

Разработка библиотеки комбинаторов запросов к графам

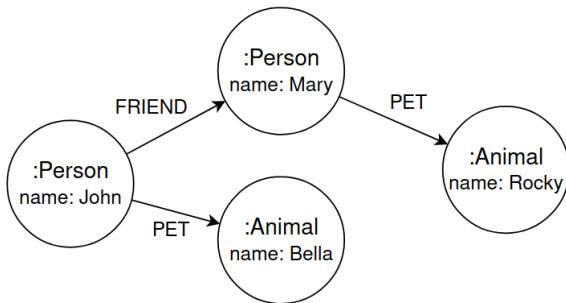
Шушаков Даниил Сергеевич

Научный руководитель: Григорьев Семен Вячеславович

28 мая 2024 г.

Графовое представление данных

- Данные могут представляться в виде графа
- Хранятся в графовых базах данных
- Вершины и ребра могут иметь свойства вида «ключ: значение»



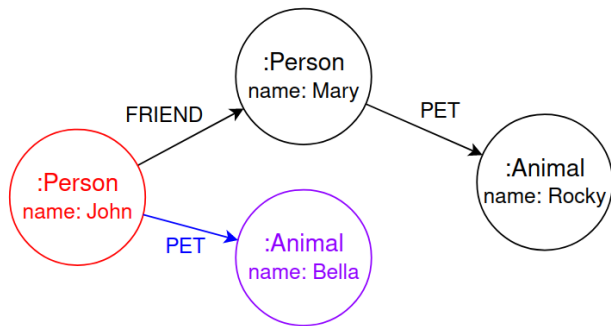
- Для доступа к данным используются языки запросов, например, Cypher для Neo4j
- В пользовательском коде они могут использоваться как строки:

```
tx.execute("MATCH (n {name: 'my node'}) RETURN n, n.name")
```

- Недостатки:
 - Отсутствие проверки корректности запроса на этапе компиляции
 - Сложно составлять и переиспользовать подзапросы
- Удобно иметь бесшовную интеграцию запросов в язык программирования

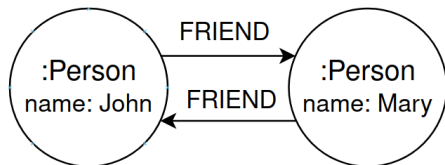
Применение парсер-комбинаторов к графам

- Запрос к графу — нахождение путей, удовлетворяющих ограничениям
- Ограничения на пути можно описать с помощью грамматики
- Грамматику в ЯП общего назначения удобно описать через парсер-комбинаторы



```
val john = v { it.name == "John" }  
val pet = outE { it.label == "PET" }  
val s = john seq pet seq outV()
```

- Описывать КС-ограничения: существуют применения в области биоинформатики¹, статическом анализе² и т.д.
- Разбирать циклы: результатов может быть бесконечно много



¹Sevon et al (2016). Subgraph Queries by Context-free Grammars

²Reps (1997). Program analysis via graph reachability

Свойства	Trails¹	Meerkat
Принцип работы	Рекурсивный спуск	Подходы из статьи ²
Сложность	Экспоненциальная	Полиномиальная
Левая рекурсия	Нет	Да
Циклы в графе	Ограничено	Да
Типизация	Контролирует корректность парсера	Возможны некорректные парсеры

¹Kröni et al (2013). Parsing graphs: applying parser combinators to graph traversals

²Izmaylova et al (2016). Practical, General Parser Combinators

Цель: разработать библиотеку комбинаторов запросов к графам на основе алгоритмов, предложенных в работе ¹, позволяющую описывать корректные парсеры.

Задачи:

- 1 Разработать базовую структуру парсера и комбинаторов с корректной типизацией
- 2 Поддержать левую рекурсию в грамматике
- 3 Поддержать разбор циклов в графе
- 4 Сравнить скорость работы с другими решениями

¹Izmaylova et al (2016). Practical, General Parser Combinators

Реализация: базовая структура парсера

Решение разработано на языке Kotlin.

```
type Parser = (In) -> ParserResult<Out, Res>
```

- Парсер принимает входящее состояние, возвращает множество выходящий состояний и результатов
- Состояние: вершина, ребро, стартовое
- Базовые парсеры: `v`, `edge`, `inE`, `outE`, `inV`, `outV`
- Комбинаторы: `seq`, `or`, `many` и т.д.
- Комбинаторы запрещают объединять парсеры с конфликтующими типами состояний

Левая рекурсия: мемоизация результатов

- Разбор может бесконечно выполняться: левая рекурсия
- Хотим, чтобы вычисление парсера исполнялось один раз
- Парсер в качестве результата теперь возвращает вычисление:

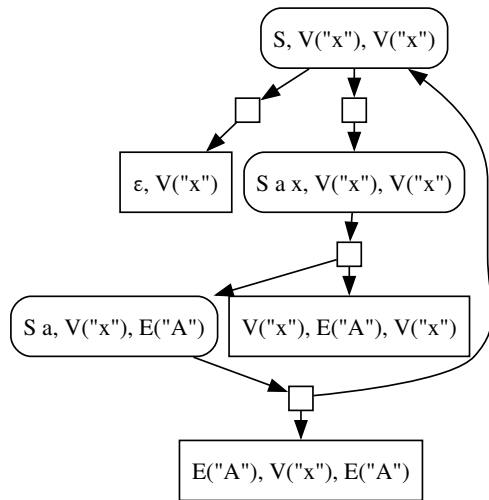
```
type Parser = (In) -> (Continuation1<Out, Res>) -> Unit
```

- Парсер мемоизирует вычисления для каждого входящего состояния
- Вычисление запоминает все результаты и продолжения, при появлении нового результата вызывает эти продолжения
- Всем продолжениям будет переданы все результаты вычисления

¹Можно воспринимать как callback

Циклы в графе: генерация SPPF

- Разбор графа с циклами потенциально может генерировать бесконечно много результатов
- Нужно представить результат такого разбора в виде конечной структуры — SPPF



Набор данных для исследования времени работы

- Для исследования добавлена поддержка базы данных Neo4j
- Взяты три набора данных: онтологии¹, SFPQ_Data², Yago³
- Для первых двух использовался одинаковый КС-запрос
- Для Yago использовался регулярный запрос

КС-запрос

$$Q_1 \rightarrow subclassof^{-1} Q_1? subclassof$$
$$Q_1 \rightarrow type^{-1} Q_1? type$$

Регулярный запрос

```
MATCH (x) - [:P74636308] -> () - [:P59561600] ->
() - [:P13305537*1..] -> () - [:P92580544*1..] ->
(:Entity{id:'40324616'}) RETURN DISTINCT x
```

¹Zhang et al (2016). Context-free path queries on RDF graphs

²https://formallanguageconstrainedpathquerying.github.io/CFPQ_Data/graphs

³<https://gitlab.inria.fr/tyrex-public/rlqdag>

Результаты исследования времени работы

Граф		Кол-во вершин	Кол-во рёбер	Мое реш. (мс)	Meerkat (мс)
ОНТОЛОГИИ	atom-primitive	291	685	19.5	22.0
	b.-m.-p. ¹	341	711	23.5	42.5
	generations	129	351	1.3	1.4
	pizza	671	2,604	61.4	123.9
SFPQ_Data	enzyme	48,815	86,543	121	116
	eclass	239,111	360,248	1220	1095
	go	582,929	1,437,437	6231	5367
	go_hierarchy	45,007	490,109	12015	8337
yago		42,832,856	62,643,951	5558	OOM

¹biomedical-mesure-primitive

- Разработаны базовая структура парсера и комбинаторов, проверяющая корректность во время компиляции
- Поддержаны леворекурсивные грамматики
- Поддержан разбор циклов в графе
- Реализована базовая поддержка Neo4j графов
- Проведено исследование времени работы: время сопоставимо с Meerkat

Meerkat — изначально библиотека с комбинаторами для разбора текста, реализующий алгоритм из статьи ¹, в последствии расширена поддержкой запросов к графам¹.

¹Izmaylova et al (2016). Practical, General Parser Combinators

¹Verbitskaia et al (2018). Parser combinators for context-free path querying

Сигнатуры комбинаторов

```
fun BaseParser<I, O1, R1>.seq(p2: BaseParser<O1, O2, R2>)
    : BaseParser<I, O2, Pair<R1, R2>>

fun BaseParser<I, O, R>.or(p2: BaseParser<I, O, R>)
    : BaseParser<I, O, R>

fun rule(first: BaseParser<I, O, R>, vararg rest: BaseParser<I, O, R>)

fun BaseParser<I, O, A>.using(f: (A) -> B): BaseParser<I, O, B>

fun BaseParser<I, O, R>.that(constraint: BaseParser<O, O2, R2>)
    : BaseParser<I, O, R>

val BaseParser<S, S, R>.many: BaseParser<S, S, List<R>>
val BaseParser<S, S, R>.some: BaseParser<S, S, List<R>>
```

Использование комбинаторов that, using:

```
val mary = outV { it.value == "Mary" }  
val loves = outE { it.label == "loves" }  
val friend = outE { it.label == "friend" }  
val maryLover = v().that(loves seq mary)  
val p = maryLover seqr (friend seq outV()).some  
val s = p using {edges -> Pair(edges.size, edges.last().second)}
```


Запросы для исследования времени (КС)

```
val subclassof1 = throughInE { it.label == "subClassOf" }
val subclassof = throughOutE { it.label == "subClassOf" }
val type1 = throughInE { it.label == "type" }
val type = throughOutE { it.label == "type" }
val Q1 = LazyParser<Neo4jVertexState, Neo4jVertexState, Any>()
Q1.p = (subclassof1 seq (Q1 or eps()) seq subclassof) or
      (type1 seq (Q1 or eps()) seq type)
```

```
val subclassof1: Symbol[L, N, _] = inE("subClassOf")
val subclassof: Symbol[L, N, _] = outE("subClassOf")
val type1: Symbol[L, N, _] = inE("type")
val _type: Symbol[L, N, _] = outE("type")
val grammar: Symbol[L, N, _] =
  syn((subclassof1 ~ syn(grammar | ε) ~ subclassof) |
      (type1 ~ syn(grammar | ε) ~ _type))
```

Запросы для исследования времени (yago)

```
(v { it.properties["id"] == "40324616" } seq  
(inE { it.label == "P92580544" } seq inV()).some seq  
(inE { it.label == "P13305537" } seq inV()).some seq  
inE { it.label == "P59561600" } seq inV() seq  
inE { it.label == "P74636308" } seq inV())
```

```
syn(V((e: Entity) => e.getProperty("id") == "40324616") ~  
inE((e: Entity) => e.label() == "P92580544").+ ~  
inE((e: Entity) => e.label() == "P13305537").+ ~  
inE((e: Entity) => e.label() == "P59561600") ~  
inE((e: Entity) => e.label() == "P74636308"))
```