



ЗАПИСКИ ПО ЛОГИЧЕСКО ПРОГРАМИРАНЕ



гл. ас. д-р Владислав Ненчев

Катедра по Математическа логика и приложенията ѝ
Факултет по математика и информатика
Софийски университет "Св. Климент Охридски"

vladislavn@fmi.uni-sofia.bg

София, 17 декември 2016г.

❖ Съдържание ❖

	Страница
Част I: Логическо програмиране	1
Декларативен стил на програмиране	2
Логическо програмиране	4
Предикатни езици от първи ред	7
Език за програмиране Пролог	15
Определимост и неопределимост	17
Логически задачи	29
Интерпретиране на Пролог програми	34
Изпълнение на Пролог програми чрез обратен извод	36
Изпълнимост на множества от затворени литерали	41
Правила за извод	45
Табло метод	51
Изпълнимост на множества от универсални формули	61
Унифициране	68
Пренексна и Скулемова нормални форми	73
Изпълнение на Пролог програми с Табло с унифициране	78
Изпълнимост на множества от дизюнкти	80
Метод на резолюцията	83
Изпълнение на Пролог програми чрез резолюция	90
Пълнота на метода на резолюцията	92
Част II: Програмиране на Пролог	97
Съставяне на Пролог клауза по формула	98
Разпознаватели и генератори	102
Разделяне на списъци на части чрез append	105
Предикати за подсписък, пермутация и подмножество	107
Сортиране на списъци	109
Предикати за работа с графи	111
Решаване на NP-пълни задачи	115
Реализиране на Пролог клаузи чрез определимост	118
Работа с аритметични операции	121
Предикати за генериране на (без)крайни множества	123
Предикати за работа с регулярни изрази	127
Предикати за работа с контекстно-свободни граматики	130



Част I: Логическо програмиране



❖ Декларативен стил на програмиране ❖

Декларативното програмиране е специфичен стил на програмиране, където целта/задачата се постига/решава не чрез изброяване на конкретни стъпки за преобразуване на началните данни в крайния резултат (това е другия основен стил на програмиране - *процедурно програмиране*), ами чрез **описване** на структурата/вида на началните данни, **описване** на крайната цел и **описване** на (често математически) принципи за свеждане между използваните понятия.

Примери за езици за програмиране, които следват декларативния стил, са: *Prolog* и *Datalog* (логическо програмиране), *Lisp*, *Scheme* и *Clojure* (Функционално програмиране), *SQL* (заявки към бази данни), *HTML* и *XML* (структурни езици), регулярни изрази и лямбда изрази (използват с в много езици за програмиране).

Пример (Обединение на списъци/таблицу).

В процедурното програмиране обединение на два списъка се прави така:

```
List result = new List();
for(int i = 0; i < list1.Length; i++)
{
    result.Add(list1[i]);
}
for(int i = 0; i < list2.Length; i++)
{
    result.Add(list2[i]);
}
return result;
```

В декларативните езици често има специални операции, които описват много по-кратко това действие. Ето как се прави това при заявки към бази данни:

```
SELECT * FROM Table1
UNION ALL
SELECT * FROM Table2
```

Също така декларативен стил на програмиране може да се използва за оформяне на кода и в програми, написани на традиционни процедурни езици.

Пример (Принадлежност към списък).

При процедурното програмиране, принадлежност на елемент към списък се определя всеки път със следните стъпки:

```
for(i = 0; i < list.Length; i++)
    if(list[i] == 2)
        return true;
```

При декларативното програмиране, след като веднъж горната логика е реализирана като отделен метод, то оттам нататък принадлежност на елемент към списък се определя със следния израз:

```
return list.Contains(2);
```

Освен това използването на регулярни изрази и ламбда изрази (например в езици като C# и Java) също позволява декларативен стил на програмиране чрез процедурни езици.

Пример (Използване на регулярни изрази).

Процедурния начин за определяне, дали даден текст *pattern* се среща като подниз на друг текст *text* е:

```
for (i = 0; i < text.Length; i++)  
    if (text.Substring(i, pattern.Length) == pattern)  
        return true;
```

Докато с регулярен израз тази проверка се извършва така:

```
return Regex.IsMatch(text, pattern);
```

Тук регулярният израз играе ролята на описание на формата на текста и така постига декларативен ефект.

Пример (Използване на ламбда изрази).

В процедурното програмиране, проверката дали в списък от числа всички елементи са ненулеви се прави така:

```
for (i = 0; i < list.Length; i++)  
    if (list[i] == 0)  
        return false;  
return true;
```

Чрез ламбда израз това може да се постигне по-декларативно:

```
return list.All(element => element != 0);
```

❖ Логическо програмиране ❖

Логическото програмиране е дял от декларативното програмиране, в който за съставянето на програми се използват методи от *Математическата логика* (това е област, която изучава формалните математически свойства на езиците и методите за извеждане на разсъждения/заключения). Една логическа програма типично се състои от формули, като всяка от формулите описва формално/математически или началните данни, или искания краен резултат, или някакво преобразуване между тях.

Непример, за решаване на задачата за намиране на Хамилтонов път в граф може да се образуват две формули - една, която описва как по списъка на върховете на графа можем да получим негова пермутация, а другата описва дали дадена поредност от върхове е път в графа. Така задачата се свежда до това, да се провери дали има пермутация на върховете на графа, която е валиден път.

Литература:

- ☛ *Logic for Applications*, Anil Nerode & Richard Shore, Springer, 1993
- ☛ *First-Order Logic*, Raymond Smullyan, Dover Publications, 1995
- ☛ *Лекции по логическо програмиране*, гл. ас. д-р Антон Зинovieв, <http://www.fmi.uni-sofia.bg/fmi/logic/zinoviev/lp/index.html>
- ☛ *Лекции по логическо програмиране*, проф. д.м.н. Димитър Скордев, <http://logic.fmi.uni-sofia.bg/skordev/lp/lp/index.html>

Следните примери описват три възможни ситуации, при които могат да се използват описания чрез формули за решаване на определена задача в компютърните науки:

Пример (Съвместяване на програми).

При следните термини

$traffic(internet)$	–	извършва се трафик през интернет,
$traffic(intranet)$	–	извършва се трафик по вътрешната мрежа,
$start(scanner)$	–	стартира се сканираща програма,
$start(filter)$	–	стартира се филтрираща програма,

имаме една програма, чието действие се описва с

- ☛ $traffic(internet) \vee traffic(intranet)$
- ☛ $\neg(traffic(internet) \& traffic(intranet))$
- ☛ $traffic(internet) \implies \neg start(scanner) \& start(filter)$
- ☛ $traffic(intranet) \implies start(scanner)$

и втора програма, действаща според

- ☛ $traffic(internet) \vee traffic(intranet)$
- ☛ $\neg(traffic(internet) \& traffic(intranet))$
- ☛ $traffic(internet) \implies start(scanner) \& \neg start(filter)$
- ☛ $traffic(intranet) \implies \neg start(scanner) \& start(filter)$

Въпрос: Съвместими ли са двете програми?

Пример (Снабдяване при търговия).

В търговска фирма имаме следните понятия:

- p — продукт,
- q — (положително) количество,
- $sell(p, q)$ — искаме да продадем количество q от продукт p ,
- $need(p, q)$ — имаме нужда да набавим количество q от продукт p ,
- $buy(p, q)$ — можем да закупим количество q от продукт p ,
- $produce(p, q)$ — можем да произведем количество q от продукт p ,
- $procure(p, q)$ — имаме начин да набавим количество q от продукт p ,
- k — коефициент на допустимо излишество.

Текущата дейност на фирмата може да се опише така:

- ☛ $\forall p \forall q (sell(p, q) \implies need(p, q))$
- ☛ $\forall p \forall q_1 \forall q_2 (buy(p, q_1) \& produce(p, q_2) \implies procure(p, q_1 + q_2))$
- ☛ $sell(bread, 40)$
- ☛ $produce(bread, 50)$
- ☛ $buy(bread, 0)$
- ☛ $sell(wine, 35)$
- ☛ $produce(wine, 20)$
- ☛ $buy(wine, 20)$
- ☛ $sell(chocolate, 70)$
- ☛ $produce(chocolate, 0)$
- ☛ $buy(chocolate, 60)$

Въпроси:

- ☛ $\exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& q_1 > q_2)$

$$\bullet \quad \exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& k.q_1 < q_2)$$

Пример (Диагностика на пациенти).

При диагностика на пациенти имаме следните понятия:

p	–	пациент,
t	–	число (за температура),
$cough(p)$	–	пациентът p кашля,
$snuffles(p)$	–	пациентът p има хрема,
$headache(p)$	–	пациентът p има главоболие,
$temperature(p, t)$	–	пациентът p има температура t ,
$flu(p)$	–	пациентът p има грип,
$pneumonia(p)$	–	пациентът p има пневмония,
$allergy(p)$	–	пациентът p има алергия,
$meningitis(p)$	–	пациентът p има менингит,
$fatigue(p)$	–	пациентът p е уморен.

Текущо, разполагаме с няколко критерия за диагностициране:

- $\forall p (\neg cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t \leq 37) \implies allergy(p))$
- $\forall p (\neg cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t > 37 \& t \leq 40) \implies flu(p))$
- $\forall p (cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t > 37 \& t \leq 40) \implies pneumonia(p))$
- $\forall p (headache(p) \& \exists t (temperature(p, t) \& t \leq 37) \implies fatigue(p))$
- $\forall p (headache(p) \& \exists t (temperature(p, t) \& t > 40) \implies meningitis(p))$
- $\forall p (\neg cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t \geq 38 \& t < 39) \implies flu(p))$

Въпрос: Има ли излишни критерии?

❖ Предикатни езици от първи ред ❖

Дефиниция 1 (Предикатен език).

Предикатен език е множество от означения, което се разделя на три основни групи:

- ☛ *логически операции:* \neg (отрицание), $\&$ (конюнкция), \vee (дизюнкция), \implies (импликация), \iff (еквиваленция), \forall (квантор за всеобщност), \exists (квантор за съществуване);
- ☛ *множество от (безкрайно много) индивидуни променливи:* $Var = \{ x, y, z, \dots \}$;
- ☛ *нелогически символи:* константи ($c, d, e \dots$), функционални символи ($f, g, h \dots$) и предикатни символи ($p, q, r \dots$); всеки от функционалните и предикатните символи има фиксиран брой аргументи.

Пример.

Езика на аритметиката се състои от следните символи:

- ☛ *константи:* $0, 1, 2, \dots, 2016, \dots$;
- ☛ *функционални символи:* $+, -, *$.
- ☛ *предикатни символи:* $=, <, >, \leq, \geq$.

Дефиниция 2 (Терм).

Термовете са изрази, формирани чрез даден предикатен език, които служат за обозначаване на обекти. Термовете биват три основни типа:

- ☛ *всяка индивидуна променлива x може да формира терм;*
- ☛ *всяка константа c може да формира терм;*
- ☛ *ако имаме n -местен функционален символ f и имаме n на брой вече формирани терма - $\tau_1, \tau_2, \dots, \tau_n$ - тогава изразът $f(\tau_1, \tau_2, \dots, \tau_n)$ също представлява терм.*

Примери.

Следните изрази са термове: $x, 17, f(x, y, z), (x + 22) * y, \log_{(x+4)} 7^{(3*z)}$.

Дефиниция 3 (Формула).

Формулите са изрази, формирани чрез даден предикатен език, които служат за записване на твърдения. Формулите биват следните основни видове:

- ☛ *ако имаме n -местен предикатен символ p и имаме n на брой терма - $\tau_1, \tau_2, \dots, \tau_n$ - тогава изразът $p(\tau_1, \tau_2, \dots, \tau_n)$ представлява формула; такива формули наричаме атомарни формули;*

- ако вече сме формирали формула α , то тогава изразът $\neg\alpha$ също представлява формула;
- ако вече сме формирали формули α и β , то тогава изразът $(\alpha \& \beta)$ също представлява формула;
- ако вече сме формирали формули α и β , то тогава изразът $(\alpha \vee \beta)$ също представлява формула;
- ако вече сме формирали формули α и β , то тогава изразът $(\alpha \implies \beta)$ също представлява формула;
- ако вече сме формирали формули α и β , то тогава изразът $(\alpha \iff \beta)$ също представлява формула;
- ако вече сме формирали формула α , а x е индивидуна променлива, то тогава изразът $\forall x\alpha$ също представлява формула;
- ако вече сме формирали формула α , а x е индивидуна променлива, то тогава изразът $\exists x\alpha$ също представлява формула.

Примери.

Следните изрази са формули: $p(x, f(y, c)), \quad p(x) \vee \neg p(x), \quad \forall x p(x), \quad \exists y (\forall x p(x) \implies q(f(y)))$.

Формулата $f(x, f(y, z)) = f(f(x, y), z)$ описва свойството за асоциативност на операцията в група.

Формулите $\forall x p(x, x), \quad \forall x \forall y (p(x, y) \implies p(y, x))$ и $\forall x \forall y \forall z (p(x, y) \& p(y, z) \implies p(x, z))$ описват релация на еквивалентност.

Формулата $\exists x (a * x * x + b * x + c = 0)$ описва, че квадратното уравнение с коефициенти a, b и c има корен.

Дефиниция 4 (Структура).

Ако имаме даден предикатен език $L = \{ c, d, e, \dots, f, g, h, \dots, p, q, r, \dots \}$, тогава структура за този език наричаме комбинацията \mathcal{A} от непразно множество от обекти A и интерпретация на символите от езика L , която:

- на всяка константа c от L съпоставя единствен постоянен елемент от A ; този елемент обозначаваме с $c^{\mathcal{A}}$;
- на всеки n -местен функционален символ f от L съпоставя n -местна функция $f^{\mathcal{A}}$ над A ; т.е. $f^{\mathcal{A}} : A^n \longrightarrow A$;
- на всеки n -местен предикатен символ p от L съпоставя n -местно свойство $p^{\mathcal{A}}$ (т.е. изображение, което за всеки n елемента от A определя единствена стойност истина T или лъжа F); т.е. $p^{\mathcal{A}} : A^n \longrightarrow \{ T, F \}$.

Структурата се записва по следния начин: $\mathcal{A} = (A, c, d, e, \dots, f, g, h, \dots, p, q, r, \dots)$. В този случай L наричаме език на структурата \mathcal{A} , а A наричаме носител на тази структура.

Примери.

Структура $\mathcal{A}_1 = (A, 0, +)$, където:

- ☛ A е множеството на естествените числа;
- ☛ $0^{\mathcal{A}_1}$ е числото нула;
- ☛ $+^{\mathcal{A}_1}$ е операцията за сбор между две числа.

Структура $\mathcal{A}_2 = (A, 0, +)$, където:

- ☛ A е множеството от всички матрици от $\mathbb{R}_{n \times m}$;
- ☛ $0^{\mathcal{A}_2}$ е нулевата матрица;
- ☛ $+^{\mathcal{A}_2}$ е сбор между матрици.

Структура $\mathcal{A}_3 = (A, 0, +)$, където:

- ☛ A е множеството от всички низове/думи над дадена азбука Σ ;
- ☛ $0^{\mathcal{A}_3}$ е празната дума;
- ☛ $+^{\mathcal{A}_3}$ е операцията за конкатенация между низове.

Структура $\mathcal{G}_1 = (V, p)$, където:

- ☛ V е множеството от всички върхове в даден граф;
- ☛ $p^{\mathcal{G}_1}(x, y)$ означава, че има ребро от x до y .

Структура $\mathcal{G}_2 = (V, p)$, където:

- ☛ V е множеството от всички върхове в даден граф;
- ☛ $p^{\mathcal{G}_2}(x, y)$ означава, че има път от x до y .

Дефиниция 5 (Оценка).

Оценка при дадена структура $\mathcal{A} = (A, \dots)$ се нарича изображение $v : Var \rightarrow A$, което на всяка индивидна променлива от Var съпоставя точно един елемент от носителя A на структурата.

Дефиниция 6 (Стойност на терм).

При дадена структура $\mathcal{A} = (A, \dots)$ и оценка v при тази структура за всеки терм τ определяме неговата стойност $v(\tau)$ така:

- ☛ ако термът е индивидната променлива x , тогава $v(\tau) = v(x)$;

- ако термът е константата c , тогава $v(\tau) = c^A$;
- ако термът е израз $f(\tau_1, \tau_2, \dots, \tau_n)$, тогава $v(\tau) = f^A(v(\tau_1), v(\tau_2), \dots, v(\tau_n))$.

Пример.

Ако имаме оценка v , за която $v(x) = 14$, $v(y) = 0$ и $x(z) = 2$, то при стандартна интерпретация на символите от езика на аритметиката имаме, че $v(44) = 44$, $v(x + 17) = 31$ и $v((x * y) + (z * 9)) = 18$.

Дефиниция 7 (Модификация на оценка).

Ако имаме структура $A = (A, \dots)$, оценка v при тази структура, n различни индивидни променливи $x_1, x_2, \dots, x_n \in Var$ и n обекта $a_1, a_2, \dots, a_n \in A$, тогава можем да дефинираме нова оценка $v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n]$, като определим стойността за всяко $y \in Var$ така:

$$v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n](y) = \begin{cases} a_1 & \text{при } y = x_1; \\ a_2 & \text{при } y = x_2; \\ \vdots & \\ a_n & \text{при } y = x_n; \\ v(y) & \text{при } y \neq x_1, y \neq x_2, \dots, y \neq x_n. \end{cases}$$

$v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n]$ наричаме модификация на оценката v .

Пример.

Ако имаме оценка v , за която $v(x) = 14$, $v(y) = 0$ и $x(z) = 2$, то при стандартна интерпретация на символите от езика на аритметиката имаме, че $v(x + 17) = 31$, $v[x = 10](x + 17) = 27$, $v[x = 4, z = 101](y + 8) = 8$ и $v[x = 9, y = 1][x = 5](x * (y + 7)) = 40$.

Дефиниция 8 (Стойност на формула).

При дадена структура $A = (A, \dots)$ и оценка v при тази структура за всяка формула α можем да определим нейната стойност. Възможните стойности са истина T или лъжа F . Т.е. за α определяме дали е вярна при тези структура и оценка - което записваме така $A, v \models \alpha$ - или дали е невярна при дадените структура и оценка - това се записва така $A, v \not\models \alpha$.

Определянето на стойността на α става индуктивно според нейното построение:

- ако α е атомарната формула $p(\tau_1, \tau_2, \dots, \tau_n)$ тогава

$$\begin{aligned} A, v \models \alpha & \longleftrightarrow \text{стойността на } p^A(v(\tau_1), v(\tau_2), \dots, v(\tau_n)) \text{ е } T; \\ A, v \not\models \alpha & \longleftrightarrow \text{стойността на } p^A(v(\tau_1), v(\tau_2), \dots, v(\tau_n)) \text{ е } F; \end{aligned}$$

- ако α е $\neg\beta$ тогава

$$\begin{aligned} A, v \models \alpha & \longleftrightarrow A, v \not\models \beta; \\ A, v \not\models \alpha & \longleftrightarrow A, v \models \beta; \end{aligned}$$

☛ ако α е $(\beta \& \gamma)$ тогава

$$\begin{aligned} \mathcal{A}, v \models \alpha &\longleftrightarrow \mathcal{A}, v \models \beta \text{ и } \mathcal{A}, v \models \gamma; \\ \mathcal{A}, v \not\models \alpha &\longleftrightarrow \mathcal{A}, v \not\models \beta \text{ или } \mathcal{A}, v \not\models \gamma; \end{aligned}$$

☛ ако α е $(\beta \vee \gamma)$ тогава

$$\begin{aligned} \mathcal{A}, v \models \alpha &\longleftrightarrow \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \models \gamma; \\ \mathcal{A}, v \not\models \alpha &\longleftrightarrow \mathcal{A}, v \not\models \beta \text{ и } \mathcal{A}, v \not\models \gamma; \end{aligned}$$

☛ ако α е $(\beta \implies \gamma)$ тогава

$$\begin{aligned} \mathcal{A}, v \models \alpha &\longleftrightarrow \mathcal{A}, v \not\models \beta \text{ или } \mathcal{A}, v \models \gamma; \\ \mathcal{A}, v \not\models \alpha &\longleftrightarrow \mathcal{A}, v \models \beta \text{ и } \mathcal{A}, v \not\models \gamma; \end{aligned}$$

☛ ако α е $(\beta \iff \gamma)$ тогава

$$\begin{aligned} \mathcal{A}, v \models \alpha &\longleftrightarrow \mathcal{A}, v \models \beta \text{ и } \mathcal{A}, v \models \gamma \text{ или } \mathcal{A}, v \not\models \beta \text{ и } \mathcal{A}, v \not\models \gamma; \\ \mathcal{A}, v \not\models \alpha &\longleftrightarrow \mathcal{A}, v \not\models \beta \text{ и } \mathcal{A}, v \models \gamma \text{ или } \mathcal{A}, v \models \beta \text{ и } \mathcal{A}, v \not\models \gamma; \end{aligned}$$

☛ ако α е $\forall x \beta$ тогава

$$\begin{aligned} \mathcal{A}, v \models \alpha &\longleftrightarrow \mathcal{A}, v[x = a] \models \beta \text{ за всяко едно } a \in A; \\ \mathcal{A}, v \not\models \alpha &\longleftrightarrow \mathcal{A}, v[x = a] \not\models \beta \text{ за поне едно } a \in A; \end{aligned}$$

☛ ако α е $\exists x \beta$ тогава

$$\begin{aligned} \mathcal{A}, v \models \alpha &\longleftrightarrow \mathcal{A}, v[x = a] \models \beta \text{ за поне едно } a \in A; \\ \mathcal{A}, v \not\models \alpha &\longleftrightarrow \mathcal{A}, v[x = a] \not\models \beta \text{ за всяко едно } a \in A. \end{aligned}$$

Примери.

$\mathcal{A}, v \models \forall x(p(x) \vee \neg p(x))$ за всеки \mathcal{A} и v .

Формулата $\forall a \forall b \forall c \exists x_1 \exists x_2 (a * x_1 * x_1 + b * x_1 + c = 0 \& a * x_2 * x_2 + b * x_2 + c = 0 \& \neg(x_1 = x_2))$ има стойност истина T за произволна оценка при структура, чийто носител е множеството на комплексните числа \mathbb{C} , а интерпретацията на $0, +, *, \text{ и } =$ е стандартна. Същата формула не е истина, когато носителя е \mathbb{R} , или \mathbb{Q} , или \mathbb{Z} .

Формулата $\forall x \forall y p(x, y)$ е истина за произволна оценка при структура, чийто носител е множеството от всички върхове на свързан граф, а $p(x, y)$ се интерпретира като наличието на път от x до y .

Дефиниция 9 (Свободно и свързано срещане на променлива).

Дадено срещане на променлива x във формула φ наричаме свързано срещане, ако то се намира в областта на действие на квантор по същата тази променлива x .

Съответно, свободно срещане на променлива x във формула φ наричаме такова срещане, което не е свързано.

Примери.

Във формулата $\forall x(p(x, t) \implies \exists z p(z, y) \ \& \ \forall z q(z, t))$, y и t се срещат свободно, а x и z свързано.

Във формулата $\forall x(p(x) \vee p(y)) \ \& \ q(x)$, първото срещане на x е свързано, а второто е свободно.

Заб.: От предните две дефиниции се вижда, че при дадени формула φ и оценка v при структура \mathcal{A} , оценката v е съществена за стойността на φ само за променливите със свободни срещания. За останалите променливи няма значение каква е тяхната стойност при v .

Дефиниция 10 (Затворен терм).

Затворен терм наричаме такъв терм, в който не се срещат никакви променливи.

Примери.

Следните изрази са затворени термове: 12 , π , $17 + (4 * 9)^3$.

Следните изрази не са затворени термове: x , $12 * \pi + y$, $(4 * 9)^{\ln(x)}$.

Дефиниция 11 (Затворена формула).

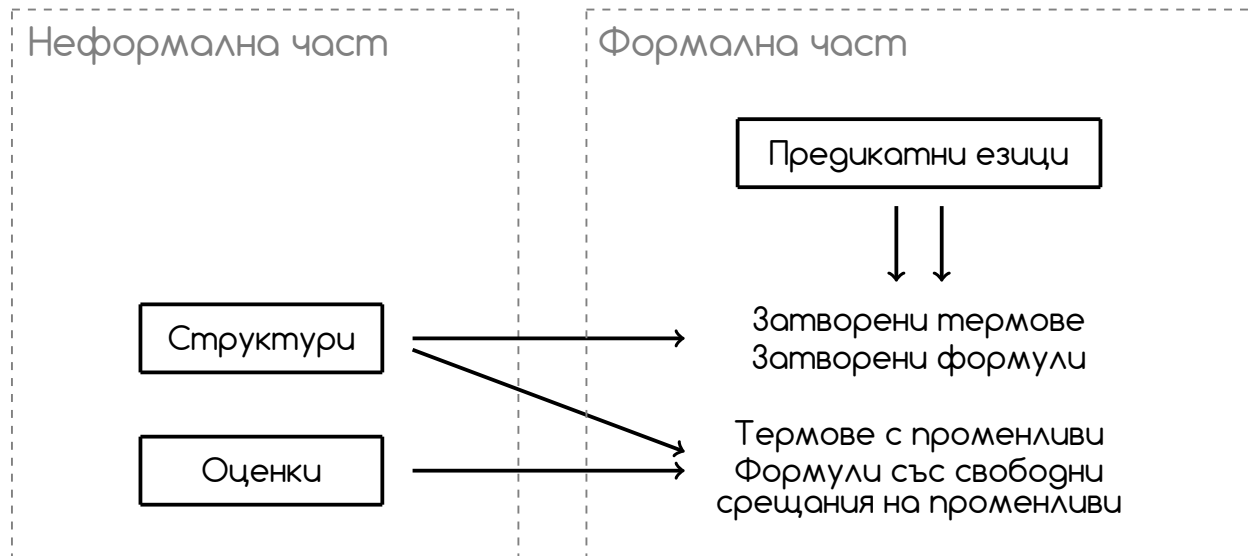
Затворена формула е такава формула, в която всички променливи се срещат само свързано, а свободни срещания на променливи няма.

Примери.

Следните изрази са затворени формули: $43 < \pi$, $\forall x(x > 0)$, $\forall x \exists y p(x, y)$, $\exists x(x^2 + 2x + 1 = 0)$, $\forall x(p(x) \ \& \ \exists y(q(x, y) \vee \forall z q(x, z)))$.

Следните изрази не са затворени формули: $x > 0$, $\exists y p(x, y)$, $f(x, f(y, z)) = f(f(x, y), z)$.

Основните понятия от този раздел могат да се подредят в следната схема



Дефиниция 12 (Литерал).

Литерал наричаме формула, която е или атомарна или отрицание на атомарна формула. Ако литералът е атомарна формула тогава ще казваме, че той е положителен. В противен случай литералът е отрицателен.

Примери.

Следните формули са литерали: $p(x)$, $\neg q(x, y, f(z))$, $q(f(h(x), f(c, d)), h(f(x, x)), t)$, $\neg p(c)$.

А тези формули са по-сложни и съответно не са литерали: $\forall p(x)$, $\neg\neg p(c)$, $q(x, y, f(z)) \vee p(z)$, $\neg(p(x) \& q(c, t))$, $\forall x(p(z) \iff \exists y \exists z(q(f(y), c) \& \neg p(z)))$.

Дефиниция 13 (Безкванторна форма).

За една формула казваме, че е безкванторна ако в нея не се срещат никакви квантори.

Дефиниция 14 (Пренексна нормална форма).

За една формула казваме, че е в пренексна нормална форма ако е от вида

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi,$$

където Q_i е квантор за всяко i , всички променливи във формулата са измежду x_1, x_2, \dots, x_n , а в φ е безкванторна.

Примери.

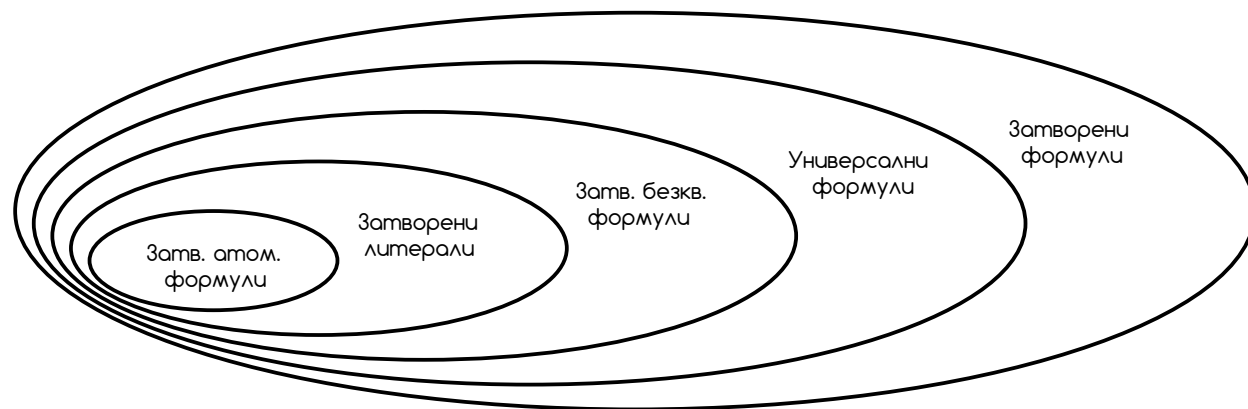
Тези формули са в пренексна нормална форма: $p(c)$, $\forall x p(x)$, $\forall x \forall y \forall z (p(x, y) \& p(y, z) \implies p(x, z))$ и $\forall x \exists y \forall z (p(x) \iff (\neg q(x, y) \implies (p(z) \& q(z, z))))$.

Формулите $p(x)$, $\neg \forall x (p(c) \vee p(x))$ и $\forall x \exists y (p(x) \implies \exists z q(y, z))$ не са в пренексна нормална форма.

Дефиниция 15 (Универсална формула).

За една формула казваме, че е универсална ако тя е в пренексна нормална форма и всички квантори в нея са универсални.

Така, по предните дефиниции можем да формуираме следната йерархия от различни типове затворени формули.



Задача.

За всяка от изброените по-долу формули да се определи:

- ☛ Дали формулата е атомарна?
- ☛ Дали формулата е затворена?
- ☛ Дали формулата е безкванторна?
- ☛ Дали формулата е универсална?

Формулите са

- ☛ $p(x, f(c));$
- ☛ $p(d, f(c));$
- ☛ $p(x, f(c)) \& \neg q(z);$
- ☛ $p(x, f(c)) \& \neg \exists z q(z);$
- ☛ $\forall x \forall y p(x, y)$
- ☛ $\forall x (p(x, f(c)) \& \neg \exists z q(z));$

където p и q са предикатни символи, f е функционален символ, а c е константа.

❖ Език за програмиране Пролог ❖

Дефиниция 16 (Елементарна дизюнкция).

Елементарна дизюнкция ще наричаме дизюнкция на краен брой литерали.

Дефиниция 17 (Дизюнкт).

Дизюнкт ще наричаме крайно множество от литерали. Празното множество ще наричаме празен дизюнкт и ще бележим с \square .

Дизюнктите ще асоциираме с елементарни дизюнкции.

Примери.

$\{ p(x), q(c, f(y)), \neg p(f(c)) \}$ е дизюнкт, който съответства на елементарната дизюнкция $p(x) \vee q(c, f(y)) \vee \neg p(f(c))$.

Също $\{ p(x), q(c, f(y)), \neg p(f(c)) \}$ се получава и от $\neg p(f(c)) \vee p(x) \vee \neg p(f(c)) \vee q(c, f(y)) \vee p(x)$.

Дефиниция 18 (Клауза).

За целите на програмиране на езика Пролог, под клауза ще разбираме непразен дизюнкт. Ще използваме следните видове клаузи:

- Хорнова клауза ще наричаме клауза, която има най-много един положителен литерал;
- програмна клауза ще наричаме Хорнова клауза, която има точно един положителен литерал;
- правило ще наричаме програмна клауза, в която има отрицателни литерали.
- факт ще наричаме програмна клауза, в която няма отрицателни литерали.
- цел ще наричаме Хорнова клауза, която няма положителни литерали.

Дефиниция 19 (Програма на Пролог).

Програма ще наричаме множество от програмни клаузи.

Означение (Синтаксис на Пролог).

Ако имаме програма на Пролог, то програмните клаузи в нея и целите се записват така:

- ако клаузата е фактът $\{ p(\dots) \}$, то тогава я записваме така: $p(\dots)$;
- ако клаузата е правилото $\{ \neg p_1(\dots), \neg p_2(\dots), \dots, \neg p_n(\dots), p(\dots) \}$, то я записваме така: $p(\dots) :- p_1(\dots), p_2(\dots), \dots, p_n(\dots)$; $p(\dots)$ ще наричаме заключение или глава на правилото, а $p_1(\dots), p_2(\dots), \dots, p_n(\dots)$ ще наричаме предпоставка или тяло на правилото;
- ако клаузата е целта $\{ \neg p_1(\dots), \neg p_2(\dots), \dots, \neg p_n(\dots) \}$, се записва по следния начин: $?- p_1(\dots), p_2(\dots), \dots, p_n(\dots)$.

Всеки идентификатор в Пролог програма, който започва с малка буква, се счита за предикат, вградена операция или ключова дума. А всеки идентификатор, започващ с главна буква, се счита за променлива.

Пример (Основни предикати за списъци).

Ето реализация на някои базови предикати за работа със списъци:

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

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

Пример (Снабдяване при търговия).

По формулите от примера за снабдяване при търговия можем да напишем следната Пролог програма:

```
need(P, Q) :- sell(P, Q).
procure(P, Q) :- buy(P, Q1), produce(P, Q2), Q is Q1 + Q2.

sell(bread, 40).
produce(bread, 10).
buy(bread, 20).
```

Така за цели на програмата можем да зададем

```
?- need(P, Q1), procure(P, Q2), Q1 > Q2.
```

или

```
?- need(P, Q1), procure(P, Q2), Q1 * k < Q2.
```

Означение (Допълнителен синтаксис на Пролог).

Допълнително в предпоставките на правилата или в целите могат да се използват и логически операции отрицание (записва се с **not** на Пролог) и дизюнкция (записва се с **;**).

Също така, към синтаксиса на Пролог за реализиране на изразните средства от предикатните езици добавяме и следните допълнителни предикати и оператори:

- = - оператор за унифициране; извършва необходимите задавания на стойности на променливи за да се уеднаквят аргументите;
- [], [|] - оператори за конструиране на списъци;
- is - оператор за присвояване на аритметична стойност на променлива;
- := - предикат за сравнение на стойностите на аритметични изрази;
- +, -, *, / - стандартни аритметични операции;
- ** - оператор за степенуване;
- // - оператор за целочислено деление;
- mod - оператор за остатък при деление;
- abs - оператор за абсолютна стойност;
- >, <, >=, <= - стандартни предикати за аритметични сравнения.

❖ Определимост и неопределимост ❖

Дефиниция 20 (Определимост на множество).

Нека имаме структура $\mathcal{A} = (A, \dots)$ и множество $D \subseteq A^n$ за някое $n \geq 1$. Тогава казваме, че множеството D е определимо с формула φ , ако са изпълнени следните две условия:

- ☛ във φ има точно n променливи, които се срещат свободно - x_1, x_2, \dots, x_n ; всички останали променливи във φ трябва задължително да се срещат само свързано;
- ☛ $\langle a_1, a_2, \dots, a_n \rangle \in D \iff \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi$ за някоя оценка v .

В такива случаи, формулата ще записваме така: $\varphi(x_1, x_2, \dots, x_n)$.

Твърдение 1 (Свойства на определимостта).

Следните свойства за изпълнени за всяка структура $\mathcal{A} = (A, \dots)$:

- ☛ ако $D \subseteq A^n$ е определимо множество, то тогава и $A^n \setminus D$ е определимо множество;
- ☛ ако $D_1 \subseteq A^n$ и $D_2 \subseteq A^n$ са определими множества, то тогава и $D_1 \cap D_2$ и $D_1 \cup D_2$ също са такива.

Доказателство.

Ако D се определя с формула $\varphi(x_1, x_2, \dots, x_n)$, то тогава $A^n \setminus D$ се определя с $\neg\varphi(x_1, x_2, \dots, x_n)$.

$$\begin{aligned} \langle a_1, a_2, \dots, a_n \rangle \in A^n \setminus D &\iff \langle a_1, a_2, \dots, a_n \rangle \notin D &\iff \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \not\models \varphi &\text{ за някоя оценка } v &\iff \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \neg\varphi &\text{ за някоя оценка } v. \end{aligned}$$

Съответно, ако D_1 и D_2 се определят с формули $\varphi_1(x_1, x_2, \dots, x_n)$ и $\varphi_2(x_1, x_2, \dots, x_n)$, които имат едни и същи променливи със свободни срещания, тогава $D_1 \cap D_2$ се определя с $\varphi_1 \& \varphi_2$, а $D_1 \cup D_2$ с $\varphi_1 \vee \varphi_2$.

$$\begin{aligned} \langle a_1, a_2, \dots, a_n \rangle \in D_1 \cap D_2 &\iff \langle a_1, a_2, \dots, a_n \rangle \in D_1 \text{ и } \langle a_1, a_2, \dots, a_n \rangle \in D_2 &\iff \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi_1 &\text{ за някоя оценка } v \text{ и} \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi_2 &\text{ за някоя оценка } v &\iff \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi_1 \& \varphi_2 &\text{ за някоя оценка } v. \end{aligned}$$

$$\begin{aligned} \langle a_1, a_2, \dots, a_n \rangle \in D_1 \cup D_2 &\iff \langle a_1, a_2, \dots, a_n \rangle \in D_1 \text{ или } \langle a_1, a_2, \dots, a_n \rangle \in D_2 &\iff \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi_1 &\text{ за произволна оценка } v \text{ или} \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi_2 &\text{ за произволна оценка } v &\iff \\ \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi_1 \vee \varphi_2 &\text{ за произволна оценка } v. \end{aligned}$$



Така, като се използва дефиницията за определимост на множество, могат да се дадат следните три дефиниции за по-специфични понятия:

Дефиниция 21 (Определимост на елемент).

Нека имаме структура $\mathcal{A} = (A, \dots)$ и елемент $a \in A$. Тогава казваме, че елементът a е определен в структурата \mathcal{A} тогава и само тогава, когато множеството $\{a\}$ е определимо в тази структура.

Дефиниция 22 (Определимост на свойство).

Нека имаме структура $\mathcal{A} = (A, \dots)$ и n -местно свойство p над A (т.е. $p : A^n \rightarrow \{T, F\}$). Тогава казваме, че свойството p е определимо в структурата \mathcal{A} тогава и само тогава, когато множеството

$$\{ \langle a_1, a_2, \dots, a_n \rangle \mid a_i \in A \text{ и стойността на } p(a_1, a_2, \dots, a_n) \text{ е } T \}$$

е определимо в тази структура.

Дефиниция 23 (Определимост на функция).

Нека имаме структура $\mathcal{A} = (A, \dots)$ и n -местна функция f над A (т.е. $f : A^n \rightarrow A$). Тогава казваме, че функцията f е определима в структурата \mathcal{A} тогава и само тогава, когато множеството

$$\{ \langle a_1, a_2, \dots, a_n, b \rangle \mid a_i \in A \text{ и } f(a_1, a_2, \dots, a_n) = b \}$$

е определимо в тази структура.

Пример.

Нека имаме структурата $\mathcal{N} = (\mathbb{N}, =, \cdot)$, където $=$ и \cdot са интерпретирани стандартно. Тогава

- 1 е определен елемент с формулата $\varphi_1(x) \equiv \forall y(x \cdot y = y)$;
- n/m е определима релация с формулата $\varphi_f(x, y) \equiv \exists z(x \cdot z = y)$;
- множеството на простите числа е определимо със следната формула $\varphi_{pr}(x) \equiv \forall y(\varphi_f(y, x) \implies (\varphi_1(y) \vee y = x))$.

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 преименувания на променливи). Например като пишем формулата $\forall y(\varphi_f(y, x) \implies (\varphi_1(y) \vee y = x))$, всъщност се има пред вид $\forall y(\exists z(y \cdot z = x) \implies (\forall z(y \cdot z = z) \vee y = x))$. Нямаме право, обаче, да използваме означения за обекти, които не са част от езика и още не са определен чрез формули!

Пример (Определимост на операции и свойства за множества).

Когато променливите се оценяват с множества, а езика се състои само от предикат за принадлежност \in , тогава имаме:

- $x \subseteq y$ е определима с $\varphi_{\subseteq}(x, y) \equiv \forall z(z \in x \implies z \in y)$;

- ☛ $x = y$ е определима с $\varphi_=(x, y) \Rightarrow \forall z(z \in x \iff z \in y)$ или $\varphi_=(x, y) \Rightarrow \varphi_\subseteq(x, y) \& \varphi_\subseteq(y, x)$;
- ☛ $x \cap y = z$ е определима с $\varphi_\cap(x, y, z) \Rightarrow \forall t(t \in z \iff t \in x \& t \in y)$;
- ☛ $x \cup y = z$ е определима с $\varphi_\cup(x, y, z) \Rightarrow \forall t(t \in z \iff t \in x \vee t \in y)$;
- ☛ $x \setminus y = z$ е определима с $\varphi_\setminus(x, y, z) \Rightarrow \forall t(t \in z \iff t \in x \& \neg(t \in y))$;
- ☛ \emptyset е определимо с $\varphi_\emptyset(x) \Rightarrow \neg \exists y(y \in x)$.

Дефиниция 24 (Изоморфизъм на структура).

Нека имаме структура $A = (A, \dots)$ и биекция h над A (т.е. $h : A \rightarrow A$ е инективна и сюрективна функция). Тогава h наричаме изоморфизъм над A ако изпълнява следните условия:

- ☛ за всяка константа c от езика на структурата е вярно, че: $h(c^A) = c^A$;
- ☛ за всеки n -местен функционален символ f от езика на структурата и за всеки $n + 1$ елемента $a_1, a_2, \dots, a_n, b \in A$ е вярно, че:

$$f^A(a_1, a_2, \dots, a_n) = b \iff f^A(h(a_1), h(a_2), \dots, h(a_n)) = h(b);$$

- ☛ за всеки n -местен предикатен символ p от езика на структурата и за всеки n елемента $a_1, a_2, \dots, a_n \in A$ е вярно, че:

$$p^A(a_1, a_2, \dots, a_n) \in T \iff p^A(h(a_1), h(a_2), \dots, h(a_n)) \in T.$$

Пример.

Идентитетът $h(x) = x$ е автоморфизъм над всяка структура.

Пример.

Нека имаме структурата $\mathcal{N} = (\mathbb{N}, =, \cdot)$, където $=$ и \cdot са интерпретирани стандартно.

Всяко естествено 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 -тото и j -тото прости числа от каноничния запис на n . За да имаме тоталност, полагаме $0^{i \leftrightarrow j} = 0$.

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

Ще докажем, че $h(n) = n^{i \leftrightarrow j}$ е автоморфизъм за \mathcal{N} за произволни i и j .

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).

Т.к. h е биекция, следователно $x = y \iff h(x) = h(y)$.

Остава да покажем, че h запазва действието и на $.$ за всеки $n \neq 0$ и $l \neq 0$. Т.е. че $n.l = k \iff h(n).h(l) = h(k)$.

Първо ще покажем че $h(n).h(l) = h(n.l)$.

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

Така чрез $h(n).h(l) = h(n.l)$ и факта, че h е биекция, можем да докажем, че $n.l = k$ 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 $.$, е тривиална.

Означение.

Нека A е множество, $D \subseteq A^n$ и h е функция над A (т.е. $h : A \rightarrow A$). Тогава с $h(D)$ означаваме следното множество:

$$h(D) = \{ \langle h(a_1), h(a_2), \dots, h(a_n) \rangle \mid \langle a_1, a_2, \dots, a_n \rangle \in D \}.$$

Твърдение 2 (Запазване на определените множества при изоморфизъм).

Нека $\mathcal{A} = (A, \dots)$ е структура и h е изоморфизъм над нея. Нека множество $D \subseteq A^n$ е определено в \mathcal{A} . Тогава е вярно, че $D = h(D)$.

Доказателство.

Т.к. D е определено, следователно по дефиниция има формула $\varphi(x_1, x_2, \dots, x_n)$ и оценка v при \mathcal{A} такива, че:

$$\langle a_1, a_2, \dots, a_n \rangle \in D \iff \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi.$$

За да получим, че $D = h(D)$ е достатъчно да покажем, че за всеки $a_1, a_2, \dots, a_n \in A$ е изпълнено

$$\mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi \iff \mathcal{A}, v[x_1 = h(a_1), x_2 = h(a_2), \dots, x_n = h(a_n)] \models \varphi.$$

Чрез този резултат можем да докажем, че

$$\begin{aligned}
 \langle a_1, a_2, \dots, a_n \rangle \in D &\iff \mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi &\iff \\
 \mathcal{A}, v[x_1 = h(a_1), x_2 = h(a_2), \dots, x_n = h(a_n)] \models \varphi &\iff \langle h(a_1), h(a_2), \dots, h(a_n) \rangle \in D
 \end{aligned}$$

от което следва, че $D = h(D)$.

От тук нататък, за краткост ще означаваме $v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n]$ с v_1 , а $v[x_1 = h(a_1), x_2 = h(a_2), \dots, x_n = h(a_n)]$ с v_2 . Така, трябва да покажем, че

$$\mathcal{A}, v_1 \models \varphi \quad \longleftrightarrow \quad \mathcal{A}, v_2 \models \varphi.$$

За целта първо ще проверим, че за всеки терм τ , чиито променливи са измежду x_1, x_2, \dots, x_n и за всеки $a_1, a_2, \dots, a_n \in A$ е изпълнено $h(v_1(\tau)) = v_2(\tau)$. Това ще направим с индукция по построението на τ :

- ☛ ако τ е индивидуалната променлива x_i (за някое i), тогава $h(v_1(x_i)) = h(a_i) = v_2(x_i)$;
- ☛ ако τ е константата c , тогава $h(v_1(c)) = h(c^A) = c^A = v_2(c)$;
- ☛ ако τ е израз $f(\tau_1, \tau_2, \dots, \tau_n)$, тогава $h(v_1(f(\tau_1, \tau_2, \dots, \tau_n))) = h(f^A(v_1(\tau_1), v_1(\tau_2), \dots, v_1(\tau_n))) = f^A(h(v_1(\tau_1)), h(v_1(\tau_2)), \dots, h(v_1(\tau_n))) = f^A(v_2(\tau_1), v_2(\tau_2), \dots, v_2(\tau_n)) = v_2(f(\tau_1, \tau_2, \dots, \tau_n))$.

Сега ще довършим доказателството, че

$$\mathcal{A}, v_1 \models \varphi \quad \longleftrightarrow \quad \mathcal{A}, v_2 \models \varphi.$$

Това ще направим с индукция по построението на φ :

- ☛ ако φ е атомарната формула $p(\tau_1, \tau_2, \dots, \tau_k)$ тогава
$$\begin{aligned} \mathcal{A}, v_1 \models p(\tau_1, \tau_2, \dots, \tau_k) &\longleftrightarrow \text{стойността на } p^A(v_1(\tau_1), v_1(\tau_2), \dots, v_1(\tau_k)) \text{ е } T \longleftrightarrow \\ &\text{стойността на } p^A(h(v_1(\tau_1)), h(v_1(\tau_2)), \dots, h(v_1(\tau_k))) \text{ е } T \longleftrightarrow \\ &\text{стойността на } p^A(v_2(\tau_1), v_2(\tau_2), \dots, v_2(\tau_k)) \text{ е } T \longleftrightarrow \mathcal{A}, v_2 \models p(\tau_1, \tau_2, \dots, \tau_k); \end{aligned}$$
- ☛ ако φ е $\neg\alpha$ тогава
$$\mathcal{A}, v_1 \models \varphi \quad \longleftrightarrow \quad \mathcal{A}, v_1 \not\models \alpha \quad \longleftrightarrow \quad \mathcal{A}, v_2 \not\models \alpha \quad \longleftrightarrow \quad \mathcal{A}, v_2 \models \varphi;$$
- ☛ ако φ е $(\alpha \& \beta)$ тогава
$$\begin{aligned} \mathcal{A}, v_1 \models \varphi &\longleftrightarrow \mathcal{A}, v_1 \models \alpha \text{ и } \mathcal{A}, v_1 \models \beta \quad \longleftrightarrow \\ \mathcal{A}, v_2 \models \alpha \text{ и } \mathcal{A}, v_2 \models \beta &\longleftrightarrow \mathcal{A}, v_2 \models \varphi; \end{aligned}$$
- ☛ ако φ е $(\alpha \vee \beta)$ тогава
$$\begin{aligned} \mathcal{A}, v_1 \models \varphi &\longleftrightarrow \mathcal{A}, v_1 \models \alpha \text{ или } \mathcal{A}, v_1 \models \beta \quad \longleftrightarrow \\ \mathcal{A}, v_2 \models \alpha \text{ или } \mathcal{A}, v_2 \models \beta &\longleftrightarrow \mathcal{A}, v_2 \models \varphi; \end{aligned}$$
- ☛ ако φ е $(\alpha \implies \beta)$ тогава
$$\begin{aligned} \mathcal{A}, v_1 \models \varphi &\longleftrightarrow \mathcal{A}, v_1 \not\models \alpha \text{ или } \mathcal{A}, v_1 \models \beta \quad \longleftrightarrow \\ \mathcal{A}, v_2 \not\models \alpha \text{ или } \mathcal{A}, v_2 \models \beta &\longleftrightarrow \mathcal{A}, v_2 \models \varphi; \end{aligned}$$
- ☛ ако φ е $(\alpha \iff \beta)$ тогава
$$\begin{aligned} \mathcal{A}, v_1 \models \varphi &\longleftrightarrow \mathcal{A}, v_1 \models \alpha \text{ и } \mathcal{A}, v_1 \models \beta \text{ или } \mathcal{A}, v_1 \not\models \alpha \text{ и } \mathcal{A}, v_1 \not\models \beta \quad \longleftrightarrow \\ \mathcal{A}, v_2 \models \alpha \text{ и } \mathcal{A}, v_2 \models \beta \text{ или } \mathcal{A}, v_2 \not\models \alpha \text{ и } \mathcal{A}, v_2 \not\models \beta &\longleftrightarrow \mathcal{A}, v_2 \models \varphi; \end{aligned}$$

☛ ако φ е $\forall x\alpha$ тогава, т.к. x не е измежду x_1, x_2, \dots, x_n

$$\begin{aligned} \mathcal{A}, v_1 \models \varphi &\longleftrightarrow \mathcal{A}, v_1[x = a] \models \alpha \text{ за всяко едно } a \in A &\longleftrightarrow \\ \mathcal{A}, v_2[x = a] \models \alpha \text{ за всяко едно } a \in A &\longleftrightarrow \mathcal{A}, v_2 \models \varphi; \end{aligned}$$

☛ ако φ е $\exists x\alpha$ тогава, т.к. x не е измежду x_1, x_2, \dots, x_n

$$\begin{aligned} \mathcal{A}, v_1 \models \varphi &\longleftrightarrow \mathcal{A}, v_1[x = a] \models \alpha \text{ за поне едно } a \in A &\longleftrightarrow \\ \mathcal{A}, v_2[x = a] \models \alpha \text{ за поне едно } a \in A &\longleftrightarrow \mathcal{A}, v_2 \models \varphi. \end{aligned}$$



От предното твърдение получаваме директното следствие

Следствие 1 (Критерий за неопределимост на множества).

Нека $\mathcal{A} = (A, \dots)$ е структура и $D \subseteq A^n$. Тогава, ако има изоморфизъм h над \mathcal{A} такъв, че $D \neq h(D)$, то като следствие имаме, че множеството D не е определимо в \mathcal{A} .

А като се приложи горния критерий в случаите за определимост на елемент, свойство или функция се получават следните по-специфични критерии:

Следствие 2 (Критерий за неопределимост на елемент).

Нека $\mathcal{A} = (A, \dots)$ е структура и $a \in A$. Тогава, ако има изоморфизъм h над \mathcal{A} такъв, че $a \neq h(a)$, то като следствие имаме, че елементът a не е определен в \mathcal{A} .

Следствие 3 (Критерий за неопределимост на свойство).

Нека $\mathcal{A} = (A, \dots)$ е структура и p е n -местно свойство над A . Тогава, ако има изоморфизъм h над \mathcal{A} и $a_1, a_2, \dots, a_n \in A$ такива, че поне един от следните два случая е в сила

- ☛ или стойността на $p(a_1, a_2, \dots, a_n)$ е T , но стойността на $p(h(a_1), h(a_2), \dots, h(a_n))$ е F ;
- ☛ или стойността на $p(a_1, a_2, \dots, a_n)$ е F , но стойността на $p(h(a_1), h(a_2), \dots, h(a_n))$ е T .

то тогава свойството p не е определимо в \mathcal{A} .

Следствие 4 (Критерий за неопределимост на функция).

Нека $\mathcal{A} = (A, \dots)$ е структура и f е n -местна функция над A . Тогава, ако има изоморфизъм h над \mathcal{A} и $a_1, a_2, \dots, a_n, b \in A$ такива, че поне един от следните два случая е в сила

- ☛ или $f(a_1, a_2, \dots, a_n) = b$, но $f(h(a_1), h(a_2), \dots, h(a_n)) \neq h(b)$;
- ☛ или $f(a_1, a_2, \dots, a_n) \neq b$, но $f(h(a_1), h(a_2), \dots, h(a_n)) = h(b)$.

то тогава функцията f не е определима в \mathcal{A} .

Пример.

Нека имаме структурата $\mathcal{N} = (\mathbb{N}, =, \cdot)$, където $=$ и \cdot са интерпретирани стандартно.

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.

Така, можем да покажем, че всяко $n > 1$ е неопределимо в тази структура.

Примери.

Ще разгледаме въпросите за определимост в следните 12 структури - (\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:

$$\begin{aligned} s(x, y, z) &\longleftrightarrow x + y = z \\ m(x, y, z) &\longleftrightarrow x \cdot y = z \end{aligned}$$

В (\mathbb{R}, s, m) са определени всички рационални числа:

$$\begin{aligned} \varphi_0(x) &\Rightarrow \forall y \, s(x, y, y) \\ \varphi_0(x) &\Rightarrow \forall y \, m(x, y, x) \\ \varphi_1(x) &\Rightarrow \forall y \, m(x, y, y) \\ \varphi_n(x) &\Rightarrow \exists y \exists z (\varphi_1(y) \& \varphi_{n-1}(z) \& s(y, z, x)) \\ \varphi_{-n}(x) &\Rightarrow \exists y \exists z (\varphi_0(y) \& \varphi_n(z) \& s(x, z, y)) \\ \varphi_q^p(x) &\Rightarrow \exists y \exists z (\varphi_p(y) \& \varphi_q(z) \& \neg \varphi_0(z) \& m(z, x, y)) \end{aligned}$$

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

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

Следните релации също са определени

$$\begin{aligned} \varphi_=(x, y) &\Rightarrow \exists z (\varphi_0(z) \& s(x, z, y)) \\ \varphi_<(x, y) &\Rightarrow \exists z \exists t (\neg \varphi_0(t) \& m(t, t, z) \& s(x, z, y)) \end{aligned}$$

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

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

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

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

Алтернативно, 0 може да се определи чрез <

$$\varphi_0(x) \equiv \neg \exists y \varphi_{<}(y, x)$$

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 определен, 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).

В структура (\mathbb{N}, m) са определими 0 и 1 (за 0 се използва формулата с m от (\mathbb{R}, s, m)). За показване на неопределимостта на останалите числа се използват автоморфизмите $h(n) = n^{i \leftrightarrow j}$.

Равенството също е определимо чрез m :

$$\varphi_{=}(x, y) \equiv \exists z (\varphi_1(z) \& m(x, z, y))$$

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

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

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.

При структура (\mathbb{Z}, m) са определими са 0, 1 и -1 . За 0 и 1 се използват формулите от (\mathbb{N}, m) , а за -1 формулата

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

За неопределимостта на останалите елементи използваме следния автоморфизъм:

$$h(x) = \text{sign}(x) \cdot |x|^{i \leftrightarrow j}$$

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

Нека разгледаме структура (\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 $k \neq 1$.

При структура (\mathbb{Q}, m) , определим 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 $h(\frac{p}{q}) = \text{sign}(\frac{p}{q}) \cdot \frac{|p|^{i \leftrightarrow j}}{|q|}$. 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 $h(\frac{p}{q}) = \text{sign}(\frac{p}{q}) \cdot \frac{|p|}{|q|^{i \leftrightarrow j}}$.

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

За (\mathbb{R}, s) положението е както при (\mathbb{Z}, s) и (\mathbb{Q}, s) .

В структура (\mathbb{R}, m) са определени $0, 1$ 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 k е нечетно естествено число (например $h(x) = x^3$ или $h(x) = \sqrt[3]{x}$). Също може да се използва $h(x) = \text{sign}(x) \cdot \sqrt{|x|}$.

Пример.

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 - $x \sqsubseteq y$: $\varphi_{\sqsubseteq}(x, y) \equiv \forall z(p(x, z) \implies p(y, z))$;
- ☛ x съвпада с y : $\varphi_{=}(x, y) \equiv \varphi_{\sqsubseteq}(x, y) \& \varphi_{\sqsubseteq}(y, x)$.

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:
 $\varphi_{\text{point}}(x, y) \equiv p(x, y) \& \neg \exists z(\varphi_{\sqsubseteq}(z, x) \& \varphi_{\sqsubseteq}(z, y))$.

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

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

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

Задача.

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

$$\begin{aligned} c^S &= 1, \\ f^S(n, m) &= n + m + 1, \\ r^S(n, m) &\longleftrightarrow n = m. \end{aligned}$$

Да се определят елементите $1, 0, 3$ и 2 .

Решение.

$$\begin{aligned}\varphi_1(x) &\Leftarrow r(x, c), \\ \varphi_0(x) &\Leftarrow r(f(x, x), c), \\ \varphi_3(x) &\Leftarrow r(f(c, c), x), \\ \varphi_2(x) &\Leftarrow \exists y(\varphi_0(y) \& r(f(y, c), x)).\end{aligned}$$



Задача.

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 от предната задача. Да се определят елементите 1 и 0.

Решение.

$$\begin{aligned}\varphi_1(x) &\Leftarrow r(x, c), \\ \varphi_0(x) &\Leftarrow r(x, g(x)) \& \neg \varphi_1(x).\end{aligned}$$



Задача.

Дадена е структурата $S'' = (\mathbb{N}, c, r)$, където интерпретацията на c и r е както от предните задачи. Да се докаже, че елементите 2 и 3 не са определими в S'' . Да се докаже и че елементите 2 и 3 не са определими също и в структурата S' от предната задача.

Решение.

За структура S'' се използва автоморфизма

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

А за S' се използва $h(x) = x^{0 \leftrightarrow 1}$.



Задача.

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

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

Да се определят елементите 1, 3, 5 и 7 и множеството

$$B = \{ n \mid n \text{ е нечетно число, което не се дели на } 3 \}$$

Решение.

$$\begin{aligned}\varphi_1(x) &\equiv \forall y s(y, x, y), \\ \varphi_3(x) &\equiv \exists y (\varphi_1(y) \& r(y, y, x)), \\ \varphi_5(x) &\equiv \exists y \exists z (\varphi_1(y) \& \varphi_3(z) \& r(y, z, x)), \\ \varphi_7(x) &\equiv \exists y \exists z (\varphi_1(y) \& \varphi_5(z) \& r(y, z, x)), \\ \varphi_B(x) &\equiv \neg \exists z \exists t (\varphi_3(z) \& s(z, t, x)).\end{aligned}$$



Задача.

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 определен, тогава и само тогава, когато $m \in \{0, 1\}$.

Решение.

$$\begin{aligned}\varphi_0(x) &\equiv \forall y s(y, x, x), \\ \varphi_1(x) &\equiv p(x, x, x) \& \neg \varphi_0(x).\end{aligned}$$

Това са формулите, които определят 0 и 1.

Неопределимостта на останалите елементи се показва с автоморфизмите $h(n) = n^{i \leftrightarrow j}$.



Задача.

Дадена е структурата $\mathcal{A} = (\mathbb{N}, \text{between})$ като

$$\text{between}(x, y, z) \longleftrightarrow x \leq z \& z \leq y.$$

Да се определят:

- ☛ стандартните аритметични предикати за $<$ и $>$;
- ☛ всички елементи на структурата - 0, 1, 2 и т.н.
- ☛ предикати $p_0(x, y)$, $p_1(x, y)$, $p_2(x, y)$ и т.н., където:

$$p_k(x, y) \longleftrightarrow x = y + k.$$

Задача.

Дадена е структурата $\mathcal{A} = (\mathbb{N}, <, >, =, \text{pow})$, където $<$, $>$ и $=$ са стандартните аритметични предикати за сравнения, а pow е следната функция:

$$\text{pow}(x, y) = z \longleftrightarrow x^y = z.$$

Да се определят:

☛ всички елементи на структурата - 0, 1, 2 и т.н.;

☛ предикат $between(x, y, z)$ такъв, че:

$$between(x, y, z) \longleftrightarrow x \leq z \ \& \ z \leq y;$$

☛ предикат $del(x, y)$ такъв, че y е делител на x ;

☛ множества на четните, на нечетните и на простите числа;

☛ стандартните аритметични операции $+$ и $*$.

☛ предикат $r(a, b, c)$ такъв, че уравнението $ax^2 + bx + c = 0$ има два различни корена;

☛ предикат $p(x)$ такъв, че $x = a^3 + b^3 + c^3$ за някои $a, b, c \in \mathbb{N}$.

❖ Логически задачи ❖

Дефиниция 25 (Изпълнимост).

Казваме, че едно непразно множество от формули $\{ \varphi_1, \varphi_2, \varphi_3, \dots \}$ е изпълнимо ако има структура \mathcal{A} и оценка при тази структура v такива, че:

$$\mathcal{A}, v \models \varphi_i \text{ за всеки индекс } i.$$

Заб.: Множеството може да е както крайно, така и безкрайно.

Съответно множество $\{ \varphi_1, \varphi_2, \varphi_3, \dots \}$ е неизпълнимо ако за всяка структура \mathcal{A} и всяка оценка v

$$\mathcal{A}, v \not\models \varphi_i \text{ за поне един индекс } i.$$

Пример.

Следното безкрайно множество от формули е изпълнимо: $\forall x \exists y (x < y), 0 < 1, 1 < 2, 2 < 3, \dots, 2015 < 2016, \dots$

Пример (Съвместяване на програми).

Дали формулите от примера за съвместяване на програми са изпълнимо множество:

- ☛ $\text{traffic}(\text{internet}) \vee \text{traffic}(\text{intranet})$
- ☛ $\neg(\text{traffic}(\text{internet}) \& \text{traffic}(\text{intranet}))$
- ☛ $\text{traffic}(\text{internet}) \implies \neg \text{start}(\text{scanner}) \& \text{start}(\text{filter})$
- ☛ $\text{traffic}(\text{intranet}) \implies \text{start}(\text{scanner})$
- ☛ $\text{traffic}(\text{internet}) \implies \text{start}(\text{scanner}) \& \neg \text{start}(\text{filter})$
- ☛ $\text{traffic}(\text{intranet}) \implies \neg \text{start}(\text{scanner}) \& \text{start}(\text{filter})$

Дефиниция 26 (Противоречивост и непротиворечивост).

Казваме, че едно непразно множество от формули е непротиворечиво, ако то е изпълнимо. Съответно, множеството е противоречиво, ако то е неизпълнимо.

Дефиниция 27 (Тъждествена вярност).

Казваме, че една формула φ е тъждествено вярна ако за всяка възможна структура \mathcal{A} и за всяка възможна оценка при тази структура v е изпълнено, че:

$$\mathcal{A}, v \models \varphi.$$

Примери.

Тъждествено вярните формули обикновено са логически закони като

$$\forall x (p(x) \vee \neg p(x))$$

или отразяват еквивалентни преобразувания като

$$\begin{aligned}
&(((\alpha \implies \beta) \& (\beta \implies \gamma)) \implies (\alpha \implies \gamma)) \\
&((\alpha \iff \beta) \iff ((\alpha \implies \beta) \& (\beta \implies \alpha))) \\
&\forall x \varphi \iff \neg \exists x \neg \varphi
\end{aligned}$$

Дефиниция 28 (Логическо следване).

Казваме, че една формула φ логически следва от множество от формули $\{\varphi_1, \varphi_2, \varphi_3, \dots\}$, ако за всяка възможна структура \mathcal{A} и за всяка възможна оценка при тази структура v е изпълнено, че:

$$\mathcal{A}, v \models \varphi_i \text{ за всеки индекс } i \quad \longrightarrow \quad \mathcal{A}, v \models \varphi.$$

Това означаваме с $\varphi_1, \varphi_2, \varphi_3, \dots \models \varphi$.

Заб.: Множеството може да е както крайно, така и безкрайно. Също може и да е празно.

Примери.

От твърдения "Всички хора са смъртни", "Сократ е човек" следва "Сократ е смъртен". Тези твърдения се записват с формули:

$$\forall x(\text{human}(x) \implies \text{mortal}(x)), \text{human}(\text{Socrates}) \models \text{mortal}(\text{Socrates})$$

Това следване също е в сила:

$$\forall x \forall y \forall z (x < y \& y < z \implies x < z), 0 < 1, 1 < 2 \models 0 < 2$$

Ето и един пример, в който последната формула не следва от първите две:

$$\forall x \forall y ((x, y) = 1 \implies (y, x) = 1), (2, 3) = 1, (3, 4) = 1 \not\models (2, 4) = 1$$

Тук (x, y) обозначава най-големия общ делител на x и y .

Пример (Снабдяване при търговия).

Дали от следните формули (от примера за снабдяване при търговия)

- ☛ $\forall p \forall q (\text{sell}(p, q) \implies \text{need}(p, q))$
- ☛ $\forall p \forall q_1 \forall q_2 (\text{buy}(p, q_1) \& \text{produce}(p, q_2) \implies \text{procure}(p, q_1 + q_2))$
- ☛ $\text{sell}(\text{bread}, 40)$
- ☛ $\text{produce}(\text{bread}, 10)$
- ☛ $\text{buy}(\text{bread}, 20)$
- ☛ $40 > 20 + 10$

следва някоя от тези формули

- ☛ $\exists p \exists q_1 \exists q_2 (\text{need}(p, q_1) \& \text{procure}(p, q_2) \& q_1 > q_2)$

$$\bullet \quad \exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& k.q_1 < q_2)$$

Пример (Диагностика на пациенти).

Също логическото следване може да се използва за откриване на излишества в множество от формули $\{\varphi_1, \varphi_2, \dots, \varphi_n\}$. За целта трябва да проверим дали всяко едно от тези следвания е в сила

$$\begin{aligned} \varphi_2, \varphi_3, \dots, \varphi_n &\models \varphi_1 \\ \varphi_1, \varphi_3, \dots, \varphi_n &\models \varphi_2 \\ &\vdots \\ \varphi_1, \varphi_2, \dots, \varphi_{n-1} &\models \varphi_n \end{aligned}$$

и ако някое от тях е изпълнено, да премахнем излишните формули.

Така, този метод можем да приложим за да проверим, дали измежду формулите от примера за диагностика на пациенти

- $\forall p (\neg cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t \leq 37) \implies allergy(p))$
- $\forall p (\neg cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t > 37 \& t \leq 40) \implies flu(p))$
- $\forall p (cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t > 37 \& t \leq 40) \implies pneumonia(p))$
- $\forall p (headache(p) \& \exists t (temperature(p, t) \& t \leq 37) \implies fatigue(p))$
- $\forall p (headache(p) \& \exists t (temperature(p, t) \& t > 40) \implies meningitis(p))$
- $\forall p (\neg cough(p) \& snuffles(p) \& \exists t (temperature(p, t) \& t \geq 38 \& t < 39) \implies flu(p))$

има излишество.

Твърдение 3 (Свеждане между логическо следване и тъждествена вярност).
За произволна формула φ е изпълнено, че:

$$\emptyset \models \varphi \quad \longleftrightarrow \quad \varphi \text{ е тъждествено вярно.}$$

Доказателство.

По дефиниция имаме, че $\Phi \models \varphi$ тогава и само тогава, когато за всяка структура \mathcal{A} и оценка при нея v е изпълнено, че ако $\mathcal{A}, v \models \varphi_i$ за всяка формула $\varphi_i \in \Phi$, то тогава трябва $\mathcal{A}, v \models \varphi$.

В случая множеството Φ е празно, което означава, че $\emptyset \models \varphi$ е еквивалентно на $\mathcal{A}, v \models \varphi$ за всяка структура \mathcal{A} и оценка при нея v . Но това е точно дефиниционното условие, че φ е тъждествено вярно. ◆

Твърдение 4 (Свеждане между тъждествена вярност и неизпълнимост).
За произволна формула φ е изпълнено, че:

$$\varphi \text{ е тъждествено вярно} \quad \longleftrightarrow \quad \{\neg \varphi\} \text{ е неизпълнимо.}$$

Доказателство.

$$\begin{aligned}
 \varphi \text{ е тѣждествено вярна} & \longleftrightarrow \\
 \mathcal{A}, v \models \varphi \text{ за всяка структура } \mathcal{A} \text{ и оценка при нея } v & \longleftrightarrow \\
 \mathcal{A}, v \not\models \neg\varphi \text{ за всяка структура } \mathcal{A} \text{ и оценка при нея } v & \longleftrightarrow \\
 \{ \neg\varphi \} \text{ е неизпълнимо.}
 \end{aligned}$$



Пример.

Например $\forall x(p(x) \vee \neg p(x))$ е тѣждествено вярна формула, а $\{ \neg\forall x(p(x) \vee \neg p(x)) \}$ е неизпълнимо множество.

Заб.: Това може лесно да се види след като формулата $\neg\forall x(p(x) \vee \neg p(x))$ се преобразува еквивалентно на $\exists x(\neg p(x) \& p(x))$.

Твърдение 5 (Свеждане между логическо следване и неизпълнимост).

За всяко множество от формули $\{ \varphi_1, \varphi_2, \varphi_3, \dots \}$ и формула φ е изпълнено, че:

$$\varphi_1, \varphi_2, \varphi_3, \dots \models \varphi \longleftrightarrow \{ \varphi_1, \varphi_2, \varphi_3, \dots, \neg\varphi \} \text{ е неизпълнимо.}$$

Доказателство.

$$\begin{aligned}
 \varphi_1, \varphi_2, \varphi_3, \dots \models \varphi & \longleftrightarrow \\
 \text{за всеки } \mathcal{A} \text{ и } v, \text{ ако } \mathcal{A}, v \models \varphi_i \text{ за всички } i, \text{ то } \mathcal{A}, v \models \varphi & \longleftrightarrow \\
 \text{за всеки } \mathcal{A} \text{ и } v, \mathcal{A}, v \not\models \varphi_i \text{ за някое } i \text{ или } \mathcal{A}, v \models \varphi & \longleftrightarrow \\
 \text{за всеки } \mathcal{A} \text{ и } v, \mathcal{A}, v \not\models \varphi_i \text{ за някое } i \text{ или } \mathcal{A}, v \not\models \neg\varphi & \longleftrightarrow \\
 \text{за всеки } \mathcal{A} \text{ и } v, \mathcal{A}, v \not\models \psi \text{ за поне една } \psi \in \{ \varphi_1, \varphi_2, \varphi_3, \dots, \neg\varphi \} & \longleftrightarrow \\
 \{ \varphi_1, \varphi_2, \varphi_3, \dots, \neg\varphi \} \text{ е неизпълнимо.}
 \end{aligned}$$



Пример.

$$\begin{aligned}
 \forall x \forall y \forall z (x < y \& y < z \implies x < z), 0 < 1, 1 < 2 \models 0 < 2 & \longleftrightarrow \\
 \longleftrightarrow \{ \forall x \forall y \forall z (x < y \& y < z \implies x < z), 0 < 1, 1 < 2, \neg(0 < 2) \} \text{ е неизпълнимо.}
 \end{aligned}$$

Пример (Снабдяване при търговия).

В примера за снабдяване при търговия, двете евентуални следствия

$$\bullet \quad \exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& q_1 > q_2)$$

$$\bullet \quad \exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& k.q_1 < q_2)$$

могат след добавянето на отрицанието да се преобразуват съответно по този начин

$$\bullet \quad \forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(q_1 > q_2))$$

$$\bullet \quad \forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(k.q_1 < q_2))$$

Задача.

Да се докаже, че една затворена формула е изпълнима в дадена структура тогава и само тогава, когато тя е тъждествено вярна в тази структура.

Упътване: Да се използва факта, че това дали една затворена формула е вярна при дадена структура и оценка, всъщност не зависи от оценката.

Задача.

Да се докаже, че крайно множество от формули е изпълнимо тогава и само тогава, когато конюнкцията му е изпълнима формула.

Задача.

Дадени са следните пет формули:

$$\bullet \quad \forall x(p(x) \& q(f(x)) \implies r(x));$$

$$\bullet \quad p(c);$$

$$\bullet \quad \forall yq(y);$$

$$\bullet \quad r(c);$$

$$\bullet \quad \forall x r(x).$$

Възможно ли е да има структура, която изпълнява първите три формули, но не и четвъртата? А възможно ли е да има структура, която изпълнява първите три формули, но не и петата? Обосновете отговорите си!

Задача.

Дадени са следните пет формули:

$$\bullet \quad \forall x(q(x) \& p(g(x)) \implies r(x));$$

$$\bullet \quad q(d);$$

$$\bullet \quad \forall x p(x);$$

$$\bullet \quad \forall z r(z);$$

$$\bullet \quad r(d).$$

Възможно ли е да има структура, която изпълнява първите три формули, но не и четвъртата? А възможно ли е да има структура, която изпълнява първите три формули, но не и петата? Обосновете отговорите си!

❖ Интерпретиране на Пролог програми ❖

Дефиниция 29 (Универсално и екзистенциално затваряне на формула).

Ако φ е формула и x_1, x_2, \dots, x_n са всички променливи в нея, които се срещат свободно, тогава универсално затваряне на φ наричаме формулата $\forall x_1 \forall x_2 \dots \forall x_n \varphi$. Тази формула още ще бележим с $UC(\varphi)$.

А екзистенциално затваряне на φ съответно ще наричаме формулата $\exists x_1 \exists x_2 \dots \exists x_n \varphi$ и ще я бележим с $EC(\varphi)$.

Примери.

$\forall x \forall y (p(f(y)) \& \neg q(x))$ е универсалното затваряне на $p(f(y)) \& \neg q(x)$. А $\exists x \exists y (p(f(y)) \& \neg q(x))$ е екзистенциалното затваряне на същата формула.

$\forall z \forall t (\forall x (p(x, z) \implies \exists y q(t, y)))$ и $\exists z \exists t (\forall x (p(x, z) \implies \exists y q(t, y)))$ са универсалното и екзистенциалното затваряния на $\forall x (p(x, z) \implies \exists y q(t, y))$.

А $p(f(c)) \& \neg \forall x q(x)$ е и универсалното и екзистенциалното затваряния на $p(f(c)) \& \neg \forall x q(x)$.

Дефиниция 30 (Интерпретация на Хорнова клауза).

На всяка Хорнова клауза ще съпоставяме затворена формула по следния начин:

- ако клаузата е правило или факт, на нея съпоставяме универсалното затваряне на дизюнкцията от елементите на клаузата;
- ако клаузата е цел, на нея съпоставяме екзистенциалното затваряне на конюнкцията от елементите на клаузата (само че, се изпускат отрицанията).

Тази затворена формула ще се нарича интерпретация на клаузата.

Примери (Интерпретация на Хорнови клаузи).

Хорнова клауза	Пролог синтаксис	Затворена формула
$\{ p(x) \}$	$p(X).$	$\forall x p(x)$
$\{ p(x), \neg q(x, y), \neg p(y) \}$	$p(X) :- q(X, Y), p(Y).$	$\forall x \forall y (p(x) \vee \neg q(x, y) \vee \neg p(y))$
$\{ \neg p(x) \}$	$?- p(X).$	$\exists x p(x)$
$\{ \neg p(x), \neg q(x, y), \neg r(z) \}$	$?- p(X), q(X, Y), r(Z).$	$\exists x \exists y \exists z (p(x) \& q(x, y) \& r(z))$

Заб.: Идеята зад по-особеното интерпретиране на целите е, че след като затворената формула се отрече (това ще е нужно при проверката, дали целта следва от клаузите на програмата), се получава точно универсалното затваряне на Хорновата клауза, която представлява целта (рабира се, след известни еквивалентни преобразувания).

Дефиниция 31 (Удовлетворяване на цел при Пролог програма).

Ще казваме, че дадена цел се удовлетворява при дадена програма на Пролог ако

$$\varphi_1, \varphi_2, \dots, \varphi_n \models \varphi,$$

където $\varphi_1, \varphi_2, \dots, \varphi_n$ са интерпретациите на програмните клаузи, а φ е интерпретацията на целта.

Така, за проверка на удовлетворяването на цел при програма трябва да сведем задачата до проверка за неизпълнимост на множеството $\{\varphi_1, \varphi_2, \dots, \varphi_n, \neg\varphi\}$. Така, от всяка програмна клауза получаваме по един дизюнкт, а също и от отрицанието на целта получаваме дизюнкт.

Пример (Снабдяване при търговия).

По формулите от примера за снабдяване при търговия можем да напишем следната Пролог програма:

```
need(P, Q) :- sell(P, Q).  
procure(P, Q) :- buy(P, Q1), produce(P, Q2), Q is Q1 + Q2.  
  
sell(bread, 40).  
produce(bread, 10).  
buy(bread, 20).
```

Нека за цел на програмата е зададена

?- need(P, Q₁), procure(P, Q₂), Q₁ > Q₂.

Така, по програмата получаваме следните формули:

- ☛ $\forall p \forall q (sell(p, q) \implies need(p, q))$
- ☛ $\forall p \forall q_1 \forall q_2 (buy(p, q_1) \& produce(p, q_2) \implies procure(p, q_1 + q_2))$
- ☛ $sell(bread, 40)$
- ☛ $produce(bread, 10)$
- ☛ $buy(bread, 20)$

като от тях трябва да следва формулата, съответстваща на целта:

- ☛ $\exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& q_1 > q_2)$

❖ Изпълнение на Пролог програми чрез обратен извод ❖

Това е алтернативен метод за проверяване дали дадена цел се удовлетворява при дадена програма на Пролог. Този метод не разчита на класическата интерпретация на програмните клаузи и целта като формули от първи ред, ами по-скоро ги разглежда като рекурсивни оператори и се опитва посредством правилата да сведе (изпълни) целта до (чрез) фактите.

За да се отрази употребата на отрицание (и евентуално и на дизюнкция) в Пролог програми трябва да се разшири съответствието между клаузи и формули от Дефиниция 30.

Дефиниция 32 (Съпоставяне между Пролог клаузи и формули).

На целта се съпоставя формула по същия начин, както в Дефиниция 30 - целта се интерпретира като безкванторна формула и се взима екзистенциалното затваряне на тази формула, като се пропускат отрицанията. Също и за фактите се процедира без промяна - взима се универсалното затваряне на атомарната формула, която се представлява от факта.

При правилата, обаче, има разлика при образуването на формулата. Първо, тялото на клаузата се интерпретира като безкванторна формула и за всяка променлива, която се среща в тялото, но не и в главата, се поставя екзистенциален квантор. Областта на действие на квантора трябва да е минималната област, която включва всички срещания на променливата. Например, ако вички срещания на променливата x в областта на действие на някое отрицание, се намира кое е отрицанието, чиято област включва тези срещания и е най-малката такава област, и екзистенциалния квантор по променливата се слага веднага след отрицанието. А ако такова отрицание няма - тогава екзистенциалния квантор се поставя в началото на формулата.

След като по тялото на правилото се получи такава формула с екзистенциални квантори, тогава се образува импликация от тази формула към главата на правилото и се взима универсалното затваряне на тази импликация. Това е финалната формула, която съответства на правилото.

Примери.

На целта $?- p(X), q(X, Y)$ съответства формулата $\exists x \exists y (p(x) \& q(x, y))$.

На факта $p(12)$ съответства формулата $p(12)$.

От правило $r(X, Y) :- (r(X, X), p(Y)) ; (r(Y, Y), p(X))$ се получава следната формула $\forall x \forall y ((r(x, x) \& p(y)) \vee (r(y, y) \& p(x)) \implies r(x, y))$.

Правило $p(X) :- q(X, Y), p(Y)$ дава формулата $\forall x (\exists y (q(x, y) \& p(y)) \implies p(x))$.

А от $p(X) :- \text{not}(q(X, Y), \text{not}(r(Y, Z), p(Z)))$ се получава следната формула като резултат $\forall x (\neg \exists y (q(x, y) \& \neg \exists z (r(y, z) \& p(z))) \implies p(x))$.

Алгоритъм (Изпълняване чрез обратен извод).

Целта и телата на правилата от програмата се считат за безкванторни формули от

вида $\varphi_1 \& \varphi_2 \& \dots \& \varphi_n$. Така, от целта или от тялото на кое да е правило може да се образува списък от по-прости формули - $\{\varphi_1, \varphi_2, \dots \varphi_n\}$.

Започва се от списъка $L = \{\varphi_1, \varphi_2, \dots \varphi_n\}$, образуван от целта, и се обработва първата формула φ_1 . Обработването става така:

- ако φ_1 е атомарна формула, която се унифицира успешно с някой факт от програмата или с главата на някое правило, тогава всяко от тези унифицирания е алтернативно продължение на изпълнението; избира се първото неизползвано до сега унифициране и се продължава, като φ_1 се премахва от списъка L ако унифицирането е било с факт, а ако унифицирането е било с глава на правило, то φ_1 се заменя с формулите от списъка, съответстващ на тялото на това правило; ако всички унифицирания вече са използвани, тогава изпълнението на програмата се връща до предишен етап, на който все още има неизползвани унифицирания;
- ако φ_1 започва с предикат за извършване на действие (като **write**, или **nl**, или аритметично присвояване), това действие се извършва и ако резултатът е успешен, то φ_1 се премахва от списъка L и се продължава по-нататък; в противен случай, изпълнението на програмата се връща до предишен етап, на който все още има неизползвани унифицирания;
- ако φ_1 е аритметично сравнение, тогава това сравнение се извършва и ако резултатът е истина то φ_1 се премахва от списъка L и се продължава по-нататък; в противен случай, изпълнението на програмата се връща до предишен етап, на който все още има неизползвани унифицирания;
- ако φ_1 е от вида $\neg\psi$, стартира се ново изпълнение на програмата с цел ψ и ако то завърши неуспешно, тогава φ_1 се премахва от списъка L и се продължава по-нататък; в противен случай, изпълнението на програмата се връща до предишен етап, на който все още има неизползвани унифицирания.

Така се получава нов списък от формули. Той всъщност представлява нова цел на програмата, до която сме свели старата цел, чрез обработка на първата ѝ формула. След това първата формула на новата цел се обработва и новата цел се свежда до следваща цел, след това се обработва първата формула на следващата цел и т.н.

Целта се удовлетворява от програмата тогава и само тогава, когато от списъка от формули от целта чрез обработване на формули се получи празен списък (т.е. всички формули от целта се изпълняват чрез факти или правила от програмата).

Заб.: Текущия метод разчита, че в Пролог програмата и целта не е употребена дизюнкция. Ако има използване на дизюнкция - това или може да предизвика допълнително разделяне на изпълнението на алтернативни случаи, или може просто дизюнкцията да се замени с отрицание и конюнкция по законите на де Морган.

Пример.

Нека имаме следната програма на Пролог и цел:

```
p(X) :- q(X).      % clause (1)
q(X) :- r(X).      % clause (2)
q(10).             % clause (3)
r(2).              % clause (4)
```

```

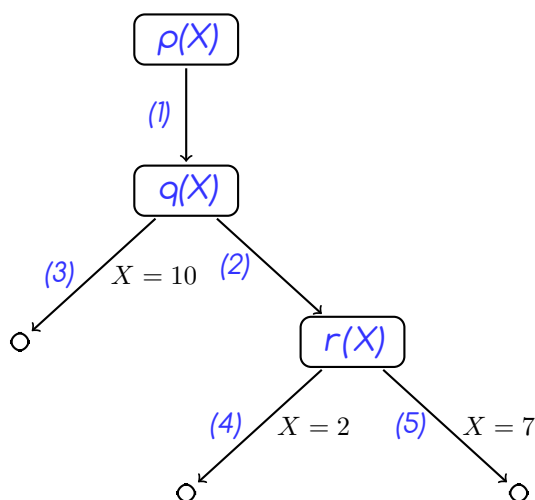
r(7).                % clause (5)
?- p(X).

```

Целта $p(X)$, чрез клауза (1), свеждаме до нова цел $q(X)$. След това $q(X)$ се унифицира с факта от клауза (3) и с главата на правилото (2). Така имаме две алтернативни продължения - чрез клауза (3) или чрез клауза (2). По първата алтернатива стигаме до празен списък от цели (т.е. изпълняваме всички цели) с унификация $X = 10$. Това е и първото удовлетворяване на целта от програмата.

Ако поискаме следващите удовлетворявания, тогава продължаваме по втората алтернатива за изпълнение на целта $q(X)$. При нея по правило (2) целта се свежда до $r(X)$. След това новата цел $r(X)$ може да се изпълни или чрез факт (4) (посредством унифициране $X = 2$) или чрез факт (5) (с унифициране $X = 7$). Тези случаи дават последните две удовлетворявания на целта.

Този алгоритъм е рекурсивен процес, който може да се изобрази чрез следната дървовидна структура:



Пример.

Ако имаме програма и цел:

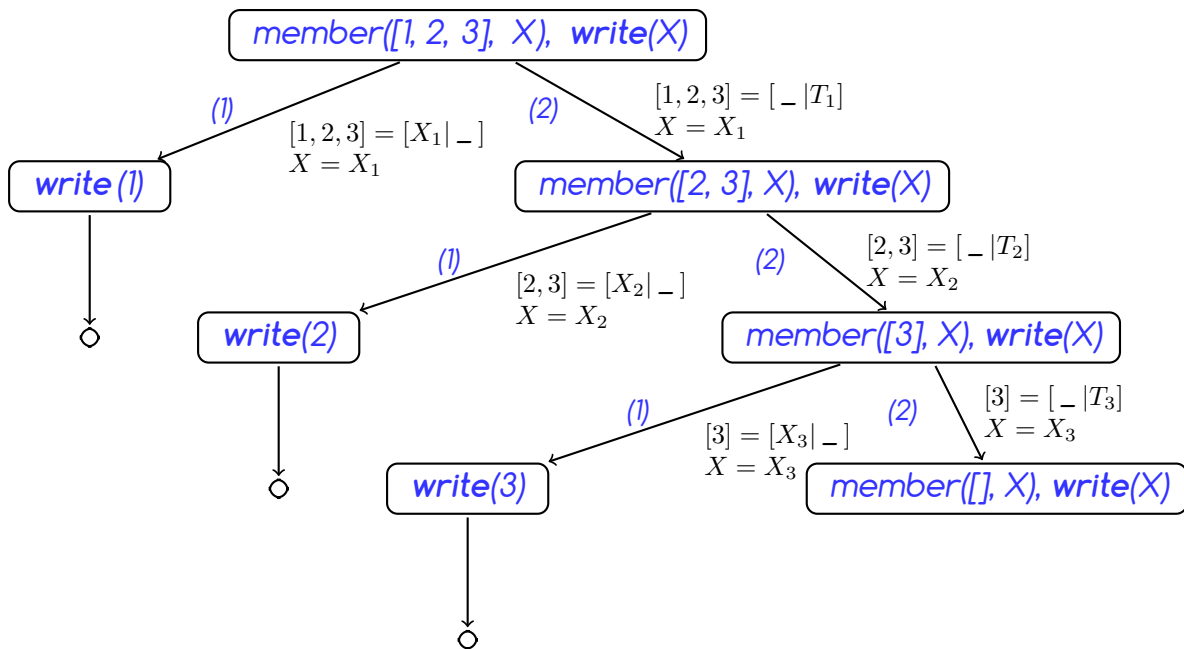
```

member([X | _], X).                % clause (1)
member([_ | T], X) :- member(T, X). % clause (2)
?- member([1, 2, 3], X), write(X).

```

Този програма може да се проследи чрез обратен извод чрез последващото дърво.

Да се обърне внимание, че когато някой факт или правило трябва да се употреби повече от веднъж при това проследяване, при всяка следваща употреба се преименуват променливите в него (например, втората клауза се употребява три пъти и всеки път е с нов набор от променливи $member([_ | T_1], X_1) :- member(T_1, X_1)$, $member([_ | T_2], X_2) :- member(T_2, X_2)$, и $member([_ | T_3], X_3) :- member(T_3, X_3)$).



Задача.

Дадени са следните три програми на Пролог:

Програма 1:	Програма 2:	Програма 3:
$\rho(5).$ $\rho(12).$ $\rho(X) :-$ $\rho(Y), X \text{ is } Y * 2.$	$\rho(5).$ $\rho(X) :-$ $\rho(Y), X \text{ is } Y * 2.$ $\rho(12).$	$\rho(X) :-$ $\rho(Y), X \text{ is } Y * 2.$ $\rho(5).$ $\rho(12).$

Ако изпълняваме програмите чрез обратен извод, какви ще са резултатите за всяка от тях при цел $?-\rho(X)$. Изобразете поне първите три нива от дърветата, които се получават при проследяването на програмите.

Какво трябва да се промени в начина на обхождане на дърветата, за да може трите програми да имат едни и същи резултати при тази цел.

Задача.

Дадени са следните три програми на Пролог:

Програма 1:	Програма 2:	Програма 3:
$\rho(20).$ $\rho(33).$ $\rho(X) :-$ $\rho(Y), X \text{ is } Y + 5.$	$\rho(X) :-$ $\rho(Y), X \text{ is } Y + 5.$ $\rho(20).$ $\rho(33).$	$\rho(20).$ $\rho(X) :-$ $\rho(Y), X \text{ is } Y + 5.$ $\rho(33).$

Ако изпълняваме програмите чрез обратен извод, какви ще са резултатите за всяка от тях при цел $?-\rho(X)$. Изобразете поне първите три нива от дърветата, които се получават при проследяването на програмите.

Какво трябва да се промени в начина на обхождане на дърветата, за да може трите програми да имат едни и същи резултати при тази цел.

❖ Изпълнимост на множества от затворени литерали ❖

Множеството от формули $\forall x \exists y (x < y)$, $0 < 1$, $1 < 2$, $2 < 3$, ... $2015 < 2016$, ... е изпълнимо. Поради употребата на числови константи е естествено да се търси структура, в която носителя е числово множество - \mathbb{N} , например.

Същия принцип може да се прилага и при употребата на не толкова традиционни символи. Например, множеството от $\forall x \forall y (p(x, y) \implies p(f(x), f(y)))$ и $p(c, f(c))$ е изпълнимо и при него може да се търси структура с обекти, съответстващи на f и c .

Дефиниция 33 (Ербранов носител).

Нека имаме предикатния език $L = \{ c, d, e, \dots, f, g, h, \dots, p, q, r, \dots \}$. Нека за всеки затворен терм τ при този език имаме специален обект, който ще бележим с o_τ . Тогава Ербранов носител за L ще наричаме множеството от всички такива специални обекти. Ербрановия носител ще бележим с H_L . Така

$$H_L = \{ o_\tau \mid \tau \text{ е затворен терм при езика } L \}.$$

Примери.

Ако имаме език $L = \{ p, f, c \}$, където p е двуместен предикатен символ, f е едноместен функционален символ, а c е константа, тогава Ербрановия носител за този език е:

$$H_L = \{ o_c, o_{f(c)}, o_{f(f(c))}, \dots \} = \{ o_{f^n(c)} \mid n \in \mathbb{N} \}$$

Ако има поне една константа и поне един функционален символ в езика, тогава H_L е безкрайно.

Ако езика не съдържа функционални символи, тогава H_L е крайно. Ако съдържа константи c, d и e , тогава $H_L = \{ o_c, o_d, o_e \}$.

А ако в L няма константи (независимо дали има функционални символи или не), то $H_L = \emptyset$.

Дефиниция 34 (Ербранова структура).

Нека имаме предикатен език L . Ербранова структура за L наричаме всяка една структура $\mathcal{H} = (H_L, \dots)$, която има същия език, нейният носител е H_L , а интерпретацията на константите и функционалните символи е както следва:

☛ за всяка константа c е изпълнено: $c^{\mathcal{H}} = o_c$;

☛ за всеки n -местен функционален символ f и всеки $o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n} \in H_L$ е изпълнено, че:
 $f^{\mathcal{H}}(o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n}) = o_{f(\tau_1, \tau_2, \dots, \tau_n)}$.

ВАЖНО: Винаги, когато говорим за Ербранови носители и Ербранови структури ще разчитаме, че в езика, с който работим, има поне една константа! В противен случай, в този език не може да има затворени термове и така Ербрановия носител ще се окаже празен.

Примери.

Ако имаме езика $L = \{ p, c, d, e \}$, където p е едноместен предикатен символ, а c, d и e са константи, тогава има осем възможни Ербранови структури за този език. Всички те имат един и същи носител - $H_L = \{ o_c, o_d, o_e \}$ - и една и съща интерпретация на константите c, d и e . Това, по което се различават е, че можем да имаме различни интерпретации на p - предиката може да дава истина или лъжа за елемент o_c , също има две възможности и за o_d и още две за o_e .

А ако добавим едноместен функционален символ към езика, тогава H_L става безкраен и също и броя на структурите - който е $2^{|H_L|}$ - става безкраен.

Твърдение 6 (Стойност на затворен терм при Ербранова структура).

Нека имаме Ербранова структура $\mathcal{H} = (H_L, \dots)$, като L е езика на структурата. Нека τ е произволен затворен терм при този език, а v е произволна оценка при \mathcal{H} . Тогава е изпълнено, че $v(\tau) = o_\tau$.

Доказателство.

Твърдението ще докажем с индукция по построението на τ :

- ако τ е константата c , тогава $v(c) = c^{\mathcal{H}} = o_c$;
- ако τ е изрази $f(\tau_1, \tau_2, \dots, \tau_n)$, където $\tau_1, \tau_2, \dots, \tau_n$ са по-прости затворени термове, тогава $v(f(\tau_1, \tau_2, \dots, \tau_n)) = f^{\mathcal{H}}(v(\tau_1), v(\tau_2), \dots, v(\tau_n)) = f^{\mathcal{H}}(o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n}) = o_{f(\tau_1, \tau_2, \dots, \tau_n)}$.



Твърдение 7 (Изпълнимост на множества от затворени атомарни формули).

Всяко множество от затворени атомарни формули е изпълнимо.

Доказателство.

Нека имаме произволно множество от затворени атомарни формули Φ и нека езика на това множество е L , като p_1, p_2, p_3, \dots са всички предикатни символи от този език. Да разгледаме Ербрановата структура $\mathcal{H} = (H_L, \dots)$, за която интерпретацията на предикатните символи е дефинирана за всяко i и за всеки $o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n} \in H_L$ точно така:

$$p_i^{\mathcal{H}}(o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n}) \text{ има стойност } T \quad \longleftrightarrow \quad p_i(\tau_1, \tau_2, \dots, \tau_n) \in \Phi.$$

Нека v е произволна оценка при тази структура. Ще покажем, че $\mathcal{H}, v \models \varphi$ за всяка $\varphi \in \Phi$.

Да разгледаме произволна $\varphi \in \Phi$. Тя е от вида $p_i(\tau_1, \tau_2, \dots, \tau_n)$, където $\tau_1, \tau_2, \dots, \tau_n$ са затворени термове. Тогава по предното твърдение имаме, че $v(\tau_i) = o_{\tau_i}$ за всяко i . Така получаваме:

$$\begin{aligned} p_i(\tau_1, \tau_2, \dots, \tau_n) \in \Phi &\longrightarrow p_i^{\mathcal{H}}(o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n}) \text{ има стойност } T \longrightarrow \\ p_i^{\mathcal{H}}(v(\tau_1), v(\tau_2), \dots, v(\tau_n)) \text{ има стойност } T &\longrightarrow \mathcal{H}, v \models p_i(\tau_1, \tau_2, \dots, \tau_n). \end{aligned}$$



Дефиниция 35 (Противоположни литерали).

За два литерала L_1 и L_2 ще казваме, че са противоположни ако единият от тях е атомарна формула, а другият е отрицание на същата тази атомарна формула.

Твърдение 8 (Критерий за изпълнимост на множества от затворени литерали).

Едно множество от затворени литерали е неизпълнимо тогава и само тогава, когато съдържа противоположни литерали.

Доказателство.

Нека Λ е множество от затворени литерали.

Правата посока (че ако Λ е неизпълнимо тогава то съдържа противоположни литерали) ще докажем в контрапозиция. Т.е., че ако Λ не съдържа противоположни литерали, тогава то е изпълнимо. Нека $\Phi = \{ L \mid L \in \Lambda \text{ и } L \text{ е положителен} \}$. Тогава Φ всъщност е множество от затворени атомарни формули. Следователно, по предното твърдение, има Ербранова структура $\mathcal{H} = (H_L, \dots)$ и оценка v при нея, чрез които всички тези формули са изпълнени. Ще покажем, че \mathcal{H} и v изпълняват и останалите литерали от Λ - отрицателните литерали. Нека $L \in \Lambda$ е отрицателен. Тогава той има вида $\neg p(\tau_1, \tau_2, \dots, \tau_n)$, където $\tau_1, \tau_2, \dots, \tau_n$ са затворени термове и $v(\tau_i) = o_{\tau_i}$. Т.к. Λ не съдържа противоположни литерали, то $p(\tau_1, \tau_2, \dots, \tau_n) \notin \Lambda$. Така получаваме

$$\begin{aligned} p(\tau_1, \tau_2, \dots, \tau_n) \notin \Lambda &\iff p^{\mathcal{H}}(o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n}) \text{ има стойност } F &\iff \\ p^{\mathcal{H}}(v(\tau_1), v(\tau_2), \dots, v(\tau_n)) \text{ има стойност } F &\iff \\ \mathcal{H}, v \not\models p(\tau_1, \tau_2, \dots, \tau_n) &\iff \mathcal{H}, v \models L. \end{aligned}$$

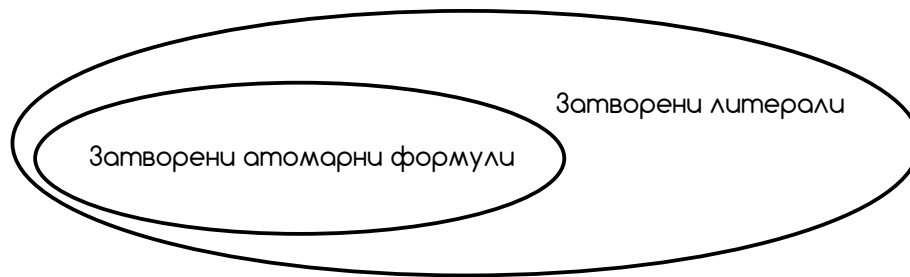
Остава да докажем само обратната посока на твърдението (че ако Λ съдържа противоположни литерали тогава то е неизпълнимо). Да допуснем, че Λ съдържа противоположни литерали L_1 и L_2 , но все пак е изпълнимо. Тогава има структура \mathcal{A} и оценка v такива, че $\mathcal{A}, v \models L_1$ и $\mathcal{A}, v \models L_2$. Но понеже L_1 и L_2 са противоположни, тогава от $\mathcal{A}, v \models L_1$ следва $\mathcal{A}, v \not\models L_2$, което е противоречие. Следователно Λ е неизпълнимо. ❖

Примери.

Множеството $\{ p(f(c), f(d)), p(d, c), q(h(f(d)), d), q(c, h(c)), \neg p(f(c), d), \neg p(d, d) \}$ е изпълнимо. В него няма двойка противоположни литерали.

Докато $\{ p(f(c), d), p(d, d), q(h(f(d)), d), q(c, h(c)), \neg p(f(c), d), \neg p(d, d) \}$ е неизпълнимо, понеже съдържа $p(f(c), d)$ и $\neg p(f(c), d)$. В него дори има и още една двойка противоположни литерали - $p(d, d)$ и $\neg p(d, d)$. Но една двойка е напълно достатъчна за показване на неизпълнимост.

Така, вече имаме критерии за определяне на изпълнимост на множества от първите два типа от йерархията на затворените формули. За всеки следващ клас от тази йерархия ще трябва да се формулират нов принцип/алгоритъм за проверяване на изпълнимост.



Задача.

Дадени са следните множества от затворени литерали

$$\Phi_1 = \{ \neg p(c), p(f(d)), q(d), p(f(c)) \}$$

$$\Phi_2 = \{ \neg p(c), p(f(c)), q(d), \neg q(f(c)) \}$$

$$\Phi_3 = \{ \neg p(c), p(f(c)), q(c), \neg p(f(c)) \}$$

$$\Phi_4 = \{ \neg q(c), p(f(d)), q(c), \neg q(f(c)) \}$$

Кои от тези множества са изпълними и кои не?

❖ Правила за извод ❖

Дефиниция 36 (Еквивалентно преобразуване).

Ако имаме две формули φ_1 и φ_2 казваме, че едната се преобразува еквивалентно в другата ако за всяка възможна структура \mathcal{A} и за всяка възможна оценка при тази структура v е изпълнено, че:

$$\mathcal{A}, v \models \varphi_1 \iff \mathcal{A}, v \models \varphi_2.$$

Това ще означаваме с $\varphi_1 \equiv \varphi_2$.

Твърдение 9 (Основни еквивалентни преобразувания).

Следните преобразувания са еквивалентни:

$$\begin{aligned} \alpha \implies \beta &\equiv \neg\alpha \vee \beta & (\implies) \\ \alpha \iff \beta &\equiv (\alpha \& \beta) \vee (\neg\alpha \& \neg\beta) & (\iff) \\ \neg(\alpha \implies \beta) &\equiv \alpha \& \neg\beta & (\neg \implies) \\ \neg(\alpha \iff \beta) &\equiv (\neg\alpha \& \beta) \vee (\alpha \& \neg\beta) & (\neg \iff) \\ \neg(\alpha \& \beta) &\equiv \neg\alpha \vee \neg\beta & (\neg \&) \\ \neg(\alpha \vee \beta) &\equiv \neg\alpha \& \neg\beta & (\neg \vee) \\ \neg\neg\alpha &\equiv \alpha & (\neg\neg) \end{aligned}$$

Доказателство.

Всяко от преобразуванията ще проверим според горната дефиниция за всяка структура \mathcal{A} и всяка оценка v при тази структура.

За преобразуване (\implies) имаме:

$$\begin{aligned} \mathcal{A}, v \models \alpha \implies \beta &\iff \mathcal{A}, v \not\models \alpha \text{ или } \mathcal{A}, v \models \beta & \iff \\ \mathcal{A}, v \models \neg\alpha \text{ или } \mathcal{A}, v \models \beta &\iff \mathcal{A}, v \models \neg\alpha \vee \beta. \end{aligned}$$

За преобразуване (\iff) получваме:

$$\begin{aligned} \mathcal{A}, v \models \alpha \iff \beta &\iff \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \not\models \alpha \text{ и } \mathcal{A}, v \not\models \beta & \iff \\ \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \models \neg\alpha \text{ и } \mathcal{A}, v \models \neg\beta &\iff \\ \mathcal{A}, v \models \alpha \& \beta \text{ или } \mathcal{A}, v \models \neg\alpha \& \neg\beta &\iff \mathcal{A}, v \models (\alpha \& \beta) \vee (\neg\alpha \& \neg\beta). \end{aligned}$$

Преобразуване $(\neg \implies)$ се показва така:

$$\begin{aligned} \mathcal{A}, v \models \neg(\alpha \implies \beta) &\iff \mathcal{A}, v \not\models \alpha \implies \beta & \iff \\ \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \not\models \beta &\iff \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \neg\beta & \iff \mathcal{A}, v \models \alpha \& \neg\beta. \end{aligned}$$

Преобразуване $(\neg \iff)$:

$$\begin{aligned} \mathcal{A}, v \models \neg(\alpha \iff \beta) &\iff \mathcal{A}, v \not\models \alpha \iff \beta & \iff \\ \mathcal{A}, v \not\models \alpha \text{ и } \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \not\models \beta &\iff \\ \mathcal{A}, v \models \neg\alpha \text{ и } \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \neg\beta &\iff \\ \mathcal{A}, v \models \neg\alpha \& \beta \text{ или } \mathcal{A}, v \models \alpha \& \neg\beta &\iff \mathcal{A}, v \models (\neg\alpha \& \beta) \vee (\alpha \& \neg\beta). \end{aligned}$$

Преобразуване ($\neg \&$):

$$\begin{aligned} \mathcal{A}, v \models \neg(\alpha \& \beta) &\longleftrightarrow \mathcal{A}, v \not\models \alpha \& \beta \longleftrightarrow \\ \mathcal{A}, v \not\models \alpha \text{ или } \mathcal{A}, v \not\models \beta &\longleftrightarrow \mathcal{A}, v \models \neg\alpha \text{ или } \mathcal{A}, v \models \neg\beta \longleftrightarrow \mathcal{A}, v \models \neg\alpha \vee \neg\beta. \end{aligned}$$

За реобразуване ($\neg \vee$) имаме:

$$\begin{aligned} \mathcal{A}, v \models \neg(\alpha \vee \beta) &\longleftrightarrow \mathcal{A}, v \not\models \alpha \vee \beta \longleftrightarrow \\ \mathcal{A}, v \not\models \alpha \text{ и } \mathcal{A}, v \not\models \beta &\longleftrightarrow \mathcal{A}, v \models \neg\alpha \text{ и } \mathcal{A}, v \models \neg\beta \longleftrightarrow \mathcal{A}, v \models \neg\alpha \& \neg\beta. \end{aligned}$$

Преобразуване ($\neg \neg$):

$$\mathcal{A}, v \models \neg\neg\alpha \longleftrightarrow \mathcal{A}, v \not\models \neg\alpha \longleftrightarrow \mathcal{A}, v \models \alpha.$$



Твърдение 10 (Еквивалентен запис на правила в Пролог програми).

Ако имаме правило $\{ \neg p_1(\dots), \neg p_2(\dots), \dots, \neg p_n(\dots), p(\dots) \}$ и x_1, x_2, \dots, x_m са всички променливи в положителния литерал $p(\dots)$, а y_1, y_2, \dots, y_k са всички останали променливи в клаузата, тогава следната формула е еквивалентна на интерпретацията на клаузата:

$$\forall x_1 \forall x_2 \dots \forall x_m (\exists y_1 \exists y_2 \dots \exists y_k (p_1(\dots) \& p_2(\dots) \& \dots \& p_n(\dots)) \implies p(\dots)).$$

Доказателство.

$$\begin{aligned} \forall x_1 \forall x_2 \dots \forall x_m \forall y_1 \forall y_2 \dots \forall y_k (\neg p_1(\dots) \vee \neg p_2(\dots) \vee \dots \vee \neg p_n(\dots) \vee p(\dots)) &\equiv \\ \forall x_1 \forall x_2 \dots \forall x_m (\forall y_1 \forall y_2 \dots \forall y_k (\neg p_1(\dots) \vee \neg p_2(\dots) \vee \dots \vee \neg p_n(\dots)) \vee p(\dots)) &\equiv \\ \forall x_1 \forall x_2 \dots \forall x_m (\forall y_1 \forall y_2 \dots \forall y_k \neg(p_1(\dots) \& p_2(\dots) \& \dots \& p_n(\dots)) \vee p(\dots)) &\equiv \\ \forall x_1 \forall x_2 \dots \forall x_m (\neg \exists y_1 \exists y_2 \dots \exists y_k (p_1(\dots) \& p_2(\dots) \& \dots \& p_n(\dots)) \vee p(\dots)) &\equiv \\ \forall x_1 \forall x_2 \dots \forall x_m (\exists y_1 \exists y_2 \dots \exists y_k (p_1(\dots) \& p_2(\dots) \& \dots \& p_n(\dots)) \implies p(\dots)) & \end{aligned}$$



Т.к. дизюнкцията може да се замени чрез законите на де Морган с отрицание и конюнкция, то при нужда можем да разчитаме, че в програмите на Пролог няма дизюнкция. Но често отсъствието на дизюнкция може да се постигне не само чрез законите на де Морган, но и чрез реструктуриране на клаузите в програмата. Например, наличието на дизюнкция може да се дължи на алтернативни случаи, които трябва да се обработят при проверката на някакво свойство $p(X)$. Да речем, че за свойството имаме следната клауза на Пролог:

$p(X) :- q(X) ; r(X).$

Ще покажем, че тази клауза е еквивалентна на комбинацията (т.е. на конюнкцията) от следните две клаузи:

$p(X) :- q(X).$
 $p(X) :- r(X).$

Първата клауза се интерпретира със следната формула $\forall x(q(x) \vee r(x) \implies p(x))$. Еквивалентни преобразувания показват

$$\begin{aligned} \forall x(q(x) \vee r(x) \implies p(x)) &\equiv \forall x(\neg(q(x) \vee r(x)) \vee p(x)) &&\equiv \\ \forall x((\neg q(x) \& \neg r(x)) \vee p(x)) &\equiv \forall x((\neg q(x) \vee p(x)) \& (\neg r(x) \vee p(x))) &&\equiv \\ \forall x((q(x) \implies p(x)) \& (r(x) \implies p(x))) &\equiv \forall x(q(x) \implies p(x)) \& \forall x(r(x) \implies p(x)) \end{aligned}$$

А $\forall x(q(x) \implies p(x))$ и $\forall x(r(x) \implies p(x))$ са точно формулите, съответстващи на вторите две клаузи.

Дефиниция 37 (Правило за извод).

Правило за извод наричаме комбинацията от множество от формули $\{ \varphi_1, \varphi_2, \dots, \varphi_n \}$ и формула φ , за които е изпълнено

$$\varphi_1, \varphi_2, \dots, \varphi_n \models \varphi.$$

Формулите $\varphi_1, \varphi_2, \dots, \varphi_n$ наричаме предпоставки на правилото, а φ е негово заключение.

Означение.

Когато имаме правило за извод от следния вид

$$\varphi_1, \varphi_2, \dots, \varphi_n \models (\psi_1^1 \& \psi_2^1 \& \dots \& \psi_{k_1}^1) \vee (\psi_1^2 \& \psi_2^2 \& \dots \& \psi_{k_2}^2) \vee \dots \vee (\psi_1^m \& \psi_2^m \& \dots \& \psi_{k_m}^m)$$

тогава ще го записваме така:

$$\frac{\varphi_1, \varphi_2, \dots, \varphi_n}{\psi_1^1, \psi_2^1, \dots, \psi_{k_1}^1 \mid \psi_1^2, \psi_2^2, \dots, \psi_{k_2}^2 \mid \dots \mid \psi_1^m, \psi_2^m, \dots, \psi_{k_m}^m}$$

В този случай, множествата $\{ \psi_1^1, \psi_2^1, \dots, \psi_{k_1}^1 \}, \{ \psi_1^2, \psi_2^2, \dots, \psi_{k_2}^2 \}, \dots, \{ \psi_1^m, \psi_2^m, \dots, \psi_{k_m}^m \}$ ще наричаме алтернативни заключения на правилото.

Твърдение 11 (Свойства на логическото следване).

За логическото следване са изпълнени следните две свойства:

- ☛ за всяка формула φ имаме, че $\varphi \models \varphi$;
- ☛ за всеки формули φ_1 и φ_2 имаме, че:

$$\varphi_1 \equiv \varphi_2 \iff \varphi_1 \models \varphi_2 \text{ и } \varphi_2 \models \varphi_1.$$

Доказателство.

Първото свойство следва от факта, че за всяка формула φ и за всяка структура \mathcal{A} и оценка v е изпълнено, че от $\mathcal{A}, v \models \varphi$ следва, че $\mathcal{A}, v \models \varphi$. Т.е. по дефиницията за логическо следване имаме $\varphi \models \varphi$.

Второ свойство се получава по следния начин

$\varphi_1 \equiv \varphi_2 \iff$
 за всеки \mathcal{A} и v , $\mathcal{A}, v \models \varphi_1$ тогава и само тогава, когато $\mathcal{A}, v \models \varphi_2 \iff$
 за всеки \mathcal{A} и v , ако $\mathcal{A}, v \models \varphi_1$ то $\mathcal{A}, v \models \varphi_2$ и за всеки \mathcal{A} и v , ако $\mathcal{A}, v \models \varphi_2$ то $\mathcal{A}, v \models \varphi_1 \iff$
 $\varphi_1 \models \varphi_2$ и $\varphi_2 \models \varphi_1$.




Твърдение 12 (Основни правила за извод).

Следните преобразувания са валидни правила за извод:

$$\begin{array}{c}
 \frac{\alpha \& \beta}{\alpha, \beta} \quad (\&) \quad \frac{\alpha \vee \beta}{\alpha \mid \beta} \quad (\vee) \quad \frac{\alpha \implies \beta}{\neg \alpha \mid \beta} \quad (\implies) \quad \frac{\alpha \iff \beta}{\alpha, \beta \mid \neg \alpha, \neg \beta} \quad (\iff) \\
 \\
 \frac{\neg(\alpha \& \beta)}{\neg \alpha \mid \neg \beta} \quad (\neg \&) \quad \frac{\neg(\alpha \vee \beta)}{\neg \alpha, \neg \beta} \quad (\neg \vee) \quad \frac{\neg(\alpha \implies \beta)}{\alpha, \neg \beta} \quad (\neg \implies) \quad \frac{\neg(\alpha \iff \beta)}{\neg \alpha, \beta \mid \alpha, \neg \beta} \quad (\neg \iff) \\
 \\
 \frac{\neg \neg \alpha}{\alpha} \quad (\neg \neg)
 \end{array}$$

Доказателство.

По първото свойство от предното твърдение имаме, че $\alpha \& \beta \models \alpha \& \beta$ и $\alpha \vee \beta \models \alpha \vee \beta$. Така, като ги запишем според уговореното означение се получават правила $(\&)$ и (\vee) .

Останалите правила се получават от едноименните еквивалентни преобразувания. Например, от преобразувание (\implies) по второто свойство от предното твърдение получаваме правилото $\alpha \implies \beta \models \neg \alpha \vee \beta$, което след това записваме по уговорения начин. 

Твърдение 13 (Вид на безкванторните формули).

Нека φ е безкванторна формула. Тогава φ или е литерал, или е от вид, който може да е предпоставка на някое от правилата от предното твърдение.

Доказателство.

Т.к. φ е безкванторна (т.е. в нея няма срещания на квантори), то според индуктивната дефиниция за построение на формула тя е съставена от атомарни формули, свързани с логически операции. Така можем да разгледаме следните възможни случаи за φ :

- ако φ е атомарна формула, то тогава φ е литерал;
- ако φ е съставена от други две безкванторни формули чрез някоя от операциите $\&$, \vee , \implies или \iff , то тогава тя може да е предпоставка на $(\&)$, (\vee) , (\implies) или (\iff) , съответно;

- ако φ е отрицание на друга безкванторна формула, то тогава тя или е отрицание на атомарна формула (т.е. е литерал), или е отрицание на по-сложна формула и може да е предпоставка на $(\neg \&)$, $(\neg \vee)$, $(\neg \implies)$, $(\neg \iff)$ или $(\neg \neg)$.



Задача.

Да се докаже, че следното правило е валидно правило за извод:

$$\frac{(\neg \alpha \vee \beta) \& (\alpha \vee \gamma)}{\beta \mid \gamma}$$

Задача.

Да се докаже, че следното правило е валидно правило за извод:

$$\frac{(\alpha \implies \beta) \& (\beta \implies \gamma)}{\alpha \implies \gamma}$$

Задача.

Да се докаже, че това е валидно правило за извод:

$$\frac{(\neg \alpha \iff \beta) \& \alpha}{\neg \beta}$$

Към кои от следните три формули е приложимо правилото:

- $(p(c) \iff r(d, d)) \& p(c);$
- $(\neg p(f(d)) \iff q(f(c), h(d))) \& p(f(d));$
- $\neg q(c, h(c)) \iff (r(d, h(c)) \& q(c, h(c))).$

Задача.

Да се докаже, че това е валидно правило за извод:

$$\frac{(\alpha \iff \neg \beta) \& \neg \alpha}{\beta}$$

Към кои от следните три формули е приложимо правилото:

☛ $p(f(d)) \iff (\neg r(d, d) \& \neg p(f(d)));$

☛ $(q(c, h(c)) \iff \neg r(d, h(c))) \& q(c, h(c));$

☛ $(p(c) \iff \neg q(f(c), h(d))) \& \neg p(c).$

❖ Табло метод ❖

Правилата за извод позволяват да свеждаме задачата за изпълнимост на множество от формули към (потенциално по-голямо) множество от по-прости формули. Например, ако приложим правилото за извод ($\&$) към първата формула от множеството

$$\{ p(c) \& q(f(c)), p(f(c)), p(d) \& \neg q(c) \}$$

то от него можем да получим ново множество

$$\{ p(c), q(f(c)), p(f(c)), p(d) \& \neg q(c) \}$$

което е също толкова изпълнимо колкото и началното. А с още едно приложение на ($\&$) получаваме множество от затворени литерали

$$\{ p(c), q(f(c)), p(f(c)), p(d), \neg q(c) \}$$

По същия начин можем да процедурираме, ако във формулите има и от останалите логически операции - \vee , \implies или \iff .

Например, ако множеството от формули е

$$\{ p(c) \vee q(f(c)), p(f(c)), p(d) \implies \neg q(c) \}$$

тогава като приложим правило (\vee) към първата формула, получаваме две алтернативни възможности

$$\{ p(c), p(f(c)), p(d) \implies \neg q(c) \} \quad \text{или} \quad \{ q(f(c)), p(f(c)), p(d) \implies \neg q(c) \}$$

А след това, като приложим правило (\implies) за последната формула $p(d) \implies \neg q(c)$, от всяка от двете възможности получаваме по още две (т.е. общо четири алтернативи), едната от които съдържа $\neg p(d)$, а другата $\neg q(c)$.

По този начин, чрез употреба на правила ($\&$), (\vee), (\implies), (\iff), ($\neg \&$), ($\neg \vee$), ($\neg \implies$), ($\neg \iff$) и ($\neg \neg$), можем от произволно множество от затворени безкванторни формули да получим едно или повече алтернативни множества от затворени литерали. Т.к. получаването на тези алтернативи е рекурсивен процес и по естествен начин образува дървовидна структура, то за да систематизираме процеса ще използваме специален вид дървета, които съдържат (множества от) формули.

Дефиниция 38 (Клон в дърво).

Клон в кореново дърво ще наричаме път от корена до някое листо в дървото. Пътят включва и корена и листото.

Дефиниция 39 (Дърво с формули).

Под дърво с формули ще разбираме кореново дърво, във всеки възел на което е записана точно една формула.

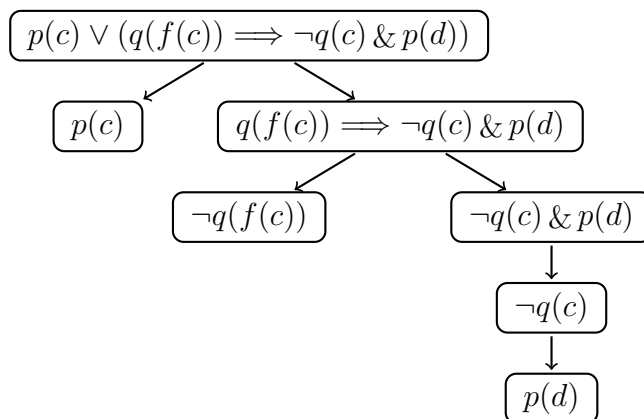
Заб.: Всеки възел от дървото съдържа само по една формула, но всеки клон от дървото може да образува множество от формули.

Дефиниция 40 (Противоречив и непротиворечив клон).

Ако имаме дърво с формули, то даден клон в него ще наричаме *противоречив* ако съдържа два противоположни литерала. В противен случай клонът е *непротиворечив*.

Пример.

Нека разгледаме следното дърво от формули:



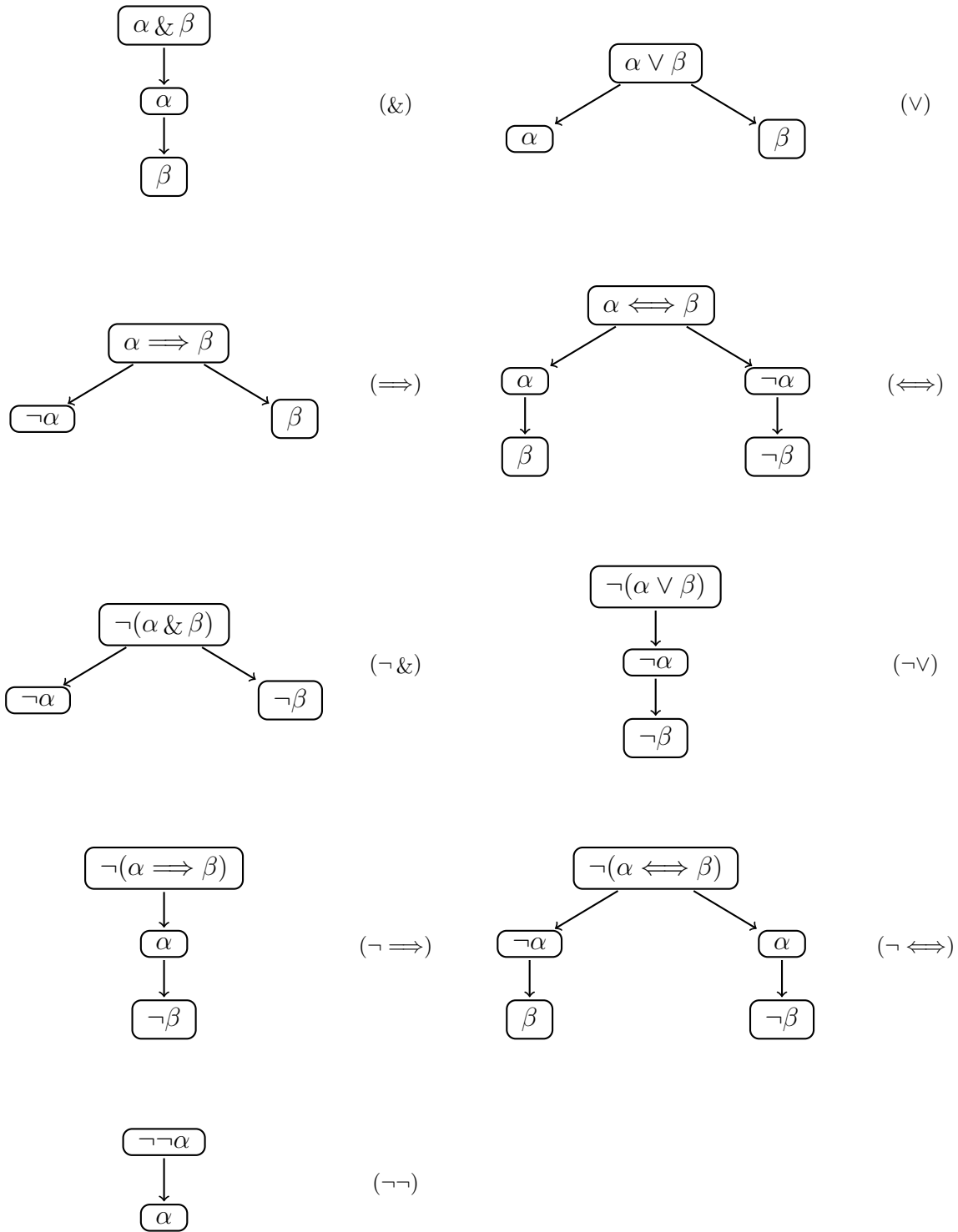
В него има три клона - първия клон съдържа формулите $\{ p(c) \vee (q(f(c)) \Rightarrow \neg q(c) \wedge p(d)), p(c) \}$, във втория се съдържат $\{ p(c) \vee (q(f(c)) \Rightarrow \neg q(c) \wedge p(d)), q(f(c)) \Rightarrow \neg q(c) \wedge p(d), \neg q(f(c)) \}$, а третия формира множество $\{ p(c) \vee (q(f(c)) \Rightarrow \neg q(c) \wedge p(d)), q(f(c)) \Rightarrow \neg q(c) \wedge p(d), \neg q(c) \wedge p(d), \neg q(c), p(d) \}$. И трите клона се оказват *непротиворечиви*.

Дефиниция 41 (Табло за формула).

Нека φ е затворена безкванторна формула. Тогава Табло за φ ще наричаме специално дърво с формули, което се построява така:

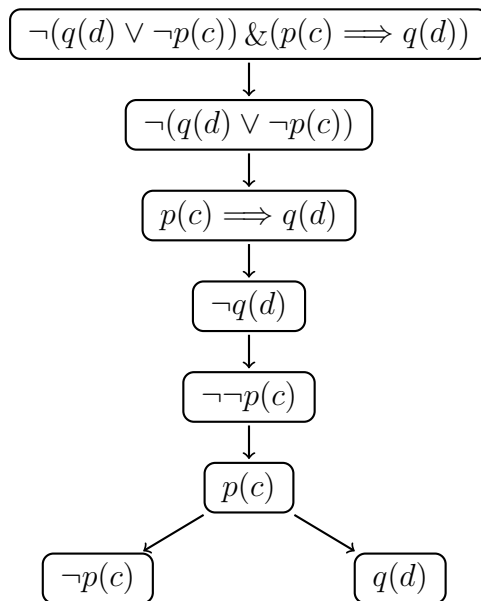
- в корена на дървото записваме φ ; ако φ е литерал, то тогава я маркираме като обработена и завършваме процеса; иначе, маркираме φ като необработена и продължаваме със следващата стъпка;
- ако в дървото има поне една необработена формула, която е предпоставка на някое от правилата $(\&)$, (\vee) , (\Rightarrow) , (\Leftarrow) , $(\neg\&)$, $(\neg\vee)$, $(\neg\Rightarrow)$, $(\neg\Leftarrow)$ или $(\neg\neg)$, то тогава разширяваме дървото като края на всеки *непротиворечив* клон, в който се съдържа необработената формула, продължим със всяко едно от алтернативните заключения на правилото; след като извършим това, маркираме формулата като обработена; съответно, всяка друга нова формула, която добавяме към дървото по време на това действие, маркираме като *обработена*, ако тя е литерал, и като *необработена* в противен случай;
- процесът приключва, когато получим дърво, в което всички формули са обработени; тогава казваме, че сме получили завършено Табло за φ .

Всяко Табло за формула може да се получи като комбинация от следните минимални дървета (всяко едно от тях се получава с еднократно прилагане на някое от правилата $(\&)$, (\vee) , (\Rightarrow) , (\Leftarrow) , $(\neg\&)$, $(\neg\vee)$, $(\neg\Rightarrow)$, $(\neg\Leftarrow)$, $(\neg\neg)$):



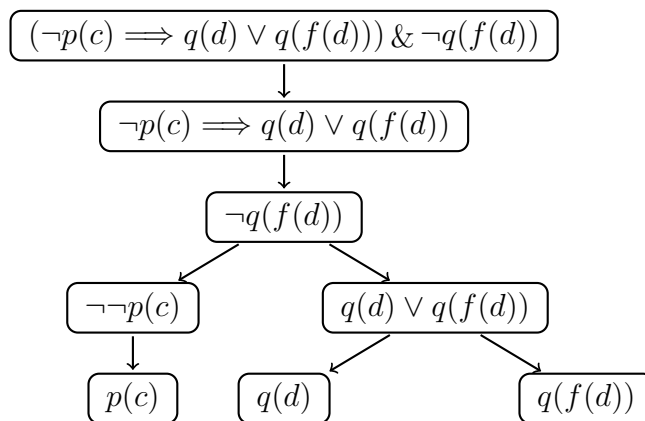
Примери.

Като построим Таблото за формула $\neg(q(d) \vee \neg p(c)) \& (p(c) \implies q(d))$



виждаме, че всички клонове в него са противоречиви. Първият е такъв, понеже съдържа $p(c)$ и $\neg p(c)$, а втория заради $\neg q(d)$ и $q(d)$.

Таблото за формула $(\neg p(c) \implies q(d) \vee q(f(d))) \& \neg q(f(d))$



обаче, има два непротиворечиви клона и един противоречив (заради $\neg q(f(d))$ и $q(f(d))$).

Дефиниция 42 (Табло за множество от формули).

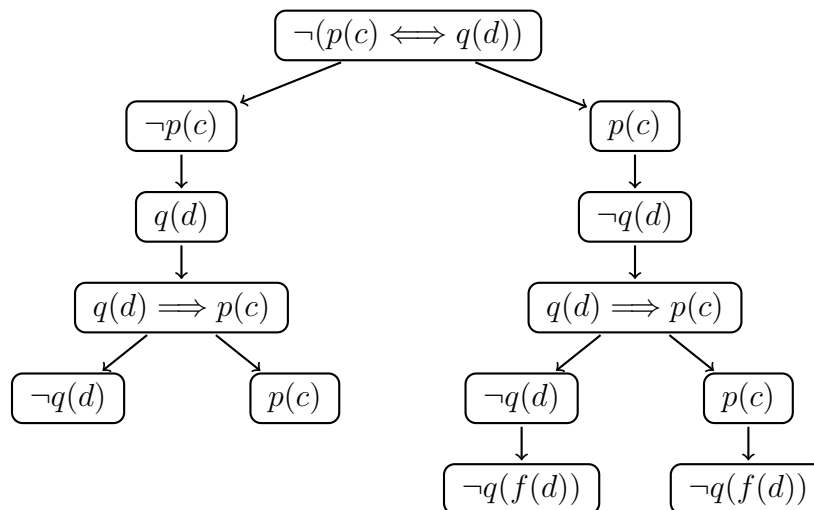
Ако имаме множество от затворени безкванторни формули $\{\varphi_1, \varphi_2, \dots\}$, тогава Табло за това множество построяваме по следния индуктивен начин:

- първо построяваме Табло за формула φ_1 ;

- да речем, че сме построили Табло за множеството $\{ \varphi_1, \varphi_2, \dots, \varphi_i \}$; тогава накрая на всеки непротиворечив клон от това Табло добавяме формулата φ_{i+1} и след това завършваме полученото дърво по дефиницията за построение на Табло.

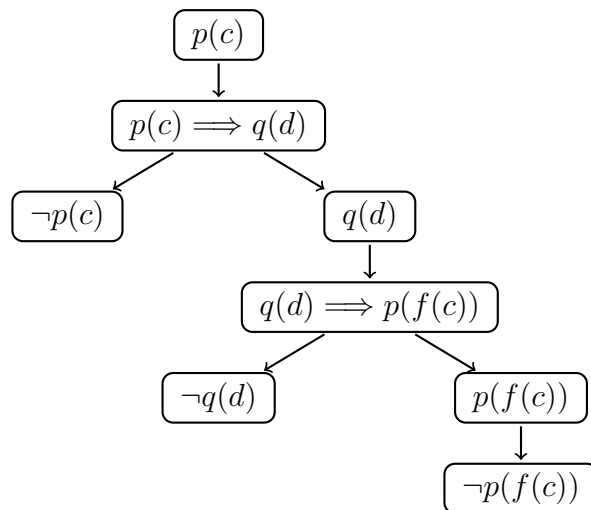
Примери.

За множество $\{ \neg(p(c) \iff q(d)), q(d) \implies p(c), \neg q(f(d)) \}$ получаваме Таблото



То съдържа два непротиворечиви и два противоречиви клона.

Ето и Табло за $\{ p(c), p(c) \implies q(d), q(d) \implies p(f(c)), \neg p(f(c)) \}$



В него всички клонове са противоречиви.

Пример (Съвместяване на програми).

Табло метода можем да приложим за да покажем, че формулите от примера за съвместяване на програми са неизпълнимо множество:

- ☛ $traffic(internet) \vee traffic(intranet)$
- ☛ $\neg(traffic(internet) \& traffic(intranet))$
- ☛ $traffic(internet) \implies \neg start(scanner) \& start(filter)$
- ☛ $traffic(intranet) \implies start(scanner)$
- ☛ $traffic(internet) \implies start(scanner) \& \neg start(filter)$
- ☛ $traffic(intranet) \implies \neg start(scanner) \& start(filter)$

Твърдение 14 (Коректност на табло метода).

Нека имаме множество от затворени безкванторни формули $\{\varphi_1, \varphi_2, \dots\}$ и нека сме построили завършеното Табло за това множество. Ако в Таблото има поне един непротиворечив клон, тогава $\{\varphi_1, \varphi_2, \dots\}$ е изпълнимо.

Доказателство.

Нека разгледаме множеството Δ от всички литерали, които са записани в непротиворечивия клон на Таблото. Т.к. формулите φ_i са затворени, следователно множеството Δ е от затворени литерали и не съдържа двойка противоположни литерали. Тогава, ако L е езика на формулите, следователно има Ербрановата структура $\mathcal{H} = (H_L, \dots)$ и оценка v , които изпълняват Δ . Ще покажем, че всяка формула, която е записана по непротиворечивия клон, е изпълнена при \mathcal{H} и v . Т.к. всяка от формулите $\{\varphi_1, \varphi_2, \dots\}$ се намира някъде по този клон от това следва, че $\mathcal{H}, v \models \varphi_i$ за всяка от тях и така цялото множество е изпълнимо.

Ще докажем нужното с индукция по вида на формулите от клона. От твърдението от предната глава за вида на безкванторните формули знаем, че всяка от формулите по клона е или затворен литерал, или е по-сложна формула, която е предпоставка за някое от правилата за извод ($\&$), (\vee), (\implies), (\iff), ($\neg \&$), ($\neg \vee$), ($\neg \implies$), ($\neg \iff$) или ($\neg \neg$). Така, имаме два основни случая:

- ☛ ако формулата е литерал, то тя е в Δ и се изпълнява от \mathcal{H} и v ;
- ☛ ако формулата е предпоставка на правило за извод, то поне едно от алтернативните заключения на правилото е част от клона и по индукционно предположение, формулите от това заключение са изпълнени при \mathcal{H} и v ; така, като използваме еквивалентното преобразуване, което съответства на правилото, можем да видим, че тогава и текущата формула-предпоставка е изпълнена при \mathcal{H} и v .



Твърдение 15 (Пълнота на табло метода).

Нека имаме множество от затворени безкванторни формули $\{\varphi_1, \varphi_2, \dots\}$ и нека сме построили завършеното Табло за това множество. Ако $\{\varphi_1, \varphi_2, \dots\}$ е изпълнимо, тогава в Таблото има поне един непротиворечив клон.

Доказателство.

Ако $\{ \varphi_1, \varphi_2, \dots \}$ е изпълнимо, то тогава има структура \mathcal{A} и оценка v , които изпълняват всяка формула от множеството. Ще покажем, че тогава има поне един клон в Таблото, за който е вярно, че всяка формула от клона е изпълнена при \mathcal{A} и v . Така, този клон не може да съдържа двойка противоположни литерали, т.к. тогава те ще трябва едновременно да са изпълнени при \mathcal{A} и v , а това ще е противоречие, и следователно клонът ще е непротиворечив.

Намирането на клона започваме от корена, в който е записана формула φ_1 . Тя е изпълнена при \mathcal{A} и v и така пътя, започващ от корена на дървото и завършващ във възела с текущата формула (в случая това е път с дължина 1), е изпълнен от \mathcal{A} и v . Останалата част от клона (докато стигнем до листо) се намира чрез следните стъпки:

- ако пътя, намерен до сега, включва формули $\{ \varphi_1, \varphi_2, \dots, \varphi_i \}$ и последната формула по пътя е последната формула от Таблото за φ_i , то тогава следващата формула в дървото непременно е φ_{i+1} и тя е изпълнена от \mathcal{A} и v и така продължаваме пътя с формула φ_{i+1} ;
- ако последната формула по пътя не е последната формула от Таблото за някое φ_i и след нея има разклонение в дървото, то тогава това разклонение се дължи на правило за извод с няколко алтернативни заключения; т.к. предпоставката на това правило участва в по-ранна част от пътя до тук, тя се изпълнява от \mathcal{A} и v и следователно поне едно от алтернативните ѝ заключения е изпълнено от \mathcal{A} и v и така продължаваме пътя с формулите от това алтернативно заключение;
- ако последната формула по пътя не е последната формула от Таблото за някое φ_i и след нея няма разклонение в дървото, то тогава последващите формули са единствено алтернативно заключение на правило за извод; така, на същия принцип както от предната стъпка, към пътя добавяме всичките формули от това заключение;
- ако последната формула от пътя до тук не е от някой от горните три вида, то тогава тя е листо и така сме намерили търсения клон.



Така, като просто комбинираме предните две твърдения получаваме следната

Теорема 1 (Коректност и пълнота на табло метода).

Нека имаме множество от затворени безкванторни формули $\{ \varphi_1, \varphi_2, \dots \}$ и нека сме построили завършеното Табло за това множество. Тогава в Таблото има поне един непротиворечив клон тогава и само тогава, когато $\{ \varphi_1, \varphi_2, \dots \}$ е изпълнимо.

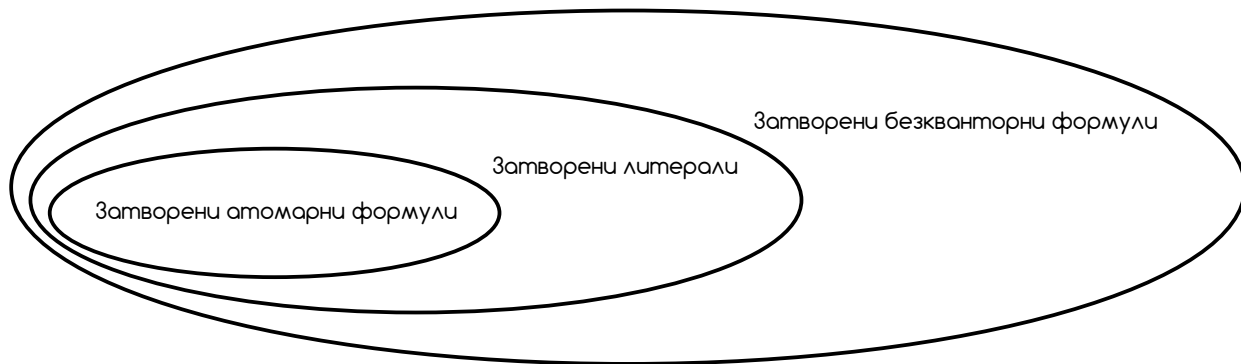
Следствие 5 (Неизпълнимост на множества от затворени безкванторни формули).

Нека имаме множество от затворени безкванторни формули $\{ \varphi_1, \varphi_2, \dots \}$ и нека сме построили завършеното Табло за това множество. Тогава всички клонове в Таблото са противоречиви тогава и само тогава, когато $\{ \varphi_1, \varphi_2, \dots \}$ е неизпълнимо.

Така, Табло метода може да се използва за следниве цели:

- ☛ да определи дали едно множество от затворени безкванторни формули е изпълнимо или не;
- ☛ допълнително предимство е, че ако множеството е изпълнимо, то по непротиворечивите клонове в Таблото могат да се построят Ербранови структури, които изпълняват това множество.

По този начин вече имаме метод за проверяване на изпълнимост или неизпълнимост за множества от по-голям клас от формули - затворените безкванторни формули.



Лема 1 (Лема на Кьониг).

Ако едно дърво с краен индекс на разклонение е безкрайно, то тогава има безкраен клон в него.

Теорема 2 (Компактност на изпълнимостта).

Едно множество от затворени безкванторни формули е изпълнимо тогава и само тогава, когато всяко негово крайно подмножество е изпълнимо.

Доказателство.

Правата посока на теоремата е сравнително тривиална. Ако дадено множество от затворени безкванторни формули е изпълнимо, то има структура \mathcal{A} и оценка v , които изпълняват всяка формула от множеството. Следователно, \mathcal{A} и v изпълняват всички формули на кое да е крайно подмножество на даденото такова и така крайното подмножество също е изпълнимо.

За обратната посока ще покажем, че ако всяко крайно подмножество е изпълнено, то и основното множество също е такова. Ако основното множество е крайно, то тогава то е крайно подмножество на себе си и заради това е изпълнено. Остава да видим случая, в който множеството е безкрайно.

Нека основното множество (което е безкрайно) е $\{ \varphi_1, \varphi_2, \dots \}$. Тогава $\{ \varphi_1, \varphi_2, \dots, \varphi_i \}$ е изпълнимо за всяко i . Така, на всяка стъпка от построението на Таблото за $\{ \varphi_1, \varphi_2, \dots \}$ имаме непротиворечив клон в Таблото, построено до тук, понеже това множество е изпълнимо. Така, винаги когато добавяме следващата формула φ_{i+1} , това можем да направим към непротиворечив клон и така Таблото се разширява. И т.к. броя на формулите е

безкраен, то и Таблото се получава безкрайно. А понеже неговия индекс на разклонение е 2, то следва, че има безкраен клон в него. Този клон не може да е противоречив, т.к. всички противоречиви клонове са крайни. Следователно клонът е непротиворечив и в него са записани всички формули от множеството и от него ще се съставят изпълняващи цялото множество структура и оценка. ❖

Следствие 6 (Компактност на неизпълнимостта).

Едно множество от затворени безкванторни формули е неизпълнимо тогава и само тогава, когато има поне едно негово крайно подмножество, което е неизпълнимо.

Задача.

Да се определи, чрез Табло метода, дали следното множество от затворени безкванторни формули е изпълнимо или не:

- ☛ $(p(f(c)) \vee \neg q(d)) \iff q(f(f(c)))$;
- ☛ $p(c) \implies q(f(f(c)))$;
- ☛ $p(c) \& q(d) \& \neg p(f(c))$.

Задача.

Да се определи, чрез Табло метода, дали следното множество от затворени безкванторни формули е изпълнимо или не:

- ☛ $(q(d) \vee \neg p(d, f(c))) \iff (p(f(c), c) \implies p(d, d))$;
- ☛ $\neg p(d, d) \& (p(d, f(c)) \implies q(c))$;
- ☛ $p(f(c), c) \& \neg q(c)$.

Задача.

Дадени са следните множества от затворени безкванторни формули:

$$\begin{aligned}\Phi_1 &= \{ \neg(p(f(c)) \implies q(d)) , p(c) , q(f(c)) \} \\ \Phi_2 &= \{ q(d) , p(f(c)) \& p(c) , p(f(d)) \} \\ \Phi_3 &= \{ p(f(c)) , q(f(c)) , \neg(\neg p(c) \vee p(f(d))) \}\end{aligned}$$

Обединението на кои две от тях е изпълнимо множество? Обосновете отговора си!

Задача.

Дадени са следните множества от затворени безкванторни формули:

$$\begin{aligned}\Phi_1 &= \{ p(d) , q(g(d)) , p(c) \& q(g(c)) \} \\ \Phi_2 &= \{ p(c) , \neg(\neg q(g(c)) \vee q(g(d))) , p(g(d)) \} \\ \Phi_3 &= \{ \neg(p(c) \implies p(d)) , q(g(c)) , p(g(d)) \}\end{aligned}$$

Обединението на кои две от тях е изпълнимо множество? Обосновете отговора си!

Задача.

Да се докаже, че следното правило е валидно правило за извод:

$$\frac{(\alpha \implies \beta) \& \alpha}{\beta}$$

Да се използва това правило при прилагане на Табло метода за определяне на изпълнимостта на следното множество от формули:

- ☛ $\{ (p(c) \implies (q(d) \vee p(f(c)))) \& p(c);$
- ☛ $(q(c) \implies \neg p(f(c))) \& q(c);$
- ☛ $\neg(q(d) \vee \neg p(f(c))) \}$.

Задача.

Да се докаже, че следното правило е валидно правило за извод:

$$\frac{(\alpha \implies \beta) \& \neg \beta}{\neg \alpha}$$

Да се използва това правило при прилагане на Табло метода за определяне на изпълнимостта на следното множество от формули:

- ☛ $\{ ((p(f(c)) \& q(d)) \implies q(f(c))) \& \neg q(f(c));$
- ☛ $(\neg q(d) \implies p(d)) \& \neg p(d);$
- ☛ $p(f(c)) \& p(f(d)) \}$.

❖ Изпълнимост на множества от универсални формули ❖

Дефиниция 43 (Субституция).

Субституция наричаме крайно множество от двойки от променлива x_i и терм τ_i . Всяка от тези двойки възприемаме като замяна на променливата със съответния терм. Субституцията ще записваме така $\sigma = \{ x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n \}$.

Заб.: Възможно е и $n = 0$. Тогава имаме празна субституция $\sigma = \{ \}$.

Дефиниция 44 (Частен случай).

Ако имаме субституция $\sigma = \{ x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n \}$ и терм τ или формула φ (в която никоя от променливите x_i не се среща свързано), тогава частен случай на τ или φ ще наричаме израза, който се получава от тях като всяка от променливите x_i заменим със съответния ѝ терм τ_i . Резултата означаваме със $\sigma\tau$ или $\sigma\varphi$ съответно.

Също, ако имаме универсалната формула $\forall x_1 \forall x_2 \dots \forall x_n \varphi$, тогава безкванторната формула $\sigma\varphi$ ще наричаме неин частен случай.

Примери.

$f(g(x, y), d)$, $f(g(d, d), d)$, $f(g(g(d, d), g(d, d)), d)$ и $f(g(h(c), z), d)$ са частни случаи на $f(g(x, y), d)$.

Частни случаи на безкванторната формула $\neg p(x) \implies (q(c) \vee p(f(y)))$ са:

$\neg p(t) \implies (q(c) \vee p(f(f(t))))$ и $\neg p(f(f(c))) \implies (q(c) \vee p(f(d)))$.

А някои от частните случаи на универсалната формула $\forall x \exists y (p(x) \implies q(y, x))$ са:

$p(x) \implies q(y, x)$, $p(y) \implies q(x, y)$ и $p(c) \implies q(f(f(z)), c)$.

Твърдение 16 (Остойностяване на терм при субституция).

Нека \mathcal{A} и v са произволни структура и оценка при нея. Тогава за всеки терм τ и за всяка субституция $\sigma = \{ x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n \}$ е изпълнено, че

$$v[x_1 = v(\tau_1), x_2 = v(\tau_2), \dots, x_n = v(\tau_n)](\tau) = v(\sigma\tau).$$

Доказателство.

За удобство, ще означим оценката $v[x_1 = v(\tau_1), x_2 = v(\tau_2), \dots, x_n = v(\tau_n)]$ с v^σ . Твърдението ще докажем с индукция по построението на τ :

- ☛ ако τ е индивидната променлива x_i за някое i , тогава $v^\sigma(x_i) = v(\tau_i) = v(\sigma x_i)$;
- ☛ ако τ е индивидна променлива x , различна от x_i за всяко i , тогава $v^\sigma(x) = v(x) = v(\sigma x)$;
- ☛ ако τ е константата c , тогава $v^\sigma(c) = c^{\mathcal{A}} = v(c) = v(\sigma c)$;
- ☛ ако τ е израза $f(\pi_1, \pi_2, \dots, \pi_m)$, тогава $v^\sigma(f(\pi_1, \pi_2, \dots, \pi_m)) = f^{\mathcal{A}}(v^\sigma(\pi_1), v^\sigma(\pi_2), \dots, v^\sigma(\pi_m)) = f^{\mathcal{A}}(v(\sigma\pi_1), v(\sigma\pi_2), \dots, v(\sigma\pi_m)) = v(f(\sigma\pi_1, \sigma\pi_2, \dots, \sigma\pi_m)) = v(\sigma f(\pi_1, \pi_2, \dots, \pi_m))$.



Твърдение 17 (Остойносттаване на безкванторна формула при субституция).

Нека \mathcal{A} и v са произволни структура и оценка при нея. Тогава за всяка безкванторна формула φ и за всяка субституция $\sigma = \{x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n\}$ е изпълнено, че

$$\mathcal{A}, v[x_1 = v(\tau_1), x_2 = v(\tau_2), \dots, x_n = v(\tau_n)] \models \varphi \quad \longleftrightarrow \quad \mathcal{A}, v \models \sigma\varphi.$$

Доказателство.

За удобство, ще означим оценката $v[x_1 = v(\tau_1), x_2 = v(\tau_2), \dots, x_n = v(\tau_n)]$ с v^σ . Твърдението ще докажем с индукция по построението на φ :

☛ ако φ е атомарната формула $p(\pi_1, \pi_2, \dots, \pi_m)$ тогава

$$\begin{aligned} \mathcal{A}, v^\sigma \models p(\pi_1, \pi_2, \dots, \pi_m) & \longleftrightarrow \\ \text{стойността на } p^{\mathcal{A}}(v^\sigma(\pi_1), v^\sigma(\pi_2), \dots, v^\sigma(\pi_m)) \text{ е } T & \longleftrightarrow \\ \text{стойността на } p^{\mathcal{A}}(v(\sigma\pi_1), v(\sigma\pi_2), \dots, v(\sigma\pi_m)) \text{ е } T & \longleftrightarrow \\ \mathcal{A}, v \models p(\sigma\pi_1, \sigma\pi_2, \dots, \sigma\pi_m) & \longleftrightarrow \quad \mathcal{A}, v \models \sigma p(\pi_1, \pi_2, \dots, \pi_m); \end{aligned}$$

☛ ако φ е $\neg\alpha$ тогава

$$\mathcal{A}, v^\sigma \models \varphi \quad \longleftrightarrow \quad \mathcal{A}, v^\sigma \not\models \alpha \quad \longleftrightarrow \quad \mathcal{A}, v \not\models \sigma\alpha \quad \longleftrightarrow \quad \mathcal{A}, v \models \sigma\varphi;$$

☛ ако φ е $(\alpha \& \beta)$ тогава

$$\begin{aligned} \mathcal{A}, v^\sigma \models \varphi & \longleftrightarrow \quad \mathcal{A}, v^\sigma \models \alpha \text{ и } \mathcal{A}, v^\sigma \models \beta \quad \longleftrightarrow \\ \mathcal{A}, v \models \sigma\alpha \text{ и } \mathcal{A}, v \models \sigma\beta & \longleftrightarrow \quad \mathcal{A}, v \models \sigma\varphi; \end{aligned}$$

☛ ако φ е $(\alpha \vee \beta)$ тогава

$$\begin{aligned} \mathcal{A}, v^\sigma \models \varphi & \longleftrightarrow \quad \mathcal{A}, v^\sigma \models \alpha \text{ или } \mathcal{A}, v^\sigma \models \beta \quad \longleftrightarrow \\ \mathcal{A}, v \models \sigma\alpha \text{ или } \mathcal{A}, v \models \sigma\beta & \longleftrightarrow \quad \mathcal{A}, v \models \sigma\varphi; \end{aligned}$$

☛ ако φ е $(\alpha \implies \beta)$ тогава

$$\begin{aligned} \mathcal{A}, v^\sigma \models \varphi & \longleftrightarrow \quad \mathcal{A}, v^\sigma \not\models \alpha \text{ или } \mathcal{A}, v^\sigma \models \beta \quad \longleftrightarrow \\ \mathcal{A}, v \not\models \sigma\alpha \text{ или } \mathcal{A}, v \models \sigma\beta & \longleftrightarrow \quad \mathcal{A}, v \models \sigma\varphi; \end{aligned}$$

☛ ако φ е $(\alpha \iff \beta)$ тогава

$$\begin{aligned} \mathcal{A}, v^\sigma \models \varphi & \longleftrightarrow \quad \mathcal{A}, v^\sigma \models \alpha \text{ и } \mathcal{A}, v^\sigma \models \beta \text{ или } \mathcal{A}, v^\sigma \not\models \alpha \text{ и } \mathcal{A}, v^\sigma \not\models \beta \quad \longleftrightarrow \\ \mathcal{A}, v \models \sigma\alpha \text{ и } \mathcal{A}, v \models \sigma\beta \text{ или } \mathcal{A}, v \not\models \sigma\alpha \text{ и } \mathcal{A}, v \not\models \sigma\beta & \longleftrightarrow \quad \mathcal{A}, v \models \sigma\varphi; \end{aligned}$$



Твърдение 18 (Правило за извод на частни случаи).

Следното правило за извод е валидно за произволна универсална формула $\forall x_1 \forall x_2 \dots \forall x_n \varphi$ и произволна субституция σ :

$$\frac{\forall x_1 \forall x_2 \dots \forall x_n \varphi}{\sigma \varphi} \quad (\sigma)$$

Доказателство.

Ще покажем, че за всяка структура $\mathcal{A} = (A, \dots)$ и за всяка оценка v при нея е изпълнено

$$\mathcal{A}, v \models \forall x_1 \forall x_2 \dots \forall x_n \varphi \quad \longrightarrow \quad \mathcal{A}, v \models \sigma \varphi.$$

Така, по дефиниция ще имаме $\forall x_1 \forall x_2 \dots \forall x_n \varphi \models \sigma \varphi$.

Нека $\mathcal{A} = (A, \dots)$ и v са произволна структура и оценка. От $\mathcal{A}, v \models \forall x_1 \forall x_2 \dots \forall x_n \varphi$ следва, че $\mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \varphi$ за всеки $a_1, a_2, \dots, a_n \in A$ и по-специално

$$\mathcal{A}, v[x_1 = v(\sigma x_1), x_2 = v(\sigma x_2), \dots, x_n = v(\sigma x_n)] \models \varphi.$$

А от последния резултат, по предното твърдение, вече получаваме $\mathcal{A}, v \models \sigma \varphi$. ✖

Пример (Снабдяване при търговия).

Чрез това правило можем да покажем, че следните универсални формули от примера за снабдяване при търговия са неизпълнимо множество

- ☛ $\forall p \forall q_1 \forall q_2 (buy(p, q_1) \& produce(p, q_2) \implies procure(p, q_1 + q_2))$
- ☛ $\forall p \forall q (sell(p, q) \implies need(p, q))$
- ☛ $sell(bread, 40)$
- ☛ $buy(bread, 20)$
- ☛ $produce(bread, 10)$
- ☛ $40 > 20 + 10$
- ☛ $\forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(q_1 > q_2))$

Например, за да се построи Табло по частни случаи на тези формули, трябва да вземем следните затворени частни случаи:

- ☛ взимаме следния частен случай на първата формула:
 $buy(bread, 20) \& produce(bread, 10) \implies procure(bread, 20 + 10);$
- ☛ за втората формула взимаме частен случай $sell(bread, 40) \implies need(bread, 40);$
- ☛ а за третата формула използваме
 $need(bread, 40) \& procure(bread, 20 + 10) \implies \neg(40 > 20 + 10).$

Към тези частни случаи добавяме формулите $sell(bread, 40)$, $buy(bread, 20)$, $produce(bread, 10)$ и $40 > 20 + 10$ и по полученото множество от затворени частни случаи построяваме Табло.

Неудобството тук е, че се налага до известна степен да се "налучква" (или да се използват някакви евристични алгоритми, примерно) кои частни случаи на оригиналните универсални формули да вземем, за да построим по тях Таблото.

Заб.: Друго неудобство на този подход е, че ако множеството се окаже изпълнимо, то вече не е толкова лесно да се извлече от Таблото структура, която изпълнява множеството. Причината е, че за да построим изпълняваща структура се налага да разгледаме всички затворени частни случаи на универсалните формули (това се показва в последващите твърдения и Теорема на Ербран), а ако затворените термове в езика са безкрайно много, то и тези случаи стават безкрайно много, а това води до нуждата да се построи безкрайно дърво (т.е. вече не можем да боравим само с крайни структури, което е алгоритмичен проблем).

Означение.

Ако имаме универсална формула $\forall x_1 \forall x_2 \dots \forall x_n \varphi$, тогава с

$$CI(\forall x_1 \forall x_2 \dots \forall x_n \varphi) = \{ \sigma \varphi \mid \text{за някоя субституция } \sigma \text{ такава, че } \sigma \varphi \text{ е затворена формула} \}$$

ще означаваме множеството на всички нейни затворени частни случаи.

Означение.

Ако имаме множество от универсални формули $\{ \varphi_1, \varphi_2, \dots \}$, тогава

$$CI(\{ \varphi_1, \varphi_2, \dots \}) = CI(\varphi_1) \cup CI(\varphi_2) \cup \dots$$

Твърдение 19 (Изпълнимост на множества от универсални формули).

Едно множество от универсални формули $\Phi = \{ \varphi_1, \varphi_2, \dots \}$ е изпълнимо тогава и само тогава, когато $CI(\Phi)$ е изпълнимо.

Доказателство.

В правата посока ще покажем, че ако Φ е изпълнимо, то тогава и $CI(\Phi)$ също е такова. Щом Φ е изпълнимо, значи има структура \mathcal{A} и оценка при нея v такива, че $\mathcal{A}, v \models \varphi_i$ за всяко i . Ще покажем и че $\mathcal{A}, v \models \psi$ за всяко $\psi \in CI(\Phi)$. Така, $CI(\Phi)$ ще е изпълнимо при същите структура и оценка, както и Φ .

Нека имаме произволна $\psi \in CI(\Phi)$. За нея имаме, че $\psi \in CI(\varphi_i)$ за някое i . Т.к. $\mathcal{A}, v \models \varphi_i$, тогава по правилото за извод (σ) имаме и че $\mathcal{A}, v \models \psi$, т.к. ψ е частен случай на φ_i . Това е достатъчно за получаване на нужния резултат.

Сега ще проверим и обратната посока на твърдението. Нека $CI(\Phi)$ е изпълнимо. Тогава по теоремата за коректност и пълнота на Табло метода от предната глава има Табло за $CI(\Phi)$ с непротиворечим клон и по този клон можем да съставим Ербранова структура $\mathcal{H} = (H_L, \dots)$ и оценка v при нея, които изпълняват $CI(\Phi)$. Сега ще покажем, че и Φ се изпълнява от \mathcal{H} и v като докажем, че $\mathcal{H}, v \models \varphi_i$ за всяко i . Нека разгледаме произволно i и нека за него φ_i има вида $\forall x_1 \forall x_2 \dots \forall x_n \chi$. Тогава:

$$\begin{aligned} \mathcal{H}, v \models \psi \text{ за всяко } \psi \in CI(\Phi) &\longrightarrow \mathcal{H}, v \models \psi \text{ за всяко } \psi \in CI(\varphi_i) &\longleftrightarrow \\ \mathcal{H}, v \models \sigma \chi \text{ за всяка субституция } \sigma, \text{ за която } \sigma \chi \text{ е затворена формула} &&\longleftrightarrow \\ \mathcal{H}, v \models \sigma \chi \text{ за всяка субституция } \sigma = \{ x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n \}, && \\ \text{където } \tau_1, \tau_2, \dots, \tau_n \text{ са произволни затворени термове} &&\longleftrightarrow \\ \mathcal{H}, v[x_1 = v(\tau_1), x_2 = v(\tau_2), \dots, x_n = v(\tau_n)] \models \chi, && \\ \text{където } \tau_1, \tau_2, \dots, \tau_n \text{ са произволни затворени термове} &&\longleftrightarrow \end{aligned}$$

$$\begin{aligned}
& \mathcal{H}, v[x_1 = o_{\tau_1}, x_2 = o_{\tau_2}, \dots, x_n = o_{\tau_n}] \models \chi, \\
& \text{където } \tau_1, \tau_2, \dots, \tau_n \text{ са произволни затворени термове} \quad \longleftrightarrow \\
& \mathcal{H}, v[x_1 = o_{\tau_1}, x_2 = o_{\tau_2}, \dots, x_n = o_{\tau_n}] \models \chi, \text{ за произволни } o_{\tau_1}, o_{\tau_2}, \dots, o_{\tau_n} \in H_L \quad \longleftrightarrow \\
& \mathcal{H}, v \models \varphi_i.
\end{aligned}$$



Сега вече имаме и начин за проверяване на изпълнимост или неизпълнимост за множества от универсални формули. Това, за сега, е най-големия клас от формули, който сме разгледали, и включва всички предишни класове - затворени атомарни формули, затворени литерали и затворените безкванторни формули.



Теорема 3 (Теорема на Ербран).

Ако имаме множество от универсални формули $\Phi = \{ \varphi_1, \varphi_2, \dots \}$, тогава някой от следните два случая е в сила:

- ☛ или Φ е изпълнимо чрез Ербранова структура;
- ☛ или Φ е неизпълнимо и по-специално има крайно множество от затворени частни случаи на формули от Φ , което е неизпълнимо.

Доказателство.

Ако Φ е изпълнимо, то тогава по предното твърдение и $CI(\Phi)$ е изпълнимо и Ербрановата структура, която построяваме за да покаже това, изпълнява и множеството Φ . Така, тя е търсената структура.

А в противен случай, ако Φ не е изпълнимо, то тогава пак по предното твърдение и $CI(\Phi)$ е неизпълнимо и според следствието за компактност на неизпълнимостта от предната глава имаме, че има крайно подмножество на $CI(\Phi)$, което е неизпълнимо. Именно това крайно подмножество е търсеното множество от затворени частни случаи. ◆

Задача.

Кои двойки от изброените по-долу формули имат общи затворени частни случаи (т.е. има затворена формула, която е частен случай едновременно и на двете формули)? За тези двойки които имат - дайте по един пример за общ частен случай, като посочите и чрез кои субституции се получава той.

Ето и формулите:

- ☛ $p(h(x), g(x, y));$
- ☛ $p(y, z);$
- ☛ $p(g(x, y), h(x));$
- ☛ $p(h(c), z);$
- ☛ $q(h(x), g(x, y)).$

Тук p и q са предикатни символи, g и h са функционални символи, а c е константа.

Задача.

Намерете поне една Ербранова структура, която изпълнява следното множество от формули:

- ☛ $\forall x \forall y \forall z (p(x, y) \& p(y, z) \implies p(x, z));$
- ☛ $\forall x \forall y (p(x, y) \implies p(f(x), f(y)));$
- ☛ $p(c, f(c)).$

Тук p е предикатен символ, f е функционален символ, а c е константа.

Задача.

Дадено е следното множество от затворени формули

- ☛ $\{ \forall x (p(f(x)) \implies p(f(f(x))));$
- ☛ $p(c), p(f(c)), p(f(d));$
- ☛ $p(f(e)) \};$

където p е предикатен символ, f е функционален символ, а c, d и e са константи. Колко Ербранови структури изпълняват това множество? Избройте ги.

Задача.

Дадено е следното множество от затворени формули

- ☛ $\{ \forall x (p(g(x)) \implies p(g(g(x))));$

☛ $p(c)$, $p(g(c))$;

☛ $p(g(g(d)))$ };

където p е предикатен символ, g е функционален символ, а c и d са константи. Колко Ербранови структури изпълняват това множество? Избройте ги.

❖ Унифициране ❖

Означение.

Ако $S = \{ \tau_1, \tau_2, \dots \}$ е множество от термове, а λ е субституция, тогава λS ще означаваме следното множество

$$\lambda S = \{ \lambda \tau_1, \lambda \tau_2, \dots \}.$$

Също, ако $S = \{ \varphi_1, \varphi_2, \dots \}$ е множество от атомарни формули, тогава λS ще е

$$\lambda S = \{ \lambda \varphi_1, \lambda \varphi_2, \dots \}.$$

Дефиниция 45 (Унифициране).

Ако S е множество от еднотипни изрази (термове или атомарни формули) ще казваме, че S е унифицируемо ако има таква субституция λ , че λS да е синглетон.

В този случай λ ще наричаме унификатор на S .

Примери.

Термове $f(y, g(c))$ и $(x, g(z))$ са унифицируеми чрез субституция $\{ x \rightarrow y, z \rightarrow c \}$. Също може да се използват и различни субституции, като $\{ y \rightarrow x, z \rightarrow c \}$, $\{ x \rightarrow f(c, c), y \rightarrow f(c, c), z \rightarrow c \}$ и др.

Термове $f(y, g(z))$ и $f(y, h(t))$ не са унифицируеми. Също и $f(c, f(z, y))$ и $f(d, f(t, x))$ (тук c и d са константи).

Атомарни формули $p(f(x, h(d)), x)$ и $p(y, c)$ са унифицируеми. Например, чрез субституция $\{ x \rightarrow c, y \rightarrow f(c, h(d)) \}$. Тук друга субституция няма.

А формулите $p(f(x), c)$ и $q(y, z)$ не са унифицируеми.

Дефиниция 46 (Композиция на субституции).

Ако $\sigma_1 = \{ x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n \}$ и $\sigma_2 = \{ y_1 \rightarrow \pi_1, y_2 \rightarrow \pi_2, \dots, y_m \rightarrow \pi_m \}$ са субституции, тогава $\sigma_1 \sigma_2$ е нова субституция, която наричаме композиция на σ_1 и σ_2 , и която дефинираме по следния начин:

$$\sigma_1 \sigma_2 = \{ y_1 \rightarrow \sigma_1 \pi_1, y_2 \rightarrow \sigma_1 \pi_2, \dots, y_m \rightarrow \sigma_1 \pi_m, x_1 \rightarrow \tau_1, x_2 \rightarrow \tau_2, \dots, x_n \rightarrow \tau_n \}.$$

Заб.: Ако за някое i се получи, че $y_i = \sigma_1 \pi_i$, тогава тази двойка се пропуска от крайния резултат. Също се пропуска и двойките $x_i \rightarrow \tau_i$, за които $x_i \in \{ y_1, y_2, \dots, y_m \}$.

Примери.

Ако двете субституции са $\sigma_1 = \{ x \rightarrow f(x), y \rightarrow h(c, z) \}$ и $\sigma_2 = \{ z \rightarrow d, t \rightarrow f(z) \}$, то тогава композицията е $\sigma_1 \sigma_2 = \{ z \rightarrow d, t \rightarrow f(z), x \rightarrow f(x), y \rightarrow h(c, z) \}$.

А когато $\sigma_1 = \{ x \rightarrow f(x), y \rightarrow h(c, z) \}$ и $\sigma_2 = \{ z \rightarrow h(x, f(y)), t \rightarrow f(x) \}$, тогава имаме $\sigma_1 \sigma_2 = \{ z \rightarrow h(f(x), f(h(c, z))), t \rightarrow f(f(x)), x \rightarrow f(x), y \rightarrow h(c, z) \}$.

Дефиниция 47 (Най-общ унификатор).

Нека S е унифицируемо множество от еднотипни изрази (термове или атомарни формули). Тогава казваме, че субституцията μ е най-общ унификатор за S , ако μ е унификатор за S и за всеки друг унификатор λ е изпълнено, че има субституция σ такава, че $\lambda = \sigma\mu$.

Пример.

Нека $S = \{ p(x, t), p(f(y), z), p(z, t), p(f(g(u)), x) \}$ е унифицируемо множество от атомарни формули. Тогава $\sigma = \{ x \rightarrow f(g(u)), z \rightarrow f(g(u)), y \rightarrow g(u), t \rightarrow f(g(u)) \}$ е най-общ унификатор за множеството.

Заб.: Унификатори на това множество, които не са най-общи, са например субституцията $\sigma' = \{ x \rightarrow f(g(c)), z \rightarrow f(g(c)), y \rightarrow g(c), t \rightarrow f(g(c)) \}$ или пък, алтернативно, тази субституция $\sigma'' = \{ x \rightarrow f(g(f(c))), z \rightarrow f(g(f(c))), y \rightarrow g(f(c)), t \rightarrow f(g(f(c))) \}$. Най-общия унификатор може да се окаже по-полезен от тези, които не са най-общи, ако има нужда да се прилагат и други субституции върху резултата от прилагането на унификатора върху множеството S . Например, $\sigma S = \{ p(f(g(u)), f(g(u))) \}$ и така в σS има поне една променлива - u - която все още може да се заменя от последващи субституции. А в $\sigma' S = \{ p(f(g(c)), f(g(c))) \}$ и $\sigma'' S = \{ p(f(g(f(c))), f(g(f(c)))) \}$ няма оставащи променливи и това предотвратява прилагането на други субституции.

Алгоритъм (Алгоритъм за намиране на най-общ унификатор).

Нека S е множество от еднотипни изрази (термове или атомарни формули). Тогава за да определим, дали S е унифицируемо или не и ако е, да намерим най-общия му унификатор, прилагаме следните стъпки:

- ☛ Стъпка 1: Задаваме $S_1 = S$ и $\mu_1 = \{\}$.
- ☛ Стъпка $k + 1$: Ако S_k е синглетон, тогава приключваме алгоритъма с резултата, че S е унифицируемо и μ_k е най-общия му унификатор. В противен случай, търсим най-малкото i такова, че има $s_1, s_2 \in S_k$ и на позиция i в s_1 и s_2 има различни символи. Проверяваме, дали на позиция i в единия от двата изрази има променлива x , а на позиция i в другия израз има терм τ , който не съдържа x . Ако това не е изпълнено, тогава приключваме алгоритъма с резултата, че S не е унифицируемо. Иначе, задаваме $\sigma_{k+1} = \{ x \rightarrow \tau \}$, $S_{k+1} = \sigma_{k+1} S_k$ и $\mu_{k+1} = \sigma_{k+1} \mu_k$ и продължаваме със стъпка $k + 2$.

Твърдение 20 (Коректност на алгоритъма за намиране на най-общ унификатор).

За произволно крайно множество S от еднотипни изрази предния алгоритъм завършва за краен брой стъпки и дава резултат μ тогава и само тогава, когато множеството е унифицируемо. В случай, че имаме резултат μ , то тогава той е най-общия унификатор на S и още повече за него е изпълнено, че за всеки друг унификатор λ на S е в сила $\lambda = \lambda\mu$.

Доказателство.


Първо, нека отбележим, че ако алгоритъма завърши с резултат, че множеството S не е унифицируемо, то това наистина е така. Това означава, че на някоя стъпка $k + 1$, S_k не е синглетон и като фиксираме въпросната най-малка позиция i на различие на два изрази от S_k , то на тази позиция или в двата изрази има различни от променлива символи, или в единия израз има променлива x , а в другия има терм τ , който съдържа x .

В първия случай имаме, че на позиция i в двата изрази има или предикатни или функционални символи. В този случай никоя субституция не може да оеднакви двата изрази (а като резултат не може да получи синглетон от S_k), т.к. субституциите не могат да променят предикатни и функционални символи. А в другия случай, за да можем да оеднаквим двата изрази, ще трябва да използваме замяна $x \rightarrow \tau$. Но тогава, т.к. τ съдържа x , то замяната ще промени и самото τ и дори и след замяната няма да имаме еднакви резултати за двата изрази. Този процес на рекурсивна замяна ще продължи без край и никога няма да може да се оеднаквят изразите от S_k .

Така получихме, че S_k не е унифицируемо. От това следва и неунифицируемостта на S , т.к. изразите в S_k са частни случаи на тези от S и ако S имаше унификатор, то той щеше да е унификатор и за S_k .

А когато алгоритъмът завърши с резултат, че S е унифицируемо, то получената субституция μ_k наистина е унификатор на S . Това се дължи на факта $S_k = \sigma_k \sigma_{k-1} \dots \sigma_1 S = \mu_k S$ и че S_k е синглетон.

Остава да покажем, че μ_k е най-общия унификатор на S . Това ще следва от твърдението, че $\lambda = \lambda \mu_k$ за всеки унификатор λ на S . Така, нека λ е произволен унификатор на S . $\lambda = \lambda \mu_k$ ще докажем с индукция по броя на стъпките k . На първа стъпка имаме $\mu_1 = \{\}$ и така $\lambda = \lambda \mu_1$. Сега нека допуснем, че сме доказали твърдението за $k-1$ стъпки. Т.е. $\lambda = \lambda \sigma_{k-1} \sigma_{k-2} \dots \sigma_1$. Ако покажем, че $\lambda = \lambda \sigma_k$, то от това ще следва нужното $\lambda = \lambda \sigma_k \sigma_{k-1} \dots \sigma_1$.

За да получим, че $\lambda = \lambda \sigma_k$ ще проверим, че λ и $\lambda \sigma_k$ имат едно и също действие върху всяка една променлива. Нека $\sigma_k = \{x \rightarrow \tau\}$. Ако разгледаме променлива y , която е различна от x , то понеже σ_k не променя y то λy и $\lambda \sigma_k y$ съвпадат. Остава да разгледаме случая, когато променливата е x . Т.к. $\lambda = \lambda \sigma_{k-1} \sigma_{k-2} \dots \sigma_1$ то следва, че $\lambda S = \lambda \sigma_{k-1} \sigma_{k-2} \dots \sigma_1 S = \lambda S_{k-1}$ и така излиза, че λ унифицира и множеството S_{k-1} . А понеже x и τ се намират на най-малката позиция на различие между два елемента на S_{k-1} и λ унифицира тези два елемента, то λ ще унифицира и x и τ . Така получаваме, че $\lambda \sigma_k x$ което след прилагане на σ_k се променя в $\lambda \tau$ съвпада с λx . 

Дефиниция 48 (Литерали с различен знак).

За двойка литерали L_1 и L_2 ще казваме, че са с различен знак, ако единият от тях е атомарна формула, а другият е отрицание на (потенциално различна) атомарна формула.

Дефиниция 49 (Унифициране на литерали с различен знак).

За двойка литерали L_1 и L_2 с различен знак ще казваме, че са унифицируеми, ако единия от тях е атомарната формула φ_1 , другия е отрицание на атомарната формула φ_2 и множеството $\{\varphi_1, \varphi_2\}$ е унифицируемо.

Дефиниция 50 (Табло с унифициране за формула).

Нека φ е универсална формула. Тогава Табло с унифициране за φ ще построяваме по същия начин, по който се строи Табло за затворена безкванторна формула, но със следните две специфични разлики:

- ☛ в корена на дървото поставяме подходящ частен случай на формулата чрез правилото (σ) (дори ако σ е тривиалната празна субституция, тогава в корена просто можем да поставим безкванторната част на φ);

- ☛ също ще променим дефиницията за противоречив клон в дървото: един клон ще е противоречив в Табло с унифициране ако в него има два литерала с различен знак, които са унифицируеми, а иначе е непротиворечив.

Вече чрез използването на унифициране отпада нуждата да се “налучкват” частните случаи на началните формули, по които да се строи Табло. Все пак остава неудобството, че метода не може да се използва ефективно за намиране на Ербранови структури, които удовлетворяват изпълними множества.

Дефиниция 51 (Табло с унифициране за множество от формули).

Ако имаме множество от универсални формули $\{ \varphi_1, \varphi_2, \dots \}$ тогава Табло с унифициране за това множество построяваме като първо съставяме единствен клон с по един възел за всяка от формулите. В първия възел записваме подходящ частен случай на φ_1 , във втория записваме частен случай на φ_2 и т.н. Частните случаи подбираме така, че да няма общи променливи в тези начални възли на дървото. Накрая завършваме получения клон по дефиницията за построение на Табло с унифициране.

Пример.

Чрез построяване на Табло с унифициране можем да покажем, че множеството от следните формули $\forall x p(x), \forall x (q(x) \implies \neg p(f(x)) \vee \neg p(f(f(x))))$, $q(c)$ е неизпълнимо.

Пример (Снабдяване при търговия).

Също чрез Табло с унифициране можем да покажем, че следните универсални формули от примера за снабдяване при търговия са неизпълнимо множество

- ☛ $\forall p \forall q_1 \forall q_2 (buy(p, q_1) \& produce(p, q_2) \implies procure(p, q_1 + q_2))$
- ☛ $\forall p \forall q (sell(p, q) \implies need(p, q))$
- ☛ $sell(bread, 40)$
- ☛ $buy(bread, 20)$
- ☛ $produce(bread, 10)$
- ☛ $40 > 20 + 10$
- ☛ $\forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(q_1 > q_2))$

Заб.: Може да се докаже, че едно множество от универсални формули е неизпълнимо тогава и само тогава, когато в Таблото с унифициране за това множество всички клонове са противоречиви. Аналогичен резултат ще докажем в следващите глави за метода на резолюцията.

Задача.

Нека φ_1 и φ_2 са атомарни формули, а освен това $\sigma_1 = \{ x \rightarrow h(c), y \rightarrow h(d), z \rightarrow d, t \rightarrow c \}$ и $\sigma_2 = \{ x \rightarrow f(c), y \rightarrow h(c), z \rightarrow d, t \rightarrow c \}$ са субституции. Нека също имаме, че $\sigma_1 \varphi_1$ е $p(h(c), f(h(d)))$ и $\sigma_2 \varphi_2$ е $p(f(c), f(h(c)))$.

Възможно ли е φ_1 и φ_2 да са унифицируеми? Дејте пример за това.

Възможно ли е φ_1 и φ_2 да не са унифицируеми? Дејте пример за това.

Тук p е предикатен символ, f и h са функционални символи, а c и d са константи.

Задача.

Да се докаже, че две атомарни формули φ_1 и φ_2 са унифицируеми тогава и само тогава, когато $CI(\varphi_1) \cap CI(\varphi_2) \neq \emptyset$.

Задача.

Използвайте Табло метода за да определите дали следното множество от формули е изпълнимо или не:

- ☛ $\forall x(p(x) \iff q(f(x), g(x)))$;
- ☛ $\forall z(q(f(h(c)), z) \implies r(z))$;
- ☛ $p(h(c))$;
- ☛ $\forall x \neg r(g(x))$.

Задача.

Използвайте Табло метода за да определите дали следното множество от формули е изпълнимо или не:

- ☛ $r(h(d))$;
- ☛ $\forall x(p(x, f(h(d))) \implies q(x))$;
- ☛ $\forall x(r(x) \iff p(g(x), f(x)))$;
- ☛ $\forall y \neg q(g(y))$.

❖ Пренексна и Скулемова нормални форми ❖

Твърдение 21 (Еквивалентни преобразувания за квантори).
Следните преобразувания са еквивалентни:

$$\neg \forall x \alpha \equiv \exists x \neg \alpha \quad (\neg \forall)$$

$$\neg \exists x \alpha \equiv \forall x \neg \alpha \quad (\neg \exists)$$

Също и тези преобразувания са еквивалентни, но само при условие, че x не се среща свободно във формула β :

$$\forall x \alpha \& \beta \equiv \forall x (\alpha \& \beta) \quad (\forall \&)$$

$$\forall x \alpha \vee \beta \equiv \forall x (\alpha \vee \beta) \quad (\forall \vee)$$

$$\exists x \alpha \& \beta \equiv \exists x (\alpha \& \beta) \quad (\exists \&)$$

$$\exists x \alpha \vee \beta \equiv \exists x (\alpha \vee \beta) \quad (\exists \vee)$$

Доказателство.

Всяко от преобразуванията ще докажем според дефиницията за еквивалентно преобразуване: за всяка структура $\mathcal{A} = (A, \dots)$ и всяка оценка v при тази структура ще проверим дали двете страни са еднакво вярни.

За преобразуване $(\neg \forall)$ имаме:

$$\begin{aligned} \mathcal{A}, v \models \neg \forall x \alpha &\iff \mathcal{A}, v \not\models \forall x \alpha &\iff \mathcal{A}, v[x = a] \not\models \alpha \text{ за поне едно } a \in A &\iff \\ \mathcal{A}, v[x = a] \models \neg \alpha \text{ за поне едно } a \in A &\iff \mathcal{A}, v \models \exists x \neg \alpha. \end{aligned}$$

За преобразуване $(\neg \exists)$ се получава:

$$\begin{aligned} \mathcal{A}, v \models \neg \exists x \alpha &\iff \mathcal{A}, v \not\models \exists x \alpha &\iff \mathcal{A}, v[x = a] \not\models \alpha \text{ за всяко едно } a \in A &\iff \\ \mathcal{A}, v[x = a] \models \neg \alpha \text{ за всяко едно } a \in A &\iff \mathcal{A}, v \models \forall x \neg \alpha. \end{aligned}$$

При доказването на останалите правила ще разчитаме, че x не се среща свободно в β .

Преобразуване $(\forall \&)$ се показва така:

$$\begin{aligned} \mathcal{A}, v \models \forall x \alpha \& \beta &\iff \mathcal{A}, v \models \forall x \alpha \text{ и } \mathcal{A}, v \models \beta &\iff \\ \mathcal{A}, v[x = a] \models \alpha \text{ за всяко едно } a \in A \text{ и } \mathcal{A}, v \models \beta &\iff \\ \mathcal{A}, v[x = a] \models \alpha \text{ за всяко едно } a \in A \text{ и } \mathcal{A}, v[x = a] \models \beta \text{ за всяко едно } a \in A &\iff \\ \mathcal{A}, v[x = a] \models \alpha \text{ и } \mathcal{A}, v[x = a] \models \beta \text{ за всяко едно } a \in A &\iff \\ \mathcal{A}, v[x = a] \models \alpha \& \beta \text{ за всяко едно } a \in A &\iff \mathcal{A}, v \models \forall x (\alpha \& \beta). \end{aligned}$$

Преобразуване $(\forall \vee)$:

$$\begin{aligned} \mathcal{A}, v \models \forall x \alpha \vee \beta &\iff \mathcal{A}, v \models \forall x \alpha \text{ или } \mathcal{A}, v \models \beta &\iff \\ \mathcal{A}, v[x = a] \models \alpha \text{ за всяко едно } a \in A \text{ или } \mathcal{A}, v \models \beta &\iff \\ \mathcal{A}, v[x = a] \models \alpha \text{ за всяко едно } a \in A \text{ или } \mathcal{A}, v[x = a] \models \beta \text{ за всяко едно } a \in A &\iff \\ \mathcal{A}, v[x = a] \models \alpha \text{ или } \mathcal{A}, v[x = a] \models \beta \text{ за всяко едно } a \in A &\iff \\ \mathcal{A}, v[x = a] \models \alpha \vee \beta \text{ за всяко едно } a \in A &\iff \mathcal{A}, v \models \forall x (\alpha \vee \beta). \end{aligned}$$

За реобразуване ($\exists \&$) имаме:

$$\begin{aligned}
\mathcal{A}, v \models \exists x \alpha \& \beta &\longleftrightarrow \mathcal{A}, v \models \exists x \alpha \text{ и } \mathcal{A}, v \models \beta &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \text{ за поне едно } a \in A \text{ и } \mathcal{A}, v \models \beta &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \text{ за поне едно } a \in A \text{ и } \mathcal{A}, v[x=a] \models \beta \text{ за същото } a &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \text{ и } \mathcal{A}, v[x=a] \models \beta \text{ за поне едно } a \in A &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \& \beta \text{ за поне едно } a \in A &\longleftrightarrow \mathcal{A}, v \models \exists x(\alpha \& \beta).
\end{aligned}$$

Преобразуване ($\exists \vee$):

$$\begin{aligned}
\mathcal{A}, v \models \exists x \alpha \vee \beta &\longleftrightarrow \mathcal{A}, v \models \exists x \alpha \text{ или } \mathcal{A}, v \models \beta &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \text{ за поне едно } a \in A \text{ или } \mathcal{A}, v \models \beta &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \text{ за поне едно } a \in A \text{ или } \mathcal{A}, v[x=a] \models \beta \text{ за поне едно } a \in A &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \text{ или } \mathcal{A}, v[x=a] \models \beta \text{ за поне едно } a \in A &\longleftrightarrow \\
\mathcal{A}, v[x=a] \models \alpha \vee \beta \text{ за поне едно } a \in A &\longleftrightarrow \mathcal{A}, v \models \exists x(\alpha \vee \beta).
\end{aligned}$$



Дефиниция 52 (Вариант на формула).

Нека имаме формула от вида $Qx\varphi$ (където Q е квантор) и индивидуна променлива y , която не се среща във φ . Нека имаме и субституцията $\sigma = \{x \rightarrow y\}$. Тогава казваме, че $Qy\sigma\varphi$ е вариант на първоначалната формула $Qx\varphi$.

Примери.

$\forall y p(y)$ е вариант на формулата $\forall x p(x)$.

Формули $\forall z \exists y(p(z, y) \& p(y, z))$, $\forall x \exists z(p(x, z) \& p(z, x))$ и $\forall z \exists t(p(z, t) \& p(t, z))$ са варианти на формулата $\forall x \exists y(p(x, y) \& p(y, x))$.

Също $\forall x(p(x) \implies \exists z q(z, x))$ е вариант на $\forall x(p(x) \implies \exists y q(y, x))$.

Твърдение 22 (Еквивалентност на вариантите).

За произволна формула е изпълнено, че нейните варианти са еквивалентни. Т.е. ако имаме $\sigma = \{x \rightarrow y\}$ то

$$\forall x \alpha \equiv \forall y \sigma \alpha \quad (\forall)$$

$$\exists x \alpha \equiv \exists y \sigma \alpha \quad (\exists)$$

Доказателство.

Отново правим проверка за всяка структура $\mathcal{A} = (A, \dots)$ и оценка v .

Твърдението (\forall) се доказва чрез:

$$\begin{aligned}
\mathcal{A}, v \models \forall x \alpha &\longleftrightarrow \mathcal{A}, v[x=a] \models \alpha \text{ за всяко едно } a \in A &\longleftrightarrow \\
\mathcal{A}, v[y=a] \models \sigma \alpha \text{ за всяко едно } a \in A &\longleftrightarrow \mathcal{A}, v \models \forall y \sigma \alpha.
\end{aligned}$$

$A (\exists)$ се получава така:

$$\begin{aligned} \mathcal{A}, v \models \exists x \alpha &\longleftrightarrow \mathcal{A}, v[x = a] \models \alpha \text{ за поне едно } a \in A &\longleftrightarrow \\ \mathcal{A}, v[y = a] \models \sigma \alpha \text{ за поне едно } a \in A &\longleftrightarrow \mathcal{A}, v \models \exists y \sigma \alpha. \end{aligned}$$



Алгоритъм (Алгоритъм за привеждане в пренексна нормална форма).

Произволна формула можем да приведем в пренексна нормална форма чрез извършване на следните действия върху нея или върху нейни подформули:

- ☛ ако (под)формулата е от вида $\neg \forall x \alpha$ или $\neg \exists x \alpha$, тогава прилагаме преобразуване $(\neg \forall)$ или $(\neg \exists)$, съответно;
- ☛ ако (под)формулата е от вида $\forall x \alpha \& \beta$, $\forall x \alpha \vee \beta$, $\exists x \alpha \& \beta$ или $\exists x \alpha \vee \beta$, тогава прилагаме преобразуване $(\forall \&)$, $(\forall \vee)$, $(\exists \&)$ или $(\exists \vee)$, съответно; ако се окаже, че x се среща свободно в β , тогава подбираме подходящ вариант на α за да отстраним проблема;
- ☛ ако (под)формулата е от вида $\alpha \implies \beta$ или $\alpha \iff \beta$, тогава прилагаме преобразуване (\implies) или (\iff) , съответно.

Пример (Снабдяване при търговия).

Отрицанията на двата въпроса от примера за снабдяване при търговия

- ☛ $\neg(\exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& q_1 > q_2))$
- ☛ $\neg(\exists p \exists q_1 \exists q_2 (need(p, q_1) \& procure(p, q_2) \& k.q_1 < q_2))$

могат еквивалентно да се преобразуват във формулите

- ☛ $\forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(q_1 > q_2))$
- ☛ $\forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(k.q_1 < q_2))$

Алгоритъм (Алгоритъм за Скулемизация).

За всяка формула в пренексна нормална форма можем да извършваме следните две операции:

- ☛ ако формулата е от вида $\exists x \varphi$, тогава можем да премахнем квантора $\exists x$ и в останалата част от формулата да приложим субституцията $\sigma = \{ x \rightarrow c \}$, където c е изцяло нова константа (т.е. не е била използвана до сега за текущата задача); субституцията се прилага само на местата, където променливата x е останала със свободно срещане; така, резултатът е $\sigma \varphi$;
- ☛ ако формулата е от вида $\forall x_1 \forall x_2 \dots \forall x_n \exists x \varphi$, тогава можем да премахнем $\exists x$ и в останалата част от формулата да приложим субституцията $\sigma = \{ x \rightarrow f(x_1, x_2, \dots, x_n) \}$, където f е изцяло нов функционален символ (т.е. не е бил използван до сега за текущата задача); субституцията се прилага само на местата, където променливата x е останала със свободно срещане; така, резултатът е $\forall x_1 \forall x_2 \dots \forall x_n \sigma \varphi$.

Извършването на краен брой операции от този тип върху формула наричаме Скулемизирание.

Примери.

Формулата $\exists x \exists y (p(x, x) \iff q(x, y))$ се Скулемизира като $p(c, c) \iff q(c, d)$.

$\forall y \forall z (p(d, f(c)) \& \neg r(g(y, z)) \vee p(z, f(y)))$ е Скулемизацията на $\exists x \forall y \forall z \exists t (p(x, f(c)) \& \neg r(t) \vee p(z, f(y)))$.

Дефиниция 53 (Скулемова нормална форма).

За дадена универсална формула казваме, че е в Скулемова нормална форма, ако е получена от (потенциално) друга формула първо чрез привеждането ѝ в пренексна нормална форма и след това чрез употреба на двете действия от предния алгоритъм.

Твърдение 23 (Запазване на изпълнимостта на формула при Скулемизация).

За всяка затворена формула φ е изпълнено, че множеството $\{\varphi\}$ е изпълнимо тогава и само тогава, когато $\{\psi\}$ е изпълнимо, където ψ е Скулемовата нормална форма на φ .

Доказателство.

Твърдението ще докажем като демонстрираме, че всяка една от двете операции при Скулемизирание не нарушава изпълнимостта на множеството.

Първо ще видим, че ако φ е от вида $\exists x \chi$ и ψ е получена от φ чрез първата операция, то $\{\varphi\}$ е изпълнимо тогава и само тогава, когато $\{\psi\}$ е изпълнимо.

В правата посока имаме, че ако $\{\varphi\}$ е изпълнимо, следователно има структура $\mathcal{A} = (A, \dots)$ и оценка при нея v такива, че $\mathcal{A}, v \models \varphi$. Това означава, че има елемент $a \in A$ такъв, че $\mathcal{A}, v[x = a] \models \chi$. Тогава можем да дефинираме нова структура $\mathcal{A}^+ = (A, \dots, c)$, в която новата константа c ще интерпретираме така: $c^{\mathcal{A}^+} = a$. Тогава имаме, че $\mathcal{A}^+, v \models \psi$ и така $\{\psi\}$ е изпълнимо.

За обратната посока нека допуснем, че $\{\psi\}$ е изпълнимо. Т.е. има структура $\mathcal{A} = (A, \dots, c)$ и оценка v , за които $\mathcal{A}, v \models \psi$. Тогава щом ψ е вярна при някаква интерпретация на константата c , следователно има стойност $a \in A$ (и по специално $a = c^{\mathcal{A}}$) за която $\mathcal{A}, v[x = a] \models \chi$. Така, $\mathcal{A}, v \models \varphi$ и $\{\varphi\}$ също е изпълнимо.

За втората операция имаме подобна задача: ако φ е от вида $\forall x_1 \forall x_2 \dots \forall x_n \exists x \chi$ и ψ е получена от φ чрез съответната операция, то $\{\varphi\}$ е изпълнимо тогава и само тогава, когато $\{\psi\}$ е изпълнимо.

В едната посока, ако $\{\varphi\}$ е изпълнимо, то има структура $\mathcal{A} = (A, \dots)$ и оценка v и $\mathcal{A}, v \models \varphi$. Така за всеки $a_1, a_2, \dots, a_n \in A$: $\mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n] \models \exists x \chi$. Т.е. за всеки $a_1, a_2, \dots, a_n \in A$ има $b \in A$ такова, че $\mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n, x = b] \models \chi$. Така, можем да разширим структурата до $\mathcal{A}^+ = (A, \dots, f)$ и в новата структура да интерпретираме f за всеки $a_1, a_2, \dots, a_n \in A$ по следния начин:

$$f^{\mathcal{A}^+}(a_1, a_2, \dots, a_n) = b, \text{ където } b \text{ е елемента, съответстващ на текущите } a_1, a_2, \dots, a_n.$$

Тогава $\mathcal{A}^+, v \models \psi$ и така $\{\psi\}$ е изпълнимо.

За другата посока, ако $\{ \psi \}$ е изпълнимо, то има $\mathcal{A} = (A, \dots, f)$ и оценка v изпълняващи ψ .
 Така за всеки $a_1, a_2, \dots, a_n \in A$ можем да намерим поне едно $b \in A$ такива, че е изпълнено $\mathcal{A}, v[x_1 = a_1, x_2 = a_2, \dots, x_n = a_n, x = b] \models \chi$ (това се гарантира от интерпретацията на f в \mathcal{A}).
 Като резултат получаваме, че $\mathcal{A}, v \models \varphi$ и така $\{ \varphi \}$ е изпълнимо. ♦

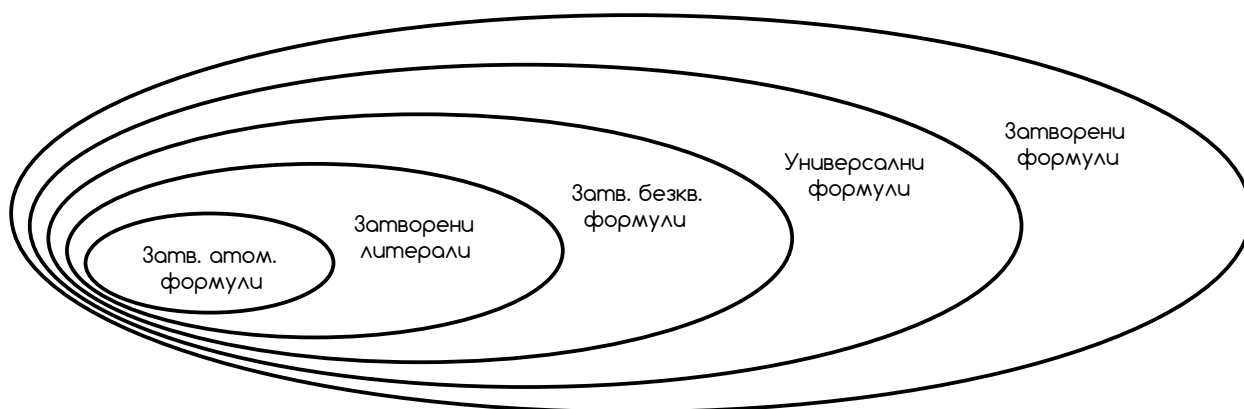
Заб.: Скулемизирването запазва изпълнимостта на формулите, но може да не запазва други техни свойства (т.е. не е еквивалентно преобразуване).

Следствие 7 (Запазване на изпълнимостта на множество при Скулемизация).

За всяко множеството от затворени формули $\{ \varphi_1, \varphi_2, \dots \}$ е изпълнено, че то е изпълнимо тогава и само тогава, когато $\{ \psi_1, \psi_2, \dots \}$ е изпълнимо, където ψ_i е Скулемовата нормална форма на φ_i за всяко i .

Съответно, по този начин задачите за изпълнимост и неизпълнимост на множество от произволни затворени формули могат да се свеждат до задачи за изпълнимост или съответно неизпълнимост на множество от универсални формули.

Така вече можем да разширим класификацията на формулите и с най-общия клас от затворени формули, като за всеки от класовете имаме методи за решаване на задачи за изпълнимост и неизпълнимост.



❖ Изпълнение на Пролог програми с Табло с унифициране ❖

Разширяването на Табло метода до Табло с унифициране позволява да дадем още един метод за определяне, дали дадена цел се удовлетворява при дадена програма на Пролог. Този начин вече разчита на класическата интерпретация на програмни клаузи и цел като формули от първи ред.

Ако използваме класически програми на Пролог (без отрицание и дизюнкция), програмните клаузи и отрицанието на целта съответстват директно на универсални формули - универсалното затваряне на съответния Хорнов дизюнкт или отрицанието на екзистенциалното затваряне на целта.

Алгоритъм (Изпълняване чрез Табло с унифициране).

Взимат се формулите, съответстващи на клаузите на програмата, и отрицанието на формулата, съответстваща на целта. Всяка от формулите от полученото множество се привежда в пренексна нормална форма и след това се Скулемизира и от получените универсални формули се построява Табло с унифициране. Ако всички клонове в дървото са противоречиви, тогава целта се удовлетворява от програмата. В противен случай целта не се удовлетворява.

Пример.

Нека имаме следната програма на Пролог:

$p(X) :- q(X).$
 $q(X) :- r(X).$
 $q(10).$
 $r(2).$
 $r(7).$

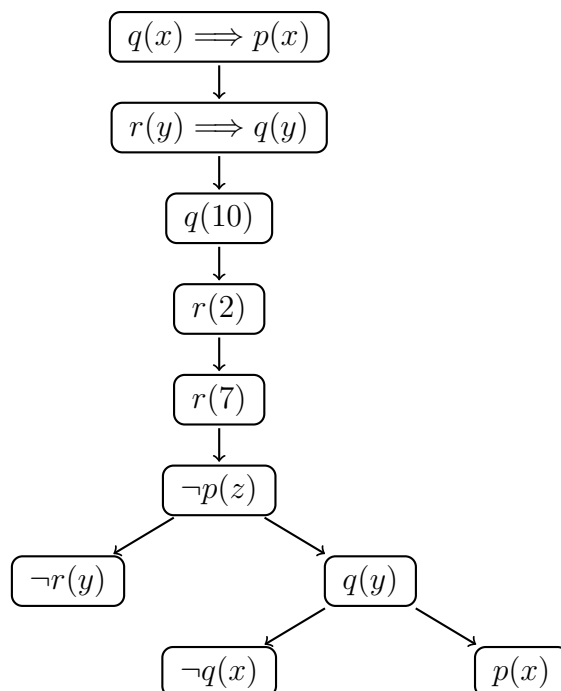
и цел към нея

$?- p(X).$

От програмните формули и от отрицанието на целта се получават следните универсални формули, от които взимаме подходящи частни случаи без общи променливи:

$p(X) :- q(X).$	$\rightarrow \forall x(q(x) \Rightarrow p(x))$	$\rightarrow q(x) \Rightarrow p(x)$
$q(X) :- r(X).$	$\rightarrow \forall x(r(x) \Rightarrow q(x))$	$\rightarrow r(y) \Rightarrow q(y)$
$q(10).$	$\rightarrow q(10)$	$\rightarrow q(10)$
$r(2).$	$\rightarrow r(2)$	$\rightarrow r(2)$
$r(7).$	$\rightarrow r(7)$	$\rightarrow r(7)$
$?- p(X).$	$\rightarrow \neg \exists x p(x)$	$\equiv \forall x \neg p(x) \rightarrow \neg p(z)$

Така от формулите $q(x) \Rightarrow p(x)$, $r(y) \Rightarrow q(y)$, $q(10)$, $r(2)$, $r(7)$ и $\neg p(z)$ построяваме следното Табло



Видно е, че в него и трите клона са противоречиви. Например, най-десният клон е противоречив заради съдържането на $\neg p(z)$ и $p(x)$. Противоречието става чрез унифицирането $z = x$. Само че, за другите два клона има няколко различни варианта, по които може да се получат противоречията:

1. В левия клон унифицираме $r(2)$ и $\neg r(y)$ чрез $y = 2$, а в средния $q(10)$ и $\neg q(x)$ чрез $x = 10$. Така чрез комбиниране на $z = x$ и $x = 10$ можем да получим $z = 10$.
2. В левия клон унифицираме $r(2)$ и $\neg r(y)$ чрез $y = 2$, а в средния $q(y)$ и $\neg q(x)$ чрез $x = y$. Така чрез комбиниране на $z = x$, $y = 2$ и $x = y$ можем да получим $z = 2$.
3. В левия клон унифицираме $r(7)$ и $\neg r(y)$ чрез $y = 7$, а в средния $q(10)$ и $\neg q(x)$ чрез $x = 10$. Така от $z = x$ и $x = 10$ имаме $z = 10$.
4. В левия клон унифицираме $r(7)$ и $\neg r(y)$ чрез $y = 7$, а в средния $q(y)$ и $\neg q(x)$ чрез $x = y$. От $z = x$, $y = 7$ и $x = y$ получаваме $z = 7$.

Заб.: Това дърво доста прилича на дървото, което се получава при проследяване на работата на Пролог програма с обратен извод. Но има и съществени разлики - тук може да има само по една формула във всеки възел на дървото, а също се използват понятията за противоречиви и непротиворечиви клонове (вместо клоновете винаги да се продължат до достигане на празен възел).

❖ Изпълнимост на множества от дизюнкти ❖

Програмните клаузи и отрицанието на целта от една класическа програма на Пролог (без отрицание и дизюнкция) съответстват на дизюнкти. При наличие на отрицание и дизюнкция, обаче, от тях вече може да не се получава директно дизюнкт. Тогава ще има нужда от преработване на формулите до Скулемова нормална форма и след това привеждане на безкванторната им част в конюнктивна нормална форма. От елементарните дизюнкции могат да се образуват нови отделни клаузи. Така можем да считаме, че всяка програма на пролог и цел могат да се сведат до множество от дизюнкти.

Дефиниция 54 (Конюнктивна нормална форма).

За една безкванторна формула ще казваме, че е в конюнктивна нормална форма ако тя представлява конюнкция от елементарни дизюнкции (така, във формулата не се срещат импликация и еквиваленция).

Примери.

$(p(f(c), z) \vee q(y) \vee q(f(y))) \& (r(c) \vee \neg p(c, f(c))) \& p(x, y) \& (\neg q(z) \vee r(f(t)))$ е безкванторна формула в конюнктивна нормална форма.

А следните формули не са в конюнктивна нормална форма: $(p(x) \vee \neg p(y)) \& \neg(\neg q(z) \vee q(x))$, $p(x) \implies q(x)$, $(p(f(f(d))) \& p(y)) \vee (\neg p(x) \& \neg r(f(c)))$.

Твърдение 24 (Дистрибутивни закони).

Следните преобразувания са еквивалентни:

$$\begin{aligned}\alpha \vee (\beta \& \gamma) &\equiv (\alpha \vee \beta) \& (\alpha \vee \gamma) && (\vee \&) \\ \alpha \& (\beta \vee \gamma) &\equiv (\alpha \& \beta) \vee (\alpha \& \gamma) && (\& \vee)\end{aligned}$$

Доказателство.

Дистрибутивните закони ще проверим за всяка структура \mathcal{A} и всяка оценка v според дефиницията за еквивалентно преобразуване.

Първия закон $(\vee \&)$ се показва така:

$$\begin{aligned}\mathcal{A}, v \models \alpha \vee (\beta \& \gamma) &\iff \mathcal{A}, v \models \alpha \text{ или } \mathcal{A}, v \models \beta \& \gamma &\iff \\ \text{ако } \mathcal{A}, v \not\models \alpha \text{ то } \mathcal{A}, v \models \beta \& \gamma &\iff \text{ако } \mathcal{A}, v \not\models \alpha \text{ то } \mathcal{A}, v \models \beta \text{ и } \mathcal{A}, v \models \gamma &\iff \\ \text{ако } \mathcal{A}, v \not\models \alpha \text{ то } \mathcal{A}, v \models \beta \text{ и също ако } \mathcal{A}, v \not\models \alpha \text{ то } \mathcal{A}, v \models \gamma &\iff \\ \mathcal{A}, v \models \alpha \text{ или } \mathcal{A}, v \models \beta \text{ и също } \mathcal{A}, v \models \alpha \text{ или } \mathcal{A}, v \models \gamma &\iff \\ \mathcal{A}, v \models \alpha \vee \beta \text{ и } \mathcal{A}, v \models \alpha \vee \gamma &\iff \mathcal{A}, v \models (\alpha \vee \beta) \& (\alpha \vee \gamma).\end{aligned}$$

За закон $(\& \vee)$ имаме:

$$\begin{aligned}\mathcal{A}, v \models \alpha \& (\beta \vee \gamma) &\iff \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \beta \vee \gamma &\iff \\ \mathcal{A}, v \models \alpha \text{ и или } \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \models \gamma &\iff \\ \text{или } \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \beta \text{ или } \mathcal{A}, v \models \alpha \text{ и } \mathcal{A}, v \models \gamma &\iff \\ \mathcal{A}, v \models \alpha \& \beta \text{ или } \mathcal{A}, v \models \alpha \& \gamma &\iff \mathcal{A}, v \models (\alpha \& \beta) \vee (\alpha \& \gamma).\end{aligned}$$



Алгоритъм (Алгоритъм за привеждане в конюнктивна нормална форма).

Произволна безкванторна формула можем да приведем в конюнктивна нормална форма чрез извършване на следните действия върху нея или върху нейни подформули:

- ако (под)формулата е от вида $\alpha \Rightarrow \beta$ или $\alpha \Leftrightarrow \beta$, тогава прилагаме преобразуване (\Rightarrow) или (\Leftrightarrow), съответно;
- ако (под)формулата е от вида $\neg\neg\alpha$, $\neg(\alpha \& \beta)$ или $\neg(\alpha \vee \beta)$, тогава прилагаме преобразуване ($\neg\neg$), ($\neg\&$) или ($\neg\vee$), съответно;
- ако (под)формулата е от вида $\alpha \vee (\beta \& \gamma)$ или $(\alpha \& \beta) \vee (\alpha \& \gamma)$, тогава прилагаме преобразуване ($\vee\&$) или ($\&\vee$), съответно.

Дефиниция 55 (Изпълнимост на дизюнкти).

За един дизюнкт $\delta = \{ L_1, L_2, \dots, L_n \}$ ще казваме, че е изпълним ако има структура \mathcal{A} и оценка v при нея такива, че универсалното затваряне на поне един литерал от δ е изпълнено при тях (т.е. $\mathcal{A}, v \models UC(L_i)$ за някое i). Това ще бележим така:

$$\mathcal{A}, v \models \delta.$$

Също, за едно множество от дизюнкти $\Delta = \{ \delta_1, \delta_2, \dots \}$ ще казваме, че е изпълнимо ако има структура \mathcal{A} и оценка v такива, че

$$\mathcal{A}, v \models \delta_i, \text{ за всяко } i.$$

Заб.: Така, може да се счита, че една безкванторна формула в конюнктивна нормална форма всъщност представлява конюнкция от дизюнкти. Дизюнктите в нея съответстват на елементарни дизюнкции.

Също по дефиницията може да се види, че празният дизюнкт \blacksquare никога не е изпълним.

Означение.

Ако φ е безкванторна формула, тогава $D(\varphi)$ е следното множество от дизюнкти

$$D(\varphi) = \{ \delta \mid \delta \text{ е множеството от всички литерали от някоя елементарна дизюнкция от конюнктивната нормална форма на } \varphi \}.$$

Означение.

Ако имаме множество от безкванторни формули $\{ \varphi_1, \varphi_2, \dots \}$, тогава

$$D(\{ \varphi_1, \varphi_2, \dots \}) = D(\varphi_1) \cup D(\varphi_2) \cup \dots$$

Твърдение 25 (Свеждане до изпълнимост на дизюнкти).

За всяко множеството от затворени формули $\{ \varphi_1, \varphi_2, \dots \}$ е вярно, че то е изпълнимо тогава и само тогава, когато $D(\{ \psi_1, \psi_2, \dots \})$ е изпълнимо, където ψ_i е безкванторната част на Скулемовата нормална форма на φ_i за всяко i .

Доказателство.

От следствието към последното твърдение от предната глава знаем, че $\{ \varphi_1, \varphi_2, \dots \}$ е изпълнено, че то е изпълнимо тогава и само тогава, когато $\{ \chi_1, \chi_2, \dots \}$ е изпълнимо, където χ_i е Скулемовата нормална форма на φ_i за всяко i . Така, ако покажем, че за произволна универсална формула χ е вярно, че

$$\{ \chi \} \text{ е изпълнимо} \iff D(\{ \psi \}) \text{ е изпълнимо,}$$

където ψ е безкванторната част на χ , то тогава изпълнимостта на $\{ \chi_1, \chi_2, \dots \}$ ще е еквивалентна на изпълнимостта на $D(\{ \psi_1, \psi_2, \dots \})$.

Знаем, че χ има вида $\forall x_1 \forall x_2 \dots \forall x_n \psi$. Ако конюнктивната нормална форма на ψ изглежда така $\alpha_1 \& \alpha_2 \& \dots \& \alpha_m$, то тогава $D(\{ \psi \}) = \{ \delta_1, \delta_2, \dots, \delta_m \}$, където за всяко i , δ_i е дизюнкта съответстващ на елементарната дизюнкция α_i . Тогава

$$\begin{aligned} \{ \chi \} \text{ е изпълнимо} & \iff \\ \text{има структура } \mathcal{A} \text{ и оценка } v \text{ такива, че } \mathcal{A}, v \models \chi & \iff \\ \text{има структура } \mathcal{A} \text{ и оценка } v \text{ такива, че } \mathcal{A}, v \models \forall x_1 \forall x_2 \dots \forall x_n \psi & \iff \\ \text{има структура } \mathcal{A} \text{ и оценка } v \text{ такива, че } \mathcal{A}, v \models \forall x_1 \forall x_2 \dots \forall x_n (\alpha_1 \& \alpha_2 \& \dots \& \alpha_m) & \iff \\ \text{има структура } \mathcal{A} \text{ и оценка } v \text{ такива, че } \mathcal{A}, v \models \forall x_1 \forall x_2 \dots \forall x_n \alpha_i, \text{ за всяко } i & \iff \\ \text{има структура } \mathcal{A} \text{ и оценка } v \text{ такива, че } \mathcal{A}, v \models \forall x_1 \forall x_2 \dots \forall x_n L, \text{ за поне един литерал } L & \\ \text{от } \alpha_i, \text{ за всяко } i & \iff \\ \text{има структура } \mathcal{A} \text{ и оценка } v \text{ такива, че } \mathcal{A}, v \models \delta_i, \text{ за всяко } i & \iff \\ D(\{ \psi \}) \text{ е изпълнимо.} \end{aligned}$$



Задача.

Да се докаже, че една безкванторна формула φ е изпълнима тогава и само тогава, когато $EC(\varphi)$ е тъждествено вярна.

Упътване: Да се използва факта, че това дали една затворена формула е вярна при дадена структура и оценка, всъщност не зависи от оценката.

Задача.

Да се докаже, че една безкванторна формула φ е тъждествено вярна тогава и само тогава, когато $UC(\varphi)$ е изпълнима.

Упътване: Да се използва факта, че това дали една затворена формула е вярна при дадена структура и оценка, всъщност не зависи от оценката.

❖ Метод на резолюцията ❖

Дефиниция 56 (Директна резолвента).

Нека имаме два дизюнкта δ_1 и δ_2 . Ако има двойка противоположни литерали $L_1 \in \delta_1$ и $L_2 \in \delta_2$, тогава следния дизюнкт

$$\delta = (\delta_1 \setminus \{L_1\}) \cup (\delta_2 \setminus \{L_2\})$$

ще наричаме директна резолвента на δ_1 и δ_2 . В този случай казваме, че извършваме директна резолюция за δ_1 и δ_2 .

Пример.

Дизюнкт $\{p(f(x)), \neg p(g(c)), q(h(y))\}$ е директна резолвента на дизюнкти $\{p(f(x)), \neg r(f(c))\}$ и $\{\neg p(g(c)), r(f(c)), q(h(y))\}$.

Твърдение 26 (Запазване на изпълнимост при директна резолюция).

Ако множеството $\{\delta_1, \delta_2\}$ е изпълнимо и δ е директна резолвента на δ_1 и δ_2 , то тогава и δ е изпълним (и то при същите структура и оценка, както и $\{\delta_1, \delta_2\}$).

Доказателство.

Ако $\{\delta_1, \delta_2\}$ е изпълнимо, следователно има структура \mathcal{A} и оценка v при нея такива, че $\mathcal{A}, v \models \delta_1$ и $\mathcal{A}, v \models \delta_2$. Нека $L_1 \in \delta_1$ и $L_2 \in \delta_2$ са двата противоположни литерали, по които сме извършили директната резолюция.

Да допуснем, че $\mathcal{A}, v \models UC(L_1)$. Тогава $\mathcal{A}, v \not\models UC(L_2)$ и т.к. $\mathcal{A}, v \models \delta_2$, следователно има $L \in \delta_2$, който е различен от L_2 и за който $\mathcal{A}, v \models UC(L)$. Тогава ще имаме, че $L \in \delta$ и така $\mathcal{A}, v \models \delta$.

Аналогично, ако допуснем че $\mathcal{A}, v \not\models UC(L_1)$, тогава ще има $L \in \delta_1$, който е различен от L_1 и за който $\mathcal{A}, v \models UC(L)$. Тогава $L \in \delta$ и като резултат получаваме, че $\mathcal{A}, v \models \delta$. ❖

Дефиниция 57 (Директен резолютивен извод).

Нека имаме множество от дизюнкти Δ . Тогава директен резолютивен извод на дизюнкт δ от Δ е редицата $\delta_1, \delta_2, \dots, \delta_n$ ако:

- ☛ δ е последният дизюнкт в редицата (т.е. съвпада с δ_n);
- ☛ за всяко δ_i от редицата е изпълнено, че или $\delta_i \in \Delta$ или има $j < i$ и $k < i$ такива, че δ_i е директна резолвента на δ_j и δ_k .

В този случай ще казваме, че δ е директно изводим от Δ .

Пример.

Нека имаме начално множество от дизюнкти $\{p(c), r(f(d))\}$, $\{\neg r(f(d)), \neg q(h(c))\}$ и $\{q(h(c))\}$. Тогава следната редица $\delta_1 = \{p(c), r(f(d))\}$, $\delta_2 = \{\neg r(f(d)), \neg q(h(c))\}$, $\delta_3 = \{p(c), \neg q(h(c))\}$, $\delta_4 = \{q(h(c))\}$, $\delta_5 = \{p(c)\}$ е извод на $\{p(c)\}$ от това множество.

Твърдение 27 (Коректност на директния резолютивен извод).

Ако Δ е множество от дизюнкти и празният дизюнкт \blacksquare е директно изводим от него, тогава Δ е неизпълнимо.

Доказателство.

Нека $\delta_1, \delta_2, \dots, \delta_n$ е извода на \blacksquare от Δ . С индукция по дължината на извода ще покажем, че ако Δ е изпълнимо, то тогава и $\{\delta_1, \delta_2, \dots, \delta_n\}$ е изпълнимо (и то при същите структура и оценка). И т.к. неизпълнителят дизюнкт \blacksquare е елемент на $\{\delta_1, \delta_2, \dots, \delta_n\}$, това означава, че $\{\delta_1, \delta_2, \dots, \delta_n\}$ е неизпълнимо и като следствие и Δ е такова.

Така, нека Δ е изпълнимо при структура \mathcal{A} и оценка v . В случая $n = 1$ имаме, че $\delta_1 \in \Delta$ и следователно $\{\delta_1\} \subseteq \Delta$ също е изпълнимо при \mathcal{A} и v .

Нека сме доказали твърдението за $n = m$. Тогава при $n = m + 1$ нека разгледаме δ_{m+1} . Ако $\delta_{m+1} \in \Delta$ то тогава $\mathcal{A}, v \models \delta_{m+1}$ и $\{\delta_1, \delta_2, \dots, \delta_{m+1}\}$ е изпълнимо при \mathcal{A} и v . В противен случай, има $j < m + 1$ и $k < m + 1$ и δ_{m+1} е директна резолвента на δ_j и δ_k . Т.к. $\mathcal{A}, v \models \delta_j$ и $\mathcal{A}, v \models \delta_k$, то по предното твърдение следва, че и $\mathcal{A}, v \models \delta_{m+1}$. \blacksquare

Пример (Съвместяване на програми).

Метода на резолюцията можем да приложим за да покажем, че формулите от примера за съвместяване на програми са неизпълнимо множество:

- ☛ $\text{traffic}(\text{internet}) \vee \text{traffic}(\text{intranet})$
- ☛ $\neg(\text{traffic}(\text{internet}) \& \text{traffic}(\text{intranet}))$
- ☛ $\text{traffic}(\text{internet}) \implies \neg \text{start}(\text{scanner}) \& \text{start}(\text{filter})$
- ☛ $\text{traffic}(\text{intranet}) \implies \text{start}(\text{scanner})$
- ☛ $\text{traffic}(\text{internet}) \implies \text{start}(\text{scanner}) \& \neg \text{start}(\text{filter})$
- ☛ $\text{traffic}(\text{intranet}) \implies \neg \text{start}(\text{scanner}) \& \text{start}(\text{filter})$

Пример.

Нека имаме следните три твърдения

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

Видно е, че третото твърдение, следва от първите две.

Това не може да се покаже, обаче, използвайки само директна резолюция. Формулите, съответстващи на твърденията, са $\forall x(g(x) \implies h(x))$, $g(a)$ 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 $\neg g(x)$ и $g(a)$, например, не са противоположни литерали.

Означение.

Ако $\delta = \{ L_1, L_2, \dots, L_n \}$ е дизюнкта, а σ е субституция, тогава с $\sigma\delta$ ще означаваме дизюнкта $\{ \sigma L_1, \sigma L_2, \dots, \sigma L_n \}$.

Дефиниция 58 (Унифициране на литерали).

Ще казваме, че едно множество от литерали е унифицируемо, ако множеството от атомарни формули, което се получава след като премахнем отрицанията от литералите, е унифицируемо.

Дефиниция 59 (Резолвента).

Нека имаме два дизюнкта δ_1 и δ_2 , които нямат общи променливи (можем спокойно да разчитаме на това условие, т.к. дори и дизюнктите да имат общи променливи, то винаги можем да извършим подходящо преименуване, за да отстраним проблема). Ако има двойка литерали $L_1 \in \delta_1$ и $L_2 \in \delta_2$ с различен знак и $\{ L_1, L_2 \}$ е унифицируемо чрез унификатор λ , тогава

$$\delta = (\lambda\delta_1 \setminus \{ \lambda L_1 \}) \cup (\lambda\delta_2 \setminus \{ \lambda L_2 \})$$

ще наричаме резолвента на δ_1 и δ_2 . В този случай също ще казваме, че извършваме резолюция за δ_1 и δ_2 .

Примери.

$\{ p(x) \}$ е резолвента на $\{ \neg q(y) \}$ и $\{ q(f(c)), p(x) \}$.

$\{ p(f(c)) \}$ е резолвента на $\{ \neg q(y) \}$ и $\{ q(f(c)), p(x) \}$.

$\{ p(g(f(c))) \}$ е резолвента на $\{ \neg q(f(c)) \}$ и $\{ q(x), p(g(x)) \}$.

$\{ r(h(c)), q(c, h(h(c))) \}$ е резолвента на следните дизюнкти: $\{ r(z), \neg p(x, f(c)), \neg p(z, f(t)) \}$ и $\{ p(h(y), f(y)), p(z, t), q(y, h(z)) \}$.

Твърдение 28 (Запазване на изпълнимост при резолюция).

Ако множеството $\{ \delta_1, \delta_2 \}$ е изпълнимо и δ е резолвента на δ_1 и δ_2 , то тогава δ е изпълнимо (и то при същите структура и оценка, както и $\{ \delta_1, \delta_2 \}$).

Доказателство.

Ако $\{ \delta_1, \delta_2 \}$ е изпълнимо, следователно има структура \mathcal{A} и оценка v при нея такива, че $\mathcal{A}, v \models \delta_1$ и $\mathcal{A}, v \models \delta_2$. Нека $L_1 \in \delta_1$ и $L_2 \in \delta_2$ са двата литерала с различни знаци, по които сме извършили резолюцията, а λ е използвания унификатор.

Ако допуснем, че $\mathcal{A}, v \models UC(\lambda L_1)$, тогава $\mathcal{A}, v \not\models UC(\lambda L_2)$, а от това по правило за извод (σ) имаме, че $\mathcal{A}, v \not\models UC(L_2)$. Така, т.к. $\mathcal{A}, v \models \delta_2$ то има $L \in \delta_2$ различен от L_2 , за който $\mathcal{A}, v \models UC(L)$. От това следва, че $\mathcal{A}, v \models UC(\lambda L)$. Също така L няма как да се унифицира с L_2 чрез λ (иначе, нямаше да е изпълнено $\mathcal{A}, v \not\models UC(\lambda L_2)$). Поради това $\lambda L \in \delta$ и така се получава, че $\mathcal{A}, v \models \delta$.

Аналогично, ако допуснем че $\mathcal{A}, v \not\models UC(\lambda L_1)$, тогава ще има $L \in \delta_1$, който е различен от L_1 и за който $\lambda L \in \delta$ и $\mathcal{A}, v \models UC(L)$. Тогава $\mathcal{A}, v \models UC(\lambda L)$ и имаме, че $\mathcal{A}, v \models \delta$. ❖

Дефиниция 60 (Резолютивен извод).

Нека имаме множество от дизюнкти Δ . Тогава резолютивен извод на дизюнкт δ от Δ е редицата $\delta_1, \delta_2, \dots, \delta_n$ ако:

- ☛ δ е последният дизюнкт в редицата (т.е. съвпада с δ_n);
- ☛ за всяко δ_i от редицата е изпълнено, че или $\delta_i \in \Delta$ или има $j < i$ и $k < i$ такива, че δ_i е резолвента на δ_j и δ_k .

В този случай ще казваме, че δ е изводим от Δ .

Пример (Снабдяване при търговия).

Метода на резолюцията можем да приложим за да покажем, че формулите от примера за снабдяване при търговия са неизпълнимо множество:

- ☛ $\forall p \forall q (sell(p, q) \implies need(p, q))$
- ☛ $\forall p \forall q_1 \forall q_2 (buy(p, q_1) \& produce(p, q_2) \implies procure(p, q_1 + q_2))$
- ☛ $sell(bread, 40)$
- ☛ $produce(bread, 10)$
- ☛ $buy(bread, 20)$
- ☛ $40 > 20 + 10$
- ☛ $\forall p \forall q_1 \forall q_2 (need(p, q_1) \& procure(p, q_2) \implies \neg(q_1 > q_2))$

Можем да забележим, че на практика метода на резолюцията извършва същите действия за унифициране и елиминирание на унифициращи се литерали, както и таблото с унификации. Разликата е, че таблото с унификации също извършва и декомпозиция на формулите, докато тук тази декомпозиция вече е извършена по време на привеждането в нормална форма и извличането на дизюнктите.

Твърдение 29 (Коректност на резолютивния извод).

Ако Δ е множество от дизюнкти и празният дизюнкт \blacksquare е изводим от него, тогава Δ е неизпълнимо.

Доказателство.

Доказателството тук е аналогично на това за коректността на директния резолютивен извод. Използвайки предишното твърдение показваме, че ако Δ е изпълнимо, то и множеството от дизюнктите от извода (което съдържа празния дизюнкт \blacksquare) също е изпълнимо. ❖

Пример (Диагностика на пациенти).

Метода на резолюцията можем да приложим за да покажем, че формулите от примера за диагностика на пациенти са неизпълнимо множество:

- ☛ $\forall p(\neg \text{cough}(p) \& \text{snuffles}(p) \& \exists t(\text{temperature}(p, t) \& t > 37 \& t \leq 40) \implies \text{flu}(p))$
- ☛ $\forall p(\neg \text{cough}(p) \& \text{snuffles}(p) \& \exists t(\text{temperature}(p, t) \& t \geq 38 \& t < 39) \implies \text{flu}(p))$
- ☛ $\forall t(t \geq 38 \implies t > 37)$
- ☛ $\forall t(t < 39 \implies t \leq 40)$

Задача.

Да се провери, чрез метода на резолюцията, че от първите две от следните формули следва последната:

1. $\forall z(\forall t(\neg s(t) \implies p(t, z)) \implies r(z))$
2. $\exists t \forall z(s(z) \vee \neg q(z, t))$
3. $\exists z \forall t(\neg r(t) \implies \exists u(\neg p(u, t) \& \neg q(u, z)))$

Задача.

Да се провери, чрез метода на резолюцията, че от първите две от следните формули следва последната:

1. $\forall u \exists v(s(v) \& \forall w(\neg q(w, v) \implies \neg r(w, u)))$
2. $\forall u(\forall v(\neg p(v) \implies q(v, u)) \implies \neg s(u))$
3. $\forall v \exists u(\neg p(u) \& \neg r(u, v))$

Задача.

Да се провери, чрез метода на резолюцията, че от първите две от следните формули следва последната:

1. $\forall u \exists v(\neg q(v) \& \forall w(s(v, w) \implies p(u, w)))$
2. $\forall u(\forall v(r(v) \implies \neg s(u, v)) \implies q(u))$
3. $\forall v \exists u(r(u) \& p(v, u))$

Задача.

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

Упътване: Приложете метода към формула $\exists x(p(x) \implies \forall x p(x))$, където $p(x)$ означава, че x получава отлична оценка по Логическо програмиране.

Задача.

Да се провери, чрез метода на резолюцията, че второто твърдение е следствие от първото. Ето твърденията:

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

Упътване: Приложете метода към формули $\forall x(d(x) \implies h(x))$ и $\forall x(\neg h(x) \implies \neg d(x))$, където $d(x)$ означава, че x пие, а $h(x)$ означава, че x получава махмурлук.

Задача.

Да се провери, чрез метода на резолюцията, че третото от долните твърдения е следствие от първите две:

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

Упътване: Приложете метода към следните формули:

1. $\exists x(p(x) \& \forall y(d(y) \implies r(x, y)))$
2. $\neg \exists x \exists y(p(x) \& s(y) \& r(x, y))$
3. $\neg \exists x(d(x) \& s(x))$

където $p(x)$ означава, че x е пациент, $d(x)$ означава, че x е доктор, $r(x, y)$ означава, че x уважава y , а $s(x)$ означава, че x е шарлатан.

Задача.

Да се провери, чрез метода на резолюцията, че последното от долните четири твърдения е следствие от останалите:

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

Упътване: Приложете метода към следните формули:

1. $\forall x(b(x) \& \neg d(x) \implies \exists y(c(y) \& s(y, x)))$
2. $\exists x(t(x) \& b(x) \& \forall y(s(y, x) \implies t(y)))$

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

4. $\exists x(c(x) \& t(x))$

където $b(x)$ означава, че x преминава границата, $d(x)$ означава, че x е дипломат, $c(x)$ означава, че x е митничар, $s(x, y)$ означава, че x обискира y , а $t(x)$ означава, че x е трафикант.

Задача.

Нека имаме две множества от дизюнкти Δ_1 и Δ_2 , като всеки дизюнкт от Δ_2 е изводим от Δ_1 , а дизюнкта δ е изводим от Δ_2 . Да се докаже, че δ е изводим от Δ_1 .

❖ Изпълнение на Пролог програми чрез резолюция ❖

Метода на резолюцията дава трети начин за определяне, дали дадена цел се удовлетворява при дадена програма на Пролог. Този начин също разчита на класическата интерпретация на програмни клаузи и цел като формули от първи ред.

Алгоритъм (Изпълняване чрез метода на резолюцията).

Взимат се формулите, съответстващи на клаузите на програмата, и отрицанието на формулата, съответстваща на целта. Всяка от формулите от полученото множество се привежда в пренексна нормална форма и след това се Скулемизира и след това безкванторната част на получената универсална формула се привежда в конюнктивна нормална форма. От всички елементарни дизюнкции, които се получат по този начин, се образува множество от дизюнкти и към него се прилага метода на резолюцията. Ако от множеството от дизюнкти е изводим празният дизюнкт, тогава целта се удовлетворява от програмата. В противен случай целта не се удовлетворява.

Пример.

Нека имаме следната програма на Пролог и цел:

$p(X) :- q(X).$
 $q(X) :- r(X).$
 $q(10).$
 $r(2).$
 $r(7).$
 $?- p(X).$

От програмните формули и от отрицанието на целта се получават следните универсални формули, чиито безкванторни части привеждаме в конюнктивна нормална форма и от тях извличаме дизюнктите:

$p(X) :- q(X).$	\rightarrow	$\forall x(q(x) \implies p(x)) \equiv \forall x(\neg q(x) \vee p(x))$	\rightarrow	$\delta_1 = \{ \neg q(x), p(x) \}$
$q(X) :- r(X).$	\rightarrow	$\forall x(r(x) \implies q(x)) \equiv \forall x(\neg r(x) \vee q(x))$	\rightarrow	$\delta_2 = \{ \neg r(x), q(x) \}$
$q(10).$	\rightarrow	$q(10)$	\rightarrow	$\delta_3 = \{ q(10) \}$
$r(2).$	\rightarrow	$r(2)$	\rightarrow	$\delta_4 = \{ r(2) \}$
$r(7).$	\rightarrow	$r(7)$	\rightarrow	$\delta_5 = \{ r(7) \}$
$?- p(X).$	\rightarrow	$\neg \exists x p(x) \equiv \forall x \neg p(x)$	\rightarrow	$\delta_6 = \{ \neg p(x) \}$

От δ_1 и δ_6 получаваме директната резолвента $\delta_7 = \{ \neg q(x) \}$. След това имаме няколко варианта за извеждане на празния дизюнкт:

1. δ_7 комбинираме с δ_3 и получаваме празния дизюнкт чрез унификация $x = 10$.
2. От δ_2 и δ_7 получаваме директната резолвента $\delta_8 = \{ \neg r(x) \}$. След това с унификация $x = 2$ от δ_4 и δ_8 получаваме празния дизюнкт.
3. δ_2 и δ_7 дават $\delta_8 = \{ \neg r(x) \}$, а от δ_5 и δ_8 извличаме празния дизюнкт чрез $x = 7$.

Заб.: При всеки един от трите начина за проследяване на работата на програма на Пролог при дадена цел (чрез Табло с унифициране, чрез резолюция или чрез обратен извод), преудовлетворяванията на целта се случват при използване на различни комбинации от унифицирания (при използване на Табло - това са унифициранията за взимане на частен случай на формула или за намиране на противоречие в клон на дървото; при метода на резолюцията - това са унифициранията при получаване на резолвента; при обратния извод - това са унифициранията на първата формула от текущата цел с факти или глави на правила от програмата).

❖ Пълнота на метода на резолюцията ❖

Означение.

Нека Δ е множество от дизюнкти, а L е литерал. Тогава с $\Delta(L)$ ще означаваме следното множество от дизюнкти:

$$\Delta(L) = \{ \delta \mid \delta \text{ е директно изводим от } \Delta \text{ и } L \notin \delta \text{ и } L' \notin \delta \}$$

където L' е противоположният на L литерал.

Твърдение 30 (Неизпълнимост на $\Delta(L)$).

Нека Δ е множество от затворени дизюнкти, а L е затворен литерал. Тогава ако Δ е неизпълнимо, то и $\Delta(L)$ е неизпълнимо множество от дизюнкти.

Доказателство.

Нека допуснем, че Δ е неизпълнимо, но все пак $\Delta(L)$ е изпълнимо. Ще покажем, че от това следва противоречие.

Щом като $\Delta(L)$ е изпълнимо, то тогава има Ербранова структура \mathcal{H} и оценка v , която изпълняват всеки дизюнкт от $\Delta(L)$. Т.к. дизюнктите от $\Delta(L)$ не съдържат нито литерала L , нито неговия противоположен литерал L' , и понеже не знаем значите на L и L' , то няма как да знаем, кой от двата литерала се изпълнява от \mathcal{H} и v (т.к. те са противоположни, то \mathcal{H} и v изпълняват точно един от тях). Затова ще образуваме две нови Ербранови структури - \mathcal{H}_1 и \mathcal{H}_2 . \mathcal{H}_1 се получава от \mathcal{H} , като променим интерпретацията на предикатния символ от двата литерала така, че $\mathcal{H}_1, v \models L$. По същия начин образуваме \mathcal{H}_2 така, че $\mathcal{H}_2, v \models L'$. Понеже \mathcal{H} и v изпълняват дизюнктите от $\Delta(L)$, то тогава те изпълняват и всички дизюнкти от Δ , които не съдържат нито L нито L' (понеже те се съдържат в $\Delta(L)$). Същото се отнася и за \mathcal{H}_1 и \mathcal{H}_2 (в комбинация със същата оценка v).

След като Δ е неизпълнимо, то има поне един дизюнкт $\delta_1 \in \Delta$ такъв, че $\mathcal{H}_1, v \not\models \delta_1$. Аналогично, има $\delta_2 \in \Delta$ така, че $\mathcal{H}_2, v \not\models \delta_2$. Ако δ_1 не съдържаше нито L нито L' , то тогава щеше да се изпълнява от \mathcal{H}_1 и v . Следователно, или $L \in \delta_1$ или $L' \in \delta_1$ и т.к. $\mathcal{H}_1, v \models L$, тогава само $L' \in \delta_1$ е възможно. Аналогично получаваме, че $L \in \delta_2$.

Сега вече имаме, че $L' \in \delta_1$ и $L \in \delta_2$ и че L и L' са противоположни литерали. Тогава следния дизюнкт е директна резолвента на δ_1 и δ_2 :

$$\delta = (\delta_1 \setminus \{ L' \}) \cup (\delta_2 \setminus \{ L \})$$

δ не съдържаше нито L нито L' и δ е директна резолвента на дизюнкти от Δ , от което следва, че $\delta \in \Delta(L)$ и така $\mathcal{H}, v \models \delta$. Това означава, че има литерал $M \in \delta$, за който $\mathcal{H}, v \models M$. Т.к. M е различно и от L и от L' , следователно $\mathcal{H}_1, v \models M$ и $\mathcal{H}_2, v \models M$. За M има два възможни случая - $M \in \delta_1 \setminus \{ L' \}$ или $M \in \delta_2 \setminus \{ L \}$. В първия случай ще имаме на практика, че $M \in \delta_1$ и от това получаваме противоречието $\mathcal{H}_1, v \models \delta_1$ (а по-горе подбрахме δ_1 точно така, че $\mathcal{H}_1, v \not\models \delta_1$ да е изпълнено). А в другия случай, ще имаме $M \in \delta_2$ и ще получим противоречие с $\mathcal{H}_2, v \not\models \delta_2$. ❖

Твърдение 31 (Пълнота на директния резолутивен извод).

Ако Δ е неизпълнимо множество от затворени дизюнкти, то тогава празният дизюнкт \blacksquare е директно изводим от него.

Доказателство.

Щом Δ е неизпълнимо множество, то спокойно можем да считаме, че то е крайно. Ако не е, по теоремата на Ербран знаем че негово крайно подмножество е неизпълнимо и ако от него докажем, че е изводим \blacksquare , то \blacksquare ще е изводим и от по-голямото множество.

Щом Δ е крайно, то в дизюнктите от него се срещат краен брой литерали. Нека ги номерираме - L_1, L_2, \dots, L_n . От това, че Δ е неизпълнимо, по предното твърдение получаваме, че $\Delta(L_1)$ също е неизпълнимо. Пак с приложение на предното твърдение можем да получим и че $\Delta(L_1)(L_2)$ е неизпълнимо. И така с многократно използване на твърдението имаме, че $\Delta(L_1)(L_2) \dots (L_n)$ е неизпълнимо.

Дизюнктите от $\Delta(L_1)(L_2) \dots (L_n)$ не съдържат никои от литералите L_1, L_2, \dots, L_n . Само че, понеже в $\Delta(L_1)(L_2) \dots (L_n)$ няма как да се получат литерали, които не са дошли от Δ (т.е. да са някои от L_1, L_2, \dots, L_n), то тогава е сигурно, че в $\Delta(L_1)(L_2) \dots (L_n)$ няма никакви литерали. Понеже \blacksquare е единственият дизюнкт без литерали, то тогава имаме, че или $\Delta(L_1)(L_2) \dots (L_n) = \emptyset$ или $\Delta(L_1)(L_2) \dots (L_n) = \{ \blacksquare \}$. И т.к. празното множество е тривиално изпълнимо, то тогава остава само възможността $\Delta(L_1)(L_2) \dots (L_n) = \{ \blacksquare \}$.

Така $\blacksquare \in \Delta(L_1)(L_2) \dots (L_n)$. Т.к. $\Delta(L_1)(L_2) \dots (L_n)$ е съставено от дизюнкти, които са директно изводими от $\Delta(L_1)(L_2) \dots (L_{n-1})$, то тогава от $\Delta(L_1)(L_2) \dots (L_{n-1})$ е директно изводим \blacksquare . Аналогично получаваме, че \blacksquare е директно изводим от $\Delta(L_1)(L_2) \dots (L_{n-2})$ и след това от $\Delta(L_1)(L_2) \dots (L_{n-3})$ и т.н. накрая ще получим, че \blacksquare е директно изводим от Δ . \blacksquare

Твърдение 32 (Получаване на резолвента от директна резолвента).

Нека δ_1 и δ_2 са дизюнкти, а σ_1 и σ_2 са такива субституции, че $\sigma_1\delta_1$ и $\sigma_2\delta_2$ са затворени дизюнкти. Тогава, ако δ е директна резолвента на $\sigma_1\delta_1$ и $\sigma_2\delta_2$, то тогава δ е резолвента на δ_1 и δ_2 .

Доказателство.

Можем спокойно да разчитаме, че δ_1 и δ_2 нямат общи променливи. Ако имат - винаги можем да преименуваме част от променливите така, че да няма вече общи променливи между δ_1 и δ_2 . Преименуването на променливи не променя свойството за изпълнимост на дизюнктите. Също така, ако преименуваме променлива в даден дизюнкт δ_i същото преименуване ще трябва да направим и в σ_i (за да може $\sigma_i\delta_i$ да продължи да бъде същия затворен дизюнкт).

Също така, можем да разчитаме, че и в σ_1 и σ_2 няма общи променливи. Понеже сме подсилили, че в δ_1 и δ_2 няма общи променливи, то единствения шанс в σ_1 и σ_2 да има такива е някое σ_i да съдържа променливи, които не участват в промяната на съответния дизюнкт δ_i . В този случай можем да премахнем правилото за замяна на тези променливи от σ_i . Това няма да промени резултата $\sigma_i\delta_i$ и ще постигне нужния ефект - в σ_1 и σ_2 да няма общи променливи.

Нека имаме, че $\sigma_1 = \{ x_1^1 \rightarrow \tau_1^1, x_2^1 \rightarrow \tau_2^1, \dots, x_n^1 \rightarrow \tau_n^1 \}$ и $\sigma_2 = \{ x_1^2 \rightarrow \tau_1^2, x_2^2 \rightarrow \tau_2^2, \dots, x_m^2 \rightarrow \tau_m^2 \}$. Тогава нека образуваме субституцията $\sigma = \sigma_1 \cup \sigma_2 = \{ x_1^1 \rightarrow \tau_1^1, x_2^1 \rightarrow \tau_2^1, \dots, x_n^1 \rightarrow \tau_n^1, x_1^2 \rightarrow \tau_1^2, x_2^2 \rightarrow \tau_2^2, \dots, x_m^2 \rightarrow \tau_m^2 \}$. Ще покажем, че δ е резолвента на δ_1 и δ_2 чрез σ .

Т.к. δ е директна резолвента на $\sigma_1\delta_1$ и $\sigma_2\delta_2$, то тогава има двойка противоположни литерали $L'_1 \in \sigma_1\delta_1$ и $L'_2 \in \sigma_2\delta_2$. Следователно има литерали $L_1 \in \delta_1$ и $L_2 \in \delta_2$ такива, че $L'_1 = \sigma_1 L_1$ и $L'_2 = \sigma_2 L_2$. Т.к. в σ_1 и σ_2 няма общи променливи, тогава $\sigma_1\delta_1 = \sigma\delta_1$ и $\sigma_2\delta_2 = \sigma\delta_2$. Също така $L'_1 = \sigma_1 L_1 = \sigma L_1$ и $L'_2 = \sigma_2 L_2 = \sigma L_2$ от което имаме, че L_1 и L_2 са унифицируеми чрез σ .
Тогава

$$\delta = (\sigma_1\delta_1 \setminus \{L'_1\}) \cup (\sigma_2\delta_2 \setminus \{L'_2\}) = (\sigma\delta_1 \setminus \{\sigma L_1\}) \cup (\sigma\delta_2 \setminus \{\sigma L_2\}),$$

което наистина показва, че δ е резолвента на δ_1 и δ_2 . ✱

Твърдение 33 (Получаване на резолютивен извод от директен резолютивен извод).

Нека Δ е множество от дизюнкти, а $\delta'_1, \delta'_2, \dots, \delta'_n$ е директен резолютивен извод на затворения дизюнкт δ'_n от $CI(\Delta)$. Нека също така $\delta'_n \notin CI(\Delta)$. Тогава има резолютивен извод на δ'_n от Δ .

Доказателство.

За всеки елемент от $CI(\Delta)$ имаме, че е от вида $\sigma\delta$, където $\delta \in \Delta$, а σ е такава субституция, че $\sigma\delta$ е затворен литерал. Тогава, образуваме нова редица $\delta_1, \delta_2, \dots, \delta_n$ по следния начин:

- ако $\delta'_i \in CI(\Delta)$, тогава то е от вида $\sigma\delta$ и избираме δ_i да бъде точно това δ ;
- иначе δ_i е δ'_i .

Ще покажем, че $\delta_1, \delta_2, \dots, \delta_n$ е резолютивен извод на δ'_n от Δ .

Първо, понеже $\delta'_n \notin CI(\Delta)$ имаме, че δ_n съвпада с δ'_n . Ще покажем и, че за всяко i или $\delta_i \in \Delta$ или има $j < i$ и $k < i$ такива, че δ_i е резолвента на δ_j и δ_k . Това ще завърши доказателството на искания резултат.

Ще приложим индукция по дължината на извода. Нека сме доказали нужното твърдение за всички $i \leq m$. Сега ще го докажем и за $i = m+1$. Ако $\delta'_{m+1} \in CI(\Delta)$, то тогава имаме, че $\delta_{m+1} \in \Delta$. А в противен случай имаме, че δ'_{m+1} е директна резолвента на предишни елементи на редицата $\delta'_1, \delta'_2, \dots, \delta'_n$. Т.е. има $j < i$ и $k < i$ такива, че δ'_{m+1} е директна резолвента на δ'_j и δ'_k . Тогава по предното твърдение получаваме, че δ'_{m+1} е резолвента на δ_j и δ_k . Но т.к. δ'_{m+1} съвпада с δ_{m+1} (понеже $\delta'_{m+1} \notin CI(\Delta)$), следователно δ_{m+1} е резолвента на δ_j и δ_k . ✱

Твърдение 34 (Пълнота на резолютивния извод).

Ако Δ е неизпълнимо множество от дизюнкти, то тогава празният дизюнкт \blacksquare е изводим от него.

Доказателство.

По теоремата на Ербран, ако Δ е неизпълнимо то тогава и $CI(\Delta)$ е неизпълнимо (а $CI(\Delta)$ е множество от затворени дизюнкти). Тогава от твърдението за пълнота на директния резолютивен извод имаме, че от $CI(\Delta)$ е директно изводим \blacksquare . Нека $\delta'_1, \delta'_2, \dots, \delta'_n$ е изводът, с който това става. Тогава по предното твърдение можем да намерим извод $\delta_1, \delta_2, \dots, \delta_n$ на \blacksquare от Δ и така \blacksquare е изводим от Δ . ✱

Така, чрез комбинация на последното твърдение и на твърдението за коректност на резолютивния извод от предната глава получаваме следната

Теорема 4 (Коректност и пълнота на метода на резолюцията).

За всяко множество от дизюнкти Δ е вярно, че то е неизпълнимо тогава и само тогава, когато празният дизюнкт \blacksquare е изводим от него.

Пример.

Метода на резолюцията обикновено се използва за доказване на неизпълнимост на множество от формули, но понякога (макар и рядко) може да се използва и за показване на изпълнимост. Това става като демонстрираме, че от текущото множество от дизюнкти не е изводим празният дизюнкт.

Например, ако имаме следното множество от дизюнкти

$$D_1 = \{ p(g(c)), p(z), q(c) \},$$

$$D_2 = \{ q(y), r(f(y)) \},$$

$$D_3 = \{ r(f(c)), p(x) \}.$$

можем да забележим, че от никои два от тях не може да се изведе резолвента, т.к. това изисква да има отрицателен литерал в поне един от дизюнктите, а те съдържат само положителни литерали. Така, няма как чрез резолютивен извод да получи от това множество дизюнкт, който да е различен от D_1 , D_2 и D_3 .

Задача.

Да се докаже, чрез метода на резолюцията, че следното множество от дизюнкти е неизпълнимо:

$$\bullet \{ \neg q(c, x), \neg q(a, x) \}$$

$$\bullet \{ q(b, g(x)) \}$$

$$\bullet \{ p(g(y), x), q(y, x) \}$$

$$\bullet \{ q(c, f(x)), \neg q(b, x) \}$$

$$\bullet \{ \neg p(x, f(x)), \neg q(b, x) \}$$

Тук p и q са предикатни символи, f и g са функционални символи, а a , b и c са константи.

Задача.

Да се докаже, чрез метода на резолюцията, че следното множество от дизюнкти е изпълнимо:

$$\bullet \{ \neg p(x, f(c)), \neg p(h(y), z), \neg q(z, x) \}$$

$$\bullet \{ \neg r(f(c)), \neg p(x, y), \neg q(f(x), g(x)), \neg q(d, h(x)) \}$$

$$\bullet \{ \neg q(z, g(x)), \neg r(g(z)) \}$$

$$\bullet \quad \{ \neg p(f(x), x), \neg q(z, z), \neg p(h(x), y) \}$$

Тук p, q и r са предикатни символи, f, g и h са функционални символи, а c е константа.

Задача.

Да се докаже, чрез метода на резолюцията, че следното множество от формули е изпълнимо:

$$\bullet \quad \neg \exists y \neg (p(x) \implies \forall x q(x, y))$$

$$\bullet \quad \exists x (\neg s(x) \implies \forall y q(y, x))$$

$$\bullet \quad \forall x \exists z (q(z, x) \implies (r(z) \& \exists t \neg r(t)))$$

Задача.

Да се докаже, чрез метода на резолюцията, че следното множество от формули е изпълнимо:

$$\bullet \quad \exists x (\neg r(x) \implies \forall y p(y, x))$$

$$\bullet \quad \neg \exists y \neg (s(x) \implies \forall x p(x, y))$$

$$\bullet \quad \forall x \exists z (p(z, x) \implies (q(z) \& \exists t \neg q(t)))$$



Част II: Програмиране на Пролог



❖ Съставяне на Пролог клауза по формула ❖

Дефиниция 32 дава възможност за прилагане на техника за бързо съставяне на правило за Пролог програма по вече налична формула:

Алгоритъм (Съставяне на Пролог клауза по формула).

Ако трябва да се реализира правило, което да разпознава дали свойство $p(\tau_1, \tau_2, \dots, \tau_n)$ е изпълнено и имаме формула $\varphi(x_1, x_2, \dots, x_k)$, където x_1, x_2, \dots, x_k са всички променливи от термовете $\tau_1, \tau_2, \dots, \tau_n$, и тази формула определя/разпознава/описва нужното свойство, то тогава образуваме правилото:

$$\rho(\tau_1, \tau_2, \dots, \tau_n) :- \psi(x_1, x_2, \dots, x_k).$$

където $\psi(x_1, x_2, \dots, x_k)$ е еквивалентно преобразуване на $\varphi(x_1, x_2, \dots, x_k)$, което спазва описанието от Дефиниция 32 за формула съответстваща на тяло на правило (т.е. във формула $\psi(x_1, x_2, \dots, x_k)$ има само екзистенциални квантори, които са по всички променливи, освен по x_1, x_2, \dots, x_k , и кванторите се намират или след отрицание или в началото на формулата). Само че, когато $\psi(x_1, x_2, \dots, x_k)$ се записва като тяло на правило, екзистенциалните квантори се пропускат, т.к. на Пролог няма синтаксис за квантори.

Означение (Квантифициране по множество).

От тук нататък, ще използваме следните съкращения:

$$(\forall x \in S)\varphi \text{ е съкращение за формулата } \forall x(x \in S \implies \varphi)$$

$$(\exists x \in S)\varphi \text{ е съкращение за формулата } \exists x(x \in S \ \& \ \varphi)$$

Пример.

За да реализираме на Пролог предикат $\rho(L)$, който проверява дали списък от числа L съдържа само строго положителни числа, започваме от формулата, която описва тази ситуация - $(\forall x \in L)(x > 0)$. Тази формула е съкращение за $\forall x(x \in L \implies x > 0)$ и така можем този израз да преобразуваме:

$$\forall x(x \in L \implies x > 0) \equiv \neg \exists x \neg (x \in L \implies x > 0) \equiv \neg \exists x(x \in L \ \& \ \neg(x > 0))$$

Съответното Пролог правило ще изглежда така:

$$\rho(L) :- \text{not}(\text{member}(L, X), \text{not}(X > 0)).$$

Задача.

Да се реализира предикат $\text{max}(L, M)$, който по подаден списък от числа L , намира неговия максимален елемент.

Решение.

Директната формална дефиниция за максимален елемент е $m \in L \ \& \ (\forall x \in L)(x \leq m)$. Съответната ѝ формула може да се преобразува така:

$$\begin{aligned} m \in L \ \& \ \forall x(x \in L \implies x \leq m) & \equiv m \in L \ \& \ \neg \exists x \neg (x \in L \implies x \leq m) & \equiv \\ m \in L \ \& \ \neg \exists x(x \in L \ \& \ \neg(x \leq m)) & \equiv m \in L \ \& \ \neg \exists x(x \in L \ \& \ x > m) \end{aligned}$$

Така се получава следната клауза на Пролог:

```
max(L, M) :- member(L, M), not(member(L, X), X > M).
```



Заб.: По-внимателен анализ на работата на горната клауза може да установи, че тя работи за квадратично време. Ясно е, че това не е най-оптималното решение за намиране на максимален елемент. Това е единият от рисковете при писане на програми по директна дефиниция - често тези дефиниции са предназначени за по-голяма яснота, но не и за по-оптимална работа на евентуален алгоритъм, базиран на дефиницията. Така, при програмиране на Пролог често се налага да се балансира между яснотата на дефинициите в програмата и бързодействието ѝ.

Ето как изглежда на Пролог оптималното линейно решение за намиране на максимален елемент:

```
max([M], M).
max([H | T], H) :- max(T, M), H > M.
max([H | T], M) :- max(T, M), H <= M.
```

Задача.

Да се реализира предикат $p(L)$, който проверява дали в даден списък от списъци от числа L , за всеки два елемента X и Y на L е изпълнено, че има трети елемент на L , който съдържа сбора на най-големите елементи на X и на Y съответно.

Решение.

Израза, който описва нужното свойство за L е

$$(\forall x, y \in L)(\exists z \in L)(\max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z)$$

Той съответства на формула

$$\forall x(x \in L \implies \forall y(y \in L \implies \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z)))$$

Така

$$\begin{aligned} & \forall x(x \in L \implies \forall y(y \in L \implies \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z))) && \equiv \\ & \neg \exists x \neg (x \in L \implies \forall y(y \in L \implies \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z))) && \equiv \\ & \neg \exists x(x \in L \& \neg \forall y(y \in L \implies \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z))) && \equiv \\ & \neg \exists x(x \in L \& \exists y \neg (y \in L \implies \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z))) && \equiv \\ & \neg \exists x(x \in L \& \exists y(y \in L \& \neg \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z))) && \equiv \\ & \neg \exists x \exists y(x \in L \& y \in L \& \neg \exists z(z \in L \& \max(x, x_m) \& \max(y, y_m) \& x_m + y_m \in z))) && \equiv \end{aligned}$$

Следователно по последната формула от редицата преобразувания можем да напишем клаузата

```
p(L) :-
    not
    (
        member(L, X), member(L, Y),
```

```

not
(
    member(L, Z), max(X, Xm), max(Y, Ym),
    C is Xm + Ym, member(Z, C)
).

```



Заб.: Тук е добре да се отбележи, че процедурното решение (както представеното по-долу) не е по-бързо от реализацията на Пролог (и двете работят с кубична сложност, ако се пренебрегне работата на `max`), но пък т.к. то се състои от три вложени цикъла е много по-дълго и е по-сложно като структура.

```

for(int i = 0; i < L.Length; i++)
{
    for(int j = 0; j < L.Length; j++)
    {
        bool found = false;
        for(int k = 0; k < L.Length; k++)
        {
            if (L[k].Contains(Max(L[i]) + Max(L[j])))
            {
                found = true;
                break;
            }
        }
        if (!found)
            return false;
    }
}
return true;

```

Това процедурно решение може да се препише/реализира на Пролог:

```
p(L) :- forallX(L, L).
```

```
forallX([], _).
```

```
forallX([X | T], L) :- forallY(X, L, L), forallX(T, L).
```

```
forallY(_, [], _).
```

```
forallY(X, [Y | T], L) :- existsZ(X, Y, L), forallY(X, T, L).
```

```
existsZ(X, Y, [Z | T]) :- max(X, Xm), max(Y, Ym), C is Xm + Ym, member(Z, C).
```

```
existsZ(X, Y, [Z | T]) :-
```

```
    max(X, Xm), max(Y, Ym), C is Xm + Ym, not(member(Z, C)),
    existsZ(X, Y, T).
```

Това решение не е по-оптимално от решението, съставено по формула. А както е видно е значително по-дълго и освен това е по-трудно за разбиране (това е най-вече в случаите,

в които не знаем условието на задачата, а се опитваме да го възстановим, като четем само кода).

Задача.

Да се напише на Пролог предикат $p(L)$, който по даден списък от списъци от числа L проверява дали за всеки два елемента на L е изпълнено, че един от тях съдържа число, равно на броя на четните числа в другия.

Решение.

```
countEven([], 0).  
countEven([H | T], C) :- countEven(T, C1), H mod 2 == 0, C is C1 + 1.  
countEven([H | T], C) :- countEven(T, C), H mod 2 == 1.  
  
p(L) :- not(member(L, A), member(L, B), not(countEven(B, C), member(A, C))).
```



Задача.

Да се дефинира предикат $p(X,Y)$, който по даден списък X генерира в Y всички списъци, чиито елементи са елементи на X и броят на срещанията на най-често срещания елемент в Y е число, което не е елемент на X .

Задача.

Да се дефинират предикати $p(X)$, $q(X)$ и $r(X)$, такива че ако X е списък от списъци, то

- | | | |
|--------|-----------------------|--|
| $p(X)$ | \longleftrightarrow | два от елементите на някой елемент на X са равни на 5; |
| $q(X)$ | \longleftrightarrow | всеки два елемента на X имат поне три различни общи елемента; |
| $r(X)$ | \longleftrightarrow | съществува такова ненулево естествено число n , че X съдържа n различни по между си елемента, всеки от които има не повече от $2n$ елемента. |

❖ Разпознаватели и генератори ❖

Предикати, реализирани на Пролог, могат да се използват по два основни начина в целите към програми - като *разпознаватели* или като *генератори*.

Използването на предикат като разпознавател става, когато зададем цел, в която този предикат има конкретни стойности като аргументи. Т.е. в нито един от аргументите му, не е употребена свободна променлива. Например, целта `?- member([1, 2, 3], 4).` използва предикат `member` като разпознавател - разпознава се, дали стойност `4` е елемент на списъка `[1, 2, 3]`. А `member` се използва като генератор, чрез употребата на свободни променливи в целта - `?- member([1, 2, 3], X).` Така в променлива `X` се задават стойности `1`, `2` и `3` при трите последователни удовлетворявания на целта. При тази употреба казваме, че в `X` при преудовлетворяване последователно се генерират елементите на списъка. Това означава, че при първото удовлетворяване на целта в `X` се задава/генерира първия елемент на списъка, при второ удовлетворяване (първо преудовлетворяване) в `X` се задава/генерира втория елемент и т.н. докато има следващи елементи.

Предикат може да влиза в различните роли на разпознавател или генератор не само когато участва директно в целта към програмата. Това може да се случва и по време на изпълнението на програмата, когато оригиналната цел се свежда до други цели чрез правилата. Например, нека разгледаме реализацията на предикат `member` чрез `append`:

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

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

Задаваме цел към тази програма `?- member([1, 2, 3], 4).` Т.е. тук използваме `member` като разпознавател. Все още не е ясно, как ще използваме `append`. Но по второто правило `member(L, X) :- append(_, [X | _], L).` виждаме, че началната цел се свежда до нова цел - която се състои от `append(_, [4 | _], [1, 2, 3]).` В тази нова цел се вижда, че `append` се употребява без свободни променливи (анонимните променливи `_` не се считат за свободни). Така, `append` също се употребява като разпознавател, но не в първоначалната цел, а това се случва по време на изпълнението на програмата.

А ако към тази програма се зададе цел `?- member([1, 2, 3], X).`, то тя се свежда до нова цел `append(_, [X | _], [1, 2, 3]).` и така `append` ще бъде употребен като генератор.

Примери.

Предикатите `member` (независимо от реализацията му) и `append` могат да се използват по следните начини (тук също се дискутират и неправилни употреби):

- ☛ `?- member([1, 2, 3], 4).` - използва `member` като разпознавател; разпознава се дали `4` е елемент на `[1, 2, 3]`;
- ☛ `?- member([1, 2, 3], X).` - използва `member` като генератор; генерира в `X` стойности `1`, `2` и `3`;
- ☛ `?- member(L, 4).` - това е неправилна употреба; не могат да се генерират всички списъци от произволни обекти, които списъци съдържат `4`;

- ☛ ?– `member(L, X)`. - това също е неправилна употреба; не могат да се генерират всички списъци от произволни обекти, които списъци съдържат произволен обект;
- ☛ ?– `append([1, 2], [3, 4], [1, 2, 3, 4])`. - използва `append` като разпознавател; разпознава се дали конкатенацията на `[1, 2]` и `[3, 4]` дава `[1, 2, 3, 4]`;
- ☛ ?– `append([1, 2], [3, 4], X)`. - използва `append` като генератор, макар че, генерира само една стойност (т.е. удовлетворява се само веднъж); намира конкатенацията на `[1, 2]` и `[3, 4]`;
- ☛ ?– `append([1, 2], X, [1, 2, 3, 4])`. - използва `append` като генератор, макар че, генерира само една стойност (т.е. удовлетворява се само веднъж); намира такова `X`, че като се конкатенира `[1, 2]` с него да се получи `[1, 2, 3, 4]`;
- ☛ ?– `append(X, [3, 4], [1, 2, 3, 4])`. - използва `append` като генератор, макар че, генерира само една стойност (т.е. удовлетворява се само веднъж); намира такова `X`, което като се конкатенира с `[3, 4]` да се получи `[1, 2, 3, 4]`;
- ☛ ?– `append(X, Y, [1, 2, 3, 4])`. - използва `append` като генератор; всяко преудовлетворяване намира едно възможно разделяне на списъка `[1, 2, 3, 4]` на две части;
- ☛ ?– `append(X, [3, 4], Y)`. - това е неправилна употреба;
- ☛ ?– `append([1, 2], X, Y)`. - неправилна употреба;
- ☛ ?– `append(X, Y, Z)`. - неправилна употреба.

Могат да се реализират по-сложни генератори чрез комбинация от по-прости и от условия, които работят като разпознавател.

Пример.

Например, генератор `p(L, X)`, който изброява всички елементи `X` на списък от числа `L`, които се делят на `10`, може да се реализира така:

`p(L, X) :- member(L, X), X mod 10 == 0.`

Тук `member(L, X)` работи като генератор, който генерира по-голямо множество от нужно-то, а след това разпознаващото/филтриращото условие `X mod 10 == 0` отсява само тези стойности, които са необходими. Често това е много по-простото решение, отколкото да се прави генератор, който директно да генерира само нужните стойности.

Да се има пред вид, обаче, че при такива комбинации генератор-разпознавател трябва да се използва отрицание само в разпознаващата част. Иначе, ако се използва отрицание в генериращата част, то стойност генерирана в рамките на това отрицание не може да се пренесе извън областта му на действие (т.е. е нещо като локална променлива). Например, при реализация `p(L, X) :- not(member(L, X)), X mod 10 == 0`, стойността `X`, създадена посредством `member(L, X)`, няма да се пренесе по-нататък и когато се стигне до изпълнението на `X mod 10 == 0` тогава `X` няма да има стойност и проверката ще пропадне.

Ето и валиден пример за използването на отрицание при генериране:

Пример.

За да реализираме генератор на всички ненулеви елементи на даден списък от числа, можем да използваме следната клауза:

```
 $\rho(L, X) :- \text{member}(L, X), \text{not}(X == 0).$ 
```

Освен комбинацията генератор-разпознавател, за реализиране на по-сложни генератори може да се използва и още една друга подобна комбинация - генератор-преобразувател. При тази схема по-прост генератор генерира множество от елементи, а след това някак-ва преобразуваща функция го трансформира до нужния резултат.

Пример.

Да се реализира предикат, който генерира точните кубове на всички членове на даден списък от числа.

```
 $\rho(L, X) :- \text{member}(L, Y), X \text{ is } Y ** 3.$ 
```

❖ Разделяне на списъци на части чрез `append` ❖

Предикатът `append` може да се използва за доста лесно разделяне на даден списък на части и за извличане на специфични части от целия списък.

Примери.

Чрез `append` лесно само с една клауза може да се извлече последния елемент на списък:

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

Също могат да се извлекат едновременно с една клауза и първия и последния елементи:

```
firstAndLast(L, X, Y) :- append([X | _], [Y], L).
```

Заб.: Този предикат работи, ако има поне два елемента в списъка.

Чрез `append` може да се извади произволен елемент от списък и след това да се получи останалата част от списъка:

```
out(L, X, M) :- append(F, [X | B], L), append(F, B, M).
```

Също може да се извлече произволна двойка последователни елементи:

```
cons(L, X, Y) :- append(_, [X, Y | _], L).
```

Заб.: Реализираните четири предиката могат да се използват и като разпознаватели и като генератори.

Задача (Палиндром).

Ако на Пролог ще записваме низ като просто списъка от символи в низа, тогава да се реализира преикат, който проверява дали даден низ (списък от символи) е палиндром.

Решение.

```
palindrome([]).  
palindrome([_]).  
palindrome(L) :- append([X | R], [X], L), palindrome(R).
```



Задача.

Да се реализира преикат, който проверява дали даден низ (списък от символи) може да се представи като конкатенация на три палиндрома.

Решение.

```
p(L) :-
    append(F, R, L), append(S, T, R),
    palindrome(F), palindrome(S), palindrome(T).
```



Задача.

На Пролог матрица ще записваме, като всеки от редовете ѝ записваме като списък от елементите му, а самата матрица е списък от редовете си. Да се дефинира предикат $t(A, B)$, който по подадена матрица A генерира в B нейната транспонирана матрица.

Решение.

```
getFirstColumn([], [], []).
getFirstColumn([[X | R] | T], [X | C], [R | T1]) :- getFirstColumn(T, C, T1).

t([], [], []).
t([A, [C | T]], [C | T1]) :- getFirstColumn(A, C, A1), t(A1, T).
```



❖ Предикати за подписък, пермутация и подмножество ❖

Задача (Префикс, суфикс и подписък).

Да се реализират на Пролог преикати за генериране и разпознаване на префикс, суфикс и инфикс (подписък) на списък, както и за пермутация на списък.

Решение.

```
prefix(L, P) :- append(P, _, L).  
suffix(L, S) :- append(_, S, L).  
sublist(L, S) :- suffix(L, F), prefix(F, S).  
permuate([], []).  
permuate(L, [X | R]) :- out(L, X, M), permuate(M, T).
```

Заб.: Реализираните три предиката могат да се използват и като разпознаватели и като генератори. ❖

Задача (Подредица и подмножество).

Да се реализират на Пролог преикати за генериране и разпознаване на подмножество на елементите на даден списък.

Решение.

Предикат за разпознаване на подмножество се реализира така:

```
isSubset(L, S) :- not(member(S, X), not(member(L, X))).
```

Този предикат не става за генератор поради употребата на отрицание и т.к. никой от употребите на `member` не може да предизвика генериране/остойносттаване на `S`.

Ето реализации на генератори на подмножества:

```
subsequence([], []).  
subsequence([H | T], S) :- subsequence(T, S).  
subsequence([H | T], [H | S]) :- subsequence(T, S).
```

Горната реализация разчита на общия принцип, че всеки елемент на `L` може или да се включи или да не се включи в резултата. Тази реализация, обаче, не може да размества елементите на `L`. Затова и името на предиката е “подредица”, което отговаря по-точно на получения резултат.

А понякога разместването на елементите е нужно. Това се постига от следното решение:

```
subset(L, S) :- permuate(L, P), sublist(P, S).
```



Заб.: Генераторите на подмножества по-горе не могат да генерират списък-подмножество, в който има повторения на елементите от основния списък. Предикат, който постига това, ще бъде разгледан по-нататък.

Задача.

Да се дефинира предикат $p(X, Y, Z)$, който по дадени списъци X и Y генерира всички подсписъци Z на Y , такива че дължината на Z е колкото дължината на X , всеки елемент на Z е по-голям от елемента, намиращ се на същата позиция в X и последните елементи на Z и Y са равни.

Задача.

Ще казваме, че списъкът $C = [c_1, c_2, \dots, c_n]$ е сливане на списъците A и B , ако съществуват такива редици $i_1 < i_2 < \dots < i_k$ и $1 = j_1 < j_2 < \dots < j_m$, че $n = k + m$, $\{1, 2, \dots, n\} = \{i_1, i_2, \dots, i_k\} \cup \{j_1, j_2, \dots, j_m\}$, $A = [c_{i_1}, c_{i_2}, \dots, c_{i_k}]$ и $B = [c_{j_1}, c_{j_2}, \dots, c_{j_m}]$. Да се дефинира на Пролог предикат $q(A, B, C)$, който по дадени списъци A и B при преудовлетворяване генерира в C всевъзможните сливания на A и B .

❖ Сортиране на списъци ❖

При наличието на предикат за генериране на пермутация, много лесно може да се напише предикат за сортиране, който използва директно дефиницията за сортиран списък:

```
isSorted(L) :- not(cons(L, X, Y), X > Y).  
sort(L, S) :- permute(L, S), isSorted(S).
```

Разбира се, това решение е прекалено неефективно. По-нататък следват реализации на някои от по-ефективните алгоритми за сортиране на списък.

Задача (Сортиране чрез извличане).

Да се реализира на Пролог сортиране на списък по метода на извличането.

Решение.

```
min([M], M).  
min([H | T], H) :- min(T, M), H < M.  
min([H | T], M) :- min(T, M), H >= M.  
  
selectionSort([], []).  
selectionSort(L, [X | S]) :- min(L, X), out(L, X, M), selectionSort(M, S).
```



Задача (Сортиране чрез сливане).

Да се реализира на Пролог сортиране на списък по метода на сливането.

Решение.

```
count([], 0).  
count([_ | T], C) :- sum(T, R), C is R + 1.  
  
split(L, F, S) :-  
    append(F, S, L), count(F, CF), count(S, CS), abs(CF - CS) =< 1.  
  
merge(L, [], L).  
merge([], L, L).  
merge([HF | TF], [HS | TS], [HF | R]) :- HF <= HS, merge(TF, [HS | TS], R).  
merge([HF | TF], [HS | TS], [HS | R]) :- HF > HS, merge([HF | TF], TS, R).  
  
mergesort([], []).  
mergesort([X], [X]).  
mergesort(L, R) :-  
    split(L, F, S), mergesort(F, SF), mergesort(S, SS), merge(SF, SS, R).
```



Задача (Бързо сортиране).

Да се реализира на Пролог сортиране на списък по метода на бързото сортиране.

Решение.

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

average(L, A) :- sum(L, S), count(L, C), A is S / C.

pivot([], _, [], []).
pivot([H | T], P, [H | F], S) :- H =< P, pivot(T, F, S).
pivot([H | T], P, F, [H | S]) :- H > P, pivot(T, F, S).

quicksort([], []).
quicksort([X], [X]).
quicksort(L, R) :-
    average(L, A), pivot(L, A, F, S),
    quicksort(F, SF), quicksort(S, SS), append(SF, SS, R).
```



Заб.: За простота, реализацията на бързото сортиране не проверява дали всички елементи на списъка са еднакви или не. Това все пак може да се наложи при реална употреба на предиката.

Задача.

Казваме, че списък X мажорира списък Y , ако всички елементи на X са елементи на Y . Да се дефинира на пролог предикат $p(L, M)$, който по даден списък от списъци L намира списък M , който съдържа всички елементи на L и в който никой елемент не се мажорира от елемент, намиращ се след него в списъка.

Решение.

```
major(X, Y) :- not(member(X, A), not(member(Y, A))).

sorted(M) :-
    not(append(_, [X | B], M), member(B, Y), major(Y, X)).

p(L, M) :- permute(L, M), sorted(M).
```



❖ Предикати за работа с графи ❖

За представяне на графи ще използваме записа $[V, E]$, където V е списъка на върховете, а E е списък от двойки от върхове $[[X_1, Y_1], [X_2, Y_2], \dots, [X_n, Y_n]]$, които представляват всички ребра $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots, \langle x_n, y_n \rangle$ в графа. Т.е. E е списъка от ребрата.

Както и при сортирането има тривиален, но много неефективен начин за намиране на път в граф от даден връх до друг връх:

```
path([V, E], X, Y) :-  
    subset(V, P), firstAndLast(P, X, Y),  
    not(cons(P, X, Y), not(member(E, [X, Y]))).
```

Ето по-ефективно решение:

```
path([V, E], X, Y) :- pathAux([V, E], X, Y, [X]).  
  
pathAux([V, E], X, X, _).  
pathAux([V, E], X, Y, Visited) :-  
    member(E, [X, Z]), not(member(Visited, Z)),  
    pathAux([V, E], Z, Y, [Z | Visited]).
```

Ако предиката `path` се използва със зададена конкретна стойност в променлива X и с оставена свободна променлива Y може да се постигне обхождане на графа в дълбочина.

Заб.: "Наивното" решение

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

не работи правилно, когато в графа има цикли.

Задача (Свойства на графи).

Да се реализират на Пролог предикати за разпознаване дали:

- ☛ дали даден граф $[V, E]$ е тотален;
- ☛ дали даден граф $[V, E]$ е ориентиран;
- ☛ дали даден граф $[V, E]$ е неориентиран;
- ☛ дали във всички върхове от даден граф $[V, E]$ има примки;
- ☛ дали даден граф $[V, E]$ е свързан;
- ☛ дали в даден граф $[V, E]$ има изолирани върхове;
- ☛ дали даден граф $[V, E]$ е транзитивен;
- ☛ дали даден граф $[V, E]$ е цикличен;
- ☛ дали даден граф $[V, E]$ е мултиграф.

Решение.

```
isTotal([V, E]) :- not(member(V, X), member(V, Y), not(member(E, [X, Y]))).
isDirected([V, E]) :- member(E, [X, Y]), not(member(E, [Y, X])).
isUndirected([V, E]) :- not(isDirected([V, E])).
allLoops([V, E]) :- not(member(V, X), not(member(E, [X, X]))).
isConnected([V, E]) :-
    not(member(V, X), member(V, Y), not(path([V, E], X, Y))).
hasIsolated([V, E]) :- member(V, X), not(member(E, T), member(T, X)).
isTransitive([V, E]) :-
    not(member(E, [X, Y]), member(E, [Y, Z]), not(member(E, [X, Z]))).
hasCycle([V, E]) :- member(V, X), member(E, [X, X]).
hasCycle([V, E]) :- path([V, E], X, Y), not(X = Y), member(E, [Y, X]).
isMultigraph([V, E]) :- out(E, T, M), member(M, T).
```



Задача.

Нека E е списък от двуелементни списъци като всеки двуелементен списък $[U, V]$ интерпретираме като ребро от връх U към V в граф. Да се дефинират на пролог следните предикати:

- ☛ $p(E)$, който разпознава дали заданият с E граф е цикличен. Един граф е цикличен, ако ребрата му образуват цикъл, в който през всяко ребро и през всеки връх се минава точно по веднъж.
- ☛ $q(E)$, който разпознава дали заданият с E граф е осморка. Един граф е осморка, ако е обединение на два циклични графа, които имат точно един общ връх помежду си.

Заб.: Граф, който е осморка, може така да се разположи в равнината, че да изписва цифрата осем.

Решение.

```
p(E) :-
    permuate(E, PE),
    not(append(_, [[A1, A2], [B1, B2] | _], PE), not(A2 = B1)),
    firstAndLast(PE, [X, _], [_, X]),
    not
    (
```

```

        append(_, [[A, B], [B, C] | _], PE), out([A, B], PE, PE1),
        out([B, C], PE1, PE2), member(PE2, D), member(D, B)
    ).

```

```

q(E) :-
    permute(E, PE), append(E0, E1, PE),
    p(E0), p(E1),
    member(E0, A), member(E1, B), member(A, X), member(B, X),
    not
    (
        member(E0, A), member(E1, B),
        member(A, Y), member(B, Y), not(X = Y)
    ).

```



Задача.

Нека E е списък от двueleментни списъци като всеки двueleментен списък $[U, V]$ интерпретираме като ребро от връх U към V в граф. Да се дефинират на пролог следните предикати:

- $p(E)$, който разпознава дали зададеният с E граф е цикличен. Един граф е цикличен, ако ребрата му образуват цикъл, в който през всяко ребро и през всеки връх се минава точно по веднъж.
- $q(E)$, който разпознава дали зададеният с E граф е трилистна детелина. Един граф е трилистна детелина, ако е обединение на три циклични графа, които имат точно един общ връх помежду си и никои два от тях нямат друг общ връх.

Заб.: Граф, който е трилистна детелина, може така да се разположи в равината, че да изобразява трилистна детелина.

Решение.

```

p(E) :-
    permute(E, PE),
    not(append(_, [[A1, A2], [B1, B2] | _], PE), not(A2 = B1)),
    firstAndLast(PE, [X, _], [_], X),
    not
    (
        append(_, [[A, B], [B, C] | _], PE), out([A, B], PE, PE1),
        out([B, C], PE1, PE2), member(PE2, D), member(D, B)
    ).

```

```

q(E) :-
    permute(E, PE), append(E0, TE, PE), append(E1, E2, TE)
    p(E0), p(E1), p(E2),
    member(E0, A1), member(E1, B1), member(E2, C1),

```

```
member(A1, X), member(B1, X), member(C1, X),  
not  
(  
    member(E0, A2), member(E1, B2), member(E2, C2),  
    member(A2, Y), member(B2, Y), member(C2, Y), not(X = Y)  
).  

```



❖ Решаване на NP-пълни задачи ❖

Пролог е особено подходящ за решаване на NP-пълни задачи, т.к. чрез него могат да се постигнат много кратки и ясни програми, които не са по-бавни (не са в друг клас на сложност) от директни (наивни) решения написани на кой да е друг език.

Задача (Подмножество с нулева сума).

Да се реализира на Пролог предикат за намиране на подмножество (на дадено множество от цели числа) с нулева сума.

Решение.

```
zeroSubset(L, S) :- subset(L, S), not(S = []), sum(S, M), M =:= 0.
```



Задача (Хамилтонов път).

Да се реализира на Пролог предикат за намиране на Хамилтоновите пътища в даден граф.

Решение.

```
hamiltonPath([V, E], P) :-  
    permute(V, P), not(cons(P, X, Y), not(member(E, [X, Y]))).
```



Задача (Максимална клика).

Да се реализира на Пролог предикат за намиране на максималните клики в даден граф.

Решение.

```
isProperSubset(L, S) :- isSubset(L, S), member(L, X), not(member(S, X)).
```

```
clique([V, E], C) :- subset(V, C), isTotal([C, E]).
```

```
maximalClique([V, E], C) :-  
    clique([V, E], C), not(clique([V, E], AC), isProperSubset(AC, C)).
```



Задача (Минимално покриващо множество от върхове).

Да се реализира на Пролог предикат за намиране на минималните покриващи множества от върхове на даден граф.

Решение.

```
coverSet([V, E], C) :-  
    subset(V, C), not(member(E, T), not(member(C, X), member(T, X))).  
  
minimalCoverSet([V, E], C) :-  
    coverSet([V, E], C), not(coverSet([V, E], AC), isProperSubset(C, AC)).
```



Задача (Задача за търговския пътник).

Да се реализира на Пролог предикат за намиране решенията на задачата за търговския пътник.

Решение.

Тук C е списък от градове, а D е списък от тройки от вида $[F_i, S_i, CD_i]$, където CD_i е разстоянието между градове F_i и S_i .

```
tsp(C, D, R) :-  
    permute(V, R), totalDistance(R, D, TD),  
    not(permute(V, AR), totalDistance(AR, D, ATD), ATD > TD).  
  
totalDistance([_], _, 0).  
totalDistance([F, S | T], D, TD) :-  
    member(D, [F, S, CD]), totalDistance([S | T], F), TD is CD + F.
```



Задача (Задачата за раницата).

Да се реализира на Пролог предикат за намиране решенията на задачата за раницата.

Решение.

Тук O е списък от предмети, V е максималния обем на раницата, а D е списък от тройки от вида $[O_i, V_i, L_i]$, където V_i и L_i са обема и стойността на предмета O_i .

```
knapsack(O, V, D, S) :-  
    solution(O, V, D, S, TL), not(solution(O, V, D, AS, ATL), ATL > TL).  
  
solution(O, V, D, S, TL) :-  
    subset(O, S), sumVolume(S, D, SV), SV < V, sumValue(S, D, TL).  
  
sumVolume([], _, 0).  
sumVolume([H | T], D, SV) :-
```

```
member(D, [H, V, _]), sumVolume(T, D, F), SV is F + V.  
sumValue([], _, 0).  
sumValue([H | T], D, TL) :-  
    member(D, [H, _, L]), sumValue(T, D, F), TL is F + L.
```



❖ Реализиране на Пролог клаузи чрез определимост ❖

Техниката за дефиниране (определяне) на нови условия (елементи, функции и релации) в структура за език от първи ред може много успешно да се комбинира с метода за съставяне на Пролог клауза по формула (в която свободните променливи са точно тези, които се срещат в главата на клаузата). По този начин се работи с презумпцията, че предикат, реализиран на Пролог, всъщност определя множеството от стойности на променливите му, за което предиката се удовлетворява.

Така, се постига техника на програмиране на Пролог, която използва математически дефиниции (определимост) както за реализиране на клаузите, така и за гарантиране на частичната коректност на програмата. Техниката, разбира се, има своите ограничения и не може да се използва за съставяне на всякакви програми на Пролог. Например, по този начин не могат да се реализират рекурсивни схеми, т.к. формулите (чрез които определяме новите условия) не могат да съдържат себе си като същински подниз.

Също така, реализираните по този начин клаузи на Пролог винаги ще работят като разпознаватели, но не винаги ще работят като генератори (за реализирането на нужния генератор може да се наложи употребата на рекурсивна схема).

Нака изпробваме тази техника със структура, в която носителя A е множеството от всички естествени числа и всички крайни списъци, съставени от естествени числа и/или други списъци. Така, това всъщност е Ербрановия носител, който се получава от език съдържащ всички естествени числа като константи, има константа за празния списък и двуместен функционален символ за съставяне на нов списък с първи елемент - първия аргумент на функционалния символ - и опашка - втория му аргумент.

Нека езика на структурата се състои от следните четири предиката:

- ☛ $append(x, y, z)$, който е истина тогава и само тогава, когато x , y и z са списъци и z е конкатенацията на x и y ;
- ☛ $head(l, x)$, който е истина тогава и само тогава, когато l е списък и x е първия елемент на l ;
- ☛ $tail(l, t)$, който е истина тогава и само тогава, когато l и t са списъци и t е опашката (всички елементи без първия) на l ;
- ☛ $perm(l, p)$, който е истина тогава и само тогава, когато l и p са списъци и p е пермутация на l .

В тази структура ще определим нови предикати за принадлежност на елемент към списък, за разпознаване на двойка от обекти, както и няколко стандартни предиката за работа с графи (които ще са представени като списъци от ребра - т.е. като списъци от двойки от обекти) - разпознаване на път в граф, разпознаване дали граф е свързан, разпознаване дали граф е Хамилтонов, разпознаване дали граф е Ойлеров. По всяка от получените формули ще съставим клауза на Пролог, която ще реализира съответния ѝ предикат.

Предикати за принадлежност на елемент към списък се определя чрез предикати $append$ и $head$ така:

$$member(l, x) \equiv \exists t(tail(l, t) \& head(t, x))$$

Тази формула можем директно да препишем като тяло на клауза и така да реализираме на Пролог предиката `member` чрез вече реализирани предикати `append` и `head`:

```
member(L, X) :- tail(L, T), head(T, X).
```

Предикат, който разпознава дали даден списък е двойка от обекти, се определя така:

$$pair(l) \Rightarrow \exists t_1 \exists t_2 (tail(l, t_1) \& tail(t_1, t_2) \& \neg \exists x member(t_2, x))$$

Преписвайки формулата като клауза директно ни дава реализацията на предиката на Пролог (като някои от променливите записваме като анонимни):

```
pair(L) :- tail(L, T1), tail(T1, T2), not(member(T2, _)).
```

Предикат за разпознаване на път в граф се определя по следния начин (тук E е списък от двойки от обекти - списъка от ребра в графа, а V е списъка от върхове):

$$\begin{aligned} path(V, E, x, y, p) \Rightarrow & \exists s \exists t (perm(V, s) \& append(p, t, s) \& \\ & head(p, x) \& \exists l \exists q \exists i (append(l, q, p) \& head(q, y) \& tail(q, i) \& \neg \exists z member(i, z)) \& \\ & \forall r (\exists k \exists m \exists n (append(k, m, p) \& append(r, n, m)) \& pair(r) \Rightarrow member(E, r))) \end{aligned}$$

След еквивалентни преобразувания от горната формула получаваме:

$$\begin{aligned} & \exists s \exists t (perm(V, s) \& append(p, t, s) \& \\ & head(p, x) \& \exists l \exists q \exists i (append(l, q, p) \& head(q, y) \& tail(q, i) \& \neg \exists z member(i, z)) \& \\ & \neg \exists r (\exists k \exists m \exists n (append(k, m, p) \& append(r, n, m)) \& pair(r) \& \neg member(E, r))) \end{aligned}$$

А ето и директната реализация на Пролог по преобразуваната формула:

```
path(V, E, X, Y, P) :-
    perm(V, S), append(P, _, S), head(P, X),
    append(_, Q, P), head(Q, Y), tail(Q, I), not(member(I, _)),
    not(append(_, M, P), append(R, _, M), pair(R), not(member(E, R))).
```

Заб.: Предикатите `member` и `path` ще работят също и като генератори.

Предикат за свързаност на граф се определя така:

$$connected(V, E) \Rightarrow \forall x \forall y (member(V, x) \& member(V, y) \Rightarrow \exists p path(V, E, x, y, p))$$

И след преобразувания се получава следната клауза на Пролог:

```
connected(V, E) :- not(member(V, X), member(V, Y), not(path(V, E, X, Y, P))).
```

За проверка за Хамилтонов и Ойлеров графи се използват тези формули:

$$\begin{aligned} hamilton(V, E) \Rightarrow & \exists p (perm(V, p) \& \\ & \forall r (\exists k \exists m \exists n (append(k, m, p) \& append(r, n, m)) \& pair(r) \Rightarrow member(E, r))) \\ euler(V, E) \Rightarrow & \exists p (perm(E, p) \& \\ & \forall r (\exists k \exists m \exists n (append(k, m, p) \& append(r, n, m)) \& pair(r) \Rightarrow \\ & \exists x \exists t \exists y \exists z \exists a (head(r, x) \& tail(x, t) \& head(t, a) \& tail(r, y) \& head(y, z) \& head(z, a)))) \end{aligned}$$

Те дават следните клаузи на Пролог (аналогична клауза беше получена и по-рано при решаването на съответната NP-пълна задача):

```

hamilton(V, E) :-
    perm(V, P),
    not(append(_, M, P), append(R, _, M), pair(R), not(member(E, R))).
euler(V, E) :-
    perm(E, P),
    not
    (
        append(_, M, P), append(R, _, M), pair(R),
        not
        (
            head(R, X), tail(X, T), head(T, A),
            tail(R, Y), head(Y, Z), head(Z, A)
        )
    ).

```

Задача.

Дадена е структурата $\mathcal{A} = (A, \text{append}, \text{head}, \text{tail}, \text{perm})$, където A е множеството от всички естествени числа и всички крайни списъци, а append , head , tail и perm са предикати за конкатенация на списъци, глава и опашка на списък и пермутация на списък. Да се дефинират следните множества и предикати:

- ☛ множеството на естествените числа;
- ☛ множеството на списъците;
- ☛ празния списък, множеството от всички едноелементни списъци, множеството от всички двueleментни списъци, и т.н.;
- ☛ предикати $\text{member}(l, x)$, $\text{first}(l, x)$, $\text{last}(l, x)$, $\text{cons}(l, x, y)$, където x и y са последователни елементи в l , $\text{sublist}(l, s)$ и $\text{subset}(l, s)$.

Да се реализират на Пролог предикатите `member`, `first`, `last`, `cons`, `sublist` и `subset` чрез формулите, с които са били определени в структурата.

❖ Работа с аритметични операции ❖

Тук се проследяват основни разлики между оператори $=$, is и предикат $==$.

Оператор $=$ е оператор за унифициране. Неговите аргументи са произволни термове и ако те са унифицируеми, то оператора предизвиква присвояване на стойности на тези променливи в термовете, които все още нямат стойност, така че да се постигне унифицирането.

Примери.

$2 + X = Y + 17$ предизвиква присвоявания $X = 17$ и $Y = 2$.

$[1 \mid T] = [X, 2, 3 \mid Q]$ предизвиква присвоявания $T = [2, 3 \mid Q]$ и $X = 1$.

$2 + 2 = 4$ никога не се удовлетворява, т.к. термовете $2 + 2$ и 4 не са унифицируеми.

Оператор is е оператор за присвояване на стойност. Той обикновено се използва в изрази от вида $X is \tau$, където X винаги е променлива или константа (не може да е по-сложен терм), а τ е аритметичен терм. Оператора предизвиква остойностяване на терма τ и ако то е успешно тогава:

- ако X все още няма стойност, ѝ присвоява стойността на τ ;
- ако X вече има стойност, тогава проверява дали тя е равна на стойността на τ .

Поради това поведение изрза $X is X + 1$ никога няма да се удовлетворява на Пролог. Ако X няма стойност, тогава остойностяването на $X + 1$ няма да е успешно, а ако X има стойност, тогава тя няма как да е равна на стойността на $X + 1$.

is не може да се употребява за списъци!

Примери.

$X is abs(7 - 9) ** 4$ предизвиква задаване на стойност $X = 16$.

$4 is 2 + 2$ се удовлетворява. $2 + 2 is 4$ е грешен синтаксис.

$X is (X + 8) / 2$ се удовлетворява тогава и само тогава, когато стойността на X е 8.

$==$ е предикат за аритметично сравнение и се употребява в изрази от вида $\tau_1 == \tau_2$, където τ_1 и τ_2 са аритметични термове. Предиката предизвиква остойностяване на термовете и ако и двете остойностявания са успешни, сравнява получените стойности.

$==$ не може да се употребява за списъци!

Примери.

$2 + X \text{ := } Y + 17$ е истина тогава и само тогава, когато и X и Y вече имат стойности и разликата на тези стойности е 15.

$2 + 2 \text{ := } 4$ се удовлетворява.

Задача (Най-голям общ делител).

Да се реализира на Пролог предикат за намиране на най-големия общ делител на две числа.

Решение.

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



Задача.

Да се дефинира на Пролог предикат $p(N, L)$, който по дадено число N и списък от положителни числа $L = [a_1, a_2, \dots, a_n]$, проверява дали е възможно в редицата a_1, a_2, \dots, a_n да се вмъкнат по такъв начин символи "(", ")" и "-", че полученият аритметичен израз да има стойност N .

Решение.

```
p(0, []).  
p(N, [N]).  
p(N, L) :- append(L1, L2, L), p(L1, S1), p(L2, S2), N is S1 - S2.
```



❖ Предикати за генериране на (без)крайни множества ❖

За генериране на крайно множество от числа - обхват от долна до горна граници - се използва следния предикат:

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

Заб.: Този предикат може да се използва и като разпознавател и като генератор.

`between` е много удобен за проверка на условия, които зависят от краен брой данни/числа. Така ако определим минималната и максималната гранични стойности за тези числа, можем да генерираме всички числа от този обхват (от минималната до максималната стойност) и да отделим с допълнително условие от генерираните числа тези, които са ни нужни за текущата проверка.

Задача (Най-голям общ делител II).

Да се реализира на Пролог предиката за намиране на най-големия общ делител на две числа чрез предикат `between`.

Решение.

```
cd(X, Y, Z) :- between(1, X, Z), Z =< Y, X mod Z =:= 0, Y mod Z =:= 0.
```

```
gcd(X, Y, Z) :- cd(X, Y, Z), not(cd(X, Y, AZ), AZ > Z).
```



Задача (Фибоначи).

Да се реализират на Пролог предикати за генериране и разпознаване на елементите на редицата на Фибоначи.

Решение.

Генератора се реализира така:

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

А за разпознаването разчитаме на факта, че редицата е строго разтяща и поради това за всеки елемент имаме $a_i \geq i$.

```
fibonacciIndexTwo(0, 0, 1).  
fibonacciIndexTwo(I, F1, F2) :-  
    I1 is I - 1, fibonacciIndexTwo(I1, F, F1), F2 is F1 + F.
```

```
isFibonacci(X) :- between(0, X, I), fibonacciIndexTwo(I, X, _).
```



За генериране на множеството на всички естествени числа се използва предикат:

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

Всички генератори на безкрайни множества по един или друг начин използват `int`.

`int` не е добре да се използва като разпознавател, т.к. реализира полуразрешим алгоритъм и при цел `?-int(1.5)` ще зацikli без да даде резултат.

Задача (Прости числа).

Да се реализират на Пролог предикати за генериране и разпознаване на простите числа.

Решение.

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



Задача.

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

Решение.

```
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).
```



Важно е да не се използват два безкрайни генератора наведнъж. Например, ако трябва да се генерираат всички двойки естествени числа "наивното" решение е:

```
pair(X, Y) :- int(X), int(Y).
```

Това решение ще генерира двойките $\langle 0, 0 \rangle$, $\langle 0, 1 \rangle$, $\langle 0, 2 \rangle$..., но никога няма да генерира $\langle 1, 0 \rangle$.

Правилното решение е:

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

Задача (Аритметични прогресии).

Да се реализира на Пролог предикат, който генерира всички крайни аритметични прогресии.

Решение.

```
triple(X, Y, Z) :-  
    int(S), between(0, S, X), S1 is S - X, between(0, S1, Y), Z is S1 - Y.
```

```
progression(0, _, _, []).
```

```
progression(C, D, F, [F | R]) :-  
    C1 is C - 1, F1 is F + D, progression(C1, D, F1, R).
```

```
arithmeticProgression(AP) :- triple(C, D, F), progression(C, S, F, AP).
```



Задача (N -орки от естествени числа).

Да се реализира на Пролог предикат за генериране на всички N -орки от естествени числа.

Решение.

```
ntuple(0, 0, []).
```

```
ntuple(1, S, [S]).
```

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

```
tuple(T) :- pair(N, S), ntuple(N, S, T).
```



Задача (Подмножество с повторения).

Да се реализира на Пролог предикат за генериране на всички подмножества на дадено множество, като се позволява повторение на елементите на началното множество.

Решение.

```

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).

multisubset([], []).
multisubset(L, M) :-
    count(L, N), int(S), ntuple(N, S, T),
    repeatcorresponding(L, T, R), permuate(R, M).

```



Задача.

Понятието *rt-терм* се дефинира по следния индуктивен начин:

- ☛ [] е *rt-терм*;
- ☛ ако *A* и *B* са *rt-термове*, то също и [*A*, [*A*, *B*]] е *rt-терм*.

Да се реализира на Пролог предикат $\rho(X)$ за генериране на всички *rt-термове*.

Решение.

```

generateRank(0, []).
generateRank(R, [X, [X, Y]]) :-
    R > 0, R1 is R - 1, between(0, R1, R2),
    generateRank(R1, X), generateRank(R2, Y).
generateRank(R, [Y, [Y, X]]) :-
    R > 0, R1 is R - 1, between(0, R1, R2),
    generateRank(R1, X), generateRank(R2, Y).

\rho(X) :- int(R), generateRank(R, X).

```



❖ Предикати за работа с регулярни изрази ❖

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

Дефиниция 61 (Регулярен език).

Празният език \emptyset и $\{a\}$, за всеки символ $a \in \Sigma$, са регулярни езици. Ако L_0 и L_1 са регулярни 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 62 (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 Пролог 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], *]**.

Заб.: В Strawberry Prolog може да се наложи да се пише **'+'** и **'*'** вместо **+** и *****.

Думите над азбука Σ също ще записваме чрез списъци. Например думата $gfdxprg$ се записва като **[g, f, d, x, x, p, g]**.

Недетерминирани автомати ще представяме като списъци **[Q, S, F, D]** от четири елемента, 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 $[[Q_1, A_1, DQ_1], [Q_2, A_2, DQ_2], \dots]$, където всяка тройка $[Q_i, A_i, DQ_i]$ представлява преход от състояние Q_i към състояние DQ_i по символ A_i .

Задача (Регулярни езици).

Да се реализира на Пролог предикат за генериране и разпознаване на всички думи от езика на даден регулярен израз.

Решение.

Първо, ще реализираме предикат, който построява недетерминиран автомат по регулярен израз:

```

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).

```

Сега ще реализираме предикат, който извършва обхождане на автомата, който е подобен на предиката `path` за графи, но е специално пригоден за автомати и освен това сформира думата, която се получава при това обхождане:

```

    automatonPath([Q, S, F, D], CQ, []) :- member(CQ, F).
    automatonPath([Q, S, F, D], CQ, [A | W]) :-
        member([CQ, A, DQ], D), automatonPath([Q, S, F, D], DQ, W).

```

```

    traverse([Q, S, F, D], W) :- member(SQ, S), path([Q, S, F, D], SQ, W).

```

Така, финалното решение за предикат, който генерира или разпознава всички думи от езика на даден регулярен израз, е:

```

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

```



Задача (Регулярни езици II).

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

Упътване: Предиката `automatonPath` трябва да извършва обхождане в широчина, а не в дълбочина.

❖ Предикати за работа с контекстно-свободни граматики ❖

Фиксираме крайна азбука от символи Σ . Думи над тази азбука ще записваме както и при регулярните езици - думата *gfdxpg* се записва като `[g, f, d, x, x, p, g]`.

Контекстно-свободните граматики ще записваме на Пролог като комбинация от списък на нетерминалите `N` и списък на правилата `R`. Всяко правило ще се записва като списък от вида `[n, [a1, a2, ..., an]]`, където тези символи представляват правилото $n \rightarrow a_1 a_2 \dots a_n$. Така, например, граматиката за езика $a^n b^n$ се описва с `R = [[s, [a, s, b]], [s, []]]` и `N = [s]`.

Задача (Контекстно-свободни граматики).

Да се реализира на Пролог предикат за генериране на всички думи от езика на дадена контекстно-свободна граматика.

Решение.

Първо, реализираме предикати, които прилагат поредица правила от граматиката върху дадена дума. Прилагането отговаря на ляв извод (т.е. на всяка стъпка се заменят най-левите срещания на нетерминалите).

```
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).
```

Сега реализираме предикат, който по дадена контекстно-свободна граматика генерира всички думи от нейния език. Той работи като първо се генерира подмножество (с повторения) на множеството на правилата и се проверява дали това подмножество представлява валиден извод на дума от началния нетерминал.

```
cfg(R, N, S, W) :-  
    multisubset(R, I), applyrules([S], I, W),  
    not(member(N, X), member(W, X)).
```



Заб.: Разпознаването на това, дали дума принадлежи на езика на дадена контекстно-свободна граматика, се реализира по различен начин - граматиката се привежда в нормална форма на Чомски и след това се използва СΥΚ алгоритъма.

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 ще използваме общи правила $b_1 b_2 \dots b_m n c_1 c_2 \dots c_k \rightarrow a_1 a_2 \dots a_n$, които се записват със списъци от следния вид `[[b1, b2, ..., bm, n, c1, c2, ..., ck], [a1, a2, ..., an]]`. 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).
```