Capitolul 5

Programare logică

Să punctăm încă o dată faptul că realitatea (sumumul cunoştinţelor noastre despre o parte a lumii reale, la un moment dat) poate fi modelată prin afirmaţii, care, la rândul lor, pot fi reprezentate în logica formală clasică, sintactic, prin formule (metaformule). Afirmaţiile au asociată o semantică, adică o valoare de adevăr. Clasei de formule alese, **FORM**, i se ataşează astfel şi o clasă de structuri, *Str*, prin care valoarea de adevăr (unică în contextul precizat) a oricărei formule poate fi efectiv calculată (pentru fiecare F ∈ **FORM** şi fiecare *S* ∈ *Str*, obţinem *S*(F) ∈ **B**). *Problemele principale privind modelarea în modul descris a părţii de realtate alese sunt legate de posibilitatea de a decide pe de o parte* ***dacă formulele corespunzătoare nu sunt cumva contradicţii (sau contradictorii ca mulţime)*** *şi pe de altă parte* ***dacă alte formule*** *(reprezentând noi cunoştinţe) reflectă sau nu realitatea existentă (altfel spus,* ***sunt sau nu consecinţe semantice din formulele iniţiale****).* Totul se reduce în final la *decidabilitatea şi tratabilitatea unor probleme de tip* **SAT, SAT1, *etc.*** Deşi rezultatele teoretice, fie privind direct structura **FORM** şi a unor sisteme deductive pentru **FORM** (de exemplu, nedecidabilitatea **SAT1** sau netratabilitatea **SAT**), fie privind legătura dintre asemenea sisteme deductive şi *Val*(**FORM**) (lipsa unor teoreme de completitudine în special) sunt mai degrabă negative, există şi câteva *concluzii optimiste: (semi)algoritmii*

sintactici pentru rezolvarea **SAT**, **SAT1** *sunt mai uşor de înțeles (pentru „calculator”, în mod sigur) şi de manipulat decât cei bazați pe semantică; ei sunt tratabili măcar pentru anumite subclase interesante de formule; chiar în lipsa unor asemenea (semi)algoritmi, se pot imagina anumite proceduri implementabile, de tip interactiv (dialog în timp real cu utilizatorul), care pot furniza, dacă nu răspunsuri complete, măcar răspunsuri parțiale, sau indicații utile despre cum (şi în ce situații) s-ar putea obține un răspuns convenabil.* **Prin urmare, tot ceea ce rămâne de făcut este să se găsească asemenea *algoritmi, semialgoritmi, proceduri automate,* etc., pentru *clase convenabile de (meta)formule,* având ataşată o *noțiune corespunzătoare de adevăr***. Din punctul de vedere al unui utilizator, alternativa propusă de **Programarea logică** este atrăgătoare. **Astfel, în loc să se utilizeze (pentru reprezentarea informației şi prelucrarea acesteia) un limbaj de programare clasic (imperativ, orientat obiect, etc.) poate fi preferat un *limbaj creat special* pentru reprezentarea de (meta)formule şi în care un (semi)algoritm (sau chiar procedură interactivă) pentru rezolvarea SAT(1) şi bazat, de exemplu, pe rezoluție, este implementat direct în compilatorul (interpreterul) asociat. Asemenea limbaje sunt cunoscute şi sub numele de limbaje de tip PROLOG**. Limbajul **PROLOG** într-o primă formă implementabilă a fost conceput de către un grup de cercetători în **Inteligența artificială**, la începutul deceniului al optulea al secolului trecut, la Universitatea din Marsilia, Franța ([ROU]), căpătând însă ulterior extensii, transformări şi utilizări nebănuite la stadiul inițial. **Este un *limbaj declarativ,* dedicat reprezentării şi prelucrării**

relațiilor (predicatelor, afirmațiilor). Esența sa este exprimată prin ***paradigma de programare***, datorată lui R. Kowalski ([KOW]): **Algoritm = Logică + Control**. În sensul celor spuse anterior, **prin *Logică* se înțelege totalitatea cunoştințelor de care dispunem în privința unei „lumi" (parte a realității), cunoştințe exprimate prin formule (aparținând, în general, unui fragment al LP1$_{=}$), iar prin *Control*, strategia (algoritmul) prin care se manipulează o clasă de asemenea formule, în vederea obținerii unui anumit răspuns (aceasta implementând, în general, un anumit tip de rezoluție)**. În cele ce urmează, vom face doar o *scurtă introducere* în această tematică, bazându-ne în principal pe [MAS1], [MAS2] (şi, desigur, bibliografia indicată în acea lucrare).

§1. Exemple de programe logice *pure*

Ținând cont că scopul principal al acestui capitol este doar unul introductiv pentru un domeniu vast, ne vom baza în principal pe exemple (cu caracter didactic) şi nu pe enumerarea unor concepte sau rezultate formale. Deoarece este posibil ca exemplele să fie reluate şi dezvoltate, vom proceda din nou la numerotarea lor în secvență.

Exemplul 5.1 (lumea lui Adam şi Eva). Se cunosc următoarele *fapte* şi *afirmații mai complexe* din/despre această lume:

Evei îi plac merele.
Evei îi plac vinurile.

Lui Adam îi place orice persoană căreia îi plac vinurile.

În condițiile de mai sus, am dori să ştim dacă:

Există o persoană pe care să o placă Adam?

Desigur că în cazul unui răspuns pozitiv, *am dori să ştim şi care anume ar fi persoana (persoanele) respectivă (respective)*.

Primul pas în scrierea unui program de tip PROLOG, pur, este să formalizăm afirmațiile anterioare (inclusiv *interogarea*) prin formule din **LP1**. Pentru aceasta, să identificăm mai întâi elementele importante din lumea considerată. Vom distinge astfel:

Obiecte. Eva, mere, vinuri şi Adam sunt singurele *obiecte* care pot fi identificate ca atare în această lume simplă (nu este neapărată nevoie să facem distincție între, de exemplu, *lucruri*, *ființe/viețuitoare*, *oameni/persoane*, etc.; adică, într-un limbaj de specialitate, nu este nevoie de *tipizare*). Ele vor interpretate drept (simboluri de) ***constante funcționale***, adică elemente ale lui F_0, pe care le vom nota prin Eva, Mere, Vinuri, Adam (faptul că începem cu litere mari în scrierea constantelor nu este întâmplător).

Nume generice pentru obiecte. Avem nevoie de acest lucru deoarece există exprimarea Lui Adam îi place *orice* persoană ... . Vom nota cu X mulțimea tuturor acestor nume (care vor fi desigur nume de ***variabile***) şi vom pune, ca şi până acum de altfel, $x, y, \ldots \in X$.

Relații (legături) între (mulțimi de) obiecte. Singura relație identificabilă este a place. Aceasta va fi reprezentată formal printr-un simbol predicativ, notat place $\in P_2$ (nici aici modalitatea de scriere nu este întâmplătoare).

Transformări între (mulțimi de) obiecte. Acestea s-ar reprezenta prin ***simboluri funcționale de aritate mai mare ca 0***. În cazul nostru, nu există nici o asemenea transformare care să poată fi identificată.

Afirmații. În acest moment, putem *traduce* cunoştințele existente în **formule**. Avem:

G_1: place(Eva, Mere) traduce *faptul* Evei îi plac merele.
G_2: place(Eva, Vinuri) traduce *faptul* Evei îi plac vinurile.

A treia frază inițială exprimă ceva puțin mai complex despre lumea în cauză şi este natural să ne gândim la o formulă compusă. Putem reformula fraza mai întâi prin Dacă există cineva căruia îi plac vinurile, atunci de aceea (acela) îi place lui Adam (oricine ar fi acel cineva) şi apoi prin Dacă lui x îi plac vinurile, atunci lui Adam îi place de x, pentru fiecare $x \in X$, adică obținem:

G_3: $(\forall x)$(place(x, Vinuri) $\rightarrow$ place(Adam, x)).

Interogarea (întrebarea). Ea se traduce imediat prin Există y astfel încât lui Adam îi place de y? (alegerea unui nume diferit de x pentru

variabila corespunzătoare nu este întâmplătoare), adică dispunem şi de formula din **LP1**:

G: (∃y)place(Adam, y).

Pentru a efectua şi ***al doilea pas*** (suntem deja într-un cadru formal cunoscut), să observăm că a răspunde la întrebare înseamnă a vedea dacă G este (sau nu) consecință semantică din $\{G_1, G_2, G_3\}$, ceea ce, conform **Teoremei 2.3, punctul (iii)**, este echivalent cu a arăta că $F = G_1 \wedge G_2 \wedge G_3 \wedge \neg G$ este contradicție. Desigur că prin transformări succesive, aducem uşor (nici măcar nu este nevoie de skolemizare) pe F la **FNSC** şi apoi obținem reprezentarea lui F* ca mulțime de mulțimi de literali:

$F \equiv place(Eva, Mere) \wedge place(Eva, Vinuri) \wedge (\forall x)(\neg place(x, Vinuri) \vee place(Adam, x)) \wedge (\forall y)\neg place(Adam, y) \equiv (\forall y)(\forall x)(\neg place(Adam, y) \wedge place(Eva, Mere) \wedge place(Eva, Vinuri) \wedge (\neg place(x, Vinuri) \vee place(Adam, x)))$.

$F^* = \neg place(Adam, y) \wedge place(Eva, Mere) \wedge place(Eva, Vinuri) \wedge (\neg place(x, Vinuri) \vee place(Adam, x))$,

adică, în scrierea cu mulțimi:

$F^* = \{\{\neg place(Adam, y)\}, \{place(Eva, Mere)\}, \{place(Eva, Vinuri)\}, \{\neg place(x, Vinuri), place(Adam, x)\}\}$.

Pentru a simplifica unele raționamente, vom nota:

$C_1 = \{\neg place(Adam, y)\}$,

C_2 = {place(Eva, Mere)},

C_3 = {place(Eva, Vinuri)},

C_4 = {¬place(x, Vinuri), place(Adam, x)}.

Al treilea pas constă în a găsi o respingere în **LP1**, pornind cu clauzele lui F*, folosind *rezoluția de bază*, adică singura metodă cunoscută de noi până în prezent. Prin urmare, calculăm mai întâi D(F) şi apoi E(F) (sau/şi E'(F)). Neexistând constante de aritate mai mare ca 1 în F*, găsim imediat:

D(F) = {Eva, Mere, Vinuri, Adam}.

E'(F) = E(C_1) ∪ E(C_2) ∪ E(C_3) ∪ E(C_4) (nu o explicităm mai mult deoarece este foarte simplă).

Găsim o demonstrație (scurtă) prin rezoluție în **LP** a clauzei vide, pornind cu elementele lui E'(F), dacă notăm mai întâi cu C_1' clauza obținută din C_1 prin aplicarea substituției de bază [y/Eva] şi cu C_4' clauza obținută din C_4 aplicând [x/Eva] (atât C_1' cât şi C_4' aparțin desigur lui E'(F)). Astfel, avem Res(C_1', C_4') = {¬place(Eva, Vinuri)} (pe care o notăm cu C'). În sfârşit, Res(C', C_3) = { }. ■

Prin urmare, am satisfăcut parțial cerințele enunțate deoarece am găsit un răspuns corect la interogarea noastră. Din păcate, *nu am aflat şi care este (sunt) acel (acele) obiect(e) pe care îl (le) place Adam*. Este adevărat că acest (unic, în cazul de față) obiect (Eva) ar putea fi cumva dedus din substituțiile făcute pentru „a avea succes"

(obținere). Practic însă, ***am avea astfel nevoie de un alt tip de rezoluție, care, aplicată unei mulțimi date de clauze din* LP1 *să producă în mod explicit (măcar ca un* efect secundar*) asemenea substituții (pe care le vom numi* substituții de succes*)***. Deocamdată, putem totuşi trage o concluzie privind aspectul general al unui ***program, în accepțiunea programării logice (program PROLOG pur)***. El conține:

- ***Fapte*** (afirmații simple, modelate prin *formule atomice de bază din* **LP1**), care sunt „formule *elementare* de program".
- **Alte formule *de program*** (*formule compuse din* **LP1**, *mai exact formule Horn închise,* probabil *pozitive*).
- **O formulă *de interogare*** (*formulă compusă din* **LP1**, având şi ea o formă mai specială, negația ei fiind probabil o *formulă Horn închisă, negativă*).

Generalizând, concluzionăm că formulele program (să spunem G_1, G_2, ... , G_n) sunt formule din **LP1** aflate în **FNSC**, clauzele fiind clauze Horn având exact un literal pozitiv (sunt ***clauze Horn pozitive***). Formula de interogare G (numită şi ***scop***), apare tot ca o formulă închisă, însă cuantificată existențial. Deşi în exemplul considerat există doar un literal (pozitiv), se admite prezența mai multor asemenea literali, formula scop fiind de fapt o ***conjuncție de literali pozitivi, închisă, aflată în FNPR, cuantificată* doar *existențial***. După cum am mai sugerat, ***programul interogat va fi reprezentat de formula $F = G_1 \wedge G_2 \wedge \ldots \wedge G_n \wedge \neg G$ (se poate considera şi reprezentarea sa ca mulțime de mulțimi de literali, F fiind în FNSC şi clauzele fiind***

clauze Horn). Înainte de a furniza detalii suplimentare, considerăm utilă prezentarea unui alt exemplu, pentru a vedea că limbajul sugerat este la fel de puternic ca orice alt limbaj de programare de nivel înalt, putându-se efectua în acesta, de exemplu, calcule aritmetice uzuale (deşi, desigur, nu aceasta este utilitatea principală a **PROLOG**-ului).

Exemplul 5.2 (adunarea în N). Să descriem *lumea adunării pe mulțimea numerelor naturale* şi apoi să *calculăm 3 + 2* folosind un *program logic interogat*. Cazul considerat reprezintă deja o parte a unei realități având o descriere formală, matematică. Astfel, putem porni de la *definiția lui Peano* pentru **N**, adică de la definiția constructivă deja folosită:

Baza. $0 \in \mathbf{N}$.
Pas constructiv. Dacă $n \in \mathbf{N}$ atunci $s(n) \in \mathbf{N}$.

Reamintim că ideea în definiția de mai sus este aceea că *obiectul* notat 0 este număr natural şi dacă un obiect numit n este considerat număr natural atunci şi *succesorul său*, notat s(n) este tot număr natural. Acum putem da o definiție constructivă a adunării, ca operație binară pe **N**, bazându-ne pe reprezentarea a doar două *proprietăți* ale acesteia (suficiente pentru a calcula 3 + 2 în cadrul lumii descrise).

Baza: x + 0 = 0, pentru fiecare număr natural notat x.
Pas constructiv: s(x + y) = x + s(y), oricare ar fi numerele naturale notate x, y.

Ultima definiție „spune” că adunarea lui 0 la orice număr natural nu are nici un efect (numărul respectiv fiind lăsat neschimbat) şi că dacă adunăm la numărul x pe succesorul numărului y, se obține acelaşi lucru ca şi în cazul în care l-am aduna pe y la x şi apoi am lua succesorul numărului rezultat. Nu vom transforma încă ceea ce cunoaştem în formule, pentru că este evident că ne-am plasa în **LP1$_{=}$,** pentru care **SAT1** este nedecidabilă. Există posibilitatea ca printr-un anumit truc de natură tehnică să „ascundem” simbolul de egalitate (simbol predicativ de aritate 2) într-un simbol predicativ de aritate mai mare. În cazul nostru, ne este de ajuns un simbol predicativ de aritate 3, notat A. Intuitiv, am avea A(a, b, c) = 1 dacă şi numai dacă a + b = c. Cu ajutorul unui asemenea simbol predicativ, cele două egalități din definiția adunării se traduc prin A(x, 0, x) respectiv prin dacă A(x, y, z) atunci A(x, s(y), s(z)). Traducerea celei de-a doua egalități nu respectă în totalitate semnificația intuitivă inițială (ea „spune” acum că dacă z este suma dintre x şi y, atunci succesorul lui z reprezintă suma dintre x şi succesorul lui y). Din nou, această exprimare este suficientă pentru a ne atinge scopul, care poate fi reformulat în cadrul logic propus prin întrebarea: **Există vreun număr natural care să reprezinte suma dintre numerele 2 şi 3 (şi, eventual, care este/sunt acesta/acestea)?** Să parcurgem cei trei paşi descrişi în exemplul anterior, necesari pentru a forma un program logic interogat.

Primul pas.

- **Obiecte**: 0.

- **Nume generice** pentru obiecte (variabile): x, y, z, u, ... $\in X$.
- **Relații** între obiecte: A $\in P_3$.
- **Transformări** între obiecte: s $\in F_1$.
- **Afirmații (formule program)**:
  $G_1 = (\forall x)A(x, 0, x)$. (**fapt**)
  $G_2 = (\forall x)(\forall y)(\forall z)(A(x, y, z) \rightarrow A(x, s(y), s(z)))$.
- **Interogarea (scopul)**. Ținând cont de definiția adoptată pentru **N**, numărul 1 va fi desemnat de termul de bază s(0), 2, de s(s(0)), etc. Datorită cerinței tehnice ca în formula scop să apară nume de variabile distincte de cele deja folosite pentru formulele program, vom pune:
  $G = (\exists u)A(s(s(s(0))), s(s(0)), u)$.

Al doilea pas. **Programul logic interogat** va fi dat de formula:
$F = G_1 \wedge G_2 \wedge \neg G =$
$\{\{A(x,0,x)\},\{\neg A(x, y, z), A(x, s(y), s(z)\},\{\neg A(s(s(s(0))), s(s(0)), u)\}\}$.

Al treilea pas. Urmărim obținerea unei respingeri pornind cu F şi, eventual, „deducerea” unei valori, care ar fi desigur s(s(s(s(s(0))))), adică 5, pentru suma dintre 3 şi 2, valoare care apare „pe undeva” în cursul procesului de aplicare a rezoluției de bază (prin intermediul substituțiilor). Lăsăm aplicarea efectivă a rezoluției de bază pe seama cititorului (a se vedea şi exercițiile din finalul capitolului), nu înainte de a observa că D(F) este, la fel ca în exemplul anterior, suficient de simplu, şi anume: $D(F) = \{0, s(0), s(s(0)), \ldots\} = \{s^{(n)}(0) \mid n \in \mathbf{N}\}$. ■

Putem trage concluzia generală că un program logic, spre deosebire de alte limbaje comerciale de nivel înalt, efectuează ***calcule simbolice****, numerele (de exemplu) fiind şiruri de caractere şi operaţiile cunoscute cu numere având corespondent în anumite transformări asupra şirurilor de caractere*. Din acest motiv timpul real necesar pentru efectuarea unor asemenea calcule este foarte mare şi nu este de aceea indicat să apelăm la programarea logică pentru a efectua calcule numerice. Este însă adevărat că implementările comerciale ale unui limbaj logic (de tip **PROLOG**, [MAS1]), oferă facilităţi „nestandard" pentru efectuarea rapidă a unor asemenea calcule (există şi *maşini* **PROLOG** *dedicate*).

Vom descrie în continuare, tot pe scurt, un alt tip de rezoluţie în **LP1**, „echivalent" cu rezoluţia de bază şi cunoscut sub numele de ***rezoluţie pură (specifică)***. Acesta va furniza, în cazul obţinerii unei respingeri, ***ca efect secundar***, şi o ***substituţie de succes*** (din care se pot/poate „extrage" simplu numele obiectelor/obiectului care satisfac(e) interogarea).

§2. Sintaxa programelor logice

Un program logic (clasic, standard) este format dintr-o mulţime finită de *formule program*, alcătuită din *fapte* şi o mulţime de *formule suplimentare*. **Un program logic interogat** este un cuplu format dintr-un program logic şi o *formulă scop*. Toate *formulele implicate sunt formule Horn, aflate în* **FNSC** *şi cuantificate universal* (clauza scop, doar în urma negării ei).

Definiția 5.1 (program logic interogat; program PROLOG pur).

- **Program logic**. Acesta conține:
  - ***Fapte***, având forma:

    P.

    unde **P** este un literal pozitiv din **LP1**. Un asemenea literal pozitiv poate avea şi variabile, presupuse a fi implicit cuantificate universal, deşi în general el reprezintă o formulă de bază. Formula reprezentată este deci $(\forall_*)(\mathbf{1} \to \mathbf{P})$ sau $(\forall_*)\mathbf{P}$, care poate fi citită *„Sigur **P**”*. Notăm cu $G_1 = \{G_1, G_2, \ldots, G_p\}$ mulțimea (finită a) faptelor programului notat F.
  - ***Clauze definite (suplimentare)***. Aspectul lor este:

    $\mathbf{P} \leftarrow \mathbf{Q_1}, \mathbf{Q_2}, \ldots, \mathbf{Q_n}$.

    unde **P** şi $\mathbf{Q_i}$, $i \in [n]$ sunt literali pozitivi din **LP1** (simbolul ÷ este uneori înlocuit prin :-, ca şi în clauza scop). Formula reprezentată este
    $(\forall_*)(\mathbf{Q_1} \wedge \mathbf{Q_2} \wedge \ldots \wedge \mathbf{Q_n} \to \mathbf{P})$ sau
    $(\forall_*)(\mathbf{P} \vee \neg \mathbf{Q_1} \vee \neg \mathbf{Q_2} \vee \ldots \vee \neg \mathbf{Q_n})$. Se citeşte *„**P**, în caz că **Q1** şi **Q2** şi ... şi **Qn**”*. Notăm cu $G_2 = \{G_{p+1}, G_{p+2}, \ldots, G_{p+q}\}$ mulțimea (finită a)

clauzelor definite ale programului F şi cu $G = G_1 \cup G_2$. (uneori, chiar acest G este considerat a fi mulțimea clauzelor suplimentare sau *de program*). Mai sus, p, q ∈ **N**, dar nu pot fi simultan egali cu 0.

- **Interogarea**. ***Clauza scop*** este scrisă:

  - G = **?** ¸ **R_1, R_2, … , R_k**.

    Din nou, **R_1, R_2, … , R_k** sunt literali pozitivi din **LP1**, de această dată variabilele care apar fiind presupuse a fi cuantificate existențial. Mai exact, clauza scop reprezintă transcrierea unei formule Horn de tipul $(\exists_*)(\mathbf{R_1} \grave{U} \mathbf{R_2} \grave{U} \ldots \grave{U} \mathbf{R_k})$, citit *„Există elemente în domeniul considerat astfel încât condițiile **R_1, R_2, … , R_k** să fie îndeplinite?”*, ceea ce prin negare furnizează formula
    ⌉G = $(\forall_*)$(ù **R_1** Ú ù **R_2** Ú **…** Ú ù **R_k**). Vom lua acum F = <G, G>. ■

Observație. După cum am putut deduce din exemple, **execuția unui program logic** înseamnă **testarea nesatisfiabilității** formulei $\bigwedge_{i=1}^{p+q} G_i \wedge \neg G$, pe care o vom nota tot cu F. F este în **FNSC** închisă (eventual, după ce se redenumesc anumite variabile, acest lucru provenind din necesități tehnice, nu din nevoia de a aduce formula la

FNR), clauzele fiind clauze Horn (eventual, reprezentate ca mulțimi). Implementarea foloseşte o strategie **SLD**. În fiecare pas se efectuează o **rezoluție pură**, una dintre clauzele implicate fiind întotdeauna **clauza scop curentă (inițial, ea este formula de interogare G)**, cealaltă clauză fiind una dintre fapte sau clauzele definite aparținând programului (pe scurt, o **clauză program**). Ținând cont de forma formulelor care intervin şi de definiția rezoluției pure, există o schemă simplă care completează strategia **SLD** (prin precizarea acelei *funcții de selecție*). Astfel, ***se alege* un literal (negativ) din clauza scop curentă** (de obicei, acesta este *primul întâlnit*, ca scriere) şi **capul** (membrul stâng al) unei formule program, care este un literal pozitiv. Dacă este posibil, ei se ***unifică***, obținându-se o **nouă clauză scop**. Procedeul continuă şi, deşi nu avem garanția terminării lui, este tot ceea ce putem spera în acest stadiu de cunoaştere. ■

Reluăm mai în detaliu câteva concepte amintite pe scurt în **Capitolul 3**.

Definiția 5.2 (unificare). Fie $L = \{L_1, L_2, \ldots, L_k\}$ o mulțime (finită), nevidă, de literali din **LP1**. Ea se numeşte **unificabilă** dacă există o substituție s astfel încât $card((L)s) = 1$. În acest caz, s se numeşte **unificator pentru** L. O substituție s se numeşte **cel mai general unificator** (**m.g.u.**, pe scurt) pentru o mulțime unificabilă L dacă orice alt unificator s' se „obține" din s, adică pentru fiecare unificator s' există o substituție sub astfel încât $s' = s \bullet sub$. ■

Să presupunem acum că avem două clauze (distincte) din **LP1** (care nu sunt neapărat clauze Horn, conținând şi variabile). *Ideea rezoluției pure se bazează pe faptul că putem unifica (identifica textual) „cât mai mulți" literali din cele două clauze cu ajutorul unei substituții convenabile şi apoi îi putem elimina pe aceştia (rezultând o nouă clauză, în final),* similar cu cazul rezoluției propoziționale.

Definiția 5.3 (rezoluția pură/specifică într-un pas, în LP1). Fie C_1, C_2 şi R clauze în **LP1**, $C_1 \neq C_2$. **R se numeşte rezolvent (pur) pentru C_1 şi C_2**, obținut **într-un pas**, dacă sunt îndeplinite condițiile:

(i) Există substituțiile „de redenumire" s_1 şi s_2 astfel încât $(C_1)s_1$ şi $(C_2)s_2$ nu au variabile comune.

(ii) Există literalii $L_1, L_2, \ldots, L_m \in (C_1)s_1$ şi $L'_1, L'_2, \ldots, L'_n \in (C_2)s_2$ astfel încât mulțimea

$$L = \{ \overline{L}_1, \overline{L}_2, \ldots, \overline{L}_m, L'_1, L'_2, \ldots, L'_n \}$$

este unificabilă. Fie sub un cel mai general unificator pentru L.

(iii) $R = (((C_1)s_1 \setminus \{L_1, L_2, \ldots, L_m\}) \cup ((C_2)s_2 \setminus \{ L'_1, L'_2, \ldots, L'_n \}))sub$.

■

Deoarece nu există pericol de confuzii, vom folosi aceleaşi notații pentru rezoluția pură identice cu cele adoptate pentru rezoluția propozițională (ceea ce se schimbă este practic doar definiția rezolvenților obținuți într-un pas). Teoremele următoare le prezentăm fără demonstrație (se poate consulta [MAS1]).

Teorema 5.1 (Julia Robinson). Orice mulțime finită, nevidă, unificabilă, de literali din **LP1**, admite un cel mai general unificator. Problema *testării faptului că o mulțime de literali este unificabilă* este decidabilă. De asemenea, găsirea unui cel mai general unificator pentru o mulțime unificabilă se poate face algoritmic. ■

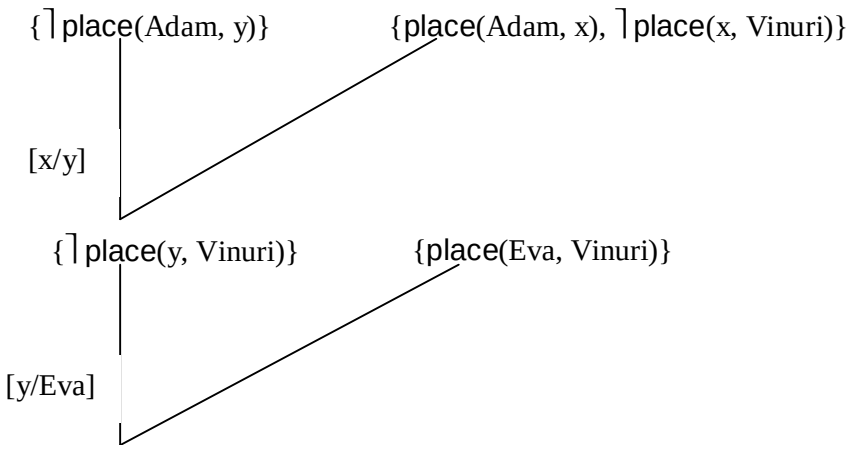
Există o metodă relativ simplă (algoritm) pentru unificarea unei mulțimi date de literali. Fără a intra în amănunte, tot ceea ce trebuie să înțelegem este că *trebuie să identificăm porțiuni de text, având (în cazul de față) un format special.* Acest lucru nu se admite a fi făcut decât prin intermediul variabilelor, folosind substituțiile. În plus, „apelurile recursive", de genul *„în substituția s, variabila x este înlocuită cu termul t, care conține x"*, sunt interzise.

Teorema 5.2 (a rezoluției pure pentru LP1). Fie $F \in$ **LP1** o formulă închisă, aflată în **FNSC**, $F = (\forall_*)F^*$ (F poate fi, în particular, un program **PROLOG** pur). Atunci, F este nesatisfiabilă dacă şi numai dacă $\in Res^*(F^*)$, adică dacă şi numai dacă există o demonstrație prin rezoluție pură a clauzei vide (o *respingere*), pornind cu clauzele lui F. ■

Teorema 5.3 (completitudinea SLD-rezoluției). Dacă F este o mulțime de clauze Horn din **LP1**, atunci, dacă F este nesatisfiabilă, există o respingere pornind cu F şi care utilizează **SLD**-rezoluția pură. ■

Desigur că **SLD-***rezoluția este şi corectă*. Din păcate însă, *problema este netratabilă relativ la complexitatea timp generală*. Mai mult, nevoia de a aborda simplu situațiile reale precum şi anumite cerințe legate de implementare conduce deseori la pierderea completitudinii rezoluției (necesitatea de a manipula şi formule care să nu fie clauze Horn, cum ar fi cele care rezultă prin apariția în clauza scop a unor literali negați sau a disjuncției în loc de conjuncție; folosirea unor structuri de date „apropiate" de programarea imperativă, ca de exemplu liste, stive, arbori; utilizarea unei strategii de construcție a arborelui de rezoluție de tip **DFS** în loc de **BFS**, conform [CRO], [MAS1], [MAS4], [MAS5]). Pentru a evita acest lucru, se pot utiliza metode interactive, în care programatorul este „invitat" să ia decizii în timp real, pentru a avea posibilitatea obținerii unui succes în execuția unui program **PROLOG** („dirijând" el însuşi execuția spre un posibil succes).

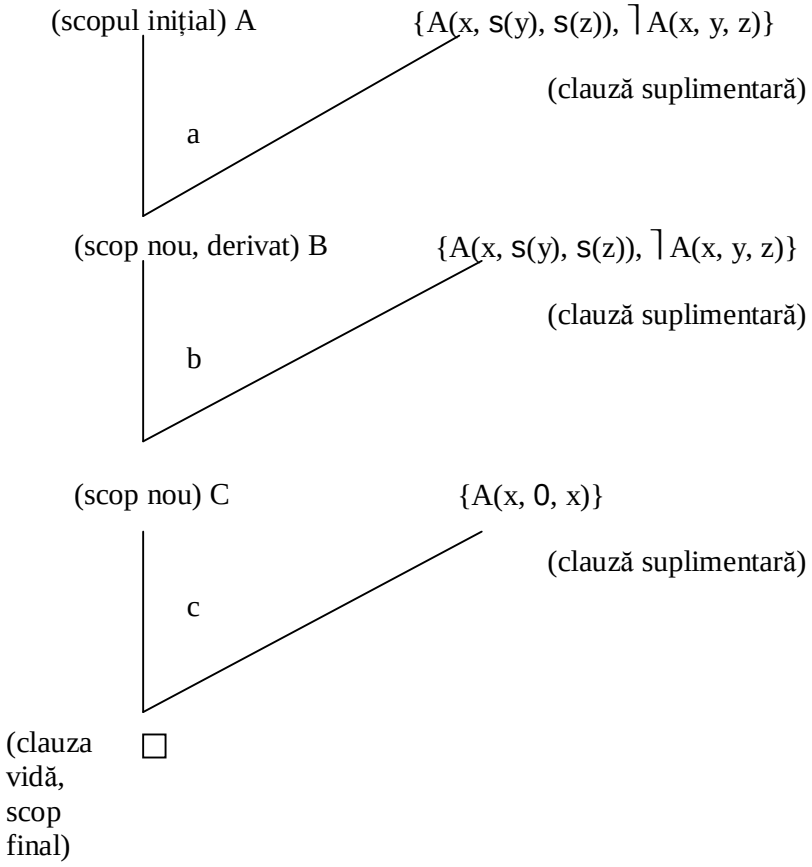
Exemplul 5.1 (reluat). Vom arăta cum putem găsi şi o *substituție finală de succes (pe post de cel mai general unificator pentru o anumită mulțime de literali), ca efect secundar al aplicării rezoluției pure*. În acest mod, obținem nu numai răspunsul de tip „DA/NU" la interogare, ci şi obiectele (obiectul) care o satisfac(e). Mai jos avem reprezentat arborele de rezoluție pură (având doar doi paşi necesari a fi aplicați) care descrie o respingere:



Reamintindu-ne că clauza scop era G = ? ÷ (∃y)place(Adam, y)., să constatăm că „execuția" programului interogat „oferă" un răspuns pozitiv, dedus în urma existenței respingerii anterioare (el fiind „DA", adică *este adevărat că G este consecință semantică din clauzele program considerate*, adică în lumea dată există într-adevăr „ceva" care îi place lui Adam). În plus, în graful de mai sus sunt prezente două substituții elementare. Prima este [x/y] şi ea reprezintă m.g.u.-ul care unifică mulțimea de literali {place(Adam, y), place(Adam, x)} (după algoritmul dedus din **Teorema lui J. Robinson**; de fapt, se putea la fel de bine obține [y/x]). A doua substituție ([y/Eva]) unifică, similar, mulțimea L = {place(y, Vinuri), place(Eva, Vinuri)}. Ca urmare, pentru a obține am folosit substituția „totală" s = [x/y]•[y/Eva] care ne „dezvăluie" unul dintre obiectele căutate şi anume Eva (dacă un asemenea obiect este sau nu

unic, este o altă problemă care poate fi rezolvată de către un interpretor **PROLOG**). ■

Exemplul 5.2 (reluat). În mod cu totul similar ca mai înainte, fără a construi întreg arborele de rezoluție pură posibil, obținem respingerea:



Respingerea anterioară ne „spune” că răspunsul la întrebarea „*există vreun număr natural care să fie suma dintre 2 şi 3 ?*” este pozitiv. Să precizăm şi că în graful de mai sus notaţiile reprezintă:

A = {⌉A(s(s (s (0)))), s (s (0)), u)}

a = sub_1=[x/ s (s (s (0)))]•[y/ s (0)]•[u/ s (z)]

B = {⌉A(s (s (s (0)))), s (0), z)}

b = sub_2 =[x/s(s(s(0)))]g[y/0]g[z/s(z′)]

(în cauza suplimentară facem mai întâi substituţia [z/z’], din motive tehnice, netransparente în acest moment fără anumite informaţii suplimentare)

C = {⌉A(s (s (s (0)))), 0, z’)},

c = sub_3 =[x/ s (s (s (0)))][z’/ s (s (s (0)))].

Substituţia finală este sub = sub_1•sub_2•sub_3. Din inspectarea atentă a acesteia (a se urmări *valoarea finală a variabilei u*, obţinută succesiv prin aplicarea substituţiilor elementare care compun sub), rezultă că răspunsul dorit este: *suma dintre 2 şi 3 este 5.* ■

§3. Rezumare şi Index

Programarea logică reprezintă o alternativă viabilă pentru programarea clasică, în momentul în care *realitatea este reprezentată şi studiată într-un mod declarativ*. Datorită unor rezultate teoretice negative, numărul de clase de formule care pot fi prelucrate convenabil este destul de restrâns. Mulţimea *clauzelor Horn din* **LP1** este însă una dintre ele. Chiar în cadrul restrâns considerat (pentru a nu mai aminti de programarea în logici de ordin superior sau neclasice), adoptarea

presupunerii lumii închise, modul de a **trata negația**, utilizarea **disjuncției în interogări**, precum şi **implementarea** unui **dialog interactiv** cu utilizatorul (prin care să se dirijeze „din exterior" execuția unui program), înseamnă extensii importante pentru programarea logică ([MAS1], [AND]) şi justifică orientarea unor grupuri semnificative de programatori într-o asemenea direcție.

Nici rezultatele teoretice, nici inovațiile de implementare, nici ariile de aplicabilitate ale ***Programării logice*** nu sunt de altfel epuizate, astfel încât acest domeniu, cu toată complexitatea sa, nu este încă unul fără viitor. Pentru informații suplimentare, de actualitate, se pot consulta pe INTERNET site-urile: http://www.amzi.com/, http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html, http://www.cs.cmu.edu/Web/Groups/AI/html/faqs/lang/prolog/prg/top.html, http://www.lpa.co.uk/, sau http://kti.ms.mff.cuni.cz/~bartak/prolog/.

Indexul care urmează este şi în acest capitol neexhaustiv (unii termeni se pot repeta, datorită revenirii la anumite aspecte tratate în capitolele anterioare):

§4. Exerciții

1. Găsiți respingerea bazată pe rezoluția de bază cerută în **Exemplul 5.2**.
2. Considerăm ([COT]) următorul program **PROLOG** notat F şi format din clauzele program (*în **toate** exercițiile care urmează adoptăm convenția că variabilele se notează prin litere latine mari, iar constantele, simbolurile funcționale şi predicative, folosind litere mici; de asemenea, simbolul ¸ este uneori înlocuit şi de* ¬):

   CP1: p(X, Z) ÷ q(X, Y), p(Y, Z).

   CP2: p(X, X) .

   CP3: q(c, b).

   Găsiți o **SLD**-respingere pentru scopul G = ? ÷ p(U, b). Se cere şi rezultatul execuției programului interogat P = <F, G>.

3. **Aritmetica** ([COT]). Conform **Exemplului 5.2**, mulțimea numerelor naturale poate fi definită prin următorul program **PROLOG**:

   numar_natural(0).

   numar_natural(s(X)) :- numar_natural(X).,

   unde predicatul numar_natural(X) „afirmă" că X este un număr natural. Adunarea poate fi dată şi de către predicatul plus1(X, Y, Z):

   plus1(0, X, X).

   plus1(s(X), Y, Z) :- plus1(X, s(Y), Z).

   Predicatul plus2 este similar cu cel dat deja de noi în **Exemplul 5.2** (şi notat acolo cu A):

   plus2(0, X, X).

   plus2(s(X), Y, s(Z)) :- plus2(X, Y, Z).

   Găsiți definiții **PROLOG** ale predicatelor minus(a, b, c), inmultire(a, b, c) şi exp(a, b, c), utilizând plus1, plus2. Intuitiv, predicatele cerute trebuie să fie adevărate dacă şi numai dacă sunt respectiv îndeplinite condițiile $a - b = c$, $a \times b = c$ şi $c = b^a$.

4. Fie **baza de cunoştințe** ([COT]) exprimată prin faptele:

   barbat(paul).

   barbat(andrei).

   femeie(maria).

   femeie(eliza).

   femeie(emilia).

   parinti(emilia, maria, paul).

parinti(andrei, maria, paul).

parinti(eliza, maria, paul).,

unde semnificaţia predicatelor implicate este evidentă. De exemplu, parinti(E, M, T) este adevărat dacă şi numai dacă *M şi T sunt părinţii lui E, M fiind mama, iar T fiind tată*. Putem atunci defini relaţia sora_lui, în conformitate cu definiţiile de mai sus, prin (sora_lui(X, Y) este adevărat dacă şi numai dacă X este sora lui Y):

sora_lui(X, Y):- femeie(X), parinti(X, M, T), parinti(Y, M, T).

Aceasta va fi (singura) clauză suplimentară a programului.

Se cere să se răspundă la interogarea:

? :- sora_lui(eliza, andrei).

5. **Arbori de căutare şi arbori de demonstrare** ([COT]). Fiind dat un program **PROLOG**, un scop iniţial şi *regula standard de selecţie a subscopurilor* (literalii din clauza scop curentă, dacă sunt mai mulţi, sunt selectaţi în ordinea scrierii lor textuale, în vederea unificării cu capul unei clauze program, care va fi aleasă ulterior), căutarea tututror alternativelor posibile se poate reprezenta printr-un arbore, numit *arbore de căutare* (*de evaluare* sau **arbore OR**). În acest arbore, rădăcina este scopul iniţial. Orice nod neterminal este etichetat cu o conjuncţie de subscopuri, derivată din *nodul tată* într-un singur pas de rezoluţie. Descendenţii imediaţi ai unui nod sunt scopuri *alternative* derivate din scopul prezent în acel nod. Căutarea se termină când toate nodurile sunt terminale. Orice nod terminal este etichetat cu „◊” în caz de terminare reuşită (*cu succes*) şi cu

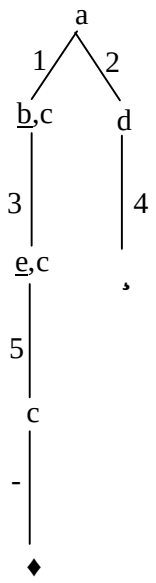
„♦” în cazul terminării cu eşec (*scopul nu poate fi satisfăcut*). Orice drum în arbore (o secvență de noduri de la rădăcină la un nod terminal) reprezintă un calcul posibil. Pot exista şi drumuri infinite. Să considerăm următoarele clauze „generice”:

```
a :- b, c.        /*clauza1*/
a :- d.           /*clauza2*/
b :- e.           /*clauza3*/
d.                /*clauza4*/
e.                /*clauza5*/
```

şi scopul inițial

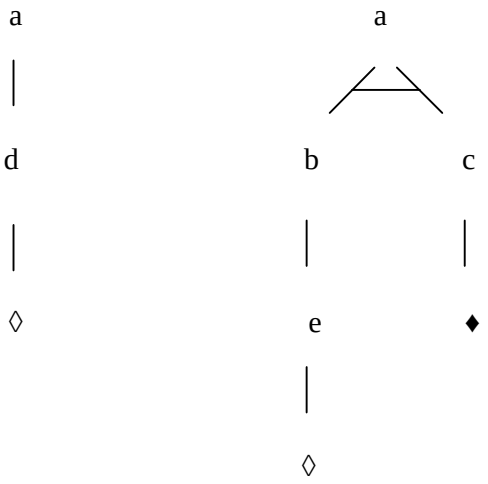
```
? :- a.
```

Arborele de căutare corespunzător programului interogat anterior este:

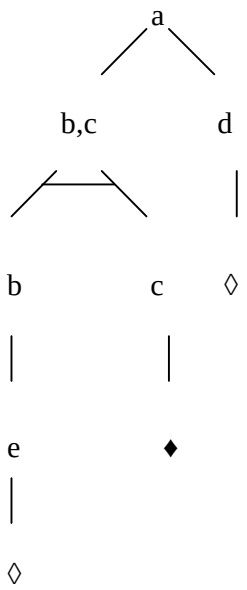


După cum am precizat, am urmat regula standard de selecție a subscopurilor (în graf este subliniat literalul selectat; arcele sunt notate cu numerele clauzelor corespunzătoare).

Fie acum, din nou, un program, un scop inițial, dar şi *regula standard de selectare a clauzelor program* (al căror cap s-ar putea unifica cu subscopul curent ales deja). Atunci subscopurile pot forma un aşa numit *arbore de demonstrare* (de derivare, **arbore AND**). În acest arbore, orice nod este un (sub)scop. Rădăcina are în calitate de descendenți imediați subscopurile scopului inițial. Fiecare dintre acestea au în calitate de descendenți imediați subscopurile clauzei selectate. Nodurile terminale sunt notate, ca şi mai înainte, prin „◊" şi „♦". Mulțimea de noduri care preced imediat nodurile terminale reprezintă subscopurile, conjuncția cărora formează scopul complex ce corespunde arborilor de demonstrare:



Ne-am plasat în contextul aceluiaşi program şi, conform celor spuse, pentru scopul inițial avem doi arbori de demonstrare (cei de mai sus) care corespund, respectiv, selecției clauzei1 şi respectiv selecției clauzei2. Arborii sunt reprezentați în figura anterioară. Acum putem „cumula" arborii anteriori în:



Mai exact, cele două tipuri de informații, furnizate de arborele/arborii de demonstrare (**AND**) şi arborele de căutare (**OR**) pot fi efectiv reprezentate printr-un singur arbore (cel de mai înainte), numit arbore **AND-OR** (sau ***arbore complet de calcul***). Arborele **AND-OR** descrie practic „spațiul total de calcul" care poate fi obținut din mulțimea de clauze ale programului interogat. Acesta este în fapt o *reprezentare unitară a tuturor tipurilor de nedeterminism* care apar, în mod implicit, în momentul execuției programelor logice ([MAS1]).

Să considerăm acum următorul program interogat:

bunic(X, Y):-tata(X, Z), tata(Z, Y).

bunic(X, Y):-tata(X, Z), mama(Z, Y).

mama(maria, paul).

mama(I, J):-mama(I, K), frate(K, J).

tata(ion, maria).

frate(paul, petru).,

scopul fiind ? :- bunic(ion, petru).

Găsiți arborele **AND-OR** corespunzător.

6. Arătați că următoarea mulțime de literali din **LP1** este unificabilă şi găsiți un (cel mai general) unificator:

   L = {P(x, y), P(f(b), g(x)), P(f(z), g(f(z)))}.

7. Exprimați următoarele afirmații prin formule din **LP1**:

   - Fiecare dragon este fericit dacă toți copiii săi pot zbura.
   - Dragonii verzi pot zbura.
   - Un dragon este verde dacă este copilul a cel puțin unui dragon verde.

   Arătați că afirmația ***Toți dragonii verzi sunt fericiți*** este consecință semantică din afirmațiile anterioare. Putem modela problema anterioară ca un program **PROLOG** interogat? Pentru „consistența" lumii modelate ar mai fi utile şi alte afirmații?