron` и обычное матричное произведение. Предпочтительный формат разреженных матриц — CSR.
- Для языка Kotlin — `la4j`. Операции: кронекер и обычное умножение.

Поэлементное сложение есть и там и там.

Для алгоритма на матричном умножении всё точно так же, как и в предыдущей ДЗ для Хеллингса.

Для тензорного произведения расширим формат представления входной грамматики. Одна строка на нетерминал. Терминалы, нетерминалы, ε обозначаются как и раньше. Как и раньше, левая часть от правой отделена пробелом. В правой части можно использовать конструкции регулярных выражений: альтернатива, звезда клин, групперирующие скобки. Этот набор можно расширять по своему усмотрению.

Пример входа, описывающего грамматику $S \rightarrow (aSb)^* \mid \varepsilon$:

`S (a S b)* | eps`

Домашнее задание. Время на выполнение — две недели. Один из алгоритмов — на первую, оставшийся и эксперименты — на вторую.

1. Реализовать алгоритм, основанный на матричном умножении. На вход принимаются два файла: с грамматикой и входным графом. В результирующий файл печатается грамматика в ослабленной НФХ (с которой непосредственно работал алгоритм) и множество пар достижимых вершин для стартового нетерминала (одна пара на строку, две вершины через пробел).

2. Реализовать алгоритм, основанный на тензорном произведении. На вход принимается файл с грамматикой и файл с графом. В результирующий файл печатается матрица смежности рекурсивного автомата (с которым непосредственно работал алгоритм, построчно, элементы разделены пробелом, пустая ячейка обозначается символом '.') и множество пар достижимых вершин для стартового нетерминала (одна пара на строку, две вершины через пробел).
3. Сравнить производительность трёх реализованных алгоритмов (Хеллингс, матричное произведение, тензорное произведение). Результат — описание эксперимента и таблица сравнения в readme.

17 Лекция 8

Нисходящий синтаксический анализ: рекурсивный спуск и LL(k).

Глава 7, 8.1 и 8.2 конспекта.

18 Практика 8

18.1 Григорьев С.В.

Разработка синтаксических анализаторов: лексический и синтаксический анализы, абстрактный и конкретный синтаксис.

18.1.1 Лексический анализ

$$E \rightarrow n \mid E + E \mid E * E$$

Что такое n (число)? Это абстракция.

$$n = [1 - 9][0 - 9]^*$$

$$n = 0 \mid ((-)?[1 - 9][0 - 9]^*)$$

$$n = 0 \mid ((-)?[1 - 9][0 - 9]^*(.)[0 - 9]^*[1 - 9])$$

То же самое верно и для других терминалов.

$$E \rightarrow n \mid E \text{ op_plus } E \mid E \text{ op_pow } E \mid E \text{ op_mult } E$$

$$\text{op_pow} = \wedge$$

$$\text{op_pow} = **$$

Для введения этой абстракции анализ языков разделяют на две стадии:

1. Лексический анализ: переводим последовательность “символов с клавиатуры” в последовательность токенов/терминальных символов. Работает на основе регулярных выражений (как правило).
2. Синтаксический анализ: переводим последовательность терминальных символов в структурное представление (дерево разбора). Работает на основе КС грамматик.

Да, можно и не разделять: scannerless parsing.

18.1.2 Абстрактный и конкретный синтаксис

Структурное представление текста, удобное для решения тех или иных задач, не содержит многие детали исходного текста.

Абстрактное описание синтаксиса:

```
type stmt =
  ...
  | IfStmt of expr*List<stmt>*List<stmt>
  ...
```

Примеры конкретного синтаксиса:

```
...
if ( a + b > 0 )
then
  ...
else
  ...
...
```

```
...
if ( a + b > 0 )
{
  ...
}
else
{
  ...
}
...
```

```
...
cond_branch ( a + b > 0 )( ... )( ... )
...
```

Ещё пример абстрактного синтаксиса:

```
type arithExpr =
  | Num of int
  | OpPlus of arithExpr * arithExpr
  | OpMult of arithExpr * arithExpr
  | OpPow of arithExpr * arithExpr
```

Важно то, что здесь нет ни слова про скобки, приоритеты, ассоциотивность.

$12 + 3 * 4 \sim 5 + 2$

$(12 + 3) * 4 \sim 5 + 2$

$12 + 3 * 4 \sim (5 + 2)$

$(12 + 3 * 4) \sim (5 + 2)$

Потому что многие свойства естественным образом выражается структурой дерева и, соответственно, детали исходного текста теряются.

$12 + 3 * 4 \sim 5 + 2$

```
OpPlus
  (Num 12)
  (OpPlus
    (OpMult
      (Num 3)
      (OpPow
        (Num 4)
        (Num 5))))
  (Num 2)
```

$(12 + 3) * 4 \sim 5 + 2$

```
OpPlus
  (OpMult
    (OpPlus
      (Num 12)
      (Num 3))
    (OpPow
      (Num 4)
      (Num 5)))
  (Num 2))
```

$12 + 3 * 4 \sim (5 + 2)$

```
OpPlus
  (Num 12)
  (OpMult
    (Num 3)
    (OpPow
      (Num 4)
      (OpPlus
        (Num 5)
        (Num 2)))))
```

$(12 + 3 * 4) \wedge (5 + 2)$

```
OpPow
  (OpPlus
    (Num 12)
    (OpMult
      (Num 3)
      (Num 4))
  )
  (OpPlus
    (Num 5)
    (Num 2))
```

Но важно помнить, что для разных задач нужна разная информация.

Можно начинать знакомство с ANTLR, так как он будет использоваться в следующих домашних работах.

19 Лекция 9

Нисходящий синтаксический анализ: GLL и его применение для поиска путей с КС ограничениями.

Глава 8.3 и 8.4 конспекта.

20 Практика 9

20.1 Григорьев С.В.

Абстрактный и конкретный синтаксис языка запросов к графам, семантика языка запросов к графам.

Абстрактный синтаксис скрывает детали синтаксического анализа и конкретного “текстового” представления синтаксиса, оставляя только важную для дальнейшего анализа информацию о структуре программы.

Первая, минималистичная, версия языка запросов к графам.

```
type script: Seq of List<stmt>
```

```
type stmt:
  | Connect of String
  | Select of obj_expr * graph_expr
```

```
type graph_expr =
  | Intersect of graph_expr * graph_expr
  | Query of pattern
  | GraphName of String
```



```

type obj_expr:
  | Edges
  | Count of obj_expr

type pattern:
  | smb of String
  | star of pattern
  | plus of pattern
  | alt of pattern * pattern
  | seq of pattern * pattern
  | option of pattern

```

Теперь поговорим о семантике данного языка.

Семантика — отображение из программ в семантический домен. Программы обычно представляются как абстрактные синтаксические деревья. Семантический домен — то, что описывает смысл программы в контексте решаемой задачи.

Семантика бывает разной. Классические примеры: динамическая семантика (вычисление программ) и статическая семантика (вывод типов).

Доменом динамической семантики арифметических выражений является множество функций из функции, сопоставляющей переменным целочисленные значения, в целые числа.

Семантика часто обозначается скобками $\llbracket \cdot \rrbracket$ и $\llbracket \cdot \rrbracket$.

Семантика арифметических выражений:

$$\llbracket \cdot \rrbracket : (X \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$$

$$\llbracket n \rrbracket(\sigma) = n, n \in \mathbb{N}$$

$$\llbracket v \rrbracket(\sigma) = \sigma(v)$$

$$\llbracket e_1 \oplus e_2 \rrbracket(\sigma) = \llbracket e_1 \rrbracket(\sigma) \otimes \llbracket e_2 \rrbracket(\sigma)$$

$$\llbracket -e \rrbracket(\sigma) = \llbracket 0 - e \rrbracket(\sigma)$$

Здесь \oplus — конструкторы бинарных операций, \otimes — соответствующие арифметические функции.

В нашем случае, исполнение скрипта на получившемся языке достаточно естественно описывается состоянием из текущей базы данных, текущего графа, и текущего результата вычислений. Значение этой тройки и будут семантическим доменом нашего языка.

При обработке непустой последовательности инструкций сначала вычисляем результат выполнения первой инструкции, затем на нем — всех остальных.

$$\frac{c \xrightarrow{hd} c', \quad c' \xrightarrow{Seq \ tl} c''}{c \xrightarrow{Seq \ (hd::tl)} c''}$$

При обработке команды **Connect** сохраняем её аргумент в конфигурации как путь к текущей базе данных.

$$\frac{}{\langle db, graph, res \rangle \xrightarrow{Connect\ db'} \langle db', graph, res \rangle}$$

При обработке инструкции **Select** сперва вычисляется граф, из которого будут выбираться объекты, затем вычисляются сами объекты.

$$\frac{c \xrightarrow{graph} c', \quad c' \xrightarrow{obj} c''}{c \xrightarrow{Select\ obj\ graph} c''}$$

При обработке инструкции **Graph** загружаем в окружение граф из файла с именем, соответствующим аргументу этой инструкции, лежащем в базе (директории), соответствующей значению `cur_db`.

$$\frac{}{\langle db, graph, res \rangle \xrightarrow{GraphName\ name} \langle db, loadGraph(String.concat(db, name)), res \rangle}$$

При обработке инструкции **Intersect** сперва вычисляем оба подвыражения, затем перескаем получившиеся графы как конечные автоматы. Считаем все вершины стартовыми и финальными одновременно.

$$\frac{\langle db, graph, res \rangle \xrightarrow{graph_1} \langle db, graph', res \rangle, \quad \langle db, graph, res \rangle \xrightarrow{graph_2} \langle db, graph'', res \rangle}{\langle db, graph, res \rangle \xrightarrow{Intersect\ graph_1\ graph_2} \langle db, intersectFA(graph', graph''), res \rangle}$$

При обработке инструкции **Query** необходимо построить минимальный детерминированный автомат по регулярному выражению, задаваемому шаблоном.

$$\frac{}{\langle db, graph, res \rangle \xrightarrow{Query\ pattern} \langle db, buildMinDFA(pattern), res \rangle}$$

При обработке инструкции **Edges** в текущий результат вычислений записывается множество троек, соответствующих рёбрам текущего графа.

$$\frac{}{\langle db, graph, res \rangle \xrightarrow{Edges} \langle db, graph, \{(v_i, e, v_j) \mid (v_i, e, v_j) \in graph.Edges\} \rangle}$$

При обработке инструкции **Count** сперва вычисляется подвыражение, а затем в текущий результат вычислений записывается размер результата вычисления подвыражения, если результат был множеством.

$$\frac{\langle db, graph, res \rangle \xrightarrow{expr} \langle db, graph, res' \rangle}{\langle db, graph, res \rangle \xrightarrow{Count\ expr} \langle db, graph, Set.count(res') \rangle} \quad type_of(res') \equiv Set$$

Расширенная версия.

```

type script = Seq of List<stmt>

type stmt =
  | Connect of String
  | NamedPattern of String * pattern
  | Select of obj_expr * graph_expr

type graph_expr =
  | Intersect of graph_expr * graph_expr
  | Query of pattern
  | GraphName of String
  | SetStartAndFinal of vertices * vertices * graph_expr

type vertices =
  | Set of Set<int>
  | Range of int * int
  | None

type obj_expr =
  | Edges
  | Filter of cond * obj_expr
  | Count of obj_expr

type cond =
  | Cond of String * String * String * bool_expr

type bool_expr =
  | LblIs of String * String
  | IsStart of String
  | IsFinal of String
  | And of bool_expr * bool_expr
  | Or of bool_expr * bool_expr
  | Not of bool_expr

type pattern =
  | Term of String
  | Nonterm of String
  | Star of pattern
  | Plus of pattern
  | Alt of pattern * pattern
  | Seq of List<pattern>
  | Option of pattern

```

Для описания семантики для добавленных конструкций потребуется изменить домен — в него необходимо добавить *grammar* (текущую грамматику).

При обработке инструкции **NamedPattern** добавляем в *grammar* правило, где левая часть — имя шаблона, а правая часть — регулятное выражение построенное по шаблону.

$$\langle db, graph, grammar, res \rangle \xrightarrow{\text{NamedPattern } name \ pattern} \langle db, graph, addRule(grammar, name, pattern), res \rangle$$

Так как теперь запрос может быть не только регулярным, то изменится обработка инструкции **Query**. Теперь вместо детерминированного конечного автомата строится рекурсивный конечный автомат с учётом накопленной грамматики.

$$\langle db, graph, grammar, res \rangle \xrightarrow{\text{Query } pattern} \langle db, buildRSM(pattern, grammar), grammar, res \rangle$$

Как следствие, изменяется обработка инструкции **Intersect**: нам необходимо отслеживать, какого типа графы пересекаются, так как для успешного выполнения операции необходимо, чтобы хотя бы один из них был обыкновенным конечным автоматом.

$$\frac{\langle db, g, grm, res \rangle \xrightarrow{g_1} \langle db, g', grm, res \rangle, \quad \langle db, g, grm, res \rangle \xrightarrow{g_2} \langle db, g'', grm, res \rangle}{\langle db, g, grm, res \rangle \xrightarrow{\text{Intersect } g_1 \ g_2} \langle db, intersect(g', g''), grm, res \rangle} \quad g' \text{ is DFA or } g'' \text{ is DFA}$$

Добавим операцию, позволяющую явно указывать стартовые и финальные состояния **SetStartAndFinal**.

$$\frac{\langle db, g, grm, res \rangle \xrightarrow{g_1} \langle db, g', grm, res \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{start} \langle db, g', grm, res' \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{final} \langle db, g', grm, res'' \rangle}{\langle db, g, grm, res \rangle \xrightarrow{\text{SetStartAndFinal } start \ final \ g_1} \langle db, (g' \text{ with } Start = res', \ Final = res''), grm, res \rangle} \quad res' \neq \emptyset, res'' \neq \emptyset$$

$$\frac{\langle db, g, grm, res \rangle \xrightarrow{g_1} \langle db, g', grm, res \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{start} \langle db, g', grm, res' \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{final} \langle db, g', grm, res'' \rangle}{\langle db, g, grm, res \rangle \xrightarrow{\text{SetStartAndFinal } start \ final \ g_1} \langle db, (g' \text{ with } Final = res''), grm, res \rangle} \quad res' = \emptyset, res'' \neq \emptyset$$

$$\frac{\langle db, g, grm, res \rangle \xrightarrow{g_1} \langle db, g', grm, res \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{start} \langle db, g', grm, res' \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{final} \langle db, g', grm, res'' \rangle}{\langle db, g, grm, res \rangle \xrightarrow{\text{SetStartAndFinal } start \ final \ g_1} \langle db, (g' \text{ with } Start = res), grm, res \rangle} \quad res' \neq \emptyset, res'' = \emptyset$$

$$\frac{\langle db, g, grm, res \rangle \xrightarrow{g_1} \langle db, g', grm, res \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{start} \langle db, g', grm, res' \rangle, \quad \langle db, g', grm, res \rangle \xrightarrow{final} \langle db, g', grm, res'' \rangle}{\langle db, g, grm, res \rangle \xrightarrow{\text{SetStartAndFinal } start \ final \ g_1} \langle db, g', grm, res \rangle} \quad res' = \emptyset, res'' = \emptyset$$

$$\langle db, graph, grammar, res \rangle \xrightarrow{\text{Set } s} \langle db, g, grammar, s \cap graph.Vertices \rangle$$

$$\langle db, graph, grammar, res \rangle \xrightarrow{\text{None}} \langle db, g, grammar, \emptyset \rangle$$

$$\langle db, graph, grammar, res \rangle \xrightarrow{Range\ x\ y} \langle db, g, grammar, graph.Vertices \cap \{\min(x, y) \dots \max(x, y)\} \rangle$$

Расширим возможности обработки результата пересечения. Для этого добавим фильтрацию рёбер. Обработка конструкции **Filter** выглядит следующим образом.

$$\frac{\langle db, graph, grammar, res \rangle \xrightarrow{src} \langle db, g, grammar, res' \rangle}{\langle db, graph, grammar, res \rangle \xrightarrow{Filter\ cond\ src} \langle db, g, grammar, Set.filter\ cond\ res' \rangle} \quad type\ of\ res' \ is\ Set<Edge>$$

21 Лекция 10

Восходящий синтаксический анализ: алгоритмы семейства LR.

Глава 9.1 конспекта.

22 Практика 10

22.1 Григорьев С.В.

Синтаксический анализ с использованием генератора синтаксических анализаторов ANTLR.

Домашняя страница ANTLR.

ANTLR для Python.

ANTLR для Java.

Задача на дом.

1. Реализовать синтаксический анализатор языка запросов к графовым БД (разработанный в рамках предыдущей ДЗ) с использованием ANTLR (или другого инструмента создания синтаксических анализаторов). Ожидаемая функциональность следующая.
 - Чтение скрипта из файла.
 - Чтение скрипта с консоли.
 - Вывод в консоль сообщения о корректности/некорректности скрипта.
 - Вывод в файл дерева разбора скрипта в формате GraphViz/DOT.

23 Лекция 11

Восходящий синтаксический анализ: алгоритмы семейства LR.

Глава 9.2 конспекта.

24 Практика 11

24.1 Григорьев С.В.

Исполнение скриптов запросов.

ANTLR-пакет для Ubuntu.

Пример калькулятора.

Задача на дом.

1. Автоматизированная сборка с генерацией файлов по грамматике. Сгенерированные файлы удалены из репозитория.
2. Поддержка выполнения части команд языка запросов. На данном этапе считаем, что скрипт полностью корректен.
 - `connect` полностью
 - `list` полностью
 - `named_pattern` можно без конструкций регулярных выражений (звезда Клини, альтернатива)
 - `select` можно только `exists` и `v_expr` только имя (`name of String`)
3. Тесты. Очень много тестов. Каждая смысловая конструкция должна быть проверена.

25 Лекция 12

Контрольная работа по КС языкам.

26 Практика 12

26.1 Григорьев С.В.

Исполнение скриптов запросов.

ANTLR-пакет для Ubuntu.

Пример калькулятора.

Задача на дом.

1. Полная поддержка языка запросов.
2. Тесты. Очень много тестов. Каждая смысловая конструкция должна быть проверена.

27 Лекция 13

Сиинтаксически управляемая трансляция, атрибутные грамматики.

Презентация.

28 Практика 11

28.1 Григорьев С.В.

Дополнительные задачи на зачёт.

Для получения зачёта необходимо выполнить следующие условия.

1. Сдать все семестровые задачи.
2. Если какие-то из задач в семестре были сданы с нарушением условий, установленных в начале семестра, то необходимо решить дополнительные задачи. Какие именно, указано в таблице с результатами, в колонке “Доп. задача”.

Дополнительные задачи. В таблице указаны их номера. Красным выделены те, которые нужно решить. Для каждой задачи должно быть представлено:

- Расширение абстрактного синтаксиса (добавлено в документацию в репозитории)
- Расширение конкретного синтаксиса (расширена документация в репозитории)
- Примеры (добавлено в документацию в репозитории)
- Реализация
- Тесты

1. Расширить команду `list` так, чтобы можно было опционально указывать путь к БД, графы в которой хочется вывести. По умолчанию всё так же выводятся графы из текущей БД.
2. Расширить команду `list` так, чтобы с её помощью можно было вывести множество различных меток рёбер в указанном графе.
3. Расширить команду `select` возможностью опционально указывать алгоритм, с помощью которого выполнять текущий запрос. Воспроизвести эксперимент из ДЗ 5. Повлияло ли использование языка запросов на результаты экспериментов?
4. Расширить команду `select` возможностью в качестве графа-источника использовать результат операций объединения, пересечения и дополнения графов из текущей базы данных. То есть во `from` можно писать выражение над графами типа $\text{from } (g_1 \cup g_2) \cap g_3$. Данные операции должны трактоваться как операции над языками, задаваемыми автоматом, где переходы определяются графом, а стартовые и финальные состояния — все вершины графа.

Список литературы