Введение в ИИ на примере языка Prolog

Операторы и термы

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

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

11 ноября 2019 г.

Сравнение термов

Сравнение термов

Унификация и сравнение

- Язык Prolog, помимо операции унификации $=$ предлагает также предикат $==$, реализующий сравнение термов.
- Ответ на вопрос вида `term1 == term2.` будет равен `true` тогда и только тогда, когда термы идентичны друг другу.
- Предикат $==$ *не присваивает значения переменным*.

Сравнение термов

Унификация и сравнение

Рассмотрим следующие примеры.

```
term == term. (true)
term1 == term2. (false)
term == 'term'. (true)
```

Сравнение термов

Унификация и сравнение

Теперь добавим переменных.

```
X == Y.
```

Сравнение термов

Унификация и сравнение

Теперь добавим переменных.

```
X == Y.
X == a.
```

Сравнение термов

Унификация и сравнение

Теперь добавим переменных.

```
X == Y.
X == a.
X = a, X == a.
```

Сравнение термов

Унификация и сравнение

Теперь добавим переменных.

```
X == Y.
X == a.
X = a, X == a.
X = Y, X == Y.
```

Сравнение термов

Унификация и сравнение

- Таким образом, можно сказать, что операции $=$ и $==$ существенно отличаются.
- Тем не менее, можно говорить, что $==$ является *более строгим* ограничением, чем $=$.
- Для любых двух термов `t1` и `t2`, из того, что выполняется `t1 == t2`, всегда следует, что `t1 = t2` также выполняется.

Сравнение термов

Отрицания операций

- Операция, обратная сравнению термов ==, записывается как \==.
- Очевидно, что `t1 \== t2` истинно тогда и только тогда, когда `t1 == t2` ложно.

```
term \== term.
term1 \== term2.
term \== 'term'.
X \== a.
X \== Y.
```

Термы с нестандартной нотацией

Термы с нестандартной нотацией

- Некоторые термы, которые, на первый взгляд, выглядят по-разному, Prolog интерпретирует как идентичные.
- Например, как было показано в примерах выше, термы вида `term` и `'term'` идентичны.
- В первую очередь, подобные термы нужны для удобства написания кода. Нотации, которые удобны для разбора интерпретатором, обычно не являются удобными для программиста.
- Гораздо удобнее писать термы в удобной для себя нотации, чтобы затем интерпретатор разбирал их аналоги, записанные в нотации, удобной для него.

Термы с нестандартной нотацией

Арифметические термы

Хорошим примером специфических нотаций являются арифметические термы. Как уже было сказано ранее, все арифметические операции $/, -, *, +$ и другие, являются функторами, а выражения вида $2 + 3$ — термами.

- `2+3 == +(2,3).`
- `2-3 == -(2,3).`
- `2*3 == *(2,3).`
- `3*(7+5) == *(3,+(7,5)).`

Термы с нестандартной нотацией

Арифметические термы

То же касается и арифметических операций сравнения.

- `(2 < 3) == <(2, 3).`
- `(2 =< 3) == =<(2, 3).`
- `(2 =:= 3) == =:=(2, 3).`
- `(2 =\= 3) == =\=(2, 3).`
- `(2 > 3) == >(2, 3).`
- `(2 >= 3) == >=(2, 3).`

Термы с нестандартной нотацией

Арифметические термы

Ниже представлены все, известные нам, операции сравнения и их отрицания.

`=`	Предикат унификации. Выполняется, если может унифицировать свои аргументы и фейлится в противном случае.
`\=`	Отрицание унификации. Выполняется в случае провала операции = и наборот.
`==`	Предикат идентичности. Выполняется, если его аргументы идентичны и фейлится в противном случае.
`\==`	Отрицание идентичности.
`=:=`	Предикат арифметического равенства. Выполняется, если его аргументы числа и равны между собой.
`=\=`	Предикат арифметического неравенства.

Термы с нестандартной нотацией

Списки как термы

- Списки также являются хорошим примером user friendly нотации термов.
- В привычной записи, список задается путем перечисления его элементов через запятую и заключения их в квадратные скобки.
- Пролог задает списки с помощью двух термов: `[]`, представляющего пустой список, и бинарного функтора `'.'` для построения непустых списков.
- Пустой список это в точности терм `[]`. Длина такого списка равна 0.
- Непустой список имеет вид `.(term,list)`, где `term` — произвольный терм, а `list` — произвольный список.

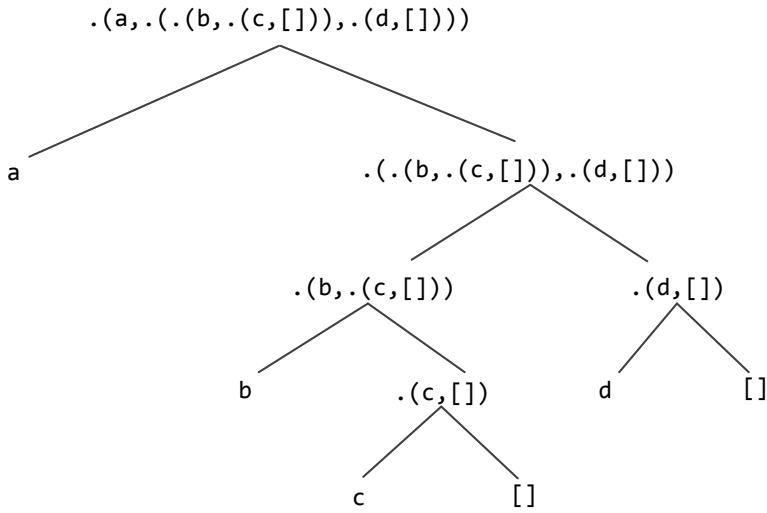
Термы с нестандартной нотацией

Списки как термы

- `.(a,[]) == [a].`
- `.(f(a,b),[]) == [f(a,b)].`
- `.(a,.(b,[])) == [a,b].`
- `.(a,.(b,.(f(x,y),[]))) == [a, b, f(x,y)].`
- `.(.(.(a,[]),[]),[]) == [[[a]]].`
- `.(a,.(.(b,.(c,[])),.(d.[]))) == [a,[b,c],d].`

Термы с нестандартной нотацией

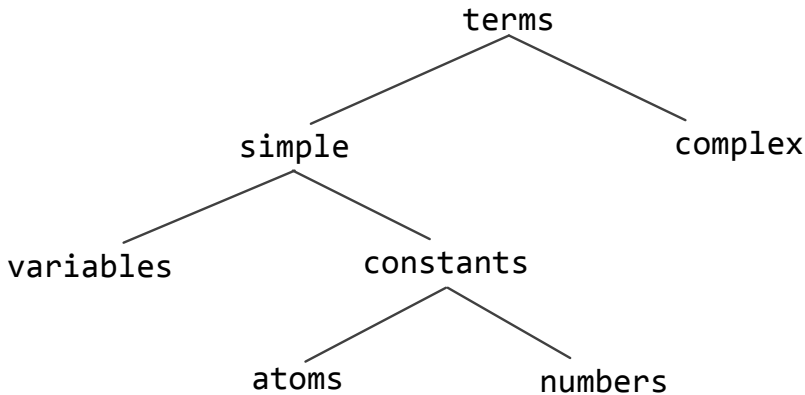
Списки как термы



Исследование термов

Исследование термов

Типы термов



Исследование термов

Типы термов

`atom/1`	Проверка, является ли аргумент атомарным
`integer/1`	Проверка, является ли аргумент целым числом
`float/1`	Является ли аргумент числом с плавающей запятой
`number/1`	Является ли аргумент числом (целым или вещественным)
`atomic/1`	Является ли аргумент константным
`var/1`	Является ли аргумент неозначенным
`nonvar/1`	Является ли аргумент означенным

Исследование термов

Структура термов

Пусть имеется некоторый терм `T`, и мы не знаем, как он выглядит, и какие аргументы имеет. Какую информацию мы хотели бы знать, чтобы иметь возможность в дальнейшем использовать данный терм?

Исследование термов

Структура термов

Пусть имеется некоторый терм `T`, и мы не знаем, как он выглядит, и какие аргументы имеет. Какую информацию мы хотели бы знать, чтобы иметь возможность в дальнейшем использовать данный терм?

1. Каков функтор терма `T`.
2. Какова арность терма `T`.
3. Каковы аргументы терма `T`.

Исследование термов

Структура термов

- Предикат `functor/3` отвечает на первые два вопроса.
- Принимая на вход составной терм в качестве первого аргумента, он присваивает его функтор и арность своим второму и третьему аргументу соответственно.

Например:

```
?- functor(f(x,y), F, A).
F = f
A = 2
?- functor(f, F, A).
F = f
A = 0
```

Исследование термов

Структура термов

При этом, с помощью предиката `functor/3` мы можем не только находить функторы и арности термов, но и **конструировать термы с нужными функтором и арностью**.

```
functor(T,f,3).
T = f(_21882, _21884, _21886).
```

Обратите внимание, что либо первый, либо второй и третий аргументы должны быть определены.

Исследование термов

Структура термов

На третий вопрос о структуре неизвестного терма отвечают два предиката. Один из них — это предикат `arg/3`. Он принимает на вход целое число N, составной терм `T` и возвращает N-й аргумент терма `T`. Также может применяться для присваивания значения N-му аргументу терма `T`.

```
?- arg(2, f(x,y), Second).
Second = y
?- arg(2, f(x,Y), y).
Y = y
?- arg(3, f(x,y), Third).
false
```

Как можно видеть, нумерация аргументов терма начинается с 1.

Исследование термов

Структура термов

Вторым предикатом, позволяющим получить доступ к аргументам терма, является `'=..'/2`. Он принимает составной терм и возвращает список, где первым элементом идет функтор терма, а далее перечислены все его аргументы. Соответственно, задав вопрос `'=..'(f(x,y), S)`, получим ответ `S = [f,x,y]`. Данный предикат называется univ и может использоваться как инфиксный оператор.

```
?- f(x, y, z) =.. Structure.
Structure = [f, x, y, z]
?- f(X) =.. F.
F = [f, X]
?- T =.. [g, x, y, z].
T = g(x, y, z)
```

Операторы

Операторы

Свойства операторов

Операторы — это функторы бинарных или унарных термов, определенные специальным образом. Существует три типа операторов.

1. *Инфиксный оператор* записывается между своими аргументами.
2. *Префиксный оператор* записывается перед своим аргументом.
3. *Постфиксный оператор* записывается после своего аргумента.

Операторы

Свойства операторов

- Каждый оператор имеет *приоритет*.
- Например, приоритет операции сложения $+$ **выше** приоритета операции умножения $*$.
- Аналогично, приоритет оператора `is` выше приоритета любой арифметической операции.
- Приоритет оператора задается числом от 0 до 1200. Чем больше число, тем выше приоритет.

Операторы

Свойства операторов

- Стандартная нотация выражений в Прологе не имеет неоднозначностей. Например, в выражении `is(11,+(2,*(3,3)))` всегда понятно, в какой последовательности должны быть выполнены операции.
- С другой стороны user friendly нотации могут быть неоднозначны. Выше был пример `2 =:= 3 == =:=(2,3)`. Операторы `=:=` и `==` обладают равными приоритетами, поэтому данное выражение не может быть оценено без постановки скобок.

Операторы

Свойства операторов

Рассмотрим пример `X is 1 + 2 + 3`. В данном случае Prolog не выдаст ошибку, а разрешит выражение без постановки скобок.

```
?- 1 + 2 + 3 == +(1, +(2,3)).
false
?- 1 + 2 + 3 == +(+(1,2), 3).
true
```

Операторы

Свойства операторов

- Prolog имеет представление об *ассоциативности* операторов.
- Операция сложения левоассоциативна. Это означает, что выражение справа от оператора сложения должно иметь строго меньший приоритет.
- Приоритет выражения равен приоритету его оператора или 0.
- Операторы `=:=` и `==` не имеют ассоциативности.

Операторы

Описание оператора

Определение нового оператора в Prolog выглядит следующим образом:

```
:- op(Precedence, Type, Name).
```

Соответственно, чтобы задать новый оператор, необходимо знать о нем три вещи:

1. Тип (префиксный, инфиксный или постфиксный).
2. Приоритет.
3. Ассоциативность.

Операторы

Описание оператора

В описании типа оператора `f` задает расположение функтора относительно аргументов, которые обозначаются буквами `x` или `y`.

infix	xfx, xfy, yfx
prefix	fx, fy
postfix	xf, yf

Буквой `x` обозначается аргумент, приоритет которого должен быть строго меньше приоритета самого оператора, тогда как буквой `y` обозначается аргумент, приоритет которого может быть меньше либо равным приоритету оператора. Таким образом, оператор типа `yfx` — это инфиксный оператор, обладающий левой ассоциативностью, а оператор `xfx` — инфиксный оператор, не обладающий ассоциативностью.

Операторы

Описание оператора

Типы и приоритеты некоторых встроенных операторов.

```
:- op(1200, xfx, [:-, ->]).
:- op(1200, fx, [:-, ?- ]).
:- op(1200, xfy, [;]).
:- op(1000, xfy, [',']).
:- op(700, xfx, [=, is, =.., ==, \==, =:=, =\=, <, >, =<, >=]).
:- op(500, yfx, [+, -]).
:- op(500, fx, [+, -]).
:- op(300, xfx, [mod]).
:- op(200, xfy, [^]).
```