# Введение в ИИ на примере языка Prolog

## Арифметика

https://github.com/Inscriptor/IntroductionToAI/tree/master/pdf

Федеральное государственное автономное образовательное учреждение высшего образования
«Новосибирский национальный исследовательский государственный университет»

9 октября 2019 г.

# Арифметика в Prolog

# Реализация арифметики в Prolog

## Численные типы и операции

В языке Prolog реализованы численные типы данных и операции над ними. Для проверки типов в программе применяются соответствующие встроенные предикаты.

Численные типы данных:

- **Integer** — целые числа: $1, 2, 3, 4, -100, 1001$. Предикат `integer/1` возвращает `True` в случае, если переданный терм представляет целое число.
- **Float** — вещественные числа: $1.2, 3.14, 2.7$. Предикат `float/1` возвращает `True`, если терм представляет вещественное число.
- **Rational** — рациональные числа: $\frac{1}{3}, \frac{2}{7}$. Для проверки используется предикат `rational/1`. Данный предикат считает рациональными и все целые числа.

Рациональные числа записываются с помощью специального предиката `rdiv`, который работает как дробная черта.

**Пример**

$\frac{1}{3}$ = `1 rdiv 3. X is (1 rdiv 3) * 3` $\Rightarrow$ `X = 1.`

# Реализация арифметики в Prolog

## Численные типы и операции

| Арифметическое выражение | Запись в синтаксисе Prolog |
|---|---|
| $10 + 5 = 15$ | `15 is 10 + 5.` |
| $10 \cdot 5 = 50$ | `50 is 10 * 5.` |
| $10 - 5 = 5$ | `5 is 10 - 5.` |
| $5 - 10 = -5$ | `-5 is 5 - 10.` |
| $10 \div 5 = 2$ | `2 is 10 / 5.` |
| $10 \div 4 = 2.5$ | `2.5 is 10 / 4.` |
| $10 \div 4 = 4 \cdot 2 + 2$ | `2 is mod(10,4).` |

# Реализация арифметики в Prolog

## Встроенные арифметические функции

Встроенные функции реализуют основные численные операции. Данных функций в большинстве случаев хватает для реализации логических программ.

**Пример**

*sin, cos, tan, log, log10, exp, **, sqrt, ceil, floor, round, abs, max, min, », «*

# Использование арифметических операций в Prolog-программах

## Унификация и вычисление

- Арифметические вычисления реализованы в Prolog как некоторое полезное дополнение к базовому функционалу.
- Базовый функционал Пролога заключается в унификации термов.
- Отдельно выражения вида `10 + 5` рассматриваются интерпретатором как термы с функтором + и двумя аргументами 10 и 5.
- Запись `10 + 5` — это более удобная нотация для выражения терма `+(10,5)`.
- Чтобы сообщить интерпретатору о необходимости вычислить (evaluate) выражение $10 + 5$, вместо унификации терма `+(10,5)`, необходимо использовать специальный предикат `is/2`.

# Использование арифметических операций в Prolog-программах

## Унификация и вычисление

Предикат `is` принимает два аргумента: число или переменную и арифметическое выражение, которое требуется вычислить. Запись `(Num is Expr)` является более удобной нотацией для терма `is(Num, Expr)`.

### Пример

*`15 is 10 + 5.` ⇔ `is(15,+(10,5)).`*

*`X is (2*5 + 10) / 4.` ⇔ `is(X,/(+(*(2,5),10),4)).`*

# Использование арифметических операций в Prolog-программах

## Унификация и вычисление

Так как арифметические вычисления не являются естественной частью Пролога, их использование имеет свои ограничения.

- Вычисляемое арифметическое выражение должно быть справа. Допустима запись `X is 10 + 5`, но недопустимо обратное `10 + 5 is X`. В целом, любая переменная в правой части на момент вычисления должна иметь численное значение. В противном случае будет выведено сообщение об ошибке.
- Правая и левая части должны иметь численные типы (левая часть также может быть переменной). Арифметика работает не так как унификация, соответственно, передача для вычисления типов данных, отличных от численных, также приведет к ошибке.

# Использование арифметических операций в Prolog-программах

## Унификация и вычисление

Для примера рассмотрим функцию инкремента.

```
inc(X,Y) :- Y is X + 1.
```

- Запрос `inc(10,11)` вернет ответ `True`.
- Запрос `inc(10,I)` вернет ответ `I = 11`.
- Однако запрос `inc(X,11)` вызовет ошибку.

# Использование арифметических операций в Prolog-программах

## Операции сравнения

| Арифметическое выражение | Запись в синтаксисе Prolog |
|---|---|
| $x < y$ | `X < Y.` |
| $x \leqslant y$ | `X =< Y.` |
| $x = y$ | `X =:= Y.` |
| $x \neq y$ | `X =\= Y.` |
| $x \geqslant y$ | `X >= Y.` |
| $x > y$ | `X > Y.` |

# Использование арифметических операций в Prolog-программах

## Арифметика применительно к спискам

Одно из главных применений численной арифметики в Prolog состоит в вычислении характеристик других структур данных. В частности — списков. Вспомним предикат `listLen/2`, вычисляющий длину списка:

```
listLen([],0).
listLen([H|T],Len) :- listLen(T,TailLen), Len is TailLen + 1.
```

# Использование арифметических операций в Prolog-программах

## Арифметика применительно к спискам

Более эффективно это можно реализовать, используя хвостовую рекурсию и дополнительную память, где будет храниться текущее значение длины списка.

```
accLen([_|T],CurrLen,Len) :- NewLen is CurrLen + 1,
            accLen(T,NewLen,Len).
accLen([],L,L).
listLen(List,Len) :- accLen(List,0,Len).
```

# Пример

## Максимальный элемент списка

```
accMax([H|T],CurrMax,Max) :- H > CurrMax,accMax(T,H,Max).
accMax([H|T],CurrMax,Max) :- H =< CurrMax,accMax(T,CurrMax,Max).
accMax([],Max,Max).
listMax([],_) :- !,fail.
listMax([H|T],Max) :- accMax([H|T],H,Max),!.
```

# Пример

## Сумма элементов списка

```
sumVec([H|T],_buffer,Sum) :- _current is _buffer + H,
            sumVec(T,_current,Sum).
sumVec([],Sum,Sum).
summarize(List,Sum) :- sumVec(List,0,Sum).
```

# Пример

## Скалярное произведение векторов

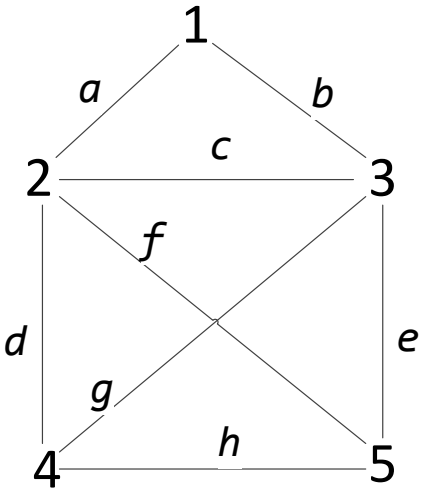
```
dotProd([H1|T1],[H2|T2],_buffer,Dot) :- _curr is _buffer + H1*H2,
            dotProd(T1,T2,_curr,Dot).
dotProd([],[],Dot,Dot).
dotproduct(_v1,_v2,Product) :- dotProd(_v1,_v2,0,Product).
```

# Задачи для самостоятельной работы

# Самостоятельная работа

1. Нарисовать конверт, не отрывая карандаша от бумаги и не проводя два раза по одной и той же линии. Начальная точка задается в качестве параметра. Если путь существует, то следует вывести его, в противном случае — вернуть `false`.
2. Усовершенствовать решение таким образом, чтобы программа на вход принимала помеченный граф (или существовала возможность описать граф и сохранить его во внутренней базе данных), и начальное положение "карандаша и возвращала либо последовательность проходов, необходимых для решения задачи, либо `false`, если задача не имеет решения.

# Самостоятельная работа



# Самостоятельная работа

3. Даны два сосуда объемом 3 и 5 литров. Также имеется источник, из которого можно наполнить сосуды и куда можно вылить лишнюю воду. Определить последовательность переливаний, необходимую для получения 4 литров воды в одном из сосудов.
4. Усовершенствовать решение предыдущей задачи таким образом, чтобы программа принимала на вход начальное состояние: целочисленные объемы сосудов V1 и V2, начальное количество воды в каждом сосуде и требуемое количество воды. При этом V1 и V2 взаимно просты. По начальному состоянию программа должна определить последовательность переливаний, необходимую для достижения целевого объема. Если задача не имеет решения — программа должна возвращать `false`.