Операции над треками

Егор Горбунов

18 апреля 2016 г.

Описание операций и языка

Основные операции

- **Арифметические операции**

  Вариант №1: $a \in \mathbb{R}, X \in \mathbb{R}^n, Y \in \mathbb{R}^n$. Кодомен всех операций — $\mathbb{R}^n$.

  - $X \diamond Y := (x_1 \diamond y_1, x_2 \diamond y_2, \ldots, x_n \diamond y_n), \diamond \in \{+, -, \cdot, /\}$
  - $a \diamond Y := (a \diamond y_1, a \diamond y_2, \ldots, a \diamond y_n), \diamond \in \{\wedge, +, -, \cdot, /\}$
  - $X \diamond a := (x_1 \diamond a, x_2 \diamond a, \ldots, x_n \diamond a), \diamond \in \{\wedge, +, -, \cdot, /\}$

  Тут возникает вопрос о том, что делать с отрицательными значениями: имеет ли смысл их отрисовывать в браузере? имеют ли вообще смысл операция «−» и домножение на отрицательное число? Мне, кажется, что они не нужны и вполне заменяются предикатами и условным оператором, описанными ниже.

  Вариант №2: Всё то же, что и выше, но убираем операцию «−» и теперь $a \in \mathbb{R}_+$, а $X, Y \in \mathbb{R}^n_+$

  Вариант №3: операции все, что в 1-ом варианте, но: $x \diamond y \to \max(0, x \diamond y)$

  Тут нужно решить, но я склоняюсь ко 2-ому варианту.

- **Предикаты** Тут всё понятно. Пусть есть $X \in \mathbb{R}^n, Y \in \mathbb{R}^n$, тогда:

$$\diamond : (\mathbb{R}^n, \mathbb{R}^n) \to \{0, 1\}^n$$
$$X \diamond Y = (x_1 \diamond y_1, x_2 \diamond y_2, \ldots, x_n \diamond y_n)$$
$$\diamond \in \{<, >, \leqslant, \geqslant, =, \neq\}$$

- **Связки на предикатах** Опять же, ничего необычного: $A, B \in \{0, 1\}^n$, тогда определяем:

$$A \ \textbf{OR} \ B, \ A \ \textbf{AND} \ B, \ \textbf{NOT} \ A \in \{0, 1\}^n$$

- **Условный оператор** Пускай теперь $P \in \{0, 1\}^n, X \in \mathbb{R}^n, Y \in \mathbb{R}^n, a \in \mathbb{R}, b \in \mathbb{R}$, тогда полезна

следующая операция:

$$\textbf{if } P \textbf{ then } X \textbf{ else } Y := P \cdot X + (1 - P) \cdot Y$$
$$\textbf{if } P \textbf{ then } a \textbf{ else } Y := P \cdot a + (1 - P) \cdot Y$$
$$\textbf{if } P \textbf{ then } X \textbf{ else } b := P \cdot X + (1 - P) \cdot b$$
$$\textbf{if } P \textbf{ then } a \textbf{ else } b := P \cdot a + (1 - P) \cdot b$$

С помощью этой операции легко выражаются всевозможные фильтры.
PS: тут можно поспорить про синтаксис.

- **Псевдонимы для треков** Называния треков довольно длинные, поэтому полезно будет в запросах уметь ссылаться на конкретные треки через псевдонимы. Предлагаю описывать псевдонимы в самом начале запроса (как минимум потому, что autocompletion сможет их подхватить) так:

$$x = longTrackName1, \ldots, z = longTrackNameK \textbf{ in } Expression(x, \ldots, z)$$

- **Имя для нового трека** Результатом каждого запроса Q является новый трек, пользователь должен иметь возможность задать ему имя. Это будет делаться так:

$$newTrack \textbf{ := } Q$$

Если имя не указано, то оно будет сгенерировано или записано с именем lastResult, что перепишет предыдущий неименованый результат.

- **Показазать отрезки, где выполнен предикат** Это может быть реализовано через условный оператор.
- **Показать трэк** Все операции генерируют трэки, но не добавляют их к отображению. Чтобы добавить какой-то трэк, который был сгенерирован нужно специальное слово, например:

$$\textbf{show } TRACK$$

Дополнительные операции:

- **Изменение размера корзины** Такая возможность понадобится, если пользователь захочет совершать операции над треками с разными размерами корзин. Кажется, что в силу серьёзности данных, это нужно делать аккуратно. Совсем честно мы можем изменить число корзин гистограммы (трека) с n на m только если $n > m$ и $m|n$, т.е. если текущий размер корзины делит новый размер. При этом ещё возникают некоторые проблемы на границах рассматриваемого участка.
Вопрос: нужна ли возможность изменять размер корзины на произвольный? Если да, то видимо наиболее разумный способ реализации - это каждую корзину равномерно разбить на корзины раз-

мера 1, после чего уже пересчитать гистограмму для нового размера корзины.

$$\textbf{resize}(\mathrm{track},\ \mathrm{bpNewBinSize})$$

- **Операции на подотрезках** Назвал страшно, а речь идёт о построении новых треков посредством сканирования окном гистограммы (трека) и подсчёта в окне какой-нибудь функции: $\max$, $\min$, $+$, $\times$, $\ldots$ Синтаксис в духе:

$$\textbf{window}(\mathrm{track},\ \mathit{windowSize},\ \mathrm{Function})$$

Эта штука перекликается с изменением размера гистограммы, но тут $\mathit{windowSize}$ — это число бинов (столбцов гистограммы), а не bp. $\mathrm{Function}$ действует из $\mathbb{R}^{\mathit{windowSize}}$ в $\mathbb{R}$ и пока кажется, что можно предоставить некоторый фиксированный выбор этих возможных функций: MEAN, MAX, MIN, SUM, MUL, VAR, MEDIAN, $\mathrm{NORMALIZE}$, хотя, конечно, это не выглядит гибко, но если подумать, то многие функции можно выразить через имеющуюся арифметику на треках и изменение размера корзины.

Примеры использования

```
newTrack := if (track1 >= track2 AND track1 <= track2 + 100) then track1 else track2

newTrack := X = track1, Y = track2 in (if X > Y then X else Y)

newTrack := 1 / (2 ^ track1)

newTrack := X = track1 in (if track2 > 20 then X else 0)

Y - X + (if (if Y > 0 then Y else X) < 0 then Z else 0) * Y
```

Грамматика (описание языка)

```
STATEMENT = ID := EXPR
        | EXPR
        | show ID
EXPR = ID
        | NUM
        | (EXPR)
        | EXPR + EXPR
        | EXPR - EXPR
        | EXPR * EXPR
        | if PRED then EXPR else EXPR
PRED = TERM
        | not TERM
        | (PRED)
        | PRED or PRED
        | PRED and PRED
TERM = true | false | REL
REL = EXPR <= EXPR | EXPR >= EXPR | EXPR == EXPR | EXPR < EXPR | EXPR > EXPR
NUM = [1-9][0-9]* | 0
ID = [a-zA-Z_][0-9a-zA-Z_]*
```

Преобразованная грамматика

- Главное выражение:

```
S -> %track_name% <- E
      | show %track_name%
      | E
```

- Арифметика на треках:

```
E -> E + T
      | E - T
      | T
T -> T * X
      | T / X
      | X
X -> X ^ P
      | P
P -> (E)
      | if PRED then E else E
      | %track_name%
      | %number%
```

- Предикаты:

```
PRED -> PRED or A
      | A
A -> A and B
      | B
B -> (PRED)
      | not PRED
      | true
      | false
      | R
R -> E < E
      | E > E
      | E <= E
      | E >= E
      | E == E
      | E != E
```

Грамматика без левой рекурсии

```
STATEMENT -> ID ":=" TRACK_EXPR
             | "show" ID
             | TRACK_EXPR
TRACK_EXPR -> E_TERM SUM_SUB
SUM_SUB -> "+" E_TERM SUM_SUB
          | "-" E_TERM SUM_SUB
          |.
E_TERM -> E_FACTOR MUL_DIV
MUL_DIV -> "*" E_FACTOR MUL_DIV
          | "/" E_FACTOR MUL_DIV
          | .
E_FACTOR -> "(" TRACK_EXPR ")"
          | "if" PREDICATE "then" TRACK_EXPR "else" TRACK_EXPR
          | ID
          | NUMBER
PREDICATE -> B_TERM OR_PRED
OR_PRED -> "or" B_TERM OR_PRED
          |.
B_TERM -> NOT_FACTOR AND_PRED
AND_PRED -> "and" NOT_FACTOR AND_PRED
          |.
NOT_FACTOR -> "not" B_FACTOR
          | B_FACTOR
B_FACTOR -> "(" PREDICATE ")"
          | RELATION
          | "true"
          | "false"
RELATION -> TRACK_EXPR "<=" TRACK_EXPR
          |TRACK_EXPR ">=" TRACK_EXPR
          |TRACK_EXPR "==" TRACK_EXPR
          |TRACK_EXPR "!=" TRACK_EXPR
          |TRACK_EXPR ">" TRACK_EXPR
          |TRACK_EXPR "<" TRACK_EXPR
ID -> [a-zA-Z_][a-zA-Z_0-9]*
NUM -> %float_num_regex%
```

Parsing Expression Grammar

```
STATEMENT <- ID ":=" TRACK_EXPR
           / "show" ID
           / TRACK_EXPR
TRACK_EXPR <- E_TERM ("+" E_TERM / "-" E_TERM)*
E_TERM <- E_FACTOR ("*" E_FACTOR / "/" E_FACTOR)*
E_FACTOR <- "(" TRACK_EXPR ")"
         / "if" PREDICATE "then" TRACK_EXPR "else" TRACK_EXPR
         / ID
         / NUMBER
PREDICATE <- B_TERM ("or" B_TERM)*
B_TERM <- NOT_FACTOR ("and" NOT_FACTOR)*
NOT_FACTOR <- "not" B_FACTOR
         / B_FACTOR
B_FACTOR <- "(" PREDICATE ")"
         / RELATION
         / "false"
         / "true"
RELATION <- TRACK_EXPR "<=" TRACK_EXPR
          / TRACK_EXPR ">=" TRACK_EXPR
          / TRACK_EXPR "==" TRACK_EXPR
          / TRACK_EXPR "!=" TRACK_EXPR
          / TRACK_EXPR ">" TRACK_EXPR
          / TRACK_EXPR "<" TRACK_EXPR
ID <- [a-zA-Z_][a-zA-Z_0-9]*
NUM <- %float_num_regex%
```