

PROGRAMARE LOGICĂ

Cursul I

Claudia MUREŞAN
cmuresan@fmi.unibuc.ro, c.muresan@yahoo.com

Universitatea din Bucureşti
Facultatea de Matematică și Informatică
Bucureşti

2019–2020, Semestrul II

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

1 Introducere

2 Inițiere în programarea în Prolog

3 Mnemonic despre proprietățile cuantificatorilor

4 Mnemonic despre funcții

5 Constantele sunt operații fără argumente

Prescurtări uzuale care vor fi folosite în lecții

- **i. e.** (*id est*) = adică
- **a. î.** = astfel încât
- **dacă** = dacă și numai dacă
- **ș. a. m. d.** = și aşa mai departe
-  : să se demonstreze că
- : contradicție

Paradigme ale programării calculatoarelor

Programarea imperativă:

- programatorul trebuie să-i spună calculatorului, pas cu pas, ce să facă: "parcurge cu un indice această mulțime, la fiecare iterație verifică această condiție..." etc..

Limbaje de programare imperativă: *Pascal, C, Java* etc..

Programarea logică/declarativă:

- programatorul descrie un cadru de lucru, și cere calculatorului să rezolve o cerință în acel cadru;
- pe baza unui backtracking și a altor tehnici de programare încorporate în interpretorul/compilatorul limbajului de programare logică folosit, calculatorul determină proprietățile pe care le poate folosi pentru a rezolva cerința respectivă și ordinea în care trebuie să le aplice.

Limbaje de programare logică/declarativă

- *Prolog* (PROGRAMMING IN LOGIC): bazat pe **predicate**, pe relații între obiecte; utilizat pentru jocuri/deduçții logice, în procesarea limbajului natural etc.;
- *CafeObj, Maude*: bazate pe **specificații**, pe descrierea unor tipuri de structuri algebrice; destinate demonstrării automate de proprietăți matematice; utilizate și în verificarea sistemelor software;
- *Haskell*: bazat pe **funcții și recursii**;

recursia este foarte importantă în toate limbajele de programare logică: este principalul mijloc de calcul, înlocuind, de exemplu, instrucțiunile repetitive

- etc..

1 Introducere

2 Inițiere în programarea în Prolog

3 Mnemonic despre proprietățile cuantificatorilor

4 Mnemonic despre funcții

5 Constantele sunt operații fără argumente

Siteul SWI-Prolog

The screenshot shows a web browser window displaying the SWI-Prolog website at <https://www.swi-prolog.org>. The page features a header with the SWI-Prolog logo (an owl icon) and the text "Robust, mature, free. Prolog for the real world." Below the header is a navigation menu with links for HOME, DOWNLOAD, DOCUMENTATION, TUTORIALS, COMMUNITY, USERS, and WIKI. A main content area contains a paragraph about the history and applications of SWI-Prolog, followed by three calls-to-action: "Download SWI-Prolog", "Get Started", and "Try SWI-Prolog online". At the bottom of the page is a search bar labeled "SEARCH DOCUMENTATION:" with a "GO" button. The browser's address bar shows the URL "swi-prolog.org". The operating system taskbar at the bottom includes icons for File Explorer, Edge, File Manager, and Task View, along with system status indicators like battery level and date/time.

Din josul paginii de la această adresă:

- se poate descărca SWI-Prolog;
- se poate căuta în documentația SWI-Prolog;
- se poate accesa versiunea online a SWI-Prolog.

De exemplu, dând o căutare după "boolean expressions", găsim:

The screenshot shows a browser window displaying the SWI-Prolog Manual. The URL is swi-prolog.org/pldoc/man?section=clpb-exprs. The left sidebar contains a navigation tree with sections like Documentation, Reference manual, The SWI-Prolog library, library(clpb): CLP(B): Constraint Logic, Introduction, Boolean expressions, Interface predicates, Examples, Obtaining BDDs, Enabling monotonic CLP(B), Example: Pigeons, Example: Boolean circuit, Acknowledgments, CLP(B) predicate index, and Packages. The main content area is titled "A.8.2 Boolean expressions". It defines a Boolean expression as one of the following:

0	false
1	true
variable	unknown truth value
atom	universally quantified variable
$\neg Expr$	logical NOT
$Expr + Expr$	logical OR
$Expr * Expr$	logical AND
$Expr \# Expr$	exclusive OR
$Var \wedge Expr$	existential quantification
$Expr =:= Expr$	equality
$Expr \neq Expr$	disEquality (same as #)
$Expr \leq Expr$	less or equal (Implication)
$Expr \geq Expr$	greater or equal
$Expr < Expr$	less than
$Expr > Expr$	greater than
$card(Is, Exprs)$	cardinality constraint (see below)
$+(Exprs)$	n-fold disjunction (see below)
$*(Exprs)$	n-fold conjunction (see below)

Below this table, it says "where $Expr$ again denotes a Boolean expression." Further down, it explains the `card` predicate: "The Boolean expression `card(Is, Exprs)` is true iff the number of true expressions in the list `Exprs` is a member of the list `Is` of integers and integer ranges of the form `From-To`. For example, to state that precisely two of the three variables `X`, `Y` and `Z` are true, you can use `sat(card([2], [X, Y, Z]))`". It also notes that `+(Exprs)` and `*(Exprs)` denote disjunction and conjunction respectively.

At the bottom, there is a toolbar with various icons (Windows Start, Search, Task View, Edge, File Explorer, Taskbar, Google Chrome, etc.) and a system tray with icons for battery, signal, volume, and date/time (5:08 PM, ENG, 2/12/2020).

Structura unui program în Prolog

Un program în Prolog este format din **clauze**; acestea sunt de trei tipuri:

① fapte:

propoziție.

predicat(listă de variabile și constante).

predicatele exprimă relații între obiecte, proprietăți ale obiectelor, și au valori booleene;

propozițiile sunt predicatele fără variabile, i.e. constantele booleene;

faptele semnifică proprietăți întotdeauna adevărate;

② reguli:

fapt :- succesiune de fapte.

faptele din partea dreaptă a unei reguli pot fi separate prin virgulă (care reprezintă *conjuncția logică*), punct și virgulă (*disjuncția logică*, având prioritate mai mică decât conjuncția) sau alți conectori logici (a se vedea slideul anterior); simbolul ":" (două puncte și minus) este numit "neck";

(semnificația unei reguli)

are loc acest fapt :- dacă au loc aceste fapte.

③ **întrebări (interrogări)**: formate din **scopuri**, constând în predicate care trebuie satisfăcute de către Prolog: prin scopurile din interrogări cerem



Prologului să determine dacă un predicat e adevărat sau valori ale variabilelor pentru care acel predicat e adevărat.

Faptele și regulile formează **baza de cunoștințe**.

Numele de **variabile** în Prolog încep cu *literă mare* sau *underscore* (_).

Variabilă nedenumită: *underscore* (_):

- simbol generic pentru variabile care apar o singură dată într-un fapt sau o regulă;
- sunt folosite doar pentru a indica locații de variabile, nu și pentru a lucra cu acele variabile.

Orice alt nume, inclusiv siruri de caractere cuprinse între apostrofuri, denumește o constantă, o funcție cu argumente (i.e. operație cu cel puțin un operand) sau un predicat.

Sintaxa pentru liste în Prolog:

[]	lista vidă
[$elem_1, elem_2, \dots, elem_n$]	lista formată din elementele $elem_1, elem_2, \dots, elem_n$
[$elem_1, elem_2, \dots, elem_n T$]	lista cu primele n elemente $elem_1, elem_2, \dots, elem_n$ și coada T

Exemplu

$$[1, 2, 3, 4] = [1, 2, 3, 4 | []] = [1 | [2, 3, 4]] = [1, 2 | [3, 4]] = [1 | [2, 3 | [4]]] = [1 | [2, 3 | [4 | []]]]$$

Exemplu

Cum putem scrie predicate pentru:

- calculul lungimii unei liste;
- concatenarea a două liste?

Să scriem un predicat $\text{lung}(L, N)$, care este satisfăcut dacă:

- L este o listă, N este un număr natural și
- N este lungimea listei L , adică numărul elementelor lui L :

```
lung([], 0).
```

```
lung([_|T], N) :- lung(T, K), N is K + 1.
```

Operatorul **is** produce executarea calculului în acea expresie aritmetică.

Înlocuiți **is** cu **=** și vedeți ce obțineți!

Și un predicat $\text{concat}(L1, L2, L)$, care este satisfăcut dacă:

- $L1$, $L2$ și L sunt liste și
- concatenarea listei $L1$ cu $L2$ este L :

```
concat([], L, L).
```

```
concat([H|T], L, [H|M]) :- concat(T, L, M).
```

Încărcăm acest program în versiunea online a SWI-Prolog

The screenshot shows the SWISH (SWI-Prolog for SHaring) web interface. On the left, the code editor displays a Prolog program with various predicates and their implementations. On the right, the execution results are shown in a list format.

Code Editor (Left):

```
% Aici: fapte și reguli.  
%  
lung([],0). % fapt: intotdeauna adevarat  
lung([_|T],N) :- lung(T,K), N is K+1. % regula: are structura:  
% are loc acest fapt :- daca au loc aceste fapte.  
%  
/* Daca denumim capul listei [_|T] in Loc sa-l dam ca  
* variabila nedenumita: _, atunci primim avertismentul:  
* variabila singleton. */  
%  
concat([],L,L). % fapt  
concat([H|T],L,[H|M]) :- concat(T,L,M). % regula  
%  
% Aici: intrebări: formate din scopuri. ----->
```

Execution Results (Right):

- lung([a,b,c],3). → true
- lung([a,b,c],5). → false
- lung([a,b,c,d,e],Cat). → Cat = 5
- concat([1,2],[3,4,5],[1,2,3,4,5]). → true
- concat([1,2],[],[1,2,3,4,5]). → false
- concat([a,b,c],[1,2,3,4,5],CeLista). → CeLista = [a, b, c, 1, 2, 3, 4, 5]
- ?- concat([a,b,c],[1,2,3,4,5],CeLista).

At the bottom, there are navigation links: Examples, History, Solutions, table results, Run!, and a refresh icon.

Interogări cu scopuri sau variabile multiple

The screenshot shows the SWISH interface for SWI-Prolog. On the left, there is a code editor window containing Prolog code. On the right, several execution results are shown in separate windows.

Code Editor (Left):

```
% Aici: fapte si reguli.  
%  
lung([],0). % fapt: intotdeauna adevarat  
lung([_|T],N) :- lung(T,K), N is K+1. % regula: are structura:  
% are loc acest fapt :- daca au loc aceste fapte.  
%  
/* Daca denumim capul listei [_|T] in Loc sa-l dam ca  
* variabila nedenumita: _, atunci primim avertismentul:  
* variabila singleton. */  
%  
concat([],L,L). % fapt  
concat([H|T],L,[H|M]) :- concat(T,L,M). % regula  
%  
% Ati observat sintaxa pentru comentarii:  
% pe un rand  
% pe mai  
% multe randuri /  
%  
% Sa punem si intrebari formate din mai multe scopuri.  
%  
% Sa vedem si intrebari care au mai multe raspunsuri.  
% Sa cerem cu "Next" mai multe rezultate.  
% In loc de "Next", in versiunea desktop a SWI-Prolog, se foloseste ";".  
% La final, se afiseaza "false", semnificand ca nu mai exista alte solutii.  
%
```

Execution Results (Right):

- Result 1:

```
concat([a,b,c],[1,2,3],Ce).concat(Ce,[10,-1],Alta).lung(Alta,Cat).
```

Alta = [a, b, c, 1, 2, 3, 10, -1].
Cat = 8.
Ce = [a, b, c, 1, 2, 3]
- Result 2:

```
concat(CeLista,[1,2,3],[-1,0,1,2,3]).
```

CeLista = [-1, 0]
false
- Result 3:

```
concat([1,2,3],CuCeLista,[1,2,3,4,5]).
```

CuCeLista = [4, 5]
- Result 4:

```
concat(CeLista,CuCeLista,[1,2,3]).
```

CeLista = [].
CuCeLista = [1, 2, 3]
CeLista = [1].
CuCeLista = [2, 3]
CeLista = [1, 2].
CuCeLista = [3]

At the bottom, there are buttons for navigation: Next, 10, 100, 1.000, Stop, and a search bar with placeholder text "Search".

At the very bottom, there are links for Examples, History, Solutions, and Run, along with a "table results" checkbox.

Întrebări cu mai multe răspunsuri posibile

SWISH -- SWI-Prolog for SHaring

swish.swi-prolog.org

SWISH

File ▾ Edit ▾ Examples ▾ Help ▾

Program +

```
1 % Aici: fapte și reguli.
2
3 lung([],0).                                % fapt: intotdeauna adevarat
4 lung([_|T],N) :- lung(T,K), N is K+1.      % regula: are structura:
5          % are loc acest fapt :- daca au loc aceste fapte.
6
7 /* Daca denumim capul listei [_|T] in Loc sa-l dam ca
8 * variabila nenumita: _, atunci primim avertismentul:
9 * variabila singleton. */
10
11 concat([],L,L).                           % fapt
12 concat([H|T],L,[H|M]) :- concat(T,L,M). % regula
13
14
15 % Ati observat sintaxa pentru comentarii:
16    % pe un rand
17    /* pe mai
18     * multe randuri */|
19
20
21
22 % Sa punem si intrebări formate din mai multe scopuri.
23
24 % Sa vedem si intrebări care au mai multe raspunsuri.
25 % Sa cerem cu "Next" mai multe rezultate.
26 % In loc de "Next", in versiunea desktop a SWI-Prolog, se foloseste ";".
27 % La final, se afiseaza "false", semnificand ca nu mai exista alte solutii.
28
```

340 users online

Search

concat(CeLista,CuCeLista,[1,2,3]).

CeLista = [],
CuCeLista = [1, 2, 3]
CeLista = [1],
CuCeLista = [2, 3]
CeLista = [1, 2],
CuCeLista = [3]
CeLista = [1, 2, 3],
CuCeLista = []
false

concat([1,2,X,Y],[5,Z,7,T],[1,2,3,4,5,6,7]).

false

concat([1,2,X,Y],[5,Z,7],[1,2,3,4,5,6,7]).

X = 3,
Y = 4,
Z = 6

?-

Examples ▾ History ▾ Solutions ▾

Run!

Vom vedea cum găsește Prologul aceste răspunsuri

The screenshot shows a browser window for the SWISH-Prolog sharing site. The title bar says "SWISH -- SWI-Prolog for SHaring". The main area has tabs for "Program" and "Examples". The "Program" tab contains the following Prolog code:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 % Sa vedem si intrebari care au mai multe raspunsuri.
23 % Sa cerem cu "Next" mai multe rezultate.
24 % In loc de "Next", in versiunea desktop a SWI-Prolog, se foloseste ";".
25 % La final, se afiseaza "false", semnificand ca nu mai exista alte solutii.
26
```

The right side shows the results of two queries:

Query 1: `concat([1,2,X,Y],L,[1,2,3,4,5,6]).`

Results for Query 1:

```
L = [5, 6].
X = 3,
Y = 4
```

Query 2: `concat([1,2,X,Y|T],L,[1,2,3,4,5,6]).`

Results for Query 2:

```
L = [5, 6].
T = [],
X = 3,
Y = 4
L = [6],
T = [5],
X = 3,
Y = 4
L = [],
T = [5, 6],
X = 3,
Y = 4
false
```

At the bottom, there is a query input field: `?- concat([1,2,X,Y|T],L,[1,2,3,4,5,6]).`

At the very bottom, there are buttons for "Examples", "History", "Solutions", "table results", and "Run".

Vedeți și celelalte opțiuni ale SWI-Prolog online

The screenshot shows the SWISH web-based SWI-Prolog environment. At the top, there's a header bar with the title "SWISH -- SWI-Prolog for SHaring" and a search bar. Below the header, there are tabs for "Program" (selected), "Examples", and "Solutions". The main area has a toolbar with icons for file operations, examples, and help.

Program Tab:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
```

Output Area:

```
L = [1, 2, 3].
M = []

concat(L,M,[1,2,3,4,5]).
```

```
L = [].
M = [1, 2, 3, 4, 5]
L = []
M = [2, 3, 4, 5]
L = [1, 2]
M = [3, 4, 5]
L = [1, 2, 3]
M = [4, 5]
L = [1, 2, 3, 4]
M = [5]
L = [1, 2, 3, 4, 5]
M = []
```

```
concat(L,M,[1,2,3,4,5,6,7]).
```

```
L = []
M = [1, 2, 3, 4, 5, 6, 7]
```

Query Input:

```
?- concat(L,M,[1,2,3,4,5,6,7]).
```

Buttons at the bottom:

Examples ▾ History ▾ Solutions ▾ table results Run!

Inclusiv opțiuni pentru editare rapidă: vedeți butonul "History"

The screenshot shows the SWISH interface for SWI-Prolog. On the left, there is a code editor window titled "Program" containing Prolog predicates for calculating the length of a list and concatenating two lists. The code is as follows:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Below the code editor, a message indicates that 10 results were found.

On the right, there are three execution history panes. The first pane shows the execution of `concat(L,M,[1,2,3,4,5]).` The second pane shows the execution of `concat(L,M,[1,2,3,4,5,6,7]).` The third pane shows the query `?- concat(L,M,[1,2,3,4,5,6,7]).` The results for each execution are listed sequentially, showing the state of variables L and M at each step.

Și acum în versiunea desktop a SWI-Prolog

Am scris cele două predicate de mai sus într-un fișier text cu extensia `.pl`, care poate fi încărcat în Prologul desktop cu **File → Consult** din meniu ferestrei interpretorului Prolog sau folosind comanda (attenție la [], apostrofuri și punct): `'unitate_disc:/calea_catre_fisier/nume_fisier.pl'`.

The screenshot shows the SWI-Prolog desktop application window. The title bar reads "SWI-Prolog (Multi-threaded, version 8.0.3)". The menu bar includes File, Edit, Settings, Run, Debug, Help. A note in the menu bar says "For online help and background, visit <http://www.swi-prolog.org>". Below the menu is a command line:
?- ['d:/work/cursurilemele/pl_2019_2020/laborator_pl_2019_2020/exspliste.pl'].
true.
?- concat(L, M, [a, b, c]).
L = [].
M = [a, b, c] ;
L = [b, c] ;
M = [a, b] ;
L = [c] ;
L = [] ; b, c).
M = [] ;
false.
?- trace
true.
[trace] ?- concat(L, M, [a, b, c]).
Call: (8) concat([], [a, b, c]) ? creep
Redo: (8) concat([], [a, b, c]) ? creep
L = [].
M = [a, b, c] ;
Call: (9) concat([], [a, b, c]) ? creep
Redo: (9) concat([], [a, b, c]) ? creep
L = [].
M = [a, b, c] ;
Call: (9) concat([4518, -4520, [a, b, c]]) ? creep
Redo: (9) concat([4518, -4520, [a, b, c]]) ? creep
L = [].
M = [a, b, c] ;
Call: (9) concat([4792, -4520, [b, c]]) ? creep
Redo: (9) concat([4792, -4520, [b, c]]) ? creep
L = [].
M = [b, c] ;
Call: (9) concat([4792, -4520, [c]]) ? creep
Redo: (9) concat([4792, -4520, [c]]) ? creep
L = [].
M = [c] ;
Call: (10) concat([4798, -4520, []]) ? creep
Redo: (10) concat([4798, -4520, []]) ? creep
L = [].
M = [] ;
Call: (11) concat([4804, -4520, []]) ? creep
Redo: (11) concat([4804, -4520, []]) ? creep
L = [].
M = [] ;
Fail: (10) concat([4798, -4520, []]) ? creep
Fail: (9) concat([4792, -4520, [b, c]]) ? creep
Fail: (8) concat([4518, -4520, [a, b, c]]) ? creep
false.
[trace] ?- notrace.
true.

Comanda `trace` produce detalierea pașilor urmați de Prolog pentru a rezolva interogările. Renunțare la această afișare detaliată: `notrace`.

trace și săgeată "Step into" în SWI-Prolog online

The screenshot shows the SWISH online Prolog interface. On the left, the code editor displays two predicates:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
```

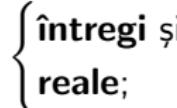
The right side shows the trace window for the call `trace.concat(L,M,[sgl]).`. The trace output is as follows:

```
Call: concat(_3768, _3764, [sgl])
Exit: concat([], [sgl], [sgl])
L = [],
M = [sgl]
Redo: concat(_3768, _3764, [sgl])
Call: concat(_4448, _3764, [])
Exit: concat([], [], [])
Exit: concat([sgl], [], [sgl])
L = [sgl],
M = []
Redo: concat(_4448, _3764, [])
Fail: concat(_4448, _3764, [])
Fail: concat(_3768, _3764, [sgl])
false
```

Below the trace window, the query `?- trace.concat(L,M,[1,2,3]).` is entered in the query input field.

Tipurile de date în Prolog

Practic, există un *unic tip de date*: **termenii**, cu, câteva *subtipuri*:

- **variabilele**;
- **atomii**: sirurile de caractere speciale și constantele, inclusiv sirurile de caractere cuprinse între apostrofuri, dar excludând lista vidă [] și numerele;
- **numerele**  **intregi** și **reale**;
- **termenii compuși**: $f(arg_1, \dots, arg_n)$, unde $n \in \mathbb{N}^*$, f este o funcție (și predicatele sunt tot funcții) n -ară (adică de aritate n , cu n argumente), iar arg_1, \dots, arg_n sunt **termeni**.

Observație (predicate, i.e. cu valori booleene, nu orice fel de funcții)

Faptele constau din predicate.

Regulile sunt formate dintr-un predicat urmat de simbolul neck, urmat de alte predicate unite prin conectori logici.

Interogările sunt formate din predicate prin conectori logici.

Observație (imbricarea)

Sintaxa Prolog permite imbricarea predicatelor cu operații arbitrară, dar imbricarea doar construiește termeni, nu produce, de exemplu, calcule aritmetice în subtermeni (a se vedea predicatele *e*, *f* din *exlisteetc.pl*).

Operatori Prolog și predicate predefinite în Prolog

Exemplu

Nu putem interoga (pentru că nu este predicat: rezultatul e aritmetic, nu boolean):

?- -((10-1)*2-3)/5.

dar putem interoga:

?- X is -((10-1)*2-3)/5.

// sau *div* câtul împărțirii întregi

Alți operatori aritmetici: *mod* restul împărțirii întregi
 ** ridicare la putere

A se vedea, în fișierul *test.pl* din prima lecție de laborator, predicatul **not** sau **\+** și operatorii de comparație între valori aritmetice și de testare a egalității ca expresii și ca valori aritmetice.

Predicatul **zeroar** (i.e. de aritate 0, i.e. fără argumente) **fail** este întotdeauna evaluat la **FALS**.

(Predicate unare (i.e. de aritate 1, i.e. cu câte un singur argument))

Predicalele **number**, **integer**, **float** testează dacă argumentul lor este număr, respectiv număr întreg, respectiv număr real (în format float).

Predicatelor predefinite **var**, **nonvar**, **atom**, **atomic** testează dacă argumentul lor este variabilă, respectiv non-variabilă, respectiv atom, respectiv atom sau număr sau lista vidă [].

?- var(A). true.	?- atom(a). true.	?- number(a). false.
?- nonvar(A). false.	?- atom(alta_constanta). true.	?- number(10+2). false.
?- var(_). true.	?- atom(-#-->). true.	?- number(12). true.
?- nonvar(_). false.	?- atom(1). false.	?- number(12.5e2). true.
?- var('A'). false.	?- atom([]). false.	?- integer(5). true.
?- nonvar('A'). true.	?- atom(p(X)). false.	?- integer(-5). true.
?- var(a). false.	?- atom(X). false.	?- integer(5.0). false.
?- nonvar(a). true.	?- atomic(a). true.	?- float(5). false.
?- var(1). false.	?- atomic(alta_constanta). true.	?- float(5.0). true.
?- nonvar(1). true.	?- atomic(-#-->). true.	?- float(-5.0e2). true.
?- var(p(X)). false.	?- atomic(1). true.	?- float(-5e2). true.
?- nonvar(p(X)). true.	?- atomic([]). true.	?- float(5e0). true.
?- var(-->). false.	?- atomic(p(X)). false.	?- float(-15.25e-2). true.
?- nonvar(-->). true.	?- atomic(X). false.	?- float(-15.25E-2). true.

Pentru a răspunde la o interogare, Prologul încearcă să UNIFICE **scopurile** din **interogare** cu predicatele din:

- **fapte**;
- **membrii stângi ai regulilor** – dacă o astfel de unificare reușește, atunci **următorul scop** pe care va încerca să-l satisfacă este membrul drept al aceleiași reguli, cu argumentele rezultate în urma unificării.

(UNIFICARE (orientativ) – detalii într-un curs următor)

Două funcții UNIFICĂ dacă: $\begin{cases} \text{acele funcții coincid și} \\ \text{au aceleași argumente:} \end{cases}$

$$f(arg_1, \dots, arg_n) = g(a_1, \dots, a_k) \iff \begin{cases} f = g (\implies n = k) \text{ și} \\ arg_1 = a_1, \\ \vdots \\ arg_n = a_n. \end{cases}$$

Mai precis: două funcții UNIFICĂ dacă: $\begin{cases} \text{acele funcții coincid și} \\ \text{RECURSIV, argumentele lor UNIFICĂ.} \end{cases}$

RECURSIA merge până la **termeni fără argumente**, adică **variabile** sau **constante**:

- **constantele** sunt **operații** (anume exact **operațiile fără argumente** – vom vedea), aşadar, conform regulii de mai sus, NU UNIFICĂ DECÂT **cu ele însesele**;
- **variabilele** UNIFICĂ, CU **orice termen care nu le conține**; (desigur, o unificare de tipul $X = h(X)$ ar conduce la $X = h(X) = h(h(X)) = h(h(X)) = \dots$, deci nu ar fi corectă.)

Exemplu (lista vidă nu unifică, cu nicio listă nevidă)

Lista vidă `[]` e o constantă, i.e. operație fără argumente (vom vedea), iar o listă nevidă `[H|T]` (care conține cărui elementul H) este termenul format din operația binară `[|]` cu argumentele H și T (`[|](H, T)`). Cum operațiile `[]` și `[|]` nu coincid, rezultă că `[]` nu unifică, cu `[H|T]`.

Exemplu (utilitate *var*, *nonvar*)

Să numărăm aparițiile unui element într-o listă, cu un predicat ternar *numar_aparitii*(*Element*, *Lista*, *NrAparitii*).

Următoarea implementare numără, de fapt, elementele listei cu care unifică acel element:

numar_aparitii(*_*, `[]`, 0).

numar_aparitii(*H*, `[H|T]`, *N*) :- *numar_aparitii*(*H*, *T*, *K*), *N* is *K* + 1.

numar_aparitii(*X*, `[H|T]`, *N*) :- *X\= H*, *numar_aparitii*(*X*, *T*, *N*).

Cu implementarea de mai sus, iată răspunsul Prologului la această interogare:

```
?- numar_aparitii(a,[X,a,1,a,a,2,3,Y,a,4,Z],N).  
X = Y, Y = Z, Z = a,  
N = 7 ;  
false.
```

Am cerut mai multe soluții, cu ";" , și răspunsul Prologului a fost "false" , așadar acesta e singurul răspuns găsit de Prolog la interogarea de mai sus.

În schimb, dacă implementăm astfel predicatul de mai sus:

```
numar_aparitii([],[],0).  
numar_aparitii(X,[H|T],N) :- var(H), numar_aparitii(X,T,N).  
numar_aparitii(X,[H|T],N) :- nonvar(H), X = H, numar_aparitii(H,T,K), N is K + 1.  
numar_aparitii(X,[H|T],N) :- X \= H, numar_aparitii(X,T,N).
```

atunci răspunsul unic găsit de Prolog la aceeași interogare este:

```
?- numar_aparitii(a,[X,a,1,a,a,2,3,Y,a,4,Z],N).  
N = 4 ;  
false.
```

A se vedea, în fișierul *.pl* și lecțiile video screenshot atașate, scrierea unui predicat care determină exact elementele listei **identice** cu acel element.

Cum răspunde Prologul la interogări?

Considerăm baza de cunoștințe:

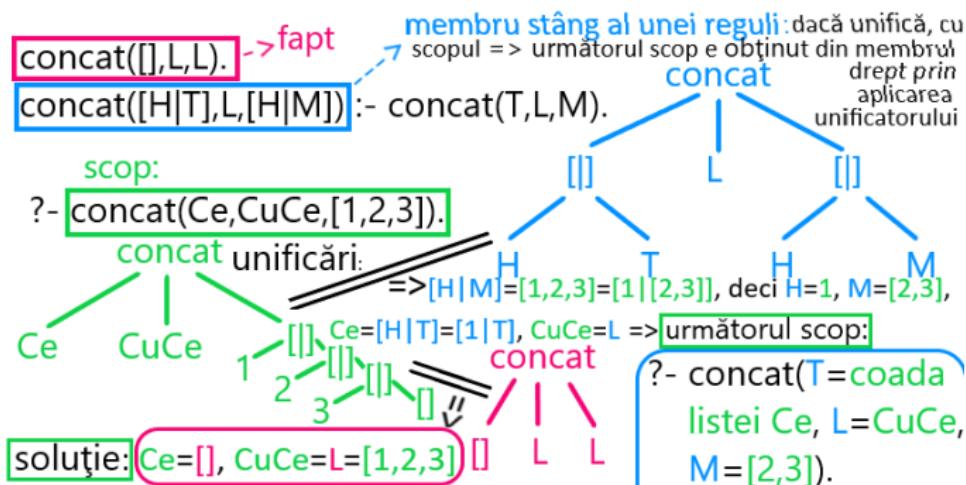
concat([], L, L).

concat([H|T], L, [H|M]) :- concat(T, L, M).

Să vedem cum răspunde Prologul la interogarea:

?- concat(Ce,CuCe,[1,2,3]).

Procedul de rezolvare a acestei interogări este cel descris mai sus:



Vom vedea într-un curs următor ce este un *unificator*.

Dacă dăm trace și apoi interogarea:

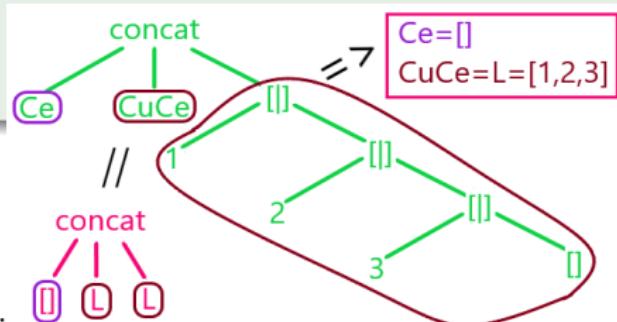
?- concat(Ce,CuCe,[a,b,c]).

observăm din pașii afișați că primul lucru executat de Prolog este redenumirea variabilelor din interogare în nume de forma *_număr*.

Dacă în baza de cunoștințe, avem și definițiile altor predicate, de exemplu definiția de mai sus a predicatului lung, atunci:

cum concat ≠ lung, scopul concat(Ce, CuCe, [a, b, c]) nu unifică:

- nici cu lung([], 0),
- nici cu lung([H|T], N).



În schimb, concat(Ce,CuCe,[1,2,3]) unifică:

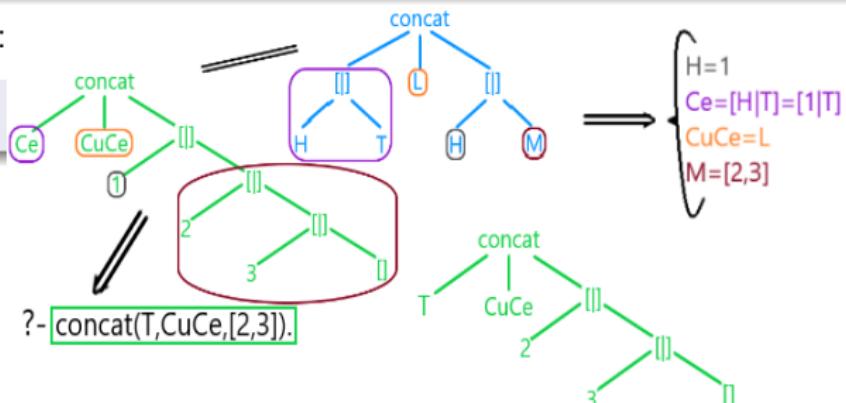
cu concat([], L, L), pentru **Ce** = [] și **CuCe** = **L** = [1, 2, 3] – aceasta este *prima soluție* a interogării,

dar și:

cu $\text{concat}([\text{H}|\text{T}], \text{L}, [\text{H}|\text{M}])$, pentru $\text{H} = 1$, $\text{Ce} = [1|\text{T}]$, $\text{CuCe} = \text{L}$ și $\text{M} = [2, 3]$,

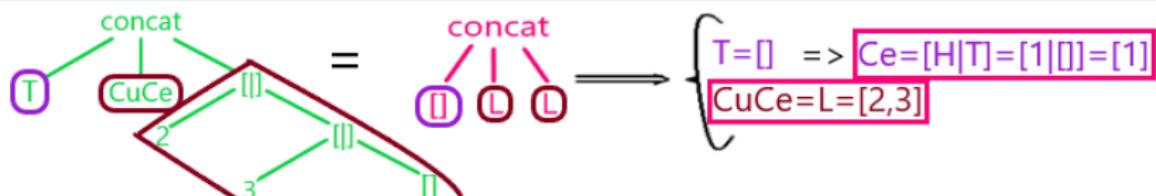
așadar **următorul scop** este:

?- $\text{concat}(\text{T}, \text{CuCe}, [2, 3]).$



Acesta unifică:

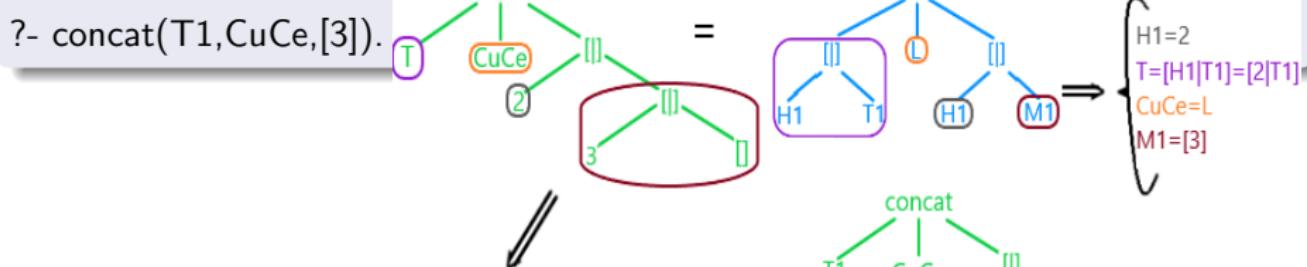
cu $\text{concat}([], \text{L}, \text{L})$, pentru $\text{T} = []$ și $\text{CuCe} = \text{L} = [2, 3]$ – așadar a doua soluție a interogării este: $\text{Ce} = [1|\text{T}] = [1|[]] = [1]$ și $\text{CuCe} = [2, 3]$,



dar și:

cu $\text{concat}([\text{H1}|\text{T1}], \text{L}, [\text{H1}|\text{M1}])$, pentru $\text{H1} = 2$, $\text{T} = [2|\text{T1}]$, $\text{Ce} = \text{L}$ și $\text{M1} = [3]$,

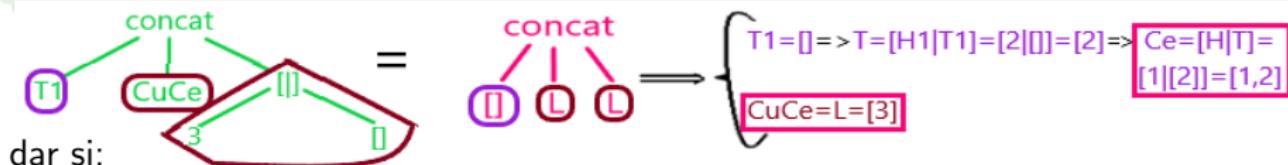
așadar **următorul scop** este:



Acesta unifică:

?- concat(T1,CuCe,[3]).

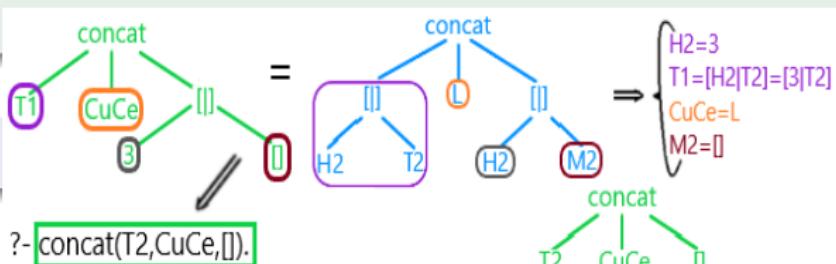
cu concat([], L, L), pentru **T1** = [] și **CuCe** = **L** = [3] – aşadar a treia soluție a interogării este: **Ce** = [1|**T**] = [1|[2|**T1**]] = [1|[2|[]]] = [1, 2] și **CuCe** = [3],



cu concat([H2|T2], L, [H2|M2]), pentru **H2** = 3, **T1** = [3|**T2**], **CuCe** = **L** și **M2** = [],

așadar **următorul scop** este:

?- concat(T2,CuCe,[]).



Întrucât:

constanta [] nu unifică, cu termenul [H3|M3], având operația dominantă [|],

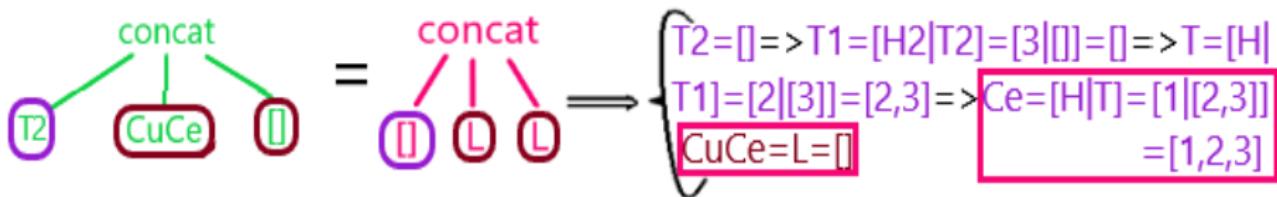
avem:

concat(T2, CuCe, []) nu unifică, cu concat([H3|T3], L, [H3|M3]).

Prin urmare:

concat(T2, CuCe, []) unifică numai cu concat([], L, L), pentru **T2 = []** și
CuCe = L = [] – aşadar a patra și ultima soluție a interogării este:

Ce = [1|T] = [1|[2|T1]] = [1|[2|[3|T2]]] = [1|[2|[3|[]]]] = [1, 2, 3] și **CuCe = []**.



În cazul unui scop compus, de exemplu:

?- concat(A,[2,3],[1,2,3]),concat(A,[4,5],C).

toate scopurile trebuie satisfăcute, cu **aceleași valori** pentru **variabilele comune**.

Pentru exemplul de mai sus, *unica soluție*: **A = [1]**, **C = [1, 4, 5]**.

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Cuantificatorii și simbolul $\exists!$

Cuantificatorii:

- *cuantificatorul universal:* \forall
- *cuantificatorul existențial:* \exists

Notație

Alăturarea de simboluri $\exists!$ semnifică “există un unic”, “există și este unic”.

Observație

$\exists!$ nu este un cuantificator, ci este o notație prescurtată pentru enunțuri compuse: dacă x este o variabilă, iar $p(x)$ este o proprietate asupra lui x , atunci scrierea $(\exists! x) (p(x))$ este o abreviere pentru enunțul scris, desfășurat, astfel:

$$(\exists x) (p(x)) \text{ și } (\forall y) (\forall z) [(p(y) \text{ și } p(z)) \Rightarrow y = z],$$

unde y și z sunt variabile.

(negarea enunțurilor cuantificate)

Dacă x este o variabilă, iar $p(x)$ este o proprietate referitoare la x (mai precis o proprietate referitoare la elementele pe care le parurge/le poate denumi x), atunci:

- non $[(\forall x) (p(x))]$ $\Leftrightarrow (\exists x) (\text{non } p(x))$
- non $[(\exists x) (p(x))]$ $\Leftrightarrow (\forall x) (\text{non } p(x))$

Negarea enunțurilor cuantificate

Cum se neagă un enunț cu mai mulți cuantificatori? Aplicând proprietățile de mai sus, și iterând acest procedeu:

Exemplu

Fie x, y, z, t, u variabile, iar $p(x, y, z, t, u)$ o proprietate depinzând de x, y, z, t, u .
Atunci:

$$\begin{aligned} \text{non } [(\forall x)(\forall y)(\exists z)(\forall t)(\exists u)(p(x, y, z, t, u))] &\Leftrightarrow \\ (\exists x)[\text{non } [(\forall y)(\exists z)(\forall t)(\exists u)(p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x)(\exists y)[\text{non } [(\exists z)(\forall t)(\exists u)(p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x)(\exists y)(\forall z)[\text{non } [(\forall t)(\exists u)(p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x)(\exists y)(\forall z)(\exists t)[\text{non } [(\exists u)(p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x)(\exists y)(\forall z)(\exists t)(\forall u)(\text{non } p(x, y, z, t, u)) \end{aligned}$$

Nu vom mai aplica acest procedeu pas cu pas. Reținem că procedeul constă în transformarea fiecărui cuantificator universal într-unul existențial și invers, și negarea proprietății de sub acești cuantificatori.

Cuantificatori aplicați fixând un domeniu al valorilor

În scierile de mai sus, domeniile în care pot lua valori variabilele sunt neprecizate, putând fi considerate, de exemplu, *universul discuției* (i.e. colecția tuturor elementelor/obiectelor cu care se lucrează în problema curentă):

- $\forall x$ semnifică: oricare ar fi x "de oriunde" sau "din universul discuției";
- $\exists x$ semnifică: există x "de oriunde" sau "în universul discuției".

Fie M o mulțime, x o variabilă, iar $p(x)$ o proprietate referitoare la elementele lui M . Atunci următoarele scieri sunt abrevieri pentru scierile fără domeniu al valorilor lângă cuantificatori:

- $(\forall x \in M) (p(x)) \stackrel{\text{not.}}{\Leftrightarrow} (\forall x) (x \in M \Rightarrow p(x))$
- $(\exists x \in M) (p(x)) \stackrel{\text{not.}}{\Leftrightarrow} (\exists x) (x \in M \text{ și } p(x))$

Toate proprietățile logice pentru enunțuri cuantificate, inclusiv negarea enunțurilor cuantificate de mai sus și comutarea cunatificatorilor de mai jos, se scriu la fel și sunt valabile și pentru cuantificatori urmați de un domeniu al valorilor pentru variabila cuantificată.

Cuantificatorii de același fel comută, cei diferenți nu

Fie x și y variabile, iar $p(x, y)$ o proprietate asupra lui x și y . Atunci:

- $(\forall x)(\forall y)(p(x, y)) \Leftrightarrow (\forall y)(\forall x)(p(x, y))$
- $(\exists x)(\exists y)(p(x, y)) \Leftrightarrow (\exists y)(\exists x)(p(x, y))$
- $(\forall x)(\exists y)(p(x, y)) \not\Leftrightarrow (\exists y)(\forall x)(p(x, y))$ (pentru fiecare valoare a lui x , valoarea lui y pentru care e satisfăcut enunțul din stânga depinde de valoarea lui x)

Analog, dacă A și B sunt multimi, avem:

- $(\forall x \in A)(\forall y \in B)(p(x, y)) \Leftrightarrow (\forall y \in B)(\forall x \in A)(p(x, y))$
- $(\exists x \in A)(\exists y \in B)(p(x, y)) \Leftrightarrow (\exists y \in B)(\exists x \in A)(p(x, y))$
- $(\forall x \in A)(\exists y \in B)(p(x, y)) \not\Leftrightarrow (\exists y \in B)(\forall x \in A)(p(x, y))$

Desigur, la fel pentru cazul în care doar unul dintre cuantificatori este aplicat cu un domeniu al valorilor pentru variabila cuantificată:

$(\forall x)(\forall y \in B)(p(x, y)) \Leftrightarrow (\forall y \in B)(\forall x)(p(x, y))$ etc..

Exemplu

Enunțul $(\forall x \in \mathbb{N})(\exists y \in \mathbb{Z})(x + y = 0)$ este adevărat.

Enunțul $(\exists y \in \mathbb{Z})(\forall x \in \mathbb{N})(x + y = 0)$ este fals.

Cuantificatori, disjuncții și conjuncții logice

Să observăm și următoarele proprietăți logice: dacă x este o variabilă, iar $p(x)$ și $q(x)$ sunt enunțuri referitoare la x , atunci:

- $(\forall x)(p(x) \text{ și } q(x)) \Leftrightarrow (\forall x)(p(x)) \text{ și } (\forall x)(q(x))$
- $(\exists x)(p(x) \text{ sau } q(x)) \Leftrightarrow (\exists x)(p(x)) \text{ sau } (\exists x)(q(x))$
- $(\forall x)(p(x) \text{ sau } q(x)) \not\Leftarrow (\forall x)(p(x)) \text{ sau } (\forall x)(q(x))$
- $(\exists x)(p(x) \text{ și } q(x)) \not\Rightarrow (\exists x)(p(x)) \text{ și } (\exists x)(q(x))$

Exemplu

Enunțul $(\forall x \in \mathbb{N})(2 \mid x \text{ sau } 2 \nmid x)$ este adevărat.

Enunțul $(\forall x \in \mathbb{N})(2 \mid x)$ este fals. Enunțul $(\forall x \in \mathbb{N})(2 \nmid x)$ este tot fals. Prin urmare, enunțul $[(\forall x \in \mathbb{N})(2 \mid x) \text{ sau } (\forall x \in \mathbb{N})(2 \nmid x)]$ este fals.

Exemplu

Enunțul $(\exists x \in \mathbb{R})(x < 0 \text{ și } x \geq 10)$ este fals.

Enunțul $(\exists x \in \mathbb{R})(x < 0)$ este adevărat. Enunțul $(\exists x \in \mathbb{R})(x \geq 10)$ este tot adevărat. Prin urmare, enunțul $[(\exists x \in \mathbb{R})(x < 0) \text{ și } (\exists x \in \mathbb{R})(x \geq 10)]$ este adevărat.

Scoaterea de sub un cuantificator a unui enunț care nu depinde de variabila cuantificată

Dacă, în enunțurile compuse cuantificate de mai sus, în locul lui $q(x)$ avem un enunț q care nu depinde de x , atunci:

$$(\forall x) q \Leftrightarrow q \Leftrightarrow (\exists x) q,$$

așadar, în acest caz, din proprietățile anterioare obținem:

- $(\forall x)(p(x) \text{ și } q) \Leftrightarrow (\forall x)(p(x)) \text{ și } q$
- $(\exists x)(p(x) \text{ sau } q) \Leftrightarrow (\exists x)(p(x)) \text{ sau } q$

dar au loc și:

- ① $(\forall x)(p(x) \text{ sau } q) \Leftrightarrow (\forall x)(p(x)) \text{ sau } q$
- ② $(\exists x)(p(x) \text{ și } q) \Leftrightarrow (\exists x)(p(x)) \text{ și } q$

după cum se poate verifica foarte ușor analizând cazurile: $\begin{cases} q \text{ e falsă} \\ \text{și} \\ q \text{ e adevărată.} \end{cases}$

Putem observa și că oricare dintre proprietățile ① și ② se deduce din celalătă folosind modalitatea în care se neagă enunțurile cuantificate și proprietățile:

- $\text{non}(p \text{ și } q) \Leftrightarrow (\text{non } p) \text{ sau } (\text{non } q)$
și
- $\text{non}(p \text{ sau } q) \Leftrightarrow (\text{non } p) \text{ și } (\text{non } q)$

pentru orice enunțuri p și q .

Mnemonic despre proprietățile cuantificatorilor

Scrieri echivalente ale enunțurilor din exemplele anterioare, fără domeniu al valorilor după cuantificatori:

$$(\forall x \in \mathbb{N}) (2 \mid x \text{ sau } 2 \nmid x) \Leftrightarrow$$

$$(\forall x) [x \in \mathbb{N} \Rightarrow (2 \mid x \text{ sau } 2 \nmid x)] \Leftrightarrow$$

$$(\forall x) [x \in \mathbb{N} \Rightarrow (2 \mid x \text{ sau } 2 \nmid x)] \Leftrightarrow$$

$$(\forall x) [(x \in \mathbb{N} \Rightarrow 2 \mid x) \text{ sau } (x \in \mathbb{N} \Rightarrow 2 \nmid x)];$$

$$[(\forall x \in \mathbb{N}) (2 \mid x) \text{ sau } (\forall x \in \mathbb{N}) (2 \nmid x)] \Leftrightarrow$$

$$[(\forall x) (x \in \mathbb{N} \Rightarrow 2 \mid x) \text{ sau } (\forall x) (x \in \mathbb{N} \Rightarrow 2 \nmid x)];$$

$$(\exists x \in \mathbb{R}) (x < 0 \text{ și } x \geq 10) \Leftrightarrow$$

$$(\exists x) (x \in \mathbb{R} \text{ și } x < 0 \text{ și } x \geq 10) \Leftrightarrow$$

$$(\exists x) [(x \in \mathbb{R} \text{ și } x < 0) \text{ și } (x \in \mathbb{R} \text{ și } x \geq 10)];$$

$$[(\exists x \in \mathbb{R}) (x < 0) \text{ și } (\exists x \in \mathbb{R}) (x \geq 10)] \Leftrightarrow$$

$$[(\exists x) (x \in \mathbb{R} \text{ și } x < 0) \text{ și } (\exists x) (x \in \mathbb{R} \text{ și } x \geq 10)].$$

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Definiția unei funcții

Definiție

Fie A și B mulțimi oarecare. Se numește *funcție* de la A la B un triplet $f := (A, G, B)$, unde $G \subseteq A \times B$, a. i., pentru orice $a \in A$, există un unic $b \in B$, cu proprietatea că $(a, b) \in G$.

Formal: $(\forall a \in A) (\exists ! b \in B) ((a, b) \in G)$. Scris desfășurat:

$$(\forall a) [a \in A \Rightarrow [(\exists b) (b \in B \text{ și } (a, b) \in G) \text{ și } (\forall c) (\forall d) [(c \in B \text{ și } d \in B \text{ și } (a, c) \in G \text{ și } (a, d) \in G) \Rightarrow c = d]]].$$

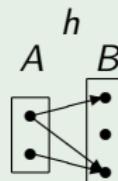
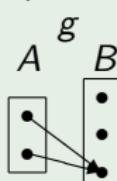
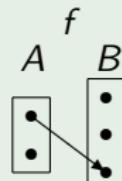
Faptul că f este o funcție de la A la B se notează cu $f : A \rightarrow B$ sau $A \xrightarrow{f} B$.

Mulțimea A se numește *domeniu* funcției f , B se numește *codomenu* sau *domeniu valorilor* lui f , iar G se numește *graficul* lui f .

Pentru fiecare $a \in A$, unicul $b \in B$ cu proprietatea că $(a, b) \in G$ se notează cu $f(a)$ și se numește *valoarea funcției f în punctul a* .

Exemplu

Care dintre următoarele corespondențe este o funcție de la A la B ?



Egalitatea a două funcții

Remarcă $((a, b) \in G \Leftrightarrow f(a) = b)$

Dacă $f = (A, G, B)$ este o funcție ($f : A \rightarrow B$), atunci graficul G al lui f este mulțimea de perechi: $G = \{(a, f(a)) \mid a \in A\} \subseteq A \times B$.

Definiție

Fie $f = (A, F, B)$ și $g = (C, G, D)$ două funcții ($f : A \rightarrow B$, iar $g : C \rightarrow D$).

- Egalitatea $f = g$ semnifică egalitatea de triplete $(A, F, B) = (C, G, D)$, i. e. spunem că $f = g$ dacă:

$A = C$ (are loc egalitatea domeniilor),
 $B = D$ (are loc egalitatea codomeniilor) și
 $F = G$ (are loc egalitatea graficelor celor două funcții,
ceea ce, conform scrierii acestor grafice
din remarcă anterioară, se transcrie în egalitate
punctuală, adică egalitate în fiecare punct:
pentru orice $a \in A = C$, $f(a) = g(a)$).

- Dacă X este o mulțime a. î. $X \subseteq A$ și $X \subseteq C$, atunci spunem că f și g coincid pe X dacă f și g au aceleasi valori în elementele lui X , adică:
oricare ar fi $x \in X$, $f(x) = g(x) \in B \cap D$.

Există o unică funcție de la \emptyset la o mulțime arbitrară

Notație (putere de mulțimi)

Pentru orice mulțimi A și B , se notează cu B^A mulțimea funcțiilor de la A la B :

$$B^A = \{f \mid f : A \rightarrow B\}$$

Remarcă (pentru orice B , B^\emptyset are exact un element)

Fie B o mulțime oarecare (**poate fi vidă și poate fi nevidă**). Atunci există o unică funcție $f : \emptyset \rightarrow B$.

Într-adevăr, o funcție $f : \emptyset \rightarrow B$ trebuie să fie un triplet $f = (\emptyset, G, B)$, cu $G \subseteq \emptyset \times B = \emptyset$, deci $G = \emptyset$. Așadar, există cel mult o funcție $f : \emptyset \rightarrow B$, anume $f = (\emptyset, \emptyset, B)$ este unica posibilitate. Să arătăm că acest triplet satisface definiția funcției:

$$(\forall a \in \emptyset) (\exists! b \in B) ((a, b) \in \emptyset), \text{ i. e.:}$$

$$(\forall a) [a \in \emptyset \Rightarrow (\exists! b) (b \in B \text{ și } (a, b) \in \emptyset)].$$

Pentru orice element a , proprietatea $a \in \emptyset$ este falsă, așadar, pentru orice element a , implicația $[a \in \emptyset \Rightarrow \dots]$ este adevărată. Iar acest lucru înseamnă exact faptul că întreaga proprietate $(\forall a)[a \in \emptyset \Rightarrow \dots]$ este adevărată, deci f este funcție. Prin urmare, există o unică funcție $f : \emptyset \rightarrow B$, anume $f = (\emptyset, \emptyset, B)$.

Nu există nicio funcție de la o mulțime nevidă la \emptyset

Remarcă (pentru orice $A \neq \emptyset$, $\emptyset^A = \emptyset$)

Fie A o mulțime **nevidă** (i. e. $A \neq \emptyset$). Atunci nu există nicio funcție $f : A \rightarrow \emptyset$. Într-adevăr, o funcție $f : A \rightarrow \emptyset$ trebuie să fie un triplet $f = (A, G, \emptyset)$, cu $G \subseteq A \times \emptyset = \emptyset$, deci $G = \emptyset$. Așadar, dacă ar exista o funcție $f : A \rightarrow \emptyset$, atunci am avea neapărat $f = (A, \emptyset, \emptyset)$. Să vedem dacă acest triplet verifică definiția funcției:

$$(\forall a \in A) (\exists ! b \in \emptyset) ((a, b) \in \emptyset), \text{ i. e.:}$$

$$(\forall a) [a \in A \Rightarrow [(\exists b) (b \in \emptyset \text{ și } (a, b) \in \emptyset)] \text{ și}$$

$$(\forall c) (\forall d) [(c \in \emptyset \text{ și } d \in \emptyset \text{ și } (a, c) \in \emptyset \text{ și } (a, d) \in \emptyset) \Rightarrow c = d]].$$

Oricare ar fi elementul b , proprietatea $b \in \emptyset$ este falsă, deci, oricare ar fi elementele a și b , conjuncția $(b \in \emptyset \text{ și } (a, b) \in \emptyset)$ este falsă, deci, oricare ar fi elementul a , proprietatea $(\exists b) (b \in \emptyset \text{ și } (a, b) \in \emptyset)$ este falsă, așadar, oricare ar fi elementul a , conjuncția care succede mai sus implicației având ca antecedent pe $a \in A$ este falsă. În schimb, întrucât A este nevidă, rezultă că proprietatea $a \in A$ este adevărată pentru măcar un element a . Prin urmare, implicația $[a \in A \Rightarrow \dots]$ de mai sus este falsă pentru cel puțin un element a , ceea ce înseamnă că întreaga proprietate $(\forall a) [a \in A \Rightarrow \dots]$ este falsă, și deci f nu este funcție.

Imaginea și preimaginea printr-o funcție

Definiție

Pentru orice mulțimi A și B , orice funcție $f : A \rightarrow B$ și orice submulțimi $X \subseteq A$ și $Y \subseteq B$, se definesc:

- *imaginea lui X prin f* sau *imaginea directă a lui X prin f* , notată $f(X)$, este submulțimea lui B :

$$f(X) = \{f(x) \mid x \in X\} \subseteq B$$

- $f(A)$ se mai notează cu $Im(f)$ și se numește *imaginea lui f* :

$$Im(f) = f(A) = \{f(a) \mid a \in A\} \subseteq B$$

- *preimaginea lui Y prin f* sau *imaginea inversă a lui Y prin f* , notată $f^{-1}(Y)$ ($f^*(Y)$) în unele cărți, pentru a o deosebi de imaginea lui Y prin inversa f^{-1} a lui f , care există numai atunci când f este inversabilă, deci numai atunci când f este bijectivă – a se vedea în cele ce urmează –, pe când preimaginea unei submulțimi a codomeniului poate fi definită pentru orice funcție), este submulțimea lui A :

$$f^{-1}(Y) = \{x \in A \mid f(x) \in Y\} \subseteq A$$

Functii injective, surjective, bijective

Definiție

Fie A și B mulțimi și $f : A \rightarrow B$ o funcție. f se zice:

- *injectivă* dacă are loc oricare dintre următoarele condiții echivalente:
 - pentru orice $b \in B$, există cel mult un $a \in A$, astfel încât $f(a) = b$
 - pentru orice $a_1, a_2 \in A$, dacă $a_1 \neq a_2$, atunci $f(a_1) \neq f(a_2)$
 - pentru orice $a_1, a_2 \in A$, dacă $f(a_1) = f(a_2)$, atunci $a_1 = a_2$
- *surjectivă* dacă are loc oricare dintre următoarele condiții echivalente:
 - pentru orice $b \in B$, există (cel puțin un) $a \in A$, astfel încât $f(a) = b$ (formal: $(\forall b \in B)(\exists a \in A)(f(a) = b)$)
 - $f(A) = B$
- *bijectivă* dacă are loc oricare dintre următoarele condiții echivalente:
 - f este simultan injectivă și surjectivă
 - pentru orice $b \in B$, există exact un $a \in A$, astfel încât $f(a) = b$ (formal: $(\forall b \in B)(\exists ! a \in A)(f(a) = b)$)

Functiile injective, surjective, respectiv bijective se mai numesc *injecții*, *surjecții*, respectiv *bijecții*.

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Familii arbitrar de elemente, familii arbitrar de multimi

- Ce este un sir de numere reale indexat de \mathbb{N} ? Un sir $(x_n)_{n \in \mathbb{N}} \subset \mathbb{R}$ este o functie $f : \mathbb{N} \rightarrow \mathbb{R}$. Pentru orice $n \in \mathbb{N}$, se noteaza $x_n := f(n) \in \mathbb{R}$.
- Ce este o familie arbitrară de numere reale? Fie I o multime arbitrară. Ce este o familie de numere reale indexata de I ? O familie $(x_i)_{i \in I} \subseteq \mathbb{R}$ este o functie $f : I \rightarrow \mathbb{R}$. Pentru orice $i \in I$, se noteaza $x_i := f(i) \in \mathbb{R}$. Elementele multimii I se numesc *indicii* familiei $(x_i)_{i \in I}$.

Dată o multime arbitrară A :

- ce este un sir de elemente ale lui A indexat de \mathbb{N} ?
- ce este o familie arbitrară de elemente ale lui A ?

Înlocuind mai sus pe \mathbb{R} cu A , se obțin definițiile acestor noțiuni. Să reținem:

Definiție (familie de elemente ale lui A indexată de I): $(x_i)_{i \in I} \subseteq A$

Fie I și A multimi arbitrarare.

O familie de elemente ale lui A indexata de I este o functie $f : I \rightarrow A$. Pentru orice $i \in I$, se noteaza $x_i := f(i) \in A$, astfel că familia f se mai notează sub forma $(x_i)_{i \in I} \subseteq A$.

Elementele multimii I se numesc *indicii* familiei $(x_i)_{i \in I}$.

- Ce este un sir de multimi indexat de \mathbb{N} ?
- Ce este o familie arbitrară de multimi?

Egalitatea între familii de elemente

- Există o singură familie vidă de elemente ale unei mulțimi arbitrară A , pentru că există o singură funcție de la \emptyset la A (a se vedea și mai jos).
- Familia vidă nu este egală cu nicio familie nevidă, ci este egală doar cu ea însăși.

Fie I și A două mulțimi nevide, iar $(a_i)_{i \in I}$ și $(b_i)_{i \in I}$ două familii de elemente din A indexate de I :

- $I \neq \emptyset, A \neq \emptyset$
- $(a_i)_{i \in I} \subseteq A$, i. e., pentru orice $i \in I, a_i \in A$
- $(b_i)_{i \in I} \subseteq A$, i. e., pentru orice $i \in I, b_i \in A$

Conform celor menționate anterior, familiile de elemente din A indexate de I sunt, prin definiție, funcții de la I la A , iar notația lor ca mai sus se adoptă pentru comoditate:

- $(a_i)_{i \in I} = f : I \rightarrow A$, unde, pentru orice $i \in I, f(i) = a_i$
- $(b_i)_{i \in I} = g : I \rightarrow A$, unde, pentru orice $i \in I, g(i) = b_i$

Egalitatea între familiile de elemente

Egalitatea de funcții semnifică egalitatea domeniilor și a codomeniilor (valabilă pentru f și g , pentru că au ambele domeniu I și codomeniu A) și *egalitatea punctuală*, adică egalitatea în fiecare punct: două funcții cu același domeniu și același codomeniu sunt egale dacă sunt egale în fiecare punct al domeniului lor comun:

$$f = g \text{ ddacă, pentru orice } i \in I, f(i) = g(i).$$

Deci ce semnifică egalitatea a două familiile de elemente din aceeași mulțime indexate de aceeași mulțime? *Egalitatea pe componente*: cele două familiile sunt egale dacă sunt egale pe componente:

$$(a_i)_{i \in I} = (b_i)_{i \in I} \text{ ddacă, pentru orice } i \in I, a_i = b_i.$$

- **Caz particular:** cazul finit nevid: $I = \overline{1, n}$, cu n natural nenul:

$$(a_1, a_2, \dots, a_n) = (b_1, b_2, \dots, b_n) \text{ ddacă } \begin{cases} a_1 = b_1, \\ a_2 = b_2, \\ \vdots \\ a_n = b_n. \end{cases}$$

Familii arbitrar de mulțimi

Definiție

Fie T o mulțime arbitrară. Se numește *șir de submulțimi ale lui T indexat de \mathbb{N}* o funcție $f : \mathbb{N} \rightarrow \mathcal{P}(T)$. Pentru fiecare $n \in \mathbb{N}$, se notează $A_n := f(n) \in \mathcal{P}(T)$, iar *șirul de submulțimi ale lui T* se notează cu $(A_n)_{n \in \mathbb{N}}$. Scriem $(A_n)_{n \in \mathbb{N}} \subseteq \mathcal{P}(T)$ cu semnificația că, pentru fiecare $n \in \mathbb{N}$, $A_n \in \mathcal{P}(T)$.

Definiție

Fie T și I două mulțimi arbitrară. Se numește *familie de submulțimi ale lui T indexată de I* o funcție $f : I \rightarrow \mathcal{P}(T)$. Pentru fiecare $i \in I$, se notează $A_i := f(i) \in \mathcal{P}(T)$, iar *familia de submulțimi ale lui T* se notează cu $(A_i)_{i \in I}$. Scriem $(A_i)_{i \in I} \subseteq \mathcal{P}(T)$ cu semnificația că, pentru fiecare $i \in I$, $A_i \in \mathcal{P}(T)$. Elementele mulțimii I se numesc *indicii* familiei $(A_i)_{i \in I}$.

Putem generaliza definițiile anterioare la șiruri de mulțimi oarecare și familii de mulțimi oarecare, nu neapărat părți ale unei mulțimi precizate, dar vom avea nevoie de acea definiție mai cuprinzătoare a noțiunii de funcție, care permite unei funcții f definite pe \mathbb{N} , respectiv pe I , să aibă drept codomeniu o clasă (nu neapărat o mulțime), anume clasa tuturor mulțimilor în acest caz.

Operații cu familii arbitrar de mulțimi

Definiție

Fie I o mulțime arbitrară și $(A_i)_{i \in I}$ o familie de mulțimi (părți ale unei mulțimi T sau mulțimi arbitrate) indexată de I .

Se definesc următoarele operații:

- *reuniunea familiei* $(A_i)_{i \in I}$ este mulțimea notată $\bigcup_{i \in I} A_i$ și definită prin:

$$\bigcup_{i \in I} A_i = \{x \mid (\exists i \in I) (x \in A_i)\} = \{x \mid (\exists i) (i \in I \text{ și } x \in A_i)\}$$

- *intersecția familiei* $(A_i)_{i \in I}$ este mulțimea notată $\bigcap_{i \in I} A_i$ și definită prin:

$$\bigcap_{i \in I} A_i = \{x \mid (\forall i \in I) (x \in A_i)\} = \{x \mid (\forall i) (i \in I \Rightarrow x \in A_i)\}$$

Operații cu familii arbitrar de mulțimi

Definiție (continuare)

- *produsul cartezian al familiei* $(A_i)_{i \in I}$ (numit și *produsul direct al familiei* $(A_i)_{i \in I}$) este mulțimea notată $\prod_{i \in I} A_i$ și definită prin:

$$\begin{aligned}\prod_{i \in I} A_i &= \{(a_i)_{i \in I} \subseteq \bigcup_{i \in I} A_i \mid (\forall i \in I) (a_i \in A_i)\} = \\ &= \{(a_i)_{i \in I} \subseteq \bigcup_{i \in I} A_i \mid (\forall i) (i \in I \Rightarrow a_i \in A_i)\},\end{aligned}$$

sau, altfel scris (cu definiția unei familii de elemente exemplificate mai sus pe familii de numere reale):

$$\begin{aligned}\prod_{i \in I} A_i &= \{f \mid f : I \rightarrow \bigcup_{i \in I} A_i, (\forall i \in I) (f(i) \in A_i)\} = \\ &= \{f \mid f : I \rightarrow \bigcup_{i \in I} A_i, (\forall i) (i \in I \Rightarrow f(i) \in A_i)\}.\end{aligned}$$

Notație

Fie $n \in \mathbb{N}^*$ și mulțimile A_1, A_2, \dots, A_n . Produsul direct $\prod_{i \in \overline{1,n}} A_i$ se mai notează cu

$\prod_{i=1}^n A_i$, iar un element $(a_i)_{i \in \overline{1,n}}$ al acestui produs direct se mai notează cu

(a_1, a_2, \dots, a_n) . În cazul particular în care $A_1 = A_2 = \dots = A_n = A$,

$\prod_{i \in \overline{1,n}} A$ notație $\prod_{i=1}^n A$ notație A^n .

Remarcă (puterile unei mulțimi: caz particular al produsului direct)

În definiția anterioară, dacă $I \neq \emptyset$ și, pentru orice $i \in I$, $A_i = A$, atunci

$\bigcup_{i \in I} A_i = \bigcup_{i \in I} A = A$, așadar $\prod_{i \in I} A = \{f \mid f : I \rightarrow A, (\forall i \in I)(f(i) \in A)\} =$

$\{f \mid f : I \rightarrow A\} = A^I$. În particular, pentru orice $n \in \mathbb{N}^*$, dacă $|I| = n$, avem:

$A^n = \{(a_1, \dots, a_n) \mid (\forall i \in \overline{1,n})(a_i \in A)\} = \{f \mid f : \overline{1,n} \rightarrow A\} = A^{\overline{1,n}} = A^I$.

Pentru orice mulțimi A, B , notăm cu $A \cong B$ faptul că există o bijecție între A și B .

Remarcă

Pentru orice mulțimi A, I și J , dacă $I \cong J$, atunci $A^I \cong A^J$.

Operații de diferite arități

Aritatea unei operații a unei structuri algebrice (cu o singură *mulțime suport*, o singură mulțime de elemente) este **numărul argumentelor (operanzilor, variabilelor) acelei operații**. Dacă structura algebrică are mulțimea suport A , iar f este o operație n -ară pe A , cu $n \in \mathbb{N}$, atunci $f : \underbrace{A \times A \times \dots \times A}_{n \text{ de } A} \rightarrow A$ (f este o funcție cu n argumente din A , cu valori tot în A).

Exemplu

Într-un grup $(G, \circ, -1, e)$, avem trei operații:

- operația *binară* \circ (**compunerea elementelor grupului, două câte două**) (operație de aritate 2, operație cu două argumente): $\circ : G \times G \rightarrow G$
- operația *unară* -1 (**inversarea fiecărui element din grup**) (operație de aritate 1, operație cu un singur argument): $-1 : G \rightarrow G$
- operația *zeroară* (sau *nulară*) e (**elementul neutru al grupului**) (operație de aritate 0, operație fără argumente, i. e. **constantă** din G): $e \in G$

Operațiile zeroare sunt elemente distinse, i. e. constante

De ce operațiile zeroare sunt același lucru cu **elemente distinse, constante** din mulțimea suport a structurii algebrice?

Urmând regula de mai sus, elementul neutru e al grupului G trebuie să fie o funcție de la produsul direct al familiei vide la G .

Produsul direct al familiei vide nu este \emptyset , cum s–ar putea crede, ci este un *singleton*, adică o mulțime cu un singur element.

Într–adevăr, dacă recitим de mai sus definiția produsului direct al unei familii arbitrarе de mulțimi, observăm că produsul direct al familiei vide este mulțimea funcțiilor de la mulțimea vidă la reuniunea familiei vide, care satisfac o proprietate întotdeauna adevărată (anume $(\forall i)(i \in \emptyset \Rightarrow \dots)$), iar $i \in \emptyset$ este fals pentru orice i , deci implicația anterioară este adevărată pentru orice i , deci această proprietate cu variabila i cuantificată universal este adevărată), adică produsul direct al familiei vide este mulțimea funcțiilor de la mulțimea vidă la reuniunea familiei vide, care are un singur element, pentru că de la \emptyset la orice mulțime există o unică funcție, aşa cum am văzut mai sus.

Remarcă (reuniunea familiei vide este vidă)

Dacă recitим și definiția reuniunii unei familii arbitrarе, observăm că reuniunea familiei vide este mulțimea elementelor pentru care există un $i \in \emptyset$ cu o anumită proprietate, condiție care este întotdeauna falsă, deci reuniunea familiei vide este \emptyset .

Operațiile zeroare sunt elemente distinse, i.e. constante

Remarcă (produsul direct al familiei vide este un singleton)

Cele de mai sus arată că produsul direct al familiei vide este mulțimea cu unicul element dat de unica funcție de la \emptyset la \emptyset , anume $(\emptyset, \emptyset, \emptyset)$, adică produsul direct al familiei vide este singletonul $\{(\emptyset, \emptyset, \emptyset)\}$.

Prin urmare, elementul neutru al grupului G este o funcție φ de la singletonul $G^\emptyset = G^0 = \{(\emptyset, \emptyset, \emptyset)\}$ la G :

$$\varphi : G^\emptyset = G^0 = \{(\emptyset, \emptyset, \emptyset)\} \rightarrow G,$$

iar o funcție definită pe un singleton are o singură valoare ($\varphi((\emptyset, \emptyset, \emptyset)) \in G$), i.e. are imaginea tot un singleton:

$$\varphi(G^\emptyset) = \varphi(G^0) = \{\varphi((\emptyset, \emptyset, \emptyset))\} \subseteq G,$$

conținând acea unică valoare, deci poate fi identificată cu această unică valoare a ei, care este un element distins, o constantă din G :

$$\varphi((\emptyset, \emptyset, \emptyset)) = e \in G,$$

și identificăm:

$$\varphi = e.$$

Exemplu (structură algebrică având mai multe mulțimi suport, i. e. mai multe mulțimi (tipuri) de elemente)

Ca o paranteză, o **structură algebrică cu două mulțimi suport** este **spațiul vectorial**, care are ca mulțimi suport **mulțimea scalarilor** (formând un corp, K), și **mulțimea vectorilor** (formând un grup abelian, V). Nu este același lucru cu o structură algebrică având ca unică mulțime suport pe $K \times V$, pentru că nu avem operații pe $K \times V$ (i. e. operații de la $K \times V \times K \times V \times \dots \times K \times V$ la $K \times V$), ci avem operații de grup pe V , operații de corp pe K și operația de compunere (înmulțire) a scalarilor cu vectorii, de la $K \times V$ la V .

Observație (cu ce unifică o constantă, i.e. o operație zeroară (operație fără argumente, fără operanzi))

Dacă revedeți discuția informală de mai sus despre **unificare**, puteți observa că o constantă **nu unifică**:

- nici cu o altă constantă,
- nici cu o expresie având ca operator dominant o funcție (i. e. operație) cu argumente (i.e. operanzi), i.e. de aritate nenulă.

O constantă **unifică** numai cu **ea însăși** sau cu o **variabilă**.

PROGRAMARE LOGICĂ

Cursurile II, III, IV, V

RECAPITULARE LOGICĂ PROPOZIȚIONALĂ

Claudia MUREŞAN
cmuresan@fmi.unibuc.ro, c.muresan@yahoo.com

Universitatea din Bucureşti
Facultatea de Matematică și Informatică
Bucureşti

2019–2020, Semestrul II

Cuprinsul acestui curs

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Logică matematică clasică. Calculul propozițional

- **Logica matematică** este o ramură a matematicii care se ocupă cu exprimarea formalizată (i. e. formală, simbolică) a legilor gândirii și studierea acestora cu mijloace matematice.
- Ne propunem să studiem **logica clasică**, în două forme ale ei: **logica clasică a propozițiilor și logica clasică a predicatelor** sau a **propozițiilor cu variabile**. Vom face deosebirea dintre aceste două tipuri de logică clasică mai târziu. Ambele sunt **logici bivalente**, adică operează cu doar două **valori de adevăr**: **fals** și **adevărat**.
- Așadar, în **logica clasică**, toate enunțurile (propozițiile, afirmațiile) sunt presupuse a fi **adevărate** sau **false**. Aceasta nu este o condiție trivială, nici măcar dacă eliminăm din discuție enunțurile interrogative și pe cele exclamative, după cum ne amintim din primul curs de logică matematică, din exemplul cu enunțul subiectiv, precum și cel cu **paradoxul mincinosului**: să se determine dacă următoarea afirmație este **adevărată** sau **falsă**: *Această afirmație este falsă*.

Logică matematică clasică. Calculul propozițional

- În acest curs vom recapitula **logica propozițională clasică** din cursurile de LOGICĂ MATEMATICĂ și COMPUTAȚIONALĂ de anul trecut.
- Vom studia **sistemul formal al calculului propozițional clasic** sub trei aspecte fundamentale:
 - ① **sintaxa**, care se ocupă de limbajul formal al calculului propozițional clasic, i.e. de cadrul formal, de exprimarea în simboluri a obiectelor matematice cu care vom lucra;
 - ② **algebra**, care asociază o structură algebrică sistemului formal descris în partea de sintaxă și folosește această asociere pentru a transfera proprietățile algebrice ale acelei structuri în proprietăți logice, și invers;
 - ③ **semantica**, aceasta fiind partea în care, pe baza structurii algebrice atașate logicii, se calculează efectiv **valorile de adevăr** ale enunțurilor (**fals** sau **adevărat**).
- Există și alte aspecte sub care poate fi studiat un sistem logic, cum ar fi: aspectul **topologic**, cel **probabilist** etc., dar studierea lor depășește cadrul și scopul acestui curs.
- Toate aceste aspecte sub care poate fi studiat un sistem logic sunt denumite *dimensiuni ale sistemului logic*.

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Definiții și notații

Următoarele **simboluri** formează **alfabetul** sistemului formal al calculului propozițional clasic:

- ① *variabilele propoziționale*, notate, de obicei, u , v , w etc., uneori cu indici, care formează o mulțime infinită, și de obicei considerată numărabilă; vom nota cu V mulțimea variabilelor propoziționale;
- ② *conectorii logici primitivi*:
 - \neg : *negația* (se citește: "non" sau "not");
 - \rightarrow : *implicația* (se citește: "implică");
- ③ parantezele: $($, $)$, $[$, și $]$.

Simbolurile enumerate mai sus se numesc *simboluri primitive* și sunt presupuse a fi două câte două distințe (de exemplu $\neg \notin V$ etc.).

La acestea se adaugă *conectorii logici derivați*, care se definesc pe baza conectorilor logici primitivi, și care vor fi prezentați mai jos.

Să notăm cu A alfabetul sistemului formal al calculului propozițional clasic, adică mulțimea simbolurilor primitive: $A = V \cup \{\neg, \rightarrow, (,), [,]\}$.

Cuvintele peste alfabetul simbolurilor primitive

Definiție

Șirurile (alăturările) finite și nevide de simboluri primitive se numesc *cuvinte*.

Notație

Așadar *mulțimea cuvintelor* calculului propozițional clasic este mulțimea A^+ a cuvintelor finite și nevide peste alfabetul A :

$$A^+ = \{a_1 a_2 \dots a_n \mid n \in \mathbb{N}^*, a_1, a_2, \dots, a_n \in A\}.$$

Exemplu

$u \rightarrow \neg v, \neg(u \rightarrow \neg v) \rightarrow w, \rightarrow u \rightarrow \neg uv \neg$ sunt **cuvinte**.

Observație

Intuiția ne determină să conferim “înțeles” simbolurilor primitive, și ne spune că primele două cuvinte din exemplul anterior “au sens”, în timp ce al treilea “nu are sens”. Dintre cuvintele peste alfabetul definit mai sus, le vom selecta pe cele care “au sens”, și le vom numi *enunțuri*. Urmează definiția lor riguroasă:

Cuvintele care “au sens”: enunțurile

Definiție

Un *enunț* este un cuvânt φ care satisfacă una dintre condițiile următoare:

- (E_1) φ este o variabilă propozițională;
- (E_2) există un enunț ψ a. î. $\varphi = \neg\psi$;
- (E_3) există două enunțuri ψ și χ a. î. $\varphi = \psi \rightarrow \chi$.

Definiție

Variabilele propoziționale se numesc *enunțuri atomice* sau *enunțuri elementare*. Enunțurile care nu sunt variabile propoziționale, adică se află în cazul (E_2) sau (E_3) din definiția anterioară, se numesc *enunțuri compuse*.

Notație

Vom nota cu E mulțimea tuturor enunțurilor.

Paranteză – intersecția familiei vide de mulțimi

Fie I o mulțime arbitrară și $(A_i)_{i \in I}$ o familie de mulțimi indexată de I – a se vedea finalul primului curs.

Să transcriem definiția intersecției unei familii arbitrate de mulțimi pentru familia vidă, i.e. pentru cazul $I = \emptyset$:

$$\bigcap_{i \in \emptyset} A_i = \{x \mid (\forall i \in \emptyset)(x \in A_i)\} = \{x \mid (\forall i)(i \in \emptyset \Rightarrow x \in A_i)\}.$$

Proprietatea $i \in \emptyset$ este falsă pentru orice element i , aşadar implicația $i \in \emptyset \Rightarrow x \in A_i$ este adevărată pentru orice i și orice x , deci proprietatea $(\forall i)(i \in \emptyset \Rightarrow x \in A_i)$ este adevărată pentru orice x . Sigur că nu există o mulțime care să conțină toate obiectele x . O astfel de mulțime ar conține, în particular, toate mulțimile, deci ar avea drept submulțime mulțimea tuturor mulțimilor, care nu există.

Intersecția familiei vide nu există decât raportat la o mulțime totală T : intersecția familiei vide de părți ale lui T se **definește** în următorul mod, și este egală cu mulțimea totală T :

$$\bigcap_{i \in \emptyset} A_i := \{x \in T \mid (\forall i \in \emptyset)(x \in A_i)\} = \{x \in T \mid (\forall i)(i \in \emptyset \Rightarrow x \in A_i)\} = T,$$

Întrucât proprietatea $(\forall i)(i \in \emptyset \Rightarrow x \in A_i)$ este adevărată pentru orice x .



Mnemonic despre operatori de închidere și familiile Moore

Notație (mulțimea părților unei mulțimi)

Pentru orice mulțime T , notăm cu $\mathcal{P}(T) = \{S \mid S \subseteq T\}$.

Fie T o mulțime.

Definiție

- Se numește *familie Moore de părți ale lui T* (sau *sistem de închidere pe mulțimea părților lui T*) o familie de părți ale lui T închisă la intersecții arbitrară, i. e. o familie de mulțimi $\mathcal{M} = (M_i)_{i \in I} \subseteq \mathcal{P}(T)$, cu I mulțime arbitrară, având proprietatea că, pentru orice $S \subseteq I$, $\bigcap_{s \in S} M_s \in \mathcal{M}$ (i. e., pentru orice $S \subseteq I$, există $i_S \in I$, astfel încât $\bigcap_{s \in S} M_s = M_{i_S}$). Familiile Moore se mai numesc *sisteme de închidere*.
- Se numește *operator de închidere pe $\mathcal{P}(T)$* o funcție $C : \mathcal{P}(T) \rightarrow \mathcal{P}(T)$, astfel încât, pentru orice $X, Y \in \mathcal{P}(T)$, au loc proprietățile:
 - $C(C(X)) = C(X)$ (C este *idempotent*);
 - $X \subseteq C(X)$ (C este *extensiv*);
 - dacă $X \subseteq Y$, atunci $C(X) \subseteq C(Y)$ (C este *crescător*).

Mnemonic despre operatori de închidere și familii Moore

Remarcă

Orice familie Moore de părți ale lui T conține intersecția familiei vide de părți ale lui T , adică pe T .

Exemplu

- $\text{id}_{\mathcal{P}(T)}$ este un operator de închidere pe $\mathcal{P}(T)$.
- Funcția constantă $C : \mathcal{P}(T) \rightarrow \mathcal{P}(T)$, $(\forall X \in \mathcal{P}(T))(C(X) = T)$, este un operator de închidere pe $\mathcal{P}(T)$.
- $\mathcal{P}(T)$ este o familie Moore de părți ale lui T .
- $\{T\}$ este o familie Moore de părți ale lui T .
- \emptyset nu este o familie Moore de părți ale lui T , pentru că nu îl conține pe T .

Așadar:

Remarcă

Orice familie Moore este nevidă.

Corespondență bijectivă între sistemele de închidere și operatorii de închidere pe o mulțime

Propoziție

Dacă \mathcal{M} este o familie Moore de părți ale lui T , atunci, pentru orice $A \in \mathcal{P}(T)$, există o (unică – să ne amintim că minimul, dacă există, este unic) cea mai mică mulțime din \mathcal{M} care include pe A (cea mai mică în sensul incluziunii), și aceasta este egală cu intersecția mulțimilor din \mathcal{M} care includ pe A .

Propoziție (*)

Fie I o mulțime nevidă și $\mathcal{M} = (M_i)_{i \in I}$ o familie Moore de părți ale lui T . Definim funcția $C_{\mathcal{M}} : \mathcal{P}(T) \rightarrow \mathcal{P}(T)$ astfel: pentru orice $X \in \mathcal{P}(T)$, $C_{\mathcal{M}}(X)$ este, prin definiție, cea mai mică mulțime din \mathcal{M} care include pe X , adică

$$C_{\mathcal{M}}(X) := \bigcap_{\substack{M \in \mathcal{M}, \\ X \subseteq M}} M.$$

Atunci $C_{\mathcal{M}}$ este un operator de închidere pe $\mathcal{P}(T)$.

Corespondență bijectivă între sistemele de închidere și operatorii de închidere pe o mulțime

Propoziție (**)

Fie $C : \mathcal{P}(T) \rightarrow \mathcal{P}(T)$ un operator de închidere pe $\mathcal{P}(T)$. Definim $\mathcal{M}_C = C(\mathcal{P}(T)) = \{C(X) \mid X \in \mathcal{P}(T)\} = \{X \in \mathcal{P}(T) \mid X = C(X)\} \subseteq \mathcal{P}(T)$ (familia mulțimilor închise din $\mathcal{P}(T)$ raportat la operatorul de închidere C). Atunci \mathcal{M}_C este o familie Moore de părți ale lui T .

Propoziție

Aplicațiile din cele Propozițiile (*) și (**) sunt inverse una alteia, adică:

- 1 pentru orice operator de închidere $C : \mathcal{P}(T) \rightarrow \mathcal{P}(T)$, $C_{\mathcal{M}_C} = C$;
- 2 pentru orice familie Moore \mathcal{M} de părți ale lui T , $\mathcal{M}_{C_{\mathcal{M}}} = \mathcal{M}$.

Așadar aceste aplicații sunt bijecții, deci mulțimea operatorilor de închidere pe $\mathcal{P}(T)$ și mulțimea familiilor Moore de părți ale lui T sunt în bijecție.

Remarcă

Conform definiției enunțurilor, **toate enunțurile** se obțin prin aplicarea regulilor (E_1) , (E_2) și (E_3) , aşadar E este cea mai mică mulțime de cuvinte peste A care include pe V și este închisă la \neg și \rightarrow , i. e. cea mai mică (în sensul incluziunii, adică în posetul $(\mathcal{P}(A^+), \subseteq)$) submulțime M a lui A^+ cu proprietățile:

- ① $V \subseteq M$,
- ② pentru orice $\varphi \in M$, rezultă că $\neg\varphi \in M$,
- ③ pentru orice $\varphi, \psi \in M$, rezultă că $\varphi \rightarrow \psi \in M$.

Remarcă (E e închiderea lui V în familia Moore a submulțimilor lui A^+ închise la \neg și \rightarrow)

Fie $\mathcal{M} = \{M \subseteq A^+ \mid (\forall \psi, \chi \in M) (\neg\psi, \psi \rightarrow \chi \in M)\}$.

\mathcal{M} este sistem de închidere pe $\mathcal{P}(A^+)$. Într-adevăr, $A^+ \in \mathcal{M}$, iar, dacă I este o mulțime nevidă și $(M_i)_{i \in I} \subseteq \mathcal{M}$, atunci, pentru orice $\psi, \chi \in \bigcap_{i \in I} M_i$, avem, pentru

fiecare $i \in I$, $\psi, \chi \in M_i$, aşadar $\neg\psi, \psi \rightarrow \chi \in M_i$, prin urmare

$\neg\psi, \psi \rightarrow \chi \in \bigcap_{i \in I} M_i$, aşadar $\bigcap_{i \in I} M_i \in \mathcal{M}$.

Dacă notăm cu $C_{\mathcal{M}} : \mathcal{P}(A^+) \rightarrow \mathcal{P}(A^+)$ operatorul de închidere asociat lui \mathcal{M} , atunci, conform remarcii anterioare, $E = C_{\mathcal{M}}(V)$.

Rolul parantezelor; parantezări corecte

Observație

În definiția de mai sus a enunțurilor, se subînțelege faptul că parantezele au rolul obișnuit: aici, parantezele pot încadra enunțuri, și se folosesc pentru a încadra enunțuri compuse în interiorul altor enunțuri compuse, indicând ordinea în care se aplică regulile (E_1) , (E_2) și (E_3) pentru obținerea unui enunț compus (impropriu spus, ordinea "aplicării conectorilor logici primitivi" pentru obținerea acelui enunț). O parantezare corectă a unui enunț este o dispunere a parantezelor în interiorul acelui enunț astfel încât fiecare pereche de paranteze să încadreze un (alt) enunț, și, desigur, astfel încât ordinea aplicării regulilor (E_1) , (E_2) și (E_3) pentru obținerea enunțului respectiv să fie indicată corect de acea parantezare.

Adică, pentru orice enunțuri ψ și $\psi \rightarrow \chi$:

- $\neg(\psi)$ este o parantezare corectă a enunțului $\neg\psi$;
- $\psi \rightarrow (\chi)$, $(\psi) \rightarrow \chi$ și $(\psi) \rightarrow (\chi)$ sunt parantezări corecte ale enunțului $\psi \rightarrow \chi$.

Prioritățile conectorilor logici

Observație

Observăm că, în scrierea enunțurilor, dată de definiția de mai sus, conectorii logici primitivi apar scriși la fel ca niște operatori:

- \neg apare scris la fel ca un operator unar;
- \rightarrow apare scris la fel ca un operator binar.

Pentru a evita încărcarea scrierii cu prea multe paranteze, se face următoarea **convenție**: se acordă prioritate mai mare conectorului logic “unar” \neg și prioritate mai mică celui “binar”, \rightarrow .

Noțiunea de **prioritate** are aici semnificația obișnuită, de determinare a ordinii “aplicării conectorilor logici”, corect spus de determinare a ordinii aplicării regulilor (E_1) , (E_2) și (E_3) pentru construirea unui enunț.

Conectorul cu prioritate mai mare “se va aplica” primul, de fapt regula (E_2) se va aplica înaintea regulei (E_3) , i. e., pentru orice enunțuri α și β , scrierea $\neg\alpha \rightarrow \beta$ va semnifica $(\neg\alpha) \rightarrow \beta$.

Egalitatea între enunțuri

Observație

Egalitatea între enunțuri care apare în scrierea regulilor (E_2) și (E_3) este egalitatea obișnuită între cuvinte peste un alfabet, între siruri de simboluri, anume **literal identitatea**, adică egalitatea simbol cu simbol, i. e. egalitatea lungimilor și identitatea (coincidența) simbolurilor de pe aceeași poziție în fiecare cuvânt (ca la siruri de caractere: identitatea literă cu literă, caracter cu caracter), desigur, modulo parantezarea aleasă.

Adică: două enunțuri **scrise numai cu simboluri primitive** (vom vedea ce sunt simbolurile derivate) sunt *egale* dacă există câte o parantezare corectă pentru fiecare astfel încât, cu acele parantezări, enunțurile respective să fie literal identice (ca și cuvinte peste alfabetul prezentat mai sus, format din simbolurile primitive).

Remarcă

Dacă recitим observația anterioară, vom remarca faptul că orice enunț se află în **exact una** (i. e. **una și numai una**) dintre cele 3 situații prezentate de regulile (E_1), (E_2) și (E_3).

Remarcă

Noțiunea de enunț este definită recursiv: se pornește de la variabilele propoziționale și se aplică recursia dată de regulile (E_2) și (E_3) .

Din faptul că enunțurile sunt siruri **finite** de simboluri primitive și observația că, prin aplicarea oricăreia dintre regulile (E_1) , (E_2) și (E_3) , lungimea enunțului format până la momentul curent crește cu cel puțin câte o unitate, deducem faptul că orice enunț se obține prin aplicarea regulilor (E_1) , (E_2) și (E_3) de un număr finit de ori, i. e. printr-un număr finit de aplicări ale regulilor (E_1) , (E_2) și (E_3) , i. e., pornind de la variabilele propoziționale (evident, de la un număr finit de variabile propoziționale), într-un număr finit de pași, fiecare pas constând în aplicarea unei reguli de recursie: (E_2) sau (E_3) .

Pentru a putea defini riguros egalitatea de enunțuri, să definim:

Definiție (arborii binari asociați enunțurilor)

Orice enunț are un *arbore binar asociat*, definit **recursiv** astfel:

- pentru orice variabilă propozițională p , arborele binar asociat lui p este arborele cu un singur nod, etichetat cu p ;
- pentru orice enunț ψ , arborele binar asociat enunțului $\neg\psi$ are rădăcina etichetată cu, conectorul logic \neg , și arborele binar asociat lui ψ ca unic subarbore;
- pentru orice enunțuri ψ și χ , arborele binar asociat enunțului $\psi \rightarrow \chi$ are rădăcina etichetată cu, conectorul logic \rightarrow , arborele binar asociat lui ψ ca subarbore stâng și arborele binar asociat lui χ ca subarbore drept.

Remarcă (unicitatea arborelui binar asociat unui enunț)

Cum un enunț se află în unul și numai unul dintre cazurile (E_1) , (E_2) și (E_3) , se demonstrează inductiv că un enunț are un unic arbore binar asociat: dacă S e mulțimea enunțurilor care au câte un unic arbore binar asociat, atunci, conform definiției anterioare:

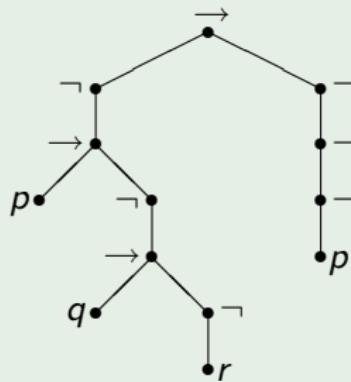
- ① $V \subseteq S$,
- ② pentru orice $\psi \in S$, rezultă $\neg\psi \in S$,
- ③ pentru orice $\psi, \chi \in S$, rezultă că $\psi \rightarrow \chi \in S$,

Remarcă (continuare)

prin urmare $S = E$, conform definiției mulțimii E a tuturor enunțurilor.

Exemplu

Arborele binar asociat enunțului $\neg(p \rightarrow \neg(q \rightarrow \neg r)) \rightarrow \neg\neg\neg p$, unde $p, q, r \in V$, este:



Definiție

Considerăm două enunțuri ca fiind *egale* (modulo parantezări corecte) dacă au același arbore binar asociat, în care ordinea fiilor unui nod (deci diferența dintre subarborele stâng și subarborele drept) contează (adică inversând doi subarbore distincți ai unui nod obținem un arbore diferit).

Riguros:

Definiție (enunțurile, arborii binari asociați lor și parantezările corecte)

Mulțimea E a *enunțurilor* calculului propozițional clasic este cea mai mică submulțime $E \subseteq A^+$ a mulțimii cuvintelor (finite și nevide) peste alfabetul A al simbolurilor primitive având proprietățile:

① $V \subseteq E$; oricărui $p \in V$ îi asociem arborele cu un singur nod, etichetat cu p ;

② dacă $\psi \in E$, atunci $\neg(\psi) \in E$;

în plus, dacă $\psi \in V$ sau există $\alpha \in E$ astfel încât $\psi \in \{\neg\alpha, \neg(\alpha)\}$, atunci avem și $\neg\psi \in E$;

enunțurilor $\neg\psi$ și $\neg(\psi)$ le asociem arborele binar având rădăcina etichetată cu \neg și, ca unic subarbore al rădăcinii, arborele binar asociat lui ψ ;

③ dacă $\psi, \chi \in E$, atunci $(\psi) \rightarrow (\chi) \in E$;

în plus, dacă $\psi \in V$ sau există $\alpha \in E$ astfel încât $\psi \in \{\neg\alpha, \neg(\alpha)\}$, atunci avem și $\psi \rightarrow (\chi) \in E$;

similar, dacă $\chi \in V$ sau există $\beta \in E$ astfel încât $\chi \in \{\neg\beta, \neg(\beta)\}$, atunci avem și $(\psi) \rightarrow \chi \in E$;

iar, dacă există $\alpha, \beta \in E$ astfel încât $\psi \in V \cup \{\neg\alpha, \neg(\alpha)\}$ și

$\chi \in V \cup \{\neg\beta, \neg(\beta)\}$, atunci avem și $\psi \rightarrow \chi \in E$;

enunțurilor $\psi \rightarrow \chi$, $\psi \rightarrow (\chi)$, $(\psi) \rightarrow \chi$ și $(\psi) \rightarrow (\chi)$ le asociem arborele binar având rădăcina etichetată cu \rightarrow , arborele binar asociat lui ψ ca subarbore stâng al rădăcinii și arborele binar asociat lui χ ca subarbore drept al rădăcinii.

Mnemonic despre relații n -are, relații binare

Definiție

Fie $n \in \mathbb{N}^*$ și A_1, A_2, \dots, A_n mulțimi. Se numește *relație n -ară* între mulțimile A_1, A_2, \dots, A_n o submulțime a produsului cartezian $A_1 \times A_2 \times \dots \times A_n$.

Observație

Pentru $n = 1$ în definiția anterioară se obține noțiunea de *relație unară* pe o mulțime: prin definiție, o *relație unară* pe o mulțime A este o submulțime a lui A . Pentru $n = 2$ în definiția anterioară se obține noțiunea de *relație binară*.

Definiție

Fie A și B două mulțimi. Se numește *relație binară* între A și B o submulțime R a produsului direct $A \times B$.

Pentru fiecare $a \in A$ și fiecare $b \in B$, faptul că $(a, b) \in R$ se mai notează cu $a R b$ și se citește: a este în relația R cu b .

Exemplu

Pentru orice mulțimi A și B , produsul direct $A \times B$ este o relație binară între A și B (evident, cea mai mare în sensul incluziunii dintre toate relațiile binare între A și B).

Mnemonic despre relații binare pe o mulțime

- Fie A o mulțime.

Definiție

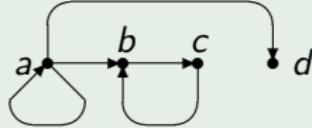
Se numește *relație binară* pe A o relație binară între A și A , i. e. o submulțime a produsului direct $A^2 = A \times A$.

Remarcă

Dacă A este finită și nevidă, iar R este o relație binară pe A , atunci perechea (A, R) este un graf orientat (cu mulțimea vârfurilor A și mulțimea arcelor R), aşadar relația binară R poate fi reprezentată grafic chiar prin acest graf orientat.

Exemplu

Relația binară $R = \{(a, a), (a, b), (a, d), (b, c), (c, b)\}$ pe mulțimea cu exact 4 elemente $A = \{a, b, c, d\}$ poate fi reprezentată grafic astfel:



Tipuri de relații binare pe o mulțime

Definiție (tipuri de relații binare pe o mulțime)

Fie $R \subseteq A^2$ (i. e. R o relație binară pe A). R se zice:

- *reflexivă* dacă, pentru orice $a \in A$, aRa ;
- *ireflexivă* dacă, pentru orice $a \in A$, $(a, a) \notin R$, i. e. nu există $a \in A$ cu aRa ;
- *simetrică* dacă, pentru orice $a, b \in A$, dacă aRb , atunci bRa ;
- *antisimetrică* dacă, pentru orice $a, b \in A$, dacă aRb și bRa , atunci $a = b$;
- *asimetrică* dacă, pentru orice $a, b \in A$, dacă $(a, b) \in R$, atunci $(b, a) \notin R$;
- *tranzitivă* dacă, pentru orice $a, b, c \in A$, dacă aRb și bRc , atunci aRc ;
- *totală* dacă, pentru orice $a, b \in A$ cu $a \neq b$, au loc aRb sau bRa ;
- *completă* dacă, pentru orice $a, b \in A$, au loc aRb sau bRa .

Tipuri de relații binare pe o mulțime

Definiție

Fie $R \subseteq A^2$ (i. e. R o relație binară pe A). R se numește:

- (*relație de*) *preordine* dacă este reflexivă și tranzitivă;
- (*relație de*) *echivalență* dacă este o preordine simetrică, i. e. o relație reflexivă, simetrică și tranzitivă;
- (*relație de*) *ordine (parțială)* dacă este o preordine antisimetrică, i. e. o relație reflexivă, tranzitivă și antisimetrică;
- (*relație de*) *ordine totală* dacă este simultan o relație de ordine și o relație totală;
- (*relație de*) *ordine strictă* dacă este asimetrică și tranzitivă (echivalent, dacă este ireflexivă și tranzitivă).

Remarcă

Întrucât orice relație de ordine este reflexivă, rezultă că o relație de ordine este totală dacă este completă.

Partiție a unei mulțimi

Definiție

Fie A nevidă și $(A_i)_{i \in I}$ o familie nevidă (i. e. cu $I \neq \emptyset$) de submulțimi ale lui A . Familia $(A_i)_{i \in I}$ se numește *partiție* a lui A dacă satisfac următoarele condiții:

- ① pentru orice $i \in I$, $A_i \neq \emptyset$
- ② pentru orice $i, j \in I$, dacă $i \neq j$, atunci $A_i \cap A_j = \emptyset$ (i. e. mulțimile din familia $(A_i)_{i \in I}$ sunt două câte două disjuncte)
- ③ $\bigcup_{i \in I} A_i = A$

Propoziție

Fie A nevidă și $(A_i)_{i \in I}$ o partiție a lui A . Atunci, pentru orice $x \in A$, există un unic $i_0 \in I$, a. i. $x \in A_{i_0}$.

Clase de echivalență, mulțime factor (mulțime cât)

Pentru cele ce urmează, vom considera mulțimea A nevidă, și o relație de echivalență \sim pe A , i. e.:

- \sim este o relație binară pe A : $\sim \subseteq A^2$
- \sim este **reflexivă**: pentru orice $x \in A$, $x \sim x$
- \sim este **simetrică**: pentru orice $x, y \in A$, dacă $x \sim y$, atunci $y \sim x$
- \sim este **tranzitivă**: pentru orice $x, y, z \in A$, dacă $x \sim y$ și $y \sim z$, atunci $x \sim z$

Să observăm că, în definiția simetriei, putem interschimba x și y și continua seria de implicații, obținând implicație dublă, adică: \sim este **simetrică** dacă, pentru orice $x, y \in A$, are loc echivalența: $x \sim y$ dacă și numai $y \sim x$.

Definiție

Pentru fiecare $x \in A$, definim *clasa de echivalență a lui x raportat la \sim* ca fiind următoarea submulțime a lui A , notată cu \hat{x} sau cu x/\sim :

$$\hat{x} := x/\sim := \{y \in A \mid x \sim y\}.$$

Remarcă

Observăm că simetria lui \sim ne asigură de faptul că: pentru orice $x \in A$, $\hat{x} = \{y \in A \mid y \sim x\}$.

Proprietățile claselor de echivalență

Propoziție

Pentru orice $x, y \in A$:

- $x \sim y$ dacă și numai dacă $y \sim x$ dacă și numai dacă $x \in \hat{y}$ dacă și numai dacă $y \in \hat{x}$ dacă și numai dacă $\hat{x} = \hat{y}$;
- $(x, y) \notin \sim$ dacă și numai dacă $(y, x) \notin \sim$ dacă și numai dacă $x \notin \hat{y}$ dacă și numai dacă $y \notin \hat{x}$ dacă și numai dacă $\hat{x} \cap \hat{y} = \emptyset$.

Definiție

Fiecare $x \in A$ se numește *reprezentant al clasei* \hat{x} .

Remarcă

- Pentru fiecare $x \in A$, orice $y \in \hat{x}$ este reprezentant al clasei \hat{x} .

Într-adevăr, conform propoziției precedente, pentru orice $x, y \in A$, are loc echivalența: $y \in \hat{x}$ dacă și numai dacă $\hat{x} = \hat{y}$, iar, conform definiției anterioare, orice y este reprezentant al clasei \hat{y} , care este egală cu \hat{x} exact atunci când $y \in \hat{x}$, deci orice $y \in \hat{x}$ este reprezentant al clasei \hat{x} .

Mai mult:

- pentru fiecare $x, y \in A$, y este reprezentant al clasei \hat{x} dacă și numai dacă $y \in \hat{x}$.

Definiție

Mulțimea claselor de echivalență ale lui \sim se notează cu A/\sim și se numește *mulțimea factor a lui A prin \sim* sau *mulțimea cât a lui A prin \sim* :

$$A/\sim = \{\hat{x} \mid x \in A\}.$$

Funcția $x \mapsto \hat{x}$ de la A la A/\sim e surjectivă și se numește *surjecția canonică*.

Observație

Denumirile de **mulțime factor** și **mulțime cât** se datorează faptului că mulțimea A/\sim din definiția anterioară se obține prin "împărțirea mulțimii A în clasele de echivalență ale lui \sim " (a se vedea următoarea propoziție).

Propoziție (clasele de echivalență formează o partiție)

Mulțimea factor A/\sim este o partiție a lui A.

Notație

Notăm cu:

- $\text{Eq}(A)$ mulțimea relațiilor de echivalență pe A ;
- $\text{Part}(A)$ mulțimea partițiilor lui A .

Propoziție ($\text{Eq}(A) \cong \text{Part}(A)$)

Aplicația $\sim \mapsto A/\sim$ de la mulțimea $\text{Eq}(A)$ a relațiilor de echivalență pe A la mulțimea $\text{Part}(A)$ a partițiilor lui A este o bijecție.

Egalitatea enunțurilor modulo parantezări corecte

Definiție (enunțurile, arborii binari asociați lor și parantezările corecte – continuare)

Considerăm relația binară \approx pe E definită prin: oricare ar fi $\varphi, \psi \in E$, $\varphi \approx \psi$ dacă și ψ au același arbore binar asociat, în care ordinea fiilor unui nod contează. Evident, \approx este o relație de echivalență pe E .

Identificăm mulțimea factor E/\approx cu E , identificând fiecare $\varphi \in E$ cu φ/\approx . Deci oricare două enunțuri φ, ψ astfel încât $\varphi \approx \psi$ vor fi considerate egale.

Exemplu

Dacă $p, q \in V$, atunci $(\neg p) \rightarrow (\neg(\neg q)) \approx \neg p \rightarrow \neg\neg q$, aşadar considerăm $(\neg p) \rightarrow (\neg(\neg q)) = \neg p \rightarrow \neg\neg q$.

Conecțorii logici derivați

Notație (abrevieri pentru enunțuri compuse)

Pentru orice enunțuri $\varphi, \psi \in E$, introducem notațiile (abrevierile):

- | | |
|--|--|
| $\varphi \vee \psi := \neg \varphi \rightarrow \psi$ | (disjuncția dintre φ și ψ ; se citește: φ "sau" ψ) |
| $\varphi \wedge \psi := \neg(\varphi \rightarrow \neg \psi)$ | (conjuncția dintre φ și ψ ; se citește: φ "și" ψ) |
| $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$ | (echivalența logică dintre φ și ψ ; se citește:
φ "echivalent cu" ψ) |

Definiție

Simbolurile \vee , \wedge și \leftrightarrow se numesc *conecțorii logici derivați*.

Observație

Conecțorii logici derivați se scriu ca niște operatori binari, și le vom acorda aceeași prioritate cu aceea a coneceptorului logic primitiv "binar" \rightarrow .

Remarcă

În această prezentare a sistemului formal al logicii propoziționale clasice, am considerat negația și implicația ca și coneceptorii logici primitivi, iar disjuncția, conjuncția și echivalența ca și coneceptorii logici derivați, introdusi prin notațiile de mai sus, pe baza celor primitivi.

Există prezentări ale sistemului formal al logicii propoziționale clasice care sunt echivalente cu cea din acest curs și care folosesc alți coneceptorii logici primitivi.

Schemele (i. e. tipurile) de axiome

- Am definit limbajul cu care vom lucra. Acum vom defini, tot la acest nivel, formal, sintactic, noțiunea de “adevăr” în logica pe care o construim. “Adevărurile sintactice” vor fi “teoremele” acestei logici, iar, pentru a le obține, vom da un set de axiome și vom defini o modalitate prin care, din adevăruri sintactice stabilite până la un moment dat, se deduc alte adevăruri sintactice. Acea modalitate se va numi **regula de deducție modus ponens**.

Definiție

O *axiomă* a sistemului formal al logicii propoziționale clasice este un enunț de oricare dintre următoarele trei forme, unde $\varphi, \psi, \chi \in E$ sunt enunțuri arbitrarе:

$$\begin{aligned}(A_1) \quad & \varphi \rightarrow (\psi \rightarrow \varphi) \\(A_2) \quad & (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)) \\(A_3) \quad & (\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi)\end{aligned}$$

Fiecare dintre scrierile (A_1) , (A_2) și (A_3) este o *schemă de axiome*, adică o regulă pentru generarea unui număr infinit de axiome.

Mulțimea axiomelor, i. e. a enunțurilor pornind de la care se deduc adevărurile sintactice

Definiție (continuare)

Axiomele logicii propoziționale clasice se obțin prin înlocuirea, în aceste scheme de axiome, a enunțurilor generice (arbitrare) φ, ψ, χ cu enunțuri precizate (date), adică axiomele sunt enunțuri de una dintre formele (A_1) , (A_2) și (A_3) , cu φ, ψ și χ enunțuri date.

Prin extensie, vom numi uneori schemele de axiome (A_1) , (A_2) și (A_3) , simplu, **axiome**.

Notație

Notăm cu Ax mulțimea axiomelor:

$$Ax = \{ \varphi \rightarrow (\psi \rightarrow \varphi), \\ (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)), \\ (\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi) \quad | \varphi, \psi, \chi \in E \}.$$

- Așa cum am anunțat, acum vom defini **deduția sintactică (inferență sintactică)**, adică modalitatea prin care, din aceste axiome, se obțin toate **teoremele (adevărurile sintactice)** în acest sistem logic.

Remarcă

Se poate demonstra că schemele de axiome de mai sus sunt **independente**, i. e. niciuna dintre ele nu poate fi dedusă din celelalte două prin modalitatea de deducție sintactică pe care o vom defini.

O modalitate pentru a demonstra acest fapt este să se definească semantica logicii propoziționale clasice, ca în cele ce urmează, să se demonstreze Teorema de Completitudine Tare, care afirmă că deducția sintactică, coincide cu deducția semantică, apoi să se arate că, pentru fiecare două dintre aceste trei axiome, există cel puțin o evaluare care le atribuie valoarea 1 (repräsentând **adevărul**), dar celei de-a treia axiome îi atribuie valoarea 0 (repräsentând **falsul**).

Acest fapt arată și că, între aceste trei scheme de axiome, nu există două din care să se deducă sintactic **toate** adevărurile sintactice (teoremele) acestui sistem logic, care se deduc din toate cele trei scheme de axiome.

Se pot, însă, defini alte seturi echivalente de scheme de axiome (adică seturi de scheme de axiome din care se deduc sintactic aceleași teoreme), cu aceeași sau cu alți conectori logici primitivi.

Notație (scrierea regulilor de deducție)

Notația uzuală pentru reguli de deducție ale unui sistem logic, pe care o vom folosi în cele ce urmează, este aceasta: $\frac{\text{condiția } C_1}{\text{consecința } C_2}$, cu semnificația că: dacă este satisfăcută condiția C_1 , atunci este satisfăcută consecința C_2 .

Regula de deducție **modus ponens** și teoremele formale

Definiție (teoremele (formale), i. e. adevărurile sintactice)

Teoremele formale (numite și, simplu, *teoreme*, sau *adevăruri sintactice*) ale logicii propoziționale clasice sunt enunțurile definite prin următoarele trei reguli:

- (T_1) orice axiomă este o teoremă formală;
- (T_2) dacă $\varphi, \psi \in E$ sunt două enunțuri a. î. ψ și $\psi \rightarrow \varphi$ sunt teoreme formale, atunci φ este o teoremă formală;
- (T_3) orice teoremă formală a logicii propoziționale clasice poate fi obținută prin aplicarea regulilor (T_1) și (T_2) de un număr finit de ori.

Notății

Mulțimea tuturor teoremelor formale va fi notată cu T .

Faptul că un enunț φ este teoremă formală se notează: $\vdash \varphi$.

Definiție (regula de deducție modus ponens (MP))

Regula (T_2) se numește *regula de deducție modus ponens* (o vom abrevia "MP") și, cu notația stabilită mai sus, poate fi scrisă astfel în formă simbolică:

$$\frac{\vdash \psi, \vdash \psi \rightarrow \varphi}{\vdash \varphi}, \text{ sau, echivalent: } \frac{\psi, \psi \rightarrow \varphi}{\varphi}.$$

Definiția recursivă a mulțimii teoremelor formale

Remarcă

Regula (T_3) , chiar fără precizarea finitudinii, spune că T este cea mai mică mulțime închisă la regulile (T_1) și (T_2) , i. e. cea mai mică mulțime de enunțuri care include mulțimea axiomelor și e închisă la regula (MP) , i. e. cea mai mică submulțime M a lui E cu proprietățile:

- ① $M \supseteq Ax$,
- ② pentru orice $\varphi, \psi \in E$, dacă $\psi, \psi \rightarrow \varphi \in M$, atunci $\varphi \in M$,

(cea mai mică în sensul incluziunii), pentru că regula (T_3) spune că nu se află în T niciun element care să nu se obțină prin aplicarea regulilor (T_1) și (T_2) , adică niciun element care să nu fie nici axiomă, nici enunț obținut prin aplicarea succesivă a regulii **MP**, pornind de la axiome.

Demonstrațiile formale

Definiție

Fie φ un enunț. O *demonstrație formală* pentru φ este un sir finit și nevid de enunțuri $\varphi_1, \dots, \varphi_n$, a. î. $n \in \mathbb{N}^*$, $\varphi_n = \varphi$ și, pentru fiecare $i \in \overline{1, n}$, este satisfăcută una dintre următoarele condiții:

- ① φ_i este o axiomă;
- ② există $k, j \in \overline{1, i - 1}$ a. î. $\varphi_k = \varphi_j \rightarrow \varphi_i$.

n se numește *lungimea* demonstrației formale $\varphi_1, \dots, \varphi_n$.

Remarcă

În scrierea definiției de mai sus, am folosit faptul că $\overline{1, 0} = \emptyset$ (pentru o scriere uniformă a definiției, fără a trata separat cazul $i = 1$). Având în vedere acest lucru, este clar că, într-o demonstrație formală $\varphi_1, \dots, \varphi_n$, φ_1 este o axiomă.

Remarcă

Este imediat că, dacă $\varphi_1, \dots, \varphi_n$ este o demonstrație formală, atunci, pentru orice $i \in \overline{1, n}$, $\varphi_1, \dots, \varphi_i$ este o demonstrație formală.

Teoremele sunt enunțurile care admit demonstrații formale

Remarcă

Este ușor de observat că regulile (1) și (2) din definiția unei demonstrații formale exprimă exact regulile (T_1) și (T_2) , respectiv, ceea ce arată imediat faptul că un enunț φ este o teoremă formală dacă există o demonstrație formală pentru φ .

Remarcă

Desigur, o teoremă formală poate avea mai multe demonstrații formale și poate avea demonstrații formale de lungimi diferite.

Enunțurile deductibile din ipoteze

Definiție

Fie $\Sigma \subseteq E$ o mulțime de enunțuri. Enunțurile care se deduc sintactic din ipotezele Σ , numite și *consecințele sintactice ale lui Σ* , se definesc astfel:

- (CS_1) orice axiomă se deduce sintactic din ipotezele Σ ;
- (CS_0) orice enunț $\varphi \in \Sigma$ se deduce sintactic din ipotezele Σ ;
- (CS_2) dacă $\varphi, \psi \in E$ sunt două enunțuri a. î. ψ și $\psi \rightarrow \varphi$ se deduc sintactic din ipotezele Σ , atunci φ se deduce sintactic din ipotezele Σ ;
- (CS_3) orice enunț care se deduce sintactic din ipotezele Σ se poate obține prin aplicarea regulilor (CS_1) , (CS_0) și (CS_2) de un număr finit de ori.

Notație

Notăm faptul că un enunț φ se deduce sintactic din ipotezele $\Sigma \subseteq E$ prin: $\Sigma \vdash \varphi$.

Definiție (și regula de deducție din ipoteze este tot modus ponens)

Regula (CS_2) se numește tot *regula de deducție modus ponens* (o vom abrevia tot "MP") și, cu notația stabilită mai sus, poate fi scrisă astfel în formă simbolică:

$$\frac{\Sigma \vdash \psi, \Sigma \vdash \psi \rightarrow \varphi}{\Sigma \vdash \varphi}.$$

Definiția recursivă a unei mulțimi de consecințe sintactice

Remarcă

Regula (CS_3) , chiar fără precizarea finitudinii numărului de aplicări ale acestor reguli, spune că mulțimea consecințelor sintactice ale unei mulțimi Σ de enunțuri este cea mai mică mulțime închisă la regulile (CS_1) , (CS_0) și (CS_2) , adică cea mai mică mulțime de enunțuri care include mulțimea axiomelor și mulțimea Σ a ipotezelor și e închisă la regula (MP) , i. e. cea mai mică submulțime M a lui E cu proprietățile:

- ① $M \supseteq Ax$,
- ② $M \supseteq \Sigma$,
- ③ pentru orice $\varphi, \psi \in E$, dacă $\psi, \psi \rightarrow \varphi \in M$, atunci $\varphi \in M$,

(cea mai mică în sensul incluziunii), pentru că (CS_3) spune că mulțimea consecințelor sintactice ale lui Σ nu conține alte elemente decât cele obținute din regulile (CS_1) , (CS_0) și (CS_2) .

Remarcă (deducția pornind de la axiome și de la ipotezele din Σ)

Definiția consecințelor sintactice ale lui Σ este exact definiția teoremelor formale în care mulțimea Ax se înlocuiește cu $Ax \cup \Sigma$.

Demonstrații formale din ipoteze

Definiție

Fie φ un enunț și Σ o mulțime de enunțuri. O Σ -demonstrație formală pentru φ este un sir finit și nevid de enunțuri $\varphi_1, \dots, \varphi_n$, a. î. $n \in \mathbb{N}^*$, $\varphi_n = \varphi$ și, pentru fiecare $i \in \overline{1, n}$, este satisfăcută una dintre următoarele condiții:

- ① φ_i este o axiomă;
- ② $\varphi_i \in \Sigma$;
- ③ există $k, j \in \overline{1, i - 1}$ a. î. $\varphi_k = \varphi_j \rightarrow \varphi_i$.

n se numește lungimea Σ -demonstrației formale $\varphi_1, \dots, \varphi_n$.

Remarcă

Amintindu-ne că $\overline{1, 0} = \emptyset$, este clar că, într-o Σ -demonstrație formală $\varphi_1, \dots, \varphi_n$, φ_1 este o axiomă sau un element al lui Σ .

Remarcă

Este imediat că, dacă Σ este o mulțime de enunțuri și $\varphi_1, \dots, \varphi_n$ este o Σ -demonstrație formală, atunci, pentru orice $i \in \overline{1, n}$, $\varphi_1, \dots, \varphi_i$ este o Σ -demonstrație formală.

Enunțurile deductibile din ipoteze sunt exact enunțurile care admit demonstrații formale din acele ipoteze

Remarcă

Este ușor de observat că regulile (1), (2) și (3) din definiția unei Σ -demonstrații formale exprimă exact regulile (CS_1) , (CS_0) și (CS_2) , respectiv, ceea ce arată imediat faptul că un enunț φ este o consecință sintactică a lui Σ dacă există o Σ -demonstrație formală pentru φ .

Remarcă

Desigur, o consecință sintactică a lui Σ poate avea mai multe Σ -demonstrații formale și poate avea Σ -demonstrații formale de lungimi diferite.

Remarcă

Este imediat, direct din definițiile date, că, pentru orice enunț φ și orice mulțime de enunțuri Σ :

- ① $\emptyset \vdash \varphi$ dacă $\vdash \varphi$;
- ② dacă $\vdash \varphi$, atunci $\Sigma \vdash \varphi$;
- ③ dacă $\varphi \in \Sigma$, atunci $\Sigma \vdash \varphi$.

Acesta este **sistemul Hilbert**

Întreaga prezentare de până acum a fost efectuată la **nivel sintactic**: am pornit de la un **alfabet** (o **mulțime de simboluri**), am definit un tip particular de **cuvinte peste acest alfabet**, numite **enunțuri**, apoi un tip particular de enunțuri, numite **teoreme formale**, și **deduția sintactică (inferența sintactică)**, care indică modul în care, din teoreme formale, se obțin alte teoreme formale, cu generalizarea la **consecințe sintactice** ale unor mulțimi de enunțuri.

Definiție

Deduția pe baza axiomelor (A_1) , (A_2) , (A_3) și a regulii de deducție **MP** se numește *sistemul Hilbert* pentru calculul propozițional clasic.

Notație

Vom nota cu \mathcal{L} acest sistem formal pentru logica propozițională clasică.

În tot restul acestui capitol – i.e. până la secțiunea despre rezoluția propozițională – ne vom referi la *sistemul Hilbert* pentru logica propozițională clasică.

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Propoziție (o numim ad-hoc Propoziția *)

Fie $\Sigma \subseteq E$, $\Delta \subseteq E$ și $\varphi \in E$. Atunci:

- ① dacă $\Sigma \subseteq \Delta$ și $\Sigma \vdash \varphi$, atunci $\Delta \vdash \varphi$;
- ② dacă $\Sigma \vdash \varphi$, atunci există $\Gamma \subseteq \Sigma$ a. î. Γ este o mulțime finită și $\Gamma \vdash \varphi$;
- ③ dacă $\Sigma \vdash \psi$ pentru orice $\psi \in \Delta$ și $\Delta \vdash \varphi$, atunci $\Sigma \vdash \varphi$.

Remarcă

Conform punctului (1), în (2) din Propoziția * avem chiar echivalență:
 $\Sigma \vdash \varphi$ dacă $(\exists \Gamma \subseteq \Sigma) (|\Gamma| < \aleph_0, \Gamma \vdash \varphi)$.

Propoziție (principiul identității și principiul terțului exclus)

Pentru orice $\varphi \in E$, următoarele enunțuri sunt teoreme formale:

- ① **principiul identității** (abbreviat PI): $\vdash \varphi \rightarrow \varphi$;
- ② **principiul terțului exclus** (abbreviat PTE): $\vdash \varphi \vee \neg \varphi$.

Teoremă (Teorema deducției – abreviată TD)

Pentru orice $\Sigma \subseteq E$ și orice $\varphi, \psi \in E$, are loc următoarea echivalență:

$$\Sigma \vdash \varphi \rightarrow \psi \quad \text{dacă} \quad \Sigma \cup \{\varphi\} \vdash \psi.$$

Limbajul acestei teorii matematice versus metalimbaj

Observație

Denumirea de **teoremă** pentru rezultatul anterior este o denumire din **metalimbaj**, pentru că acest rezultat este o proprietate a sistemului formal al logicii propoziționale clasice.

Denumirea de **teoremă formală** este din limbajul sistemului formal al logicii propoziționale clasice, ea denumește un tip de obiect cu care lucrează acest sistem formal.

Este importantă această distincție. Similaritatea celor două denumiri se datorează faptului că acest sistem formal (al logicii propoziționale clasice) este o formalizare a unor legi ale gândirii (în special a procedeelor gândirii care sunt, în mod curent, folosite în elaborarea raționamentelor matematice), și conține denumiri care îi sugerează întrebuițarea, destinația.

Aceleași considerații sunt valabile pentru **conectorii logici** din sistemul formal al calculului propozițional clasic: \neg , \rightarrow , \vee , \wedge , \leftrightarrow , versus **conectorii logici din metalimbaj**, folosiți în enunțurile referitoare la calculul propozițional clasic, scriși fie prin denumirile lor: "non" sau "not" sau "negație", "implică" sau "rezultă", "sau", "și", "echivalent" sau "ddacă", fie prin simbolurile consacrate, în cazul implicației și echivalenței: \Rightarrow și \Leftrightarrow , respectiv.

La fel pentru alți termeni din acest sistem logic, precum și din sistemul formal al calculului cu predicate clasic, prezentat în ultimul curs.

Fie $n \in \mathbb{N}^*$ și $\varphi, \varphi_1, \dots, \varphi_n \in E$, arbitrare, fixate.

Notație (pentru următoarele reguli de deducție)

Regula de deducție $\frac{\varphi_1, \dots, \varphi_n}{\varphi}$ va semnifica faptul că φ se deduce printr-o demonstrație formală din ipotezele $\varphi_1, \dots, \varphi_n$, adică: $\{\varphi_1, \dots, \varphi_n\} \vdash \varphi$.

Remarcă (cu notația de tip $\frac{\text{ipoteze}}{\text{concluzie}}$)

Regula de deducție $\frac{\varphi_1, \dots, \varphi_n}{\varphi}$ implică regula $\frac{\Sigma \vdash \varphi_1, \dots, \Sigma \vdash \varphi_n}{\Sigma \vdash \varphi}$ pentru orice $\Sigma \subseteq E$, de unde, luând $\Sigma = \emptyset$, obținem și cazul particular: $\frac{\vdash \varphi_1, \dots, \vdash \varphi_n}{\vdash \varphi}$.

Într-adevăr, dacă avem $\frac{\varphi_1, \dots, \varphi_n}{\varphi}$, adică $\{\varphi_1, \dots, \varphi_n\} \vdash \varphi$, și au loc $\Sigma \vdash \varphi_1, \dots, \Sigma \vdash \varphi_n$, atunci, conform afirmației (3) din Propoziția \star , rezultă că $\Sigma \vdash \beta$.

Remarcă (modus ponens, cu scrierea de mai sus)

Conform afirmației (3) din Propoziția \star , pentru orice $\Sigma \subseteq E$, mulțimea consecințelor sintactice ale lui Σ , în particular mulțimea teoremelor formale, este închisă la regula $\frac{\varphi, \varphi \rightarrow \psi}{\psi}$, pe care o numim tot **modus ponens**.

Notație (disjuncție și conjuncție logică între n enunțuri – abbrevieri definite recursiv)

Fie $\gamma_1, \gamma_2, \dots, \gamma_n, \dots \in E$, arbitrar. Atunci, pentru orice $n \in \mathbb{N}^*$, avem următoarele scrieri (prioritățile: ca la operatori unari, deci aceeași prioritate ca \neg):

$$\bigvee_{i=1}^n \gamma_i := \begin{cases} \gamma_1, & n = 1, \\ (\bigvee_{i=1}^{n-1} \gamma_i) \vee \gamma_n, & n > 1, \end{cases} \quad \text{și} \quad \bigwedge_{i=1}^n \gamma_i := \begin{cases} \gamma_1, & n = 1, \\ (\bigwedge_{i=1}^{n-1} \gamma_i) \wedge \gamma_n, & n > 1. \end{cases}$$

Exercițiu (temă)

Să se arate că, pentru orice $\varphi, \psi, \chi \in E$:

$$\vdash ((\varphi \wedge \psi) \rightarrow \chi) \rightarrow (\varphi \rightarrow (\psi \rightarrow \chi)),$$

iar, folosind acest fapt, să se arate că, pentru orice $n \in \mathbb{N}^*$ și orice $\gamma_1, \gamma_2, \dots, \gamma_n, \varphi \in E$, au loc echivalențele:

$$\{\gamma_1, \gamma_2, \dots, \gamma_n\} \vdash \varphi \quad \text{ddacă} \quad \{\bigwedge_{i=1}^n \gamma_i\} \vdash \varphi \quad \text{ddacă} \quad \vdash (\bigwedge_{i=1}^n \gamma_i) \rightarrow \varphi.$$

A se vedea și Propoziția 7.2.53/pagina 160/carte de G. Georgescu și A. Iorgulescu din bibliografia din breviarul cursului de logică matematică și computațională, precum și demonstrația Propoziției \star de mai sus.

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Conecțorii logici derivați și axiomele

Notație

$E :=$ mulțimea enunțurilor.

Notație (conectorii logici derivați)

Pentru orice $\varphi, \psi \in E$:

- $\varphi \vee \psi := \neg \varphi \rightarrow \psi$
- $\varphi \wedge \psi := \neg (\varphi \rightarrow \neg \psi)$
- $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

((schemele de) axiome)

Pentru orice $\varphi, \psi, \chi \in E$:

- (A₁) $\varphi \rightarrow (\psi \rightarrow \varphi)$
(A₂) $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$
(A₃) $(\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi)$

Notăție

$Ax :=$ mulțimea axiomelor, i. e.: $Ax := \{\varphi \rightarrow (\psi \rightarrow \varphi), (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)), (\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi) \mid \varphi, \psi, \chi \in E\}$.

Deducrea sintactică: pornind de la axiome și, eventual, ipoteze, pe baza regulii de deducție **modus ponens** (MP)

Notație

$T :=$ mulțimea teoremelor formale (i. e. a adevărurilor sintactice).

Fie $\varphi, \psi, \chi \in E$, $\Sigma \subseteq E$ și $\Delta \subseteq E$, arbitrale.

Notație

- $\vdash \varphi$: φ este teoremă formală.
- $\Sigma \vdash \varphi$: φ este consecință sintactică a mulțimii Σ de ipoteze.

(regula de deducție **modus ponens** (MP))

$$\frac{\psi, \psi \rightarrow \varphi}{\varphi}$$

Remarcă

- $Ax \subseteq T$
- $\vdash \varphi \iff \emptyset \vdash \varphi \implies \Sigma \vdash \varphi$
- $(\forall \sigma \in \Sigma) (\Sigma \vdash \sigma)$

Propoziție (\star)

- ① $\Sigma \subseteq \Delta$ și $\Sigma \vdash \varphi \implies \Delta \vdash \varphi$
- ② $\Sigma \vdash \varphi \iff (\exists \Gamma \subseteq \Sigma) (|\Gamma| < \aleph_0, \Gamma \vdash \varphi)$
- ③ $(\forall \psi \in \Delta) (\Sigma \vdash \psi)$ și $\Delta \vdash \varphi \implies \Sigma \vdash \varphi$

Propoziție

- **principiul identității (PI):** $\vdash \varphi \rightarrow \varphi$
- **principiul terțuluiui exclus (PTE):** $\vdash \varphi \vee \neg \varphi$

Teoremă (Teorema deducției (TD))

$$\Sigma \vdash \varphi \rightarrow \psi \iff \Sigma \cup \{\varphi\} \vdash \psi$$

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Ordine versus ordine strictă

Remarcă

Dacă A e o mulțime nevidă, atunci nu există nicio relație binară pe A care să fie și reflexivă, și ireflexivă, prin urmare nu există nicio relație binară pe A care să fie și relație de ordine, și relație de ordine strictă.

Exercițiu

Fie A o mulțime, O mulțimea relațiilor de ordine pe A și S mulțimea relațiilor de ordine strictă pe A .

Să se demonstreze că aplicațiile $\varphi : O \rightarrow S$ și $\psi : S \rightarrow O$, definite prin:

- pentru orice $\leq \in O$,
 $\varphi(\leq) = \leq \setminus \{(a, a) \mid a \in A\} = \{(x, y) \in A^2 \mid x \leq y \text{ și } x \neq y\}$ (de obicei notată $\varphi(\leq) = <$ și numită *relația de ordine strictă asociată lui \leq*),
- pentru orice $< \in S$,
 $\psi(<) = < \cup \{(a, a) \mid a \in A\} = \{(x, y) \in A^2 \mid x < y \text{ sau } x = y\},$

sunt:

- corect definite, i. e. într-adevăr $\varphi(O) \subseteq S$ și $\psi(S) \subseteq O$;
- inverse una alteia, deci sunt bijectii între O și S .

Mulțimi ordonate și elemente distinse ale acestora

Definiție

- O mulțime A înzestrată cu o relație de ordine $\leq \subseteq A^2$ se notează (A, \leq) și se numește *mulțime (parțial) ordonată* sau *poset* (de la englezescul “partially ordered set”).
- Dacă, în plus, \leq este o relație de ordine totală, atunci (A, \leq) se numește *mulțime total ordonată* sau *mulțime liniar ordonată* sau *lanț*.
- Fie (A, \leq) un poset și $X \subseteq A$.

Definiție

Un element $a \in A$ se numește:

- *minorant pentru X* dacă, pentru orice $x \in X$, $a \leq x$
- *majorant pentru X* dacă, pentru orice $x \in X$, $x \leq a$

Remarcă

X poate avea mai mulți minoranți (majoranți), și poate să nu aibă niciun minorant (majorant).

Definiție

- Un minorant al lui X care aparține lui X (i. e. un element $m \in X$ cu $m \leq x$ pentru orice $x \in X$) se numește *minim al lui X* sau *prim element al lui X* sau *cel mai mic element al lui X* și se notează cu $\min(X)$ sau $\min(X, \leq)$.
- Un majorant al lui X care aparține lui X (i. e. un element $M \in X$ cu $x \leq M$ pentru orice $x \in X$) se numește *maxim al lui X* sau *ultim element al lui X* sau *cel mai mare element al lui X* și se notează cu $\max(X)$ sau $\max(X, \leq)$.

Remarcă

Minimul nu există întotdeauna. Dar antisimetria lui \leq implică faptul că minimul (dacă există) este unic (ceea ce justifică notația de mai sus pentru minim, care indică faptul că minimul lui X este unic determinat de X (și \leq)).

La fel pentru maxim.

Definiție

Un poset cu minim și maxim se numește *poset mărginit*. (Minimul și maximul trebuie să fie ale întregului poset, deci trebuie luat $X := A$ în definiția anterioară.)

Remarcă

O mulțime care are minim sau maxim are cel puțin un element (pentru că minimul unei mulțimi aparține acelei mulțimi și la fel și maximul), deci nu poate fi vidă.

Alte elemente distinse într-un poset

Definiție

Infimumul lui X este cel mai mare minorant al lui X , adică maximul mulțimii minoranților lui X , și se notează cu $\inf(X)$ sau $\inf(X, \leq)$.

Supremumul lui X este cel mai mic majorant al lui X , adică minimul mulțimii majoranților lui X , și se notează cu $\sup(X)$ sau $\sup(X, \leq)$.

Remarcă

Infimumul nu există întotdeauna, nici măcar atunci când mulțimea minoranților este nevidă.

Dar, fiind maximul unei mulțimi, infimumul (dacă există) este unic (ceea ce îi justifică denumirea cu articol hotărât și notația, fiecare dintre acestea indicând faptul că infimumul este unic determinat de X (și \leq)).

La fel pentru supremum.

Definiția unei algebre Boole

O **algebră Boole** este o **latice distributivă mărginită complementată**, i. e. o structură $(B, \vee, \wedge, \leq, \neg, 0, 1)$ compusă din:

- o mulțime B ,
- o relație de ordine parțială \leq pe B ,
- două operații binare \vee și \wedge pe B , notate infixat,
- două constante $0, 1 \in B$,
- o operație unară \neg pe B ,

iar aceste componente au proprietățile:

- (B, \vee, \wedge, \leq) este o **latice**, i. e.:
 - oricare ar fi $x, y \in B$, există $\text{sup}\{x, y\}$ și $\text{inf}\{x, y\}$ în posetul (B, \leq) ;
 - \vee și \wedge sunt **idempotente**, **comutative** și **asociative**, i. e.: pentru orice $x, y, z \in B$, au loc: $x \vee x = x$, $x \vee y = y \vee x$, $x \vee (y \vee z) = (x \vee y) \vee z$, și la fel pentru \wedge ;
 - \vee și \wedge verifică **absorbția**: pentru orice $x, y \in B$, $x \vee (x \wedge y) = x$ și $x \wedge (x \vee y) = x$;
 - pentru orice $x, y \in B$, $x \leq y$ dacă $x \vee y = y$ dacă $x \wedge y = x$;
 - pentru orice $x, y \in B$, $x \vee y = \text{sup}\{x, y\}$;
 - pentru orice $x, y \in B$, $x \wedge y = \text{inf}\{x, y\}$;

Definiția unei algebre Boole

- laticea (B, \vee, \wedge, \leq) este **distributivă**, i. e.:
 - \vee este **distributivă** față de \wedge , i. e.: pentru orice $x, y, z \in B$,
 $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z);$
 - \wedge este **distributivă** față de \vee , i. e.: pentru orice $x, y, z \in B$,
 $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z);$
- $(B, \vee, \wedge, \leq, 0, 1)$ este o **latice mărginită**, i. e., în plus:
 - 0 este **minimul** posetului (B, \leq) ;
 - 1 este **maximul** posetului (B, \leq) ;
- laticea mărginită $(B, \vee, \wedge, \leq, 0, 1)$ este **complementată** și satisfacă **unicitatea complementului**, datorită **distributivității**, iar \neg este operația de **complementare**:
 - pentru orice $x \in B$, \bar{x} este **unicul complement** al lui x , adică **unicul element** $\bar{x} \in B$ care satisfacă:
$$\begin{cases} x \vee \bar{x} = 1 \\ \text{și} \\ x \wedge \bar{x} = 0. \end{cases}$$

În plus, în orice algebră Boole $(B, \vee, \wedge, \leq, \neg, 0, 1)$, se definesc următoarele **operații binare deriveate**:

- **implicația (booleană)**, \rightarrow : pentru orice $x, y \in B$, $x \rightarrow y := \bar{x} \vee y;$
- **echivalența (booleană)**, \leftrightarrow : pentru orice $x, y \in B$,
 $x \leftrightarrow y := (x \rightarrow y) \wedge (y \rightarrow x).$

Proprietăți ale unei algebre Boole

- Fie $\mathcal{B} := (B, \vee, \wedge, \leq, \bar{\cdot}, 0, 1)$ o algebră Boole.

Propoziție (autodualitatea complementării)

Pentru orice $x \in B$, $\bar{\bar{x}} = x$.

Propoziție (legile lui de Morgan)

Pentru orice $x, y \in B$:

- ① $\overline{x \vee y} = \bar{x} \wedge \bar{y}$
- ② $\overline{x \wedge y} = \bar{x} \vee \bar{y}$

Propoziție

Pentru orice $x, y \in B$, au loc următoarele echivalențe:

- ① $x = y$ dacă $\bar{x} = \bar{y}$
- ② $x \leq y$ dacă $\bar{y} \leq \bar{x}$
- ③ $x \leq y$ dacă $x \rightarrow y = 1$
- ④ $x = y$ dacă $x \leftrightarrow y = 1$

Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice

- Această secțiune a cursului expune construcția unei algebrelor Boole care este asociată în mod canonico-sistemului formal \mathcal{L} .
 - Prin această asociere, proprietățile sintactice ale lui \mathcal{L} se reflectă în proprietăți booleene, și invers.
-
- Pe tot parcursul acestei secțiuni, $\Sigma \subseteq E$ va fi o mulțime arbitrară fixată de enunțuri ale lui \mathcal{L} .
 - Σ va reprezenta o mulțime de ipoteze, ceea ce este adesea numită o *teorie* a lui \mathcal{L} .

Definiție (o relație de echivalență pe mulțimea enunțurilor)

Definim o relație binară \sim_{Σ} pe mulțimea E a enunțurilor lui \mathcal{L} , astfel: pentru orice $\varphi, \psi \in E$,

$$\varphi \sim_{\Sigma} \psi \text{ dacă } \Sigma \vdash \varphi \leftrightarrow \psi.$$

Lemă

\sim_{Σ} este o relație de echivalență pe E .

Notație

Să notăm, pentru fiecare $\varphi \in E$, cu $\hat{\varphi}^{\Sigma} := \{\psi \in E \mid \varphi \sim_{\Sigma} \psi\}$ clasa de echivalență a lui φ raportat la relația de echivalență \sim_{Σ} , și să considerăm mulțimea factor $E/\sim_{\Sigma} = \{\hat{\varphi}^{\Sigma} \mid \varphi \in E\}$.

Definiție (o relație de ordine pe mulțimea factor)

Pe mulțimea factor E/\sim_{Σ} , definim relația binară \leq_{Σ} , prin: oricare ar fi $\varphi, \psi \in E$, $\hat{\varphi}^{\Sigma} \leq_{\Sigma} \hat{\psi}^{\Sigma}$ dacă $\Sigma \vdash \varphi \rightarrow \psi$.

Lemă

\leq_{Σ} este bine definită (i. e. nu depinde de reprezentanții claselor de echivalență ale lui \sim_{Σ}) și este o relație de ordine parțială pe E/\sim_{Σ} .

Propoziție

Posetul $(E / \sim_\Sigma, \leq_\Sigma)$ este o latice distributivă, în care, pentru orice $\varphi, \psi \in E$,

$$\inf\{\hat{\varphi}^\Sigma, \hat{\psi}^\Sigma\} = \widehat{\varphi \wedge \psi}^\Sigma \text{ și } \sup\{\hat{\varphi}^\Sigma, \hat{\psi}^\Sigma\} = \widehat{\varphi \vee \psi}^\Sigma.$$

Vom nota, pentru orice $\varphi, \psi \in E$, $\hat{\varphi}^\Sigma \wedge_\Sigma \hat{\psi}^\Sigma := \widehat{\varphi \wedge \psi}^\Sigma$ și $\hat{\varphi}^\Sigma \vee_\Sigma \hat{\psi}^\Sigma := \widehat{\varphi \vee \psi}^\Sigma$.

Remarcă

Pentru orice $\varphi, \psi \in E$, $\Sigma \vdash (\varphi \wedge \neg \varphi) \rightarrow \psi$ și $\Sigma \vdash \psi \rightarrow (\varphi \vee \neg \varphi)$, așadar, pentru orice $\varphi, \psi \in E$, $\widehat{\varphi \wedge \neg \varphi}^\Sigma \leq_\Sigma \widehat{\psi}^\Sigma \leq_\Sigma \widehat{\varphi \vee \neg \varphi}^\Sigma$. Aceasta înseamnă că, indiferent cine este $\varphi \in E$, $\widehat{\varphi \wedge \neg \varphi}^\Sigma$ și $\widehat{\varphi \vee \neg \varphi}^\Sigma$ sunt, respectiv, primul element și ultimul element al laticii $(E / \sim_\Sigma, \vee_\Sigma, \wedge_\Sigma, \leq_\Sigma)$. Vom nota $0_\Sigma := \widehat{\varphi \wedge \neg \varphi}^\Sigma$ și

$1_\Sigma := \widehat{\varphi \vee \neg \varphi}^\Sigma$, pentru un $\varphi \in E$, arbitrar. Unicitatea minimului și a maximului unui poset arată că această definiție nu depinde de alegerea lui $\varphi \in E$, adică 0_Σ și 1_Σ sunt bine definite.

Așadar:

Propoziție

$(E / \sim_\Sigma, \vee_\Sigma, \wedge_\Sigma, \leq_\Sigma, 0_\Sigma, 1_\Sigma)$ este o latice distributivă mărginită.

Definiție

Pentru orice $\varphi \in E$, definim: $\overline{\hat{\varphi}^\Sigma} := \widehat{\neg \varphi}^\Sigma$.

Remarcă

Definiția de mai sus pentru operația unară $\neg^\Sigma : E/\sim_\Sigma \rightarrow E/\sim_\Sigma$ este corectă, pentru că nu depinde de reprezentanții claselor din E/\sim_Σ .

Propoziție

$(E/\sim_\Sigma, \vee_\Sigma, \wedge_\Sigma, \leq_\Sigma, \neg^\Sigma, 0_\Sigma, 1_\Sigma)$ este o algebră Boole.

Definiție

Algebra Boole $(E/\sim_\Sigma, \vee_\Sigma, \wedge_\Sigma, \leq_\Sigma, \neg^\Sigma, 0_\Sigma, 1_\Sigma)$ se numește *algebra Lindenbaum–Tarski a lui Σ asociată sistemului formal \mathcal{L}* .

Remarcă (surjecția canonică transformă conectorii logici în operații booleene: nu e morfism boolean, pentru că E nu e algebră Boole)

Dacă notăm cu $p_\Sigma : E \rightarrow E/\sim_\Sigma$ surjecția canonică ($p_\Sigma(\varphi) := \hat{\varphi}^\Sigma$ pentru orice $\varphi \in E$), atunci, oricare ar fi $\varphi, \psi \in E$, au loc următoarele identități (unde \rightarrow_Σ și \leftrightarrow_Σ sunt, respectiv, implicația și echivalența booleană în algebra Boole $(E/\sim_\Sigma, \vee_\Sigma, \wedge_\Sigma, \leq_\Sigma, \neg^\Sigma, 0_\Sigma, 1_\Sigma)$):

- (a) $p_\Sigma(\varphi \vee \psi) = p_\Sigma(\varphi) \vee_\Sigma p_\Sigma(\psi);$
- (b) $p_\Sigma(\varphi \wedge \psi) = p_\Sigma(\varphi) \wedge_\Sigma p_\Sigma(\psi);$
- (c) $p_\Sigma(\neg \varphi) = \overline{p_\Sigma(\varphi)}^\Sigma;$
- (d) $p_\Sigma(\varphi \rightarrow \psi) = p_\Sigma(\varphi) \rightarrow_\Sigma p_\Sigma(\psi);$
- (e) $p_\Sigma(\varphi \leftrightarrow \psi) = p_\Sigma(\varphi) \leftrightarrow_\Sigma p_\Sigma(\psi).$

Într-adevăr, identitățile (a), (b) și (c) sunt chiar definițiile operațiilor algebrei Boole E/\sim_Σ . Definiția implicației booleene, (a) și (c) arată că

$p_\Sigma(\varphi) \rightarrow_\Sigma p_\Sigma(\psi) = \overline{p_\Sigma(\varphi)}^\Sigma \vee_\Sigma p_\Sigma(\psi) = p_\Sigma(\neg \varphi \vee \psi)$, ceea ce arată că (d) este echivalent cu faptul că $\Sigma \vdash (\varphi \rightarrow \psi) \leftrightarrow (\neg \varphi \vee \psi)$, care rezultă din (15), (16) și prima remarcă din această secțiune. (e) se obține, direct, din (b) și (d).

Cele cinci identități de mai sus arată cum conectorii logici sunt convertiți în operații booleene.

Enunțurile deductibile din Σ sunt în 1_Σ

Lemă

Pentru orice $\varphi \in E$, $\Sigma \vdash \varphi$ dacă $\hat{\varphi}^\Sigma = 1_\Sigma$.

Remarcă

Lema anterioară ne oferă o metodă algebrică pentru a verifica dacă un enunț este o consecință sintactică a lui Σ .

Notă

- În mod tipic, pentru a folosi lema anterioară în cadrul unei **demonstrații algebrice**, prin calcul boolean, pentru o deducție formală din ipoteze: $\Sigma \vdash \varphi$, cu $\Sigma \subseteq E$ și $\varphi \in E$, se folosește faptul că, pentru orice ipoteză $\sigma \in \Sigma$, are loc $\Sigma \vdash \sigma$, așadar $\hat{\sigma}^\Sigma = 1_\Sigma$.

Algebra Lindenbaum-Tarski a Logicii Propoziționale Clasice

Notăție

În cazul în care $\Sigma = \emptyset$:

- relația de echivalență \sim_\emptyset se notează, simplu, \sim , și are următoarea definiție: pentru orice $\varphi, \psi \in E$,

$$\varphi \sim \psi \text{ ddacă } \vdash \varphi \leftrightarrow \psi;$$

- clasele de echivalență ale lui \sim , $\hat{\varphi}^\emptyset$ ($\varphi \in E$), se notează $\hat{\varphi}$;
- relația de ordine \leq_\emptyset se notează \leq ;
- operațiile \vee_\emptyset , \wedge_\emptyset , \neg^\emptyset , 0_\emptyset și 1_\emptyset se notează, respectiv, \vee , \wedge , \neg , 0 și 1.

Definiție

\sim se numește *echivalență logică* sau *echivalență semantică* între enunțuri.

Algebra Boole $(E/\sim, \vee, \wedge, \leq, \neg, 0, 1)$ se numește *algebra Lindenbaum-Tarski asociată sistemului formal \mathcal{L}* .

Lema anterioară devine, în acest caz, o caracterizare a teoremelor formale:

Lemă

Pentru orice $\varphi \in E$, $\vdash \varphi$ ddacă $\hat{\varphi} = 1$.

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Interpretări (evaluări, semantici) pentru logica \mathcal{L}

Definiție (o interpretare e o funcție de la mulțimea variabilelor propoziționale la algebra Boole standard: $\mathcal{L}_2 = \{0, 1\}$, cu $0 < 1$, i. e. o funcție care dă valori de adevăr variabilelor propoziționale)

O *interpretare (evaluare, semantică)* a lui \mathcal{L} este o funcție oarecare $h : V \rightarrow \mathcal{L}_2$.

Propoziție (există o unică funcție care prelungește o interpretare la mulțimea tuturor enunțurilor și transformă conectorii logici primitivi în operații Booleene, aşadar calculează valorile de adevăr ale tuturor enunțurilor pornind de la cele ale variabilelor propoziționale)

Pentru orice interpretare $h : V \rightarrow \mathcal{L}_2$, există o unică funcție $\tilde{h} : E \rightarrow \mathcal{L}_2$ care satisface următoarele proprietăți:

- (a) $\tilde{h}(u) = h(u)$, pentru orice $u \in V$;
- (b) $\tilde{h}(\neg \varphi) = \overline{\tilde{h}(\varphi)}$, pentru orice $\varphi \in E$;
- (c) $\tilde{h}(\varphi \rightarrow \psi) = \tilde{h}(\varphi) \rightarrow \tilde{h}(\psi)$, pentru orice $\varphi, \psi \in E$.

Definiție

Funcția $\tilde{h} : E \rightarrow \mathcal{L}_2$ din propoziția anterioară se numește tot *interpretare*.

Observație

Condiția (a) din propoziția anterioară spune că $\tilde{h}|_{V=h}$, adică funcția \tilde{h} prelungește pe h la E .

În condițiile (b) și (c), în membrii stângi, în argumentele lui \tilde{h} , \neg și \rightarrow sunt conectorii logici primitivi, pe când, în membrii drepti, \neg și \rightarrow sunt operațiile de complementare și, respectiv, implicație ale algebrei Boole \mathcal{L}_2 . Așadar, putem spune că funcția \tilde{h} transformă conectorii logici în operații booleene în algebra Boole standard.

Vom păstra notația \tilde{h} pentru această unică funcție depinzând de interpretarea h .

Corolar (prelungirea unei interpretări la mulțimea enunțurilor transformă toți conectorii logici în operații booleene)

Pentru orice interpretare h și orice $\varphi, \psi \in E$, au loc:

- (d) $\tilde{h}(\varphi \vee \psi) = \tilde{h}(\varphi) \vee \tilde{h}(\psi)$
- (e) $\tilde{h}(\varphi \wedge \psi) = \tilde{h}(\varphi) \wedge \tilde{h}(\psi)$
- (f) $\tilde{h}(\varphi \leftrightarrow \psi) = \tilde{h}(\varphi) \leftrightarrow \tilde{h}(\psi)$

Satisfacere și mulțimi satisfiabile

Fie h o interpretare, φ un enunț și Σ o mulțime de enunțuri.

Definiție (enunțuri adevărate pentru anumite valori de adevăr atribuite variabilelor propoziționale din scrierea lor)

Spunem că φ este *adevărat* în interpretarea h sau că h *satisfacă* φ dacă $\tilde{h}(\varphi) = 1$.
 φ se zice *fals* în interpretarea h dacă $\tilde{h}(\varphi) = 0$.

Spunem că h *satisfacă* Σ sau că h este un *model* pentru Σ dacă h satisfacă toate elementele lui Σ .

Spunem că Σ *admite un model* sau că mulțimea Σ este *satisfiabilă* dacă există un model pentru Σ .

Spunem că φ *admete un model* sau că φ este *satisfiabil* dacă $\{\varphi\}$ este satisfiabilă.

Notație

Faptul h *satisfacă* enunțul φ se notează cu: $h \models \varphi$.

Faptul că h *satisfacă* mulțimea Σ de enunțuri se notează cu: $h \models \Sigma$.

Remarcă

Dacă h este model pentru Σ , atunci h este model pentru orice submulțime a lui Σ .

Vom vedea că orice interpretare satisface mulțimea T a teoremelor formale. Deocamdată, să observăm că:

Remarcă

Orice interpretare satisface \emptyset .

Într-adevăr, pentru orice interpretare h , avem:

$$h \models \emptyset \iff (\forall \varphi \in \emptyset) (\tilde{h}(\varphi) = 1) \iff \overbrace{(\forall \varphi) \left(\underbrace{\varphi \in \emptyset}_{\text{fals pentru orice } \varphi} \Rightarrow \tilde{h}(\varphi) = 1 \right)}^{\substack{\text{adevărat} \\ \text{adevărat pentru orice } \varphi}}.$$

Definiție (adevărurile semantice și deducția semantică)

Enunțul φ se zice *universal adevărat* dacă φ este adevărat în orice interpretare.

Enunțurile universal adevărate se mai numesc *adevărurile semantice* sau *tautologiile* lui \mathcal{L} .

Spunem că φ se deduce semantic din Σ sau că φ este o consecință semantică a lui Σ dacă φ este adevărat în orice interpretare care satisface pe Σ .

Notație

Faptul că φ este universal adevărat se notează cu: $\models \varphi$.

Faptul că φ se deduce semantic din Σ se notează cu: $\Sigma \models \varphi$.

Le fel cum adevărurile sintactice (i. e. teoremele formale) sunt exact enunțurile deductibile sintactice din \emptyset :

Remarcă (adevărurile semantice sunt exact enunțurile deductibile semantic din \emptyset)

$$\models \varphi \text{ ddacă } \emptyset \models \varphi.$$

Într-adevăr, conform definițiilor de mai sus, remarcii anterioare și faptului că o implicație cu antecedentul adevărat este adevărată ddacă și concluzia implicației este adevărată:

$$\emptyset \models \varphi \Leftrightarrow (\forall h : V \rightarrow \mathcal{L}_2) \left(\underbrace{h \models \emptyset}_{\text{adevărat}} \Rightarrow h \models \varphi \right) \Leftrightarrow (\forall h : V \rightarrow \mathcal{L}_2) (h \models \varphi) \Leftrightarrow \models \varphi.$$

Observație (valorile de adevăr atribuite enunțurilor de o interpretare se calculează pe baza valorilor de adevăr atribuite variabilelor propoziționale de acea interpretare, folosind proprietatea interpretării de a transforma conectorii logici în operații booleene)

Valoarea unei interpretări într-un anumit enunț, uneori numită *interpretarea acelui enunț*, este valoarea de adevăr 0 sau 1 care se obține atunci când se atribuie, prin acea interpretare, valori de adevăr din \mathcal{L}_2 tuturor variabilelor propoziționale care apar în acel enunț. Un enunț universal adevărat, i. e. un adevăr semantic, o tautologie, este un enunț a cărui valoare de adevăr este 1 pentru orice valori de adevăr atribuite variabilelor propoziționale care apar în acel enunț.

În cele ce urmează, vom vedea două rezultate deosebit de importante privind sistemul formal \mathcal{L} : **Teorema de completitudine** și o generalizare a ei, **Teorema de completitudine tare**, numită și **Teorema de completitudine extinsă**.

Teorema de completitudine a lui \mathcal{L} afirmă că adevărurile sintactice ale lui \mathcal{L} coincid cu adevărurile semantice ale lui \mathcal{L} , i. e. teoremele formale ale lui \mathcal{L} sunt exact enunțurile universal adevărate, tautologiile lui \mathcal{L} . **Teorema de completitudine tare pentru \mathcal{L}** afirmă că, în \mathcal{L} , consecințele sintactice ale unei mulțimi Σ de enunțuri coincid cu consecințele semantice ale lui Σ , i. e. enunțurile care se deduc sintactic din Σ sunt exact enunțurile care se deduc semantic din Σ .

Teoremă (Teorema de completitudine tare (extinsă) pentru \mathcal{L} (TCT))

Pentru orice enunț φ și orice multime de enunțuri Σ :

$$\Sigma \vdash \varphi \quad \text{dacă} \quad \Sigma \vDash \varphi.$$

Teoremă (Teorema de completitudine pentru \mathcal{L} (TC))

Pentru orice enunț φ ,

$$\vdash \varphi \quad \text{dacă} \quad \vDash \varphi.$$

Demonstrație: Se aplică **Teorema de completitudine tare** pentru $\Sigma = \emptyset$.

Notă

Uneori,

- implicația $\vdash \varphi \Rightarrow \vDash \varphi$ este numită *corectitudinea lui \mathcal{L}* ,
- iar implicația $\vdash \varphi \Leftarrow \vDash \varphi$ este numită *completitudinea lui \mathcal{L}* .

Dar, cel mai adesea, **echivalența** din teorema anterioară este numită *completitudinea lui \mathcal{L}* .

Corolar (noncontradicția lui \mathcal{L} (principiul noncontradicției))

Niciun enunț φ nu satisfacă și $\vdash \varphi$, și $\vdash \neg \varphi$.

Propoziție

Algebra Lindenbaum–Tarski a logicii propoziționale clasice, E/\sim , este netrivială.

Propoziție

Pentru orice interpretare $h : V \rightarrow \mathcal{L}_2$ și orice $\Sigma \subseteq E$ a. i. $h \models \Sigma$, există un unic morfism boolean $\tilde{h}^\Sigma : E/\sim_\Sigma \rightarrow \mathcal{L}_2$ care face următoarea diagramă comutativă, anume morfismul definit prin: pentru orice $\varphi \in E$, $\tilde{h}^\Sigma(\hat{\varphi}^\Sigma) := \tilde{h}(\varphi)$:

$$\begin{array}{ccccc} V & \xhookrightarrow{\quad} & E & \xrightarrow{P_\Sigma} & E/\sim_\Sigma \\ & \searrow h & \downarrow \tilde{h} & \nearrow \tilde{h}^\Sigma & \\ & & \mathcal{L}_2 & & \end{array}$$

Corolar (din propoziția precedentă pentru $\Sigma = \emptyset$)

Pentru orice interpretare $h : V \rightarrow \mathcal{L}_2$, există un unic morfism boolean $\tilde{h} : E/\sim \rightarrow \mathcal{L}_2$ care face următoarea diagramă comutativă, anume morfismul boolean definit prin: pentru orice $\varphi \in E$, $\tilde{h}(\hat{\varphi}) := \tilde{h}(\varphi)$:

$$\begin{array}{ccccc} V & \xhookrightarrow{\quad} & E & \xrightarrow{P} & E/\sim \\ & \searrow h & \downarrow \tilde{h} & \nearrow \tilde{h} & \\ & & \mathcal{L}_2 & & \end{array}$$

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Definiție

O mulțime Σ de enunțuri se numește *sistem deductiv* dacă este închisă la deducții, i. e., pentru orice $\varphi \in E$, are loc:

$$\Sigma \vdash \varphi \Rightarrow \varphi \in \Sigma, \text{ adică:}$$

$$\{\varphi \in E \mid \Sigma \vdash \varphi\} \subseteq \Sigma.$$

Exemplu

Mulțimea E a tuturor enunțurilor este sistem deductiv.

Propoziție

Mulțimea teoremelor formale este cel mai mic sistem deductiv (în raport cu incluziunea, i.e. în posetul $(\mathcal{P}(E), \subseteq)$).

Spunem că o mulțime Σ de enunțuri este *închisă la modus ponens* dacă, pentru orice enunțuri φ, ψ , dacă $\psi, \psi \rightarrow \varphi \in \Sigma$, atunci $\varphi \in \Sigma$.

Propoziție (caracterizare pentru sistemele deductive)

Pentru orice $\Sigma \subseteq E$, sunt echivalente:

- ① Σ este sistem deductiv;
- ② Σ include mulțimea axiomelor și este închisă la modus ponens;
- ③ Σ include mulțimea teoremelor formale și este închisă la modus ponens.

Mulțimea sistemelor deductive este familie Moore

Propoziție

*Intersecția oricărei familii de sisteme deductive este sistem deductiv, i. e. mulțimea sistemelor deductive este un **sistem de închidere**.*

Notație

Notăm cu $D : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$ operatorul de închidere asociat sistemului de închidere format din sistemele deductive.

Corolar

Pentru orice mulțime Σ de enunțuri, $D(\Sigma)$ este cel mai mic sistem deductiv care include pe Σ , anume intersecția tuturor sistemelor deductive care includ pe Σ .

Definiție

Pentru orice $\Sigma \subseteq E$, $D(\Sigma)$ se numește *sistemul deductiv generat de Σ* .

Remarcă

Conform definiției unui operator de închidere, pentru orice $\Sigma, \Delta \in \mathcal{P}(E)$:

- ① $\Sigma \subseteq D(\Sigma)$ (D este **extensiv**);
- ② $\Sigma \subseteq \Delta$ implică $D(\Sigma) \subseteq D(\Delta)$ (D este **crescător**);
- ③ $D(D(\Sigma)) = D(\Sigma)$ (D este **idempotent**).

Propoziție (sistemul deductiv generat de o mulțime Σ de enunțuri este mulțimea consecințelor sintactice ale lui Σ)

Pentru orice mulțime Σ de enunțuri:

$$D(\Sigma) = \{\varphi \in E \mid \Sigma \vdash \varphi\}.$$

Corolar

Operatorul de închidere D este finitar, adică, oricare ar fi $\Sigma \subseteq E$, are loc:

$$D(\Sigma) = \bigcup \{D(\Gamma) \mid \Gamma \subseteq \Sigma, |\Gamma| < \aleph_0\}.$$

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 **Mulțimi consistente**
- 9 Rezoluția în calculul propozițional clasic

Mulțimi consistente (i. e. necontradictorii)

Definiție

Fie Σ o mulțime de enunțuri.

- Σ se zice *inconsistentă* dacă $\Sigma \vdash \varphi$ pentru orice $\varphi \in E$ (i. e. dacă orice enunț este consecință sintactică a lui Σ);
- Σ se zice *consistentă* dacă Σ nu este inconsistentă.

Exemplu

Mulțimea E a tuturor enunțurilor este inconsistentă.

Remarcă

Orice submulțime a unei mulțimi consistente este consistentă.

Prin urmare, orice mulțime care include o mulțime inconsistentă este inconsistentă.

Remarcă

Mulțimea T a teoremelor formale este consistentă.

Într-adevăr, conform unei propoziții de mai sus, T este sistem deductiv, deci este egală cu mulțimea enunțurilor φ cu $T \vdash \varphi$, iar $T \subseteq E$, conform **principiului noncontradicției**.

Exemplu (consecință a celor două remarcă precedente)

\emptyset și Ax sunt mulțimi consistente.

Propoziție

Sistemul deductiv generat de o mulțime consistentă este o mulțime consistentă.

Propoziție (mulțimile consistente sunt mulțimile de enunțuri din care nu se deduc contradicții)

Pentru orice $\Sigma \subseteq E$, sunt echivalente:

- ① Σ este inconsistentă;
- ② există $\varphi \in E$, astfel încât $\Sigma \vdash \varphi \wedge \neg\varphi$;
- ③ există $\varphi \in E$, astfel încât $\Sigma \vdash \varphi$ și $\Sigma \vdash \neg\varphi$;
- ④ există $\varphi \in E$, astfel încât $\Sigma \vdash \neg(\varphi \rightarrow \varphi)$;
- ⑤ pentru orice $\varphi \in E$, $\Sigma \vdash \neg(\varphi \rightarrow \varphi)$.

Corolar

Fie $\Sigma \subseteq E$ și $\varphi \in E$. Atunci:

- ① $\Sigma \cup \{\varphi\}$ este inconsistentă dacă $\Sigma \vdash \neg\varphi$;
- ② $\Sigma \cup \{\neg\varphi\}$ este inconsistentă dacă $\Sigma \vdash \varphi$.

Propoziție

Pentru orice $\Sigma \subseteq E$, are loc echivalența: Σ e consistentă dacă algebra Boole E/\sim_Σ e netrivială.

Definiție (mulțimi consistente maximale)

Un element maximal al mulțimii mulțimilor consistente raportat la incluziune se numește *mulțime consistentă maximală*.

Propoziție

Orice mulțime consistentă este inclusă într-o mulțime consistentă maximală.

Corolar

Există mulțimi consistente maximale.

Demonstrație: Aplicăm propoziția anterioară, de exemplu, pentru mulțimea consistentă \emptyset .

Propoziție

Dacă Σ este o mulțime consistentă maximală, atunci:

- ① Σ este sistem deductiv;
- ② pentru orice $\varphi, \psi \in E$, avem: $\varphi \vee \psi \in \Sigma$ dacă $\begin{cases} \varphi \in \Sigma \text{ sau} \\ \psi \in \Sigma; \end{cases}$
- ③ oricare ar fi $\varphi \in E$, are loc: $\varphi \in \Sigma$ dacă $\neg \varphi \notin \Sigma$;
- ④ pentru orice $\varphi, \psi \in E$: $\varphi \rightarrow \psi \in \Sigma$ dacă $\neg \varphi \vee \psi \in \Sigma$ dacă $\begin{cases} \neg \varphi \in \Sigma \text{ sau} \\ \psi \in \Sigma. \end{cases}$

Mulțimile consistente sunt exact mulțimile satisfiabile

Remarcă

T (și, aşadar, orice submulțime a lui T) admite ca model orice interpretare. Într-adevăr, pentru orice $\varphi \in T$ și orice interpretare h , avem $\vdash \varphi$, aşadar $\models \varphi$ conform (TC), prin urmare $\tilde{h}(\varphi) = 1$.

Propoziție

O mulțime de enunțuri e satisfiabilă dacă e consistentă.

Notă

A se vedea, în cartea de G. Georgescu și A. Iorgulescu din bibliografia din breviarul cursului de logică matematică și computațională, cum se obține **Teorema de completitudine tare** din **Teorema de completitudine**, folosind faptul că orice mulțime consistentă e satisfiabilă.

- 1 Ce este logica matematică? Care sunt dimensiunile unui sistem logic?
- 2 Sintaxa Sistemului Formal al Logicii Propoziționale Clasice
- 3 Proprietăți Sintactice ale Logicii Propoziționale Clasice
- 4 Mnemonic din Sintaxa Logicii Propoziționale Clasice
- 5 Algebra Lindenbaum–Tarski a Logicii Propoziționale Clasice
- 6 Semantica Logicii Propoziționale Clasice
- 7 Sisteme deductive
- 8 Mulțimi consistente
- 9 Rezoluția în calculul propozițional clasic

Rezoluția în calculul propozițional clasic

Pentru această secțiune a cursului, precum și pentru rezoluția în calculul cu predicate, care stă la baza implementării limbajului Prolog, se poate consulta carte următoare:

-  G. Metakides, A. Nerode, *Principles of Logic and Logic Programming*
 - traducere de A. Florea, B. Boldur: *Principii de Logică și Programare Logică*, Editura Tehnică, București, 1998.

Definiție (FNC și FND)

- Un *literal* este o variabilă propozițională sau negația unei variabile propoziționale:

$$p \text{ sau } \neg p, \text{ cu } p \in V.$$

- O *clauză* este o disjuncție de literali.

Orice clauză se identifică cu mulțimea literalilor care o compun.

- Un enunț $\varphi(\in E)$ este în formă normală conjunctivă (sau este o formă normală conjunctivă) (FNC) dacă φ este o conjuncție de clauze, i. e. o conjuncție de disjuncții de literali.

Orice FNC se identifică cu mulțimea clauzelor care o compun.

- Un enunț $\varphi(\in E)$ este în formă normală disjunctivă (sau este o formă normală disjunctivă) (FND) dacă φ este o disjuncție de conjuncții de literali.

Observație

Întrucât toate enunțurile au lungime finită (i. e. sunt siruri finite de simboluri primitive), conjuncțiile și disjuncțiile la care face referire definiția de mai sus sunt finite.

Relația de echivalență semantică: $\sim \sim_\emptyset \in \text{Eq}(E)$: relația de echivalență pe E care servește la construirea algebrei Lindenbaum–Tarski a logicii propoziționale clasice: pentru orice $\varphi, \psi \in E$,

$$\varphi \sim \psi \text{ dacă } \vdash \varphi \leftrightarrow \psi.$$

Folosind definiția lui \sim , **Teorema de completitudine**, definiția adevărurilor semantice, compatibilitatea oricărei interpretări cu conectorii logici și o proprietate valabilă în orice algebră Boole, obținem:

Remarcă (două enunțuri sunt echivalente semantic dacă au aceeași valoare în orice interpretare)

Oricare ar fi $\varphi, \psi \in E$, au loc următoarele echivalențe:

$$\varphi \sim \psi \Leftrightarrow \vdash \varphi \leftrightarrow \psi \Leftrightarrow \models \varphi \leftrightarrow \psi \Leftrightarrow (\forall h : V \rightarrow L_2) (\tilde{h}(\varphi \leftrightarrow \psi) = 1) \Leftrightarrow (\forall h : V \rightarrow L_2) (\tilde{h}(\varphi) \leftrightarrow \tilde{h}(\psi) = 1) \Leftrightarrow (\forall h : V \rightarrow L_2) (\tilde{h}(\varphi) = \tilde{h}(\psi)).$$

Așadar:

Remarcă

Dacă $\varphi, \psi \in E$ astfel încât $\varphi \sim \psi$, atunci: φ e satisfiabil dacă ψ e satisfiabil.

Punerea unui enunț în FNC (sau FND)

Propoziție (existența FNC și FND pentru orice enunț)

Oricare ar fi $\varphi \in E$, există o FNC $\psi \in E$ și o FND $\chi \in E$ (care nu sunt unice), astfel încât $\varphi \sim \psi \sim \chi$.

Remarcă

Oricare ar fi $\varphi \in E$, putem determina o FNC (sau FND) $\psi \in E$ cu $\varphi \sim \gamma$, folosind un tabel semantic pentru φ , sau folosind următoarele proprietăți imediate, valabile pentru orice $\alpha, \beta, \gamma \in E$:

- **înlocuirea implicațiilor și echivalențelor:**

$$\alpha \rightarrow \beta \sim \neg \alpha \vee \beta \text{ și } \alpha \leftrightarrow \beta \sim (\neg \alpha \vee \beta) \wedge (\neg \beta \vee \alpha)$$

- **idempotența lui \vee și \wedge :**

$$\alpha \vee \alpha \sim \alpha \wedge \alpha \sim \alpha$$

- **comutativitatea lui \vee și \wedge :**

$$\alpha \vee \beta \sim \beta \vee \alpha \text{ și } \alpha \wedge \beta \sim \beta \wedge \alpha$$

Remarcă (continuare)

- **asociativitatea lui \vee și \wedge :**

$$(\alpha \vee \beta) \vee \gamma \sim \alpha \vee (\beta \vee \gamma) \text{ și } (\alpha \wedge \beta) \wedge \gamma \sim \alpha \wedge (\beta \wedge \gamma)$$

- **principiul dublei negații:**

$$\neg \neg \alpha \sim \alpha$$

- **legile lui de Morgan:**

$$\neg(\alpha \vee \beta) \sim \neg \alpha \wedge \neg \beta \text{ și } \neg(\alpha \wedge \beta) \sim \neg \alpha \vee \neg \beta$$

- **absorbția:**

$$\alpha \vee (\alpha \wedge \beta) \sim \alpha \wedge (\alpha \vee \beta) \sim \alpha$$

- **legile de distributivitate:**

$$\alpha \vee (\beta \wedge \gamma) \sim (\alpha \vee \beta) \wedge (\alpha \vee \gamma) \text{ și } \alpha \wedge (\beta \vee \gamma) \sim (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$$

- **proprietățile:**

$$\alpha \vee (\beta \wedge \neg \beta) \sim \alpha \vee (\beta \wedge \neg \beta \wedge \gamma) \sim \alpha \wedge (\beta \vee \neg \beta) \sim \alpha \wedge (\beta \vee \neg \beta \vee \gamma) \sim \alpha$$

Observație (echivalență semantică (\sim) versus egalitatea de enunțuri)

Fie $\varphi \in E$. Atunci $\varphi \sim \neg\neg\varphi$, dar $\varphi \neq \neg\neg\varphi$. De exemplu, enunțurile “*Plouă.*” și “*Nu e adevărat că nu plouă.*” sunt echivalente semantic (adică au același sens, același înțeles), dar nu coincid (sunt fraze diferite, nici nu sunt compuse din același număr de cuvinte).

- Următoarea notație este definită, recursiv, pe întreaga mulțime E :

Notăție (mulțimea variabilelor propoziționale care apar într-un enunț φ se notează $V(\varphi)$)

Pentru orice $p \in V$ și orice $\varphi, \psi \in E$, notăm:

- ① $V(p) = \{p\}$
- ② $V(\neg\varphi) = V(\varphi)$
- ③ $V(\varphi \rightarrow \psi) = V(\varphi) \cup V(\psi)$

- Amintesc că, într-un tabel semantic pentru un enunț φ , ne interesează variabilele propoziționale care apar în φ , adică elementele lui $V(\varphi)$.

A se vedea, la seminar, metoda prin care un enunț φ poate fi pus în FNC folosind un tabel semantic pentru φ .

Definiție (formă clauzală)

Fie $\varphi \in E$ și $M \subseteq E$, astfel încât M este finită.

- O *formă clauzală* pentru φ este o FNC (i. e. o mulțime de clauze) ψ cu $\psi \sim \varphi$.
- O *formă clauzală* pentru M este o reuniune de forme clauzale pentru elementele lui M .

Remarcă

Orice mulțime finită de enunțuri poate fi pusă într-o formă clauzală.

Propoziție

O mulțime de enunțuri e satisfiabilă dacă orice submulțime finită a sa este satisfiabilă.

Schița demonstrației: Fie $\Gamma \subseteq E$.

" \Rightarrow :" Orice model pentru Γ este model pentru toate submulțimile finite ale lui Γ .

" \Leftarrow :" Fie $\varphi \in \Gamma$.

Ca în demonstrația propoziției următoare sau direct din faptul că Γ e satisfiabilă dacă e consistentă și o caracterizare a mulțimilor consistente, Γ e satisfiabilă dacă $\Gamma \setminus \{\varphi\} \not\vdash \neg\varphi$ dacă, pentru orice submulțime finită $\Lambda \subseteq \Gamma \setminus \{\varphi\}$, $\Lambda \not\vdash \neg\varphi$ conform Propoziției \star , (2).

Deducție semantică versus satisfiabilitate

Propoziție (conform căreia tehnica rezoluției de mai jos poate fi folosită pentru a demonstra teoreme formale sau deducții formale din ipoteze)

Fie $n \in \mathbb{N}^*$, $\varphi_1, \varphi_2, \dots, \varphi_n, \psi \in E$ și $\Gamma \subseteq E$.

(a) Atunci următoarele afirmații sunt echivalente:

- ① $\{\varphi_1, \dots, \varphi_n\} \models \psi$
- ② $\{\varphi_1, \dots, \varphi_n, \neg \psi\}$ nu e satisfiabilă
- ③ $\varphi_1 \wedge \dots \wedge \varphi_n \wedge \neg \psi$ nu e satisfiabil

(b) În plus, următoarele afirmații sunt echivalente:

- ① $\models \psi$
- ② $\neg \psi$ nu e satisfiabil

(c) Mai mult, următoarele afirmații sunt echivalente:

- ① $\Gamma \models \psi$
- ② $\Gamma \cup \{\neg \psi\}$ nu e satisfiabilă
- ③ $\neg \psi$ nu e satisfiabil, sau există $k \in \mathbb{N}^*$ și $\psi_1, \dots, \psi_k \in \Gamma$ astfel încât $\psi_1 \wedge \dots \wedge \psi_k \wedge \neg \psi$ nu e satisfiabil.

Deducre semantică versus satisfiabilitate

Demonstrație: Folosim definiția deducției semantice și a adevărurilor semantice, proprietatea interpretărilor de a transforma conectorii logici în operații booleene în \mathcal{L}_2 , precum și faptul că $\mathcal{L}_2 = \{0, 1\}$, cu $0 \neq 1$.

(a) $\{\varphi_1, \dots, \varphi_n\} \models \psi$ dacă, pentru orice $h : V \rightarrow \mathcal{L}_2$,

$h(\varphi_1) = \dots = h(\varphi_n) = 1 \Rightarrow h(\psi) = 1$ dacă nu există $h : V \rightarrow \mathcal{L}_2$ cu

$h(\varphi_1) = \dots = h(\varphi_n) = h(\neg\psi) = 1$ dacă $\{\varphi_1, \dots, \varphi_n, \neg\psi\}$ nu e satisfiabilă
dacă nu există $h : V \rightarrow \mathcal{L}_2$ cu $h(\varphi_1 \wedge \dots \wedge \varphi_n \wedge \neg\psi) = 1$ dacă

$\varphi_1 \wedge \dots \wedge \varphi_n \wedge \neg\psi$ nu e satisfiabil.

(b) $\models \psi$ dacă, pentru orice $h : V \rightarrow \mathcal{L}_2$, $h(\psi) = 1$ dacă nu există $h : V \rightarrow \mathcal{L}_2$ cu
 $h(\neg\psi) = 1$ dacă $\neg\psi$ nu e satisfiabil.

(c) $\Gamma \models \psi$ dacă, pentru orice $h : V \rightarrow \mathcal{L}_2$, $h \models \Gamma \Rightarrow h(\psi) = 1$ dacă nu există
 $h : V \rightarrow \mathcal{L}_2$ cu $h \models \Gamma \cup \{\neg\psi\}$ dacă $\Gamma \cup \{\neg\psi\}$ nu e satisfiabilă dacă $\neg\psi$ nu e
satisfiabil, sau există $k \in \mathbb{N}^*$ și $\psi_1, \dots, \psi_k \in \Gamma$ astfel încât $\{\psi_1, \dots, \psi_k, \neg\psi\}$ nu e
satisfiabilă, conform lemei anterioare, dacă $\neg\psi$ nu e satisfiabil, sau există $k \in \mathbb{N}^*$
și $\psi_1, \dots, \psi_k \in \Gamma$ astfel încât $\psi_1 \wedge \dots \wedge \psi_k \wedge \neg\psi$ nu e satisfiabil, conform
punctului (a).

Problema satisfiabilității

Remarcă

Formele clauzale pentru o mulțime finită de enunțuri $\{\varphi_1, \dots, \varphi_n\}$ sunt exact formele clauzale pentru enunțul $\varphi_1 \wedge \dots \wedge \varphi_n$.

Problemă

Fie dat un enunț φ în FNC, să se determine dacă φ e satisfiabil.

- O soluție computațională pentru problema de mai sus este **algoritmul Davis–Putnam**, bazat pe **rezoluție**.
- **Rezoluția propozițională** poate fi privită ca o regulă de deducție pentru calculul propozițional clasic.
- Utilizând **rezoluția**, se poate construi un demonstrator automat **corect și complet** pentru calculul propozițional clasic, fără alte teoreme formale și reguli de deducție, pentru că **regula rezoluției este echivalentă cu schemele de axiome (A_1) , (A_2) , (A_3) plus regula MP**.
- Limbajul de programare logică PROLOG este fundamentat pe rezoluția pentru calculul cu predicate clasic (care înglobează rezoluția propozițională).

Clauze și mulțimi de clauze

Definiție (și mnemonic)

- O *clauză* este o mulțime finită de literali ($\{L_1, \dots, L_n\}$, cu $n \in \mathbb{N}$ și $L_1, \dots, L_n \in V \cup \{\neg p \mid p \in V\}$).
- Clauza vidă** (i. e. clauza fără literali, clauza fără elemente) se notează cu \square (pentru a o deosebi de **mulțimea vidă de clauze**, \emptyset , în cele ce urmează).
- O clauză C se zice *trivială* dacă există $p \in V$ cu $p, \neg p \in C$.
- Orice clauză nevidă $C = \{L_1, \dots, L_n\}$ (cu $n \in \mathbb{N}^*$ și $L_1, \dots, L_n \in V \cup \{\neg p \mid p \in V\}$) se identifică cu enunțul în FND $\varphi = L_1 \vee \dots \vee L_n$. Clauza C se zice *satisfiabilă* dacă enunțul φ e satisfiabil.
- Clauza vidă (\square) e considerată **nesatisfiabilă (justificare)**: \square se identifică cu $\bigvee_{i \in \emptyset} L_i$; pentru orice $h : V \rightarrow L_2$, $\tilde{h}(\bigvee_{i \in \emptyset} L_i) = \bigvee_{i \in \emptyset} \tilde{h}(L_i) = 0 \neq 1$ în L_2 .
- Orice mulțime finită de clauze $M = \{C_1, \dots, C_k\}$ (cu $k \in \mathbb{N}$ și C_1, \dots, C_k clauze) se identifică cu $C_1 \wedge \dots \wedge C_k$, deci cu o FNC.
- O mulțime finită de clauze se zice *satisfiabilă* dacă toate clauzele din componența ei sunt satisfăcute de o aceeași interpretare (au un același model, au un model comun).

Satisfiabilitate pentru (mulțimi de) clauze

Remarcă

Sunt imediate următoarele proprietăți:

- o mulțime finită de clauze este satisfiabilă dacă FNC asociată ei este satisfiabilă;
- \emptyset (mulțimea vidă de clauze) este satisfiabilă, chiar este satisfăcută de orice interpretare;
- orice clauză trivială este satisfiabilă, chiar este satisfăcută de orice interpretare;
- orice mulțime finită de clauze triviale este satisfiabilă, chiar este satisfăcută de orice interpretare.

Pentru orice clauze C, D , dacă $p \in V$ astfel încât $p, \neg p \notin C$ și $p, \neg p \notin D$, atunci:

$$\frac{C \cup \{p\}, D \cup \{\neg p\}}{C \cup D}, \text{ adică } \frac{(C \vee p) \wedge (D \vee \neg p)}{C \vee D}$$

Propoziție

Fie C și D clauze, iar S, T și U mulțimi finite de clauze. Atunci:

- ① dacă C e satisfiabilă și $C \subseteq D$, atunci D e satisfiabilă; prin urmare, dacă D e nesatisfiabilă și $C \subseteq D$, atunci C e nesatisfiabilă;
- ② $C \cup D$ e satisfiabilă dacă C e satisfiabilă sau D e satisfiabilă;
- ③ dacă $p \in V \setminus V(C)$, atunci $C \cup \{p\}$ și $C \cup \{\neg p\}$ sunt satisfiabile;
- ④ dacă S e nesatisfiabilă și $S \subseteq T$, atunci T e nesatisfiabilă; prin urmare, dacă T e satisfiabilă și $S \subseteq T$, atunci S e satisfiabilă;
- ⑤ dacă U e satisfiabilă și există $p \in V \setminus V(U)$, $G \in S$ și $H \in T$ astfel încât $p \in G \setminus H$ și $\neg p \in H \setminus G$, atunci $U \cup S$ și $U \cup T$ sunt satisfiabile;
- ⑥ dacă $p \in V$ astfel încât $p, \neg p \notin C$ și $p, \neg p \notin D$, iar mulțimea de clauze $\{C \cup \{p\}, D \cup \{\neg p\}\}$ e satisfiabilă, atunci $C \cup D$ e satisfiabilă (**regula rezoluției**).

Definiție (derivări prin rezoluție)

Fie o mulțime finită de clauze $\{D_1, \dots, D_k\}$ și φ enunțul în FNC corespunzător acestei mulțimi de clauze:

$$\varphi = D_1 \wedge \dots \wedge D_k.$$

Dacă $i, j \in \overline{1, k}$ a. î. $i \neq j$ și există $p \in V$ cu $p \in D_i$ și $\neg p \in D_j$, atunci mulțimea de clauze $R := \{(D_i \setminus \{p\}) \cup (D_j \setminus \{\neg p\})\} \cup \{D_t \mid t \in \overline{1, k} \setminus \{i, j\}\}$ sau o FNC corespunzătoare ei, de exemplu enunțul

$$((D_i \setminus \{p\}) \vee (D_j \setminus \{\neg p\})) \wedge \bigwedge_{t \in \overline{1, k} \setminus \{i, j\}} D_t,$$

se numește *rezolvent* al enunțului φ sau al mulțimii de clauze $\{D_1, \dots, D_k\}$.

Deduția

$$\frac{D_1, \dots, D_k}{R}$$

se numește *derivare prin rezoluție* a lui φ sau a mulțimii $\{D_1, \dots, D_k\}$.

Vom numi orice succesiune de derivări prin rezoluție tot *derivare prin rezoluție*. O succesiune de derivări prin rezoluție care începe cu o FNC/mulțime de clauze μ și se termină cu o FNC/mulțime de clauze ν se numește *derivare prin rezoluție a lui ν din μ* .

Notă

Rezoluția este o metodă de verificare a satisfiabilității pentru mulțimi (finite) de enunțuri în formă clauzală.

Notă

Aplicarea regulii rezoluției simultan pentru două variabile diferite este **greșită**.

Remarcă

- Dacă într-o derivare prin rezoluție a unei mulțimi finite M de enunțuri în **formă clauzală** apare \square , atunci M nu e satisfiabilă.
- În schimb, o derivare prin rezoluție a lui M în care nu apare \square **nu arată** că M ar fi satisfiabilă.
- Pentru a arăta că o mulțime finită M de enunțuri (în formă clauzală) este satisfiabilă, putem găsi un model pentru M sau putem aplica **algoritmul Davis–Putnam**, care este echivalent cu obținerea tuturor derivărilor posibile prin rezoluție ale formei clauzale a lui M .

Algoritmul Davis–Putnam (abreviat DP)

INPUT: mulțime finită și nevidă S de clauze netriviale;

$S_1 := S; i := 1;$

PASUL 1: luăm o $v_i \in V(S_i)$;

$$T_i^0 := \{C \in S_i \mid \neg v_i \in C\};$$

$$T_i^1 := \{C \in S_i \mid v_i \in C\};$$

$$T_i := T_i^0 \cup T_i^1;$$

PASUL 2: dacă $T_i^0 \neq \emptyset$ și $T_i^1 \neq \emptyset$,

atunci $U_i := \{(C_0 \setminus \{\neg v_i\}) \cup (C_1 \setminus \{v_i\}) \mid C_0 \in T_i^0, C_1 \in T_i^1\}$;

altfel $U_i := \emptyset$;

PASUL 3: $S_{i+1} := (S_i \setminus T_i) \cup U_i$;

$S_{i+1} := S_{i+1} \setminus \{C \in S_{i+1} \mid (\exists p \in V) (p, \neg p \in C)\}$

(eliminăm din S_{i+1} clauzele triviale);

PASUL 4: dacă $S_{i+1} = \emptyset$,

atunci OUTPUT: S e satisfiabilă;

altfel, dacă $\square \in S_{i+1}$,

atunci OUTPUT: S nu e satisfiabilă;

altfel $i := i + 1$ și mergi la PASUL 1.

Propoziție (terminarea algoritmului DP)

Algoritmul DP se termină după cel mult $|V(S)|$ execuții ale pașilor 1 – 4, cu $S_{i+1} = \emptyset$ sau $\square \in S_{i+1}$.

Demonstrație: Cu notațiile din algoritmul DP, are loc, pentru fiecare i :

$$V(S_{i+1}) \subsetneq V(S_i),$$

așadar algoritmul DP se termină după cel mult $|V(S)|$ iterații.

Propoziție

Fie S o mulțime finită și nevidă de clauze.

Dacă S e satisfiabilă, atunci orice rezolvent al lui S e satisfiabil.

Demonstrație: Fie φ enunțul în FNC corespunzător lui S și $\rho \in E$ un rezolvent al lui φ . Atunci, pentru un enunț γ în FNC, o variabilă propozițională p și două clauze C și D cu $p \notin V(C) \cup V(D)$, avem:

$$\varphi = (C \vee p) \wedge (D \vee \neg p) \wedge \gamma \quad \text{și} \quad \rho = (C \vee D) \wedge \gamma,$$

așadar, pentru orice interpretare h , avem:

$$\begin{aligned}\tilde{h}(\varphi) &= (\tilde{h}(C) \vee h(p)) \wedge (\tilde{h}(D) \vee \overline{h(p)}) \wedge \tilde{h}(\gamma) = \\ ((\tilde{h}(C) \wedge \tilde{h}(D)) \vee (\tilde{h}(C) \wedge \overline{h(p)}) \vee (\tilde{h}(D) \wedge h(p)) \vee (h(p) \wedge \overline{h(p)})) \wedge \tilde{h}(\gamma) = \\ ((\tilde{h}(C) \wedge \tilde{h}(D)) \vee (\tilde{h}(C) \wedge \overline{h(p)}) \vee (\tilde{h}(D) \wedge h(p))) \wedge \tilde{h}(\gamma) &\leq ((\tilde{h}(C) \vee \tilde{h}(D)) \wedge \tilde{h}(\gamma)) = \tilde{\rho},\end{aligned}$$

prin urmare orice interpretare care satisfacă pe φ satisfacă și pe ρ ; în particular, dacă φ e satisfiabil, atunci ρ e satisfiabil.

Corolar

Fie S o mulțime finită și nevidă de clauze.

Dacă S e satisfiabilă, atunci algoritmul DP, aplicat lui S , se termină cu $S_{i+1} = \emptyset$.

Teoremă

Fie S o mulțime finită și nevidă de clauze. Atunci următoarele afirmații sunt echivalente:

- ① S nu e satisfiabilă;
- ② există o derivare prin rezoluție a lui \square (sau a unei mulțimi de clauze conținând pe \square) din S .

Demonstrație: ② \Rightarrow ①: A se observă că \square se poate obține prin rezoluție numai din două clauze de forma $\{p\}$, $\{\neg p\}$, cu $p \in V$, și orice mulțime formată din două astfel de clauze e nesatisfiabilă.

\square nu e satisfiabilă, așadar, conform propoziției anterioare, dacă există o derivare prin rezoluție a lui \square din S , atunci S e nesatisfiabilă.

① \Rightarrow ②: **Schița demonstrației**, după articolul:

-  J. Gallier, The Completeness of Propositional Resolution: a Simple and Constructive Proof, *Logical Methods in Computer Science* 2(5:3) (2006), 1–7.

Presupunem că S e nesatisfiabilă.

Pentru orice clauză D , notăm $c(D) := |D| - 1$ (numărul de literali din compoziția lui D minus o unitate). Pentru orice mulțime finită și nevidă de cluze $M = \{D_1, \dots, D_k\}$, notăm $c(M) := c(D_1) + \dots + c(D_k)$.
Procedăm prin inducție după $c(S)$.

Pasul de verificare: Dacă $c(S) = 0$, atunci $c(D) = 0$ pentru fiecare clauză D a lui S , aşadar fiecare clauză a lui S este un literal. Cum S e nesatisfiabilă, există $p \in V$ a. î. p și $\neg p$ sunt cluze ale lui S , prin urmare din S se poate deriva prin rezoluție o mulțime de cluze care conține pe \square .

Pasul de inducție: Presupunem că $c(S) > 0$ și că orice mulțime M de cluze cu $c(M) < c(S)$ are o derivare prin rezoluție care se termină cu o mulțime conținând pe \square .

Cum $c(S) > 0$, există o clauză D a lui S cu $c(D) > 0$, prin urmare $D = C \vee L$ pentru un literal L și o clauză nevidă C în care nu apare variabila propozițională din L , așa că avem $c(C) < c(D)$.

Fie φ enunțul în FNC corespunzător lui S , care e nesatisfiabil conform ipotezei acestei implicații. Atunci, pentru un enunț γ în FNC, avem:

$$\varphi \sim (C \vee L) \wedge \gamma \sim (C \wedge \gamma) \vee (L \wedge \gamma),$$

prin urmare enunțurile $C \wedge \gamma$ și $L \wedge \gamma$ sunt nesatisfiabile. Observăm că au loc: $c(C \wedge \gamma) < c(\varphi)$ și $c(L \wedge \gamma) < c(\varphi)$, aşadar, conform ipotezei de inducție, fiecare dintre enunțurile $C \wedge \gamma$ și $L \wedge \gamma$ admite o derivare prin rezoluție în care apare \square .

Să considerăm o astfel de derivare pentru $C \wedge \gamma$ și să înlocuim pe C cu $C \vee L = D$, transformând enunțul $C \wedge \gamma$ în $(C \vee L) \wedge \gamma \sim \varphi$, și modificând corespunzător această derivare, astfel că, la finalul ei:

- fie apare tot \square , caz în care am obținut deja o derivare a lui φ care conduce la \square ,
- fie apare clauza L în locul lui \square , caz în care procedăm astfel:

înănd seama de faptul că $\varphi \sim D \wedge \gamma \sim D \wedge \gamma \wedge \gamma$, la fiecare pas al derivării prin rezoluție a lui φ obținute prin modificarea de mai sus, la mulțimea curentă de clauze adăugăm o mulțime de clauze corespunzând lui γ , astfel că la finalul acestei noi derivări vom avea o mulțime de clauze corespunzătoare enunțului $L \wedge \gamma$, din care, conform celor de mai sus, se poate deriva prin rezoluție o mulțime de clauze conținând pe \square . Punând "cap la cap" aceste derivări, obținem o derivare prin rezoluție din φ (echivalent, din S) a unei mulțimi de clauze conținând pe \square .

Corolar (Teorema Davis–Putnam)

Algoritmul DP este corect și complet.

Notație

Dacă $\Gamma \subseteq E$ și $\varphi \in E$, atunci notăm cu $\Gamma \vdash_R \varphi$ faptul că există o derivare prin rezoluție a lui \square dintr-o formă clauzală a lui $\Gamma \cup \{\neg \varphi\}$.

Rezoluția propozițională \iff sistemul Hilbert

Corolar

Pentru orice $\Gamma \subseteq E$ și orice $\varphi \in E$, următoarele afirmații sunt echivalente:

- ① $\Gamma \models \varphi$;
- ② $\Gamma \vdash_R \varphi$.

Remarcă

Conform corolarului anterior și (TCT), **regula rezoluției** este corectă și completă pentru calculul propozițional clasic, adică deducția pe baza **regulii rezoluției** este echivalentă cu deducția pe baza axiomelor (A_1) , (A_2) , (A_3) și a regulii de deducție **MP**, i.e. cu **sistemul Hilbert**.

Așadar folosind **regula rezoluției** putem realiza o prezentare echivalentă pentru logica propozițională clasică.

PROGRAMARE LOGICĂ

Cursurile VI, VII și VIII

Claudia MUREŞAN
cmuresan@fmi.unibuc.ro, c.muresan@yahoo.com

Universitatea din Bucureşti
Facultatea de Matematică și Informatică
Bucureşti

2019–2020, Semestrul II

Cuprinsul acestui curs

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasic
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasice
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

1 Recapitulare Logica Clasică a Predicatelor

• Calculul clasic cu predicate

- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasice
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

Ce este un predicat?

- **Predicatelor** se mai numesc **propoziții cu variabile**.
- **Exemplu de predicat:** “ x este un număr prim” este un predicat cu variabila x ; acest predicat nu are o valoare de adevăr; pentru a obține din el un enunț cu valoare de adevăr, trebuie să-i dăm valori variabilei x .
- Variabilei x i se indică un domeniu al valorilor posibile, de exemplu \mathbb{N} . Se dorește ca, prin înlocuirea lui x din acest predicat cu o valoare arbitrară din acest domeniu, să se obțină o propoziție adevărată sau falsă.
- Înlocuind în acest predicat pe $x := 2 \in \mathbb{N}$, se obține propoziția adevărată “ 2 este un număr prim”, iar înlocuind $x := 4 \in \mathbb{N}$, se obține propoziția falsă “ 4 este un număr prim”.
- Dacă desemnăm pe \mathbb{N} ca domeniu al valorilor pentru variabila x din predicatul de mai sus, atunci putem să aplicăm un cuantificator acestei variabile, obținând, astfel, tot un enunț cu valoare de adevăr: propoziția $(\forall x \in \mathbb{N}) (x \text{ este un număr prim})$ este falsă, în timp ce propoziția $(\exists x \in \mathbb{N}) (x \text{ este un număr prim})$ este adevărată. De fapt, în enunțuri cuantificate, vom considera domeniul valorilor variabilelor stabilit de la început, și nu-l vom mai preciza după variabilele cuantificate (“ $\in \mathbb{N}$ ”).

În ce fel de structuri algebrice pot lua valori variabilele?

- Așadar, pentru a exprima matematic modul de lucru cu predicate, avem nevoie nu numai de noțiunea de propoziție cu sau fără variabile și de conectori logici, ci și de un domeniu al valorilor pentru variabilele care apar în predicate (i. e. în propozițiile cu variabile).
- Pentru a descrie sistemul formal al calculului cu predicate clasic, vom avea nevoie de noțiunea de **structură de ordinul I**, reprezentând un anumit gen de structuri algebrice. Variabilele vor fi considerate ca luând valori în diverse **structuri de ordinul I**, și **clasa structurilor de ordinul I de un anumit tip** va avea asociată propria ei logică clasică cu predicate (îi vom asocia un limbaj, apoi un sistem logic bazat pe acel limbaj).
- Intuitiv, **structurile de ordinul I** sunt structuri algebrice care posedă o mulțime suport și operații, relații și constante (operații zeroare) pe această mulțime suport, i. e. operații și relații care acționează (numai) asupra elementelor mulțimii suport.
- Când, într-o structură algebrică, există operații sau relații care acționează asupra submulțimilor mulțimii suport, i. e. asupra unor mulțimi de elemente din mulțimea suport, atunci spunem că structura respectivă este o **structură de ordinul II**. În același mod (referindu-ne la mulțimi de mulțimi de elemente și. a. m. d.) pot fi definite **structurile de ordinul III, IV etc..**

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasic
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

Structuri algebrice cu operații și relații

Definiție

O structură de ordinul I este o structură de forma

$$\mathcal{A} = (A, (f_i)_{i \in I}, (R_j)_{j \in J}, (c_k)_{k \in K}),$$

unde:

- A este o mulțime nevidă, numită *universul structurii* \mathcal{A}
- I, J, K sunt mulțimi oarecare de indici (care pot fi și vide)
- pentru fiecare $i \in I$, există $n_i \in \mathbb{N}^*$, a. i. $f_i : A^{n_i} \rightarrow A$ (f_i este o operație n_i -ară pe A)
- pentru fiecare $j \in J$, există $m_j \in \mathbb{N}^*$, a. i. $R_j \subseteq A^{m_j}$ (R_j este o relație m_j -ară pe A)
- pentru fiecare $k \in K$, $c_k \in A$ (c_k este o constantă din A)

În general, operațiilor și relațiilor din componența lui \mathcal{A} li se atașează indicele \mathcal{A} , pentru a le deosebi de simbolurile de operații și relații din limbajul pe care îl vom construi în continuare; astfel, structura de ordinul I de mai sus se notează, de regulă, sub forma: $\mathcal{A} = (A, (f_i^{\mathcal{A}})_{i \in I}, (R_j^{\mathcal{A}})_{j \in J}, (c_k^{\mathcal{A}})_{k \in K})$.

Vom avea clase de structuri de același tip

Definiție

Tipul sau signatura structurii de ordinul I \mathcal{A} din definiția anterioară este tripletul de familii de numere naturale: $\tau := ((n_i)_{i \in I}; (m_j)_{j \in J}; (0)_{k \in K})$.

Orice structură de forma lui \mathcal{A} de mai sus se numește *structură de ordinul I de tipul (sau signatura) τ* .

Exemplu

- Orice poset nevid este o structură de ordinul I de forma $\mathcal{P} = (P, \leq)$, de tipul (signatura) $\tau_1 = (\emptyset; 2; \emptyset)$ (\leq este o relație binară). **Nu orice** structură de ordinul I de signatura τ_1 este un poset.
- Orice latice nevidă este o structură de ordinul I de forma $\mathcal{L} = (L, \vee, \wedge, \leq)$, de tipul (signatura) $\tau_2 = (2, 2; 2; \emptyset)$ (\vee și \wedge sunt operații binare (i. e. de aritate 2, i. e. cu câte două argumente), iar \leq este o relație binară). **Nu orice** structură de ordinul I de signatura τ_2 este o latice.
- Orice algebră Boole este o structură de ordinul I de forma $\mathcal{B} = (B, \vee, \wedge, \neg, \leq, 0, 1)$, de tipul (signatura) $\tau_3 = (2, 2, 1; 2; 0, 0)$ (\vee și \wedge sunt operații binare, \neg este o operație unară, \leq este o relație binară, iar 0 și 1 sunt constante (operații zeroare, i. e. de aritate zero, i. e. fără argumente)). **Nu orice** structură de ordinul I de signatura τ_3 este o algebră Boole.

Limbajul asociat unei signaturi

- Fiecarei signaturi τ a unei structuri de ordinul I (fiecarei clase de structuri de ordinul I de o anumită signură τ) i se asociază un limbaj, numit **limbaj de ordinul I** și notat, de obicei, cu \mathcal{L}_τ , în care pot fi exprimate proprietățile algebrice ale structurilor de ordinul I de signura τ .
- Să considerăm o signură $\tau := ((n_i)_{i \in I}; (m_j)_{j \in J}; (0)_{k \in K})$, pe care o fixăm.
- **Alfabetul limbajului de ordinul I** \mathcal{L}_τ este format din următoarele **simboluri primitive**:
 - ① o mulțime infinită de **variabile**: $Var = \{x, y, z, u, v, \dots\}$;
 - ② **simboluri de operații**: $(f_i)_{i \in I}$; pentru fiecare $i \in I$, numărul natural nenul n_i se numește *ordinul* sau *aritatea* lui f_i ;
 - ③ **simboluri de relații (simboluri de predicate)**: $(R_j)_{j \in J}$; pentru fiecare $j \in J$, numărul natural nenul m_j se numește *ordinul* sau *aritatea* lui R_j ;
 - ④ **simboluri de constante**: $(c_k)_{k \in K}$;
 - ⑤ **simbolul de egalitate**: $=$ (un semn egal îngroșat) (*a nu se confunda cu egalul simplu!*);
 - ⑥ **conectorii logici primitive**: \neg (*negația*), \rightarrow (*implicația*);
 - ⑦ **cuantificatorul universal**: \forall (*oricare ar fi*)
 - ⑧ paranteze: $(,), [,]$, precum și virgula.
- Pentru comoditate, vom spune uneori: “operații”, “relații”/“predicate” și “constante” în loc de “simboluri de operații”, “simboluri de relații/predicate” și “simboluri de constante”, respectiv

Termeni și formule

Definiție

Termenii limbajului \mathcal{L}_τ se definesc, recursiv, astfel:

- ① variabilele și simbolurile de constante sunt termeni;
- ② dacă f este un simbol de operație n -ară și t_1, \dots, t_n sunt termeni, atunci $f(t_1, \dots, t_n)$ este un termen;
- ③ orice termen se obține prin aplicarea regulilor (1) și (2) de un număr finit de ori.

Definiție

Formulele atomice ale limbajului \mathcal{L}_τ se definesc astfel:

- ① dacă t_1 și t_2 sunt termeni, atunci $t_1 = t_2$ este o formulă atomică;
- ② dacă R este un simbol de relație m -ară și t_1, \dots, t_m sunt termeni, atunci $R(t_1, \dots, t_m)$ este o formulă atomică.

Observație

Majoritatea autorilor consideră virgula ca având semnificație implicită, subînțeleasă, în scrierea termenilor și a formulelor atomice, și nu includ virgula în limbajul \mathcal{L}_τ .

Parantezele care încadrează formule (nu argumentele unei funcții sau relații) au același rol ca în sintaxa logicii propoziționale.

Definiție

Formulele limbajului \mathcal{L}_τ se definesc, recursiv, astfel:

- ① formulele atomice sunt formule;
- ② dacă φ este o formulă, atunci $\neg\varphi$ este o formulă;
- ③ dacă φ și ψ sunt formule, atunci $\varphi \rightarrow \psi$ este o formulă;
- ④ dacă φ este o formulă și x este o variabilă, atunci $\forall x\varphi$ este o formulă;
- ⑤ orice formulă se obține prin aplicarea regulilor (1), (2), (3) și (4) de un număr finit de ori.

Notație

Se notează cu $Form(\mathcal{L}_\tau)$ mulțimea formulelor limbajului \mathcal{L}_τ .

Ceilalți conectori logici și celălalt cuantificator

Notație

Introducem abrevierile: pentru orice formule φ, ψ și orice variabilă x :

- **conectorii logici derivați** \vee (*disjuncția*), \wedge (*conjuncția*) și \leftrightarrow (*echivalența*) se definesc astfel:

$$\varphi \vee \psi := \neg \varphi \rightarrow \psi$$

$$\varphi \wedge \psi := \neg (\varphi \rightarrow \neg \psi)$$

$$\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

- **cuantificatorul existențial** \exists (*există*) se definește astfel:

$$\exists x \varphi := \neg \forall x \neg \varphi$$

Observație (convenție privind scrierea conectorilor logici, a cuantificatorilor și a simbolului de egalitate)

- \neg, \forall, \exists vor avea prioritate mai mare;
- $\rightarrow, \vee, \wedge, \leftrightarrow, =$ vor avea prioritate mai mică.

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- **Variabile și substituții**
- Sintaxa calculului cu predicate clasice
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

Variabilele care apar într-un termen

Notație (mulțimile din această notație vor fi definite recursiv mai jos)

Pentru orice termen t și orice formulă φ , notăm:

- $V(t) :=$ mulțimea variabilelor termenului t
- $FV(\varphi) :=$ mulțimea variabilelor *libere* ale formulei φ

Definiție

Pentru orice termen t :

- dacă $t = x$, unde x este o variabilă, atunci $V(t) := \{x\}$
- dacă $t = c$, unde c este o constantă, atunci $V(t) := \emptyset$
- dacă $t = f(t_1, \dots, t_n)$, unde f este un simbol de operație n -ară și t_1, \dots, t_n sunt termeni, atunci $V(t) := \bigcup_{i=1}^n V(t_i)$

Variabilele libere dintr-o formulă

Definiție

Pentru orice formulă φ :

- dacă $\varphi = (t_1 = t_2)$, unde t_1 și t_2 sunt termeni, atunci $FV(\varphi) := V(t_1) \cup V(t_2)$
- dacă $\varphi = R(t_1, \dots, t_m)$, unde R este un simbol de relație m -ară și t_1, \dots, t_m sunt termeni, atunci $FV(\varphi) := \bigcup_{i=1}^m V(t_i)$
- dacă $\varphi = \neg \psi$, pentru o formulă ψ , atunci $FV(\varphi) := FV(\psi)$
- dacă $\varphi = \psi \rightarrow \chi$, pentru două formule ψ, χ , atunci $FV(\varphi) := FV(\psi) \cup FV(\chi)$
- dacă $\varphi = \forall x\psi$, pentru o formulă ψ și o variabilă x , atunci $FV(\varphi) := FV(\psi) \setminus \{x\}$

Remarcă

Este imediat, din definiția anterioară și definiția conectorilor logici derivați și a cuantificatorului existențial, că, pentru orice formule ψ, χ și orice variabilă x :

- $FV(\psi \vee \chi) = FV(\psi \wedge \chi) = FV(\psi \leftrightarrow \chi) = FV(\psi) \cup FV(\chi)$
- $FV(\exists x\psi) = FV(\psi) \setminus \{x\}$

Variabilele legate dintr-o formulă

Definiție

Pentru orice variabilă x și orice formulă φ :

- dacă $x \in FV(\varphi)$, atunci x se numește *variabilă liberă a lui φ* ;
- dacă $x \notin FV(\varphi)$, atunci x se numește *variabilă legată a lui φ* ;
- dacă $FV(\varphi) = \emptyset$ (i. e. φ nu are variabile libere), atunci φ se numește *enunț*.

Exemplu

- În formula $\exists x(x^2=x)$, x este variabilă legată. Această formulă este un enunț.
- În formula $\forall y\forall z(z \cdot x \leq y \cdot z)$, x este variabilă liberă.

Notăție (cum specificăm că nu avem alte variabile libere)

Fie $n \in \mathbb{N}^*$ și x_1, \dots, x_n variabile.

Dacă t este un termen cu $V(t) \subseteq \{x_1, \dots, x_n\}$, atunci vom nota $t(x_1, \dots, x_n)$.

Dacă φ este o formulă cu $FV(\varphi) \subseteq \{x_1, \dots, x_n\}$, atunci vom nota $\varphi(x_1, \dots, x_n)$.

Variabile libere și variabile legate

Observație (diferența dintre tipurile de variabile, intuitiv)

- **Variabilele libere** sunt variabilele care nu intră sub incidența unui cuantificator, variabilele cărora "avem libertatea de a le da valori".
- **Variabilele legate** sunt variabilele care intră sub incidența unui cuantificator, deci sunt destinate parcurgerii unei întregi multimi de valori.

Definiție

Fie x o variabilă, $\varphi(x)$ o formulă și t un termen.

Formula obținută din φ prin *substituția lui x cu t* se notează cu $\varphi(t)$ și se definește astfel:

- fiecare $y \in V(t)$ se înlocuiește cu o variabilă $v \notin V(t)$ care nu apare în $\varphi(x)$, în toate aparițiile *legate* ale lui y în $\varphi(x)$;
- apoi se înlocuiește x cu t .

Exemplu

Fie variabilele x, y, z , formula $\varphi(x) := \exists y(x < y)$ și termenul $t := y + z$.

Atunci $\varphi(t)$ se obține astfel:

- $\varphi(x) = \exists y(x < y)$ se înlocuiește cu $\exists v(x < v)$;
- prin urmare, $\varphi(t) = \exists v(y + z < v)$.

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- **Sintaxa calculului cu predicate clasic**
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

Axiomele logicii clasice a predicatorilor

Axiomele calculului cu predicate clasic: pentru φ, ψ, χ formule arbitrar, t termen arbitrar, n, i numere naturale nenule arbitrar și $x, y, y_1, \dots, y_n, v_1, \dots, v_n$ variabile arbitrar:

- axiomele calculului propozițional:

$$(G_1) \quad \varphi \rightarrow (\psi \rightarrow \varphi)$$

$$(G_2) \quad (\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$$

$$(G_3) \quad (\neg \varphi \rightarrow \neg \psi) \rightarrow (\psi \rightarrow \varphi)$$

- regula ($\rightarrow \forall$):

$$(G_4) \quad \forall x(\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \forall x \psi), \text{ dacă } x \notin FV(\varphi)$$

- o regulă privind substituțiile:

$$(G_5) \quad \forall x \varphi(x, y_1, \dots, y_n) \rightarrow \varphi(t, y_1, \dots, y_n)$$

- axiomele egalității:

$$(G_6) \quad x = x$$

$$(G_7)$$

$$(x = y) \rightarrow (t(v_1, \dots, v_{i-1}, x, v_{i+1}, \dots, v_n) = t(v_1, \dots, v_{i-1}, y, v_{i+1}, \dots, v_n))$$

$$(G_8)$$

$$(x = y) \rightarrow (\varphi(v_1, \dots, v_{i-1}, x, v_{i+1}, \dots, v_n) = \varphi(v_1, \dots, v_{i-1}, y, v_{i+1}, \dots, v_n))$$

Teoremele formale, i.e. adevărurile sintactice în logica clasică a predicatelor

Notație

Faptul că o formulă φ este *teoremă (formală) (adevăr sintactic)* a(l) lui \mathcal{L}_τ se notează cu $\vdash \varphi$ și se definește, recursiv, ca mai jos.

Definiție

- ① Orice axiomă e teoremă formală a lui \mathcal{L}_τ .
- ② Pentru orice formule φ, ψ ,
$$\frac{\vdash \psi, \vdash \psi \rightarrow \varphi}{\vdash \varphi}$$
 (regula de deducție *modus ponens* (MP)).
- ③ Pentru orice formulă φ și orice variabilă x ,
$$\frac{\vdash \varphi}{\vdash \forall x \varphi}$$
 (regula de deducție numită *principiul generalizării* (PG)).
- ④ Orice teoremă formală se obține prin aplicarea regulilor (1), (2) și (3) de un număr finit de ori.

Deducre sintactică în logica clasică a predicatelor

Notăție

Fie Σ o mulțime de formule ale lui \mathcal{L}_τ . Faptul că o formulă φ se deduce (formal) din ipotezele Σ (φ este consecință sintactică a mulțimii de ipoteze Σ) se notează cu $\Sigma \vdash \varphi$ și se definește, recursiv, ca mai jos.

Definiție

Fie Σ o mulțime de formule ale lui \mathcal{L}_τ .

- ① Orice axiomă a lui \mathcal{L}_τ se deduce formal din Σ .
- ② $\Sigma \vdash \varphi$, oricare ar fi $\varphi \in \Sigma$.
- ③ Pentru orice formule φ, ψ ,
$$\frac{\Sigma \vdash \psi, \Sigma \vdash \psi \rightarrow \varphi}{\Sigma \vdash \varphi} \text{ (regula de deducție } modus ponens \text{ (MP))}$$
- ④ Pentru orice formulă φ și orice variabilă x ,
$$\frac{\Sigma \vdash \varphi}{\Sigma \vdash \forall x \varphi} \text{ (regula de deducție numită } principiul generalizării \text{ (PG))}$$
- ⑤ Orice consecință sintactică a lui Σ se obține prin aplicarea regulilor (1), (2), (3) și (4) de un număr finit de ori.

Teorema deducției sintactice în logica clasică a predicatorilor

Remarcă

Pentru orice formulă φ , are loc echivalența:

$$\emptyset \vdash \varphi \Leftrightarrow \vdash \varphi.$$

Teoremă (Teorema deducției)

Pentru orice mulțime de **formule** Σ , orice **enunț** φ și orice **formulă** ψ , are loc echivalența:

$$\Sigma \vdash \varphi \rightarrow \psi \Leftrightarrow \Sigma \cup \{\varphi\} \vdash \psi.$$

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasic
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

O interpretare/evaluare/semantică dă variabilelor valori într-o structură de ordinul I

- Fie \mathcal{A} o structură de ordinul I de signatură τ .
- Fixăm pe \mathcal{A} pentru cele ce urmează.
- A va fi universul structurii \mathcal{A} (mulțimea ei suport).
- Pentru fiecare simbol de operație f , fiecare simbol de relație R și fiecare simbol de constantă c din signatura τ , notăm cu $f^{\mathcal{A}}$, respectiv $R^{\mathcal{A}}$, respectiv $c^{\mathcal{A}}$ operația, respectiv relația, respectiv constanta corespunzătoare din \mathcal{A} .

Definiție

O *interpretare* (sau *evaluare*, sau *semantică*) a limbajului \mathcal{L}_{τ} în structura \mathcal{A} este o funcție $s : \text{Var} \rightarrow A$.

Fiecare variabilă $x \in \text{Var}$ este “interpretată” prin elementul $s(x) \in A$.

Prelungim o interpretare s la mulțimea tuturor termenilor, obținând o funcție $s : \text{Term}(\mathcal{L}_\tau) \rightarrow A$.

Definiție (valorile asociate termenilor de o interpretare)

Pentru orice interpretare s și orice termen t , definim recursiv elementul $s(t) \in A$:

- dacă $t = x \in \text{Var}$, atunci $s(t) = s(x)$;
- dacă $t = c \in C$, atunci $s(t) = c$;
- dacă $t = f(t_1, \dots, t_n)$, unde $f \in F$ are aritatea $n \in \mathbb{N}^*$, iar $t_1, \dots, t_n \in \text{Term}(\mathcal{L}_\tau)$, atunci $s(t) = f(s(t_1), \dots, s(t_n))$.

Notație

Pentru orice interpretare $s : \text{Var} \rightarrow A$, orice $x \in \text{Var}$ și orice $a \in A$, notăm cu $s[x]_a : \text{Var} \rightarrow A$ interpretarea definită prin: oricare ar fi $v \in \text{Var}$,

$$s[x]_a(v) = \begin{cases} a, & \text{dacă } v = x, \\ s(v), & \text{dacă } v \neq x. \end{cases}$$

$s[x]_a$ ia valoarea a în variabila x și aceeași valoare ca s în celelalte variabile.

Definiție (valorile booleene asociate formulelor de o interpretare)

Pentru orice interpretare s și orice formulă φ , valoarea de adevăr a lui φ în interpretarea s este un element din algebra Boole standard $\mathcal{L}_2 = \{0, 1\}$, notat cu $s(\varphi)$, definit, recursiv, astfel:

- dacă $\varphi = (t_1 = t_2)$, pentru doi termeni t_1, t_2 , atunci

$$s(\varphi) = \begin{cases} 1, & \text{dacă } s(t_1) = s(t_2), \\ 0, & \text{dacă } s(t_1) \neq s(t_2) \end{cases}$$

- dacă $\varphi = R(t_1, \dots, t_m)$, unde $R \in \mathcal{R}$ are aritatea $m \in \mathbb{N}^*$, iar t_1, \dots, t_m sunt

$$\text{termeni, atunci } s(\varphi) = \begin{cases} 1, & \text{dacă } (s(t_1), \dots, s(t_m)) \in R, \\ 0, & \text{dacă } (s(t_1), \dots, s(t_m)) \notin R \end{cases}$$

- dacă $\varphi = \neg \psi$, pentru o formulă ψ , atunci $s(\varphi) = \overline{s(\psi)}$;

- dacă $\varphi = \psi \rightarrow \chi$, pentru două formule ψ, χ , atunci $s(\varphi) = s(\psi) \rightarrow s(\chi)$;

- dacă $\varphi = \forall x \psi$, unde $x \in \text{Var}$, iar ψ este o formulă, atunci

$$s(\varphi) = \bigwedge_{a \in A} s[a](\psi).$$

Remarcă

Este imediat că, pentru orice interpretare $s : \text{Var} \rightarrow A$, orice formule ψ, χ și orice $x \in \text{Var}$, au loc egalitățile:

- $s(\psi \vee \chi) = s(\psi) \vee s(\chi);$
- $s(\psi \wedge \chi) = s(\psi) \wedge s(\chi);$
- $s(\psi \leftrightarrow \chi) = s(\psi) \leftrightarrow s(\chi);$
- $s(\exists x\psi) = \bigvee_{a \in A} s[a](\psi).$

Lemă (dacă două interpretări coincid pe variabilele dintr-un termen, atunci ele coincid în acel termen)

Fie $s_1, s_2 : \text{Var} \rightarrow A$ două interpretări. Atunci, pentru orice termen t , are loc implicația: $s_1|_{V(t)} = s_2|_{V(t)} \Rightarrow s_1(t) = s_2(t)$.

Propoziție (dacă două interpretări coincid pe variabilele libere dintr-o formulă, atunci ele coincid în acea formulă)

Fie $s_1, s_2 : \text{Var} \rightarrow A$ două interpretări. Atunci, pentru orice formulă φ , are loc implicația: $s_1|_{FV(\varphi)} = s_2|_{FV(\varphi)} \Rightarrow s_1(\varphi) = s_2(\varphi)$.

În mod trivial, oricare două interpretări $s_1, s_2 : \text{Var} \rightarrow A$ coincid pe \emptyset :
 $(\forall x)(x \in \emptyset \Rightarrow s_1(x) = s_2(x))$ e adevărată, pentru că $x \in \emptyset$ e falsă pentru orice x ,
așadar $x \in \emptyset \Rightarrow s_1(x) = s_2(x)$ e adevărată pentru orice x . Prin urmare:

Corolar (cum enunțurile nu au variabile libere (i.e. $FV(\varphi) = \emptyset$ pt. orice enunț φ), rezultă că, într-un enunț, toate interpretările au aceeași valoare)

Dacă φ este un enunț, atunci $s(\varphi)$ nu depinde de interpretarea $s : \text{Var} \rightarrow A$.

Notăție (această valoare nu depinde decât de structura algebrică \mathcal{A})

Corolarul anterior ne permite să notăm, pentru orice enunț φ , cu $||\varphi||_{\mathcal{A}} = s(\varphi)$ valoarea oricărei interpretări $s : \text{Var} \rightarrow A$ în enunțul φ .

Definiție

Pentru orice enunț φ , notăm:

$$\mathcal{A} \models \varphi \text{ dacă } ||\varphi||_{\mathcal{A}} = 1.$$

În acest caz, spunem că \mathcal{A} *satisfacă* φ sau φ este *adevărat* în \mathcal{A} sau \mathcal{A} este *model pentru* φ .

Pentru orice mulțime Γ de enunțuri, spunem că \mathcal{A} *satisfacă* Γ sau că \mathcal{A} este *model pentru* Γ dacă $\mathcal{A} \models \varphi$, pentru orice $\varphi \in \Gamma$. Notăm acest lucru cu $\mathcal{A} \models \Gamma$.

Recapitulare Logica Clasică a Predicatelor

Remarcă

Este imediat, din definiția mulțimii variabilelor libere ale unei formule, că, dacă $n \in \mathbb{N}^*$, $x_1, \dots, x_n \in V$ și $\varphi(x_1, \dots, x_n)$ este o formulă, atunci $\forall x_1 \dots \forall x_n \varphi(x_1, \dots, x_n)$ este un enunț.

Definiție

Pentru orice $n \in \mathbb{N}^*$, orice variabile x_1, \dots, x_n și orice formulă $\varphi(x_1, \dots, x_n)$, notăm:

$$\mathcal{A} \models \varphi(x_1, \dots, x_n) \text{ ddacă } \mathcal{A} \models \forall x_1 \dots \forall x_n \varphi(x_1, \dots, x_n).$$

În acest caz, spunem că \mathcal{A} *satisfacă* $\varphi(x_1, \dots, x_n)$ sau $\varphi(x_1, \dots, x_n)$ este *adevărată în* \mathcal{A} sau \mathcal{A} este *model pentru* $\varphi(x_1, \dots, x_n)$.

Pentru orice mulțime Σ de formule, spunem că \mathcal{A} *satisfacă* Σ sau că \mathcal{A} este *model pentru* Σ ddacă \mathcal{A} este model pentru fiecare formulă din Σ . Notăm acest lucru cu $\mathcal{A} \models \Sigma$.

Remarcă

$$\mathcal{A} \models \emptyset.$$

Tautologiile, i.e. adevărurile semantice ale logicii clasice a predicatelor

- Renunțăm la fixarea structurii \mathcal{A} (**nu** și la fixarea signaturii τ).

Definiție

Dacă φ este un enunț, atunci spunem că φ este *universal adevărat* (*adevăr semantic, tautologie*) dacă $\mathcal{A} \models \varphi$, oricare ar fi structura de ordinul I \mathcal{A} de signură τ . Notăm acest lucru cu $\models \varphi$.

Definiție

Dacă $n \in \mathbb{N}^*$, $x_1, \dots, x_n \in V$ și $\varphi(x_1, \dots, x_n)$ este o formulă, atunci spunem că $\varphi(x_1, \dots, x_n)$ este *universal adevărată* (*adevăr semantic, tautologie*) dacă enunțul $\forall x_1 \dots \forall x_n \varphi(x_1, \dots, x_n)$ este universal adevărat. Notăm acest lucru cu $\models \varphi(x_1, \dots, x_n)$.

Deducrea semantică în logica clasică a predicatorilor

Definiție

Pentru orice mulțime Σ de formule și orice formulă φ , spunem că φ se deduce semantic din ipotezele Σ sau că φ este consecință semantică a mulțimii de ipoteze Σ dacă φ este adevărată în orice model \mathcal{A} al lui Σ , i. e., pentru orice structură de ordinul I \mathcal{A} de signură τ , are loc implicația: $\mathcal{A} \models \Sigma \Rightarrow \mathcal{A} \models \varphi$. Notăm acest lucru prin: $\Sigma \models \varphi$.

Remarcă

Pentru orice formulă φ , are loc echivalența:

$$\emptyset \models \varphi \Leftrightarrow \models \varphi.$$

Teoremă (Teorema deducției semantice)

Pentru orice mulțime de formule Σ , orice enunț φ și orice formulă ψ , are loc echivalența:

$$\Sigma \models \varphi \rightarrow \psi \Leftrightarrow \Sigma \cup \{\varphi\} \models \psi.$$

Teorema de completitudine pentru logica clasică a predicatelor

Teoremă (Teorema de completitudine tare (Teorema de completitudine extinsă))

Pentru orice formulă φ și orice multime de formule Σ , are loc echivalența:

$$\Sigma \vdash \varphi \Leftrightarrow \Sigma \vDash \varphi.$$

În cazul particular în care $\Sigma = \emptyset$, din **Teorema de completitudine extinsă** obținem:

Corolar (Teorema de completitudine)

Pentru orice formulă φ , are loc echivalența:

$$\vdash \varphi \Leftrightarrow \vDash \varphi.$$

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasice
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

Vom vedea că un program în Prolog definește un limbaj de ordinul I, nu doar o algebră

```
femeie(ana). femeie(carmen). femeie(elena). femeie(eva).
femeie(maria). femeie(sara). femeie(valentina).
barbat(adam). barbat(constantin). barbat(eugen). barbat(george).
barbat/ion). barbat(iosif). barbat(tudor). barbat(victor).
parinte(sara, adam). parinte(sara, eva).
parinte(constantin, iosif). parinte(constantin, elena).
parinte(valentina, ion). parinte(valentina, sara).
parinte(victor, constantin). parinte(victor, maria).
parinte(ana, victor). parinte(ana, valentina).
parinte(carmen, victor). parinte(carmen, valentina).
parinte(george, victor). parinte(george, valentina).
parinte(eugen, tudor). parinte(eugen, ana).
frati(X, Y) :- X \= Y, parinte(X, P), parinte(Y, P).
tata(X, T) :- parinte(X, T), barbat(T).
mama(X, M) :- parinte(X, M), femeie(M).
unchi(X, U) :- parinte(X, P), frati(P, U), barbat(U).
matusa(X, M) :- parinte(X, P), frati(P, M), femeie(M).
bunic(X, B) :- parinte(X, P), parinte(P, B).
stramos(X, X, 0). % al treilea argument e numarul de generatii
stramos(X, S, G) :- parinte(X, P), stramos(P, S, H), G is H + 1.
```

În programul în Prolog de mai sus:

- $X, P, T, M, U, B, S, G, H$ sunt **variabile**;
- $+$ este **operație**;
- *femeie, barbat, frati, parinte, tata, mama, unchi, matusa, bunic, stramos* sunt **predicate**;
- $H + 1$ este un **termen**;
- *ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor, 0 și 1* sunt **constante Prolog**;
- de exemplu, *femeie(eva), frati(victor, george), parinte(victor, ion), parinte(X, M), stramos($X, X, 0$), stramos(X, S, G)* sunt **formule atomice**.

REGULILE și ultimul FAPT din programul anterior corespund **formulelor**:

(atenție: toate variabilele cuantificate UNIVERSAL)

- $(\forall X) (\forall Y) (\forall P) [(\neg(X = Y) \wedge \text{parinte}(X, P) \wedge \text{parinte}(Y, P)) \rightarrow \text{frati}(X, Y)]$
- $(\forall X) (\forall T) [(\text{parinte}(X, T) \wedge \text{barbat}(T)) \rightarrow \text{tata}(X, T)]$
- $(\forall X) (\forall M) [(\text{parinte}(X, M) \wedge \text{femeie}(M)) \rightarrow \text{mama}(X, M)]$
- $(\forall X) (\forall U) (\forall P) [(\text{pariente}(X, P) \wedge \text{frati}(P, U) \wedge \text{barbat}(U)) \rightarrow \text{unchi}(X, U)]$
- $(\forall X) (\forall M) (\forall P) [(\text{pariente}(X, P) \wedge \text{frati}(P, M) \wedge \text{femeie}(M)) \rightarrow \text{matusa}(X, M)]$
- $(\forall X) (\forall B) [(\text{pariente}(X, P) \wedge \text{pariente}(P, B)) \rightarrow \text{bunic}(X, B)]$

- $(\forall X) (stramos(X, X, 0))$
- $(\forall X) (\forall S) (\forall G) [(parinte(X, P) \wedge stramos(P, S, H) \wedge (G = H + 1)) \rightarrow stramos(X, S, G)]$

INTEROGĂRILE:

- ?- $parinte(C, victor)$. % Care sunt copiii lui Victor?
- ?- $parinte(F, victor), femeie(F)$. % Care sunt fiicele lui Victor?
- ?- $stramos(carmen, S, 3), barbat(S)$. /* Care sunt strabunicii (cunoscuti, adica din baza de cunostinte de mai sus, ai) lui Carmen? */
- ?- $tata(X, T), stramos(T, elena, G)$. /* Pentru cine este Elena stramos din partea tatalui, cine este tatal acelui urmas si cate generatii sunt intre tata si Elena? */

corespund **formulelor**:

(atenție: toate variabilele cuantificate EXISTENȚIAL)

- $(\exists C) (pariente(C, victor))$
- $(\exists F) (pariente(F, victor) \wedge femeie(F))$
- $(\exists S) (stramos(carmen, S, 3) \wedge barbat(S))$
- $(\exists X) (\exists T) (\exists G) (tata(X, T) \wedge stramos(T, elena, G))$

Care este **signatura** limbajului de ordinul I definit în programul Prolog de mai sus?

Pentru început, să considerăm programul anterior, mai puțin definiția predicatului **stramos**:

```
femeie(ana). femeie(carmen). femeie(elena). femeie(eva).
femeie(maria). femeie(sara). femeie(valentina).
barbat(adam). barbat(constantin). barbat(eugen). barbat(george).
barbat/ion). barbat(iosif). barbat(tudor). barbat(victor).
parinte(sara, adam). parinte(sara, eva).
parinte(constantin, iosif). parinte(constantin, elena).
parinte(valentina, ion). parinte(valentina, sara).
parinte(victor, constantin). parinte(victor, maria).
parinte(ana, victor). parinte(ana, valentina).
parinte(carmen, victor). parinte(carmen, valentina).
parinte(george, victor). parinte(george, valentina).
parinte(eugen, tudor). parinte(eugen, ana).
frati(X, Y) :- X \= Y, parinte(X, P), parinte(Y, P).
tata(X, T) :- parinte(X, T), barbat(T).
mama(X, M) :- parinte(X, M), femeie(M).
unchi(X, U) :- parinte(X, P), frati(P, U), barbat(U).
matusa(X, M) :- parinte(X, P), frati(P, M), femeie(M).
bunic(X, B) :- parinte(X, P), parinte(P, B).
```

Vom avea, de fapt, un întreg limbaj, cu simboluri de operării, predicate, constante

Signatura corespunzătoare programului de mai sus este

aritățile operațiilor aritățile relațiilor aritățile constantelor

Amintesc că aritățile sunt numerele de argumente.

Structura de ordinul I de această signatură definită în programul de mai sus este:

$\mathcal{A} = (A; \emptyset; frati, parinte, tata, mama, unchi, matusa, bunic, femeie, barbat)$

ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor), cu:

- mulțimea suport (i.e. mulțimea elementelor) $A = \{ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor\}$;
 - nicio operație;
 - 7 predicate (i.e. relații) binare (i.e. de aritate 2, i.e. cu 2 argumente): $frati, parinte, tata, mama, unchi, matusa, bunic \subseteq A^2$;
 - 2 predicate (i.e. relații) unare (i.e. de aritate 1, i.e. cu 1 argument): $femeie, barbat \subseteq A$;
 - 15 constante (amintesc că acestea sunt operațiile zeroare, operațiile de aritate 0, i.e. operațiile fără argumente): $ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor \in A$.

Primele fapte din acest program definesc relațiile unare *femeie* și *barbat*:

- $femeie = \{ana, carmen, elena, eva, maria, sara, valentina\}$;
- $barbat = \{adam, constantin, eugen, george, ion, iosif, tudor, victor\}$.

Pentru orice $x \in A$, faptul că $x \in femeie$ se mai scrie: $femeie(x)$. La fel pentru relația unară *barbat*.

Următoarele fapte definesc relația binară *parinte*:

- $parinte = \{(sara, adam), (sara, eva), (constantin, iosif), (constantin, elena), (valentina, ion), (valentina, sara), (victor, constantin), (victor, maria), (ana, victor), (ana, valentina), (carmen, victor), (carmen, valentina), (george, victor), (george, valentina), (eugen, tudor), (eugen, ana)\}$.

Pentru orice $x, y \in A$, faptul că $(x, y) \in parinte$ se mai scrie: $parinte(x, y)$. La fel pentru următoarele relații binare.

Regulile următoare definesc relațiile binare *frati*, *tata*, *mama*, *unchi* și *matusa*:

- $frati = \{(x, y) \in A^2 \mid x \neq y, (\exists p \in A)((x, p), (y, p) \in parinte)\}$;
- $tata = \{(x, t) \in A^2 \mid (x, t) \in parinte, t \in barbat\}$;
- $mama = \{(x, m) \in A^2 \mid (x, m) \in parinte, m \in femeie\}$;
- $unchi = \{(x, u) \in A^2 \mid (\exists p \in A)((x, p) \in parinte, (p, u) \in frati, u \in barbat)\}$;
- $matusa = \{(x, m) \in A^2 \mid (\exists p \in A)((x, p) \in parinte, (p, m) \in frati, m \in femeie)\}$;
- $bunic = \{(x, b) \in A^2 \mid (\exists p \in A)((x, p), (p, b) \in parinte)\}$.

Și acum să reanalizăm programul inițial, care include și predicatul ternar (relația ternară, i.e. de aritate 3, i.e. cu 3 argumente) $\text{stramos} \subseteq A \times A \times \mathbb{N}$, definită prin ultimul fapt și ultima regulă, care dau următoarea definiție recursivă:

- $\{(x, x, 0) \mid x \in A\} \subseteq \text{stramos}$ și:
- pentru orice $x, s, p \in A$ și orice $h \in \mathbb{N}$, dacă $(x, p) \in \text{parinte}$ și $(p, s, h) \in \text{stramos}$, atunci $(x, s, h + 1) \in \text{stramos}$.

Pentru orice $x, s \in A$ și orice $g \in \mathbb{N}$, faptul că $(x, s, g) \in \text{stramos}$ se mai scrie $\text{stramos}(x, s, g)$.

Dar stramos nu este o relație ternară pe A , ci are primele două argumente din A , iar al treilea din \mathbb{N} !

În plus, 0 și 1 sunt constante din \mathbb{N} , nu din A , iar $+ : \mathbb{N}^2 \rightarrow \mathbb{N}$, deci $+$ este o operație binară pe \mathbb{N} , nu pe A .

Așadar cadrul de lucru din secțiunea anterioară a acestui curs trebuie extins! Putem face acest lucru înlocuind mulțimea suport a structurii \mathcal{A} cu $A \cup \mathbb{N}$ și permitând operațiilor să fie parțial definite.

Definiție (mnemonic din cursul de logică matematică)

Dacă M și N sunt mulțimi, atunci o *funcție parțială (relație binară funcțională)* de la M la N este o relație binară între M și N $\varphi \subseteq M \times N$ cu proprietatea că:

pentru orice $x \in M$, există **cel mult un** $y \in N$ cu proprietatea că $(x, y) \in \varphi$.

Definiție (continuare)

Dacă, pentru un $x \in M$, există un $y \in N$ cu $(x, y) \in \varphi$, atunci, ca în cazul *funcțiilor* (i.e. al *relațiilor binare funcționale totale*), se notează $\varphi(x) = y$ și acest $y \in N$ se numește *valoarea funcției parțiale* φ în punctul x .

Notație

Dacă M și N sunt mulțimi, atunci faptul că φ este o funcție parțială de la M la N se notează:

$$\varphi : M \rightharpoonup N.$$

Exemplu (să analizăm această soluție pe un tip cunoscut de algebră)

Dacă V este un spațiu vectorial, format din:

- un grup abelian $(G, \circ, \bar{}, e)$, cu $\circ : G^2 \rightarrow G$ (operație binară pe G : legea de compozitie), $\bar{} : G \rightarrow G$ (operație unară pe G : inversarea) și $e \in G$ (constantă din G , i.e. operație zeroară pe G : elementul neutru),
- un corp $(K, +, \cdot, -, -^{-1}, 0, 1)$, cu $+ : K^2 \rightarrow K$ și $\cdot : K^2 \rightarrow K$ (operații binare pe K), $- : K \rightarrow K$ și $-^{-1} : K \rightarrow K$ (operații unare pe K), iar $0, 1 \in K$ (constante din K , i.e. operații zeroare pe K),
- și o lege de compozitie între scalarii din K și vectorii din G : $* : K \times G \rightarrow G$.

O structură algebrică având mulțimea suport $K \times V$ ar trebui să aibă orice operație f de aritate $n \in \mathbb{N}$ de forma $f : (K \times V)^n \rightarrow K \times V$, iar, în cazul unei

Exemplu (continuare)

structuri cu mulțimea suport $K \cup V$, o operație g de aritate n ar trebui să fie o funcție $g : (K \cup V)^n \rightarrow K \cup V$.

Așadar, putem privi spațiul vectorial \mathcal{V} ca pe o structură algebrică având mulțimea suport $K \cup V$ dacă permitem operațiilor sale să fie funcții parțiale.

O altă posibilitate este să privim spațiul vectorial \mathcal{V} ca pe o structură algebrică având două mulțimi suport: K și G , mai precis familia de mulțimi (K, G) , cu operațiile total definite, dar astfel încât, pentru orice $m, p \in \mathbb{N}$, o operație h de aritate $m + p$ (i.e. cu $m + p$ argumente) a lui \mathcal{V} să poată fi de forma:

$h : K^m \times G^p \rightarrow K$ sau $h : K^m \times G^p \rightarrow G$.

O astfel de structură algebrică se numește *algebră bisortată*, i.e. *cu două sorturi*, cu două mulțimi suport, două tipuri de elemente, în cazul acesta: SCALARI din K și VECTORI din G .

La fel ca în exemplul anterior, o *algebră multisortată*, i.e. *cu mai multe sorturi*, cu mai multe mulțimi suport, mai multe tipuri de elemente, și care, pe lângă operații, e înzestrată și cu, relații, va fi o algebră de forma:

$\mathcal{M} = ((M_s)_{s \in S}; (f_i)_{i \in I}; (\rho_j)_{j \in J}; (c_k)_{k \in K})$, unde S e *mulțimea de sorturi*, elementele structurii algebrice \mathcal{M} sunt elementele mulțimilor din familia de mulțimi $(M_s)_{s \in S}$, $(f_i)_{i \in I}$ este familia de operații, $(\rho_j)_{j \in J}$ este familia de relații, iar $(c_k)_{k \in K}$ este familia de constante ale structurii algebrice \mathcal{M} :



Pentru orice mulțime **nevidă** S , o algebră S -sortată încestrată și cu operații, și cu relații este o structură algebrică $\mathcal{M} = ((M_s)_{s \in S}; (f_i)_{i \in I}; (\rho_j)_{j \in J}; (c_k)_{k \in K})$, unde:

- elementele lui S se numesc *sorturi* (intuitiv, *sorturile* sunt indici corespunzători TIPURILOR DE ELEMENTE, în acest caz TIPURILOR DE DATE cu care lucrează programul de mai sus);
- $(M_s)_{s \in S}$ este *mulțimea S -sortată a elementelor* lui \mathcal{M} , adică M este o familie de mulțimi indexată de S și, pentru fiecare $s \in S$, elementele lui M_s se numesc *elementele lui \mathcal{M} de sort s* ;
- I, J, K sunt mulțimi, nu neapărat nevide;
- pentru fiecare $i \in I$, există $n_i \in \mathbb{N}^*$ și sorturile $s_{i,1}, \dots, s_{i,n_i}, s_i \in S$ astfel încât $f_i : M_{s_{i,1}} \times \dots \times M_{s_{i,n_i}} \rightarrow M_{s_i}$: f_i este o operație de aritate n_i a lui \mathcal{M} ;
- pentru fiecare $j \in J$, există $m_j \in \mathbb{N}^*$ și sorturile $t_{j,1}, \dots, t_{j,m_j} \in S$ astfel încât $\rho_j \subseteq M_{t_{j,1}} \times \dots \times M_{t_{j,m_j}} \rightarrow M_{t_j}$: ρ_j este o relație de aritate m_j a lui \mathcal{M} ;
- pentru fiecare $k \in K$, există sortul $u_k \in S$ astfel încât $c_k \in M_{u_k}$: c_k este o constantă a lui \mathcal{M} .

Putem defini structura de ordinul I corespunzătoare programului de mai sus ca pe o algebră bisortată, cu mulțimea suport (A, \mathbb{N}) .

Desigur, operația binară $+$ pe \mathbb{N} nu este definită în programul de mai sus, ci este predefinită în Prolog, împreună cu alte operații aritmetice și relații, de exemplu $<$, $=<$ etc..

Să mai observăm că relațiile (predicale) pot fi privite ca operații având *sortul rezultatului* BOOLEAN: adăugăm o a treia mulțime suport, conținând valorile booleene: $\{\text{false}, \text{true}\}$ sau $\mathcal{L}_2 = \{0, 1\}$ ($0 = \text{false}$, $1 = \text{true}$), astfel că structura algebrică asociată programului de mai sus devine o *algebră trisortată*, cu mulțimea suport $(A_s)_{s \in \overline{1,3}} = (A, \mathbb{N}, \{\text{false}, \text{true}\})$, și orice relație din această algebră devine o operație cu valori în $\{\text{false}, \text{true}\}$: pentru orice $n \in \mathbb{N}^*$ și orice relație n -ară $\rho \subseteq A_{s_1} \times \dots \times A_{s_n}$, unde s_1, \dots, s_n sunt SORTURI, în acest caz aparținând mulțimii de sorturi $\overline{1,3}$, în Prolog această relație este implementată ca o operație

$$\rho' : A_{s_1} \times \dots \times A_{s_n} \rightarrow \{\text{false}, \text{true}\},$$

definită astfel: pentru orice $x_1 \in A_{s_1}, \dots, x_n \in A_{s_n}$,

$$\rho'(x_1, \dots, x_n) = \begin{cases} \text{true}, & \text{dacă } (x_1, \dots, x_n) \in \rho, \\ \text{false}, & \text{altfel.} \end{cases}$$

De acum încolo, operația ρ' va fi notată tot ρ . De exemplu, pentru programul de mai sus:

$\text{frati}(ana, carmen) = \text{true}$,

$\text{frati}(ana, eva) = \text{false}$,

pentru orice $X \in A$, $\text{stramos}(X, X, 0) = \text{true}$, iar $\text{stramos}(X, X, 1) = \text{false}$.

Iar, cum **constantele** sunt *operații zeroare (nulare)*, i.e. operații de aritate 0, operații fără argumente, nu e nevoie să facem distincție între constante și operații.

Ce este o *algebră multisortată* înzestrată numai cu OPERAȚII? Aceasta este noțiunea propriu-zisă de algebră multisortată.

Definiție

Pentru orice mulțime **nevidă** S , o *algebră S -sortată* este o structură algebraică $\mathcal{M} = ((M_s)_{s \in S}; (f_i)_{i \in I})$, unde:

- elementele lui S se numesc *sorturi* (repet că *sorturile* sunt indici corespunzători TIPURILOR DE ELEMENTE, TIPURILOR DE DATE);
- mulțimea suport, i.e. mulțimea *elementelor* algebrei \mathcal{M} este *mulțimea S -sortată*, adică familia de mulțimi indexată de S $(M_s)_{s \in S}$; pentru fiecare $s \in S$, elementele lui M_s se numesc *elementele lui \mathcal{M} de sort s* ;
- $(f_i)_{i \in I}$ este familia *operațiilor* lui \mathcal{M} ; pentru fiecare $i \in I$, există $n_i \in \mathbb{N}$ și sorturile $s_{i,1}, \dots, s_{i,n_i}, s_i \in S$ astfel încât $f_i : M_{s_{i,1}} \times \dots \times M_{s_{i,n_i}} \rightarrow M_{s_i}$; f_i este o operație de aritate n_i a lui \mathcal{M} ; dacă $n_i = 0$, atunci această declarație a lui f_i devine: $f_i \in M_{s_i}$ (a se revedea discuția dintr-un curs anterior privind operațiile zeroare).

Așadar ce structură algebraică multisortată corespunde programului anterior în Prolog? Cum am procedat și mai sus, să scriem operațiile (acestea incluzând relațiile, predicatele) descrescător după arități.

$\mathcal{A} = ((A, \mathbb{N}, \{\text{false}, \text{true}\}); stramos, +, frati, parinte, tata, mama, unchi, matusa, bunic, femeie, barbat, ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor, 0, 1, \text{false}, \text{true})$, unde:

- $A = \{ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor\}$;
- $stramos \subseteq A \times A \times \mathbb{N}$ (relație ternară, predicat ternar), așadar, ca operație ternară: $stramos : A \times A \times \mathbb{N} \rightarrow \{\text{false}, \text{true}\}$;
- $+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ (operație binară);
- $frati, parinte, tata, mama, unchi, matusa, bunic \subseteq A \times A$ (relații binare, predicate binare), așadar, ca operații binare:
 $frati, parinte, tata, mama, unchi, matusa, bunic : A \times A \rightarrow \{\text{false}, \text{true}\}$;
- $femeie, barbat \subseteq A$ (relații unare, predicate unare), așadar, ca operații unare:
 $femeie, barbat : A \rightarrow \{\text{false}, \text{true}\}$;
- $ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor \in A$ (constante);
- $0, 1 \in \mathbb{N}$ (constante);
- $\text{false}, \text{true} \in \{\text{false}, \text{true}\}$ (constante);

operațiile nonnulare (i.e. cu argumente) ale algebrei \mathcal{A} sunt definite astfel:

operația binară $+$: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ este predefinită în Prolog;

$$femeie(x) = \begin{cases} \text{true}, & x \in \{\text{ana, carmen, elena, eva, maria, sara, valentina}\}, \\ \text{false}, & x \in A \setminus \{\text{ana, carmen, elena, eva, maria, sara, valentina}\}; \end{cases}$$

$$barbat(x) = \begin{cases} \text{true}, & x \in \{\text{adam, constantin, eugen, george, ion, iosif, tudor, victor}\}, \\ \text{false}, & x \in A \setminus \{\text{adam, constantin, eugen, george, ion, iosif, tudor, victor}\}; \end{cases}$$

pentru orice $u \in A \times A$, $parinte(u) =$

$$\begin{cases} \text{true}, & u \in \{(\text{sara, adam}), (\text{sara, eva}), (\text{constantin, iosif}), (\text{constantin, elena}), \\ & (\text{valentina, ion}), (\text{valentina, sara}), (\text{victor, constantin}), (\text{victor, maria}), \\ & (\text{ana, victor}), (\text{ana, valentina}), (\text{carmen, victor}), (\text{carmen, valentina}), \\ & (\text{george, victor}), (\text{george, valentina}), (\text{eugen, tudor}), (\text{eugen, ana})\}, \\ \text{false}, & \text{altfel}; \end{cases}$$

pentru orice $(x, y) \in A^2$, (eliminând, conform convenției folosite uzual, o pereche de paranteze din scrierea $frati((x, y))$, și la fel mai jos)

$$frati(x, y) = \begin{cases} \text{true}, & x \neq y \text{ și } (\exists p \in A) (parinte}(x, p) = parinte(y, p) = \text{true}), \\ \text{false}, & \text{altfel}; \end{cases}$$

altfel scris, cu notațiile obișnuite \wedge pentru conjuncția logică și \neg pentru negația logică, $frati(x, y) = \neg(x = y) \wedge \bigwedge_{p \in A} (parinte}(x, p) \wedge parinte(y, p);$

în același mod pot fi scrise definițiile următoarelor predicate: pentru orice

$(x, y) \in A^2$, $tata(x, y) = parinte(x, y) \wedge barbat(y)$,

$mama(x, y) = parinte(x, y) \wedge femeie(y)$,

$unchi(x, y) = \bigwedge_{p \in A} (parinte(x, p) \wedge frati(p, y) \wedge barbat(y))$,

$matusa(x, y) = \bigwedge_{p \in A} (parinte(x, p) \wedge frati(p, y) \wedge femeie(y))$ și

$bunic(x, y) = \bigwedge_{p \in A} (parinte(x, p) \wedge parinte(p, y))$;

iar, dacă analizăm definiția predicatului *stramos*, observăm că poate fi scrisă în

modul următor: pentru orice $x, y \in A$, $stramos(x, y, 0) = \begin{cases} \text{true}, & x = y, \\ \text{false}, & x \neq y, \end{cases}$ și,

pentru orice $g \in \mathbb{N}^*$, $stramos(x, y, g) = \bigwedge (parinte(x, p) \wedge stramos(p, y, g - 1))$.

Dacă adoptăm prima soluție, a operațiilor definite parțial, atunci structura algebrică definită de programul de mai sus devine o algebră monosortată, i.e. cu o unică mulțime suport (nu familie de mulțimi ca suport):

(algebră cu unica mulțime de elemente $A \cup \mathbb{N} \cup \{\text{false}, \text{true}\}$)

$\mathcal{A}' = (A' = A \cup \mathbb{N} \cup \{\text{false}, \text{true}\}); stramos, +, frati, parinte, tata, mama, unchi, matusa, bunic, femeie, barbat, ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor, 0, 1, \text{false}, \text{true})$, unde:

- $A = \{ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor\}$, aşadar mulțimea elementelor lui \mathcal{A}' este $A' = \{ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor, \text{false}, \text{true}\} \cup \mathbb{N}$;
- $stramos \subseteq (A')^3 = A' \times A' \times A'$ (relație ternară, predicat ternar), aşadar, ca operație ternară parțială: $stramos : (A')^3 \rightarrow A'$;
- $+ : A' \times A' \rightarrow A'$ (operație binară parțială);
- $frati, parinte, tata, mama, unchi, matusa, bunic \subseteq A' \times A'$ (relații binare, predicate binare), aşadar, ca operații binare parțiale:
 $frati, parinte, tata, mama, unchi, matusa, bunic : A' \times A' \rightarrow A'$;
- $femeie, barbat \subseteq A'$ (relații unare, predicate unare), aşadar, ca operații unare parțiale: $femeie, barbat : A' \rightarrow A'$;
- $ana, carmen, elena, eva, maria, sara, valentina, adam, constantin, eugen, george, ion, iosif, tudor, victor, 0, 1, \text{false}, \text{true} \in A'$ (constante);

operațiile nonnulare (i.e. cu argumente, de aritate nonzero) ale algebrei \mathcal{A}' sunt definite ca în cazul algebrei trisortate \mathcal{A} , și rămân nedefinite în celelalte elemente ale domeniului lor $((A')^2, A' \times A'$ sau A' , după cum sunt ternare, binare, respectiv unare).

Cum răspunde Prologul la următoarea interogare?

?- *stramos(carmen, S, 3), barbat(S).*

Prologul trebuie să găsească valorile variabilei *S* pentru care *stramos(carmen, S, 3) ∧ barbat(S)* = **true**, unde \wedge este operația booleană de conjuncție pe mulțimea **{false, true}** a valorilor booleene predefinite în Prolog. Respectând notația pentru variabile din Prolog și fără a intra în detalii (pentru acestea a se vedea al doilea seminar), Prologul execută următorii pași:
stramos(carmen, S, 3) nu unifică, cu *femeie(X)*, *barbat(X)*, *frati(X, Y)*, *parinte(X, Y)*, *tata(X, Y)*, *mama(X, Y)*, *unchi(X, Y)*, *matusa(X, Y)* sau *bunic(X, Y)*, pentru că predicatul *stramos* nu coincide cu niciunul dintre predicatele *femeie*, *barbat*, *frati*, *parinte*, *tata*, *mama*, *unchi*, *matusa*, *bunic*;
stramos(carmen, S, 3) nu unifică, cu *stramos(X, X, 0)*, pentru că, constantele 0 și 3 nu coincid;
stramos(carmen, S, 3) unifică, cu *stramos(X, S, G)* pentru *X = carmen*, aşadar următorul scop compus de satisfăcut este:

?- *parinte(carmen, P), stramos(P, S, H).*

parinte(carmen, P) unifică numai cu *parinte(carmen, victor)* și *parinte(carmen, valentina)*, pentru *P = victor*, respectiv *P = valentina*. Aceste două clauze apar ca fapte în programul Prolog de mai sus, aşadar nu se intră în alte recursii după aceste unificări, ci se continuă cu *satisfacerea scopurilor*.

stramos(victor, S, H), respectiv *stramos(valentina, S, H)* (desigur, pe rând).

Acestea unifică, cu *stramos(victor, victor, 0)*, respectiv

stramos(valentina, valentina, 0), dar:

- *barbat(valentina)* nu unifică, cu niciunul dintre faptele care definesc predicatul *barbat* și, desigur, cu niciun alt fapt sau membru stâng de regulă,
- în timp ce *barbat(victor)* e satisfăcut, pentru că este unul dintre faptele din acest program, însă $0 + 1 = 1 \neq 3$, aşadar *stramos(carmen, S, 3)* nu unifică, cu *stramos(carmen, victor, 1)*.

Se continuă cu satisfacerea scopurilor *stramos(victor, S, H)* și

stramos(valentina, S, H). Acestea au fost deja unificate cu clauza din faptul

stramos(X, X, 0); acum sunt unificate cu membrul stâng al regulii din definiția predicatului *stramos*.

Și recursia continuă în acest mod. Sunt satisfăcute *parinte(victor, constantin)*, *parinte(victor, maria)*, *parinte(valentina, ion)* și *parinte(valentina, sara)*, precum și *stramos(constantin, constantin, 0)*, *stramos(maria, maria, 0)*, *stramos/ion, ion, 0* și *stramos(sara, sara, 0)*, dar nu și *barbat(maria)* sau *barbat(sara)*, iar $0 + 1 = 1$ și $1 + 1 = 2 \neq 3$.

parinte(maria, P) și *parinte/ion, P)* nu sunt satisfăcute, pentru că acestea nu unifică, cu niciun fapt sau membru stâng al unei reguli.

Dar sunt satisfăcute *parinte(constantin, iosif)*, *parinte(constantin, elena)*, *parinte(sara, adam)* și *parinte(sara, eva)*, precum și *stramos(iosif, iosif, 0)*, *stramos(elena, elena, 0)*, *stramos(adam, adam, 0)* și *stramos(eva, eva, 0)*.



barbat(elena) și *barbat(eva)* nu sunt satisfăcute, dar *barbat(iosif)* și *barbat(adam)* sunt satisfăcute, iar $0 + 1 = 1$, $1 + 1 = 2$ și $2 + 1 = 3$.

Așadar răspunsurile (dacă le cerem pe amândouă, cu ";" / "Next") Prologului la interogarea:

```
?- stramos(carmen, S, 3), barbat(S).
```

vor fi:

$S = iosif$ și $S = adam$.

Cum sunt efectuate unificările de mai sus?

Prologul aplică **algoritmul de unificare**.

Amintesc că și **constantele** sunt **operații**, anume *operații fără argumente*, *operații cu 0 argumente*, *operații de aritate 0*, numite și *operații zeroare sau nulare*, așadar nu trebuie tratate ca un caz separat în cele ce urmează.

Și, referitor la terminologia din secțiunea recapitulativă care deschide acest curs, pentru că **predicalele**, i.e. **relațiile**, sunt considerate OPERAȚII AVÂND REZULTATUL BOOLEAN, **formulele atomice** vor fi considerate tot **termeni**, având OPERAȚIA DOMINANTĂ DE REZULTAT BOOLEAN.

În esență, pentru orice $n, k \in \mathbb{N}$, orice operații (inclusiv și predicalele, relațiile) f și g de arități n , respectiv k și orice termeni (care pot avea operația dominantă de rezultat boolean, i.e. predicat, relație) $t_1, \dots, t_n, u_1, \dots, u_k$:



termenii $f(t_1, \dots, t_n)$ și $g(u_1, \dots, u_k)$ unifică dacă: $\begin{cases} n = k, \\ t_1 \text{ și } u_1 \text{ unifică,} \\ \vdots \\ t_n \text{ și } u_n \text{ unifică,} \end{cases}$

și această recursie continuă până la unificări de termeni cu termeni care sunt:

- fie variabile, iar acestea unifică, cu orice termen care nu le conține,
- fie constante, iar acestea sunt operații fără argumente, aşadar, conform regulii de mai sus, nu unifică decât cu ele însese.

Exemplu

stramos(carmen, S, 3):

- nu unifică, cu $bunic(X, Y)$, pentru că $stramos \neq bunic$;
- nu unifică, cu $stramos(X, X, 0)$, pentru că $3 \neq 0$;
- unifică, cu $stramos(X, X, Y)$ pentru $carmen = X = S$ și $Y = 3$;
- nu unifică, cu $stramos(X, X, X)$ pentru că, dacă ar avea loc această unificare, atunci am avea unificările $carmen = X$, $S = X$ și $3 = X$, deci $X = carmen = S = 3$, aşadar constantele $carmen$ și 3 ar trebui să unifice, dar $carmen \neq 3$.

1 Recapitulare Logica Clasică a Predicatelor

- Calculul clasic cu predicate
- Structuri de ordinul I și limbaje asociate signaturilor lor
- Variabile și substituții
- Sintaxa calculului cu predicate clasic
- Semantica logicii clasice a predicatelor

2 Să exemplificăm noțiunile anterioare într-un program în Prolog

3 Algoritmul de unificare

Să adaptăm cadrul de lucru din prima secțiune a acestui curs la cadrul de lucru pentru programarea în Prolog: vom considera o **structură de ordinul I** \mathcal{A} , fie multisortată, fie cu operațiile parțial definite, și în care relațiile (predicale) sunt considerate operații având rezultatul **true** sau **false**, iar constantele sunt operațiile zeroare, astfel că \mathcal{A} este înzestrată doar cu o familie $(f_i)_{i \in I}$ de operații.

Notație

Pentru fiecare $i \in I$, să notăm cu $\text{dom}(f_i)$ domeniul funcției f_i , adică:

- ① dacă, pentru o mulțime nevidă S , \mathcal{A} este o algebră S -sortată cu mulțimea suport $A = (A_s)_{s \in S}$ (mulțime S -sortată, adică familie de mulțimi indexată de S), atunci, pentru fiecare $i \in I$, există $n_i \in \mathbb{N}$ și $s_{i,1}, \dots, s_{i,n_i}, s_i \in S$ astfel încât $f_i : A_{s_{i,1}} \times \dots \times A_{s_{i,n_i}} \rightarrow A_{s_i}$; atunci $\underline{\text{dom}(f_i)} = A_{s_{i,1}} \times \dots \times A_{s_{i,n_i}}$;
- ② dacă \mathcal{A} este o algebră monosortată cu mulțimea suport A (monosortată, i.e. o unică mulțime) și cu operațiile parțial definite, atunci, pentru fiecare $i \in I$, există $n_i \in \mathbb{N}$ astfel încât $f_i : A^{n_i} \rightarrowtail A$; atunci $\underline{\text{dom}(f_i)}$ este mulțimea elementelor produsului cartezian A^{n_i} în care funcția parțială f_i este definită.

Notăm cu $\text{dom}(\mathcal{A})$ mulțimea elementelor lui \mathcal{A} , indiferent de sort/tip de date, i.e.:

- ① în cazul S -sortat de mai sus, $\text{dom}(\mathcal{A}) = \bigcup_{s \in S} A_s$;
- ② în cazul monosortat de mai sus, $\text{dom}(\mathcal{A}) = A$.

Observație (putem pune tipurile de date laolaltă, la fel ca în Prolog, unde toate sunt înglobate în tipul TERMEN)

Cazul multisortat poate fi înglobat în cazul monosortat cu operații parțiale, astfel: pentru orice $i \in I$, dacă f_i are aritatea $n_i \in \mathbb{N}^*$, atunci considerăm

$$f_i : \text{dom}(\mathcal{A})^{n_i} \multimap \text{dom}(\mathcal{A}),$$

f_i definită numai pe elementele lui $\text{dom}(f_i)$.

În cele ce urmează, vom lucra cu algebre monosortate înzestrate cu operații parțiale.

Definiție

Tipul sau *signatura* structurii algebrice \mathcal{A} este $\tau = (n_i)_{i \in I} \subseteq \mathbb{N}$, unde, pentru fiecare $i \in I$, n_i este aritatea lui f_i .

Pentru fiecare $i \in I$, vom considera un **simbol de operatie** n_i -ară φ_i . Elementele mulțimii $\{\varphi_i \mid i \in I\}$ le considerăm **două căte două distințe**.

Definiție

O *structură algebrică de signură* τ este o algebră $\mathcal{M} = (M; (\varphi_i^{\mathcal{M}})_{i \in I})$, unde, pentru fiecare $i \in I$, $\varphi_i^{\mathcal{M}} : M^{n_i} \multimap M$.

Exemplu

Astfel, algebra \mathcal{A} de mai sus este algebra de signură τ ($\text{dom}(\mathcal{A})$; $(\varphi_i^{\mathcal{A}})_{i \in I}$), unde, pentru fiecare $i \in I$, $\varphi_i^{\mathcal{A}} = f_i$.

Să considerăm și o mulțime infinită Var de **variabile**, ale cărei elemente le considerăm **două câte două distințe**.

De asemenea, considerăm mulțimea de variabile *disjunctă* de cea a simbolurilor de operații: $Var \cap \{\varphi_i \mid i \in I\} = \emptyset$.

Deocamdată, vom considera doar **alfabetul** format din **variabile**, **simboluri de operații**, virgula și parantezele rotunde: $Var \cup \{\varphi_i \mid i \in I\} \cup \{,\} \cup \{(,)\}$.

Definiție și notație (termenii peste alfabetul de mai sus)

Considerăm următoarea mulțime de cuvinte finite și nevide peste alfabetul de mai sus: $Term \subseteq (Var \cup \{\varphi_i \mid i \in I\} \cup \{,\} \cup \{(,)\})^+ = \{a_1 \dots a_n \mid n \in \mathbb{N}^*, a_1, \dots, a_n \in Var \cup \{\varphi_i \mid i \in I\} \cup \{,\} \cup \{(,)\}\}$, ale cărei elemente le vom numi *termeni*, definită recursiv în modul următor:

- ① $Var \subseteq Term$ (variabilele sunt termeni de lungime 1, i.e. constând dintr-o singură literă: variabila respectivă);
- ② pentru orice $i \in I$, dacă $n_i = 0$ (adică φ_i este un *simbol de operație zeroară*, un *simbol de constantă*, atunci $\varphi_i \in Term$ (simbolurile de constante sunt termeni, tot de lungime 1, constând dintr-o singură literă, anume acel simbol de constantă));
- ③ pentru orice $i \in I$, dacă $n_i \in \mathbb{N}^*$ (adică φ_i este un simbol de operație cu $n_i \neq 0$ argumente), atunci, pentru orice $t_1, \dots, t_{n_i} \in Term$, rezultă că $\varphi_i(t_1, \dots, t_{n_i}) \in Term$ (orice simbol de operație cu argumente aplicată unor termeni are drept rezultat un termen); termenii formați în acest al treilea mod se numesc *termeni compuși*.

Observație

Cum variabilele și simbolurile de operații sunt două câte două distincte, rezultă că orice termen are o unică scriere ca:

- variabilă,
- simbol de constantă sau
- termen compus, de forma $\varphi(t_1, \dots, t_n)$, cu simbolul de operație φ , $n \in \mathbb{N}^*$ și termenii t_1, \dots, t_n unic determinați.

Definiție și notație

Pentru orice termen $t \in Term$, notăm cu $V(t)$ mulțimea variabilelor care apar în t , definită, recursiv, astfel:

- oricare ar fi $v \in Var$, $V(v) = \{v\}$;
- pentru orice $i \in I$ cu $n_i = 0$, $V(\varphi_i) = \emptyset$ (i.e. simbolurile de constante nu au variabile);
- pentru orice $i \in I$ cu $n_i \in \mathbb{N}^*$ și orice termeni $t_1, \dots, t_{n_i} \in Term$,
 $V(\varphi_i(t_1, \dots, t_{n_i})) = V(t_1) \cup \dots \cup V(t_{n_i})$.

Definiție (substituțiile dău variabilelor valori în mulțimea termenilor)

O substituție este o funcție $\sigma : Var \rightarrow Term$.

Notație

Dacă $n \in \mathbb{N}^*$, $\{v_1, \dots, v_n\} \subseteq Var$ este o mulțime finită de variabile, iar $\{t_1, \dots, t_n\} \subseteq Term$ este o mulțime de termeni, atunci se notează cu $\{v_1/t_1, \dots, v_n/t_n\}$ următoarea substituție:

$\{v_1/t_1, \dots, v_n/t_n\} : Var \rightarrow Term$, pentru orice $v \in Var$,

$$\{v_1/t_1, \dots, v_n/t_n\}(v) = \begin{cases} t_i, & (\exists i \in \overline{1, n}) (v = v_i), \\ v, & v \in Var \setminus \{v_1, \dots, v_n\}. \end{cases}$$

Definiție și notație

Fie $\sigma : Var \rightarrow Term$ o substituție. Următoarea extindere a lui σ la întreaga mulțime a termenilor se numește tot *substituție* (și, de obicei, se notează tot cu σ): $\tilde{\sigma} : Term \rightarrow Term$, definită, recursiv, astfel:

- pentru orice $v \in Var$, $\tilde{\sigma}(v) = \sigma(v)$ (adică $\tilde{\sigma}|_{Var} = \sigma$: pe variabile, $\tilde{\sigma}$ coincide cu σ);
- pentru orice $i \in I$ astfel încât $n_i = 0$, $\tilde{\sigma}(\varphi_i) = \varphi_i$ (adică, pe simbolurile de constante, $\tilde{\sigma}$ este funcția identică);
- pentru orice $i \in I$ astfel încât $n_i \in \mathbb{N}^*$ și orice termeni $t_1, \dots, t_{n_i} \in Term$ (astfel încât $\tilde{\sigma}$ a fost definită în t_1, \dots, t_{n_i}),
 $\tilde{\sigma}(\varphi_i(t_1, \dots, t_{n_i})) = \varphi_i(\tilde{\sigma}(t_1), \dots, \tilde{\sigma}(t_{n_i}))$.

Extinderea \tilde{s} din definiția anterioară este:

- **complet definită**, pentru că toți termenii se scriu ca mai sus, i.e. sunt obținuți prin acea recursie;
- **corect definită**, pentru că orice termen are o unică scriere ca mai sus.

Definiție (să dăm variabilelor valori într-o algebră de signatură τ , apoi să calculăm valorile tuturor termenilor pe baza celor date variabilelor – reținem această definiție pentru mai târziu)

Pentru orice algebră $\mathcal{M} = (M; (\varphi_i^{\mathcal{M}})_{i \in I})$ de signatură τ , o funcție $s : Var \rightarrow M$ va fi numită *interpretare*.

Următoarea prelungire a lui s la mulțimea *Term* a tuturor termenilor se numește tot *interpretare* și se notează, de obicei, tot cu s : $\tilde{s} : Term \rightarrow M$, definită, recursiv, astfel:

- pentru orice $v \in Var$, $\tilde{s}(v) = s(v)$ (i.e. $\tilde{s}|_{Var} = s$);
- pentru orice $i \in I$ cu $n_i = 0$, $\tilde{s}(\varphi_i) = \varphi_i^{\mathcal{M}}$ (valoarea lui \tilde{s} într-un simbol de constantă este constanta din \mathcal{M} corespunzătoare acelui simbol de constantă);
- pentru orice $i \in I$ cu $n_i \in \mathbb{N}^*$ și orice $t_1, \dots, t_{n_i} \in Term$ astfel încât (\tilde{s} a fost definită în termenii t_1, \dots, t_{n_i} și) $(\tilde{s}(t_1), \dots, \tilde{s}(t_{n_i})) \in dom(\varphi_i^{\mathcal{M}})$,
 $\tilde{s}(\varphi_i(t_1, \dots, t_{n_i})) = \varphi_i^{\mathcal{M}}(\tilde{s}(t_1), \dots, \tilde{s}(t_{n_i})).$

Algoritm pentru rezolvarea unei probleme de unificare

Definiție (ce înseamnă a unifica mai mulți termeni)

Fie $k \in \mathbb{N} \setminus \{0, 1\}$ și $t_1, \dots, t_k \in \text{Term}$. Spunem că termenii t_1, \dots, t_k unifică dacă există o substituție $\sigma : \text{Term} \rightarrow \text{Term}$, numită *unificator* pentru t_1, \dots, t_k , cu proprietatea că $\sigma(t_1) = \dots = \sigma(t_k)$.

Fixăm un număr natural $k \geq 2$ și k termeni $t_1, \dots, t_k \in \text{Term}$.

Cerința de a determina dacă termenii t_1, \dots, t_k unifică și, în caz afirmativ, a determina și un unificator pentru acești termeni este o *problemă de unificare*.

Observație

În cazul în care t_1, \dots, t_k unifică, următorul algoritm obține, într-un număr finit de pași, *un cel mai general unificator* pentru t_1, \dots, t_k , adică un unificator $\mu : \text{Term} \rightarrow \text{Term}$ cu proprietatea că orice unificator $\sigma : \text{Term} \rightarrow \text{Term}$ pentru t_1, \dots, t_k este de forma $\sigma = \lambda \circ \mu$ pentru o substituție $\lambda : \text{Term} \rightarrow \text{Term}$.

În cazul în care t_1, \dots, t_k nu unifică, algoritmul următor se termină într-un număr finit de pași cu ESEC, i.e. fără a găsi un unificator.

(Algoritmul de unificare)

Vom reține două liste (mulțimi) de ecuații (egalități, probleme de unificare între câte doi termeni):

o listă soluție S și o listă de rezolvat R .

Inițial:

- $S = \emptyset$;
- $R = \{t_1 = t_2, t_2 = t_3, \dots, t_{k-1} = t_k\}$.

Se aplică, pe rând, următorii pași, în orice ordine posibilă:

- SCOATERE (ELIMINARE, ȘTERGERE): orice ecuație de forma $t = t$ (i.e. între un termen și el însuși) este eliminată din lista R ;
- DESCOMPUNERE: orice ecuație de forma $\varphi_i(u_1, \dots, u_{n_i}) = \varphi_i(w_1, \dots, w_{n_i})$, unde $i \in I$ astfel încât $n_i > 0$ și $u_1, \dots, u_{n_i}, w_1, \dots, w_{n_i} \in \text{Term}$ din lista R este înlocuită cu următoarele n_i ecuații: $u_1 = w_1, \dots, u_{n_i} = w_{n_i}$;
- REZOLVARE: orice ecuație din R de forma $v = t$ sau $t = v$, unde $v \in \text{Var}$ și $t \in \text{Term}$, este scoasă din R și introdusă în lista soluție S sub forma $v = t$ (dacă și $t \in \text{Var}$, atunci nu contează cum orientăm ecuația: putem alege pe oricare dintre variabile ca membru stâng), apoi toate aparițiile lui v în termenii care apar în ecuațiile din lista de rezolvat R și din lista soluție S se înlocuiesc cu t , adică toți acești termeni u se înlocuiesc cu $\{v/t\}(u)$.

Algoritmul se ÎNCHEIE în oricare dintre cazurile următoare:

- ① dacă, după execuția unui pas de SCOATERE sau REZOLVARE, obținem $R = \emptyset$, i.e. lista de rezolvat devine vidă, iar lista soluție curentă este $S = \{v_1 = u_1, \dots, v_n = u_n\}$, atunci **un (cel mai general) unificator** pentru t_1, \dots, t_k este substituția $\{v_1/u_1, \dots, v_n/u_n\}$: IEȘIRE CU SUCCES;
- ② dacă, în cursul execuției pașilor de mai sus, apare în lista de rezolvat R :
 - fie o ecuație de forma $v = t$, cu $t \in \text{Term}$ și $v \in V(t)$ (i.e. cu variabila v apărând în t),
 - fie o ecuație de forma $\varphi_i(u_1, \dots, u_{n_i}) = \varphi_j(w_1, \dots, w_{n_j})$, cu $i, j \in I$, $n_i, n_j \in \mathbb{N}$ (nu neapărat nenule), $u_1, \dots, u_{n_i}, w_1, \dots, w_{n_j} \in \text{Term}$ și $i \neq j$, așadar cu simbolurile de operații dominante ale termenilor din cei doi membri diferite: $\varphi_i \neq \varphi_j$,atunci se IESE CU ESEC: nu s-a găsit niciun unificator, așadar **termenii t_1, \dots, t_k nu unifică**.

Exercițiu (temă pentru seminar)

Să considerăm două simboluri de operații binare distințe f și g , unul de operație unară h , două simboluri de constante diferite a și b și patru variabile $V, X, Y, Z \in \text{Var}$, două câte două distințe.

Considerăm următorii termeni formați cu simbolurile de operații și variabilele de mai sus: $r, s, t, u, w \in \text{Term}$,

Exercițiu (continuare: termenii de unificat, dați prin expresiile lor și prin arborii asociați acestor expresii)

$$r = f(h(h(a)), g(g(X, h(Y)), X)),$$

$$s = f(h(X), g(g(X, X), h(Y))),$$

$$t = f(h(V), g(Z, Z)),$$

$$u = f(h(h(Z)), g(g(X, h(b)), h(Z))),$$

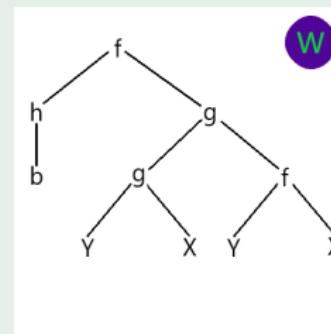
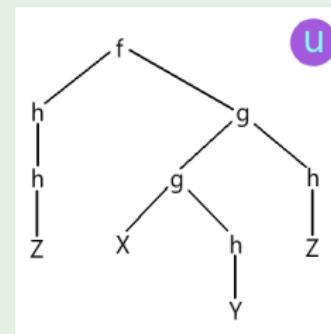
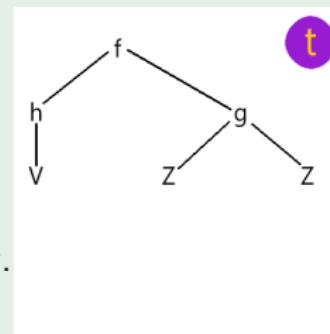
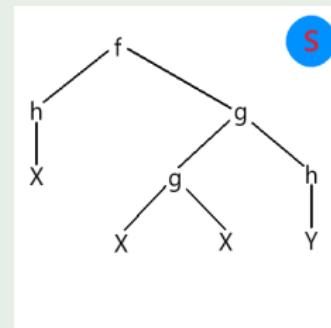
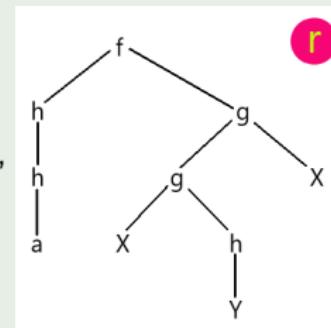
$$w = f(h(b), g(g(V, Z), f(V, Z))).$$

Să se unifice acești termeni doi căte doi, adică să se rezolve, pe rând, problemele de unificare:

$$\begin{aligned} r &= s, r = t, \\ r &= u, r = w, \end{aligned}$$

$$\begin{aligned} s &= t, s = u, \\ s &= w, t = u, \end{aligned}$$

$$\begin{aligned} t &= w, u = w. \end{aligned}$$



Pentru perechile $\{p, q\}$ de termeni $p, q \in \{r, s, t, u, w\}$ care unifică, să se unifice și reuniunea tuturor acestor perechi.

Pentru **rezolvare**, a se vedea SEMINARUL.

PROGRAMARE LOGICĂ

Cursurile IX, X, XI, XII, XIII, XIV

Claudia MUREŞAN
cmuresan@fmi.unibuc.ro, c.muresan@yahoo.com

Universitatea din Bucureşti
Facultatea de Matematică şi Informatică
Bucureşti

2019–2020, Semestrul II

Cuprinsul acestui curs

- 1 Să reluăm noțiunile din Logica Clasică a Predicatelor care stau la baza funcționării Prologului
- 2 Cu ce fel de signuri și structuri algebrice de acele signuri și cu ce tipuri/cazuri particulare de formule lucrează limbajul Prolog?
- 3 Modele pentru formule, satisfiabilitate în Logica Clasică a Predicatelor
- 4 Rezoluția în Logica Clasică a Predicatelor – regula de deducție care stă la baza funcționării limbajului Prolog
- 5 Rezoluția în Prolog

- 1 Să reluăm noțiunile din Logica Clasică a Predicatelor care stau la baza funcționării Prologului
- 2 Cu ce fel de signaturi și structuri algebrice de acele signaturi și cu ce tipuri/cazuri particulare de formule lucrează limbajul Prolog?
- 3 Modele pentru formule, satisfiabilitate în Logica Clasică a Predicatelor
- 4 Rezoluția în Logica Clasică a Predicatelor – regula de deducție care stă la baza funcționării limbajului Prolog
- 5 Rezoluția în Prolog

Structuri algebrice cu operații parțiale și relații

Definiție

Vom numi *structură de ordinul I* o structură algebrică de forma

$$\mathcal{A} = (A; (f_i)_{i \in I}; (R_j)_{j \in J}; (c_k)_{k \in K}),$$

unde:

- A este o mulțime nevidă, numită *universul structurii* \mathcal{A} sau *mulțimea elementelor structurii algebrice* \mathcal{A} ;
- I, J, K sunt mulțimi oarecare de indici (care pot fi și vide);
- pentru fiecare $i \in I$, există $n_i \in \mathbb{N}^*$, a. î. $f_i : A^{n_i} \rightarrow A$ (f_i este o operație parțială pe A cu n_i argumente, adică n_i -ară); pentru fiecare $i \in I$, notăm cu $dom(f_i)$ mulțimea n_i -uplurilor din A^{n_i} în care f_i este definită;
- pentru fiecare $j \in J$, există $m_j \in \mathbb{N}^*$, a. î. $R_j \subseteq A^{m_j}$ (R_j este o *relație* m_j -ară pe A);
- pentru fiecare $k \in K$, $c_k \in A$ (c_k este o constantă din A).

De obicei, operațiilor și relațiilor din componența lui \mathcal{A} li se atașează indicele \mathcal{A} , pentru a le deosebi de simbolurile de operații și relații din limbajul pe care îl vom construi în continuare; astfel, structura de ordinul I de mai sus mai se notează sub forma: $\mathcal{A} = (A, (f_i^{\mathcal{A}})_{i \in I}, (R_j^{\mathcal{A}})_{j \in J}, (c_k^{\mathcal{A}})_{k \in K})$.

Vom avea clase de structuri de același **tip**, iar acestor **tipuri** de structuri algebrice le vom asocia **limbaje** ale logicii clasice a predicatelor

Definiție

Tipul sau signatura structurii de ordinul I \mathcal{A} din definiția anterioară este tripletul de familii de numere naturale: $\tau = ((n_i)_{i \in I}; (m_j)_{j \in J}; (0)_{k \in K})$.

Orice structură de forma lui \mathcal{A} de mai sus se numește *structură de ordinul I de tipul (sau signatura) τ* .

- Fiecărei signaturi τ a unei structuri de ordinul I (fiecărei clase de structuri de ordinul I de o anumită signură τ) i se asociază un limbaj, numit **limbaj de ordinul I** și notat, de obicei, cu \mathcal{L}_τ , în care pot fi exprimate proprietățile algebrice ale structurilor de ordinul I de signura τ .
- Să considerăm o signură $\tau = ((n_i)_{i \in I}; (m_j)_{j \in J}; (0)_{k \in K})$, pe care o fixăm.

Alfabetul limbajului de ordinul I \mathcal{L}_τ este format din următoarele **simboluri primitive**:

- o mulțime infinită de **variabile**: $Var = \{x, y, z, u, v, \dots\}$;
- **simboluri de operații**: $(f_i)_{i \in I}$; pentru fiecare $i \in I$, numărul natural nenul n_i se numește *aritatea* lui f_i ;

Limbajul asociat unei signaturi

- **simboluri de relații (simboluri de predicate)**: $(R_j)_{j \in J}$; pentru fiecare $j \in J$, numărul natural nenul m_j se numește *aritatea* lui R_j ;
- **simboluri de constante**: $(c_k)_{k \in K}$;
- **simbolul de egalitate**: $=$ (un semn egal îngroșat) (*a nu se confunda cu egalul simplu!*);
- **conectorii logici primitivi**: \neg (*negația*), \rightarrow (*implicația*);
- **cuantificatorul universal**: \forall (*oricare ar fi*)
- paranteze: $(,)$, $[,]$, precum și virgula.

Pentru comoditate, vom spune uneori: “operații”, “relații” / “predicate” și “constante” în loc de “simboluri de operații”, “ simboluri de relații/predicate” și “simboluri de constante”, respectiv.

Observație

Majoritatea autorilor consideră virgula ca având semnificație implicită, subînțeleasă, în scrierea termenilor și a formulelor atomice, și nu includ virgula în limbajul \mathcal{L}_τ .

Parantezele care încadrează formule (nu argumentele unei funcții sau relații) au același rol ca în sintaxa logicii propoziționale.

Termeni: variabile, constante, sau funcții cu argumente aplicate unor termeni, recursiv

Definiție

Termenii limbajului \mathcal{L}_τ sunt cuvintele finite (i.e. de lungime finită, adică formate dintr-un număr finit de litere) peste alfabetul de mai sus (i.e. ale căror litere sunt printre simbolurile de mai sus) definite, recursiv, astfel:

- ① variabilele și simbolurile de constante sunt termeni;
- ② dacă $n \in \mathbb{N}^*$, f este un simbol de operație n -ară și t_1, \dots, t_n sunt termeni, atunci $f(t_1, \dots, t_n)$ este un termen;
 f se numește *operația dominantă* sau *operatorul dominant* al termenului $f(t_1, \dots, t_n)$.

Cum termenii sunt cuvinte de lungime finită peste alfabetul de mai sus, rezultă că orice termen se obține prin aplicarea regulilor ① și ② de un număr finit de ori.

Notație

Vom nota cu $\text{Term}(\mathcal{L}_\tau)$ mulțimea termenilor din limbajul \mathcal{L}_τ .

Formule atomice: relații aplicate unor termeni

Definiție

Formulele atomice ale limbajului \mathcal{L}_τ sunt cuvintele finite peste alfabetul de mai sus definite astfel:

- ① dacă $t_1, t_2 \in \text{Term}(\mathcal{L}_\tau)$, atunci $t_1 = t_2$ este o formulă atomică;
- ② dacă R este un simbol de relație m -ară și $t_1, \dots, t_m \in \text{Term}(\mathcal{L}_\tau)$, atunci $R(t_1, \dots, t_m)$ este o formulă atomică.

Cum formulele atomice au lungimi finite, rezultă că orice formulă atomică se obține prin aplicarea regulilor ① și ② de un număr finit de ori

Formule: obținute din formulele atomice prin aplicarea conectorilor logici și a cuantificatorilor

Definiție

Formulele limbajului \mathcal{L}_τ sunt cuvintele finite peste alfabetul de mai sus definite, recursiv, astfel:

- ① formulele atomice sunt formule;
- ② dacă φ este o formulă, atunci $\neg\varphi$ este o formulă;
- ③ dacă φ și ψ sunt formule, atunci $\varphi \rightarrow \psi$ este o formulă;
- ④ dacă φ este o formulă și x este o variabilă, atunci $\forall x\varphi$ este o formulă.

Cum formulele au lungimi finite, rezultă că orice formulă se obține prin aplicarea regulilor ①, ②, ③ și ④ de un număr finit de ori.

Notație

Se notează cu $Form(\mathcal{L}_\tau)$ mulțimea formulelor limbajului \mathcal{L}_τ .

Formule fără cuantificatori: obținute din formulele atomice prin aplicarea conectorilor logici

Putem defini astfel **formulele fără cuantificatori**:

Definiție

Formulele fără cuantificatori ale limbajului \mathcal{L}_τ sunt cuvintele finite peste alfabetul de mai sus definite, recursiv, astfel:

- ① formulele atomice sunt formule fără cuantificatori;
- ② dacă φ este o formulă fără cuantificatori, atunci $\neg\varphi$ este o formulă fără cuantificatori;
- ③ dacă φ și ψ sunt formule fără cuantificatori, atunci $\varphi \rightarrow \psi$ este o formulă fără cuantificatori.

Cum formulele fără cuantificatori au lungimi finite, rezultă că orice formulă fără cuantificatori se obține prin aplicarea regulilor ①, ② și ③ de un număr finit de ori.

Ceilalți conectori logici și celălalt cuantificator

Notație

Introducem abrevierile: pentru orice formule φ, ψ și orice variabilă x :

- **conectorii logici derivați** \vee (*disjuncția*), \wedge (*conjuncția*) și \leftrightarrow (*echivalența*) se definesc astfel:

$$\varphi \vee \psi = \neg \varphi \rightarrow \psi$$

$$\varphi \wedge \psi = \neg (\varphi \rightarrow \neg \psi)$$

$$\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

- **cuantificatorul existențial** \exists (*există*) se definește astfel:

$$\exists x \varphi = \neg \forall x \neg \varphi$$

Observație (convenție privind scrierea conectorilor logici, a cuantificatorilor și a simbolului de egalitate)

- \neg, \forall, \exists vor avea prioritate mai mare;
- $\rightarrow, \vee, \wedge, \leftrightarrow, =$ vor avea prioritate mai mică.

Variabilele care apar într-un termen

Notație (mulțimile din această notație vor fi definite recursiv mai jos)

Pentru orice termen t și orice formulă φ , notăm:

- $V(t) =$ mulțimea variabilelor termenului t
- $FV(\varphi) =$ mulțimea variabilelor *libere* ale formulei φ
- $V(\varphi) =$ mulțimea variabilelor formulei φ

Definiție

Pentru orice termen t :

- dacă $t = x$, unde x este o variabilă, atunci $V(t) = \{x\}$
- dacă $t = c$, unde c este o constantă, atunci $V(t) = \emptyset$
- dacă $n \in \mathbb{N}^*$ și $t = f(t_1, \dots, t_n)$, unde f este un simbol de operație n -ară și t_1, \dots, t_n sunt termeni, atunci $V(t) = \bigcup_{i=1}^n V(t_i)$

Notație ($t = t(x_1, \dots, x_n)$)

Dacă t este un termen și $V(t) = \{x_1, \dots, x_n\}$, cu $n \in \mathbb{N}^*$, atunci termenul t se mai scrie și sub forma $t(x_1, \dots, x_n)$.

Variabilele și variabilele libere dintr-o formulă

Definiție

Pentru orice formulă φ :

- dacă $\varphi = (t_1 = t_2)$, unde t_1 și t_2 sunt termeni, atunci $V(\varphi) = FV(\varphi) = V(t_1) \cup V(t_2)$
- dacă $\varphi = R(t_1, \dots, t_m)$, unde R este un simbol de relație m -ară și t_1, \dots, t_m sunt termeni, atunci $V(\varphi) = FV(\varphi) = \bigcup_{i=1}^m V(t_i)$
- dacă $\varphi = \neg \psi$, pentru o formulă ψ , atunci $V(\varphi) = V(\psi)$ și $FV(\varphi) = FV(\psi)$
- dacă $\varphi = \psi \rightarrow \chi$, pentru două formule ψ, χ , atunci $V(\varphi) = V(\psi) \cup V(\chi)$ și $FV(\varphi) = FV(\psi) \cup FV(\chi)$
- dacă $\varphi = \forall x \psi$, pentru o formulă ψ și o variabilă x , atunci $V(\varphi) = V(\psi)$, iar $FV(\varphi) = FV(\psi) \setminus \{x\}$

Notație ($\varphi = \varphi(x_1, \dots, x_n)$)

Dacă φ este o formulă și, pentru un $n \in \mathbb{N}^*$, $V(\varphi) \subseteq \{x_1, \dots, x_n\}$, atunci formula φ se mai scrie și sub forma $\varphi(x_1, \dots, x_n)$.

Variabilele și variabilele libere dintr-o formulă

Remarcă

Este imediat, din definiția anterioară și definiția conectorilor logici derivați și a cuantificatorului existențial, că, pentru orice formule ψ, χ și orice variabilă x :

- $V(\psi \vee \chi) = V(\psi \wedge \chi) = V(\psi \leftrightarrow \chi) = V(\psi) \cup V(\chi)$ și
 $FV(\psi \vee \chi) = FV(\psi \wedge \chi) = FV(\psi \leftrightarrow \chi) = FV(\psi) \cup FV(\chi)$
- $V(\exists x\psi) = V(\psi)$, iar $FV(\exists x\psi) = FV(\psi) \setminus \{x\}$

Definiție

Pentru orice variabilă x și orice formulă φ :

- dacă $x \in FV(\varphi)$, atunci x se numește *variabilă liberă a lui φ* ;
- dacă $x \in V(\varphi) \setminus FV(\varphi)$, atunci x se numește *variabilă legată a lui φ* ;
- dacă $FV(\varphi) = \emptyset$ (i. e. φ nu are variabile libere), atunci φ se numește *enunț*.

Remarcă

Formulele fără cuantificatori sunt exact formulele φ cu $V(\varphi) = FV(\varphi)$, adică exact formulele fără variabile legate, i.e. formulele în care toate variabilele sunt libere (adică nu sunt cuantificate, nu li se aplică niciun cuantificator).

- 1 Să reluăm noțiunile din Logica Clasică a Predicatelor care stau la baza funcționării Prologului
- 2 Cu ce fel de signaturi și structuri algebrice de acele signaturi și cu ce tipuri/cazuri particulare de formule lucrează limbajul Prolog?
- 3 Modele pentru formule, satisfiabilitate în Logica Clasică a Predicatelor
- 4 Rezoluția în Logica Clasică a Predicatelor – regula de deducție care stă la baza funcționării limbajului Prolog
- 5 Rezoluția în Prolog

În primul rând, ce tip de signatură avem în Prolog și cu ce fel de structuri algebrice de acea signatură lucrăm?

Am observat că avem nevoie de **operații parțiale**, fiecare având în *domeniul de definiție* elemente de un anumit **tip de date** (sau de mai multe tipuri de date). Toate tipurile de date sunt înglobate în **tipul termen**, adică sunt **subtipuri** ale tipului termen.

Să considerăm o algebră $\mathcal{A} = (A; (f_i)_{i \in I}; (R_j)_{j \in J}; (c_k)_{k \in K})$ de o signatură $\tau = ((n_i)_{i \in I}; (m_j)_{j \in J}; (0)_{k \in K})$.

Cum trebuie să arate algebra \mathcal{A} ca să corespundă unui program în Prolog?

Predicalele vor fi reprezentate prin **relațiile** din familia de relații $(R_j)_{j \in J}$. Știm că, în Prolog, putem imbrica predicalele, aşadar avem nevoie de următoarele:

- mulțimea A trebuie să conțină constantele booleene **true** și **false**;
- o parte dintre operațiile parțiale din familia $(f_i)_{i \in I}$ sunt asociate relațiilor din familia $(R_j)_{j \in J}$, astfel:

considerăm $J \subseteq I$;

pentru fiecare $j \in J$, avem: $n_j = m_j$ și $f_j : A^{n_j} \rightarrow A$ (cu

$Im(f_j) \subseteq \{\text{true}, \text{false}\}$, unde, prin **definiție**, imaginea funcției parțiale f_j coincide cu imaginea funcției $f_j|_{dom(f_j)}: dom(f_j) \rightarrow A$, definită prin: oricare ar fi $(a_1, \dots, a_{n_j}) \in dom(f_j)$,

$$f(a_1, \dots, a_{n_j}) = \begin{cases} \text{true, dacă } (a_1, \dots, a_{n_j}) \in R_j; \\ \text{false, altfel;} \end{cases}$$

În practică, vom nota funcțiile f_i la fel ca pe relațiile (predicale) R_j :

- iar **conectorii logici** fac parte dintre operațiile din familia $(f_i)_{i \in I}$, adică există $i_1, i_2, i_3, i_4, i_5 \in I$, astfel încât:

$$n_{i_1} = 1, \text{ dom}(f_{i_1}) = \{\text{true}, \text{false}\} \text{ și } \begin{cases} f_{i_1}(\text{true}) = \text{false} \text{ și} \\ f_{i_1}(\text{false}) = \text{true}; \end{cases}$$

$$n_{i_2} = n_{i_3} = n_{i_4} = n_{i_5} = 2,$$

$$\text{dom}(f_2) = \text{dom}(f_3) = \text{dom}(f_4) = \text{dom}(f_5) = \{\text{true}, \text{false}\}^2 \text{ și:}$$

$$\begin{cases} f_{i_2}(\text{true}, \text{false}) = \text{false} \text{ și} \\ f_{i_2}(x, y) = \text{true}, \text{ dacă } (x, y) \in \{\text{true}, \text{false}\}^2 \setminus \{\text{true}, \text{false}\}; \end{cases}$$

$$\text{pentru orice } (x, y) \in \{\text{true}, \text{false}\}^2, f_{i_3} = f_{i_2}(f_{i_1}(x), y);$$

$$\text{pentru orice } (x, y) \in \{\text{true}, \text{false}\}^2, f_{i_4} = f_{i_1}(f_{i_2}(x, f_{i_1}(y)));$$

$$\text{pentru orice } (x, y) \in \{\text{true}, \text{false}\}^2, f_{i_5} = f_{i_4}(f_{i_2}(x, y), f_{i_2}(y, x)).$$

Vom nota funcțiile de mai sus la fel ca pe conectorii logici cărora le corespund:

$f_{i_1} = \neg$, $f_{i_2} = \rightarrow$, $f_{i_3} = \vee$, $f_{i_4} = \wedge$ și $f_{i_5} = \leftrightarrow$, cu aceeași regulă a priorităților pentru a evita parantezări excesive și cu scriere infixată pentru f_{i_2} , f_{i_3} , f_{i_4} și f_{i_5} , la fel ca în cazul conectorilor logici.

Pentru orice operație f cu $\text{Im}(f) \subseteq \{\text{true}, \text{false}\}$, compunerea $f_{i_1} \circ f$ va fi notată, simplu, $\neg f$. La fel pentru f_{i_2} , f_{i_3} , f_{i_4} , f_{i_5} aplicate unor termeni cu alți operatori dominanți.

Cu ce tipuri/cazuri particulare de formule lucrează Prologul?

De exemplu, Prologul acceptă următoarea bază de cunoștințe:

$b(1)$. % corespunde formulei fără variabile: $b(1)$

$a(0)$. % corespunde formulei fără variabile: $a(0)$

$a(false)$. % corespunde formulei fără variabile: $a(false)$

$a(b(_))$. % corespunde formulei: $\forall X a(b(X))$

Dar la interogarea:

?- $a(not(b(1)))$. % corespunde formulei fără variabile: $a(\neg(b(1)))$

răspunsul este **false**, deși $not(b(1))$ este evaluat la **false**, adică operația (mai precis compunerea de operații) $\neg b$ ia în argumentul 1 valoarea **false**; răspunsul la interogarea de mai sus este **false** pentru că termenul $not(b(1))$, de operator dominant *not* (adică \neg cu notația de mai sus), nu unifică, cu niciunul dintre termenii 0, *false* și $b(X)$, unde $X \in Var$ (termenii care apar ca argumente ale lui *a* în faptele de mai sus care definesc predicatul *a*).

În schimb, dacă adăugăm la baza de cunoștințe regula:

$c(X) :- not(X)$. % corespunde formulei care poate fi scrisă astfel pentru a % concorda cu teoria anterioară: $\forall X (\neg(X=true) \rightarrow c(X))$

atunci la interogarea:

?- $c(not(b(1)))$. % corespunde formulei fără variabile: $c(\neg(b(1)))$

Prologul răspunde **true**, pentru că X și $not(b(1))$ unifică, iar $b(1)$ e satisfăcut, prin urmare $not(b(1))$ nu e satisfăcut, aşadar $not(not((b(1)))$ (adică $not(X)$) pentru X luând valoarea $not(b(1))$ este satisfăcut, prin urmare $c(not(b(1)))$ (adică $c(X)$ pentru X luând valoarea $not(b(1))$) e satisfăcut.

Observăm că Prologul nu acceptă fapte sau membri stângi de reguli conținând conectori logici. De exemplu, nu acceptă regula sau fapta următoare:

```
a(X), b(X) :- a(b(X)).  
a(X); b(X).
```

În schimb, acceptă fapte sau reguli de forma:

```
d(a(X), b(X)).          % corespunde formulei:  $\forall X d(a(X) \wedge b(X))$   
e(a(X); b(X)) :- a(b(X)). % corespunde formulei:  $\forall X [a(b(X)) \rightarrow e(a(X) \vee b(X))]$ 
```

Cu notațiile de operații de mai sus, argumentele predicatelor (relațiilor) d , respectiv e sunt **termenii** $a(X) \wedge b(X)$, respectiv $a(X) \vee b(X)$. Fă A se vedeă și fișierul *testfaptemreguli.pl*; vom reveni la modul în care Prologul răspunde interogărilor. Așadar:

Cu ce tipuri/cazuri particulare de formule lucrează Prologul?

Faptele corespund unor formule de tipul:

$|\forall x_1 \forall x_2 \dots \forall x_n \varphi|$, unde $n \in \mathbb{N}^*$ și:

- φ este o formulă atomică;
- $V(\varphi) = FV(\varphi) = \{x_1, x_2, \dots, x_n\}$, așadar $\forall x_1 \forall x_2 \dots \forall x_n \varphi$ este un enunț.

Regulile corespund unor formule de tipul:

$|\forall x_1 \forall x_2 \dots \forall x_n (\varphi \rightarrow \psi)|$, unde $n \in \mathbb{N}^*$ și:

- φ este o formulă fără cuantificatori (φ este membrul drept al regulii);
- ψ este o formulă atomică (ψ este membrul stâng al regulii);
- așadar $V(\varphi \rightarrow \psi) = V(\varphi) \cup V(\psi) = FV(\varphi) \cup FV(\psi) = FV(\varphi \rightarrow \psi)$, deci $\varphi \rightarrow \psi$ este o formulă fără cuantificatori;
- $V(\varphi \rightarrow \psi) = \{x_1, x_2, \dots, x_n\}$, așadar $\forall x_1 \forall x_2 \dots \forall x_n (\varphi \rightarrow \psi)$ este un enunț.

Interogările corespund unor formule de tipul:

$|\exists x_1 \exists x_2 \dots \exists x_n \varphi|$, unde $n \in \mathbb{N}^*$ și:

- φ este o formulă fără cuantificatori;
- $V(\varphi) = FV(\varphi) = \{x_1, x_2, \dots, x_n\}$, așadar $\exists x_1 \exists x_2 \dots \exists x_n \varphi$ este un enunț.

- 1 Să reluăm noțiunile din Logica Clasică a Predicatelor care stau la baza funcționării Prologului
- 2 Cu ce fel de signaturi și structuri algebrice de acele signaturi și cu ce tipuri/cazuri particulare de formule lucrează limbajul Prolog?
- 3 Modele pentru formule, satisfiabilitate în Logica Clasică a Predicatelor
- 4 Rezoluția în Logica Clasică a Predicatelor – regula de deducție care stă la baza funcționării limbajului Prolog
- 5 Rezoluția în Prolog

Fie:

- $\tau = ((n_i)_{i \in I}; (m_j)_{j \in J}; (0)_{k \in K})$ o signatură;
- $\mathcal{A} = (A; (f_i)_{i \in I}; (R_j)_{j \in J}; (c_k)_{k \in K})$ o structură de ordinul I de signatură τ ;

notăm:
$$\begin{cases} \mathcal{F} = \{f_i \mid i \in I\}, \\ \mathcal{R} = \{R_j \mid j \in J\}, \\ \mathcal{C} = \{c_k \mid k \in K\}; \end{cases}$$
 adică:

\mathcal{F} este mulțimea operațiilor cu argumente ale lui \mathcal{A} ;

\mathcal{R} este mulțimea relațiilor lui \mathcal{A} ;

\mathcal{C} este mulțimea constantelor lui \mathcal{A} .

Amintesc că am notat cu Var mulțimea **variabilelor** din limbajul \mathcal{L}_τ (limbajul de ordinul I asociat signaturii τ).

Elementele lui Var nu sunt variabile propoziționale, ci sunt variabilele care apar în predicate, i.e. în propozițiile cu variabile.

În continuare ne vom referi la **termenii** și **formulele** din limbajul \mathcal{L}_τ .

Definiție

O *interpretare* (sau *evaluare*, sau *semantică*) a limbajului \mathcal{L}_τ în structura algebrică \mathcal{A} este o funcție $s : Var \rightarrow A$.

Fiecare variabilă $x \in Var$ este “interpretată” prin elementul $s(x) \in A$.

Prelungim o interpretare s la mulțimea tuturor termenilor, obținând o funcție $s : \text{Term}(\mathcal{L}_\tau) \rightarrow A$.

Definiție (valorile asociate termenilor de o interpretare)

Pentru orice interpretare s și orice termen t , definim recursiv elementul $s(t) \in A$:

- dacă $t = x \in \text{Var}$, atunci $s(t) = s(x)$;
- dacă $t = c \in C$, atunci $s(t) = c$;
- dacă $t = f(t_1, \dots, t_n)$, unde $f \in F$ are aritatea $n \in \mathbb{N}^*$, iar $t_1, \dots, t_n \in \text{Term}(\mathcal{L}_\tau)$, atunci $s(t) = f(s(t_1), \dots, s(t_n))$.

Notație

Pentru orice interpretare $s : \text{Var} \rightarrow A$, orice $x \in \text{Var}$ și orice $a \in A$, notăm cu $s[x]_a : \text{Var} \rightarrow A$ interpretarea definită prin: oricare ar fi $v \in \text{Var}$,

$$s[x]_a(v) = \begin{cases} a, & \text{dacă } v = x, \\ s(v), & \text{dacă } v \neq x. \end{cases}$$

$s[x]_a$ ia valoarea a în variabila x și aceeași valoare ca s în celelalte variabile.

Acum definim valorile unei interpretări s în formule; acestea vor fi valori din **algebra Boole standard**, **algebra Boole a valorilor de adevăr**

$\mathcal{L}_2 = (\{0, 1\}, \vee, \wedge, \neg, 0, 1)$, unde $0 \neq 1$, \vee este **disjuncția** și \wedge este **conjuncția** (operații de latice), \neg este **complementul** (operație booleană), 0 este **primul element**, iar 1 este **ultimul element**. Notăm cu \rightarrow **implicația booleană**, iar cu \leftrightarrow **echivalența booleană** în \mathcal{L}_2 (operații booleene derivate). A se revedea lecția de curs recapitulativă despre algebrele Boole.

Astfel, obținem o funcție $s : \text{Term}(\mathcal{L}_\tau) \cup \text{Form}(\mathcal{L}_\tau) \rightarrow A \cup \mathcal{L}_2$.

Definiție (valorile booleene asociate formulelor de o interpretare)

Pentru orice interpretare s și orice formulă φ , *valoarea de adevăr a lui φ în interpretarea s* este un element din algebra Boole standard $\mathcal{L}_2 = \{0, 1\}$, notat cu $s(\varphi)$, definit, recursiv, astfel:

- dacă $\varphi = (t_1 = t_2)$, pentru doi termeni t_1, t_2 , atunci

$$s(\varphi) = \begin{cases} 1, & \text{dacă } s(t_1) = s(t_2), \\ 0, & \text{dacă } s(t_1) \neq s(t_2) \end{cases}$$

- dacă $\varphi = R(t_1, \dots, t_m)$, unde $R \in \mathcal{R}$ are aritatea $m \in \mathbb{N}^*$, iar t_1, \dots, t_m sunt

$$\text{termeni, atunci } s(\varphi) = \begin{cases} 1, & \text{dacă } (s(t_1), \dots, s(t_m)) \in R, \\ 0, & \text{dacă } (s(t_1), \dots, s(t_m)) \notin R \end{cases}$$

Definiție (continuare – am definit mai sus valorile lui φ în formulele atomice; acum extindem definiția lui φ la toate formulele)

- dacă $\varphi = \neg\psi$, pentru o formulă ψ , atunci $s(\varphi) = \overline{s(\psi)}$;
- dacă $\varphi = \psi \rightarrow \chi$, pentru două formule ψ, χ , atunci $s(\varphi) = s(\psi) \rightarrow s(\chi)$;
- dacă $\varphi = \forall x\psi$, unde $x \in Var$, iar ψ este o formulă, atunci $s(\varphi) = \bigwedge_{a \in A} s[a](\psi)$.

Remarcă

Este imediat că, pentru orice interpretare $s : Var \rightarrow A$, orice formule ψ, χ și orice $x \in Var$, au loc egalitățile:

- $s(\psi \vee \chi) = s(\psi) \vee s(\chi)$;
- $s(\psi \wedge \chi) = s(\psi) \wedge s(\chi)$;
- $s(\psi \leftrightarrow \chi) = s(\psi) \leftrightarrow s(\chi)$;
- $s(\exists x\psi) = \bigvee_{a \in A} s[a](\psi)$.

Fie $s : Var \rightarrow A$, $x \in Var$ și $\psi \in Form(\mathcal{L}_\tau)$. Cum $s(\alpha) \in \mathcal{L}_2 = \{0, 1\}$ pentru orice $\alpha \in Form(\mathcal{L}_\tau)$, din cele de mai sus rezulta că: $s(\forall x\psi) = 1$ dacă $s[a](\psi) = 1$ pentru toți $a \in A$, iar $s(\exists x\psi) = 1$ dacă există un $a \in A$ a.i. $s[a](\psi) = 1$.



Lemă (dacă două interpretări coincid pe variabilele dintr-un termen, atunci ele coincid în acel termen)

Fie $s_1, s_2 : \text{Var} \rightarrow A$ două interpretări. Atunci, pentru orice termen t , are loc implicația: $s_1|_{V(t)} = s_2|_{V(t)} \Rightarrow s_1(t) = s_2(t)$.

Propoziție (dacă două interpretări coincid pe variabilele libere dintr-o formulă, atunci ele coincid în acea formulă)

Fie $s_1, s_2 : \text{Var} \rightarrow A$ două interpretări. Atunci, pentru orice formulă φ , are loc implicația: $s_1|_{FV(\varphi)} = s_2|_{FV(\varphi)} \Rightarrow s_1(\varphi) = s_2(\varphi)$.

În mod trivial, oricare două interpretări $s_1, s_2 : \text{Var} \rightarrow A$ coincid pe \emptyset :

$(\forall x)(x \in \emptyset \Rightarrow s_1(x) = s_2(x))$ e adevărată, pentru că $x \in \emptyset$ e falsă pentru orice x , așadar $x \in \emptyset \Rightarrow s_1(x) = s_2(x)$ e adevărată pentru orice x . Prin urmare:

Corolar (cum enunțurile nu au variabile libere (i.e. $FV(\varphi) = \emptyset$ pt. orice enunț φ), rezultă că, într-un enunț, toate interpretările au aceeași valoare)

Dacă φ este un enunț, atunci $s(\varphi)$ nu depinde de interpretarea $s : \text{Var} \rightarrow A$.

Notăție (această valoare nu depinde decât de structura algebrică \mathcal{A})

Corolarul anterior ne permite să notăm, pentru orice enunț φ , cu $||\varphi||_{\mathcal{A}} = s(\varphi)$ valoarea oricărei interpretări $s : \text{Var} \rightarrow A$ în enunțul φ .

Satisfiabilitate, adevăruri semantice, deducție semantică

Definiție (modele pentru enunțuri și multimi de enunțuri)

Pentru orice enunț φ , notăm:

$$\mathcal{A} \models \varphi \text{ ddacă } \|\varphi\|_{\mathcal{A}} = 1,$$

și, în acest caz, spunem că \mathcal{A} *satisfacă* φ sau φ *este adevărat* în \mathcal{A} sau \mathcal{A} *este model pentru* φ .

Pentru orice mulțime Γ de enunțuri, notăm:

$$\mathcal{A} \models \Gamma \text{ ddacă } (\forall \gamma \in \Gamma) (\mathcal{A} \models \gamma),$$

și, în acest caz, spunem că \mathcal{A} *satisfacă* Γ sau că \mathcal{A} *este model pentru* Γ .

Definiție (satisfiabilitate)

Spunem că un enunț φ e *satisfiabil* sau are un *model* ddacă există o structură de ordinul I \mathcal{M} de signatură τ astfel încât $\mathcal{M} \models \varphi$.

Spunem că o mulțime Γ de enunțuri e *satisfiabilă* sau are un *model* ddacă există o structură de ordinul I \mathcal{M} de signatură τ astfel încât $\mathcal{M} \models \Gamma$.

Definiție (adevărurile semantice și deducția semantică)

Fie φ un enunț. Spunem că φ este *universal adevărat* (*adevăr semantic, tautologie*) ddacă $\mathcal{M} \models \varphi$ pentru orice structură de ordinul I \mathcal{M} de signatură τ .

Echivalență semantică pe mulțimea enunțurilor lui \mathcal{L}_τ

Definiție (continuare)

Notăm acest lucru cu $\models \varphi$.

Fie Γ o mulțime de enunțuri. Spunem că φ este *consecință semantică* a lui Γ sau φ enunț *deductibil semantic din Γ* dacă orice model pentru Γ este model pentru φ , adică, pentru orice structură de ordinul I \mathcal{M} de signatură τ , $\mathcal{M} \models \Gamma$ implică $\mathcal{M} \models \varphi$. Notăm acest lucru cu $\Gamma \models \varphi$.

Amintesc că mulțimea enunțurilor lui \mathcal{L}_τ este $\{\varphi \in \text{Form}(\mathcal{L}_\tau) \mid FV(\varphi) = \emptyset\}$.

Definiție (echivalență semantică)

Considerăm pe mulțimea enunțurilor din limbajul \mathcal{L}_τ o relație binară notată \equiv , definită astfel: oricare ar fi enunțurile φ și ψ , $\varphi \equiv \psi$ dacă, oricare ar fi structura algebrică \mathcal{M} de signatură τ , avem: $\mathcal{M} \models \varphi$ dacă $\mathcal{M} \models \psi$.

Adică o pereche de enunțuri $(\varphi, \psi) \in \equiv$ dacă φ și ψ au aceleași modele.

Relația binară \equiv se numește *echivalență semantică* în limbajul \mathcal{L}_τ . Pentru orice enunțuri φ, ψ , spunem că φ și ψ sunt *echivalente semantic* dacă $(\varphi, \psi) \in \equiv$.

Remarcă

\equiv este o relație de echivalență pe mulțimea enunțurilor lui \mathcal{L}_τ . Într-adevăr, reflexivitatea, simetria și tranzitivitatea lui \equiv sunt imediate.

Remarcă (două enunțuri sunt echivalente semantic dacă echivalența lor logică este tautologie)

Pentru orice enunțuri φ, ψ , avem:

$$\varphi \models \psi \text{ dacă } \models \varphi \leftrightarrow \psi.$$

Într-adevăr, în primul rând să observăm că $\varphi \leftrightarrow \psi$ este un enunț, întrucât $FV(\varphi \leftrightarrow \psi) = FV(\varphi) \cup FV(\psi) = \emptyset \cup \emptyset = \emptyset$.

Acum, considerând o interpretare arbitrară $s : Var \rightarrow A$, avem:

$$||\varphi \leftrightarrow \psi||_{\mathcal{A}} = s(\varphi \leftrightarrow \psi) = s(\varphi) \leftrightarrow s(\psi) = ||\varphi||_{\mathcal{A}} \leftrightarrow ||\psi||_{\mathcal{A}}.$$

Prin urmare: $\models \varphi \leftrightarrow \psi$ dacă, pentru orice algebră \mathcal{M} de tip τ , $\mathcal{M} \models \varphi \leftrightarrow \psi$, dacă, pentru orice algebră \mathcal{M} de tip τ , $||\varphi \leftrightarrow \psi||_{\mathcal{M}} = 1$, dacă, pentru orice algebră \mathcal{M} de tip τ , $||\varphi||_{\mathcal{M}} \leftrightarrow ||\psi||_{\mathcal{M}} = 1$, dacă, pentru orice algebră \mathcal{M} de tip τ , $||\varphi||_{\mathcal{M}} = ||\psi||_{\mathcal{M}}$, dacă, pentru orice algebră \mathcal{M} de tip τ , avem echivalența: $||\varphi||_{\mathcal{M}} = 1 \Leftrightarrow ||\psi||_{\mathcal{M}} = 1$, dacă, pentru orice algebră \mathcal{M} de tip τ , avem echivalența: $\mathcal{M} \models \varphi \Leftrightarrow \mathcal{M} \models \psi$, dacă $\varphi \models \psi$.

Am folosit următoarele fapte:

- pentru orice elemente a, b ale unei algebre Boole, avem: $a \leftrightarrow b = 1$ dacă $a = b$;
- oricare ar fi algebra \mathcal{M} de tip τ : $||\varphi||_{\mathcal{M}}, ||\psi||_{\mathcal{M}} \in \mathcal{L}_2 = \{0, 1\}$, așadar $||\varphi||_{\mathcal{M}} = ||\psi||_{\mathcal{M}}$ dacă fie ambele sunt egale cu 1, fie ambele sunt egale cu 0.

Remarcă

Conform remarcii anterioare, pentru orice enunțuri φ, ψ , avem: $\varphi \models \psi$ dacă, pentru orice algebră \mathcal{M} de tip τ , $\|\varphi\|_{\mathcal{M}} = \|\psi\|_{\mathcal{M}}$.

Din această exprimare a relației \models de echivalență semantică pentru \mathcal{L}_τ rezultă că \models are proprietățile relației \models de echivalență semantică din logica propozițională clasică, anume cele provenite din proprietățile booleene (*a se vedea mai jos*).

În plus, au loc proprietăți de tipul următor (*a se vedea o listă extinsă mai jos*), pentru orice $x, y \in \text{Var}$ și orice $\alpha, \beta \in \text{Form}(\mathcal{L}_\tau)$:

- dacă $FV(\alpha) \subseteq \{x, y\}$, astfel că următoarele formule sunt enunțuri, atunci:
 $\forall x \forall y \alpha \models \forall y \forall x \alpha$;
- dacă α este un enunț, $x \notin V(\alpha)$ și $FV(\beta) \subseteq \{x\}$, astfel că următoarele formule sunt enunțuri, atunci: $\alpha \models \forall x \alpha \models \exists x \alpha$ și
 $\forall x (\alpha \vee \beta) \models \alpha \vee \forall x \beta \models \forall x \alpha \vee \forall x \beta$.

Într-adevăr, considerând $x, y \in \text{Var}$, două **formule arbitrare** α, β , o algebră \mathcal{M} de tip τ , cu mulțimea elementelor M , și o interpretare $s : \text{Var} \rightarrow M$, avem: notând, pentru orice $a, b \in M$, cu $t_{a,b} : \text{Var} \rightarrow M$ interpretarea definită prin:

$$\text{oricare ar fi } v \in \text{Var}, t_{a,b}(v) = \begin{cases} a, & \text{dacă } v = x, \\ b, & \text{dacă } v = y, \\ s(v), & \text{dacă } v \notin \{x, y\}, \end{cases} \quad \text{observăm că}$$

$$t_{a,b} = (s[a^x])^y[b] = (s[b^y])^x[a], \text{ așadar:}$$

$$s(\forall x \forall y \alpha) = \bigwedge_{a \in A} s[a](\forall y \alpha) = \bigwedge_{a \in A} \bigwedge_{b \in A} (s[a])^y_b(\alpha) = \bigwedge_{a \in A} \bigwedge_{b \in A} t_{a,b}(\alpha) =$$

$$\bigwedge_{b \in A} \bigwedge_{a \in A} t_{a,b}(\alpha) = \bigwedge_{b \in A} \bigwedge_{a \in A} (s[b])^x_a(\alpha) = \bigwedge_{b \in A} s[b](\forall x \alpha) = s(\forall y \forall x \alpha);$$

iar, dacă $\forall x \forall y \alpha$ este un **enunț**, adică $FV(\alpha) \subseteq \{x, y\}$, astfel că și $\forall y \forall x \alpha$ este un enunț, atunci, conform calculului anterior,

$||\forall x \forall y \alpha||_{\mathcal{M}} = s(\forall x \forall y \alpha) = s(\forall y \forall x \alpha) = ||\forall y \forall x \alpha||_{\mathcal{M}}$, **și la fel mai jos**;

- dacă $x \notin FV(\alpha)$, în particular dacă $x \notin V(\alpha)$, atunci, pentru orice $a \in A$, $s|_{FV(\alpha)} = s[a]|_{FV(\alpha)}$, aşadar, conform propoziției precedente, pentru orice $a \in A$, $s(\alpha) = s[a](\alpha)$, prin urmare:

$$s(\forall x \alpha) = \bigwedge_{a \in A} s[a](\alpha) = s(\alpha) = \bigvee_{a \in A} s[a](\alpha) = s(\exists x \alpha);$$

întrucât algebra Boole \mathcal{L}_2 este finită și, în particular, completă, fapt pe care l-am folosit deja când am admis scrierii de tipul $\bigwedge_{a \in A} s[a](\alpha)$ sau $\bigvee_{a \in A} s[a](\alpha)$, deci am folosit existența conjuncțiilor și disjuncțiilor arbitrarе, avem:

$$s(\forall x(\alpha \vee \beta)) = \bigwedge_{a \in A} s[a](\alpha \vee \beta) = \bigwedge_{a \in A} (s[a](\alpha) \vee s[a](\beta)) = \bigwedge_{a \in A} (s(\alpha) \vee s[a](\beta)) =$$

$$s(\alpha) \vee \bigwedge_{a \in A} s[a](\beta) = s(\alpha) \vee s(\forall x \beta) = s(\alpha \vee \forall x \beta), \text{ dar, conform celor de mai sus,}$$

avem și: $s(\alpha) \vee s(\forall x \beta) = s(\forall x \alpha) \vee s(\forall x \beta) = s(\forall x \alpha \vee \forall x \beta)$.

Ce definește un program în Prolog și ce semnifică interogările?

Considerăm o signatură τ , un $n \in \mathbb{N}^*$ și o bază de cunoștințe în Prolog formată din n fapte și reguli corespunzătoare formulelor $\varphi_1, \dots, \varphi_n \in \text{Form}(\mathcal{L}_\tau)$.

Această bază de cunoștințe definește clasa tuturor algebrelor de signatură τ care satisfac mulțimea de enunțuri $\{\varphi_1, \dots, \varphi_n\}$, adică mulțimea modelelor lui $\{\varphi_1, \dots, \varphi_n\}$.

Dacă o interogare corespunde formulei $\varphi \in \text{Form}(\mathcal{L}_\tau)$, atunci a rezolva interogarea respectivă semnifică a determina dacă are loc deducția semantică

$$\{\varphi_1, \dots, \varphi_n\} \models \varphi,$$

adică dacă orice algebră din clasa algebrelor definite de această bază de cunoștințe este model pentru φ .

- 1 Să reluăm noțiunile din Logica Clasică a Predicatelor care stau la baza funcționării Prologului
- 2 Cu ce fel de signaturi și structuri algebrice de acele signaturi și cu ce tipuri/cazuri particulare de formule lucrează limbajul Prolog?
- 3 Modele pentru formule, satisfiabilitate în Logica Clasică a Predicatelor
- 4 Rezoluția în Logica Clasică a Predicatelor – regula de deducție care stă la baza funcționării limbajului Prolog
- 5 Rezoluția în Prolog

Formă prenex și formă normală conjunctivă prenex

Fie τ și \mathcal{A} ca în secțiunea anterioară a cursului.

Definiție

Se numește *formulă prenex* sau *formulă în formă prenex* o formulă de tipul

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

unde $n \in \mathbb{N}^*$, $x_1, \dots, x_n \in Var$, $Q_1, \dots, Q_n \in \{\forall, \exists\}$ și φ este o formulă fără cuantificatori.

Observație

Toate formulele cu care lucrează Prolog sunt în formă prenex: atât cele corespunzătoare **faptelor** și **regulilor**, cât și cele corespunzătoare **interrogărilor**.

Remarcă (orice enunț poate fi pus în formă prenex)

Pentru orice enunț ε , există o formulă prenex ψ astfel încât $\varepsilon \models \psi$.

Putem obține o formulă prenex echivalentă semantic cu ε folosind următoarele echivalențe semantice valabile pentru orice enunțuri α, β, γ , orice formule φ, ψ, χ și orice variabile x, y a.î. $FV(\varphi) \cup FV(\psi) \subseteq \{x\}$, iar $FV(\chi) \subseteq \{x, y\}$, astfel că următoarele formule sunt enunțuri:

(proprietăți pentru mutarea cuantificatorilor în față)

- $\forall x \forall y \chi \models \forall y \forall x \chi$ și $\exists x \exists y \chi \models \exists y \exists x \chi$
- $\neg \forall x \chi \models \exists x \neg \chi$ și $\neg \exists x \chi \models \forall x \neg \chi$
- $\forall x (\varphi \wedge \psi) \models \forall x \varphi \wedge \forall x \psi$ și $\exists x (\varphi \vee \psi) \models \exists x \varphi \vee \exists x \psi$
- dacă $x \notin V(\varphi)$, atunci:
$$\begin{cases} \varphi \models \forall x \varphi \models \exists x \varphi \\ \varphi \vee \forall x \psi \models \forall x (\varphi \vee \psi) \models \forall x \varphi \vee \forall x \psi \\ \varphi \wedge \exists x \psi \models \exists x (\varphi \wedge \psi) \models \exists x \varphi \wedge \exists x \psi \end{cases}$$

(proprietăți din calculul propozițional)

- ① $\alpha \rightarrow \beta \models \neg \alpha \vee \beta$ și $\alpha \leftrightarrow \beta \models (\neg \alpha \vee \beta) \wedge (\neg \beta \vee \alpha)$
- ② $\alpha \vee \alpha \models \alpha \wedge \alpha \models \alpha$
- ③ $\alpha \vee \beta \models \beta \vee \alpha$ și $\alpha \wedge \beta \models \beta \wedge \alpha$
- ④ $(\alpha \vee \beta) \vee \gamma \models \alpha \vee (\beta \vee \gamma)$ și $(\alpha \wedge \beta) \wedge \gamma \models \alpha \wedge (\beta \wedge \gamma)$
- ⑤ $\alpha \vee (\alpha \wedge \beta) \models \alpha \wedge (\alpha \vee \beta) \models \alpha$
- ⑥ $\alpha \vee (\beta \wedge \gamma) \models (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$ și $\alpha \wedge (\beta \vee \gamma) \models (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
- ⑦ $\neg \neg \alpha \models \alpha$
- ⑧ $\neg(\alpha \vee \beta) \models \neg \alpha \wedge \neg \beta$ și $\neg(\alpha \wedge \beta) \models \neg \alpha \vee \neg \beta$
- ⑨ $\alpha \vee (\beta \wedge \neg \beta) \models \alpha \vee (\beta \wedge \neg \beta \wedge \gamma) \models \alpha \wedge (\beta \vee \neg \beta) \models \alpha \wedge (\beta \vee \neg \beta \vee \gamma) \models \alpha$

Remarcă

Cum, în ultima remarcă din secțiunea anterioară a cursului, α și β sunt **formule arbitrară**, rezultă că **echivalențele semantice** de mai sus au loc și în **subformule** care nu sunt neapărat enunțuri, adică putem aplica faptul că, pentru orice $\varphi, \psi \in Form(\mathcal{L}_\tau)$, orice algebră M de signură τ , cu mulțimea elementelor M , și orice interpretare $s : Var \rightarrow M$, au loc, de exemplu:

- $s(\neg(\alpha \vee \beta)) = s(\neg\alpha \wedge \neg\beta)$, $s(\exists x \exists y \alpha) = s(\exists y \exists x \alpha)$;
- dacă $x \notin FV(\alpha)$, atunci $s(\alpha) = s(\forall x \alpha) = s(\exists x \alpha)$ și $s(\alpha \wedge \exists x \beta) = s(\exists x (\alpha \wedge \beta)) = s(\exists x \alpha \wedge \exists x \beta)$ etc..

Definiție (FNC prenex)

- Un *literal* este o formulă atomică sau negația unei formule atomice.
- O *clauză* este o disjuncție de literali.

Orice clauză se identifică cu mulțimea literalilor care o compun.

- O *formă normală conjunctivă prenex* sau un *enunț în formă normală conjunctivă prenex (FNC prenex)* este un enunț de forma:

$$\forall x_1 \dots \forall x_n \varphi,$$

unde $n \in \mathbb{N}^*$, $x_1, \dots, x_n \in Var$ și φ este o conjuncție de clauze, i. e. o

Definiție (continuare)

conjuncție de disjuncții de literali, implicit φ este o formulă fără cuantificatori, aşadar $x_1, \dots, x_n \in Var$ este o formulă prenex.

Orice conjuncție de clauze se identifică, cu mulțimea acelor clauze.

Orice FNC prenex $\forall x_1 \dots \forall x_n \varphi$ se identifică cu mulțimea clauzelor care îl compun pe φ .

Observație

Întrucât toate enunțurile au lungime finită, conjuncțiile și disjuncțiile la care face referire definiția de mai sus sunt finite.

Remarcă

O mulțime finită și nevidă de enunțuri este satisfiabilă dacă, conjuncția enunțurilor din acea mulțime e satisfiabilă.

Într-adevăr, dacă $n \in \mathbb{N}^*$ și $\gamma_1, \dots, \gamma_n$ sunt enunțuri, iar \mathcal{M} este o algebră de signatură τ , atunci: $||\gamma_1 \wedge \dots \wedge \gamma_n||_{\mathcal{M}} = ||\gamma_1||_{\mathcal{M}} \wedge \dots \wedge ||\gamma_n||_{\mathcal{M}}$, aşadar:

$\mathcal{M} \models \gamma_1 \wedge \dots \wedge \gamma_n$ dacă $||\gamma_1 \wedge \dots \wedge \gamma_n||_{\mathcal{M}} = 1$ dacă $||\gamma_1||_{\mathcal{M}} \wedge \dots \wedge ||\gamma_n||_{\mathcal{M}} = 1$ dacă $||\gamma_1||_{\mathcal{M}} = \dots = ||\gamma_n||_{\mathcal{M}} = 1$ dacă, pentru fiecare $i \in \overline{1, n}$, $\mathcal{M} \models \gamma_i$, dacă $\mathcal{M} \models \{\gamma_1, \dots, \gamma_n\}$.

Remarcă

În mod trivial, mulțimea vidă de enunțuri este satisfiabilă, întrucât următoarea afirmație este adevărată: pentru orice algebră \mathcal{M} de tip τ și orice γ , $\gamma \in \emptyset \Rightarrow \mathcal{M} \models \gamma$.

Definiție

Dacă ε este un enunț, iar ψ este o FNC prenex cu $\varepsilon \models \psi$, atunci mulțimea de clauze care îl compun pe ψ se numește *formă clauzală* pentru ε .

Dacă $n \in \mathbb{N}^*$, iar $\varepsilon_1, \dots, \varepsilon_n$ sunt enunțuri, atunci reuniunea unor forme clauzale pentru $\varepsilon_1, \dots, \varepsilon_n$ se numește *formă clauzală* pentru $\{\varepsilon_1, \dots, \varepsilon_n\}$.

Definiție

- **Clauza vidă** (i. e. clauza fără literali, clauza fără elemente) se notează cu \square (pentru a o deosebi de **mulțimea vidă de clauze**, \emptyset).
- O clauză C se zice *trivială* dacă există o formulă atomică α astfel încât $\alpha, \neg \alpha \in C$.
- O clauză nevidă $C = \{L_1, \dots, L_n\}$ (cu $n \in \mathbb{N}^*$ și L_1, \dots, L_n literali) se zice *satisfiabilă* dacă enunțul $L_1 \vee \dots \vee L_n$ corespunzător lui C e satisfiabil.
- O mulțime finită de clauze se zice *satisfiabilă* dacă enunțul în FNC corespunzător acelei mulțimi de clauze e satisfiabil.

Remarcă

Pentru orice enunț φ , are loc echivalența:

φ e nesatisfiabil (adică nu are model) dacă $\models \neg\varphi$.

Că o consecință a Teoremei de Completitudine Tare și a Teoremei Deducției, pentru orice $n \in \mathbb{N}^*$ și orice enunțuri $\varphi_1, \dots, \varphi_n, \varphi$, are loc echivalența:

$(\varphi_1 \wedge \dots \wedge \varphi_n) \rightarrow \varphi$ e nesatisfiabil dacă $\{\varphi_1, \dots, \varphi_n\} \models \varphi$.

Definiție (substituțiile sunt funcțiile de la variabile la termeni)

O substituție este o funcție $\sigma : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$.

Definiție

Fie $\sigma : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$. Următoarea extindere a lui σ la $\text{Term}(\mathcal{L}_\tau)$ se numește tot substituție (și, de obicei, se notează tot cu σ): $\tilde{\sigma} : \text{Term}(\mathcal{L}_\tau) \rightarrow \text{Term}(\mathcal{L}_\tau)$, definită, recursiv, astfel:

- $\tilde{\sigma}|_{\text{Var}} = \sigma$;
- oricare ar fi $c \in \mathcal{C}$, $\tilde{\sigma}(c) = c$;
- pentru orice $n \in \mathbb{N}^*$, orice $f \in \mathcal{F}$ având aritatea n și orice $t_1, \dots, t_{n_i} \in \text{Term}(\mathcal{L}_\tau)$, $\tilde{\sigma}(f(t_1, \dots, t_{n_i})) = f(\tilde{\sigma}(t_1), \dots, \tilde{\sigma}(t_n))$.

Definiție (definim substituțiile pe literali (adică pe formulele atomice și pe negațiile formulelor atomice), apoi pe clauze)

Fie $\sigma : \text{Term}(\mathcal{L}_\tau) \rightarrow \text{Term}(\mathcal{L}_\tau)$ o substituție (adică o extindere ca în definiția anterioară a unei funcții de la Var la $\text{Term}(\mathcal{L}_\tau)$).

Definim σ pe formulele atomice, recursiv, astfel:

- pentru orice $t_1, t_2 \in \text{Term}(\mathcal{L}_\tau)$, $\sigma(t_1=t_2) = \sigma(t_1)=\sigma(t_2)$;
- pentru orice $m \in \mathbb{N}^*$, orice $R \in \mathcal{R}$ de aritate m și orice $t_1, \dots, t_m \in \text{Term}(\mathcal{L}_\tau)$, $\sigma(R(t_1, \dots, t_m)) = R(\sigma(t_1), \dots, \sigma(t_m))$.

Pentru orice formulă atomică φ , definim $\sigma(\neg \varphi) = \neg \sigma(\varphi)$.

Pentru orice clauză $C = \{L_1, \dots, L_p\}$, unde $p \in \mathbb{N}^*$, iar L_1, \dots, L_p sunt literali, definim $\sigma(C) = \{\sigma(L_1), \dots, \sigma(L_p)\}$.

Notație

Fie $n \in \mathbb{N}^*$, $x_1, \dots, x_n \in \text{Var}$, două câte două distințe, și $t_1, \dots, t_n \in \text{Term}(\mathcal{L}_\tau)$.

Notăm cu

$$\{x_1/t_1, \dots, x_n/t_n\} : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$$

substituția definită prin:

$$\begin{cases} (\forall i \in \overline{1, n}) (\{x_1/t_1, \dots, x_n/t_n\}(x_i) = t_i); \\ (\forall x \in \text{Var} \setminus \{x_1, \dots, x_n\}) (\{x_1/t_1, \dots, x_n/t_n\}(x) = x). \end{cases}$$

Notație

Fie $n, k \in \mathbb{N}^*$, $x_1, \dots, x_n \in Var$, două câte două distințe, $i_1, \dots, i_k \in \overline{1, n}$, astfel încât $1 \leq i_1 < i_2 < \dots < i_k \leq n$, $t_{i_1}, \dots, t_{i_k} \in Term(\mathcal{L}_\tau)$ și $\varphi(x_1, \dots, x_n)$ o formulă cu $V(\varphi) \subseteq \{x_1, \dots, x_n\}$.

Atunci notăm cu

$$\varphi(x_1, \dots, x_{i_1-1}, t_{i_1}, x_{i_1+1}, \dots, x_{i_2-1}, t_{i_2}, x_{i_2+1}, \dots, x_{i_k-1}, t_{i_k}, x_{i_k+1}, \dots, x_n)$$

formula $\{x_{i_1}/t_{i_1}, \dots, x_{i_k}/t_{i_k}\}(\varphi)$.

Pentru a aplica rezoluția unui enunț, enunțul trebuie pus într-un anumit tip de FNC, numit **formă Skolem**, despre care se poate demonstra că există pentru orice enunț și satisfiabilă dacă acel enunț este satisfiabil.

Fie ε un enunț. Folosind proprietățile de mai sus și, eventual, redenumind variabilele pentru a nu avea variabile cuantificate și universal, și existențial (mai multe detalii în SEMINAR), se determină o formă prenex pentru ε :

$$\varepsilon \models Q_1 x_1 \dots Q_n x_n \varphi(x_1, \dots, x_n),$$

unde $n \in \mathbb{N}^*$, $x_1, \dots, x_n \in Var$, $Q_1, \dots, Q_n \in \{\forall, \exists\}$ și $\varphi(x_1, \dots, x_n)$ este o formulă fără cuantificatori. Apoi, folosind proprietățile ①–⑨ de mai sus, folosite în calculul propozițional pentru a pune enunțurile în FNC, se pune $\varphi(x_1, \dots, x_n)$ într-o FNC ψ , astfel obținându-se o altă formă prenex pentru ε :

$$\varepsilon \models Q_1 x_1 \dots Q_n x_n \psi(x_1, \dots, x_n),$$



Pentru fiecare cuantificator existențial $Q_i = \exists$, cu $i \in \overline{1, n}$, se adaugă la signatura τ câte o operație "fictivă" h_i , numită *funcție Skolem*, de aritate $i - |\{j \in \overline{1, i} \mid Q_j = \exists\}|$ (astfel că, în cazul în care $Q_1 = \exists$, h_1 este o constantă), și se înlocuiește fiecare apariție a variabilei x_i în $\psi(x_1, \dots, x_n)$ cu termenul h_i având $V(h_i) = \{x_1, \dots, x_i\} \setminus \{x_j \mid j \in \overline{1, i}\}$, iar cuantificatorii existențiali sunt eliminați din forma prenex anterioară.

Astfel obținem următoarea FNC: $\chi = \forall x_1, \dots, \forall x_{i_1-1} \forall x_{i_1+1} \dots \forall x_{i_2-1} \forall x_{i_2+1} \dots \forall x_{i_k-1} \forall x_{i_k+1} \dots \forall x_n \varphi(x_1, \dots, x_{i_1-1}, h_{i_1}(x_1, \dots, x_{i_1-1}), x_{i_1+1}, \dots, x_{i_2-1}, h_{i_2}(x_1, \dots, x_{i_1-1}, x_{i_1+1}, \dots, x_{i_2-1}), x_{i_2+1}, \dots, x_{i_k-1}, h_{i_k}(x_1, \dots, x_{i_1-1}, x_{i_1+1}, \dots, x_{i_2-1}, x_{i_2+1}, \dots, x_{i_k-1}), x_{i_k+1}, \dots, x_n)$.

Fie $p = n - k$ și (doar pentru a simplifica următoarea scriere) să redenumim variabilele $x_1, \dots, x_{i_1-1}, x_{i_1+1}, \dots, x_{i_2-1}, x_{i_2+1}, \dots, x_{i_k-1}, x_{i_k+1}, \dots, x_n$ în y_1, \dots, y_p , respectiv. Dacă $C_1(y_1, \dots, y_p), \dots, C_r(y_1, \dots, y_p)$ sunt clauzele formulei fără cuantificatori în FNC $\varphi(x_1, \dots, x_{i_1-1}, h_{i_1}(x_1, \dots, x_{i_1-1}), x_{i_1+1}, \dots, x_{i_2-1}, h_{i_2}(x_1, \dots, x_{i_1-1}, x_{i_1+1}, \dots, x_{i_2-1}), x_{i_2+1}, \dots, x_{i_k-1}, h_{i_k}(x_1, \dots, x_{i_1-1}, x_{i_1+1}, \dots, x_{i_2-1}, x_{i_2+1}, \dots, x_{i_k-1}), x_{i_k+1}, \dots, x_n)$, atunci:

$$\chi = \forall y_1 \dots \forall y_p C_1(y_1, \dots, y_p) \wedge \dots \wedge C_r(y_1, \dots, y_p) \models$$

$$(\forall y_1 \dots \forall y_p C_1(y_1, \dots, y_p)) \wedge \dots \wedge (\forall y_1 \dots \forall y_p C_r(y_1, \dots, y_p)),$$

iar această din urmă formulă este satisfiabilă dacă fiecare dintre formulele $\forall y_1 \dots \forall y_p C_1(y_1, \dots, y_p), \dots, \forall y_1 \dots \forall y_p C_r(y_1, \dots, y_p)$ este satisfiabilă.

Acum redenumim variabilele $\{y_1, \dots, y_p\}$ în fiecare dintre clauzele C_1, \dots, C_r astfel încât mulțimile $V(C_1), \dots, V(C_r)$ să devină două câte două disjuncte: $V(C_1) = \{z_{11}, \dots, z_{1p}\}, \dots, V(C_r) = \{z_{r1}, \dots, z_{rp}\}$, și, folosind, ca și mai sus, distributivitatea cuantificatorului universal (\forall) față de conjuncție (\wedge), obținem:

$$\xi \models \gamma = \forall z_{11} \dots \forall z_{1p} \dots \forall z_{r1} \dots \forall z_{rp} (C_1(z_{11}, \dots, z_{1p}) \wedge \dots \wedge C_r(z_{r1}, \dots, z_{rp})).$$

FNC γ se numește *formă Skolem* pentru ε , și este satisfiabilă dacă ε e satisfiabil. Nu putem spune că $\varepsilon \models \gamma$, întrucât ε și γ nu sunt satisfăcute de aceleași algebri: la signura τ trebuie să adăugăm simboluri de operații/constante corespunzătoare funcțiilor Skolem pentru a obține signura algebrelor în care este evaluată valoarea de adevăr a enunțului γ .

Definiție (unificare între literali)

Prin **unificare** între **formule atomice** înțelegem unificarea acelor formule privite ca termeni cu operația dominantă de rezultat boolean asociată relației din acele formule atomice, ca în Prolog.

Dacă φ și ψ sunt formule atomice, vom spune că $\neg\varphi$ și $\neg\psi$ *unifică* dacă φ și ψ unifică.

Definiție (clauze triviale)

Numim *clauză trivială* o clauză care conține doi literali φ și $\neg\psi$, unde φ și ψ sunt formule atomice care **unifică**.

(Rezoluția (regulă de deducție pentru logica clasică a predicatorilor))

Pentru orice clauze C și D cu $V(C) \cap V(D) = \emptyset$, dacă φ și ψ sunt formule atomice astfel încât $\varphi, \neg\varphi$ nu unifică, cu niciun literal din C și $\psi, \neg\psi$ nu unifică, cu niciun literal din D , iar $\sigma : Term(\mathcal{L}_\tau) \rightarrow Term(\mathcal{L}_\tau)$ este o substituție cu proprietatea că $\sigma(\varphi) = \sigma(\psi)$, atunci:

$$\frac{C \cup \{\varphi\}, D \cup \{\neg\psi\}}{\sigma(C) \cup \sigma(D)}.$$

Definiție (derivări prin rezoluție)

Fie o mulțime finită de clauze $\{D_1, \dots, D_k\}$.

Dacă $i, j \in \overline{1, k}$ a. î. $i \neq j$ și există două formule atomice φ și ψ astfel încât $\varphi, \neg\varphi$ nu unifică, cu niciun literal din D_i , $\psi, \neg\psi$ nu unifică, cu niciun literal din D_j , iar $\sigma : Term(\mathcal{L}_\tau) \rightarrow Term(\mathcal{L}_\tau)$ este o substituție cu proprietatea că $\sigma(\varphi) = \sigma(\psi)$, atunci mulțimea de clauze $R = \{\sigma(D_1), \dots, \sigma(D_k)\}$ se numește *rezolvent* al mulțimii de clauze $Q = \{D_i \cup \{\varphi\}, D_j \cup \{\neg\psi\}\} \cup \{D_t \mid t \in \overline{1, k} \setminus \{i, j\}\}$.

Deduția $\frac{Q}{R}$ se numește *derivare prin rezoluție* a mulțimii Q . Vom numi orice succesiune de derivări prin rezoluție tot *derivare prin rezoluție*. O succesiune de derivări prin rezoluție care începe cu o FNC/mulțime de clauze μ și se termină cu o FNC/mulțime de clauze ν se numește *derivare prin rezoluție a lui ν din μ* .

(Algoritmul Davis–Putnam (abreviat DP) pentru Logica Predicatelor)

INPUT: $m \in \mathbb{N}^*$ și o mulțime $S = \{C_1, \dots, C_m\}$ de clauze netriviale având $V(C) \cap V(D) = \emptyset$ pentru orice clauze $C, D \in S$ cu $C \neq D$;

$i := 1$; $S_1 := S$;

PASUL 1: considerăm o formulă atomică α_i care apare în măcar una dintre clauzele din S_i ;

$$T_i^0 := \{j \in \overline{1, m} \mid C_j \in S_i, (\exists \beta_i)(\neg \beta_{i,j} \in C_j \text{ și } \beta_{i,j} \text{ unifică, cu } \alpha_i)\};$$

$$T_i^1 := \{j \in \overline{1, m} \mid C_j \in S_i, (\exists \beta_i)(\beta_{i,j} \in C_j \text{ și } \beta_{i,j} \text{ unifică, cu } \alpha_i)\};$$

$$T_i := T_i^0 \cup T_i^1;$$

fie σ_i un unificator pentru elementele mulțimii $\{\beta_{i,j} \mid j \in T_i\}$;

PASUL 2: dacă $T_i^0 \neq \emptyset$ și $T_i^1 \neq \emptyset$,

atunci $U_i := \{\sigma_i(C_j \setminus \{\neg \beta_{i,j}\}) \cup \sigma_i(C_k \setminus \{\beta_{i,k}\}) \mid j \in T_i^0, k \in T_i^1\}$;

altfel $U_i := \emptyset$;

PASUL 3: $S_{i+1} := (S_i \setminus \{C_j \mid j \in T_i\}) \cup U_i$;

$S_{i+1} := S_{i+1} \setminus \{C \in S_{i+1} \mid (\exists \beta \in V) (\beta \text{ formulă atomică și } \beta, \neg \beta \in C)\}$ (eliminăm din S_{i+1} clauzele triviale);

PASUL 4: dacă $S_{i+1} = \emptyset$,

atunci OUTPUT: S e satisfiabilă;

altfel, dacă $\square \in S_{i+1}$,

atunci OUTPUT: S nu e satisfiabilă;

altfel $i := i + 1$ și mergi la PASUL 1.

În derivările prin rezoluție, inclusiv în unificările efectuate în cadrul acestor derivări, deci și în aplicarea algoritmului DP, funcțiile Skolem sunt tratate ca orice funcții. Rezultatul algoritmului DP e influențat de FORMA SKOLEM obținută pentru enunțul despre care dorim să aflăm dacă e satisfiabil sau nu. A se vedea un exemplu mai jos.

Pentru **tipurile de enunțuri** pentru care **Prologul** aplică tehnica rezoluției, algoritmul DP este **corect și complet**: se termină într-un număr finit de pași și dă rezultatul corect: determină dacă un astfel de enunț e satisfiabil sau nu.

În general însă, spre deosebire de cazul logicii propozițiilor, rezultatul aplicării algoritmului DP unui enunț cu sau fără cuantificatori existențiali nu este neapărat corect.

Revenind la o **formă Skolem** pentru un enunț ε : ce rol au funcțiile Skolem?

Dacă $x, y \in Var$, iar φ este o formulă cu $FV(\varphi) \subseteq \{x, y\}$, de ce avem:
 $\exists x \forall y \varphi \models \forall y \exists x \varphi$, dar nu și $\forall y \exists x \varphi \not\models \exists x \forall y \varphi$? Pentru că, în enunțul $\forall y \exists x \varphi$, o valoare a lui x care satisface acest enunț poate să depindă de valoarea b a lui y pentru care perechea (a, b) satisface acest enunț.

Exemplu

- $(\exists x \in \mathbb{R}) (\forall y \in \mathbb{R}) (x + y = 0)$: **fals**: nu există un astfel de x comun tuturor valorilor lui $y \in \mathbb{R}$;
- $(\forall y \in \mathbb{R}) (\exists x \in \mathbb{R}) (x + y = 0)$: **adevărat**;

Exemplu (continuare)

- $(\exists x \in \mathbb{N}) (\forall y \in \mathbb{Z}) (x \in \{y, -y\})$: **fals**: nu există un astfel de x comun tuturor valorilor lui $y \in \mathbb{Z}$;
- $(\forall y \in \mathbb{Z}) (\exists x \in \mathbb{N}) (x \in \{y, -y\})$: **adevărat**.

Fie $f : \mathbb{R} \rightarrow \mathbb{R}$, având cel puțin 3 valori distincte, i.e. cardinalul imaginii sale $f(\mathbb{R})$ mai mare sau egal cu 3.

- $(\exists x \in [0, +\infty)) (\forall y \in \mathbb{R}) (x \in \{f(y), -f(y)\})$: **fals**: nu există un astfel de x comun tuturor valorilor lui $y \in \mathbb{R}$, pentru că alegerea lui f ne asigură de faptul că există $y_0, y_1 \in \mathbb{R}$ cu proprietatea că $f(y_1) \notin \{f(y_0), -f(y_0)\}$, așadar perechea $(x_1 = f(y_1), y_0)$ nu satisface proprietatea $x_1 \in \{f(y_0), -f(y_0)\}$;
- $(\forall y \in \mathbb{R}) (\exists x \in [0, +\infty)) (x \in \{f(y), -f(y)\})$: **adevărat**.

O **formă Skolem** pentru enunțul $(\forall y) (\exists x) (x + y = 0)$ este:

$(\forall y) (g(y) + y = 0)$, unde g este o funcție Skolem unară, iar acest enunț în formă Skolem, pentru domeniul valorilor \mathbb{R} pentru ambele variabile, x și y , semnifică faptul că există o funcție $g : \mathbb{R} \rightarrow \mathbb{R}$ astfel încât $(\forall y \in \mathbb{R}) (g(y) + y = 0)$.

O **formă Skolem** pentru enunțul $(\forall y) (\exists x) (x \in \{y, -y\})$ este:

$(\forall y) (h(y) \in \{y, -y\})$, unde h este o funcție Skolem unară, iar acest enunț în formă Skolem, pentru domeniile valorilor \mathbb{N} , respectiv \mathbb{Z} pentru variabilele x , respectiv y , semnifică faptul că există o funcție $h : \mathbb{Z} \rightarrow \mathbb{N}$ astfel încât $(\forall y \in \mathbb{Z}) (h(y) \in \{y, -y\})$.

O formă Skolem pentru enunțul $(\forall y)(\exists x)(x \in \{f(y), -f(y)\})$ este:
 $(\forall y)(k(y) \in \{f(y), -f(y)\})$, unde k este o funcție Skolem unară, iar acest enunț în formă Skolem, pentru pentru domeniile valorilor $[0, +\infty)$, respectiv \mathbb{R} pentru variabilele x , respectiv y , semnifică faptul că există o funcție $k : \mathbb{R} \rightarrow [0, +\infty)$ astfel încât $(\forall y \in \mathbb{R})(k(y) \in \{f(y), -f(y)\})$. Într-adevăr, acest enunț este satisfăcut de funcția $k : \mathbb{R} \rightarrow [0, +\infty)$, definită prin:

$$(\forall y \in \mathbb{R}) \left(k(y) = |f(y)| = \begin{cases} f(y), & \text{dacă } f(y) \geq 0, \\ -f(y), & \text{altfel.} \end{cases} \right).$$

Deci faptul că funcția k nu coincide nici cu f , nici cu $-f$ nu influențează satisfiabilitatea enunțului anterior.

Exemplu

Să considerăm un limbaj de ordinul I conținând două simboluri de relații unare p și q . Fie x, y variabile distințte. Considerăm enunțurile:

- $\varphi = \forall x p(x) \wedge \exists y q(y)$;
- $\psi = \exists x [(p(x) \vee q(x)) \wedge \forall y ((p(x) \vee \neg q(y)) \wedge \neg p(x))]$.

$\varphi \models \forall x \exists y (p(x) \wedge q(y))$, dar avem și

$\varphi \models \exists y q(y) \wedge \forall x p(x) \models \exists y \forall x (q(y) \wedge p(x)) \models \exists y \forall x (p(x) \wedge q(y))$.

Așadar două forme Skolem diferite pentru enunțul φ sunt:

- $\forall x (p(x) \wedge q(c))$, unde c este o constantă Skolem; această formă Skolem corespunde mulțimii de clauze $\{\{p(x)\}, \{q(c)\}\}$, care nu are derivări prin rezoluție, deci algoritmul DP determină satisfiabilitatea acestei mulțimi de clauze;
- $\forall x (p(x) \wedge q(f(x)))$, unde f este o funcție Skolem unară; această formă Skolem corespunde mulțimii de clauze $\{\{p(x)\}, \{q(f(x))\}\}$, care nu are derivări prin rezoluție, deci algoritmul DP determină și satisfiabilitatea acestei mulțimi de clauze.

La fel ca mai sus:

- $\psi \models \exists x \forall y [(p(x) \vee q(x)) \wedge (p(x) \vee \neg q(y)) \wedge \neg p(x)]$, având drept formă Skolem enunțul $\forall y [(p(c) \vee q(c)) \wedge (p(c) \vee \neg q(y)) \wedge \neg p(c)]$, unde c este o constantă Skolem, corespunzător mulțimii de clauze $\{\{p(c), q(c)\}, \{p(c), \neg q(y)\}, \{\neg p(c)\}\}$;
- $\psi \models \forall y \exists x [(p(x) \vee q(x)) \wedge (p(x) \vee \neg q(y)) \wedge \neg p(x)]$, având drept formă Skolem enunțul $\forall y [(p(f(y)) \vee q(f(y))) \wedge (p(f(y)) \vee \neg q(y)) \wedge \neg p(f(y))]$, unde f este o funcție Skolem unară, corespunzător mulțimii de clauze $\{\{p(f(v)), q(f(v))\}, \{p(f(y)), \neg q(y)\}, \{\neg p(f(z))\}\}$.

Pentru prima dintre mulțimile de clauze de mai sus avem următoarea derivare prin rezoluție a clauzei vide, prin urmare algoritmul DP determină nesatisfiabilitatea acestei mulțimi de clauze:

$\{p(c), q(c)\}, \{p(c), \neg q(y)\}$ (cu unificatorul $\{y/c\}$), $\{\neg p(c)\}$
$\{p(c)\}, \{\neg p(c)\}$

□

Pentru a doua dintre mulțimile de clauze de mai sus avem următoarele derivări prin rezoluție, dintre care niciuna nu ajunge la clauza vidă, prin urmare algoritmul DP determină satisfiabilitatea acestei mulțimi de clauze:

$\{p(f(v)), q(f(v))\}, \{p(f(y)), \neg q(y)\}$ (cu unificatorul $\{y/f(v)\}$), $\{\neg p(f(z))\}$
$\{p(f(v)), p(f(f(v)))\}, \{\neg p(f(z))\}$ (cu unificatorul $\{z/v\}$)

$\{p(f(f(v)))\}$

$\{p(f(v)), q(f(v))\}, \{p(f(y)), \neg q(y)\}$ (cu unificatorul $\{y/f(v)\}$), $\{\neg p(f(z))\}$
$\{p(f(v)), p(f(f(v)))\}, \{\neg p(f(z))\}$ (cu unificatorul $\{z/f(v)\}$)

$\{p(f(v))\}$

$\{p(f(v)), q(f(v))\}, \{p(f(y)), \neg q(y)\}, \{\neg p(f(z))\}$ (cu unificatorul $\{z/v\}$)
$\{q(f(v))\}, \{p(f(y)), \neg q(y)\}$ (cu unificatorul $\{y/f(v)\}$)

$\{p(f(f(v)))\}$

Vom vedea că Prologul aplică rezoluția numai pentru FNC prenex (FNC fără cuantificatori precedate numai de cuantificatori universali) obținute fără Skolemizare (i.e. fără înlocuirea unor variabile cu funcții Skolem), mai precis pentru un tip particular de astfel de FNC prenex: **Prologul** întotdeauna aplică **regula rezoluției** pentru o **clauză Horn** și o **clauză scop**: definițiile mai jos.

- 1 Să reluăm noțiunile din Logica Clasică a Predicatelor care stau la baza funcționării Prologului
- 2 Cu ce fel de signaturi și structuri algebrice de acele signaturi și cu ce tipuri/cazuri particulare de formule lucrează limbajul Prolog?
- 3 Modele pentru formule, satisfiabilitate în Logica Clasică a Predicatelor
- 4 Rezoluția în Logica Clasică a Predicatelor – regula de deducție care stă la baza funcționării limbajului Prolog
- 5 Rezoluția în Prolog

Amintesc că:

- *formulele atomice* sunt formulele fără cuantificatori și fără conectori logici;
- *literalii* sunt formulele atomice și negațiile formulelor atomice;
- *clauzele* sunt disjuncțiile finite (nu neapărat nevide) de literali; clauzele se identifică, cu mulțimile literalilor pe care îi conțin; dacă sunt nevide, clauzele sunt formule fără cuantificatori;
- *FNC prenex* sunt enunțurile date de conjuncții finite și nevide de clauze nevide precedate de cuantificatori universali; FNC prenex se identifică, cu mulțimile clauzelor pe care le conțin.

Definiție

O *cluză Horn* sau *cluză definită* este o cluză care conține exact o formulă atomică nenegată (implicit e o cluză nevidă), iar ceilalți literali pe care îi conține sunt formule atomice negate.

O *cluză scop* este o cluză nevidă care conține numai formule atomice negate.

Amintesc cu ce tipuri de formule (mai precis enunțuri) lucrează Prologul:

- **faptele** corespund unor enunțuri de tipul următor: $\forall x_1 \forall x_2 \dots \forall x_n \alpha$, unde $n \in \mathbb{N}^*$ și α este o formulă atomică;
- **regulile** corespund unor enunțuri de tipul: $\forall y_1 \forall y_2 \dots \forall y_k (\varphi \rightarrow \beta)$, unde $k \in \mathbb{N}^*$, φ este o formulă fără cuantificatori, iar β este o formulă atomică;
- **interrogările** corespund unor formule de tipul $\exists z_1 \exists z_2 \dots \exists z_p \psi$, unde $p \in \mathbb{N}^*$ și ψ este o formulă fără cuantificatori.

Pentru cele ce urmează, cu notațiile de mai sus, vom presupune că φ și ψ sunt conjuncții finite (desigur, nevide) de formule atomice.

Dacă în φ sau ψ apare negația unei formule atomice, de exemplu

$\rho = \neg R(t_1, \dots, t_m)$, unde $m \in \mathbb{N}^*$, t_1, \dots, t_m sunt termeni și R este un simbol de relație m -ară, atunci transformăm pe ρ într-o formulă atomică introducând în signatură simbolul de relație m -ară $\neg R$ astfel încât, în orice algebră \mathcal{A} de signatura definită de respectivul program în Prolog care satisface enunțurile date de faptele și regulile din programul respectiv, dacă A este multimea de elemente ale lui \mathcal{A} , atunci $(\neg R)^{\mathcal{A}} = A^m \setminus R^{\mathcal{A}} = \{(a_1, \dots, a_m) \in A^m \mid (a_1, \dots, a_m) \notin R\}$. Într-un mod similar se procedează dacă în formule precum φ , ψ apar, de exemplu, disjuncții de formule atomice, nu numai conjuncții.

Să presupunem, aşadar, că: $\varphi = \gamma_1 \wedge \dots \wedge \gamma_q$ și $\psi = \delta_1 \wedge \dots \wedge \delta_s$, unde $q, s \in \mathbb{N}^*$, iar $\gamma_1, \dots, \gamma_q, \delta_1, \dots, \delta_s$ sunt formule atomice. Atunci:

$$\forall y_1 \forall y_2 \dots \forall y_k (\varphi \rightarrow \beta) = \forall y_1 \forall y_2 \dots \forall y_k ((\gamma_1 \wedge \dots \wedge \gamma_q) \rightarrow \beta) \models$$

$$\forall y_1 \forall y_2 \dots \forall y_k (\neg(\gamma_1 \wedge \dots \wedge \gamma_q) \vee \beta) \models \forall y_1 \forall y_2 \dots \forall y_k (\neg \gamma_1 \vee \dots \vee \neg \gamma_q \vee \beta),$$

iar $\neg \gamma_1 \vee \dots \vee \neg \gamma_q \vee \beta$ este o **clauză Horn**.

Cum α este o formulă atomică, în particular α este o **clauză Horn**.

Așadar **clauzele din baza de cunoștințe** constau în enunțuri date de **clauze Horn** precedate de cuantificatori universali.

Amintesc că, dacă baza de cunoștințe este formată din enunțurile $\varepsilon_1, \dots, \varepsilon_r$, cu $r \in \mathbb{N}^*$, iar ε este un enunț corespunzător unei interogări, atunci cerința adresată Prologului prin această interogare este de a determina dacă:

$$\{\varepsilon_1, \dots, \varepsilon_r\} \models \varepsilon,$$

sau, echivalent:

$$\{\varepsilon_1 \wedge \dots \wedge \varepsilon_r\} \models \varepsilon,$$

sau, echivalent, conform Teoremei Deducției Semantice:

$$\models (\varepsilon_1 \wedge \dots \wedge \varepsilon_r) \rightarrow \varepsilon,$$

fapt echivalent cu:

enunțul $\neg [(\varepsilon_1 \wedge \dots \wedge \varepsilon_r) \rightarrow \varepsilon]$ e nesatisfiabil,
adică enunțul $\varepsilon_1 \wedge \dots \wedge \varepsilon_r \wedge \neg \varepsilon$ e nesatisfiabil,

întrucât

$$\neg [(\varepsilon_1 \wedge \dots \wedge \varepsilon_r) \rightarrow \varepsilon] \models \neg [\neg (\varepsilon_1 \wedge \dots \wedge \varepsilon_r) \vee \varepsilon] \models \varepsilon_1 \wedge \dots \wedge \varepsilon_r \wedge \neg \varepsilon,$$

iar acest din urmă enunț poate fi transformat într-o **FNC prenex** prin mutarea cuantificatorilor în față (a se vedea mai jos).

Definiție

O derivare prin rezoluție a clauzei vide \square din FNC prenex $\varepsilon_1 \wedge \dots \wedge \varepsilon_r \wedge \neg \varepsilon$ se numește *SLD–respingere* a clauzei ε din mulțimea de clauze $\{\varepsilon_1, \dots, \varepsilon_r\}$.

(Principala regulă de deducție utilizată de Prolog)

Pentru a satisface scopul ε , Prologul caută o SLD–respingere a clauzei ε din mulțimea de clauze corespunzătoare bazei de cunoștințe.

Acest caz al rezoluției aplicat de Prolog este numit *rezoluție SLD* (*Selective Linear Definite Clause Resolution*: rezoluție liniară selectivă pe clauze definite).

Baza de cunoștințe se identifică, cu, conjuncția enunțurilor corespunzătoare faptelor și regulilor ($\varepsilon_1 \wedge \dots \wedge \varepsilon_r$ cu notația de mai sus), care, prin mutarea cuantificatorilor în față, devine o **FNC prenex** constând din **clauze Horn**.

Într–adevăr, de exemplu, considerând faptul și regula de mai sus, dacă mulțimea de variabile $\{x_1, \dots, x_n\} \cup \{y_1, \dots, y_k\} = \{v_1, \dots, v_j\}$, avem:

$$\forall x_1 \dots \forall x_n \alpha \wedge \forall y_1 \dots \forall y_k (\varphi \rightarrow \beta) \models \forall v_1 \dots \forall v_j [\alpha \wedge (\varphi \rightarrow \beta)] =$$

$$\forall v_1 \dots \forall v_j [\alpha \wedge ((\gamma_1 \wedge \dots \wedge \gamma_q) \rightarrow \beta)] \models \forall v_1 \dots \forall v_j [\alpha \wedge (\neg \gamma_1 \vee \dots \vee \neg \gamma_q \vee \beta)].$$

La fel pentru mai multe fapte și/sau reguli.

De exemplu, dacă baza de cunoștințe este formată din faptul și regula de mai sus, iar $\{v_1, \dots, v_j\} \cup \{z_1, \dots, z_p\} = \{w_1, \dots, w_l\}$ atunci interogarea de mai sus presupune să verifice nesatisfiabilitatea enunțului:

$$\forall x_1 \dots \forall x_n \alpha \wedge \forall y_1 \dots \forall y_k (\varphi \rightarrow \beta) \wedge \neg(\exists z_1 \exists z_2 \dots \exists z_p \psi) \models$$
$$\forall x_1 \dots \forall x_n \alpha \wedge \forall y_1 \dots \forall y_k (\varphi \rightarrow \beta) \wedge \forall z_1 \dots \forall z_p \neg \psi \models$$
$$\forall w_1 \dots \forall w_l [\alpha \wedge (\varphi \rightarrow \beta) \wedge \neg \psi] \models \forall w_1 \dots \forall w_l [\alpha \wedge (\varphi \rightarrow \beta) \wedge \neg(\delta_1 \wedge \dots \wedge \delta_s)] \models$$
$$\forall w_1 \dots \forall w_l (\alpha \wedge (\neg \gamma_1 \vee \dots \vee \neg \gamma_q \vee \beta) \wedge (\neg \delta_1 \vee \dots \vee \neg \delta_s)).$$

Observăm că negația interogării de mai sus, anume enunțul

$\forall z_1 \dots \forall z_p (\neg \delta_1 \vee \dots \vee \neg \delta_s)$ este un enunț dat de o **clauză scop** precedată de cuantificatori universali.

Tehnica descrisă mai sus presupune adăugarea acestei clauze scop la mulțimea de cluze Horn corespunzătoare bazei de cunoștințe și aplicarea rezoluției pentru mulțimea de cluze astfel obținută:

- dacă această mulțime de cluze este **nesatisfiabilă**, atunci răspunsul la interogarea de mai sus este **true**: enunțul $\exists z_1 \dots \exists z_p (\delta_1 \wedge \dots \wedge \delta_s)$ se deduce semantic din baza de cunoștințe, deci scopul dat de acest enunț este satisfăcut;
- dacă această mulțime de cluze este **satisfiabilă**, atunci răspunsul la interogarea de mai sus este **false**: enunțul $\exists z_1 \dots \exists z_p (\delta_1 \wedge \dots \wedge \delta_s)$ nu este consecință semantică a bazei de cunoștințe, deci scopul dat de acest enunț eșuează.

Exemplu

Considerăm baza de cunoștințe:

a. b.

c :- a, b.

și interogarea: ?- c.

Regula din programul de mai sus corespunde formulei fără variabile:

$$(a \wedge b) \rightarrow c \models \neg(a \wedge b) \vee c \models \neg a \vee \neg b \vee c,$$

așadar întreaga bază de cunoștințe corespunde FNC:

$$a \wedge b \wedge [(a \wedge b) \rightarrow c] \models a \wedge b \wedge (\neg a \vee \neg b \vee c),$$

având forma clauzală: $\{\{a\}, \{b\}, \{\neg a, \neg b, c\}\}$.

Să vedem dacă, adăugând la această mulțime de clauze clauza $\{\neg c\}$, obținem o

$$\begin{array}{c} \{a\}, \{b\}, \{\neg a, \neg b, c\}, \{\neg c\} \\ \hline \{a\}, \{b\}, \{\neg a, \neg b\} \\ \hline \{a\}, \{\neg a\} \\ \hline \square \end{array}$$

Am ajuns la **clauza vidă**, așadar mulțimea de clauze

$\{\{a\}, \{b\}, \{\neg a, \neg b, c\}, \{\neg c\}\}$ e **nesatisfiabilă**, prin urmare:

$$\{a, b, (a \wedge b) \rightarrow c\} \vDash c,$$

Să ne amintim ce sunt substituțiile

adică scopul c e satisfăcut, deci răspunsul la interogarea ?- c este **true**.

Cum, în exemplul anterior, nu avem variabile, ne situăm în cazul **rezoluției propoziționale**. Cazul general al **rezoluției pentru logica predicatelor** presupune determinarea unui **unificator** la fiecare aplicare a **regulii rezoluției** și aplicarea acestui unificator la clauzele rămase după aplicarea acelui pas de rezoluție.

Definiție (o *substituție* este o funcție de la variabile la termeni)

O *substituție* este o funcție $\sigma : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$.

Definiție și notație (prelungirea unei substituții la termeni)

Fie $\sigma : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$. Următoarea extindere a lui σ la întreaga mulțime a termenilor se numește tot *substituție* (și, de obicei, se notează tot cu σ):

$\tilde{\sigma} : \text{Term}(\mathcal{L}_\tau) \rightarrow \text{Term}(\mathcal{L}_\tau)$, definită, recursiv, astfel:

- pentru orice $v \in \text{Var}$, $\tilde{\sigma}(v) = \sigma(v)$;
- pentru orice (simbol de) constantă c , $\tilde{\sigma}(c) = c$;
- pentru orice $h \in \mathbb{N}^*$, orice (simbol de) operație h -ară f și orice $t_1, \dots, t_h \in \text{Term}(\mathcal{L}_\tau)$, $\tilde{\sigma}(f(t_1, \dots, t_h)) = f(\tilde{\sigma}(t_1), \dots, \tilde{\sigma}(t_h))$.

Un **unificator** pentru o mulțime de termeni este o substituție care are aceeași valoare în acei termeni, i.e., aplicată acelor termeni, are ca rezultat termeni identici

Definiție

O *substituție* $\sigma : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$ se numește *unificator* pentru $h \in \mathbb{N}^*$ termeni $t_1, \dots, t_h \in \text{Term}(\mathcal{L}_\tau)$ dacă $\tilde{\sigma}(t_1) = \dots = \tilde{\sigma}(t_h)$.

Algoritmul de unificare aplicat unor termeni t_1, \dots, t_h ($h \in \mathbb{N}^*$) întoarce un **cel mai general unificator** al termenilor t_1, \dots, t_h , adică un unificator μ al acestor termeni cu proprietatea că orice unificator al termenilor t_1, \dots, t_h este dat de compunerea unei substituții cu μ .

Amintesc și această:

Notație

Dacă $h \in \mathbb{N}^*$, $v_1, \dots, v_h \in \text{Var}$ și $t_1, \dots, t_h \in \text{Term}(\mathcal{L}_\tau)$, atunci substituția pe care o notăm $\{v_1/t_1, \dots, v_h/t_h\} : \text{Var} \rightarrow \text{Term}(\mathcal{L}_\tau)$ este definită prin:

$$\begin{cases} \{v_1/t_1, \dots, v_h/t_h\}(v_i) = t_i, & \text{pentru orice } i \in \overline{1, h}, \\ \{v_1/t_1, \dots, v_h/t_h\}(v) = v, & \text{pentru orice } v \in \text{Var} \setminus \{v_1, \dots, v_h\}. \end{cases}$$

Cu notațiile de mai sus, avem, în Prolog:

- **faptul** $\forall x_1 \forall x_2 \dots \forall x_n \alpha$:

α. % poate fi privit ca o **regulă** cu **membrul stâng** α și **premisa vidă**

- **regula** $\forall y_1 \forall y_2 \dots \forall y_k [(\gamma_1 \wedge \dots \wedge \gamma_q) \rightarrow \beta]$:

$\beta :- \gamma_1, \dots, \gamma_q.$ % cu **membrul stâng** β și **premisa** $\gamma_1, \dots, \gamma_q$

- **interogarea** $\exists z_1 \exists z_2 \dots \exists z_p (\delta_1 \wedge \dots \wedge \delta_s)$:

?- $\delta_1, \dots, \delta_s.$

unde α, β, γ₁, ..., γ_q, δ₁, ..., δ_s sunt **formule atomice**, adică relații aplicate la termeni, astfel că, în Prolog, pot fi considerate **termeni** cu operatorii dominanți dați de **operațiile de rezultat boolean asociate acelor relații** (a se vedea mai sus).

(Cum rezolvă Prologul interogarea ?- δ₁, ..., δ_s.?)

Prologul rezolvă o interogare ?- δ₁, ..., δ_s (adică satisfacă **scopul compus** δ₁, ..., δ_s) satisfăcând, PE RÂND, **subscopurile** δ₁, ..., δ_s, DE LA STÂNGA LA DREAPTA.

Să detaliem modul în care Prologul răspunde interogărilor

Cum satisfac Prologul un **subscop** δ_i , cu $i \in \overline{1, s}$? Considerând **faptele** și **regulile** din **baza de cunoștințe DE SUS ÎN JOS** și unificând (prin aplicarea **algoritmului de unificare**), cu un **cel mai general unificator** μ , pe δ_i cu **formula atomică** dintr-un:

- **fapt** α , caz în care următorul scop compus este:

?- $\mu(\delta_1), \dots, \mu(\delta_{i-1}), \mu(\delta_{i+1}), \dots, \mu(\delta_s)$.

adică se scoate din scopul compus $\delta_1, \dots, \delta_s$ subscopul δ_i , iar la celelalte subscopuri se aplică unificatorul μ ; acesta este un **caz particular** al aplicării unei **reguli**, pentru **premisa vidă**;

sau din

- **membrul stâng** β al unei reguli $\beta :- \gamma_1, \dots, \gamma_q$, caz în care următorul scop compus este:

?- $\mu(\delta_1), \dots, \mu(\delta_{i-1}), \mu(\gamma_1), \dots, \mu(\gamma_q), \mu(\delta_{i+1}), \dots, \mu(\delta_s)$.

adică, în scopul compus $\delta_1, \dots, \delta_s$, se înlocuiește subscopul δ_i , cu **membrul drept al regulei** $\beta :- \gamma_1, \dots, \gamma_q$, iar la subscopurile din lista astfel obținută se aplică unificatorul μ .

Procedeul de mai sus este aplicat în manieră **backtracking**. Mai precis, Prologul aplică:

Algoritmul **backward chaining**

Mai jos, **cazul particular** $q = 0$ este cazul aplicării unui **fapt**.

backwardChaining(INTEGER s ; literali $\delta_1, \dots, \delta_s$; unificator μ)

FOR $i = 1$ TO s

$\delta'_i := \mu(\delta_i)$

FOR $i = 1$ TO s

FOR toate clauzele $\beta :- \gamma_1, \dots, \gamma_q$ din baza de cunoștințe

IF ν este un cel mai general unificator pentru δ'_i și β THEN

$s' := s - 1 + q$

IF $s' = 0$ THEN RETURN **true**, WRITE soluție: unificatorul $\nu \circ \mu$

ELSE **backwardChaining**(s' ; $\nu(\delta'_1), \dots, \nu(\delta'_{i-1}), \nu(\gamma_1), \dots, \nu(\gamma_q), \nu(\delta'_{i+1}), \dots, \nu(\delta'_s)$; $\nu \circ \mu$)

RETURN **false**

Observăm că algoritmul de mai sus constă într-un **backtracking recursiv**.

Dezavantaj: căutarea de mai sus este de tip **depth first**, aşadar, de exemplu, dacă inversăm ultimele două clauze din următoarea bază de cunoștințe:

$arc(N, M) :- M \text{ is } N + 1.$

$drum(X, Y) :- arc(X, Y).$

$drum(X, Y) :- arc(X, Z), drum(Z, Y).$

De ce aplicarea unui **fapt** sau a unei **reguli** ca mai sus semnifică aplicarea **regulii rezoluției**

atunci, de exemplu, la interogarea: `?- drum(1, 3)`, recursia nu se termină.
Cu notațiile de mai sus, amintesc că:

enunțul $\forall x_1 \dots \forall x_n \alpha$ are forma clauzală $\{\alpha\}$,

$$\begin{aligned} \forall y_1 \dots \forall y_k [(\gamma_1 \wedge \dots \wedge \gamma_q) \rightarrow \beta] &\models \forall y_1 \dots \forall y_k [\neg(\gamma_1 \wedge \dots \wedge \gamma_q) \vee \beta] \\ &\models \forall y_1 \dots \forall y_k (\neg \gamma_1 \vee \dots \vee \neg \gamma_q \vee \beta), \end{aligned}$$

având forma clauzală $\{\neg \gamma_1, \dots, \neg \gamma_q, \beta\}$,

$$\begin{aligned} \neg [\exists z_1 \exists z_2 \dots \exists z_p (\delta_1 \wedge \dots \wedge \delta_s)] &\models \forall z_1 \forall z_2 \dots \forall z_p \neg(\delta_1 \wedge \dots \wedge \delta_s) \\ &\models \forall z_1 \forall z_2 \dots \forall z_p (\neg \delta_1 \vee \dots \vee \neg \delta_s), \end{aligned}$$

având forma clauzală $\{\neg \delta_1, \dots, \neg \delta_s\}$.

Pasul de aplicare a unui **fapt** sau a unei **reguli** descris mai sus din backtracking-ul urmat de Prolog pentru a răspunde interogării `?- $\delta_1, \dots, \delta_s$` constă în **regula rezoluției** aplicată clauzei corespunzătoare negației acestei interogări și clauzei corespunzătoare faptului, respectiv regulii de mai sus, în modul următor:



pentru a satisface **subscopul** δ_i pentru un $i \in \overline{1, s}$:

- cu **faptul** α :

$$\frac{\{\neg \delta_1, \dots, \neg \delta_{i-1}, \neg \delta_i, \neg \delta_{i+1}, \dots, \neg \delta_s\}, \{\alpha\} \text{ (cu unificatorul } \mu\text{)}}{\{\neg \mu(\delta_1), \dots, \neg \mu(\delta_{i-1}), \neg \mu(\delta_{i+1}), \dots, \neg \mu(\delta_s)\}}$$

- prin **regula** $\beta :- \gamma_1, \dots, \gamma_q$:

$$\frac{\{\neg \delta_1, \dots, \neg \delta_{i-1}, \neg \delta_i, \neg \delta_{i+1}, \dots, \neg \delta_s\}, \{\neg \gamma_1, \dots, \neg \gamma_q, \beta\} \text{ (cu unificatorul } \mu\text{)}}{\{\neg \mu(\delta_1), \dots, \neg \mu(\delta_{i-1}), \neg \mu(\gamma_1), \dots, \neg \mu(\gamma_q), \neg \mu(\delta_{i+1}), \dots, \neg \mu(\delta_s)\}}$$

Într-adevăr, următorul scop compus pe care trebuie să îl satisfacă Prologul după aplicarea **faptului**, respectiv **regulii** de mai sus este:

$$\mu(\delta_1), \dots, \mu(\delta_{i-1}), \mu(\delta_{i+1}), \dots, \mu(\delta_s),$$

respectiv

$$\mu(\delta_1), \dots, \mu(\delta_{i-1}), \mu(\gamma_1), \dots, \mu(\gamma_q), \mu(\delta_{i+1}), \dots, \mu(\delta_s).$$

Toate derivările posibile prin rezoluție respectând strategia de mai sus pot fi reprezentate printr-un **arbore de derivare prin rezoluție SLD**:



- **nodurile** acestui arbore sunt etichetate cu **negațiile scopurilor compuse curente**;
- **rădăcina** arborelui de derivare este etichetată cu **negația scopului compus** din care constă interogarea;
- pentru fiecare nod al arborelui și fiecare clauză din baza de cunoștințe care, împreună cu eticheta acelui nod, are o derivare prin rezoluție, se introduce în arbore un fiu al acelui nod, etichetat cu **negația noului scop compus** obținută prin acea derivare prin rezoluție;
- muchiile arborelui se etichetează cu clauzele din baza de cunoștințe pentru care s-a aplicat regula rezoluției și unificatorii folosiți la fiecare astfel de derivare prin rezoluție.

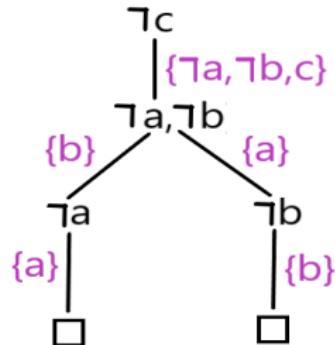
Conform celor de mai sus:

- dacă măcar una dintre frunze este etichetată cu \square , atunci răspunsul la acea interogare este **true**, iar soluția interogării este compunerea unificatorilor de pe drumul de la acea frunză la rădăcină;
- dacă niciuna dintre frunze nu este etichetată cu \square , atunci răspunsul la acea interogare este **false**.

Exemplu

Pentru exemplul de mai sus în care am aplicat rezoluția propozițională, avem arborele de derivare prin rezoluție SLD:

a. {a}
 b. {b}
 c :- a,b. {¬a, ¬b, c}
 ?- c.



Pentru simplitate, putem omite accoladele din etichetele nodurilor arborelui de derivare, ca mai sus.

(Rezoluția SLD pentru o clauză scop și o mulțime de clauze Horn este corectă și completă)

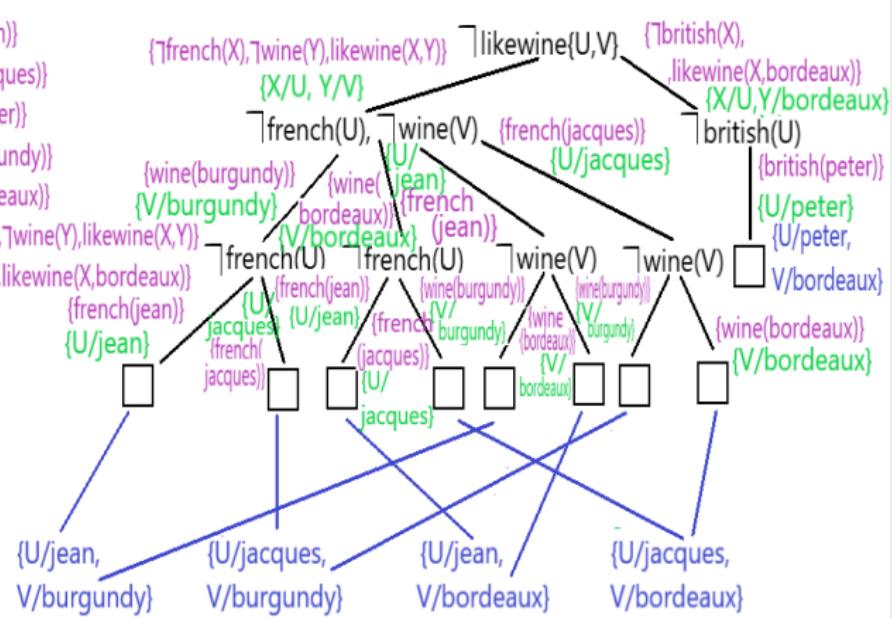
Rezoluția SLD aplicată ca mai sus, pornind de la o **clauză scop** și aplicând, la fiecare pas, **regula rezoluției** pentru **clauza scop curentă** și o **clauză Horn**, întotdeauna se termină într-un număr finit de pași și dă răspunsul corect: există o astfel de derivare prin rezoluție a clauzei vide \square dacă negația acelei clauze scop se deduce semantic din acea mulțime de clauze Horn.

În plus, arborele de derivare prin rezoluție dă **toate soluțiile** unei interogări, sub forma compunerilor unificatorilor pe fiecare drum de la o frunză etichetată cu \square rădăcina arborelui.

Exemplu (nu neapărat soluții distincte pentru diferitele frunze etichetate cu \square)

french(jean).
 french(jacques).
 british(peter).
 wine(burgundy).
 wine(bordeaux).
 likewine(X,Y) :- french(X), wine(Y).
 likewine(X,bordeaux) :- british(X).
 ?- likewine(U,V).

{french(jean)}
 {french(jacques)}
 {british(peter)}
 {wine(burgundy)}
 {wine(bordeaux)}



Soluțiile interogării:

(Desigur, muchiile albastre nu fac parte din arborele de derivare prin rezolutie.)