Язык логического программирования для поиска информации в базе данных

Пичугин Владислав, ИУ9-72Б

Москва, 2021

Цель

Реализовать интерпретатор для языка логического программирования, являющегося языком запросов к простой базе данных

Структура базы данных

Элементы – утверждения (англ. assertions)

```
(position (Pichugin Vladislav) (junior developer))

(salary John (90 EUR))

((birth info) (Pichugin Vladislav) (date (19 April)))
```

Простые запросы

Переменные образца

```
(position $x (junior developer))
```

Списковые переменные

```
(position $x (developer . $type))
```

Возможный результат

```
(position (Pichugin Vladislav) (developer frontend backend))
(position (Ivan Ivanov) (developer android))
(position (Nikita Nikitin) (developer))
```

Составные запросы

Комбинирующие средства

```
(@and <query1> ... <queryN>)

(@or <query1> ... <queryN>)

(@not <query>)

(@apply <predicate> <arg1> ... <argN>)
```

Пример

```
(@and (salary $person $amount) (@apply > $amount 30000))
```

Правила

Общий вид

```
(@rule <conclusion> <body>)
```

Пример

```
(@rule (liveNear $person1 $person2)
    (@and (address $person1 ($town . $rest1))
          (address $person2 ($town . $rest2))
          (@not (same $person1 $person2))))
(@rule (same $x $x))
```

Логический вывод

```
(@rule (append () $y $y))

(@rule (append ($u . $v) $y ($u . $z)) (append $v $y $z))
```

Запрос

```
(append $x $y (a b c d))
```

Результат

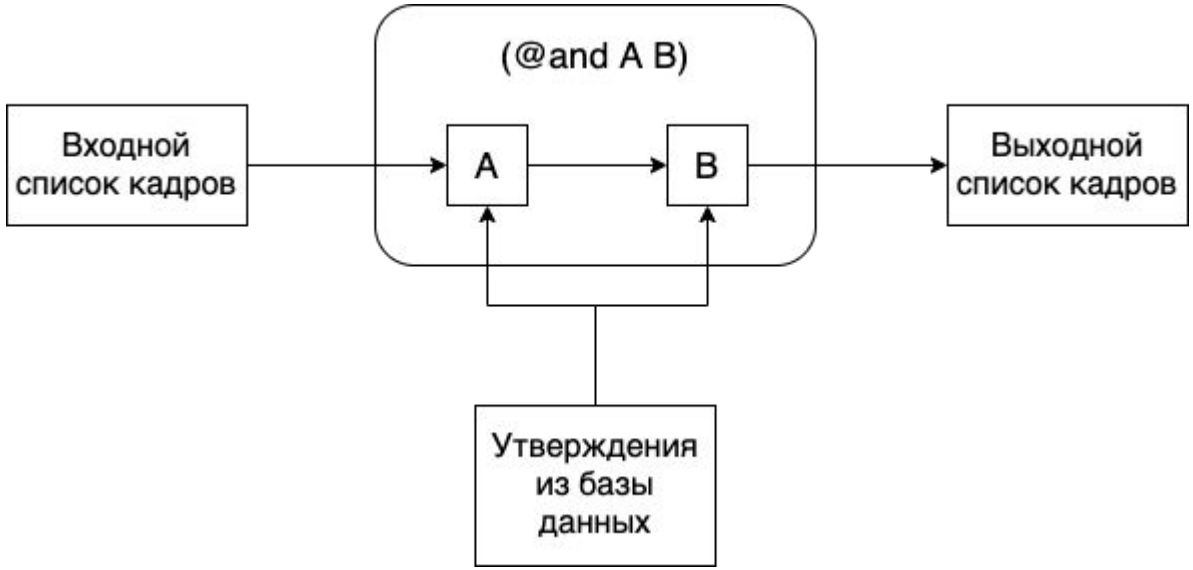
```
(append () (a b c d) (a b c d))

(append (a) (b c d) (a b c d))

(append (a b) (c d) (a b c d))

(append (a b c) (d) (a b c d))

(append (a b c d) () (a b c d))
```

Как это работает?

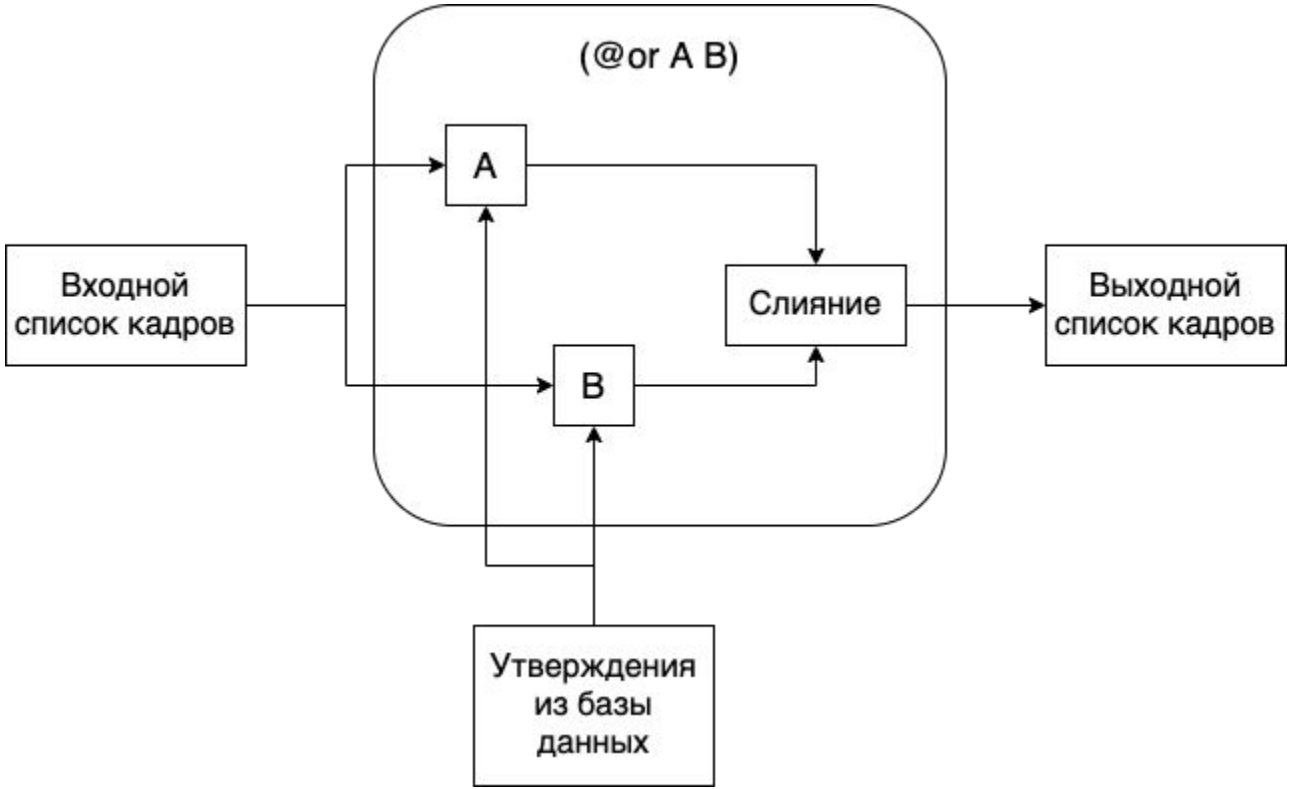
- Сопоставление с образцом
- Списки кадров
- Унификация

Сопоставление с образцом

Интерпретация `@and`



Интерпретация `@or`



Интерпретация `@not` и `@apply`

- Работают как фильтры
- `@not` – уничтожает все кадры, для которых его подзапрос можно удовлетворить
- `@apply` – с помощью кадра конкретизируются переменные образца, затем применяется предикат

Унификация

Унификатор решает систему уравнений, описывающую компоненты образцов

Пример: унификация `($x $x)` и `((a $y c) (a b $z))`

```
$x = (a $y c)

$x = (a b $z)

(a $y c) = (a b $z)

$y = b, $z = c, $x = (a b c)
```

В общем случае успешная унификация может не полностью определить значения переменных

Применение правил

- Образец-запрос унифицируется с заключением правила
- В результате получается кадр, расширенный новыми связываниями
- По отношению к полученному кадру вычисляется запрос в теле правила
- Успешные сопоставления расширяют кадр, который теперь можно использовать при конкретизации исходного образца-запроса

Взаимодействие с пользователем

- Управляющий цикл (чтение команд с терминала)
- Координирующая функция
- Конкретизация исходного запроса (распечатка на экран)
- Особая форма `@new`

  Пример

```
(@new

    (@rule (same $x $x))

    (position Vlad developer)

)
```

Лексическая структура языка

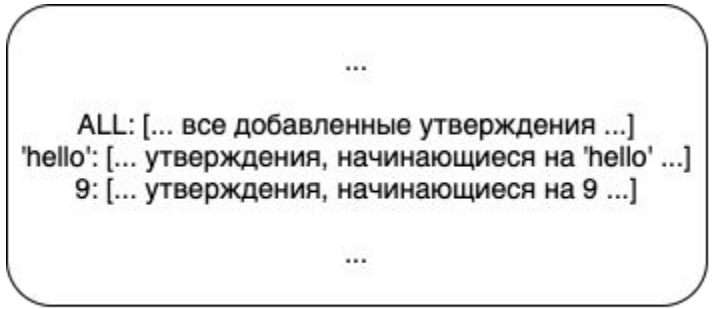
- Числа: `[0-9]+`
- Слова: `[a-zA-Z]+[0-9]*`
- Переменные: `\$[a-zA-Z]+[0-9]*`
- Особые имена: `\(|\)|@new|@rule|@apply|@and|@or|@not|<|>|\.`

Грамматика

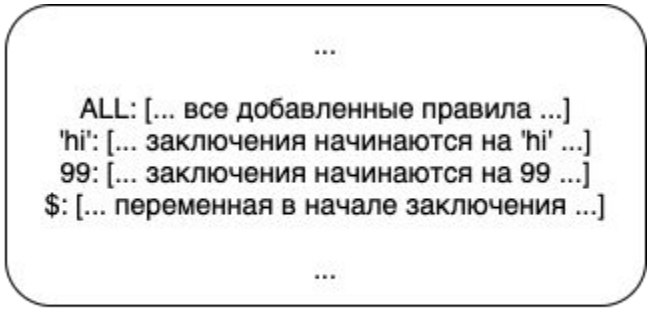
```
Command ::= '(' Insert | Query ')'
Insert ::= '@new' Entity+
Entity ::= '(' Assertion | Rule ')'
Assertion ::= ('(' Assertion ')' | Word | Number)*
Rule ::= '@rule' '(' SimpleQuery ')' ('(' Query ')')?
Query ::= SimpleQuery | AndQuery | OrQuery | NotQuery
AndQuery ::= '@and' InnerQueries
OrQuery ::= '@or' InnerQueries
NotQuery ::= '@not' InnerQuery
InnerQuery ::= '(' Query | Apply ')'
InnerQueries ::= InnerQuery+
Apply ::= '@apply' Predicate ApplyArguments
Predicate ::= '<' | '>' | Word
ApplyArguments ::= (Var | Word | Number)+
SimpleQuery ::= ('(' SimpleQuery ')' | Var | Word | Number)* ('.' Var)?
```

Реализация системы обработки запросов

- Python
- Лексер на основе регулярных выражений
- Парсер написан методом рекурсивного спуска
- Интерпретация AST
- Сущности базы данных хранятся в оперативной памяти
- Индексация



Утверждения



Правила

Тестирование

- Линтеры + проверка типов
- Юнит-тесты
- Покрытие 100%

Coverage report: 100%

Module ↑	statements	missing	excluded	branches	partial	coverage
Total	**613**	**0**	**7**	**172**	**0**	**100%**

coverage.py v5.3.1, created at 2021-01-18 22:52 +0500

Заключение

Реализован интерпретатор для языка логического программирования, являющегося языком запросов к примитивной базе данных

Возможные улучшения в будущем:

- GUI для системы обработки запросов
- Сохранение сущностей базы данных на диск
- Улучшение индексирования
- Создание новых особых форм и дополнительных предикатов для `@apply`