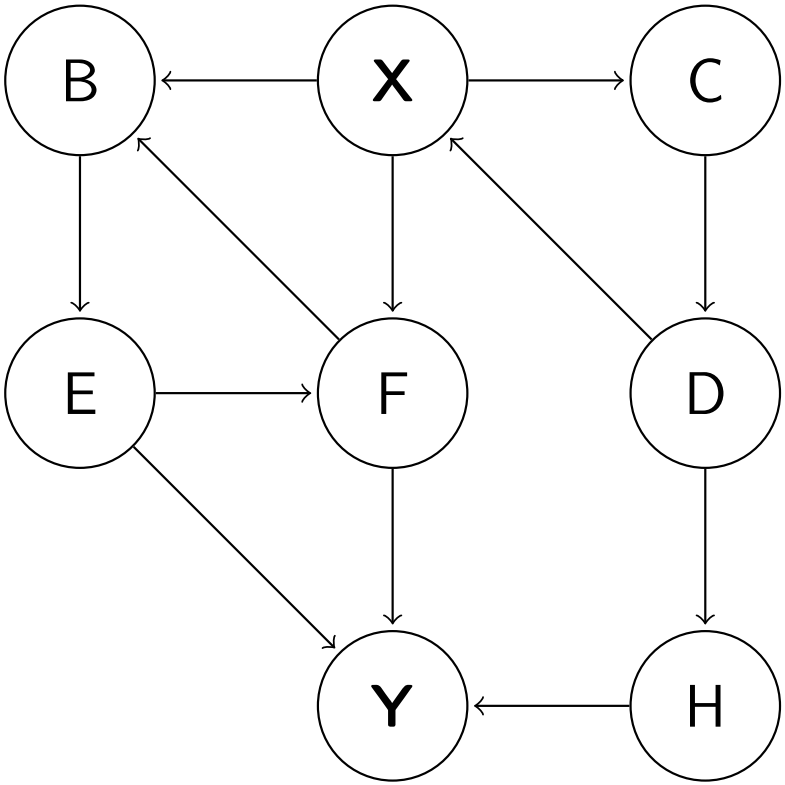
Использование алгоритмов синтаксического анализа для поиска путей с контекстно-свободными ограничениями

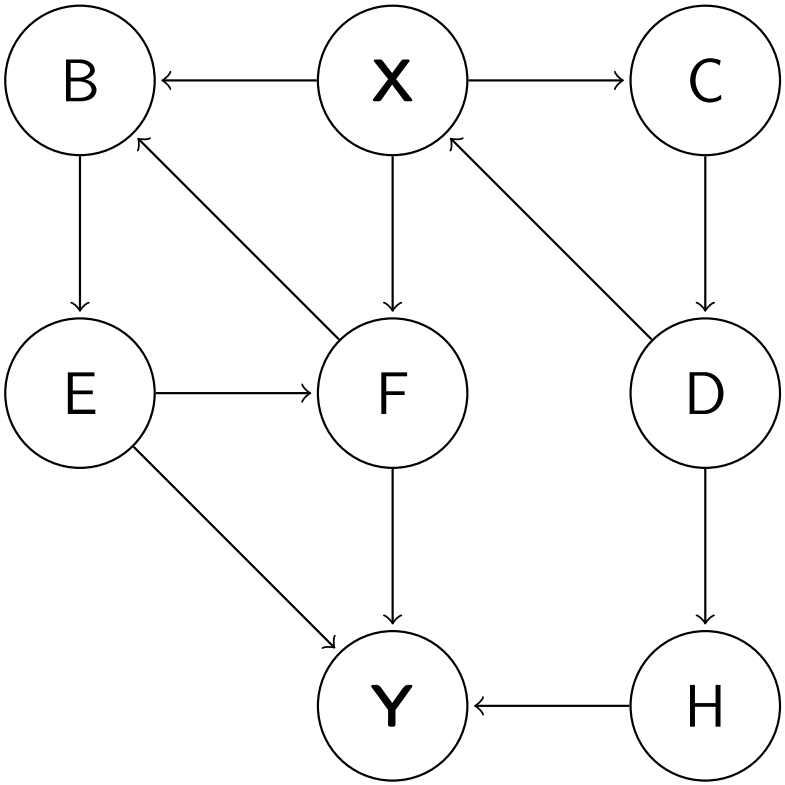
Екатерина Вербицкая
Научный руководитель: д.т.н. Д.В. Кознов
Рецензент: д.т.н. Ф.А. Новиков

JetBrains Research, Лаборатория языковых инструментов
Санкт-Петербургский государственный университет

Поиск путей в графах

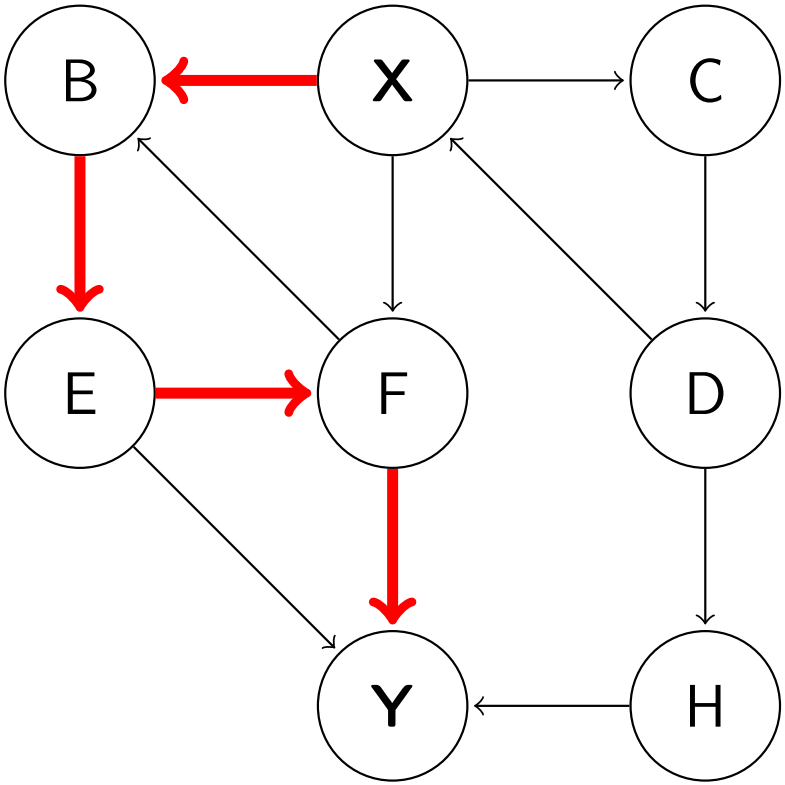


Поиск путей в графах



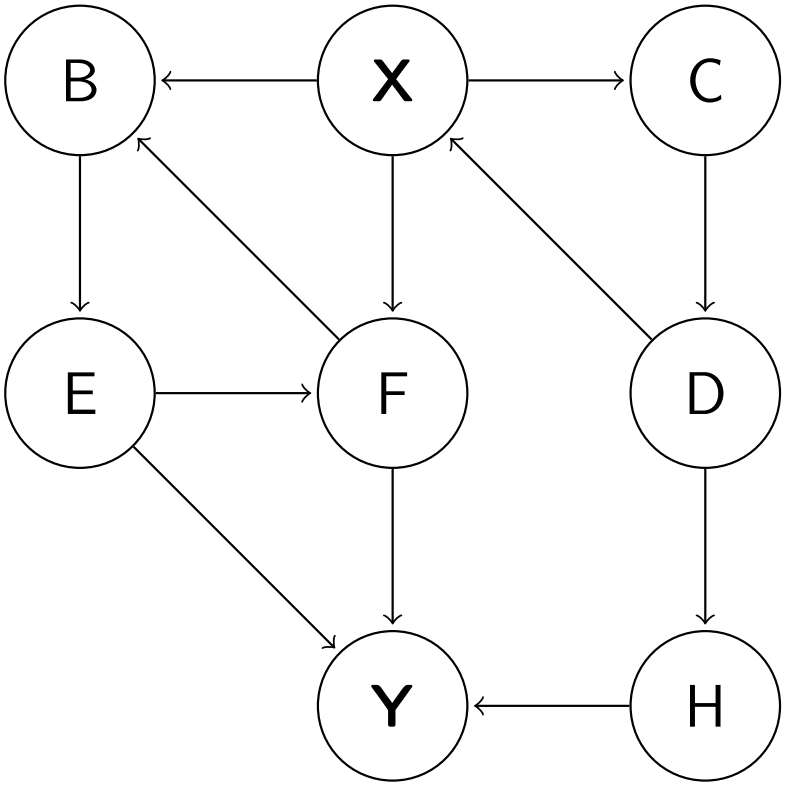
Найти путь из **X** в **Y**

Поиск путей в графах



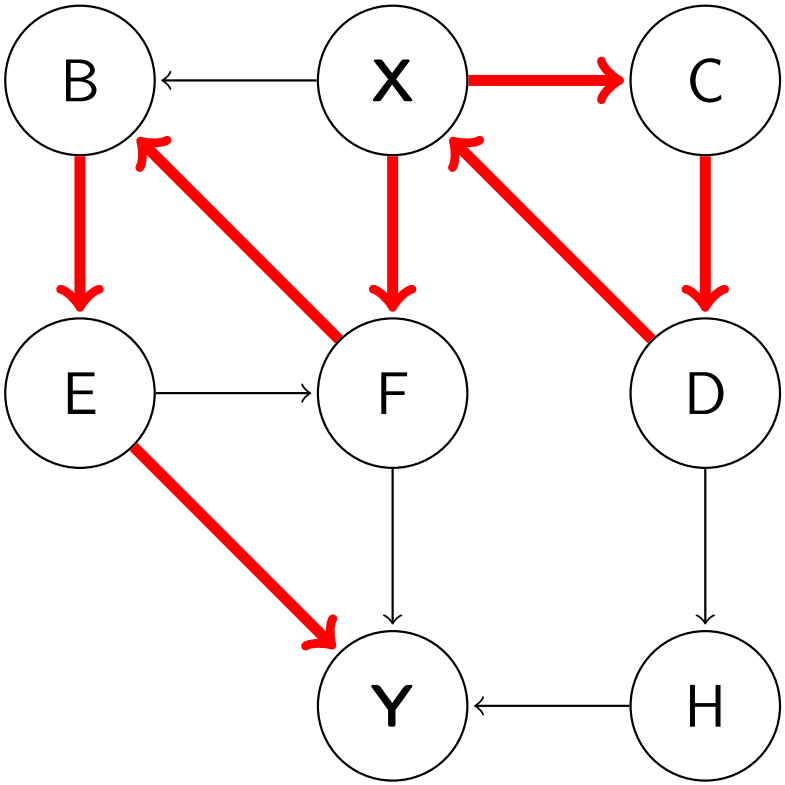
Найти путь из **X** в **Y**

Поиск путей в графах



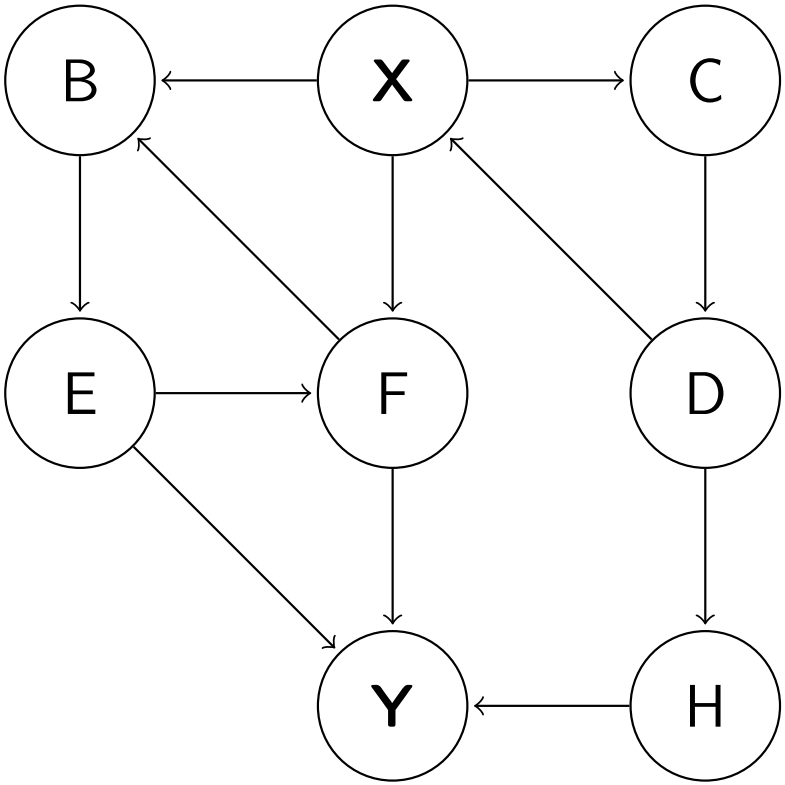
Найти путь длины 13 из **X** в **Y**

Поиск путей в графах



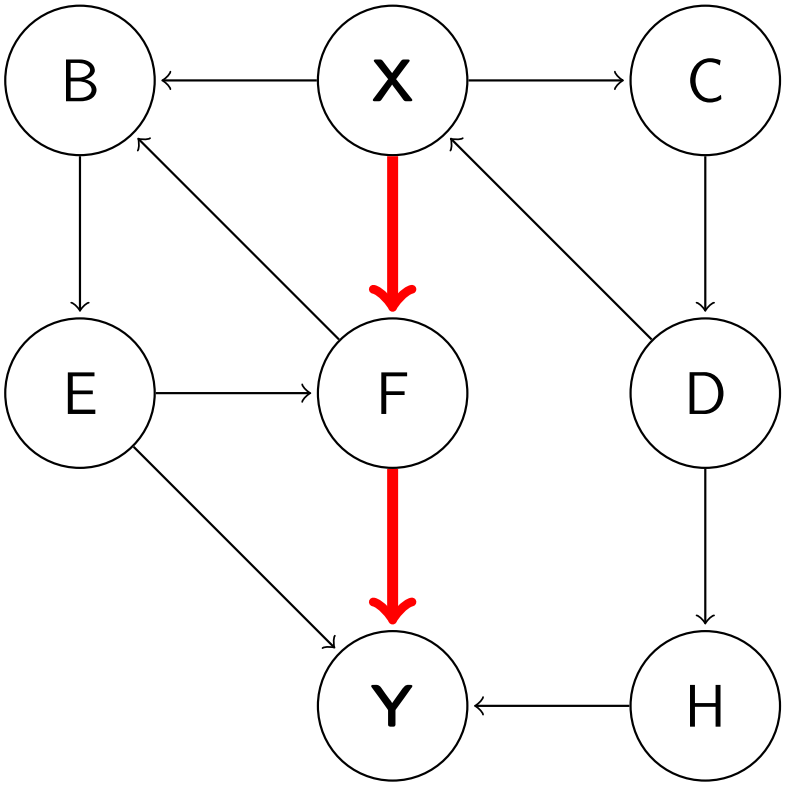
Найти путь длины 13 из **X** в **Y**

Поиск путей в графах



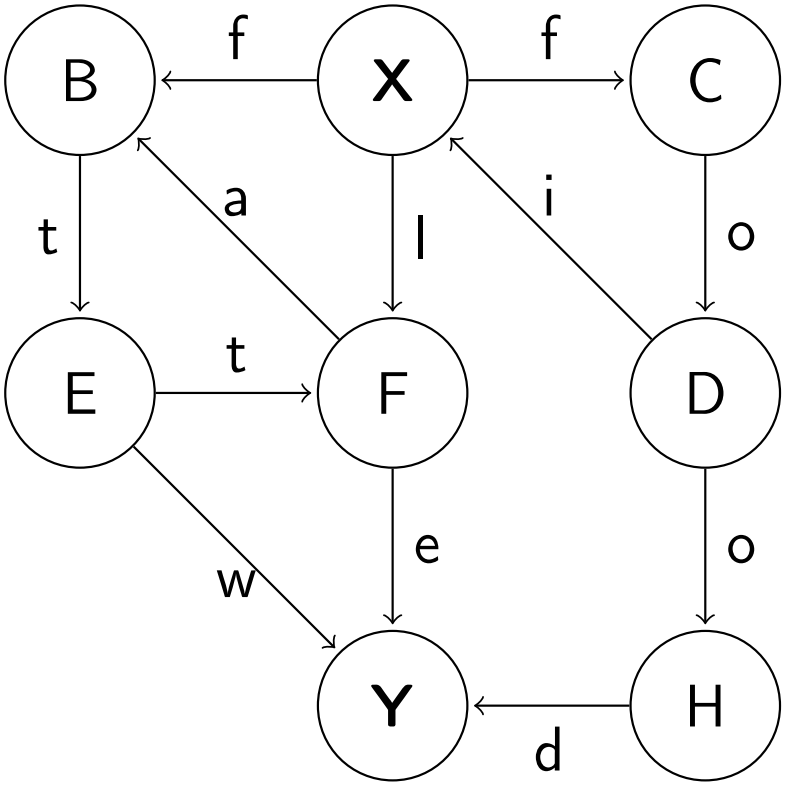
Найти кратчайший путь из **X** в **Y**

Поиск путей в графах

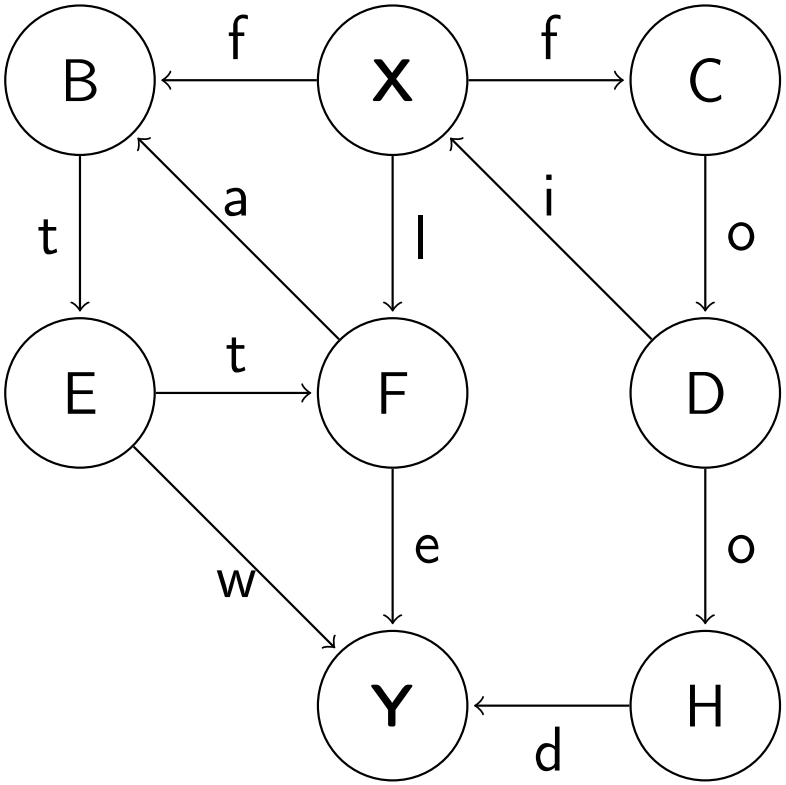


Найти кратчайший путь из **X** в **Y**

Поиск путей в графах: регулярные ограничения

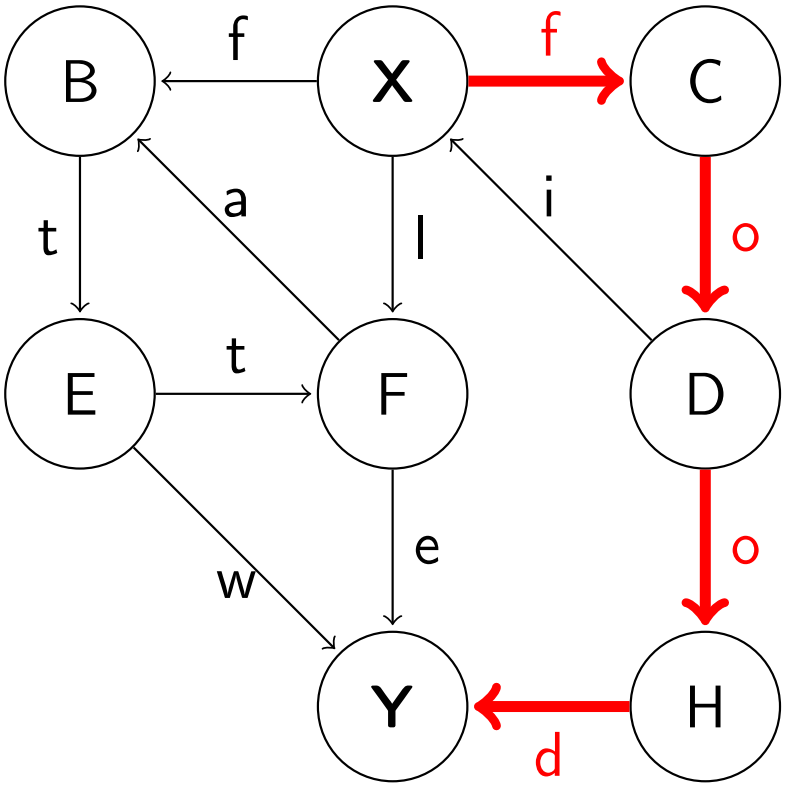


Поиск путей в графах: регулярные ограничения



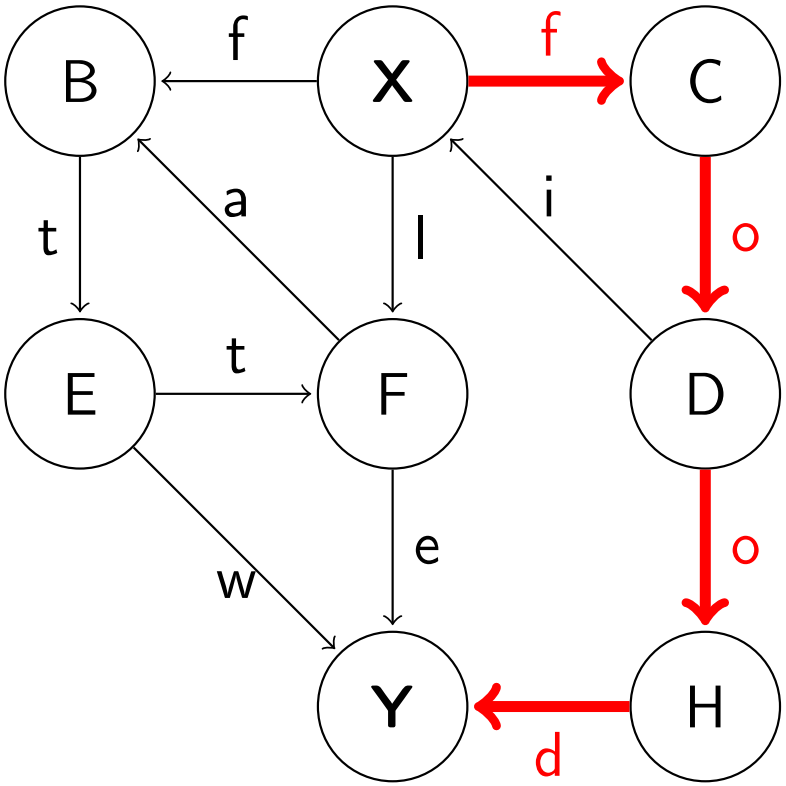
Найти путь из **X** в **Y**, удовлетворяющий регулярному выражению $f\ o^*(d \mid l)$

Поиск путей в графах: регулярные ограничения



Найти путь из **X** в **Y**, удовлетворяющий регулярному выражению $f\ o^*(d \mid l)$

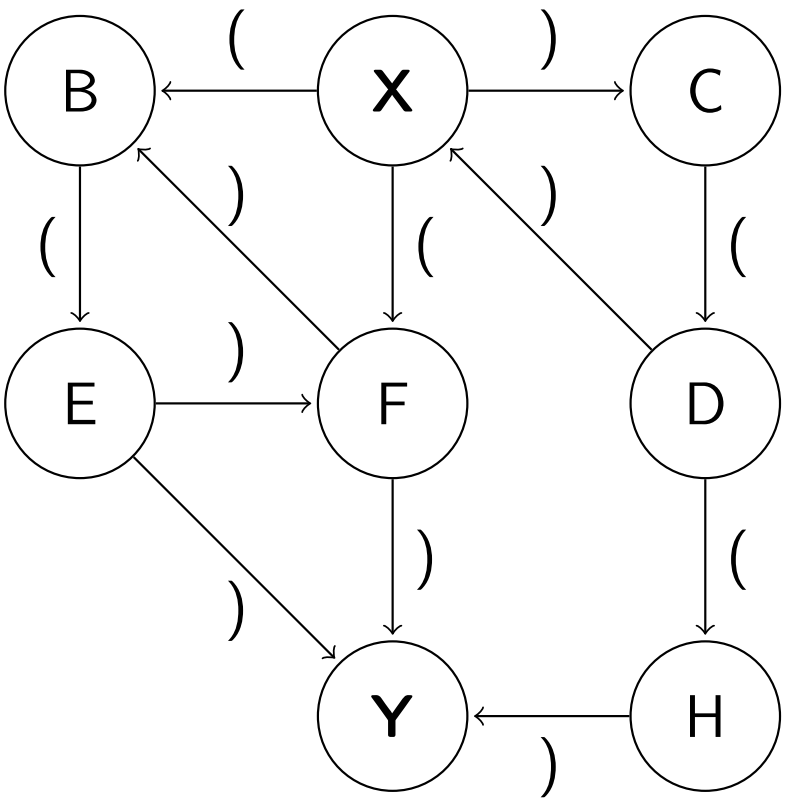
Поиск путей в графах: регулярные ограничения



Найти путь из **X** в **Y**, удовлетворяющий регулярному выражению $f\ o^*(d \mid l)$

Ограничения — регулярный язык

Поиск путей в графах: КС ограничения

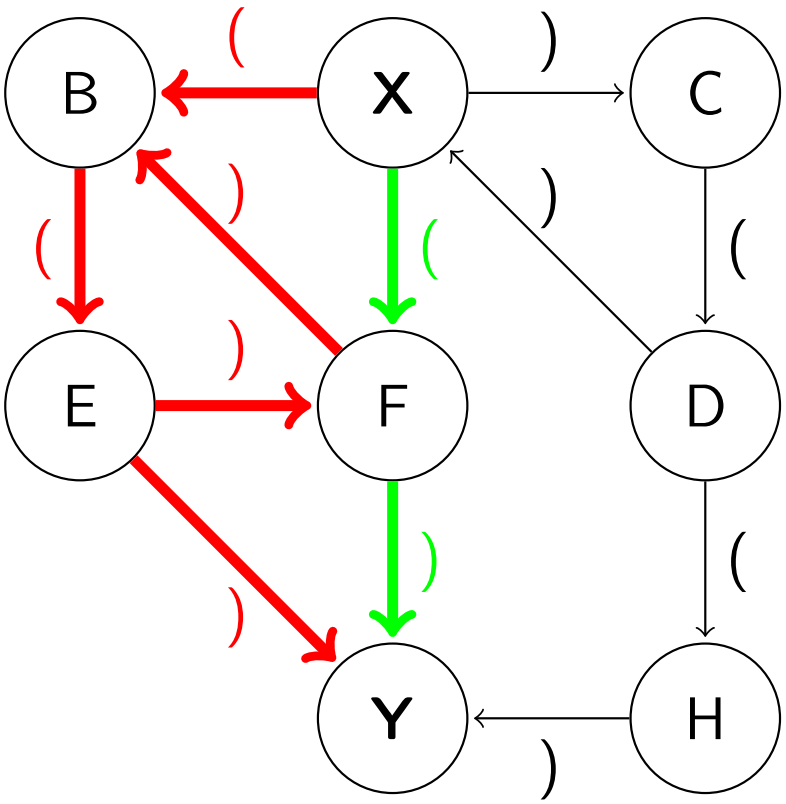


Найти путь из **X** в **Y**, который является сбалансированной скобочной последовательностью

Ограничения — контекстно-свободный язык, заданный грамматикой:

$$S \to \varepsilon \mid (S)S$$

Поиск путей в графах: КС ограничения



Найти путь из **X** в **Y**, который является сбалансированной скобочной последовательностью

Ограничения — контекстно-свободный язык, заданный грамматикой:

$$S \to \varepsilon \mid (S)S$$

(())() () ()()

Поиск путей в графах с КС ограничениями

Входные данные:

- ориентированный граф с метками на ребрах из алфавита Σ
- описание контекстно-свободного языка над алфавитом Σ

Задача: найти все пути в графе, которые являются строками языка

Области применения

- Запросы к графовым базам данных
- Статический анализ кода
- Анализ семантических сетей

Синтаксический анализ для поиска путей

Необходимо производить синтаксический анализ строк на путях

Обобщаем синтаксический анализ для работы с графами

Существующие решения

- Hellings Jelle “Conjunctive context-free path queries” 2014 — теория КС запросов
- Xiaowang Zhang et al “Context-free path queries on RDF graphs” 2016 — основано на CYK
- Kröni Daniel et al “Parsing Graphs: Applying Parser Combinators to Graph Traversals” 2013 — комбинаторы для обходов графов
- Grigorev Semyon et al “Context-free Path Querying with Structural Representation of Result” 2017 — основано на GLL
- Verbitskaia Ekaterina et al “Relaxed parsing of regular approximations of string-embedded languages” 2015 — основано на RNGLR

Не предоставляют средство прозрачной интеграции запросов в основной язык программирования

Постановка задачи

Цель: разработка средства прозрачной интеграции КС запросов в язык программирования общего назначения
Задачи:

- Разработать библиотеку парсер-комбинаторов для КС запросов, которая:
  - поддерживает произвольные КС грамматики
  - поддерживает произвольные входные графы
  - не ограничивает формат входных данных
  - позволяет в явном виде получать пути, удовлетворяющие ограничениям
  - предоставляет механизм выполнения пользовательских семантических функций над путями
- Сравнить производительность реализованной библиотеки с существующими решениями

Парсер-комбинаторы: прозрачная интеграция

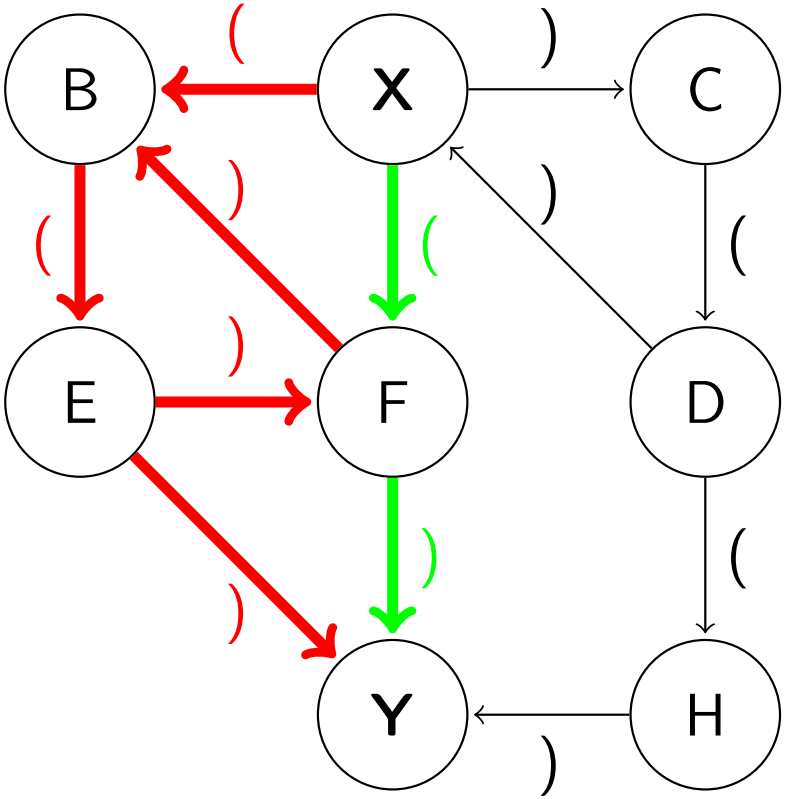
- КС спецификация формулируется на языке программирования общего назначения
- Статические свойства основного языка затрудняют описание бессмысленных КС спецификаций
- Естественное представление результата
- Композициональность

Библиотека Meerkat

- Библиотека обобщенных парсер-комбинаторов на Scala
- Поддерживает произвольные КС спецификации (включая леворекурсивные и неоднозначные)
- Синтаксический анализ в стиле передачи продолжений с мемоизацией
- Вычисляет пользовательскую семантику

```
val E: Nonterminal
= syn ( E ~ "^" ~ E
      | "_" ~ E
      | E ~ "*" ~ E
      | E ~ "/" ~ E
      | E ~ "+" ~ E
      | E ~ "_" ~ E
      | "(" ~ E ~ ")"
      | "[0-9]".r
      )
```

Поиск путей при помощи Meerkat



Найти путь из **X** в **Y**, который является сбалансированной скобочной последовательностью

```
val S: Nonterminal
= syn ( "(" ~ S ~ ")" ~ S
      | ""
      )
```

(())() () ()()

Набор парсер-комбинаторов

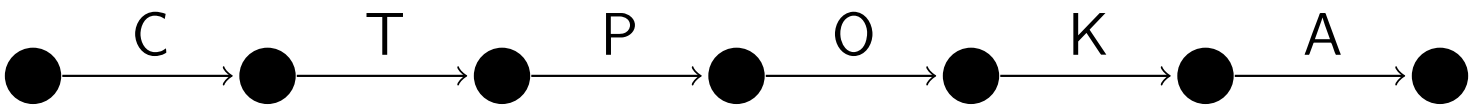
Комбинатор	Описание
a ~ b	последовательное применение: a затем b
a | b	выбор: a или b

Набор парсер-комбинаторов

Комбинатор	Описание
a ~ b	последовательное применение: a затем b
a | b	выбор: a или b
a ?	опциональный разбор: a или ничего
a *	повторение, 0 или больше a
a +	повторение, 1 или больше a
a ^ f	применение функции f к a, если a — терминал
a ^^	вернуть результат применения a, если a — терминал
a & f	применение функции f к a, если a — парсер
a &&	вернуть результат a, если a — парсер

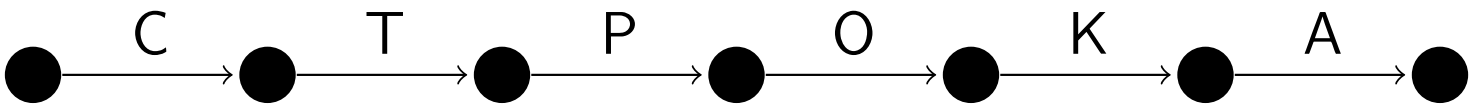
Реализация: входные данные

Строка = линейный граф



Реализация: входные данные

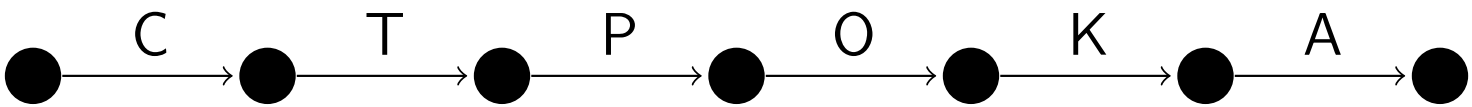
Строка = линейный граф



В чем отличия графа от строки?

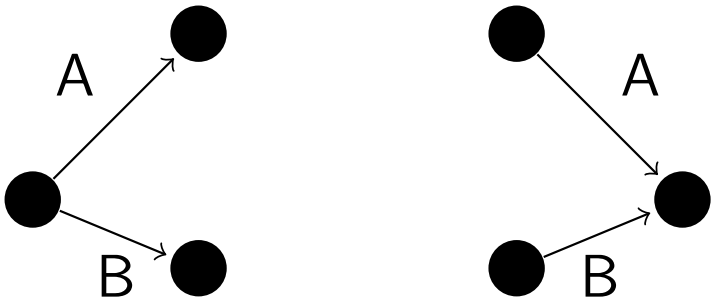
Реализация: входные данные

Строка = линейный граф



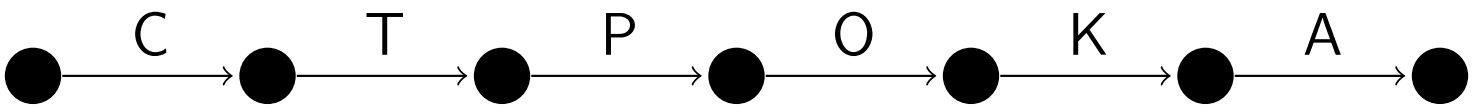
В чем отличия графа от строки?

Ветвления



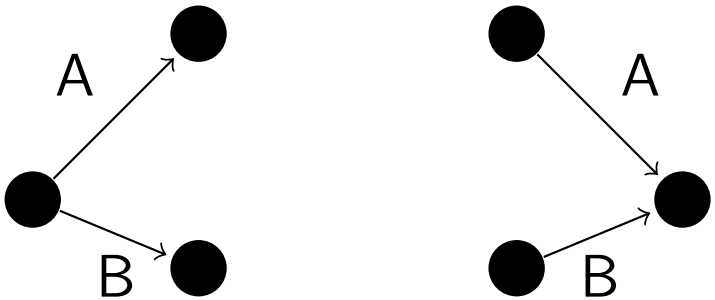
Реализация: входные данные

Строка = линейный граф

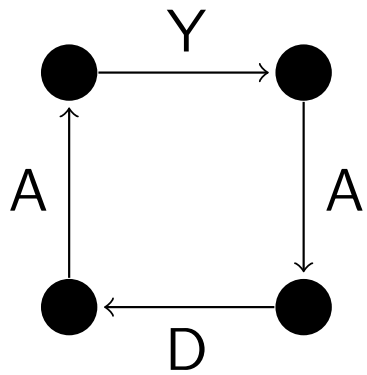


В чем отличия графа от строки?

Ветвления



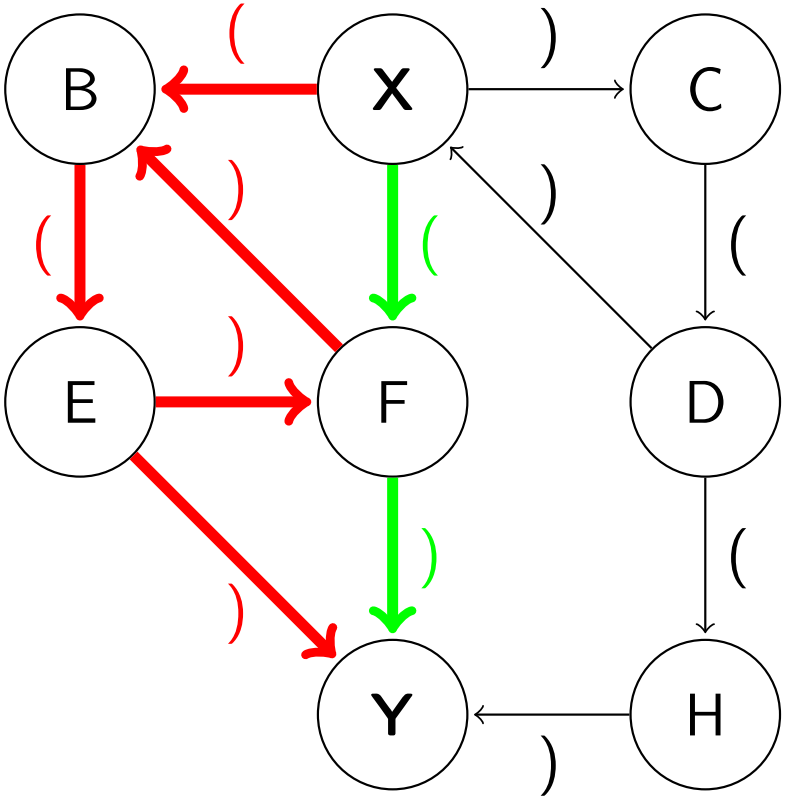
Циклы



Общая идея

- Каждая позиция во входном потоке соответсвует набору промежуточных результатов синтаксического анализа
- В случае ветвления осуществляем синтаксический анализ независимо вдоль исходящих ребер
- Объединяем промежуточные результаты, если несколько путей приводят в одну вершину
- Не анализируем одни и те же позиции дважды
- Мемоизируем промежуточные результаты

Учет информации в вершинах



Найти путь из **X** в **Y**, который является сбалансированной скобочной последовательностью

```
val Query : Nonterminal
= syn (LV("X") ~ S ~
       LV("Y"))

val S: Nonterminal
= syn ( "(" ~ S ~ ")" ~ S
      | ""
      )
```

(())() () ()()

Абстракция входных данных

```
trait Input[+E, +N] {
  def checkNode(nodeId: Int,
      predicate: N => Boolean): Option[N]
  def filterEdges(nodeId: Int,
      predicate: E => Boolean): Seq[(E, Int)] }
```

Мы предоставляем интеграцию с scalax.collection.Graph и Neo4j

Пример

```
val query =
  syn((
    syn(LV("Actor") ^^) ~ outLE("ACTS_IN") ~
    LV("Movie")) & (a => (a.name, a.toInt)))
executeQuery(query, input)
  .groupBy{ case(a, i) => i }
  .toIndexedSeq
  .map{ case(i, ms) => (ms.head._1, ms.length) }
  .sortBy{ case(a, mc) => -mc }}
  .take (10)
```

Эксперименты

- Запросы к онтологиям
  - Примерно в 3 раза быстрее, чем решение, основанное на GLL
- Статический анализ кода: may-alias отношения
  - До 5 раз быстрее, чем Trails
- Запросы к базе данных “Фильмы”
  - До 10 раз медленее, чем Neo4j + Cypher
  - Все запросы регулярны

Ограничения

- Накладные расходы при выполнении регулярных запросов
- Неизвестно, как вычислять произвольную семантику для найденных путей
  - Если извлекать пути лениво, то в каком порядке?
  - Как вычислять семантику, если в графе есть циклы?

Результаты

- На языке программирования Scala реализована библиотека парсер-комбинаторов для запросов к графам
- Произведено сравнение производительности реализованной библиотеки с существующими решениями: она не уступает другим в выполнении контекстно-свободных запросов
- Основные результаты, полученные в данной работе были представлены на конференциях Scala Symposium 2018 (SCOPUS) и “Языки программирования и компиляторы 2017” (РИНЦ)