записки по

Л о г и ч е с к о п р о г р а м и р а н е

October 1, 2015

За тези записки

Материала от упражненията по Логическо програмиране.

Задачите са авторство на Тинко Тинчев, Антон Зиновиев, Борислав Ризов, Николай Белухов, Стефан Герджиков

Автор: Владислав Ненчев

1 Въведение

Какво е това "Логическо програмиране"?
Декларативен стил на програмиране, разлики с императивния стил.
При декларативното програмиране не се задават изричните стъпки/инструкции по които се реализира целта. Задават се факти и правила, чрез които от вече известни факти получаваме нови. Целта се описва, а engina-а на езика/платформата, която използваме. я реализира чрез въведените факти и правила.

Prolog е един от най-разпространените езици за логическо програмиране.

Приложения в изкуствения интелект и автоматичното доказване на теореми.
Други ползи от знанията за декларативно програмиране: умения за проверки на коректност на програми, еквивалентност на програми, работа със спецификации и др.

1.1 Език на предикатното смятане от първи ред

Формулите са начин за точно и структурирано записване на твърдения. Те се остойностяват с истина - И - или с лъжа - Л.

Най-естествени формули са тези, които боравят само с булевите стойности И и Л и с булеви функции, приложени върху тях. Тези формули се наричат съждителни. Най-простите съждителни формули - атомарни формули - се състоят само от една съждителна променлива. Например p или q. По-сложни формули съставяме чрез булевите функции $\neg$, $\&$, $\vee$, $\Rightarrow$ и $\Leftrightarrow$. Например $p \Rightarrow q \vee r$, $\neg(p \,\&\, q) \Leftrightarrow \neg p \vee \neg q$. Стойностни таблици за $\Rightarrow$ и $\Leftrightarrow$.

Формули с променливи. Термове. Изпълнимост и вярност на формули.

Квантори. Превод на естествени изречения като формули: "Всички студенти обичат бира." , "Всички студенти, които посещават лекции, обичат бира.", същото изречение, когато работим с множество от хора, а не от студенти, "Пловдивчаните и бургазлиите са южняци." и др.

Разлика между $\forall x \exists y$ и $\exists x \forall y$. Примери с наредби. Примери с графи.

Елементарен извод. $p(x) \Rightarrow q(x)$, $p(a)$. Какво следствие имаме?

1.2 Еквивалентни преобразувания на формули

При писане на програми на Prolog и работа с формули се налага да преработваме външния вид на формулите/условията, като запазваме вярността и изпълнимостта им (т.е. да правим еквивалентни преобразувания). Например $\forall x(p(x) \vee b(x) \Rightarrow s(x))$ и $\forall x(p(x) \Rightarrow s(x)) \;\&\; \forall x(b(x) \Rightarrow s(x))$ са два еквивалентни записа на "Пловдивчаните и бургазлиите са южняци.".

Ето основните еквивалентни преобразувания, които ще използваме:

- закон за двойното отрицание: $\phi \equiv \neg\neg\phi$;
- закони на де Морган: $\neg(\phi \,\&\, \psi) \equiv \neg\phi \vee \neg\psi$ и $\neg(\phi \vee \psi) \equiv \neg\phi \,\&\, \neg\psi$;
- дистрибутивни закони за $\&$ и $\vee$: $\phi \,\&\, (\psi \vee \chi) \equiv (\phi \,\&\, \psi) \vee (\phi \,\&\, \chi)$ и $\phi \vee (\psi \,\&\, \chi) \equiv (\phi \vee \psi) \,\&\, (\phi \vee \chi)$;
- преобразуване на $\Rightarrow$ и $\Leftrightarrow$: $\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$, $\phi \Leftrightarrow \psi \equiv (\phi \Rightarrow \psi) \,\&\, (\psi \Rightarrow \phi)$ и $\phi \Leftrightarrow \psi \equiv (\phi \,\&\, \psi) \vee (\neg\phi \,\&\, \neg\psi)$;

Две основни преобразувания, полезни при съставяне на Prolog клаузи:
$\neg(\phi \Rightarrow \psi) \equiv \neg(\neg\phi \vee \psi) \equiv \neg\neg\phi \,\&\, \neg\psi \equiv \phi \,\&\, \neg\psi$;
$\phi \Rightarrow (\psi \Rightarrow \chi) \equiv \phi \Rightarrow (\neg\psi \vee \chi) \equiv \neg\phi \vee (\neg\psi \vee \chi) \equiv \neg\phi \vee \neg\psi \vee \chi \equiv \neg(\phi \,\&\, \psi) \vee \chi \equiv \phi \,\&\, \psi \Rightarrow \chi$.

- комбинации на квантори и отрицание: $\neg\forall x\phi(x) \equiv \exists x\neg\phi(x)$ и $\neg\exists x\phi(x) \equiv \forall x\neg\phi(x)$;

- комбинации на квантори и & и $\vee$: $\forall x(\phi(x)\ \&\ \psi(x)) \equiv \forall x\phi(x)\ \&\ \forall x\psi(x)$ и $\exists x(\phi(x) \vee \psi(x)) \equiv \exists x\phi(x) \vee \exists x\psi(x)$.
  Примери, че $\forall x(\phi(x) \vee \psi(x)) \equiv \forall x\phi(x) \vee \forall x\psi(x)$ и $\exists x(\phi(x)\ \&\ \psi(x)) \equiv \exists x\phi(x)\ \&\ \exists x\psi(x)$ не са закони: $\forall x(x < 0 \vee 0 \leq x)$, $\exists x(x < 0)\ \&\ \exists x(0 \leq x)$.

Ето поредица от преобразувания от $\forall x(p(x) \vee b(x) \Rightarrow s(x))$ до $\forall x(p(x) \Rightarrow s(x))\ \&\ \forall x(b(x) \Rightarrow s(x))$, които показват еквивалентността им:

$$
\begin{aligned}
&\forall x(p(x) \vee b(x) \Rightarrow s(x)) && \equiv\\
&\forall x(\neg(p(x) \vee b(x)) \vee s(x)) && \equiv\\
&\forall x((\neg p(x)\ \&\ \neg b(x)) \vee s(x)) && \equiv\\
&\forall x((\neg p(x) \vee s(x))\ \&\ (\neg b(x) \vee s(x))) && \equiv\\
&\forall x((p(x) \Rightarrow s(x))\ \&\ (b(x) \Rightarrow s(x))) && \equiv\\
&\forall x(p(x) \Rightarrow s(x))\ \&\ \forall x(b(x) \Rightarrow s(x))
\end{aligned}
$$

1.3 Програмен език Пролог. Синтаксис.

Идентификатори и константи - зпочват с малка буква.
Променливи - започват с главна буква. Анонимни променливи.

Атоми. Удовлетворяване или пропадане.
Ариметични предикати и операции: $<$, $>$, $=<$, $>=$, $+$, $-$, $/$, $*$, $**$, **mod**, $=:=$.
Предикат за унифицируемост: $=$.
Примери: $\mathrm{like}(\mathrm{john},\ \mathrm{mary})$, $\mathrm{X} = \mathrm{Y}$, $\mathrm{N} < 100$.
Предикат за присвояване на стойност: **is**.
Разлика между X **is** $\mathrm{Y} + 1$ и $\mathrm{X} = \mathrm{Y} + 1$. Както в c между $\mathrm{x} = \mathrm{y} + \mathrm{z};$ и $\mathrm{x} == \mathrm{y} + \mathrm{z};$.
Отрицание **not** или $\backslash$. Примери: **not**$(\mathrm{like}(\mathrm{john},\ \mathrm{mary}))$, $\mathrm{X} \backslash = \mathrm{Y}$.
Конюнкция - ,. $\mathrm{like}(\mathrm{john},\ \mathrm{mary}),\ \mathrm{like}(\mathrm{mary},\ \mathrm{john})$.
Дизюнкция - ;. $\mathrm{like}(\mathrm{john},\ \mathrm{mary});$ **not**$(\mathrm{like}(\mathrm{john},\ \mathrm{mary}))$.
Импликация - $:-$, правила. Примери: $\mathrm{like}(\mathrm{john},\ \mathrm{X}) :- \mathrm{woman}(\mathrm{X}).$, $\mathrm{X} =< \mathrm{Y} :- \mathrm{X} = \mathrm{Y};\ \mathrm{X} < \mathrm{Y}$.
Съответствие между формули и Prolog клаузи. Квантори - подразбират се. Универсални във фактите и следствията на правилата, екзистенциални в условията на правилата и след отрицание.

Конструкции за работа със списъци: [], [|].

```
like(mary, wine).
like(john, mary).
like(james, mary).
```

Цели:

```
?- like(mary, wine).
?- like(mary, john).
?- like(X, mary).
```

Интуитивни обяснения - с формален извод

Вариации на програмата с по-сложни условия. Реализиране на различните логически операции - $\neg$, &, $\vee$ (отново може чрез задачата за харесване). Рекурсия.

```
woman(mary).
like(john, X) :- woman(X).
like(john, X) :- woman(X), like(X, wine).
like(john, X) :- woman(X), (like(X, wine); like(X, chocolate)).
```

Задачата за родословно дърво. Задават се факти: woman(xxxx), man(xxxx), parent(xxxx, yyyy). Да се напишат правила за: майка, баща, сестра, брат, братовчет, чичо ...
Пример:

```
mother(X, Y) :- parent(X, Y), woman(X).
brother(X, Y) :- man(X), not(X = Y), parent(Z, X), parent(Z, Y).
```

Последен пример за предикат за роднина. Примери, с предиката за роднина, за зацикляне и неизвеждане на всички решения (т.е. цикъл в графа на родните).

```
edge(a, b).
edge(b, a).
edge(b, c).
edge(c, d).
```

С дърво на изпълнение ... Загатване за нужда от списъци ...

1.4 Работа с термове

Да се напише предикат за определяне на производна d(F, G): $G = dF/dx$.

```
d(0, 0).
d(1, 0).
d(2, 0).
d(x, 1).
d(z, 0).
d(F1+F2, G1+G2) :- d(F1, G1), d(F2, G2).
d(F1-F2, G1-G2) :- d(F1, G1), d(F2, G2).
d(F1*F2, G1*F2+G2*F1) :- d(F1, G1), d(F2, G2).
d(F1/F2, (G1*F1-G2*F1)/(F2*F2)) :- d(F1, G1), d(F2, G2).
d(sin(F), cos(F)*G) :- d(F, G).
```

1.5 Начин на работа на Пролог. Дървета на изпълнение.

За по-сложни програми ни трябва по добър метод за проследяване на изпълнението. Дървета на изпълнение. Така работи Strawberry Prolog. Накрая ще бъде представен и метода на резолюцията. Повечето интерпретатори на Пролог работят с този метод. Например SWI Prolog, GNU Prolog - SLD резолюция??

Интерпретатори SWI Prolog (отговаря на стандартите), Strawberry Prolog (debugger, дърво на извода), GNU Prolog (най-бърз).

Как се удовлетворяват целите. Извод.
Унифициране.
Преудовлетворяване.

Изводът (или проверката дали целта е удовлетворима при дадена програма) се извършва, като към клаузите от програмата се прибави отрицанието на целта и чрез метода на резолюцията се опитаме да изведем празното множество. Ако успеем - значи целта е валидна при дадената програма. В противен случай - не е. Така калузите действат като аксиоми и проверяваме дали целта е изводима при тези аксиоми. Гореописания метод всъщност доказва чрез допускане на противното. Извършването на резолюцията може опростено да се представи като дърво на извод.

Построяване на дърво на извода по дадена цел за дадена програма:
Всеки възел в дървото представлява цел към даден момент.
Корен на дървото е целта, зададена на програмата.

Наследниците на даден възел се определят от първото условие (атом) от целта, записана във възела, така:

- ако условието е вграден предикат за действие (като **write** или **nl**), действието се изпълнява, условието се премахва от целта и така се формира новата цел за дъщерния възел;
- ако условието е аритметична операция, тя се извършва, като се променя стойността на променливата в останалите условия от целта и се процедира, както в горния случай;
- когато условието е аритметична проверка или проверка за унифицируемост (= или \=) ако то е вярно се процедира, както в предните два случая, а в противен случай изпълнението на програмата по този път спира;
- ако условието е предикат, за който имаме клаузи в програмата, се създава по един наследник за всяка клауза, за която условието се унифицира успешно с главата на клаузата. Целта в дъщерния възел се формита като махнем условието, приложим унификацията върху останалите условия в целта, а на мястото на условието сложим предусловията от клаузата.

Реда на наследниците от ляво на дясно се определя от реда на клаузите в програмата. Изпълнението на програмата се извършва като дървото се обхожда в дълбочина от ляво на дясно. Първия път, по който се стига до успешен завършек (празен възел), дава първото решение на програмата. При преудовлетворяване на целта се минава по следващия алтернативен успешен път в дървото.

Има и алтернативни видове обхождания: в широчина например. Тогава реда на клаузите може да няма значение.

Примерна програма:

```
p(a).                    % clause #1
p(X) :- q(X), r(X).      % clause #2
p(X) :- u(X).            % clause #3
q(X) :- s(X).            % clause #4
r(a).                    % clause #5
r(b).                    % clause #6
s(a).                    % clause #7
s(b).                    % clause #8
s(c).                    % clause #9
u(d).                    % clause #10
```

Цел:

```
?- p(X).
```

Дърво на извод за дадените програма и цел:
Резултат: X = a, X = a, X = b, X = d, no.

Втори пример (с безкрайно дърво):

```
evenNumber(0).
evenNumber(X) :- evenNumber(Y), X is Y + 2.
```

Резултата са всички четни числа.

Дърво на изпълнение на програмата при някой от дадените примери за задачата за харесване ...

1.6 Логическо програмиране

Например в задачата за харесване клаузата

```
like(john, X) :- woman(X), (like(X, wine); like(X, chocolate)).
```

е еквивалентна на двете клаузи

```
like(john, X) :- woman(X), like(X, wine).
like(john, X) :- woman(X), like(X, chocolate).
```

На първата клауза съответства формулата

$$\forall x(woman(x)\ \&\ (like(x, wine) \vee like(x, chocolate)) \Rightarrow like(john, x))$$

Ще покажем че тя е еквивалентна на конюнкцията от формулите, съответстващи на вторите две клаузи

$$\begin{aligned}&\forall x(woman(x)\ \&\ like(x, wine) \Rightarrow like(john, x))\ \&\\ &\quad \forall x(woman(x)\ \&\ like(x, chocolate) \Rightarrow like(john, x))\end{aligned}$$

Ето преобразуванията:

$$\begin{aligned}&\forall x(woman(x)\ \&\ (like(x, wine) \vee like(x, chocolate)) \Rightarrow like(john, x)) && \equiv\\ &\forall x((woman(x)\ \&\ like(x, wine)) \vee (woman(x)\ \&\ like(x, chocolate)) \Rightarrow like(john, x)) && \equiv\\ &\forall x(woman(x)\ \&\ like(x, wine) \Rightarrow like(john, x))\ \&\\ &\quad \forall x(woman(x)\ \&\ like(x, chocolate) \Rightarrow like(john, x))\end{aligned}$$

Последните преобразувания са същите, както при еквивалентността между $\forall x(p(x) \vee b(x) \Rightarrow s(x))$ и $\forall x(p(x) \Rightarrow s(x))\ \&\ \forall x(b(x) \Rightarrow s(x))$.

2 Работа със списъци

2.1 Основни методи и предикати

Как се дифинира списък - изброяване на елементите и конструкция с глава и опашка. Празен списък. Вложени списъци.
Примери за конструиране на списъци:
[1, 2, 3], [head | Queue], [X, [inner, list , []], Y], [1 | [2 | [3]]] .
Примери за неправилен запис:
[X | Y | Queue], [head | X, Y].

Упражнения:
Колко елемента имат списъците:
[1, 2, 3], [1, 2, 2], [] , [[]] , [[[]], [[], [[]] .
Какъв е резултатът от следните унификации:
[A, 2| X] = [1, 2, 3],
[_ | [[X | _], _ | [Y | _]]] = [[1, 2], [3, 4], [5, 6], [7, 8]] .
Примери за неуспешни унификации:
[1, 2, 3] = [_, 3 | Q], [_ | [[1] | _]] = [1, 2, 3].

Има два основни метода за реализиране на програми за списъци:

- последователно обхождане на елементите на списъка и проверяване на условието за всеки елемент;
- заване на условието, на което трябва да отговаря списъка, чрез използване на предикат, чието изпълнение извършва обхождането.

Основни предикати за работа със списъци - append и member.

```
append([], Y, Y).
append([A | X], Y, [A | Z]) :- append(X, Y, Z).
```

append е фундаментален предикат, който позволява да представяме списък, като конкатенация от два или повече подсписъка. Например по следния начин можем да разделим списък L на три списъка X, Y и Z

```
append(X, M, L), append(Y, Z, M)
```

Обяснение защо, като разменим двата append-а, програмата може да зацикли. Обяснение кои аргументи на append трябва да имат стойност, за да може да има успешно удовлетворяване ...
Реализации на member. Първи начин - чрез последователен достъп до елементите:

```
member(A, [A|_]).
member(A, [_|X]) :- member(A,X).
```

Втори начин - декларативно, чрез append:

```
member(A, L) :- append(_, [A | _], L).
```

Предикатите append и member могат да бъдат използвани и като разпознаватели и като генератори.

Предикат за проверка, дали даден списък от числа е сортиран - sorted.
Първи начин - последователен достъп:

```
sorted([]).
sorted([_]).
sorted([X, Y | Q]) :- X =< Y, sorted([Y | Q]).
```

Втори начин - декларативно:
За целта ще изразим чрез Prolog клауза, условието един списък да е сортиран - "за всеки два sпоследователни елемента на списъка трябва първият да е по-малък или равен на втория". Това е формулата $\forall l(\forall x \forall y(consecutiveMembers(x,y,l) \Rightarrow x \leq y) \Rightarrow sorted(l))$. Сега ще преработим формулата, така че да можем да я запишем като клауза:

$$\begin{aligned}
&\forall l(\forall x \forall y(consecutiveMembers(x,y,l) \Rightarrow x \leq y) \Rightarrow sorted(l)) && \equiv \\
&\forall l(\forall x \forall y(\neg consecutiveMembers(x,y,l) \vee x \leq y) \Rightarrow sorted(l)) && \equiv \\
&\forall l(\forall x \forall y \neg(consecutiveMembers(x,y,l) \;\&\; \neg x \leq y) \Rightarrow sorted(l)) && \equiv \\
&\forall l(\neg \exists x \exists y(consecutiveMembers(x,y,l) \;\&\; x > y) \Rightarrow sorted(l))
\end{aligned}$$

Така Prolog клаузата изглежда по следния начин:

```
sorted(X) :- not(append(_, [A,B|_], X), A>B).
```

По подобен начин може да се реши следната задача:
Да се реализира предикат p, който разпознава списъци от числа, такива че за всеки елемент x на списъка, има елемент y, такъв че $x + y < 0$.
Ето формулата, която изразява това условие и преобразуванията 'и до вид, в който можем да я запишем като клауза:

$$\begin{aligned}
&\forall l(\forall x(member(x,l) \Rightarrow \exists y(member(y,l) \;\&\; x + y < 0)) \Rightarrow p(l)) && \equiv \\
&\forall l(\forall x(\neg member(x,l) \vee \exists y(member(y,l) \;\&\; x + y < 0)) \Rightarrow p(l)) && \equiv \\
&\forall l(\forall x \neg(member(x,l) \;\&\; \neg \exists y(member(y,l) \;\&\; x + y < 0)) \Rightarrow p(l)) && \equiv \\
&\forall l(\neg \exists x(member(x,l) \;\&\; \neg \exists y(member(y,l) \;\&\; x + y < 0)) \Rightarrow p(l))
\end{aligned}$$

Така Proglog клаузата излгежда по този начин:

```
p(L) :- not(member(X, L), not((member(Y, L), X + Y < 0))).
```

Предикат за дължина на списък - count:

```
count([], 0).
count([H | Q], C) :- count(Q, CR), C is CR + 1.
```

По същия начин се реализира предикат за сума на елементите на списък:

```
sum([], 0).
sum([H | Q], S) :- sum(Q, SR), S is SR + H.
```

Предикат за списък с четен брой елемента:

```
evenCount(L) :- count(L, N), N mod 2 =:= 0.
```

Предикат за последен елемент:

```
last(X, [X]).
last(X, [_ | Q]) :- last(X, Q).
```

чрез append:

```
last(X, L) :- append(_, [X], L).
```

Предикат за нечетен елемент на списък:

```
odd(X, [X | _]).
odd(X, [_, _ | Q]) :- odd(X, Q)
```

Предикат за четен елемент на списък:

```
even(X, [_, X | _]).
even(X, [_, _ | Q]) :- even(X, Q).
```

свеждане на четността към нечетност:

```
even(X, [_ | Q]) :- odd(X, Q).
```

Да се реализират предикатите за нечетен елемент и четен елемент чрез предиката за четен брой елемента и append.

2.2 Предикати за подсписък, подмножество и пермутации

Предикат че един списък е начало на друг:

```
begin([], _).
begin([H | Q], [H | L]) :- begin(Q, L)
```

Предикат че един списък е подсписък на друг:

```
sublist(L, M) :- begin(L, M).
sublist(L, [_ | M]) :- sublist(L, M).
```

Да се даде пример с изхода при генериране за M = [1, 2, 3, 4] - първо 5 списъка, после още 4, още 3 ... Добре илюстрира генерирането.
И чрез append:

```
sublist(L, M) :- append(P, Q, M), append(L, R, Q).
```

Да се припомни опастността при размяна на append-ите.

Предикат кога един списък е подмножество на друг:

```
subset([], _).
subset([X | Q], L) :- member(X, L), subset(Q, L).
```

Става само за разпознаване. Защо не става за генериране - примерен изход при L = [1, 2, 3]: [], [1], [1, 1] ...

Предикат за това дали един списък е множество - може и за домашно:

```
set(L) :- not((append(_, [H | Q], L), member(H, Q))).
```

Защо генерирането на подмножества не става с предиката за множество.

Реализация на генератор на подмножества, чрез принципа че всеки елемент или влиза в подмножеството или не:

```
subset([], []).
subset([H | S], [H | Q]) :- subset(S, Q).
subset(S, [H | Q]) :- subset(S, Q).
```

Предикат за генериране на подмножества:

```
rest(_, [], []).
rest(X, [X | Q], L) :- rest(X, Q, L).
rest(X, [Y | Q], [Y | L]) :- X \= Y, rest(X, Q, L).
subset([], _).
subset([X | Q], L) :- member(X, L), rest(X, L, M), subset(Q, M).
```

I начин: взимаме произволен елемент от списъка. слагаме го за начало на резултата, а за опашка на резултата слагаме пермутацията на остатъка от списъка.

```
out(X, L, M) :- append(N, [X | Q], L), append(N, Q, M).
permutate([], []).
permutate(L, [X | Q]) :- out(X, L, M), permutate(M, Q).
```

II начин: взимаме главата на списъка и я вмъкваме на произволно място в пермутацията на остатъка от списъка.

```
in(X, L, M) :- out(X, M, L).
permutate([], []).
permutate([H | Q], M) :- permutate(Q, L), in(H, L, M).
```

Използване на предиката за пермутации за генериране на подмножество:

```
subset(M, L) :- permutate(L, K), sublist(M, K).
```

2.3 Предикати за сортиране

Предикат за сортиране на списък чрез пермутации и разпознаване на сортиран списък (наивен начин):

```
sort(L, S) :- permutate(L, S), sorted(S).
```

По-ефективни начини:

I начин: Взимаме първия елемент, сортираме остатъка и вмъкваме елемента на правилното място във вече сортирания остатък (insertion sort).

```
sortIn(X, [], [X]).
sortIn(X, [H | Q], [X, H | Q]) :- X =< H.
sortIn(X, [H | Q], [H | L]) :- H < X, sortIn(X, Q, L).
sort([], []).
sort([H | Q], L) :- sort(Q, M), sortIn(H, M, L).
```

II начин: Измъкваме минималния елемент от списъка, сортираме остатъка и го слагаме след елемента (selection sort).

```
outMin(X, [X], []).
outMin(H, [H | Q], Q) :- outMin(X, Q, _), H =< X.
outMin(X, [H | Q], [H | L]) :- outMin(X, Q, L), H > X.
sort([], []).
sort(L, [H | Q]) :- outMin(H, L, M), sort(M, Q).
```

Тук нямаме свеждане между операциите за вмъкване и измъкване, както при пермутациите. Причината е че нямаме огледалност на операциите при вмъкването и измъкването. В единия случай вмъкваме произволен елемент в сортиран списък, а в другия случай измъкваме минимален елемент от произволен списък.

“Бързо” сортиране на списък:

```
quicksort([], []).
quicksort([H | Q], S) :-
    split(H, Q, K, L), quicksort(K, SK), quicksort(L, SL),
    append(SK, [H], R), append(R, SL, S).
split(_, [], [], []).
split(X, [H | Q], [H | K], L) :- H < X, split(X, Q, K, L).
split(X, [H | Q], K, [H | L]) :- H >= X, split(X, Q, K, L).
```

2.4 Графи

Представяне със списък от ребра [[v0, u0], [v1, u1], ... [vn, un]].

Път в граф. Референция към задачата за роднини от по-рано ...
Реализираме path(G, X, Y) - има път в графа G от връх X до връх Y.
"Наивно"решение:

```
path(G, X, Y) :- member([X, Y], G).
path(G, X, Y) :- member([X, Z], G), path(G, Z, Y).
```

Проблем със зацикляне на отговорите, например, при [[a, b], [b, c], [c, a], [c, d], [d, e]].

Правилно решение:

```
pathVisited(_, X, X, _, [X]).
pathVisited(G, X, Y, V, [X | P]) :-
    member([X, Z], G), not(member(Z, V)), pathVisited(G, Z, Y, [Z | V], P).
path(G, X, Y) :- pathVisited(G, X, Y, [X], _).
```

Използваме pathVisited(G, X, Y, V, P) - P е път без повторения в графа G от връх X до връх Y, като V е списъка от преминатите до сега върхове.

Предикат за върховете на графа vertices(G, V) - V е списък от върховете на графа G:

```
vertices([], []).
vertices([[X, Y] | T], V) :-
    vertices(T, TV), addVertex(X, TV, TV2), addVertex(Y, TV2, V).
addVertex(V, VL, VR) :- not(member(V, VL)), append([V], VL, VR).
addVertex(V, VL, VL) :- member(V, VL).
```

Декларативно решение за предиката за път:

```
pathHelp(G, X, Y, P) :-
    vertices(G, V), subset(P, V), P = [X | _], last(Y, P),
    not(append(_, [A, B | _], P), not(member([A, B], G))).
path(G, X, Y) :- pathHelp(G, X, Y, _).
```

Използваме pathHelp(G, X, Y, P) - P е път без повторения в графа G от връх X до връх Y.

Предикат за обхождане в широчина: bfs(G, X, T) - T е обхождане на графа G, като започваме от връх X.

```
bfs(G, X, T) :-
    vertices(G, V), permutate(V, T),
    not
    (
        append(_, [C, D | _], T), pathHelp(G, X, C, PC), count(PC, CC),
        pathHelp(G, X, D, PD), count(PD, CD), CC > CD
    ).
```

Горното решение предполага, че всеки връх на графа е достижим от върха X. Ето решение, което обхожда само достижимите върхове.

```
bfs(G, X, T) :-
    vertices(G, V), subset(T, V),
    not(member(A, V), path(G, X, A), not(member(A, S))),
    not(member(B, T), not(path(G, X, B))),
    not
```

```
(
    append(_, [C, D | _], T), pathHelp(G, X, C, PC), count(PC, CC),
    pathHelp(G, X, D, PD), count(PD, CD), CC > CD
).
```

Обхождане в дълбочина се прави по подобен начин, като се използва условието, например, че пътищата от X до всеки един от върховете от T са подредени лексикографски, по отношение на наредбата, зададена от списъка T.

3 Работа с числа и аритметични операции

Припомняне на основните операции и предикати за числа. които използваме: +, −, *, /, //, **, **mod**, **is**, <, >, =<, >=, =:= и =\=. Инфиксен запис вместо стандартния префиксен.

За числови операции е по-добре да се използва SWI Prolog вместо Strawberry Prolog. Това е т.к. Strawberry Prolog има нестандартно поведение при аритметични изчисления - опитва се да пресмята аритметичните изрази, независимо дали това е нужно или не (например, пресмята ги и при употребата на =).

Операция за присвояване **is**: `X is Y + 1`. Изчислява се изразът отдясно и стойността му се свързва (присвоява) на свободната променлива отляво.

Например за генериране на естествени числа с определени свойства може да се използва помощен генератор на всички числа:

```
int(0).
int(X) :- int(Y), X is Y + 1.
```

Не става за разпознавател.

Разлика при употребата на **is** и предиката за аритметично равенство =:= и предиката за унифициране =.
Разлика между `2 + 2 = 4` и `2 + 2 =:= 4`. Например в предикат за НОД:

```
gcd(X, Y, Y) :- X mod Y =:= 0.
gcd(X, Y, Z) :- R is X mod Y, gcd(Y, R, Z).
```

Разлика между `X = Y + 2` и `X is Y + 2`:

```
evenNumber(0).
evenNumber(X) :- evenNumber(Y), X = Y + 2.
```

Изход при цел evenNumber(X): 0, 0 + 2, 0 + 2 + 2, 0 + 2 + 2 + 2, ...

```
evenNumber(0).
evenNumber(X) :- evenNumber(Y), X is Y + 2.
```

Изход на програмата при цел evenNumber(X): 0, 2, 4, 6, ...
Не става за разпознавател.

Да се съобрази защо генераторите, които до сега сме написали, не могат да работят като разпознаватели. Възможно решение за разпознавател:

```
evenNumber(0).
evenNumber(X) :- X > 1, Y is X - 2, evenNumber(Y).
```

(работи "наопаки" на генератора, но с условие за ограничаване)
Важно: Тези решения може да работят само като разпознаватели, но не и като генератори.

Ето вариант на evenNumber, който работи и като генератор и като разпознавател (стига да му се подават само естествени числа)

```
evenNumber(X) :- int(X), X mod 2 =:= 0.
```

is, =:= и =\= да се използват само за числа. Никога за списъци или за символни стойности!!

Предикат за n-то чиско от редицата на Фибоначи:

```
fibonacci(0, 0).
fibonacci(1, 1).
fibonacci(N, X) :-
```

```
    N > 1, N1 is N - 1, N2 is N - 2,
    fibonacci(N1, X1), fibonacci(N2, X2), X is X1 + X2.
```

Генератор за Фибоначи:

```
fibonacciGenerator(X) :- int(N), fibonacci(N, X).
```

По-ефективна реализация на редицата на Фибоначи:
Досегашния предикат работи за експоненциално време. Да се прецени по вида на дървото за извод как може да се постигне линейно време.
Реализация:

```
fibonacci(0, 1).
fibonacci(F1, F2) :- fibonacci(F, F1), F2 is F1 + F.
```

Втори вариант за генератор за Фибоначи (с по-ефективния предикат):

```
fibonacciGenerator(X) :- fibonacci(X, _).
```

Задача за упражнение в къщи (за домашно): Разпознавател за редицата на Фибоначи.

Модел за писане на генератори:
Шаблони за реализиране на генератори: "помощен генератор, разпознавател" и "помощен генератор, преобразувател". Използва се помощен генератор, който генерира повече стойности (по-голямо множество, което се описва по-лесно) и след това се използва друг предикат, който да "филтрира" тези стойности.

Много полезен предикат при работа с числа - генериране на числата между две дадени граници. Да се запомни, също както предикатите append и member при работа със списъци.
Да се обсъди защо първосигналната идея

```
between(A, B, A).
between(A, B, X) :- between(A, B, Y), X is Y + 1.
```

не работи. Може да се разгледа дървото на извод за този вариант. Трябва да се добавят условия, които да ограничат разрастването на дървото.
Правилна реализация:

```
between(A, B, A) :- A =< B.
between(A, B, X) :- A < B, A1 is A + 1, between(A1, B, X).
```

Да се напишат предикати, разпознаващи и генериращи прости числа.

```
isPrime(N) :- N1 is N // 2, not(between(2, N1, X), N mod X =:= 0).
prime(N) :- int(N), isPrime(N).
```

Да се реализира предикат, който по подадено число, създава списък с разлагането му на прости делители, във възходящ ред

```
primeDivisor(N, P) :-
    N1 is N // 2, between(2, N1, P), N mod P =:= 0, isPrime(P).
minPrimeDivisor(N, P) :- primeDivisor(N, P), not(primeDivisor(N, Q), Q < P).
decomposition(0, []).
decomposition(1, []).
decomposition(N, [P | T]) :-
    minPrimeDivisor(N, P), N1 is N / P, decomposition(N1, T).
```

Да се реализира предикат, който по подадено число, създава списък с простите делители (във възходящ ред) и техните степени от каноничното му представяне

```
degrees(0, []).
degrees(1, []).
degrees(N, D) :-
    minPrimeDivisor(N, P), N1 is N / P, decomposition(N1, T), add(T, P, D).
add(T, P, D) :-
    append(B, [[P, CR] | E], T), C is CR + 1, append(B, [[P, C] | E], D).
add(T, P, [[P, 1] | T]) :- not(member([P, _], T)).
```

Да се генерират всички числа между 0 и 1000, които са от следния вид/формат: $3 * (n+2)^5 * (k+3)^2$.
Реализация:

```
format(X) :-
    between(0, 1000, Y), between(0, 1000, Z),
    X is 3 * (Y + 2) ** 5 * (Z + 3) ** 2, X >= 0, X =< 1000.
```

Втори начин:

```
format(X) :-
    between(0, 1000, S), between(0, S, Z),
    Y is S - Z, X is 3 * (Y + 2) ** 5 * (Z + 3) ** 2, X >= 0, X =< 1000.
```

Важно: тук between(0, 1000, S), between(0, S, Z), Y is S − Z не генерира всички двойки числа между 0 и 1000, ами всички двойки числа със сума между 0 и 1000.

Да се генерират всички точки с цели координати, които се намират във вътрешността на елипса с полуоси 113 и 71.

```
elipse(X, Y) :-
    between(-113, 113, X), between(-71, 71, Y),
    (X * X) / (113 * 113) + (Y * Y) / (71 * 71) < 1.
```

4 Регулярни изрази и автомати

Фиксираме крайна азбука от символи Σ. Чрез нея ще дадем дефиниции за регулярни езици и регулярни изрази.

Дефиниция 1 (Регулярен език) *Празният език $\emptyset$ и $\{ a \}$, за всеки символ $a \in \Sigma$, са регулярни езици. Ако L_0 и L_1 са регукярни езици, то тяхното обединение $L_0 \cup L_1$, тяхната композиция $L_0 \bullet L_1$ и ${L_0}^*$ (звезда на Клини) също са регулярни езици.*

Примери: $\{ \epsilon \} = \emptyset^*$, $L \cup \emptyset = L$, $L \bullet \emptyset = \emptyset$, $L \bullet \{ \epsilon \} = L$.

Дефиниция 2 (Регулярен израз) *Регулярните изрази са начин за записване на регулярни езици.*

- *регулярният израз $\emptyset$ е означение за празния език;*
- *регулярният израз ϵ е означение за езика, който се състои само от празната дума;*
- *регулярният израз a, за всеки символ $a \in \Sigma$, е означение за $\{ a \}$;*

Ако R_0 и R_1 са означения за езиците L_0 и L_1, тогава

- *регулярният израз $R_0 + R_1$ е означение за езика $L_0 \cup L_1$;*
- *регулярният израз $R_0 R_1$ е означение за езика $L_0 \bullet L_1$;*
- *регулярният израз ${R_0}^*$ е означение за езика ${L_0}^*$.*

Пример: израза $(ab + c)^*$ е обозначение на езика $\{ \epsilon, ab, c, abab, abc, cab, cc, \dots \}$.

В Prolog програми ще записваме регулярните изрази чрез константи и списъци. Т.к. $\emptyset$ и ϵ имат свойствата на неутрални елементи, относно операциите обединение и композиция, тях ще записваме с константите 0 и 1. Останалите регулярни изрази записваме така: a, [R0, +, R1], [R0, R1], [R0, *]. Например израза abc записваме като [[a, b], c], $(ab + c)^*$ записваме като [[[a, b], +, c], *].
Думите над азбука Σ също ще записваме чрез списъци. Например думата $gfdxxpg$ се записва като [g, f, d, x, x, p, g].
Заб.: В Strawberry Prolog може да се наложи да се пише '+' и '*' вместо + и *.

Ще дефинираме предикат match, който по подаден регулярен израз разпознава или генерира думите от езика, съответстващ на израза. За целта по регулярния израз построяваме недетерминиран автомат без ϵ-преходи (чрез предикат automaton). След това чрез предикат traverse обхождаме автомата за да определим дали думата е част от езика, който се разпознава от автомата, или да генерираме всички думи от езика.

```
match(R, W) :- automaton(R, A), traverse(A, W).
```

Недетерминирани автомати ще представяме като списъци от четири елемента [Q, S, F, D], където Q е множеството от състояния на автомата (състоянията ще означаваме с числа), S и F са подмножества на Q и представляват множествата от начални и крайни състояния на автомата, а D е релацията на преходите. Релацията на преходите записваме, като списък от тройки
[[Q1, A1, DQ1], [Q2, A2, DQ2], ...], където всяка тройка [Qi, Ai, DQi] представлява преход от състояние Qi към състояние DQi по символ Ai.
Така, реализираме traverse по следния начин:

```
traverse([Q, S, F, D], W) :- member(SQ, S), path([Q, S, F, D], SQ, W).
path([Q, S, F, D], CQ, []) :- member(CQ, F).
path([Q, S, F, D], CQ, [A | W]) :-
    member([CQ, A, DQ], D), path([Q, S, F, D], DQ, W).
```

Предикатът path всъщност генерира/разпознава езика на състоянието CQ от подадения автомат - $L_A(cq)$.

Реализация на automaton:

```
automaton(0, [[0], [0], [], []]).
automaton(1, [[0], [0], [0], []]).
automaton(A, [[0, 1], [0], [1], [[0, A, 1]]]) :-
    A \= 0, A \= 1, A \= [], A \= [_ | _].

renumber([Q, S, F, D], N, [QN, SN, FN, DN]) :-
    renumberList(Q, N, QN), renumberList(S, N, SN), renumberList(F, N, FN),
    renumberRelation(D, N, DN).
renumberList([], _, []).
renumberList([H | T], N, [HN | TN]) :- HN is H + N, renumberList(T, N, TN).
renumberRelation([], _, []).
renumberRelation([[Q, A, DQ] | T], N, [[QN, A, DQN] | TN]) :-
    QN is Q + N, DQN is DQ + N, renumberRelation(T, N, TN).

automaton([R0, +, R1], [Q, S, F, D]) :-
    automaton(R0, [Q0, S0, F0, D0]), automaton(R1, A), count(Q0, N),
    renumber(A, N, [Q1, S1, F1, D1]), append(Q0, Q1, Q),
    append(S0, S1, S), append(F0, F1, F), append(D0, D1, D).

preFinalToStart(F0, [], S1, []).
preFinalToStart(F0, [[Q, A, DQ] | T], S1, AD) :-
    member(DQ, F0), toStart(Q, A, S1, DS), preFinalToStart(F0, T, S1, ADT),
    append(DS, ADT, AD).
preFinalToStart(F0, [[Q, A, DQ] | T], S1, AD) :-
    not(member(DQ, F0)), preFinalToStart(F0, T, S1, AD).
toStart(Q, A, [], []).
toStart(Q, A, [SQ | T], [[Q, A, SQ] | DT]) :- toStart(Q, A, T, DT).

additionalStart(S0, F0, S1, S1) :- member(X, S0), member(X, F0).
additionalStart(S0, F0, S1, []) :- not((member(X, S0), member(X, F0))).

automaton([R0, R1], [Q, S, F1, D]) :-
    R1 \= *, R1 \= +, automaton(R0, [Q0, S0, F0, D0]), automaton(R1, A),
    count(Q0, N), renumber(A, N, [Q1, S1, F1, D1]), append(Q0, Q1, Q),
    preFinalToStart(F0, D0, S1, AD), append(D0, AD, ID), append(ID, D1, D),
    additionalStart(S0, F0, S1, AS), append(S0, AS, S).

automaton([R0, +], [Q0, S0, F0, D]) :-
    automaton(R0, [Q0, S0, F0, D0]),
    preFinalToStart(F0, D0, S0, AD), append(D0, AD, D).
automaton([R0, *], A) :- automaton([1, +, [R0, +]], A).
```

Какъв е изхода при цел ?- match([[a, *], +, b], X)? Как може да се избегне това, че се зацикля с резултати [], [a], [a, a] и т.н. и не се извежда [b]?

Допълнително самостоятелно упражнение: да се избере някое представяне на дървета (списък от двойки, списък с наследници, вложени списъци или др.) и да се реализират обхождания в дълбочина и в широчина.

5 Граматики

Целта е да се реализира на Пролог предикат, който по дадена (контекстно-свободна) граматика, генерира чрез преудовлетворяване всички думи, които се пораждат от граматиката. Основната идея е, да се генерира редица от правилата и да се провери, дали тази редица е коректен извод на дума от граматиката.

Затова първо се реализира предикат за генериране на всички мултимножества (редици), които са подмножество на даден начален списък от правила. Това е предиката multisubset:

```
multisubset([], []).
multisubset(L, M) :-
        count(L, N), ntuple(N, T), repeatcorresponding(L, T, R),
        permutate(R, M).

repeat(_, 0, []).
repeat(H, N, [H | T]) :- N1 is N - 1, repeat(H, N1, T).

repeatcorresponding([], [], []).
repeatcorresponding([H | T], [N | TN], R) :-
        repeat(H, N, HR), repeatcorresponding(T, TN, TR), append(HR, TR, R).
```

Принципа на работа на multisubset е по началния списък L, да се генерира редица от n естесвени числа T, където n е дължината на L, и след това всеки елемент от L да се повтори толкова пъти, колкото указва съответното му число от редицата T. Това се прави от repeatcorresponding. Така, елемент на L няма да е наличен в резултата, ако съответстващото му число е 0. След това разултата се пермутира, за да се постигнат всички възможни подредби. Предиката за пермутиране се реализира така:

```
out(H, L, M) :- append(N, [H | T], L), append(N, T, M).
permutate([], []).
permutate(L, [H | T]) :- out(H, L, M), permutate(M, T).
```

При реализацията на multisubset се използва предикат ntuple, който по подадено число N, генерира в T всички n-орки естествени числа.

```
ntuple(N, T) :- int(S), ntuplesum(N, S, T).

ntuplesum(0, 0, []).
ntuplesum(1, S, [S]).
ntuplesum(N, S, [H | T]) :-
        N > 1, between(0, S, H),
        S1 is S - H, N1 is N - 1, ntuplesum(N1, S1, T).
```

За да се определи, дали дадена редица от правила от граматиката е извод на дума, трябва да се реализират предикати, които прилагат правилата върху празната дума и дават крайния резултат. Предикат applyrule прилага едно правило, докато applyrules поредица от правила. Всяко правило се записва като списък от вида [N, [a1, a2, ..., an]], където тези символи представляват правилото $N \to a_1 a_2 \dots a_n$.

```
applyrule(W, [N, A], NW) :-
        append(B, [N | E], W), not(member(B, N)),
        append(B, A, MW), append(MW, E, NW).
```

```
applyrules(W, [], W).
applyrules(W, [R | T], NW) :- applyrule(W, R, MW), applyrules(MW, T, NW).
```

Накрая реализираме финалния предикат generate, който по подадена граматика (граматиката се описва със списък от правилата L, списък от нетерминалите N и начална аксиома S) генерира редица от правила I и проверява, дали тази редица е извод (т.е. дали започва с началната аксиома и накрая се получава дума без нетерминали).

```
generate(L, N, S, W) :-
        multisubset(L, I), I = [[S, _] | _], applyrules([], I, W),
        not(member(N, A)), member(W, A)).
```

Спомагателните предикати за генериране на естествени числа, определяна на дължината на списък, конкатениране на списъци и определяне на принадлежност към списък се реализират така:

```
int(0).
int(X) :- int(Y), X is Y + 1.

between(A, B, A) :- A =< B.
between(A, B, C) :- A < B, A1 is A + 1, between(A1, B, C).

count([], 0).
count([_ | T], C) :- count(T, C1), C is C1 + 1.

append([], L, L).
append([H | T], L, [H | R]) :- append(T, L, R).

member(L, H) :- append(_, [H | _], L).
```

Така, например, граматиката за езика $a^n b^n$ се описва с L = [[s, [a, s, b]], [s, []]] и N = [s]. Думите, пораждани от тази граматика, могат да се генерират с целта

```
?- generate([[s, [a, s, b]], [s, []]], [s], s, W).
```

По същия начин можем да боравим и с граматики от общ вид. Трябва само да променим начина, по който се записват правилата. Вместо да използваме правила с един нетерминал в лявата част - [N, [a1, a2, ..., an]], ще използваме правила от вида [[b1, b2, ..., bm, N, c1, c2, ..., ck], [a1, a2, ..., an]], които записват $b_1 b_2 \ldots b_m N c_1 c_2 \ldots c_k \to a_1 a_2 \ldots a_n$. Остава само да се промени предиката applyrule така:

```
applyrule(W, [F, A], NW) :-
        append(MB, E, W), append(B, F, MB), not(sublist(B, F)),
        append(B, A, MW), append(MW, E, NW).

sublist(L, M) :- append(_, K, L), append(M, _, K).
```

Останалите предикати са без промяна ...

6 Задачи за упражнение: Пролог

Даден е списък от списъци. Да се напише предикат, който разпознава дали всички членове на спсъка са непразни.

I начин:

```
check(L) :- not(member([], L)).
```

II начин:

```
check([]).
check([H | T]) :- not(H = []), check(T).
```

III начин:

```
check([]).
check([[_ | _] | T]) :- check(T).
```

Даден е списък от списъци. Да се напише предикат, който разпознава дали има елемент на списъка, който съдържа елементите на всички други елементи на списъка.

I начин:

```
check(L) :- member(X, L), not(member(Y, L), not(subset(Y, X))).
```

II начин:

```
check(L) :- member(X, L), not(member(Y, L), member(Z, Y), not(member(Z, X))).
```

Даден е списък от списъци. Да се напише предикат, който разпознава дали за всеки елемент X на списъка, има елемент Y на списъка, така че X и Y нямат общи елементи.

```
check(L) :-
    not(member(X, L), not(member(Y, L), not(member(Z, X), member(Z, Y)))).
```

Даден е списък от списъци от числа. Да се напише предикат, който проверява дали за всеки член на списъка със сума по-малка от 10, има друг член, който го съдържа и е със сума по-голяма от 50.

I начин:

```
helpCheck([], _).
helpCheck([H | Q], L) :-
    sum(H, S), S < 10,
    member(X, L), sum(X, T), T > 50, subset(H, X),
    helpCheck(Q, L).
helpCheck([H | Q], L) :- sum(H, S), S >= 10, helpCheck(Q, L).
check(L) :- helpCheck(L, L).
```

II начин:

```
check(L) :- not((member(X, L), sum(X, S), S < 10)).
check(L) :-
    member(X, L), sum(X, S), S < 10,
    member(Y, L), sum(Y, T), T > 50, subset(X, Y),
    rest(X, L, M), check(M).
```

III начин:

```
check(L) :-
    not
    ((
        member(X, L), sum(X, S), S < 10,
        not((member(Y, L), sum(Y, T), T > 50, subset(X, Y)))
    )).
```

Казваме, че списъкът от числа $x_0, x_1, \ldots, x_n$ е специален ако за всяко k, такова че $1 \leq k \leq \frac{n}{2}$, е изпълнено

- $x_{2k} = x_{x_{k+2}}$, ако $1 \leq k+2 \leq n$ и $1 \leq x_{k+2} \leq n$;
- $x_{2k} = 2x_k + 2$, в противен случай.

Да се напише предикат, който разпознава дали даден списък е специален.

```
memberAt([X | _], 1, X).
memberAt([_ | T], N, X) :- N1 is N - 1, memberAt(T, N1, X).
kCondition(L, N, K) :-
    K1 is K + 2,1 =< K1, K1 =< N, memberAt(L, K1, X), 1 =< X, X =< N.
checkK(L, N, K) :-
    kCondition(L, N, K), K1 is K + 2, memberAt(L, K1, X), memberAt(L, X, Y),
    K2 is K * 2, memberAt(L, K2, Y).
checkK(L, N, K) :-
    not(kCondition(L, N, K)), memberAt(L, K, X), Y is 2* X + 2,
    K2 is K * 2, memberAt(L, K2, Y).
special(L) :-
    count(L, N), N1 is N // 2, not(between(1, N1, K), not(checkK(L, N, K))).
```

Казваме, че естествените числа n, m и k образуват питагорова тройка, ако $n^2 + m^2 = k^2$. Да се дефинира на Prolog предикат p(X,N,M,K), който по даден списък от списъци от числа X генерира в N, M и K всички възможни питагорови тройки, такива че сумата N+M+K е число, по-малко от последен елемент на някой елемент на X, N и M не са втори елементи на никой елемент на X и K е равно на сумата на елементите на някой елемент на X.

```
p(X,N,M,K) :-
    member(Y, X), last(L, Y), L1 is L - 1, between(0, L1, S),
    between(0, S, N), S1 is S - N, between(0, S1, M), K is S1 - M,
    not(member([_, N | _], X)), not(member([_, M | _], X)),
    member(Z, X), sum(Z, K), N * N + M * M =:= K * K.
```

Да се напише предикат който намира всички положителни дроби M/N и K/L, такива че N > M > 0, K > L > 0, M/N * K/L = 2 и M + K < A, където A е дадено като параметър.

```
fractions(A, M/N, K/L) :-
    A1 is A - 1, between(3, A1, S), between(1, S, M), K is S - M, K > 0,
    K1 is K - 1, between(1, K1, L), AS is 2 * A * A,
    M1 is M + 1, between(M1, AS, N),
    M * K =:= 2 * N * L.
```

Даден е списък от списъци от числа, като всеки от списъците от числа има точно по четири елемента, които представляват координатите на долния ляв ъгъл и горния десен ъгъл на правоъгълник. Да се напише предикат, който генерира координатите на всички точки от равнината, които се съдържат в точно нечетен брой от правоъгълниците.

```
min(A, B, A) :- A =< B.
min(A, B, B) :- A > B.
max(A, B, A) :- A > B.
max(A, B, B) :- A =< B.
minmaxX([[MinX, _, MaxX, _]], MinX, MaxX).
minmaxX([[X1, _, X2, _] | T], MinX, MaxX) :-
    minmaxX(T, MinXR, MaxXR), min(X1, MinXR, MinX), max(X2, MaxXR, MaxX).
minmaxY([[_, MinY, _, MaxY]], MinY, MaxY).
minmaxY([[_, Y1, _, Y2] | T], MinY, MaxY) :-
    minmaxY(T, MinYR, MaxYR), min(Y1, MinYR, MinY), max(Y2, MaxYR, MaxY).
inRectangle(X, Y, [X1, Y1, X2, Y2]) :- X1 =< X, X =< X2, Y1 =< Y, Y =< Y2.
inRectanglesCount([], _, _, 0).
inRectanglesCount([[X1, Y1, X2, Y2] | T], X, Y, C) :-
    inRectangle(X, Y, [X1, Y1, X2, Y2]), inRectanglesCount(T, X, Y, CR),
    C is CR + 1.
inRectanglesCount([[X1, Y1, X2, Y2] | T], X, Y, C) :-
    not(inRectangle(X, Y, [X1, Y1, X2, Y2])), inRectanglesCount(T, X, Y, C).
points(L, X, Y) :-
    minmaxX(L, MinX, MaxX), minmaxY(L, MinY, MaxY),
    between(MinX, MaxX, X), between(MinY, MaxY, Y),
    inRectanglesCount(L, X, Y, C), C mod 2 =:= 1.
```

Ако условието беше да търсим точките, които се съдържат в точно четен брой правоъгълници, тогава щеше да се наложи да генерираме всички точки от равнината и да ги тестваме, дали са решения, вместо само точките от определен правоъгълник (безкрайно много от точките се съдържат в 0 правоъгълници). За целта ни трябва предикат, който да изброява двойките естествени числа.

Пример за предикат за генериране на всички двойки естествени числа (без ограничение).
Демонстриране, че int(X), int(Y) не върши работа (някои двойки никога не се генерират).
I начин:

```
pair(X, Y) :- int(N), between(0, N, X), Y is N - X.
```

II начин:

```
stage(N, N, Y) :- N > 0, N1 is N - 1, between(0, N1, Y).
stage(N, X, N) :- N > 0, N1 is N - 1, between(0, N1, X).
stage(N, N, N).
pair(X, Y) :- int(N), stage(N, X, Y).
```

III начин: Чрез функция, която кодира двойките естествени. Например $c(x, y) = 2^x(2y+1)$. Предиката всъщност реализира c^{-1}.

```
powerOf2(C, X) :-
    between(0, C, X), P is 2 ** X, C mod P =:= 0,
    not(between(0, C, X1), X < X1, P1 is 2 ** X1, C mod P1 =:= 0).
pair(X, Y) :- int(N), powerOf2(N, X), M is N / (2 ** X), Y is M // 2.
```

7 Изпълнимост на формули

- Семантика на формули от първи ред; Примери;
- Структури за формулите от програми на Prolog (намек за Ербранови структури).
- Свободни и свързани променливи;
- Преименуване на свързани променливи;
- Изпълнимост и вярност;
- Отворени и затворени формули; Изпълнимост и вярност на затворени формули;
- Формули от вида "за всеки елемент с дадено свойство" и "съществува елемент с дадено свойство": $\forall x(\varphi(x) \Rightarrow \dots)$ и $\exists x(\varphi(x)\ \&\ \dots)$.
- Изпълнимост на множество от формули;
- Примери за изпълнимост на множества от формули:
  - формули описващи носители с един елемент, два елемента, ...;
  - наредби;
  - групи и полета; $f(f(a,b),c) = f(a,f(b,c))$, $\exists x \forall y(f(x,y) = y\ \&\ f(y,x) = y)$ ...;
  - аритметични закони (може би аритметика на Пресбургер без аксиомата за индукция);
- Игра за намиране на структури, изпълняващи множества от формули.
- Формули описващи носител с безкраен брой елементи:
  - с безброй много формули: $\neg\forall x \forall y(x = y)$, $\neg\forall x\forall y\forall y(x = y \vee y = z \vee x = z)$, ...;
  - с една формула: $\forall x \neg p(x,x)\ \&\ \forall x\forall y\forall z(p(x,y)\ \&\ p(y,z) \Longrightarrow p(x,z))\ \&\ \forall x \exists y\ p(x,y)$.

8 Определимост и неопределимост

8.1 Определимост: основни дефиниции

Дефиниция 3 (Определимост на множества) *Нека $\mathcal{A} = (A, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ е структура. Казваме че множеството $B \subseteq A^n$ за $n \geq 1$ е* определимо чрез формула $\varphi(x_0, \ldots, x_{n-1})$ *с точно n свободни променливи, ако*

$$\langle a_0, \ldots, a_{n-1} \rangle \in B \Longleftrightarrow \mathcal{A}, v^{x_0,\ldots,x_{n-1}}_{a_0,\ldots,a_{n-1}} \models \varphi(x_0, \ldots, x_{n-1})$$

Свойства:
Ако $\mathcal{A} = (A, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ е структура и $B \subseteq A^n$ и $C \subseteq A^n$ са две определими множества, съответно чрез формулите φ_B и φ_C, то тогава

- $A^n \setminus B$ е определимо с формулата $\neg\varphi_B$;
- $B \cup C$ е определимо с формулата $\varphi_B \vee \varphi_C$;
- $B \cap C$ е определимо с формулата $\varphi_B \,\&\, \varphi_C$.

Дефиниция 4 (Определимост на елементи, функции и релации) *Нека $\mathcal{A} = (A, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ е структура. Нека $a \in A$, p е релация над A, а f е функция в A. Тогава казваме, че*

- *a е* определим *ако $\{\ a\ \}$ е определимо чрез някоя формула;*
- *p е* определима *ако $\{\ \langle a_0, \ldots, a_n \rangle \mid a_i \in A, p(a_0, \ldots, a_n)\ \}$ е определимо чрез някоя формула;*
- *f е* определима *ако $\{\ \langle a_0, \ldots, a_m, b \rangle \mid a_i \in A, b \in A, f(a_0, \ldots, a_m) = b\ \}$ е определимо чрез някоя формула.*

Пример I: В структурата $\mathcal{N} = (\mathbb{N}, =, .)$

- 1 е определим елемент с формулата $\varphi_1(x) \rightleftharpoons \forall y(x.y = y)$;
- n/m е определима релация с формулата $\varphi_/(x, y) \rightleftharpoons \exists z(x.z = y)$;
- множеството на простите числа е определимо с формулата $\varphi_{pr}(x) \rightleftharpoons \forall y(\varphi_/(y, x) \Rightarrow (\varphi_1(y) \vee y = x))$.

Заб.: Когато определяме нови елементи, релации или функции, можем да използваме означения за вече дефинирани обекти. Така финалната формула се получава, като заместим означенията с формулите, които ги определят (след евентуални преименувния на променливи). Например като пишем формулата $\forall y(\varphi_/(y, x) \Rightarrow (\varphi_1(y) \vee y = x))$, всъщност се има пред вид $\forall y(\exists z(y.z = x) \Rightarrow (\forall z(y.z = z) \vee y = x))$. Нямаме право, обаче, да използваме означения за обекти, които не са част от езика и още не са определен чрез формули!

Пример II: Когато променливите се оценяват с множества, а езика се състои само от предикат за принадлежност $\in$

- $x \subseteq y$ е определима с $\varphi_\subseteq(x, y) \rightleftharpoons \forall z(z \in x \Rightarrow z \in y)$;
- $x = y$ е определима с $\varphi_=(x, y) \rightleftharpoons \forall z(z \in x \Leftrightarrow z \in y)$ или $\varphi_=(x, y) \rightleftharpoons \varphi_\subseteq(x, y) \,\&\, \varphi_\subseteq(y, x)$;
- $x \cap y = z$ е определима с $\varphi_\cap(x, y, z) \rightleftharpoons \forall t(t \in z \Leftrightarrow t \in x \,\&\, t \in y)$;
- $x \cup y = z$ е определима с $\varphi_\cup(x, y, z) \rightleftharpoons \forall t(t \in z \Leftrightarrow t \in x \vee t \in y)$;
- $x \setminus y = z$ е определима с $\varphi_\setminus(x, y, z) \rightleftharpoons \forall t(t \in z \Leftrightarrow t \in x \,\&\, \neg t \in y)$;

- $\emptyset$ е определимо с $\varphi_\emptyset(x) \rightleftharpoons \neg\exists y(y \in x)$.

Пример III: Определимостта съответства на реализирането на нови методи (предикати в Prolog) в програма, чрез медоти (предикати) с които вече разполагаме. Например, нека имаме структура $(\mathbb{N}, p, m, <, =, 0)$, където p е едноместен предикат и $p(x)$ означва, че x е просто, а m е двуместна операция, такава че $m(x, y)$ е остатъка на x при деление на y. Тогава в тази структура определяме предиката $d(x, y)$, който означава че y е най-малкия прост делител на x, чрез формулата

$$m(x, y) = 0 \ \& \ p(y) \ \& \ \neg\exists z(m(x, z) = 0 \ \& \ p(z) \ \& \ z < y)$$

Таза формула съответства на тялото на клаузата

```
d(N, P) :- N mod P =:= 0, p(P), not(N mod Q =:= 0, p(Q), Q < P).
```

от Пролог програма, която реализира същия предикат, при положение че разполагаме с предикат за разпознаване на простите числа.

8.2 Критерий за неопределимост

Дефиниция 5 (Изоморфизъм между структури) *Нека $\mathcal{A} = (A, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ и $\mathcal{B} = (B, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ са две структури в един и същи език. Нека h е фукнкция от A в B. h се нарича* изоморфизъм от $\mathcal{A}$ в $\mathcal{B}$ *ако h е биекция и h запазва действието на символите от езика:*

- $p_i^{\mathcal{A}}(a_0, a_1, \ldots, a_{n_i})$ *т.с.т.* $p_i^{\mathcal{B}}(h(a_0), h(a_1), \ldots, h(a_{n_i}))$, *за всеки предикат p_i и за всеки $a_0, a_1, \ldots, a_{n_i} \in A$;*
- $f_i^{\mathcal{A}}(a_0, a_1, \ldots, a_{k_i}) = b$ *т.с.т.* $f_i^{\mathcal{B}}(h(a_0), h(a_1), \ldots, h(a_{k_i})) = h(b)$, *за всяка функция f_i и за всеки $a_0, a_1, \ldots a_{k_i}, b \in A$;*
- $c_i^{\mathcal{B}} = h(c_i^{\mathcal{A}})$, *за всяка константа c_i.*

Примери: Нека $\mathcal{A} = (\mathbb{N}^+, p, c)$ е структура, така че $p^{\mathcal{A}} =<$ и $c^{\mathcal{A}} = 5$, а $\mathcal{B} = (\mathbb{Z}^-, p, c)$ е структура, така че $p^{\mathcal{B}} =>$ и $c^{\mathcal{B}} = -5$. Тогава $h(x) = -x$ е изоморфизъм от $\mathcal{A}$ в $\mathcal{B}$.

Дефиниция 6 (Автоморфизъм върху структура) *Нека $\mathcal{A}$ е структура и h е изоморфизъм от $\mathcal{A}$ в $\mathcal{A}$. Тогава h наричаме* автоморфизъм върху $\mathcal{A}$.

Примери: Идентитетът $h(x) = x$ е автоморфизъм върху всяка структура.
Заб.: За разлика от ограниченията при писане на формули за определимост, тук за дефиниране на h (и за изоморфизми и за автоморфизми) можем свободно да използваме обекти и конструкции, които не са от езика и не са определими в структурата.

Твърдение 1 *Нека $\mathcal{A} = (A, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ е структура, $B \subseteq A^n$ за $n \geq 1$ е определимо множество, а h е автоморфизъм върху $\mathcal{A}$. Тогава B остава непроменено под действието на h. Т.е.*

$$\langle a_0, \ldots, a_{n-1}\rangle \in B \longleftrightarrow \langle h(a_0), \ldots, h(a_{n-1})\rangle \in B.$$

Това твърдение позволява да се формулира следния достатъчен критерий за неопределимост

Критерий 1 (Критерий за неопределимост) *Нека $\mathcal{A} = (A, p_0, p_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots)$ е структура. Нека $a \in A$, p е релация над A, а f е функция в A. Тогава*

- *ако има автоморфизъм h върху $\mathcal{A}$, такъв че $a \neq h(a)$, то a не е определим в $\mathcal{A}$;*
- *ако има автоморфизъм h върху $\mathcal{A}$ и $a_0, \ldots, a_n \in A$, такива че или $p(a_0, \ldots, a_n)$ и $\neg p(h(a_0), \ldots, h(a_n))$ или $\neg p(a_0, \ldots, a_n)$ и $p(h(a_0), \ldots, h(a_n))$, то p не е определима в $\mathcal{A}$;*

- *ако има автоморфизъм h върху $\mathcal{A}$ и $a_0, \ldots, a_m, b \in A$, такива че или $f(a_0, \ldots, a_m) = b$ и $f(h(a_0), \ldots, h(a_m)) \neq h(b)$ или $f(a_0, \ldots, a_m) \neq b$ и $f(h(a_0), \ldots, h(a_m)) = h(b)$, то f не е определима в $\mathcal{A}$.*

Този критерий не е единствен начин за доказване на неопределимост. Например, има структури, за които единствения автоморфизъм е идентитета, и все пак има неопределими множества в структурата.

8.3 Определимост и неопределимост в числови структури с предикати за събиране и умножение

Разглеждаме структури от вида $(\mathbb{A}, s, m)$, $(\mathbb{A}, s)$ и $(\mathbb{A}, m)$, където $\mathbb{A}$ е $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ или $\mathbb{R}$, а s и m са триместни предикати, описващи графиките на операциите събиране и умножение във всяко от числовите множества:

$$s(x, y, z) \longleftrightarrow x + y = z$$
$$m(x, y, z) \longleftrightarrow x.y = z$$

Целта е да се покаже за всяка от тези 12 структури, кои елементи на носителя са определими и кои не.

Определимост и неопределимост в $(\mathbb{R}, s, m)$

Определими са всички рационални числа.

$$\varphi_0(x) \rightleftharpoons \forall y\ s(x, y, y)$$
$$\varphi_0(x) \rightleftharpoons \forall y\ m(x, y, x)$$
$$\varphi_1(x) \rightleftharpoons \forall y\ m(x, y, y)$$
$$\varphi_n(x) \rightleftharpoons \exists y \exists z(\varphi_1(y)\ \&\ \varphi_{n-1}(z)\ \&\ s(y, z, x))$$
$$\varphi_{-n}(x) \rightleftharpoons \exists y \exists z(\varphi_0(y)\ \&\ \varphi_n(z)\ \&\ s(x, z, y))$$
$$\varphi_{\frac{p}{q}}(x) \rightleftharpoons \exists y \exists z(\varphi_p(y)\ \&\ \varphi_q(z)\ \&\ \neg\varphi_0(z)\ \&\ m(z, x, y))$$

Освен това са определими и следните ирационални числа: рационално число на рационална степен.

$$\varphi_{\sqrt[3]{-\frac{19}{7}}}(x) \rightleftharpoons \exists y(\varphi_{-\frac{19}{7}}(y)\ \&\ \exists z(m(x, x, z)\ \&\ m(z, x, y)))$$

Следните релации също са определими

$$\varphi_=(x, y) \rightleftharpoons \exists z(\varphi_0(z)\ \&\ s(x, z, y))$$
$$\varphi_<(x, y) \rightleftharpoons \exists z \exists t(\neg\varphi_0(t)\ \&\ m(t, t, z)\ \&\ s(x, z, y))$$

Определимост и неопределимост в $(\mathbb{N}, s)$

В тази структура са определими всички елементи от носителя. 0 и релацията $=$ се определят със същите формули, както в структурата $(\mathbb{R}, s, m)$. Релацията $<$ се опеделя така

$$\varphi_<(x, y) \rightleftharpoons \exists z(\neg\varphi_0(z)\ \&\ s(x, z, y))$$

Така всяко число, по-голямо от 0, се определя по следния начин

$$\begin{aligned}\varphi_n(x) \rightleftharpoons \exists x_0 \ldots \exists x_{n-1}(&\varphi_0(x_0)\ \&\ \ldots\ \&\ \varphi_{n-1}(x_{n-1})\ \&\\ &\varphi_<(x_0, x)\ \&\ \ldots\ \&\ \varphi_<(x_{n-1}, x)\ \&\\ &\forall y(\varphi_<(y, x) \Rightarrow \varphi_=(y, x_0) \vee \cdots \vee \varphi_=(y, x_{n-1})))\end{aligned}$$

Алтернативно, 0 може да се определи чрез $<$

$$\varphi_0(x) \rightleftharpoons \neg\exists y \varphi_<(y,x)$$

Така същия резултат, че всички елементи са определими, може да се постигне и в $(\mathbb{N}, <, =)$.

В такива структури, в които всички елементи са определими, има единствен автоморфизъм - идентитетът. Това се показва така: ако допуснем че има друг автоморфизъм h, то ще има елемент a, такъв че $h(a) \neq a$ и тогава, по критерия, този елемент няма да е определим, което е противоречие.

Въпреки че няма друг автоморфизъм, освен идентитета, в тези структури има неопределими множества. Това, например, се вижда т.к. броят на подмножествата на $\mathbb{N}$ е $2^{\mathbb{N}}$, което е мощност строго по-голяма от броя на формулите (те са изброимо много).

Определимост и неопределимост в $(\mathbb{N}, m)$

Тук определими са 0 и 1 (за 0 се използва формулата с m от $(\mathbb{R}, s, m)$).

За показване на неопределимостта на останалите числа се използва следния автоморфизъм:

Всяко естесвено число, различно от 0, може да се представи в следния каноничен вид $2^{s_0}3^{s_1}\dots p_i^{s_i}p_{i+1}^{s_{i+1}}\dots$, където $2, 3, \dots p_i, \dots$ е редицата на простите числа и само краен брой от $s_0, s_1, \dots s_i, \dots$ са ненулеви. С $n^{i\leftrightarrow j}$ означаваме числото $2^{s_0}3^{s_1}\dots p_i^{s_j}\dots p_j^{s_i}\dots$, където $2^{s_0}3^{s_1}\dots p_i^{s_i}\dots p_j^{s_j}\dots$ е каноничния запис на n. Така $n^{i\leftrightarrow j}$ просто разменя степените на $i^{\text{-тото}}$ и $j^{\text{-тото}}$ прости числа от каноничния запис на n. За да имаме тоталност, полагаме $0^{i\leftrightarrow j} = 0$.

Например $6^{0\leftrightarrow 2} = 15$, $6^{1\leftrightarrow 2} = 10$, $6^{0\leftrightarrow 1} = 6$, $35^{0\leftrightarrow 1} = 35$, $1^{i\leftrightarrow j} = 1$.

Нека n е такова, че $n > 1$. Тогава нека p_i е някой прост делител на n, а p_j е просто число, което не дели n. Тогава $h(n) = n^{i\leftrightarrow j}$ е функция, която не запазва n. Така за показване на неопределимостта на всички числа, освен 0 и 1, ще са нужни безкрайно много такива функции (това не винаги е така).

Остава само да докажем, че $h(n) = n^{i\leftrightarrow j}$ наистина е автоморфизъм.

Нека $n \neq l$ за $n \neq 0$ и $l \neq 0$. Тогава има просто число p_k, което има различни степени в каноничните записи на n и l. Ако $k = i$, тогава $h(n)$ и $h(l)$ ще се различават по степента на p_j. Ако $k = j$, $h(n)$ и $h(l)$ ще се различават по p_i. В противен случай, $k \neq i$ и $k \neq j$ и тогава $h(n)$ и $h(l)$ ще продължат да се различават по p_k. Така във всички случаи $h(n) \neq h(l)$ (при $n = 0$ или $l = 0$ това се показва лесно). Значи h е инекция.
h е сюрекция т.к. за всяко число l, $h(l)$ е неговия първообраз - т.е. $h(h(l)) = l$ (това не е вярно за всяка биекция).
Доказателство, че h запазва действието на m за всеки $n \neq 0$ и $l \neq 0$: първо ще покажем че $h(n).h(l) = h(n.l)$.

$$\begin{aligned}
h(n).h(l) = & \\
&= h(2^{s_0}3^{s_1}\dots p_i^{s_i}\dots p_j^{s_j}\dots).h(2^{r_0}3^{r_1}\dots p_i^{r_i}\dots p_j^{r_j}\dots) && = \\
&= 2^{s_0}3^{s_1}\dots p_i^{s_j}\dots p_j^{s_i}\dots \cdot 2^{r_0}3^{r_1}\dots p_i^{r_j}\dots p_j^{r_i}\dots && = \\
&= 2^{s_0+r_0}3^{s_1+r_1}\dots p_i^{s_j+r_j}\dots p_j^{s_i+r_i}\dots && = \\
&= h(2^{s_0+r_0}3^{s_1+r_1}\dots p_i^{s_i+r_i}\dots p_j^{s_j+r_j}\dots) && = \\
& && = h(n.l)
\end{aligned}$$

Така чрез $h(n).h(l) = h(n.l)$ и факта, че h е биекция, можем да докажем, че $m(n,l,k) \longleftrightarrow m(h(n),h(l),h(k))$ (т.е. че $n.l = k$ е изпълнено, тогава и само тогава, когато $h(n).h(l) = h(k)$ е изпълнено).
При $n = 0$ или $l = 0$ проверката, че h запазва действието на m, е тривиална.

Равенството също е определимо чрез m

$$\varphi_=(x,y) \rightleftharpoons \exists z(\varphi_1(z)\ \&\ m(x,z,y))$$

Така, след като определим и предиката за делимост (както в $(\mathbb{N},=,.)$ от началните примери), вече можем да определим и множеството на простите числа.

Определимост и неопределимост в $(\mathbb{N},s,m)$

В тази структура също са определими всички елементи. Могат да се използват формулите от $(\mathbb{N},s)$ или $(\mathbb{R},s,m)$.

Определимост и неопределимост в $(\mathbb{Z},s)$

Определим елемент е само 0. За останалите елементи се използва автоморфизма $h(x)=-x$. Тук можем да минем и само с един общ автоморфизъм.

Определимост и неопределимост в $(\mathbb{Z},m)$

Определими са 0, 1 и -1. За 0 и 1 се използват формулите от $(\mathbb{N},m)$, а за -1 формулата

$$\varphi_{-1}(x) \rightleftharpoons \exists y(\varphi_1(y)\ \&\ m(x,x,y)\ \&\ \neg\varphi_1(x))$$

За неопределимостта на останалите елементи използваме следния автоморфизъм:
$h(x)=sign(x).|x|^{i\leftrightarrow j}$.

Определимост и неопределимост в $(\mathbb{Z},s,m)$

Тук също са определими всички елементи (чрез формулите от $(\mathbb{R},s,m)$).

Определимост и неопределимост в $(\mathbb{Q},s)$

Тук положението е както при $(\mathbb{Z},s)$. За неопределимостта можем да използваме не само автоморфизма $h(x)=-x$, но и всеки автоморфизъм от вида $h(x)=kx$, където $k\neq 0$ и $k\neq 1$.

Определимост и неопределимост в $(\mathbb{Q},m)$

Определими са 0, 1 и -1 чрез формулите от $(\mathbb{Z},m)$.

За неопределимостта на останалите рационални числа $\frac{p}{q}$ гледаме дали $p\neq 0$ и $p\neq 1$ и $p\neq -1$. Тогава използваме автоморфизма $h(\frac{p}{q})=sign(\frac{p}{q}).\frac{|p|^{i\leftrightarrow j}}{|q|}$. В противен случай, т.к. $\frac{p}{q}\neq 0$ и $\frac{p}{q}\neq 1$ и $\frac{p}{q}\neq -1$, то $q\neq 0$ и $q\neq 1$ и $q\neq -1$ и тогава използваме $h(\frac{p}{q})=sign(\frac{p}{q}).\frac{|p|}{|q|^{i\leftrightarrow j}}$.

Определимост и неопределимост в $(\mathbb{Q},s,m)$

Тук също са определими всички елементи (чрез формулите от $(\mathbb{R},s,m)$).

Определимост и неопределимост в $(\mathbb{R},s)$

Положението е както при $(\mathbb{Z},s)$ и $(\mathbb{Q},s)$.

Определимост и неопределимост в $(\mathbb{R}, m)$

Определими са 0, 1 и -1 чрез формулите от $(\mathbb{Z}, m)$ и $(\mathbb{Q}, m)$.
За неопределимостта на останалите числа може да се използва $h(x) = x^k$ или $h(x) = x^{\frac{1}{k}}$, където h е нечетно естествено число (например $h(x) = x^3$ или $h(x) = \sqrt[3]{x}$). Също може да се използва $h(x) = sign(x).\sqrt{|x|}$.

9 Задачи за упражнение: изпълнимост и (не)определимост

Нека структурата $\mathcal{A} = (A, p)$ е такава, че A е множеството от затворени отсечки в равнината, с дължина по-голяма от 0, а $p(x, y)$ е вярно т.с.т. x и y имат поне една обща точка. В тази структура ще определим следните понятия:

- x е подотсечка на y - $x \sqsubseteq y$: $\varphi_{\sqsubseteq}(x, y) \rightleftharpoons \forall z(p(x, z) \Rightarrow p(y, z))$;
- x съвпада с y: $\varphi_{=}(x, y) \rightleftharpoons \varphi_{\sqsubseteq}(x, y)\ \&\ \varphi_{\sqsubseteq}(y, x)$.

Всяка точка ще означаваме с произволна двойка отсечки, които се пресичат точно в точката и в никоя друга. Първо е нужно да имаме условие, което определя, че дадена двойка отсечки дефинира точка (т.е. че имат точно една пресечна точка). След това можем да определяме и други условия за точки.

- условие, че x и y определят точка:
  $\varphi_{\text{point}}(x, y) \rightleftharpoons p(x, y)\ \&\ \neg\exists z(\varphi_{\sqsubseteq}(z, x)\ \&\ \varphi_{\sqsubseteq}(z, y))$.

Всяка права ще означаваме с произволна отсечка, която лежи на нея.

- отсечка x лежи на права y (същото условие определя и съвпадане на прави):
  $\varphi_{\text{sl}}(x, y) \rightleftharpoons \exists z(\varphi_{\sqsubseteq}(x, z)\ \&\ \varphi_{\sqsubseteq}(y, z))$;
- точка (x, y) лежи на отсечка z: $\varphi_{\in}(x, y, z) \rightleftharpoons p(x, z)\ \&\ p(y, z)\ \&\ \forall t(\varphi_{\sqsubseteq}(t, z) \Rightarrow (p(x, t) \Leftrightarrow p(y, t)))$;
- точка (x, y) лежи на права z: $\varphi_{\in line}(x, y, z) \rightleftharpoons \exists t(\varphi_{\in}(x, y, t)\ \&\ \varphi_{\text{sl}}(z, t))$;
- точка (x, y) съвпада с точка (z, t): $\varphi_{=\text{point}}(x, y, z, t) \rightleftharpoons \forall s(\varphi_{\in}(x, y, s) \Leftrightarrow \varphi_{\in}(z, t, s))$;
- три прави x, y и z се пресичат в една точка:
  $\varphi_{3\text{ lines}}(x, y, z) \rightleftharpoons \exists s\exists t(\varphi_{\text{point}}(s, t)\ \&\ \varphi_{\in line}(s, t, x)\ \&\ \varphi_{\in line}(s, t, y)\ \&\ \varphi_{\in line}(s, t, z))$;
- права x е успоредна на права y: $\varphi_{||}(x, y) \rightleftharpoons \neg\exists z\exists t(\varphi_{\text{sl}}(z, x)\ \&\ \varphi_{\text{sl}}(t, y)\ \&\ p(z, t))$.

Неопределимо условие е това, че две отсечки имат равна дължина (автоморфизмът е хомотетия).

Да се намери структура изпълняваща формулата $\exists x\neg p(x)$. Решения:

- $\mathcal{A} = (\mathbb{N}, p)$, където $p^{\mathcal{A}}(x) \longleftrightarrow 0 < x$;
- $\mathcal{A} = (\mathbb{N}, p)$, където $p^{\mathcal{A}}(x) \longleftrightarrow x \neq c$ за фиксирана константа c.

Да се намери структура изпълняваща формулата $\forall x\exists y(p(x, y)\ \&\ \neg p(y, x)\ \&\ p(y, y))$. $\mathcal{A} = (\mathbb{N}, \leq)$ е едно решение.

Да се намери структура изпълняваща формулите:

$$\neg\forall x\forall y\forall z(p(x, y)\ \&\ p(y, z) \Rightarrow p(x, z))$$
$$\forall x\exists y(p(x, y)\ \&\ \neg p(y, x)\ \&\ p(y, y))$$

Решения:

- $\mathcal{A} = (\mathbb{N}, \leq \setminus \{\ \langle 0, 2\rangle\ \})$;
- $\mathcal{A} = (\mathbb{N} \setminus \{\ 0, 1\ \}, p)$, където $p^{\mathcal{A}}(x, y) \longleftrightarrow x \leq y\ \&\ (x, y) \neq 1$;
- $\mathcal{A} = (\mathbb{N} \cup \{\ \star\ \}, \leq \cup\{\ \langle\star, 0\rangle\ \})$.

Дадена е структурата $S = (\mathbb{N}, c, f, r)$, където интерпретацията на c, f и r е:

$$c^S = 1,$$
$$f^S(n, m) = n + m + 1,$$
$$r^S(n, m) \longleftrightarrow n = m.$$

Да се определят елементите 1, 0, 3 и 2. Решения:

$$\varphi_1(x) \rightleftharpoons r(x, c),$$
$$\varphi_0(x) \rightleftharpoons r(f(x, x), c),$$
$$\varphi_3(x) \rightleftharpoons r(f(c, c), x),$$
$$\varphi_2(x) \rightleftharpoons \exists y(\varphi_0(y) \,\&\, r(f(y, c), x)).$$

Дадена е структурата $S' = (\mathbb{N}, c, g, r)$, където интерпретацията на g е:

$$g^{S'}(n) = n^2,$$

а интерпретацията на c и r е както в S. Да се определят елементите 1 и 0.

$$\varphi_1(x) \rightleftharpoons r(x, c),$$
$$\varphi_0(x) \rightleftharpoons r(x, g(x)) \,\&\, \neg\varphi_1(x).$$

Дадена е структурата $S'' = (\mathbb{N}, c, r)$, където интерпретацията на c и r е както по-горе. Да се докаже, че елементите 2 и 3 не са определими в S''. Доказва се с автоморфизма

$$h(x) = \begin{cases} 3, & x = 2; \\ 2, & x = 3; \\ x, & x \neq 2 \,\&\, x \neq 3. \end{cases}$$

Да се докаже, че елементите 2 и 3 не са определими също и в S'. Доказва се с автоморфизма $h(x) = x^{0 \leftrightarrow 1}$.

Дадена е структурата $\mathcal{A} = (A, r, s)$, където A е множеството на нечетните естествени числа, а интерпретацията на r и s е:

$$r^{\mathcal{A}}(a, b, c) \longleftrightarrow a + b + 1 = c,$$
$$s^{\mathcal{A}}(a, b, c) \longleftrightarrow ab = c.$$

Да се определят елементите 1, 3, 5 и 7 и множеството
$B = \{\, n \mid n$ е нечетно число, което не се дели на 3 $\}$.

$$\varphi_1(x) \rightleftharpoons \forall y\ s(y, x, y),$$
$$\varphi_3(x) \rightleftharpoons \exists y(\varphi_1(y) \,\&\, r(y, y, x)),$$
$$\varphi_5(x) \rightleftharpoons \exists y \exists z(\varphi_1(y) \,\&\, \varphi_3(z) \,\&\, r(y, z, x)),$$
$$\varphi_7(x) \rightleftharpoons \exists y \exists z(\varphi_1(y) \,\&\, \varphi_5(z) \,\&\, r(y, z, x)),$$
$$\varphi_B(x) \rightleftharpoons \neg\exists z \exists t(\varphi_3(z) \,\&\, s(z, t, x)).$$

Дадена е структурата $\mathcal{A} = (\mathbb{N}, p)$, където интерпретацията на p е:

$$p^{\mathcal{A}}(n, m, k) \longleftrightarrow n^3 m^6 = k.$$

Да се докаже, че елементът m е определим, тогава и само тогава, когато $m \in \{\, 0, 1 \,\}$.

$$\varphi_0(x) \rightleftharpoons \forall y\ s(y, x, x),$$
$$\varphi_1(x) \rightleftharpoons p(x, x, x) \,\&\, \neg\varphi_0(x).$$

Това са формулите, които определят 0 и 1. Неопределимостта на останалите елементи се показва с автоморфизмите $h(n) = n^{i \leftrightarrow j}$.

10 Нормани форми

Нормална форма на формула е еквивалентно преобразуване на формулата, което има специфичен вид\запис.

10.1 Конюнктивна нормална форма

Обикновено, съждителни или безкванторни формули се привеждат в Конюнктивна Нормална Форма (КНФ). Единствените булеви операции в конюнктивната нормална форма са $\neg$, $\&$ и $\vee$.

Дефиниция 7 Литерал *е или атомарна формула или отрицание на атомарна формула.*

Примери: $p(x)$, $\neg q(x,y)$, $r(f(x),c)$. Създителни литерали: p, $\neg q$.

Дефиниция 8 Елементарна дизюнкция *е формула от вида* $L_0 \vee L_1 \vee \cdots \vee L_k$, *където* L_i *е литерал за всяко* i.

Дефиниция 9 *Една безкванторна или съждителна формула е в* Конюнктивна нормална форма *ако е от вида* $D_0 \,\&\, D_1 \,\&\, \ldots \,\&\, D_n$, *където* D_i *е елементарна дизюнкция за всяко* i.

Употреба на КНФ. Сравнение с Дизюнктивна Нормална Форма (ДНФ).

Всяка безкванторна или съждителна формула може да се приведе в КНФ. Например, чрез следните операции:

1. елиминиране на $\Rightarrow$ и $\Leftrightarrow$;
2. придвижване на отрицанията към вътрешността на формулата (чрез законите на де Морган) и елиминиране (при първа възможност) на двойни отрицания;
3. прилагане на дистрибутивния закон $\varphi \vee (\psi \,\&\, \chi) \equiv (\varphi \vee \psi) \,\&\, (\varphi \vee \chi)$, докато формулата е приведрна в КНФ.

Упражнение: $\neg((p(x) \vee \neg r(y)) \,\&\, (q(y) \vee \neg p(x)))$

10.2 Пренексна нормална форма

Вид, идея. Всички квантори са в началото на формулата. По-добра интуиция за зависимостите между променливите.

Дефиниция 10 *Една предикатна формула е в* Пренексна нормална форма *ако е от вида* $Q_0 x_0 Q_1 x_1 \ldots Q_n x_n \varphi(x_0, x_1 \ldots x_n)$, *където всяко* Q_i *е или* $\forall$ *или* $\exists$ *и в* $\varphi(x_0, x_1 \ldots x_n)$ *не се срещат квантори. Безкванторната част* $\varphi(x_0, x_1 \ldots x_n)$ *се нарича още* матрица *на формулата.*

Може да има допълнително изискване за привеждане на матрицата в КНФ или ДНФ или друга нормална форма.

Правила за ивеждане на квантор пред скоби:

- $\forall x \varphi(x) \,\&\, \psi \equiv \forall x(\varphi(x) \,\&\, \psi)$, ако x не се среща свободно в ψ;
- $\exists x \varphi(x) \,\&\, \psi \equiv \exists x(\varphi(x) \,\&\, \psi)$, ако x не се среща свободно в ψ;
- $\forall x \varphi(x) \vee \psi \equiv \forall x(\varphi(x) \vee \psi)$, ако x не се среща свободно в ψ;
- $\exists x \varphi(x) \vee \psi \equiv \exists x(\varphi(x) \vee \psi)$, ако x не се среща свободно в ψ.

Правилата за извеждане на квантори са същите, когато горните формули са с разменени дясни и ляви аргументи на конюнкциите и дизюнкциите.

Когато x се среща свободно и не можем да прилагаме правилата, тогава преименуваме променливата от квантора.

Преименувания: $\forall x\varphi(x) \equiv \forall y\varphi(y), \exists x\varphi(x) \equiv \exists y\varphi(y)$.

Правила за преминаване на отрицание през квантор:

- $\neg\forall x\varphi \equiv \exists x\neg\varphi$;
- $\neg\exists x\varphi \equiv \forall x\neg\varphi$.

Операции за привеждане на произволна предикатна формула в ПНФ:

1. елиминиране на $\Rightarrow$ и $\Leftrightarrow$;
2. правят се достатъчно на брой преименувания, така че да нямаме два квантора по една и съща променлива (ако формулата е затворена, това ни гарантира, че няма да са нужни повече преименувания);
3. придвижване на отрицанията към вътрешността на формулата (чрез законите на де Морган и правилата за преминаване на отрицания през квантори) и елиминиране (при първа възможност) на двойни отрицания;
4. придвижване на кванторите към началото на формулата чрез горните четири правила за ивеждане на квантор пред скоби;
5. прилагане на дистрибутивния закон $\varphi \vee (\psi \,\&\, \chi) \equiv (\varphi \vee \psi) \,\&\, (\varphi \vee \chi)$, докато безкванторната част на формулата (матрицата) е приведрна в КНФ.

Правила за ивеждане на квантор пред скоби, когато имаме $\Rightarrow$:

- $\forall x\varphi(x) \Rightarrow \psi \equiv \exists x(\varphi(x) \Rightarrow \psi)$, ако x не се среща свободно в ψ;
- $\exists x\varphi(x) \Rightarrow \psi \equiv \forall x(\varphi(x) \Rightarrow \psi)$, ако x не се среща свободно в ψ;
- $\varphi \Rightarrow \forall x\psi(x) \equiv \forall x(\varphi \Rightarrow \psi(x))$, ако x не се среща свободно в φ;
- $\varphi \Rightarrow \exists x\psi(x) \equiv \exists x(\varphi \Rightarrow \psi(x)$, ако x не се среща свободно в φ.

Тук има съществено значение дали квантора е преди или след импликацията.

Упражнение: различна поредност на кванторите в крайния резултат.
Например ПНФ на $\forall x\varphi(x) \vee \exists y\psi(y)$ може да е както $\forall x\exists y(\varphi(x) \vee \psi(y))$, така и $\exists y\forall x(\varphi(x) \vee \psi(y))$.

10.3 Скулемова нормална форма

Вид, идея. Премахване на екзистенциалните квантори и получаване на универсална формула. Запазва се изпълнимост, но не е еквивалентна форма.

Дефиниция 11 *Една предикатна формула е в* Скулемова нормална форма *ако не съдържа екзистенциални квантори и е получена от формула в Пренексна нормална форма чрез прилагане на долните две правила за премахване на екзистенциалните квантори.*

Привеждаме в СНФ, като първо привеждаме в ПНФ и след това премахваме последователно от ляво надясно екзистенциалните квантори. Премахването става така:

- ако най-левия екзистенциален квантор не е предшестван от универсални квантори, тогава премахваме квантора и заместваме променливата в матрицата на формулата с нова константа: $\exists x(\ldots x \ldots x \ldots) \longrightarrow \ldots c \ldots c \ldots$;
- ако най-левия екзистенциален квантор е предшестван от универсални квантори, тогава премахваме квантора и заместваме променливата в матрицата на формулата с нова функция на променливите от предшестващите квантори:
$\forall x_1, \ldots, \forall x_n \exists y(\ldots y \ldots y \ldots) \longrightarrow \forall x_1, \ldots, \forall x_n(\ldots f(x_1, \ldots, x_n) \ldots f(x_1, \ldots, x_n) \ldots)$.

Заб.: Нямаме еквивалентност. Запазва се само изпълнимост в разширената структура.

Операции за привеждане на произволна предикатна формула в СНФ:

1. елиминиране на $\Rightarrow$ и $\Leftrightarrow$;
2. правят се достатъчно на брой преименувания, така че да нямаме два квантора по една и съща променлива (ако формулата е затворена, това ни гарантира, че няма да са нужни повече преименувания);
3. придвижване на отрицанията към вътрешността на формулата (чрез законите на де Морган и правилата за преминаване на отрицания през квантори) и елиминиране (при първа възможност) на двойни отрицания;
4. придвижване на кванторите към началото на формулата чрез горните четири правила за ивеждане на квантор пред скоби;
5. Скулемизираме, докато премахнем всички екзистенциални квантори от формулата;
6. прилагане на дистрибутивния закон $\varphi \vee (\psi \,\&\, \chi) \equiv (\varphi \vee \psi) \,\&\, (\varphi \vee \chi)$, докато безкванторната част на формулата (матрицата) е приведрна в КНФ.

Упражнение: $\exists x(\neg(\exists y(\forall x p(x, y) \Rightarrow r(y)) \vee (r(z) \Leftrightarrow \forall z p(z, x))))$.

11 Метод на резолюцията

Същност: показва неизпълнимост на множество от формули. Смисъла му е, че показваме че нещо не е вярно, като достигаме до противоречие.

Приложения:

- да покажем, че множеството $\{ \varphi_1, \varphi_2, \ldots, \varphi_n \}$ е неизпълнимо;
- в частност, когато множеството се състои само от една формула, можем да показваме, че формула не е изпълнима;
- можем да показваме вярност на формули от вида: $\{ \varphi_1 \,\&\, \varphi_2 \,\&\, \ldots \,\&\, \varphi_n \Rightarrow \psi_1 \vee \psi_2 \vee \ldots \vee \psi_m \}$, като разгледаме множеството $\{ \varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi_1, \neg\psi_2, \ldots, \neg\psi_m \}$ и приложим метода върху него;
- един специален случай на горната формула е когато имаме само едно следствие ψ и тогава можем да показваме вярност на ψ, като приложим метода върху $\neg\psi$.

Последните две приложения съответстват на метода на допускане на противното.

Употреба в интерпретатори на Prolog.

Пълнота и коректност на метода на резолюцията.

Метода може да показва само неизпълнимост. Рядко може да се използва за показване на изпълнимост.

11.1 Съждителна резолюция

Дефиниция 12 Дизюнкт *наричаме множество от литерали. Празното множество от литерали наричаме* празен дизюнкт. *Казваме, че дизюнкт е изпълним в дадена структура при дадена оценка ако поне един от литералите е изпълним в структурата при тази оценка. Празният дизюнкт е тривиално неизпълним.*

Интуицията за дизюнкт е да съответства на елементарна дизюнкция ....

Твърдение 2 *Нека имаме оценка v и два дизюнкта $D_1 = D_1' \bigcup \{ L \}$ и $D_2 = D_2' \bigcup \{ \neg L \}$, които съдържат противоположните литерали L и $\neg L$, са изпълними при тази оценка. Тогава дизюнкта $D_3 = D_1' \bigcap D_2'$ също е изпълним при оценка v.*

Дефиниция 13 *Нека дизюнкт D_1 съдържа литерал L_1, а дизюнкт D_2 съдържа литерал L_2, като двата литерала L_1 и L_2 са противоположни. Тогава дизюнкта $D_3 = (D_1 \setminus \{ L_1 \}) \bigcup (D_2 \setminus \{ L_2 \})$ се нарича (непосредствена/съждителна) резолвента на D_1 и D_2.*

Внимание: Може да се вади само по една двойка противоположни литерали. Например и $\{ \neg q(x), q(x), r(x) \}$ и $\{ p(x), \neg p(x), r(x) \}$ са резолвенти на $\{ p(x), \neg q(x) \}$ и $\{ \neg p(x), q(x), r(y) \}$, но $\{ r(y) \}$ не е.

Дефиниция 14 *Дизюнкта D е* резолвентно изводим *от множеството от дизюнкти $\mathcal{D}$ ако е последен член в редица $D_0, D_1, \ldots D_n$ и за всяко D_i или $D_i \in \mathcal{D}$ или D_i е резолвента на D_j и D_k, където $j < i$ и $k < i$.*

Пример: Да се покаже, че множеството $\{ p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q \}$ е неизпълнимо.

Съвет: Най-добре е първо да се пробват да се комбинират най-кратките (с най-малко елементи) дизюнкти, които съдържат противоположни литерали. Например, да се започне със синглетоните ...

11.2 Предикатна резолюция

Пример: Третото, от долните твърдения, следва от първите две:

1. Всички гърци са хора;
2. Аристотел е грък;
3. Аристотел е човек.

Това не може да се покаже, използвайки само съждителна резолюция. Формулите, съответстващи на твърденията, са $\forall x(g(x) \Rightarrow h(x))$, $g(a)$ и $h(a)$, където $h(x)$ означава, че x е човек, $g(x)$ означава, че x е грък, а a е константа обозначаваща Аристотел. Причината е, че $\neg g(x)$ $g(a)$, например, не са противоположни литерали.

Нужда от унификации. Примери за унифицируемост.

Дефиниция 15 *Нека дизюнкт D_1 съдържа литерал L_1, а дизюнкт D_2 съдържа литерал L_2, като $\sigma(L_1)$ и $\tau(L_2)$ са противоположни, където σ и τ са субституции. Тогава дизюнкта $D_3 = \sigma(D_1 \setminus \{ L_1 \}) \bigcup \tau(D_2 \setminus \{ L_2 \})$ се нарича либерална/предикатна резолвента на D_1 и D_2.*

Съответно можем да разширим дефиницията за резолвентно изводим дизюнкт, като позволим използването и на либерални резолвенти, а не само непосредствени такива.

Пример с програма на Prolog.

```
p(X) :- q(X).
p(X) :- r(X).
q(a).
r(b).
r(c).
?- p(X).
```

Чрез дърво на извод можем да видим, че целта се удовлетворява при стойности $X = a$, $X = b$ и $X = c$. Същия резултат може да се получи и като приложим метода на резолюцията върху формулите, съответстващи на клаузите от програмата - $\forall x(q(x) \Rightarrow p(x))$, $\forall x(r(x) \Rightarrow p(x))$, $r(a)$, $r(b)$, $r(c)$ и $\exists x p(x)$. Съответствие между успешните пътища в дървото на извод (с клаузите и унификациите по тях) и изводите на празния дизюнкт от формулите (с дизюнктите от формулите и субституциите в изводите).

Ако приложим метода на резолюцията върху множество от формули, което е изпълнимо, е възможно да преценим, че празния дизюнкт не е изводим от множеството, но това не винаги е възможно.

Например, да вземем множеството от аксиомите за релация на еквивалентност -
$\forall x p(x,x)$, $\forall x \forall y(p(x,y) \Rightarrow p(y,x))$ и $\forall x \forall y \forall z(p(x,y)$ & $p(y,z) \Rightarrow p(x,z))$. Дизюнктите в този случай са $\{ p(x,x) \}$, $\{ \neg p(x,y), p(y,x) \}$ и $\{ \neg p(x,y), \neg p(y,z), p(x,z) \}$ и т.к. във всеки от тях има по един положителен литерал, то е видно, че от това множество не може да се изведе празния дизюнкт.

11.3 Задачи

Задача 1: Да се докаже, чрез метода на резолюцията, че следното твърдение е тавтология: Има студент, който ако получи отлична оценка по Логическо програмиране, то всеки студент ще получи отлична оценка по Логическо програмиране.

Решение:
Чрез неформални съображения виждаме, че ако всички студенти получат отлична оценка, то всеки един от тях би могъл да бъде този от предпоставката на твърдението (всичко това е при положение, че има поне един студент). Ако, обаче, има поне един студент, който не получи отлична оценка, то тогава той няма да изпълнява предпоставката и така твърдението е тривиално вярно (т.к. е импликация с

грешна предпоставка).
А решението чрез метода на резолюцията е, като приложим метода върху формулата, съответстваща на твърдението:

$\exists x(p(x) \Rightarrow \forall x p(x))$, тук $p(x)$ означава, че x получава отлична оценка по Логическо програмиране.

Задача 2: Да се докаже, чрез метода на резолюцията, че твърдение 2 е следствие от твърдение 1.

1. Който пие получава махмурлук;
2. Хората пият заради махмурлука.

Решение:
Прилагаме метода на резолюцията върху формули (това са формулите, съответстващи на твърдения 1 и 2):

I вариант:

1. $\forall x(d(x) \Rightarrow h(x))$, тук $d(x)$ означава, че x пие, а $h(x)$ означава, че x получава махмурлук;
2. $\forall x(\neg h(x) \Rightarrow \neg d(x))$, тази формула съответства на твърдението, че ако никой не получаваше махмурлук, то никой нямаше да пие.

При този запис на твърденията се вижда и без метода на резолюцията, че второто твърдение е еквивалентно на първото:

$$\begin{aligned}
&\forall x(d(x) \Rightarrow h(x)) && \equiv \\
&\forall x(\neg d(x) \vee h(x)) && \equiv \\
&\forall x(h(x) \vee \neg d(x)) && \equiv \\
&\forall x(\neg\neg h(x) \vee \neg d(x)) && \equiv \\
&\forall x(\neg h(x) \Rightarrow \neg d(x))
\end{aligned}$$

Иначе, чрез метода на резолюцията получаваме:

II вариант:

1. $\forall x(\exists y(a(y)\ \&\ d(x,y)) \Rightarrow \exists y(h(y)\ \&\ g(x,y))$, тук $a(x)$ означава, че x е алкохол, $d(x,y)$ означава, че x пие y, $h(x)$ означава, че x е махмурлук, а $g(x,y)$ означава, че x получава y;
2. $\neg\exists x h(x) \Rightarrow \neg\exists x\exists y(a(y)\ \&\ d(x,y))$.

Задача 3: Да се докаже, чрез метода на резолюцията, че твърдение 3 е следствие от твърдения 1 и 2.

1. Някои пациенти уважават докторите;
2. Никой пациент не уважава шарлатаните;
3. Никой доктор не е шарлатан.

Решение:
Прилагаме метода на резолюцията върху формули:

1. $\exists x(p(x)\ \&\ \forall y(d(y) \Rightarrow r(x,y)))$, тук $p(x)$ означава, че x е пациент, $d(x)$ означава, че x е доктор, а $r(x,y)$ означава, че x уважава y;
2. $\neg\exists x\exists y(p(x)\ \&\ s(y)\ \&\ r(x,y))$, тук $s(x)$ означава, че x е шарлатан;

3. $\neg\exists x(d(x)\ \&\ s(x))$.

Задача 4: Да се докаже, чрез метода на резолюцията, че твърдение 4 е следствие от твърдения 1, 2 и 3.

1. Митничарите обискират всеки, който преминава границата и не е дипломат;
2. Някои трафиканти са преминали границата и са били обискирани само от трафиканти;
3. Дипломатите не са трафиканти.
4. Някои митничари са трафиканти.

Решение:
Прилагаме метода на резолюцията върху формули:

1. $\forall x(b(x)\ \&\ \neg d(x) \Rightarrow \exists y(c(y)\ \&\ s(y,x)))$, тук $b(x)$ означава, че x преминава границата, $d(x)$ означава, че x е дипломат, $c(x)$ означава, че x е митничар, а $s(x,y)$ означава, че x обискира y;
2. $\exists x(t(x)\ \&\ b(x)\ \&\ \forall y(s(y,x) \Rightarrow t(y)))$, тук $t(x)$ означава, че x е трафикант;
3. $\neg\exists x(d(x)\ \&\ t(x))$;
4. $\exists x(c(x)\ \&\ t(x))$.