Разбор разного рода выражений на ProLog:

main::

```
men(clara).
```

тесты:

```
2 ?- men(clara).
true.

3 ?- men(X).
X = clara.

4 ?- men(fedor).
false.
```

База использования ProLog - поиск соответствий:

```
speciality(X, tech_translator) :- studied_languages(X), studied_technical(X).
speciality(X, programmer) :- studied(X, mathematics), studied(X, compsciense).
speciality(X, lit_translator) :- studied_languages(X), studied(X, literature).

studied_languages(X) :- studied(X, eng).
studied_languages(X) :- studied(X, ger).
studied_technical(X) :- studied(X, mathematics).
studied_technical(X) :- studied(X, compsciense).

studied(petr, mathematics).
studied(petr, compsciense).
studied(petr, eng).
studied(vasya, ger).
studied(vasya, literature).
```

Воспринимать нужно так:
Строки 10-14 - задаем описание студент + что он сдавал
Строки 5 - 8 задаем правила, что считается за “изучал языки”, за “изучал технические предметы”
Строки 1-3 задаем специальности для студентов на основе их предметов изучения

Тесты:

```
6 ?- studied(petr,X).
X = mathematics ;
X = compsciense ;
X = eng.

7 ?- studied(X, eng).
X = petr.

8 ?- studied_languages(X).
X = petr ;
X = vasya.

9 ?- studied_languages(petr).
true .

10 ?- studied_technical(X).
X = petr ;
X = petr.

11 ?- speciality(X,tech_translator).
X = petr ;
X = petr ;
false.

12 ?- speciality(X,Y).
X = petr,
Y = tech_translator ;
X = petr,
Y = tech_translator ;
X = petr,
Y = programmer ;
X = vasya,
Y = lit_translator.

13 ?- speciality(_,_).
true ;
true ;
true ;
true.
```

определение натурального числа:

```
natural(0). %/*0 - натуралное*/
natural(s(X)) :- natural(X). %число, на 1 большее - натуральное, если само число натуральное
```

tests:

```
15 ?- natural(0).
true.

16 ?- natural(X).
X = 0 ;
X = s(0) ;
X = s(s(0)) ;
X = s(s(s(0))) ;
X = s(s(s(s(0)))) ;
X = s(s(s(s(s(0))))) ;
X = s(s(s(s(s(s(0)))))) ;
X = s(s(s(s(s(s(s(0))))))) ;
X = s(s(s(s(s(s(s(s(0)))))))) ;
X = s(s(s(s(s(s(s(s(s(0))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) ;
X = s(s(s(s(s(s(s(s(s(...)))))))))) .

17 ?- natural(s(s(0))).
true.
```

Определение сложения чисел:

```
5   add(0, X, X).
6   add(s(X), Y, s(Z)) :- add(X,Y,Z).
```

Данное определение основано на таком свойстве:
0 + X = X;
X+Y = Z => (X+1) + Y = Z + 1; так как (X+1) + Y = X + Y + 1 = Z + 1;

tests:

```
18 ?- add(s(0),s(0),X).
X = s(s(0)).

19 ?- add(s(s(0)), s(s(s(0))), X).
X = s(s(s(s(s(0))))).

20 ?- add(s(s(0)), X, s(s(s(s(s(0)))))).
X = s(s(s(0))).
```

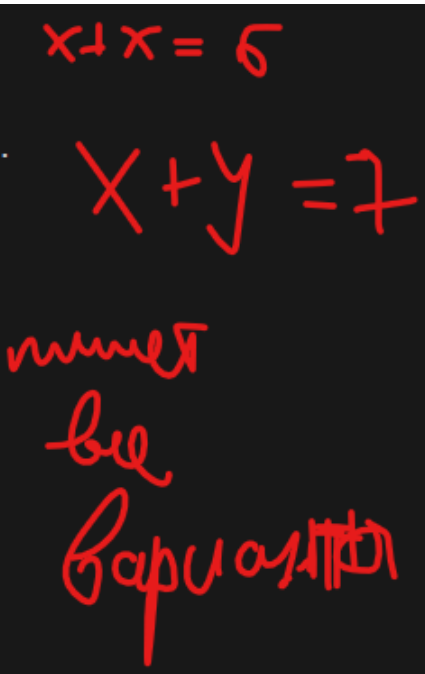
1+1=2=X

2+3=5=X

2+X=5, X=3

```
21 ?- add(X,X, s(s(s(s(s(s(0)))))).
X = s(s(s(0))) ;
false.

22 ?- add(X,Y,s(s(s(s(s(s(s(0)))))))).
X = 0,
Y = s(s(s(s(s(s(0)))))) ;
X = s(0),
Y = s(s(s(s(s(0))))) ;
X = s(s(0)),
Y = s(s(s(s(0)))) ;
X = s(s(s(0))),
Y = s(s(s(0))) ;
X = s(s(s(0))),
Y = s(s(0)) ;
X = s(s(s(s(0)))),
Y = s(0) ;
X = s(s(s(s(s(0))))),
Y = 0 ;
false.
```



Определение умножения чисел:

```
multiply(0, s(X), 0).
multiply(s(X), Y, R2) :- multiply(X, Y, R1), add(R1, Y, R2).
```

Описание:

1) 0*X = 0;
2) x*y = r1;
3) (x + 1)*y = r2; => x*y + y = r2;
4) x*y + y = r1 + y = r2;

tests:

```
23 ?- multiply(0,X,Y).
X = s(_),
Y = 0.

24 ?- multiply(0,X,0).
X = s(_).

25 ?- multiply(s(s(0)),s(s(0)),X).
X = s(s(s(s(0)))).

26 ?- multiply(s(s(0)),X,s(s(0))).
X = s(0) ;
false.
```

```
28 ?- multiply(s(s(s(0))), s(s(0)), X).
X = s(s(s(s(s(s(0)))))).

29 ?- multiply(X, s(s(0)), s(s(s(s(s(s(0)))))).
X = s(s(s(0))) ;
```

Факториал числа:

```
fact(0, 1).
fact(N, Res) :-
    N>0,
    M is N-1,
    fact(M, Res1),
    Res is N*Res1.
```

Описание:

1) факториал 0 = 1;
2) Если число положительное, то уменьшаем его на единичку и берем факториал от числа-1;
3) В какой-то момент мы доберемся до элементарного правила - правило 1, после чего будем последовательно умножать 1 на 2, на 3, на 4 и так далее.

Тесты:

```
2 ?- fact(0,X).
X = 1 ;
false.

3 ?- fact(3,X).
X = 6 ;
false.

4 ?- fact(5,X).
X = 120 ;
false.
```

И даже так!

```
6 ?- fact(21,X).
X = 51090942171709440000 .

7 ?- fact(100,X).
X = 933262154439441526816992388562667004907159682643816214685
```

даже посчитает 1000! и так далее…

Оптимизированные числа Фибоначчи

```
45  fib(1,1,0).
46  fib(X,Y,Z) :-
47      X>0,
48      X1 is X-1,
49      fib(X1, Z, Y1),
50      Y is Z+Y1.
51
52
```

Описание:

1) задаем начальное правило - первое число Фибоначчи == 1, 0 - предыдущее число Фибоначчи

2) Теперь для n-го числа Фибоначчи мы будем спускаться вниз,.уменьшая искомое число на 1, пока не доберемся до элементарного правила.

Тесты:

```
2 ?- fib(1,Y,Z).
Y = 1,
Z = 0 ;
false.

3 ?- fib(5,Y,Z).
Y = 5,
Z = 3 ;
false.

4 ?- fib(7,Y,Z).
Y = 13,
Z = 8 ;
false.

5 ?- fib(100,Y,Z).
Y = 354224848179261915075,
Z = 218922995834555169026 ;
false.

6 ?- fib(200,Y,Z).
Y = 280571172992510140037611932413038677189525,
Z = 173402521172797813159685037284371942044301 ;
false.
```

```
8 ?- fib(X, 120, 24).
ERROR: Arguments are not sufficiently instantiated
ERROR: In:
ERROR:   [11] _112>0
ERROR:   [10] fib(_138,120,24) at c:/users/fedor/onedrive/рабочий стол/logprog/arifmetics.pl:47
ERROR:    [9] toplevel_call(user:user: ...) at c:/program files/swipl/boot/toplevel.pl:1173
```

Жаль она не может найти позицию известного нам числа Фибоначчи.

Далее пойдут примеры из лекции на списки + переложение списков на разные рекурсии(обычная рекурсия - принцип уменьшения, хвостовая рекурсия - принцип уменьшения, но без возвращения элементов “наверх”, корекурсия - принцип наращивания(на момент написания вообще понятия не имею, как такую фигню реализовывать в ProLog)).

Итак, определение длины списка:

обычная рекурсия:

```
1   length([], 0).
2   length([_|Y], N) :-
3       length(Y, N1),
4       N is N1 + 1.
5
```

Ну, вроде все понятно:

1) если в списке нет элементов, то его длина нулевая
2) отсекаем от списка его головной элемент
3) вызываем length для подсчета длины хвоста списка
4) на каждом шаге переменную длины увеличиваем на 1

Тесты:

```
23 ?- length([],X).
X = 0.

24 ?- length_tail([a,b,c],X).
X = 3.

25 ?- length([a,b,c,c,d,e,f,g,h,j,k,l,m,n,p],X).
X = 15.
```

Хмм, попробуем собрать хвостовую рекурсию для этой же функции:

```
11  length_tail(List, Length):-
12      length_tail(List, 0, Length).
13
14  length_tail([], Length, Length):-!.  % ! - определитель остановки
15  length_tail([_|End], Summ, Length):-
16      NewSumm is Summ + 1,
17      length_tail(End, NewSumm, Length).
```

Описание:

1) вызывать будем length_tail от двух аргуметов (список и длина), который будет вызывать length_tail от трех элементов (+0);
2) терминирующий элемент - пустой массив;
3) на каждом шаге увеличиваем Summ на 1 и уменьшаем список на 1 элемент;

Тесты:

```
24 ?- length_tail([a,b,c],X).
X = 3.

25 ?- length([a,b,c,c,d,e,f,g,h,j,k,l,m,n,p],X).
X = 15.

26 ?- length_tail([],X).
X = 0.

27 ?- length_tail([a],X).
X = 1.

28 ?- length_tail([a,b,c],X).
X = 3.

29 ?- length_tail([a,b,c,d,e,f,g,h,k],X).
X = 9.
```

Проверка на принадлежность элемента списку:

```
20  member(X, [X|_]).
21  member(X, [_|Y]) :-
22      member(X, Y).
23
```

Собственно говоря, уменьшаем список, пока не наткнемся на искомое значение.

Тесты:

```
31 ?- member(a, [b,c,a,g,r]).
true ;
false.

32 ?- member(X, [b,c,a,g,r]).
X = b ;
X = c ;
X = a ;
X = g ;
X = r ;
false.

33 ?- member(a, [b,c,X,g,r]).
X = a ;
false.
```

Конкатенация двух списков

```
25  append([], X, X).% Если первый список пустой, третий список равен второму списку.
26  append([A|X], Y, [A|Z]) :-
27      append(X,Y,Z).
```

Описание:
По сути в X постепенно будем записывать элементы первого списка, и когда он закончится, просто перенесем элементы второго списка. Вопрос для обсуждения - хвостовая ли это рекурсия? - да
Тесты:

```
35 ?- append([d],[a,b,c],X).
X = [d, a, b, c].

36 ?- append([d],X,[d,k,s,l]).
X = [k, s, l].

37 ?- append(X,Y,[a,b,c,d,l]).
X = [],
Y = [a, b, c, d, l] ;
X = [a],
Y = [b, c, d, l] ;
X = [a, b],
Y = [c, d, l] ;
X = [a, b, c],
Y = [d, l] ;
X = [a, b, c, d],
Y = [l] ;
X = [a, b, c, d, l],
Y = [] ;
false.
```

Вторая реализация, возможно кому-то покажется проще:

```
30   append_2([], L, L) :- !. % Если первый список пустой, третий список равен второму списку.
31   append_2([H|T], L, Result) :- append_2(T, L, TailResult), Result = [H|TailResult].% Рекур
32
```

Метод такой же, просто теперь мы все стало явно. НО мы потеряли возможность перебора значений! Смотрите сами - те же самые тесты:

```
 6 ?- append_2([a,b,c], X, [a,b,c,o]).
X = [o].

 6 ?- append_2([a,b,c], [k,l], X).
X = [a, b, c, k, l].

 7 ?- append_2(X,Y,[a,b,c,d,e,f]).
X = [],
Y = [a, b, c, d, e, f].
```

Задание 1: Определить следующего за Ивановым ученика

append(_,[ivanov,X|_],L).

Попробуем-ка навоять:

```
8 ?- append(_, [ivanov, X|_], [petrov, sidorov, kolomeev, ivanov, ptushkin, kevlarov]).
X = ptushkin ;
false.

9 ?- append(_, [ivanov, X|_], [petrov, sidorov, kolomeev, ivanov]).
false.

10 ?- append(_, [ivanov, X|_], [ivanov]).
false.
```

есть. Здесь по сути просто смотрим на значение после Иванова.

Задание 2: Отделить последний элемент списка

del_last(L,R):- append(R,[_],L).

А ну-ка:

```
34   delete_last(X,Y) :-
35       append(Y, [_], X).
```

```
2 ?- delete_last([a,b,c,d,r],X).
X = [a, b, c, d] ;
false.
```

Работает!
Алгоритм невероятно четкий - чтобы получить список без последнего элемента, проконкатенируем его же с одним элементом так, чтобы на выходе получился входной список.

Удаление из списка.

```
37    remove(X,[X|T],T).
38    remove(X,[Y|T],[Y|T1]):-remove(X,T,T1).
39
```

Естественно, рекурсивная реализация. Смысл в чем -
мы идем по списку и удаляем элементы, которые не похожи на то, что мы хотим удалить. если наткнулись на такой элемент, то его выбрасываем, а список собираем заново.
Главная проблема - данное выражение удаляет лишь первое вхождение элемента в список, что хорошо видно на тестах.

```
4 ?- remove(c, [a,b,c], X).
X = [a, b] ;
false.

5 ?- remove(c, [a,c,b], X).
X = [a, b]
Unknown action: / (h for help)
Action? .

6 ?- remove(c, [a,c,b,c], X).
X = [a, b, c] ;
X = [a, c, b] ;
false.
```

Хотя и предлагает выбор…. не канон!
Попробуем это обойти:

```
delete(_, [], []). % Если список пустой, то результатом будет пустой список.
delete(X, [X|T], Result) :- delete(X, T, Result). % Если голова списка равна удаляемому эл
delete(X, [H|T], Result) :- X \= H, delete(X, T, Temp), Result = [H|Temp]. % Если голова с
```

все элементы, отличные от удаляемого, мы записываем в начало списка, а элементы, равные удаляемому - пропускаем. Идем таким макаром, пока список не опустеет.
пример:

```
10 ?- delete(c, [b,c,k,c],X).
X = [b, k] ;
false.
```

Работает, хоть все с введи:

```
11 ?- delete(c, [c,c,c,c,c,c,c,c,c], X).
X = [] ;
false.
```

Однако не все идеально и здесь.

```
12 ?- remove(X, [a,b,c], [a,b]).
X = c ;
false.

13 ?- delete(X, [a,b,c], [a,b]).
false.
```

И все-таки вопрос открыт - хвостовая ли рекурсия в remove?

Перестановки списка.
Жаришка начинается, друзья

```
46    permute([],[]).
47    permute(L,[X|T]):-remove(X,L,R),permute(R,T).
48
```

Описание:
терминатор - пустая строка.
ух, я не знаю как это описать
таааак, короче мы заходим в remove и он возвращает нам каждое из значений списка (n штук), он же укорачивает массив на один элемент. на следующем шаге remove возвращает нам снова каждое из значений списка(n - 1 штука), и так далее, пока их не останется 0. Одновременно с этим у нас получается n! решений/перестановок.

ПРИМЕР.

```
18 ?- permute([a,b,c],X).
X = [a, b, c] ;
X = [a, c, b] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [c, a, b] ;
X = [c, b, a] ;
false.

19 ?- permute([a,b,c,h],X).
X = [a, b, c, h] ;
X = [a, b, h, c] ;
X = [a, c, b, h] ;
X = [a, c, h, b] ;
X = [a, h, b, c] ;
X = [a, h, c, b] ;
X = [b, a, c, h] ;
X = [b, a, h, c] ;
X = [b, c, a, h] ;
X = [b, c, h, a] ;
X = [b, h, a, c] ;
X = [b, h, c, a] ;
X = [c, a, b, h] ;
X = [c, a, h, b] ;
X = [c, b, a, h] ;
X = [c, b, h, a] ;
X = [c, h, a, b] ;
X = [c, h, b, a] ;
X = [h, a, b, c] ;
X = [h, a, c, b] ;
X = [h, b, a, c] ;
X = [h, b, c, a] ;
X = [h, c, a, b] ;
X = [h, c, b, a] ;
false.
```

Задача *

Имеются возраста 3 подруг Иры, Маши, Саши: 10, 13, 15 лет. Ира старше Маши, а суммарный возраст Саши и Иры больше удвоенного возраста Маши. Кому сколько лет?

Решение: 2 перестановки подошли.

```
20 ?- permute([I,M,S],[10,13,15]), I>M, S+I>2*M.
I = 13,
M = 10,
S = 15 ;
I = 15,
M = 10,
S = 13 ;
false.
```

Предикаты поиска решений:

В Prolog предикат **`bagof`**/3` используется для создания списка всех возможных решений для заданного шаблона. Он имеет следующий формат:

```prolog
bagof(Template, Goal, Bag)
```

- `Template` - переменная или шаблон, который будет использоваться для формирования элементов списка `Bag`.
- `Goal` - цель или условие, которое должно быть выполнено для каждого элемента списка `Bag`.
- `Bag` - переменная, в которую будут помещены все возможные значения, удовлетворяющие `Goal`.

`bagof/3` создает список `Bag`, содержащий все значения, которые удовлетворяют `Goal`, при этом каждое значение будет соответствовать шаблону `Template`. Если `Goal` не имеет решений, то `bagof/3` вернет `false`.

Пример использования:

```prolog
likes(john, pizza).
likes(john, sushi).
likes(anna, sushi).
likes(mark, pizza).

?- bagof(Food, likes(john, Food), JohnLikes).
JohnLikes = [pizza, sushi].
```

В этом примере `bagof(Food, likes(john, Food), JohnLikes)` вернет `JohnLikes = [pizza, sushi]`, так как `john` нравится пицца и суши, и `Food` будет соответствовать этим значениям.

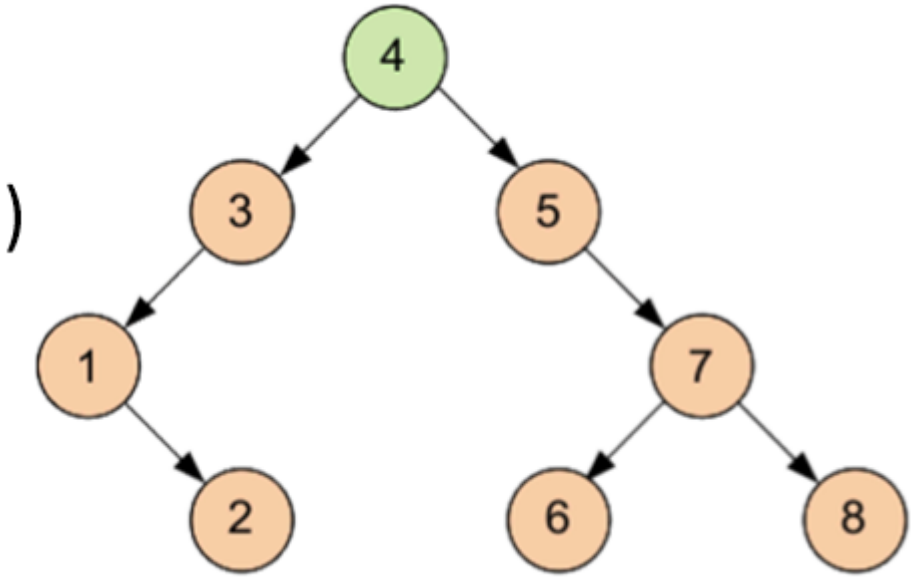
setof = bagof + *сортировка + удаление повторов L*

findall==bagof

Бинарные деревья в ProLog.

bt(LD,X,RD).

Лист: bt(nil,c,nil)



Тогда это дерево можно описать так:

bt(bt(bt(nil, 1, bt(nil,2,nil)),3,nil),4,bt(nil, 5, bt(bt(nil,6,nil),7,bt(nil, 8, nil))));

Включение по возрастанию:

insert (nil,D,bt(nil,D,nil)).

insert (bt(LT,K,RT),X,bt(Ltnew,K,RT)):-X<K,insert(LT,X,Ltnew).

insert (bt(LT,K,RT),X,bt(LT,K,Rtnew)):-X>K,insert(RT,X,Rtnew).

Список в дерево:

```
ltotree([],nil).

ltotree([H|X],T):-ltotree(X,T1),insert(T1,H,T).
```

Дерево в список

```
treetol(nil,[]).

treetol(bt(LT,K,RT),S):- treetol(LT,S1), treetol(RT,S2), append(S1,[K|S2],S).
```

Сортировка:

```
sort(L,L1):-ltotree(L,T),treetol(T,L1).

?-sort([2,5,3,1,7],L),write(L).
```

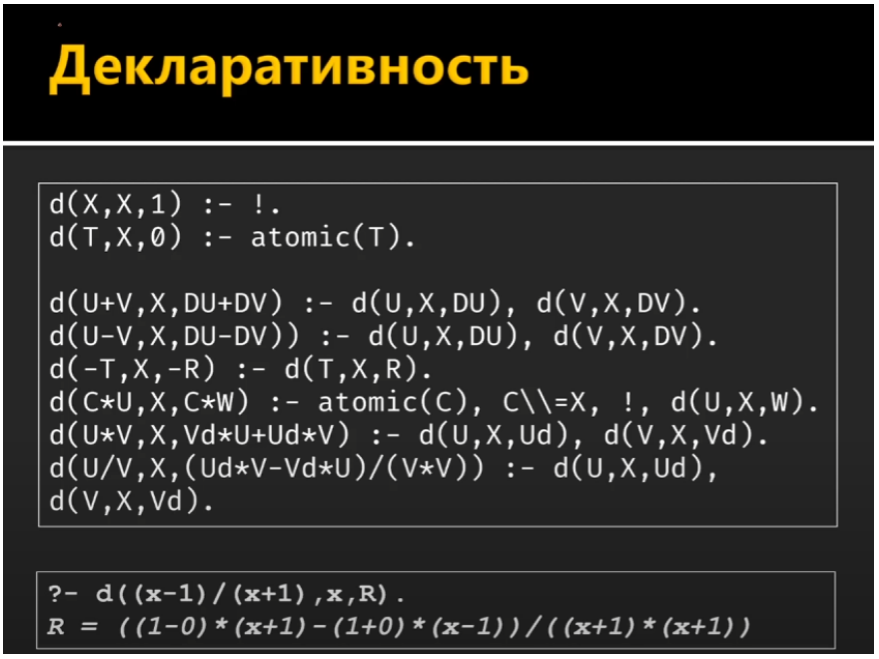
L=[1,2,3,5,7]

Я устал.

Здесь мы просто верим.

(c) ykwais

Диффенцирование, надеюсь не пригодится



Резолюция (или метод резолюции) - это основной метод доказательства теорем в логике высказываний и логике предикатов. Этот метод был впервые введен логиком и философом Готлобом Фреге и впоследствии развит логиком Куртом Геделем и др.

Основная идея резолюции состоит в том, чтобы привести логическое высказывание (или систему высказываний) к логической форме, называемой КНФ (конъюнктивная нормальная форма), и затем применять резолюцию для поиска противоречия. Если удается вывести пустую дизъюнкцию (ложь), то изначальное утверждение считается доказанным.

Процесс резолюции включает в себя следующие шаги:

1. Приведение к КНФ: Преобразование логического выражения к форме, где каждая дизъюнкция (ИЛИ) содержит конъюнкцию (И) литералов.

2. Применение резолюции: Применение правила резолюции, которое заключается в выведении нового высказывания путем упрощения дизъюнкции, содержащей противоречивые литералы.

Повторение шагов 1 и 2: Процесс продолжается до тех пор, пока не будет получена пустая дизъюнкция (ложь) или до тех пор, пока не будет достигнут предопределенный лимит шагов.

Резолюция является основой для многих систем автоматического доказательства теорем и используется в различных областях, включая искусственный интеллект, верификацию программ и математику.

Унификация

УНИФИКАЦИЯ (операция отождествления), при которой происходит конкретизация переменных.

Находит *наиболее общее решение* (НОР) уравнения T1 = T2,

т.е. такую наиболее общую подстановку R, что T1[R] == T2[R].

Пример:

«Собака(X)» унификация с «Собака(Рекс)» => X= Рекс

Правила унификации:

- Константа унифицируется с такой же константой
- Свободная переменная (не унифицируемая) унифицируется 1 раз, при этом *связывается*
- *Связанная* переменная унифицируется так же как ее значение
- Два структурных терма унифицируются если у них **два одинаковых** функтора, **одинаково** число аргументов, унифицируются **все** аргументы

f(a,X) = f(Y,b) (X=b,Y=a)
f(a,X) = f(Y,b) (не унифицируются)
f(a,X,X) = f(Y,Y,b) (не унифицируются)

Равенство и унификация

Предикат	Результат
T1 == T2	Истина, если термы T1 и T2 совпадают
T1 \== T2	Истина, если термы T1 и T2 не совпадают
T1 = T2	Истина, если термы T1 и T2 унифицируемы
T1 \= T2	Истина, если термы T1 и T2 не унифицируемы

Арифметика

В Прологе для вычисления арифметических выражений используется оператор **is**:

A **is** sqrt(2+2)

== тождественное равенство (= унифицируемость)

2+2 =:= 3+1 yes
2+2 == 3+1 no
2+2 = 3+1 no

X =:= Y ошибка
X == X yes
X == Y no
X = Y yes

A is sqrt(2+2)

вычисляет

2+2 =:= 3+1	yes
2+2 == 3+1	no
2+2 = 3+1	no

== тождественное равенст

X =:= Y	ошибка
X == X	yes
X == Y	no
X = Y	yes

Задача *

Имеются возраста 3 подруг Иры, Маши, Саши: 10, 13, 15 лет. Ира старше Маши, а суммарный возраст Саши и Иры больше удвоенного возраста Маши. Кому сколько лет?

```
?- permute([I,M,S],[10,13,15]), I>M, S+I>2*M.
```

ОТВЕТ :

I= 13, M= 10, S= 15

Или I= 15, M= 10, S= 13

- Через !-fail можно определить not!

```
not(P) :- P, !, fail.
not(_).
```

- ! является более общим механизом, чем отрицание
- Во многих случаях можно использовать отсечение вместо отрицания и наоборот
- Что предпочесть?
  - ! не имеет декларативной семантики
  - not имеет декларативную семантику, отличную
    семантики отрицания
- Использование not или ! нарушают чистоту

Предикат *solve*

```
solve(Street,Z,W):-
Street = [[norwegian,_,_,_,_],[_,_,blue,_,_],[_,_,_,_,milk],_,_],
member([english,_,red,_,_],Street),
member([spanish,_,_,dog,_],Street),
member([japanese,painter,_,_,_],Street),
member([italian,_,_,_,tea],Street),
member([_,_,green,_,coffee],Street),
next(Street,[_,_,white,_,_],[_,_,green,_,_]),
member([_,sculptor,_,snails,_],Street),
member([_,diplomat,yellow,_,_],Street),
member([_,musician,_,_,juice],Street),
neighbour(Street,[_,_,_,fox,_],[_,doctor,_,_,_]),
neighbour(Street,[_,_,_,horse,_],[_,diplomat,_,_,_]),
member([Z,_,_,zebra,_],Street),
member([W,_,_,_,water],Street).
```

- Англичанин живет в красном доме.
- У испанца есть собака.
- Японец является художником.
- Итальянец пьет чай.
- Норвежец живет в первом доме слева.
- Владелец зеленого дома пьет кофе.
- Зеленый дом находится справа от белого.
- Скульптор разводит улиток.
- Дипломат живет в желтом доме.
- В доме посередине пьют молоко.
- Норвежец живет рядом с голубым домом.
- Скрипач пьет фруктовые соки.
- Лису держат в доме, соседнем с домом врача.
- Лошадь держат по соседству с домом дипломата.

```
solve(B,I,S) :- permute([B,I,S],[slesar,tokar,svarshik]),not(contradiction([B,I,S])).

contradiction(V):-  data(V,F,A,TF), logicalnot(TF,FT),   data(V,F,A,FT).

logicalnot(true,false).
logicalnot(false,true).

data(_,sister,[slesar],false).

 data(_,old,[tokar,slesar],true).
data(_,old,[slesar,tokar],false).

data(_,old,[svarshik,slesar],true).
data(_,old,[slesar,svarshik],false).

data([B,_,_],sister,[B],true).

data([_,_,S],old,[S,tokar],true).
data([_,_,S],old,[tokar,S],false).
```

Решение

Пары: Лена – X
Y - Z

Список возрастов (от старшего к младшему): Age = [X1,X2,X3,X4]

Предикат «старше»:

```
more(X,Y,[X|T]):-member(Y,T).
more(X,Y,[_|T]):-member(Y,T),member(X,T),more(X,Y,T).
```

```
solve :- Age = [X1,X2,X3,X4], permute(Age,[alla,galia,lena,marina]),
remove(X,[alla,galia,marina],[Y,Z]),
more(galia,marina,Age),
more(lena,Y,Age), more(lena,Z,Age),
member(Pair,[[lena,X],[Y,Z]]), remove(marina,Pair,[MP]),
more(marina,MP,Age),
not(permute([alla,galia],[X3,X4])), not(permute([alla,galia],[X2,X4])),
write('lena-'),write(X),nl,
write(Y),write('-'),write(Z),nl,
write(Age).
```

Простые числа

Генерация простых чисел (способ 1)

```
prime(X):-not(not_prime(X)).
not_prime(X):-X1 is X-1, for(Y,2,X1), 0 is X mod Y.

primes(2,[2]).
primes(N,[N|L]):- prime(N),  ! , N1 is N-1, primes(N1,L).
primes(N,L):-N1 is N-1, primes(N1,L).
```

Предикат *primes*

```
primes(N,L):-integers(2,N,IL), filter(IL,L).
% сгенерировать список
integers(Min,Max,[Min|L]):-Min=<Max, M1 is Min+1,
    integers(M1,Max,L).
filter([],[]).
filter([I|L],[I|R]):-process(I,L,L1),filter(L1,R).
% вычеркивание числа
process(_,[],[]).
process(X,[A|L],[A|R]):-not(0 is A mod X), process(X,L,R).
process(X, [A|L],R):-0 is A mod X,process(X,L,R).
```

Решение

```
solve(B,T,V) :-
    for(B,0,10),      for(T,0,10),      for(V,0,10),
        check(b,[B,T,V]),      check(t,[B,T,V]),      check(v,[B,T,V]).

for(A,A,_).
for(X,A,B) :- A < B, A1 is A+1, for(X,A1,B).
check(N,S):-
    remove(T1,[1,2,3,4],R), remove(T2,R,[L1,L2]),      say(N,T1,S),      say(N,T2,S),
                                                       not(say(N,L1,S)),      not(say(N,L2,S)).

say(b,1,[_,2,_]).
say(b,2,[_,T,V]):- V is T+1.
say(b,3,[B,T,V]):- B+V =:= T+8.
say(b,4,[B,T,V]):- B > T+V.
say(t,1,[_,_,0]).
say(t,2,[_,T,_]):- T =\= 2.
say(t,3,[B,T,_]):- B =\= T.
say(t,4,[B,_,V]):- B+V =:= 13.
say(v,1,[B,T,V]):- T > B, T > V.
say(v,2,[B,_,V]):- B =:= V-3.
say(v,3,[_,_,V]):- V > 0.
say(v,4,[BT,BT,_]).
```

Линк

```
Боря.-   Толя поймал только две рыбки.
         Володя поймал на штуку больше, чем Толя.
         Мы с Володей поймали на восемь рыбок больше,  чем Толя.
         Я наловил рыбы больше, чем Толя и Володя вместе.
Толя.-   Володя не выудил ни одной рыбешки.
         Боря говорит неправду, что я поймал только две штуки.
         У меня и у Бори улов  неодинаковый.
         Володя  и  Боря вместе поймали тринадцать рыбок.
Володя.- Толя - самый удачливый  из всех нас.
         Я поймал на три рыбки  больше, чем Боря.
         Толя говорит неправду, что я ничего не поймал.
         Боря и Толя наловили  поровну.
```

Объявление оператора

В Prolog пользовательские операторы могут быть определены с помощью `op/3` :

`op(+Precedence, +Type, :Operator)`

- Объявляет оператора оператором типа с приоритетом. Оператор также может быть списком имен, и в этом случае все элементы списка объявляются одинаковыми операторами.
- Приоритет - целое число от 0 до 1200, где 0 удаляет объявление.
- Тип является одним из: `xf` , `yf` , `xfx` , `xfy` , `yfx` , `fy` или `fx` где `f` указывает положение функтора, а `x` и `y` указывают позиции аргументов. `y` обозначает термин с приоритетом, меньшим или равным приоритету функтора, тогда как `x` обозначает строго более низкий приоритет.
  - Префикс: `fx` , `fy`
  - `xfx` : `xfx` (не ассоциативный), `xfy` (правый ассоциативный), `yfx` (левый ассоциативный)
  - Postfix: `xf` , `yf`

Пример использования:

```
:- op(900, xf, is_true).

X_0 is_true :-
  X_0.
```

Пример запроса:

```
?- dif(X, a) is_true.
dif(X, a).
```